LPIC 1 400

MATERIAL DE ESTUDIO LPIC-1 400

Linux Professional Institute Certification

LPIC – 1 400

- Contenido adaptado a la nueva actualización **LPIC-1** modelo **400** de Enero del 2015
- Es un resumen personal del libro *"Guía de estudio – Exámenes **101** y **102** (Tercera edición – ANAYA / 671 páginas / Roderick W.Smith)"*
- Contiene extractos de documentación de terceros (*bloggers, 'wikis', 'howtos'…*), así como de las páginas man de distribuciones *RMP* y *Debian*
- La adaptación para la nueva certificación es la comparación entre el contenido del libro (*Approved training materials LPI*) y el borrador http://wiki.lpi.org/wiki/LPIC-1_Objectives_V4 , podéis comprobar el contenido añadido en la versión 4 en la siguiente página oficial http://wiki.lpi.org/wiki/LPIC1SummaryVersion3To4#2.0_Changes_in_LPIC-1
- Está dividido en dos partes; texto-resumen por un lado y comandos por otro (desde mi punto de vista facilita el estudio)
- Ver Notas sobre la estructura y disposición del material aquí presente, así como aclaraciones, material extra, contenido de cada examen, etc.

Nota: El contenido extra añadido a esta guía de estudio se referenciará mediante links a las páginas oportunas o se comentará de que libro se ha extraído

Tabla de contenido

Examen 101

LPIC-1 Capítulo 1

EN ESTE CAPÍTULO SE ABARCAN LOS SIGUIENTES OBJETIVOS DEL EXAMEN:

- 103.1: Trabajar desde la línea de comandos (4)
- 103.2: Procesar flujos de texto mediante filtros (3)
- 103.4: Uso de flujos, pipes, y redirecciones (4)
- 103.7: Buscar archivos de texto mediante expresiones regulares (2)

Herramientas básicas para la línea de comandos Linux

A lo largo de este capítulo veremos que es la consola de Linux, algunas de sus variantes y como utilizarla. Se nombrarán muchos de los comandos con los que un administrador de sistemás debe de estar familiarizado, ya que ocuparán gran parte de su jornada laboral en cuanto a la administración de un sistema *GNU/Linux.* Aprenderemos como movernos por el shell, por ejemplo para buscar ayuda sobre la utilización de un comando concreto, la instalación de un programa o el formato de su archivo de configuración. Para terminar, cerraremos el capítulo con algunos meta caracteres importantes del *shell*, los posibles atajos del teclado que nos facilitarán la interacción con la consola y como no, el uso de expresiones regulares con las que podremos manipular archivos y en concreto el flujo de texto.

La Shell

El *shell* es un programa más del operativo cuya función es la de leer e interpretar los comandos que en ella se teclean y ejecutan, que a su vez son “mini programás” que interactúan con el sistema operativo, es decir, el *Shell* es una especie de *frontend* que nos permite comunicarnos con el sistema operativo *GNU/Linux*. El *shell* ofrece un lenguaje de programación interpretado con el que poder escribir, modificar y verificar programás rápidamente y de una forma sencilla. La programación *Shell* es una parte fundamental de la administración de sistemás basados en *UNIX*, ya que mediante el uso de las herramientas (comandos) que nos ofrece el operativo, estructuras condicionales, bucles, funciones, variables, etc… podremos automatizar una gran parte de procesos rutinarios del sistema, como por ejemplo respaldar archivos, verificar procesos, creación de usuarios y grupos, notificarnos acciones concretas que se realicen en el sistema y cuál ha sido su resultado, etc… Existen diferentes consolas con las que podremos trabajar en un sistema *UNIX*. No obstante durante el curso que ofrece esta guía de estudios nos centraremos en el Shell Bash (*Bourne Again Shell*), ya que por defecto es la consola que se instala en la mayoría de distribuciones *GNU/Linux*.

Las consolas más comunes en Linux.

- **bsh** *(Bourne Shell)*: *Shell* original de *UNIX* creado a mediado de los 70's por *Stephen R. Bourne*. Su nombre de archivo es */bin/sh* y en los *OS Linux* suele ser un enlace simbólico a *bash*. Algunas de sus características podrían ser:
  - Control de procesos.
  - Control de flujo de entrada y salida (I/O)
  - Uso de expresiones regulares, variables y funciones.

Por contra, *bourne shell* carece de autocompletado de nombres de archivos, directorios y comandos (uso del tabulador), no contiene un historial de comandos y posee dificultades para la ejecución de varios procesos en background.

- **csh** (*C Shell*): La consola C original. No se usa (normalmente). Adiciona algunas características de funcionalidad con respecto a bsh como el historial de comandos y un mejor control de procesos.

- **tcsh**: Basada en la antigua consola csh, es bastante popular en *Linux* aunque no es la consola por defecto. Difiere en algunos aspectos a bash. Ofrece complexión de comandos y nombres.

- **ksh** (*Korn Shell*): Escasos usuarios, pero fieles. Diseñada para tener lo mejor de bsh y csh (Bourne y C)

- **bash** (*Bourne Again Shell):* Desarrollado por el proyecto *GNU* y basado en la consola *Bourne* de *UNIX*. Es la más común entre los *OS Linux* (reemplazado al antiguo *bsh /bin/sh*). Ofrece características de C Shell (*csh*), Korn Shell (*ksh*) y bsh. Podríamos nombrar algunas como:
  - Autocompletado de nombres de archivos, directorio, variables, usuarios, comandos y nombre de máquinas.
  - Corrección de rutas de archivos y directorios mediante el comando cd
  - Aritmética de enteros y soporte para Arrays de tamaño ilimitado.

- **zsh**: Consola Z. Va aún más allá de Korn, incorporando nuevas funcionalidades y mejoras.

Diferencia entre comandos *internos* y *externos* del bash

Lo normal de un sistema GNU/Linux es que este posea instalado un gran número de programás o comandos útiles para el buen funcionamiento y administración del sistema. Podremos encontrar tanto comandos internos como comandos externos:

- Los comandos **internos** son los que vienen integrado con la consola (*man bash*, para conocer los de bash) algunos de ellos pueden ser: *cd*, *pwd*, *echo*, *exec*, *time*, *set*, *exit*…
- Los comandos **externos** son aquellos que aún viniendo instalados (o no) en el sistema, necesitan que la ruta en la que se encuentran instalados (por ejemplo */usr/bin*, */usr/local/sbin*, */usr/games*, etc…) sea referenciada en el valor de la variable *PATH* del usuario. Solo de esta forma bash localizará al comando y posteriormente lo ejecutará.

Los comandos *internos* y *externos* tienen comportamientos diferentes y pueden aceptar parámetros diferentes. Por ejemplo con respecto a los enlaces simbólicos (archivos que apuntan a otros archivos o directorios, parecidos a los accesos directos de Windows). Si utilizamos el comando interno *pwd* (muestra el directorio de trabajo actual) estando situados en un enlace simbólico, este mostrará la ruta "*destino*" del enlace, es decir en vez del directorio en el que estamos situado, imprimirá la ruta de aquel directorio a la que en realidad apunta el enlace simbólico. Por su parte el comando externo *pwd*, mostrará el origen, es decir la propia ruta del enlace simbólico. **Advertencia**: La cuenta **root** JAMÁS debe de tener en su variable **PATH** la ruta de directorio actual (**./**), y con respecto a los usuarios, en caso de que sea necesario, deberá de ir al final del *PATH*.

Uso de variables

Las variables son "nombres" que utiliza el *Shell* (en este caso) para almacenar un valor, de manera que cuando llamemos a esa variable en realidad lo que estaremos haciendo será pedir un cierto valor que necesitaremos (en este caso, necesitará el *Shell*) para realizar una determinada acción. Por ejemplo comentábamos antes la importancia de añadir a la variable *PATH* el valor de una ruta en la que cabe la posibilidad de que se encuentre un programa que necesitaremos ejecutar sea cual sea nuestro directorio de trabajo, pero… ¿por qué hay que añadirlo a la variable *PATH*? pues porque cuando invocamos a un comando, la *Shell* por defecto busca si ese comando pertenece a si misma, es decir se trata de un comando interno, de lo contrario necesitará ciertas rutas en las que poder localizar este comando. Estudiaremos más sobre las variables en el capítulo destinado a la administración y configuración de la Shell pero es importante saber que para que la Shell haga uso de una variable, antes deberemos de exportarla.

- Exportar una variable desde la línea de comandos:

```
$ export NOM_VAR=valor_variable
```

- Comprobar el valor de una variable desde la línea de comandos:

```
$ echo "$NOM_VAR"
```

Algunos metacaracteres útiles de shell

Cuando comencemos a interactuar con la *Shell* o crear pequeños programás para automatizar una tarea concreta (*shell scrip*) nos vendrá bien conocer algunos metacaracteres que nos sirvan para enlazar acciones. Por ejemplo podremos utilizar el metacaracter '||' (barra barra) para representar un OR lógico, '**&&**' para representar un AND o '**;**' para separar dos comandos.

- Ejecutar un listado del directorio */home/agomez* o mostrar el mensaje "*El usuario no existe*" en caso de que el listado devuelva código de error:

```
$ ls -lt /home/agomez || echo "El usuario no existe"
```

- Listar el mismo contenido que en el ejemplo anterior, pero si el usuario existe mostraremos un "OK, directorio listado":

```
$ ls -lt /home/agomez && echo "OK, directorio listado"
```

- Ahora simplemente vamos a enlazar tres comandos. Se ejecutarán en orden secuencial, hasta que no termine uno, no comenzará el otro:

```
$ date; sleep 10;cal
```

Buscar ayuda dentro del Shell

Linux posee diferentes 'canales' para buscar ayuda de forma local sobre la utilización de sus comandos o incluso mini manuales para realizar una tarea concreta, estamos hablando de las páginas de manuales, de extractos de información o pequeños tutoriales que nos harán la vida más fácil dentro de un sistema *Línux* y en concreto con su línea de comandos o *shell*.

- **Páginas man**: Las páginas man son archivos de manuales que se encuentran en formato comprimido y localizadas normalmente en el directorio */usr/lib/man* o */user/share/man.* Para acceder a las páginas de manuales bastará con escribir *$man <comando>* y se desplegarán mediante un paginador (**more** o **less**). Las páginas man suelen estar estructurada por secciones normalmente:
  - NOMBRE
  - SIPNOSIS

- DESCRIPCIÓN
- OPCIONES
- VEA TAMBIÉN
- DIAGNÓSTICOS
- ARCHIVOS
- ERRORES/BUGS
- HISTORIA
- AUTOR

No todas las páginas man cuentan con estas secciones. Podemos buscar ayuda para man desplegando man **$man man**

- **info**: Con el comando info podremos leer la documentación de una forma más fácil que la mostrada por las páginas man. La navegación por la información dada por info suele ser un poco más complicada, pero si nos encontramos con la info de un comando o utilidad que presenta un menú, podremos pulsar 'm' e introducir el nombre de la sección del menú a la que queremos acceder o 'h' si necesitamos ayuda para movernos por la página.
- **HOWTOs**: Son documentos detallados que describen métodos para realizar ciertas tareas, como por ejemplo la configuración de un dispositivo. Un documento *HOWTO* suele cubrir un tema complejo y por ello tienden a ser extensos. Estos documentos suelen encontrarse bajo el directorio */usr/share/doc/utilidad/howto* y normalmente podremos abrirlo con nuestro navegador preferido.
- **FAQ**: Preguntas más frecuentes. Como su nombre indica son documentos en los que se detallan las respuestas a las preguntas más frecuentes que surgen sobre distintas utilidades, comandos o métodos para realizar ciertas tareas. Estos documentos igualmente podremos abrir en su mayor parte con el navegador y localizarlos bajo /usr/share, quizás necesitemos hacer una búsqueda con **find** para ser más precisos.

Linux nos ofrece otras dos herramientas de búsqueda de información que en realidad lo que hacen es derivar su información de las páginas de manual anteriormente mencionadas. Estas herramientas son **whatis** y **apropos**. Estos comandos aceptan como argumentos nombres de comandos o *expresiones* (**-r**) y *wildcards* (**-w**) para buscar en las descripciones de las páginas man y mostrar las páginas man que contienen el patrón. Si usamos ambas herramientas sin opciones *apropos* suele ofrecer más resultados ya que no precisa del nombre exacto del comando, es decir si hacemos un *whatis passwd* y un *apropos passwd*, *whatis* se limitará a resultados exactos para *passwd* mientras que *apropos* mostrará las páginas man de otros comandos como por ejemplo *gpasswd* o *chgpasswd*. Ambas herramientas tienen su símil con el comando man acompañado de opciones, por ejemplo **whatis** es igual a **man -f** mientras que **apropos** es igual a **man -k**. **Conclusión**: Podemos usar *man -f* o *whatis* cuando queramos localizar las páginas man existentes para un determinado comando, mientras que *apropros* y *man -k* servirán cuando no recordemos el nombre de un comando y podamos ir averiguarlo mediante

patrones. También disponemos de pequeños mini tutoriales y FAQ en html, así como información detallada a través del comando *info*.

Comandos GNU/Linux

En esta sección del capítulo estudiaremos los comandos más básicos del *Shell* de *GNU/Linux*, aquellos con los que un administrador de sistema debe estar totalmente familiarizado.

Comandos para la ayuda de otros comandos

Si queremos obtener información sobre la utilización de un comando, podemos hacerlo a través de **man** (*manual*) el cual desplegará a través de un paginador (*more* o *less*) toda la información relativa al comando. No obstante con *man* podremos además de informarnos sobre un comando, aprender el formato de un archivo de configuración o los archivos de dispositivos. Esto es posible gracias a que *man* esta dividido en secciones, siendo la sección uno (la sección por defecto) la relativa a los programás ejecutables y comandos de consola, la sección cinco al formato de los archivos de configuración o la sección cuatro para los archivos de dispositivos. Si queremos conocer todas sus secciones y como utilizar man, podremos pedir ayuda sobre el mismo a través de:

```
$ man man
```

Si queremos abrir una sección concreta lo haremos así:

```
$ man 5 passwd
```

De esta manera en vez de abrir la información sobre el comando *passwd*, se desplegará aquella que muestra el formato y características del archivo */etc/passwd*. Otros comandos con los que podremos encontrar información sobre comandos son **help** e **info**. **Nota**: Si queremos conocer otros comandos de comandos o informarnos sobre el uso de estos, podremos ir a la sección de comandos de este mismo tema: Comandos para la ayuda de otros comandos

Canalizadores y redirecciones

Gracias a una serie de caracteres podremos desviar la información de entrada o salida del *Shell* a distintos archivos, dispositivos o pantallas. Por ejemplo imaginemos que queremos almacenar la salida del comando *ls* (list) en un archivo de manera que posteriormente se lo podamos enviar a un tercero para que estudie tranquilamente el contenido del directorio. Para conseguir esto basta con utilizar el caracter '>'.

```
$ ls -lt /home/agomez/Documentos/ > listado-documentos.txt
```

Con el comando anterior creamos "*listado-documento.txt*" en caso de no existir y/o machacaremos su contenido si ya existiese dicho archivo. Si no queremos machacar el contenido podremos usar '>>' de manera que si no existe el archivo, se creará y si ya existe solo el nuevo contenido será añadido al ya existente. De la misma manera que creamos o añadimos contenido en un archivo a partir de la salida de un comando (lo que estamos haciendo es reenviar lo que se debería de mostrar por la salida estándar **stdout** hacia un archivo, es decir modificamos la *stdout*) podemos redireccionar la salida de error (***stderr***) usando el descriptor **2** de la siguiente manera:

```
$ cat /home/agomez/Documentos/contabilidad.txt 2> stderr-
contabilidad.txt
```

De esta manera si se produjese un error al desplegar el contenido de "*contabilidad.txt*", este se almacenaría en el archivo "*stderr-contabilidad.txt*". Podemos emplear **2>>** de la misma manera que >>, es decir añadiendo contenido en vez de reemplazando. Igual que modificamos la *stdout* podemos modificar la entrada estándar (**stdin**) con el caracter '<'. Por ejemplo imaginemos que tenemos un archivo "*guia.txt*" cuyo contenido corresponde a una guía telefónica la cual tenemos desordenada. Para ordenar esta guía podríamos hacer que la entrada estándar del comando *sort* (ordena) fuese en vez del teclado, este archivo. De esta manera la salida de *sort* será el contenido del archivo "*guía.txt*" pero ordenado. Otra redirección muy útil es el caracter '|' que más que una redirección se trata de un canalizador, tubería o **pipe**. Lo que hace esta pipe es enlazar la salida de un comando de manera que sea la entrada de otro. Por ejemplo podríamos desplegar el contenido de un archivo desordenado y que directamente fuese la entrada al comando *sort* de manera que este lo ordene y lo despliegue ordenado:

```
$ cat desordenado.txt | sort
```

Si además queremos conservarlo en otro archivo:

```
$ cat desordenado.txt | sort > ordenado.txt
```

Podemos aprender más sobre las redirecciones y canalizadores, además de otros comandos como **tee**, **xargs** o **exec** en el apartado de comandos: Redirecciones y canalizadores

Comandos para combinar archivos

Existen ciertos comandos que nos facilitarán el tratamiento de archivos, *sort* podría ser uno de ellos, aunque en esta sección solo nombraremos a **cat**, **join** y **paste**. El comando *cat* es útil si queremos desplegar el contenido de uno o varios archivos, bastará con pasárselos como argumento. Otro utilitario importante de *cat* es cuando queremos almacenar el contenido de varios archivos en solo uno. Para ello solo deberemos de pasarlos como argumentos a *cat* y terminar redireccionando a un archivo único tal que así:

```
$ cat primero.txt segundo.txt tercero.txt > todos.txt
```

Nota: Si queremos realizar la acción de desplegar archivos o añadirlos a otro pero de forma inversa, usaremos el comando **tac**. *join* por su parte combina archivos pero a partir de una columna en común. Para ello los archivos deberán de estar previamente ordenados (podremos usar *sort* para esto), y *paste* desplegará el contenido de varios archivos pero paralelando sus columnas. Por ello ambos archivos deberán de tener el mismo número de líneas. Podemos recurrir a las páginas man de estos comandos o a Comandos para combinar archivos para obtener más información.

Comandos para la transformación de archivos

Existen multitud de comandos para lograr transformar un archivo, todo dependerá de las modificaciones que queramos realizar sobre ellos. Por ejemplo si lo que queremos es convertir una tabulación en espacios (una tabulación equivale a ocho espacios por defecto) utilizaremos el comando **expand**, mientras que si queremos obtener el resultado opuesto, es decir convertir ocho espacios en una tabulación (comprimiremos el archivo) utilizaremos **unexpand**. Ya hemos visto que el comando **sort** puede realizar transformaciones en archivos, por ejemplo modificando su contenido de forma que quede en un determinado orden. Un comando importante cuando tenemos archivos demásiado grandes es **split** con el que podremos dividir un archivo en varias partes. Por ejemplo podremos utilizar su opción **-b** para dividirlo según una cierta cantidad de *bytes* o **-l** para dividir por líneas. Otro comando con el que podremos comprimir un archivo, más bien reducirlo de tamaño para que al ser impreso utilice un menor número de hojas es **mpage**, que lee archivos en texto plano o en formato *PostScript* y les aplica un cierto formato reduciendo el tamaño por páginas. Para terminar, nombraremos los comandos **tr**, con el que podremos convertir caracteres de un archivo (por ejemplo algunas mayúsculas por minúsculas, ciertas letras por otras, utilizar grupos de letras, etc..), **unique**, con el que poder eliminar líneas duplicadas en un archivo, o **iconv** para modificar la codificación de un archivo. Ir a las páginas man de los respectivos comandos o a Comandos para la transformación de archivos para saber más sobre su uso.

Comandos para dar formatos a archivos

Algunos de los comandos con los que podremos dar formato a un archivo son:

- **fmt** : Da formato a los archivos que tienen líneas exageradamente largas, longitudes de líneas irregulares ,etc… Podremos utilizar su opción **-w** para limitar el ancho (en caracteres) de la línea.
- **nl** : Nos permite enumerar las líneas de un archivo de forma sofisticada/compleja. Algunas de sus opciones más básicas serían, **-a** para enumerar todas las líneas, **-t** para enumerar solo las líneas vacias, **-n** para ocultar el número de líneas (útil por ejemplo en encabezados o piés de páginas) o **p***PATRÓN* para enumerar solo las líneas que contengan a *PATRÓN*. Podemos justificar un texto mediante el uso de formatos como **nl** para justificar a la izquierda o **rn** justificar a la derecha.

- **pr** : Prepara un archivo para su impresión añadiendo encabezados (con fecha y hora actual, nombre del archivo y número de página), pies, etc… Acepta opciones como **-t** para eliminar el encabezado, definir un ancho de página (según caracteres) **-w**, definir la longitud de la página **-l** o, **-o** para definir un margen izquierdo.

Nota: Más información sobre estos comandos en Comandos para dar formatos a archivos o en sus respectivas páginas man.

Comandos para la mensajería desde la línea de comandos

Tenemos la posibilidad de utilizar una serie de programás/comandos para enviar y recibir mensajes a determinados (o a todos) usuarios conectados a un sistema (local o remoto), y que se encuentran con una sesión de consola abierta. Mediante el comando **wall** enviaremos un mensaje a todos los usuarios conectados al sistema. Si lo que queremos es enviar un mensaje a la consola de un determinado usuario conectado al mismo sistema utilizaremos **write**. Podemos activar o desactivar los mensajes recibidos en una consola a través de *write* mediante el comando **mesg**, para ello deberemos de utilizar una línea como la siguiente:

- Bloquear los mensajes entrantes por la terminal cuatro (pts4):

```
$mesg n < /dev/pts/4
```

- Volver a activar los mensajes en esta misma terminal:

```
$mesg y < /dev/pts/4
```

Otros comandos son **echo** y **talk**. Con *echo* nos enviaremos a nosotros mismos un mensaje. En realidad más que mandarnos un mensaje lo que hace *echo* es desplegar el texto escrito entre comillas por la salida estándar. Con *talk* mantendremos una conversación bidireccional con otro usuario (tipo chat). Más sobre estos comandos en Comandos para la mensajería desde línea de comandos

Comandos para la visualización de archivos

Para visualizar el contenido de los archivos, ya hemos mencionado el comando *cat*. No obstante existen otros con mayor utilidad, o mejor dicho enfocados principalmente a la lectura de documentos. Estos comandos son **tail**, **head**, **more** y **less**. Con *tail* y *head* podremos desplegar las últimás, o primeras líneas (respectivamente) de un documento. Muy útil para la visualización de logs. Por ejemplo si queremos ver las últimás diez líneas (número de líneas por defecto) escritas en un archivo de de registro, bastará con pasarle el nombre del archivo al comando *tail*. Si en cambio lo que necesitamos es ver las cuatro primeras líneas de un archivo de configuración, bastará con pasarle la opción **-n** y el número de líneas, más el nombre del archivo al comando *head*. *more* y *less* son los paginadores que normalmente utilizaremos para visualizar archivos. Estos

paginadores nos permiten movernos por la pantalla con los cursores además de poder avanzar o retroceder páginas con tan solo pulsar las teclas destinadas a ello. *more* es una versión anterior a *less* por lo que muchas de las opciones de *less* no funcionarán en *more*. Las páginas man de los comandos utilizan normalmente el paginador *less* por defecto para mostrarnos el contenido del manual.

Comandos para resumir archivos

En realidad lo que nos permiten hacer los siguientes comandos, más que resumir archivos es ofrecernos una "porción" de ellos según que opciones utilicemos o desplegarnos información sobre el archivo. Veremos esto fácilmente con los siguientes ejemplos.

- Imprimir tan solo las dos primeras letras de cada línea de un archivo, para lo que utilizaremos el comando **cut**:

```
$ cut -c -2 archivo.txt
```

Nota: Podríamos haber utilizado *1-2* en lugar de *-2*. *–n* representa desde 1 a *n*, mientras que *n–* significa que muestre desde *n* a último.

- Tenemos un archivo con los nombres y primer apellido de una serie de usuarios (uno por línea) separados por un espacio. Vamos a mostrar tan solo el primer apellido, lo que correspondería con el segundo campo de la línea (el primer campo sería el nombre). Por defecto *cut* utiliza el tabulador como delimitador de campo, por lo que tendremos que asignar el espacio **" "** como nuevo delimitador:

```
$ cut -d " " -f2 apellidos.txt
```

- Si lo que queremos es mostrar ciertas características de un archivo, como el número de línea, cuantas palabras o caracteres contiene, etc… usamos el comando **wc**. Vamos a mostrar el número de líneas del archivo apellidos, así veremos cuantos usuarios tenemos registrado:

```
$ wc -l apellidos.txt
```

Importante: Estos comandos suelen usarse normalmente como parte de una cadena de comandos separados por *pipes*. Como hasta ahora, si queremos aprender más, visitaremos las páginas man de estos dos comandos o bien Comandos para resumir archivos

Complexión de comandos y atajos de teclado

- Limpiar la pantalla: **CTRL + L** (igual al comando clear)
- Retroceder media pantalla: **SHIFT + Re Pág**

- Avanzar media pantalla: **SHIFT + Av Pág**
- Desplazarnos una palabra a la izquierda: **CTRL + Flecha izq.** / *Esc + B*
- Desplazarnos una palabra a la derecha: **CTRL + Flecha Der.** / *Esc + F*
- Desplazarnos al inicio de una línea: **CTRL + A**
- Desplazarnos al final de una línea: **CTRL + E**
- Borrar un carácter (el que está sobre el cursor): **CTRL + D** / *Supr*
- Eliminar la última palabra escrita: **CTRL + W**
- Borrar todo el texto entre el cursor y el final de linea: **CTRL + K**
- Borrar todo el texto entre el cursor y el principio de linea: **CTRL + U**
- Intercambiar o desplazar la letra anterior al cursor un carácter hacia adelante: **CTRL + T**
- Intercambiar o desplazar la palabra anterior al cursor una palabra hacia adelante: **Esc + T** ; González Mesas, si el cursor está en la M y pulsamos *Esc+T* ahora quedará Mesas González.
- Cambiar de minúsculas a mayúsculas los caracteres que hay entre el cursor y el final de la palabra: **Esc + U**
- Cambiar de mayúsculas a minúsculas los caracteres que hay entre el cursor y el final de la palabra: **Esc + L**
- Cambiar a mayúscula la letra que hay sobre el cursor, si el cursos está en el espacio, se cambiará la primera letra de la siguiente palabra: **Esc + C**
- Limpiar el historial: **history -c**
- Buscar comando en el historial hacia arriba (comandos más antiguos): **Flecha arriba** / **CTRL + P**
- Buscar comandos en el historial hacia abajo (comandos más nuevos): **Flecha abajo** / **CTRL + N**
- Buscar comandos en el historial por caracteres: **CTRL-R** y escribo caracteres (vuelvo a pulsar CTRL + R para recorrer la lista de comandos que contienen esos caracteres), **CTRL-G** para salir de la búsqueda
- Ejecutar un comando del historial conociendo su número en la lista: **!210**
- Abrir un editor de texto: **CTRL + X** + **CTRL + E**

Uso de expresiones regulares

Las expresiones regulares son patrones con una sintaxis específica utilizadas para igualar cadenas de caracteres, ya sea desde la ausencia de caracter, a uno o más caracteres. Las expresiones regulares se utilizan con diferentes herramientas de *Linux* ya sea para buscar dentro del contenido de un archivo especifico mediante un editor de texto, buscar con comandos como *grep* o *sed* dentro de un archivo para procesar líneas o incluso con comandos como *find* para localizar archivos en el disco. A continuación veremos algunos caracteres especiales o agrupaciones de ellos con los que deberemos de estar familiarizado para hacer un buen uso de las expresiones regulares.

Metacaracteres y rangos

- \ : Sirve para escapar un caracter especial dentro de una expresión regular

Ejemplo: . (punto) hace referencia a cualquier caracter, **\.** hace referencia a "**.**" (punto)

- / : Sirve para acotar una expresión regular:

Ejemplo: /expresión regular/

- ^ : Indica que la línea comienza con un determinado caracter. Dentro de corchetes y al principio su significado es de inversión.

Ejemplo: **^a** la línea comienza con el caracter '**a**', **[^a]** la línea comienza con cualquier caracter menos con '**a**'

- **$** : Indica que el final de línea termina con un determinado caracter.
- **.** (punto) : Hace referencia a cualquier carácter, pero solo a 1. No incluye el salto de línea.
- **?** : Hace referencia a 0 o 1 carácter. En expresiones regulares también puede representar la opcionalidad de aparición de uno o varios caracteres.
- **+** : Referencia a 1 o más caracteres
- ***** : Refiere a 0 o más caracteres
- **[]** : Referencia a un conjunto de caracteres (expresiones entre corchetes) o rango (expresiones de rango).

Ejemplo: **c[ao]ma** (cama o coma), **c[1-2A-Z]ma** (c2ma, c1-ma, cIma, cLma…)

- **()** : Al igual que con expresiones aritméticas, sirve para agrupar una parte de la expresión
- | : Significa opción 'OR'.

Ejemplo: **cama|coma** , la posibilidad de concordar con cama o coma dentro de un texto

- **{}** : Indica un rango de repetición.

Ejemplo: **r{1,2}** indica 1 o 2 concurrencias de **r**

- **[:alnum:]** : rango alfanumérico. Equivale a [0-9a-zA-Z] incluso es más apropiado, pues no depende de la codificación del archivo.
- **[:alpha:]** : rango alfabético [a-zA-Z]
- **[:digit:]** : Dígitos [0-9]
- **[:xdigit:]** : Dígitos hexadecimal
- **[:lower:]** : Minúsculas
- **[:upper:]** : Mayúsculas
- **[:space:]** : Espacios
- **[:print:]** : Imprimibles

- **[:graph:]** : Gráficos

Diferencias entre expresiones regulares básicas y extendidas

La diferencia entre expresiones **básicas** y **extendidas** "*es simple*", las expresiones *básicas* no hacen uso de caracteres especiales como **?**,+,|, **()**, **{}**… por lo que su utilidad se ve disminuida.

Algunos ejemplos de *expresiones regulares*

- **[0-9]*** : Cadena vacía, 1, 43, 6542, etc…
- **^.*$** : Cualquier línea completa
- **[ab]*** : a, aaaaaa, aa, bb, b, bbbb….
- **(ab)*** : a, b, ba, ab, bababa, abab, aaaaabbbbabab…
- **^[0-9]?b** : Puede o no (?) empezar por un dígito o por b. Ejemplo 3b, b, 0b…
- **([0-9]+ab)*** : El * fuera del paréntesis indica que puede existir una cadena vacia dentro del parentesis, también infinitas a y b, pero el + del corchete obliga a que haya uno o más digitos númericos. Ejemplo: cadena vacia, 023a, 1aa34bbaa23….
- /^#/ : Cualquier línea que comience por #
- /^$/ : Cualquier línea en blanco
- **/}$/** : Cualquier línea que termine con }
- **/} *$/** : Cualquier línea que termine con } y tenga 0 o más espacios

Comandos útiles en expresiones regulares: grep y sed

grep

El comando grep (*Global Regular Expresion Print*) busca un patrón o cadena simple de caracteres dentro del contenido de un archivo. grep no efectúa cambios en el archivo simplemente despliega a pantalla cada línea del archivo que contiene o iguala el patrón de búsqueda. Existen otros dos comandos relacionados con grep que en realidad no son más que un '*front end*' de dos de las opciones que se le pueden pasar a grep, estos son egrep (grep -E) y fgrep (grep -F). Las diferencias son algunas como por ejemplo el uso de expresiones regulares en grep, permitiéndonos egrep utilizar expresiones extendidas, además de una ligera diferencia en la sintaxis del comando. Por su parte fgrep utiliza cadenas simple de caracteres en vez de expresiones regulares. Estudiaremos el uso de estos comandos en el apartado de comandos de este mismo capítulo. No obstante dejaremos un ejemplo del uso simple de cada uno de ellos:

- Localizar con **grep** la palabra 'computación' en el archivo "*Fundamentos sobre Sistemás Operativos.odt*"

```
$ grep computación "Fundamentos sobre Sistemás Operativos.odt"
```

- Aquí vamos a complicar la cosa solo para mostrar que con **egrep** es posible utilizar caracteres especiales formando expresiones entendidas, algo que con *grep* sin su opción **-E** no sería posible. Vamos a suponer que tenemos un archivo de red con la siguiente información(*) y queremos listar las líneas 2 y 3 (insisto que como casi siempre esto se podría haber realizado de manera más simple, pero está realizado así para identificar diferentes caracteres especiales):

```
 $ cat red.txt
eth0 Link encap:Ethernet HWaddr 88:ae:1d:69:5a:5a
 inet addr:192.168.1.33 Bcast:192.168.1.255 Másk:255.255.255.0
 inet6 addr: fe80::8aae:1dff:fe69:5a5a/64 Scope:Link
 UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
 RX packets:273267 errors:0 dropped:0 overruns:0 frame:0
 TX packets:174060 errors:0 dropped:0 overruns:0 carrier:0
 collisions:0 txqueuelen:1000
 RX bytes:350125144 (350.1 MB) TX bytes:24182345 (24.1 MB)
 Interrupt:16
$ egrep "^ *inet *[a-z]*:[0-9]*(192.168)|^ *inet6" red.txt
 inet addr:192.168.10.13 Bcast:192.168.10.255 Másk:255.255.255.0
 inet6 addr: fe80::8bbj:1dcf:fe96:5h5h/64 Scope:Link
```

- Localizar con **fgrep** el nombre y apellido de un determinado alumno en el archivo *alumnos.txt*

```
$ fgrep "Julio Ortiz" alumnox.txt
Julio Ortiz Maldonado
```

sed

Stream EDitor, es un editor de línea no interactivo. Útil para procesar textos extendidos. El comando sed trata el contenido de los archivos pero al igual que grep no los modifica. No obstante podemos redirigir el contenido modificado a un archivo haciendo permanente estos cambios. El comportamiento por defecto de sed es imprimir todas las líneas procesadas se hayan "aplicado" cambios o no, evidentemente este comportamiento puede ser modificado mediante opciones de sed. Sed tiene sus propios comandos que serán los encargados de ejecutar una acción u otra sobre el archivo en cuestión, comandos que podrán ser pasados igualmente a través de la línea de comandos o un archivo en cuya línea se identifique un comando a ejecutar por sed. Además sed permite el uso de expresiones regulares para realizar las búsquedas en el contenido de los archivos. La sintaxis de sed puede ser complicada e incluso representada de diferentes formás (en el apartado de comandos para el uso de expresiones regulares de este mismo capítulo hablaremos más sobre sed). La sintaxis básica de ser es:

```
sed [-n] [-e comandos] [-f script-de-comandos] archivo_a_procesar.txt
> archivo_para_guardar_cambios.txt
```

La opción **-n** evita que sed envíe su contenido modificado a la salida estándar a menos que la opción **-p** (print) haya sido igualmente declarada. Con **-e** podremos pasar comandos. Algunos comandos podrían ser: **d** (elimina), **s** (sustituye), **q** (imprime las líneas especificadas y sale) y **p** (imprime). Podemos pasar varios comandos separados por comás '**,**' y encerrados entre **{**comando,domando2..**}**. La opción .f indica a sed que los comandos serán pasados mediante un script de comandos. Los comandos aceptan direcciones, que son las líneas identificadas bien por su número o por patrones, y que servirán para tratar exactamente dichas líneas en vez del contenido completo de un archivo

```
$sed -n '4,6 !p' archivo.txt
```

En el ejemplo anterior no (negación **!**) se imprimirán (comando **p**) las líneas (dirección) de la 4 a la 6 Los comandos de sed además de aceptar direcciones y expresiones regulares aceptarán flags, que son etiquetas que modifican la funcionalidad del comando. Por ejemplo el comando s (sustituir) por defecto sustituye la primera ocurrencia del patrón, pero si se le añade la etiqueta g serán sustituidas todas aquellas ocurrencias de patrón.

```
$sed 's/\t/;/g' archivo.txt
```

El comando anterior sustituye todas (**g**) las tabulaciones (**\t**) del fichero *archivo.txt* por el carácter **;** El comando sed se estudiará de forma más completa (sintaxis, opciones, comandos, etiquetas, expresiones regulares y direcciones en el apartado de comandos para expresiones regulares de este mismo capítulo).

Consejos para posibles preguntas en este tema:

- El método más seguro para ejecutar un programa es estando en el directorio actual, escribir, ./nom_prog, en el caso de usar solo el nombre, es posible que hayan modificado la ruta del PATH malintencionadamente y hayan colocado una ruta que desconoces en la que se encuentra una copia de nom_prog maliciosa, será esta la que se ejecute al no especificar el directorio de trabajo (.)
- Suelen ser preguntas tontas o básicas, te recomiendo que las leas detenidamente (como en cualquier examen tipo test) si no quieres tirarte de los pelos cuando te den la nota.
- Para trabajar desde línea con un programa interactivo y querer guardar la salida en un archivo, no uses ">", usa $nom_prog | tee archivo.txt ya que este comando divide la salida estándar.

LPIC-1 Capítulo 2

EN ESTE CAPÍTULO SE ABARCAN LOS SIGUIENTES OBJETIVOS DEL EXAMEN:

- 102.3: Administrar bibliotecas compartidas (1)
- 102.4: Usar los administradores de paquetes Debian (3)
- 102.5: Utilizar los administradores de paquetes RPM y Yum (3)
- 103.5: Crear, monitorizar y destruir procesos (4)
- 103.6: Modificar las prioridades de ejecución de procesos (2)

Herramientas de administración del software: RPM y Debian

Las herramientas principales para la administración de paquetes Linux son: **RPM** y **YUM** (*Red Hat*) y **DPKG**, **DSELECT** y **APT-GET** (entre otros, para *Debian*). La mayoría de las distribuciones instalan un solo sistema de paquetes, aunque es posible instalar más de uno (incluso algunos programás como alien, requieren ambos, para una funcionalidad completa), pero se desaconseja esta práctica, debido a que las bases de datos de cada gestor de herramientas son disyuntas, pudiendo producir esto, como mínimo errores en versiones de paquetes y dependencias.

Dejar claro que **alien** podrá producir estos errores, si intentamos instalar software que previamente estaba instalado ya con su gestor de paquetes, ya sea *RPM* o *Debian*.

RPM (Red Hat Package Manager)

RPM es el administrador de paquetes más popular, creado por *Red Hat*. Algunas distribuciones hacen uso de ese gestor como *Fedora* y *CentOS* (derivados de RH), *Mandriva, Yellow Dog* (Power PC) y *SUSE*. *RPM* es compatible con arquitecturas *x86, x86_64, PowerPC, IA-64, Alpha y SPARC.*

Convención para nombrar paquetes *RPM*:

`nombre_paquete-a.b.c-x.arq.rpm`

samba-client.3.6.5-86.fc17.1.x86_64.rpm

- *samba-client* : Nombre de paquete
- *3.6.5* : Nº de versión
- *86.fc17.1* : El número de compilación o release.
- *x86_64* : Indica que es una versión adaptada a la arquitectura x86_64

Nota: Hemos puesto un ejemplo en el que además se ha añadido un nombre que distingue para que distribución está compilado el paquete, en este caso indica que es la compilación **86.fc17.1** de *Fedora*.

RPM es un sistema de administración de paquetes muy flexible, es comparable al administrador de paquetes Debian y ofrece muchas más funcionalidades que los *tarball*. Existen dos herramientas principales para la gestión de paquetes RPM.

- **rpm** que instala paquetes/archivos
- **yum** que es un meta-empaquetador que instala de forma cómoda los paquetes y sus dependencias.

El archivo de configuración de *RPM* se encuentra en: */usr/lib/rpm/rpmrc*. La mayoría de las opciones de este archivo son relativas a la optimización de la CPU a la hora de compilar un paquete fuente. Cuando se quieren realizar cambios, no se modifica este archivo directamente si no que se modifica */etc/rpmrc* (para efectos globales) o *~/.rpmrc* para realizar cambios en función del usuario.

¿Como obtener archivos individuales de un paquete RPM? existen al menos 2 posibilidades, bien con el comando *alien* (el cual convierte un paquete *RPM* en paquete *Debian* o *tarball*) o bien utilizando el comando **rpm2cpio** para convertir un paquete *rpm* en un archivo *cpio* y posteriormente obtener archivos individuales de ese paquete con el comando **cpio**, pero ¿Que es un archivo cpio?, Un archivo *cpio* es un tipo de archivo principalmente creado para el almacenamiento de copias de seguridad en cintas magnéticas de una forma contigua y tiene un funcionamiento parecido al formato tar, a diferencia que se forma por una serie de archivos y/o directorios tanto con los encabezados usados por *GNU CPIO* para extraer el archivo, como con encabezados extras que sirven para identificar el nombre, fecha de creación, permisos y propietarios de cada archivo y/o directorio.

Nota: Cuando descomprimimos un archivo **cpio**, lo ideal es usar un directorio vacío a modo de contenedor, para que no modifiquemos o reemplacemos ningún archivo del sistema.

Los sitios webs http://rpmfind.net y http://freshrpms.net incluyen enlaces a *RPM* compilados por los autores de los programás, *RPMs* de distribuciones específicas o los compilados por terceros.

YUM (*Yellow dog Updater Modified*) es un meta-empaquetador que nos permite instalar con facilidad un paquete y todas sus dependencias desde un único comando. **yum** busca estos paquetes en unos repositorios que se encuentran configurado en el sistema (*/etc/yum.repos.d/*) que son "cajones en Internet" donde hay paquetes disponibles para su descarga e instalación, selecciona la versión más actualizada y la descarga e instala, por el contrario si solo quiere descargar el paquete para posteriormente instalarlo en un OS que no se encuentra conectado a Internet puedes usar el comando **yumdownloader**. No todas las distribuciones que usan *RPM* utilizan *YUM*, **SUSE** y **Mandriva** utilizan su propio meta-empaquetador, a diferencia de las distribuciones basadas en *Debian* que si que utilizan el meta-empaquetador **apt-get**.

El archivo de configuración de *yum* se encuentra en */etc/yum.conf*, además de otros archivos de configuración adicional, localizados en */etc/yum.repos.d/* (equivalen a la configuración de los repositorios mencionados anteriormente), o */etc/yum.repos.d/.yum.conf* que especifica varias acciones a realizar por *yum* a la hora de ser ejecutado.

Existen alternativas de *yum* para entornos gráficos como: **kyum** y **yumex**

DEBIAN

Al igual que en *RPM*, los paquetes **Debian** son utilizados por diferentes distribuciones que derivan de *Debian*, como pueden ser **Ubuntu**, **Linux Mint** o **Xandros** e igualmente funcionan en diferentes arquitecturas. Prácticamente igual que en *RPM*, los paquetes *Debian* siguen una convención de nomenclatura para sus paquetes, estos incluyen información de dependencias, archivos y utilidades, además, igualmente se usa una herramienta para la instalación de estos (**dpkg**), o otras más cómodas que tiene el efecto de *yum* en *RPM* (la de instalar paquetes y dependencias de forma cómoda) como **APT-GET**, **dselect**, **aptitude** o **Synaptic** (*GUI*).

Dentro del sistema de paquetes *Debian* podemos distinguir entre cinco tipos de paquetes según su grado de dependencia con el propio sistema:

- **Required**: Se trata de paquetes indispensables para el correcto funcionamiento del sistema.
- **Important**: Se trata de paquetes que deberían de estar presentes en cualquier sistema tipo *UNIX*.
- **Standard**: Paquetes que comúnmente se encuentran en un sistema *GNU/Linux*. No son indispensables para el correcto funcionamiento del sistema.
- **Optional**: Paquetes que pueden o no estar en un sistema *GNU/Linux*, como por ejemplo los referentes al sistema gráfico (en el caso de que vayamos a usar el sistema en modo servidor (minimalista).

- **Extra**: Paquetes no integrados como *optional* debido a que puedan surgir conflictos con paquetes de mayor prioridad o requiere un tipo de configuración especial.

Nota: Podemos ver la prioridad de un determinado paquete mediante el comando **apt-cache show <paquete> | grep Priority** o **aptitude show samba |grep Priority**

A diferencia de *YUM*, *APT* usa el archivo */etc/apt/source.list* para especificar las ubicaciones desde las que se pueden obtener los paquetes, ya sean directorios de ubicaciones en CD-ROM, sitios FTP o webs.

/etc/apt/source.list Es el archivo de texto en el que añadiremos los **PPA** (*Personal Package Archives*), es decir las direcciones desde las que queremos obtener paquetes. Estas direcciones pueden tener un un formato tipo "*deb http://packages.linuxmint.com rebecca main upstream import #id:linuxmint_main*" si se trata de un repositorio de paquetes web, o cambiando *http* por *ftp*, *smb*, *file* o *cdrom* (dependiendo de que fuente de paquetes se trate).

Nota: Si la dirección pertenece a una *URL* donde se encuentra código fuente en vez de paquetes binarios, será **deb-src** lo que aparezca en lugar de **deb**

También es posible que cuando hayamos añadido un repositorio este se encuentre en **/etc/apt/source.list.d/nombrerepo.list** y dentro de este archivo la *URL* del repositorio. Existen programás específicos para ciertas distribuciones como es el caso de *Ubuntu* que añaden o eliminan de forma automática tanto el repositorio como la clave pública (*GPG*).

Instalar un nuevo *PPA* de forma automática en *Ubuntu*:

```
$ sudo add-apt-respository ppa:nombrerepo/ppa
```

Para eliminarlo:

```
$ sudo add-apt-repository --remove ppa:nombrerepo/ppa
```

En *Debian* podremos añadir un *CD-ROM* como fuente mediante el siguiente comando:

```
$ sudo apt-CD-ROM add
```

Si no tenemos la posibilidad de utilizar estas herramientas en nuestra distribución, deberemos de añadir los *ppa* a mano y eliminarlos mediante la edición del archivo *source.list* o con **rm** si se trata de un archivo en */etc/apt/source.list.d/*

Ya hayamos realizado cambios (instalar, editar o eliminar un *ppa*) de forma automática o manual, deberemos de actualizar la información mediante el comando **apt-get update**. A partir de este momento dispondremos del nuevo *ppa* desde el que poder

instalar un nuevo software que no encontrábamos en nuestros antiguos repositorios utilizando por ejemplo **apt-get install <nombre_programa>**.

dpkg hace la función de *rpm* en sistemás basados en *RPM*, es decir, instalador de paquetes. Sus archivos principales se encuentran en */var/lib/dpkg* y el archivo de configuración en: */etc/dpkg/dpkg.cfg*

apt-get es una herramienta de línea de comandos y es semejante a la herramienta *yum*, utilizada en sistemás *RPM* para la instalación de paquetes y sus dependencias. *apt* tiene su archivo de configuración el cual también afecta a la herramienta *dselect*: */etc/apt/apt.conf*. Si quisiéramos ver algunos ejemplos de configuración podemos acceder al archivo de ejemplo: */user/share/doc/apt/examples/apt.conf*

apt-cache es una herramienta que nos permite mantener actualizada y saneada la base de datos de paquetes para *APT*

dselect es una herramienta potente con una gran cantidad de opciones nada obvias y que posee una interfaz de usuario interactiva en modo texto. Para usarla basta con escribir el comando *dselect* en la línea de comandos y aparecerá un menú de tipo texto. *dselect* funciona como una interfaz de *dpkg*. También es posible pasarle opciones y comandos con la siguiente sintaxis:

```
dselect [opciones] [comandos]
```

El archivo de configuración principal se encuentra en: */etc/dpkg/dselect.cfg*

aptitude es bastante similar a *dselect* por su formato interactivo , aunque *aptitude* incorpora menús por los que nos podemos mover usando **Control-T**. *Aptitude* también permite que se le pasen comandos por la línea de comandos.

Synaptic es una herramienta de entorno gráfico similar en muchos aspectos a *dselect* y *aptitude*, pero de fácil manejo.

Nota: Por lo general **dselect**, **aptitude** y **synaptic** son herramientas para encontrar software de los que desconoce su nombre exacto.

APT también puede funcionar en sistemás basados en *RPM* (http://www.apt4rpm.sourceforge.net).

Estudiaremos más sobre los comandos de paquetes deb en su respectivo apartado de comandos al final del tema.

Convirtiendo paquetes *RPM* y *Debian* en *tarball*

Como vimos con el gestor de paquetes *RPM*, es posible convertir paquetes *rpm* en paquetes **tarball** y *debian*, pues para *Debian* pasa lo mismo, una buena solución para convertir paquetes *rpm* o *tarball* en paquetes *Debian* es mediante el software *Alien*, que como ya comentamos, es obligatorio tener el sistema de paquetes *RPM* y de *Debian* instalados en el mismo sistema. Una pega es que *alien* no convierte toda la información de las dependencias de manera totalmente correcta. Cuando convierte desde un *tarball*, *alien* copia los archivos exactamente igual que se encontraba en el paquete *tarball*, por lo que es primordial asegurarnos de que el paquete *tarball* contiene una ruta de directorios correcta, es decir, o descomprimimos el paquete en un directorio contenedor y luego creamos enlaces, copiamos o sustituimos los archivos originales, o bien desempaquetamos el contenido, creamos una estructura acorde, lo volvemos a empaquetar y cuando descomprimamos, cada archivo se situará en su directorio.

Para conocer los comandos de conversión, vaya a la página de comandos LPIC-1
Convertir paquetes

Soluciones para problemás de dependencias

Forzar la instalación: Una solución es ignorar el problema, esto no es lo adecuado pero dependiendo de donde se dé la dependencia sabremos si es buena opción o no. Por ejemplo:

Si hemos compilado un paquete y a la hora de instalarlo nos diese problemás de dependencias, podríamos en principio ignorarlo.

- *RPM*: Opciones —**nodeps** y **–force** (*rpm -i nombrepaquete.rpm –force/–nodeps*)
- *DPKG*: Opciones **–ignore-depends=paquete**, **–force-depends** ó —**force-conflicts**

Actualizar o reemplazar el paquete del que se depende: Es una buena forma de solucionar conflictos de dependencias, pero a veces engordamos el problema si el paquete que necesitamos no es el adecuado para nuestra distribución, en este caso, quizás nos aparezcan otros errores de dependencias en otros software que dependen de ese paquete, de su versión antigua (la instalada anteriormente) o bien que a la hora de volver a actualizar el programa o el sistema, no nos reconozca ese paquete, esto sucede porque interpretará las dependencias de forma que lo hace su distribución y no en la que la hemos instalado.

Recompilar el paquete problemático: Algunas dependencias surgen de bibliotecas u otras herramientas instaladas en el ordenador que compiló el paquete y no del código fuente del programa, por lo tanto, si volvemos a compilar el paquete en otro ordenador que tiene otras herramientas y software instalado, algunas dependencias cambiarán. En el caso de distribuciones basadas en *Debian* es raro que tenga que compilar un paquete a partir de su paquete fuente, pero para *RPM,* podremos usar el comando **rpmbuild** con la

opción **–rebuild** y el paquete fuente. Esto nos puede generar uno o más paquetes binarios en */usr/src/nom_distro/RPMS/tipo_arq*

Localizar otra versión del software a instalar: Podemos volver a una versión anterior o posterior, siempre y cuando esto no produzca errores de seguridad o simplemente esa versión no satisfaga nuestras necesidades por tener implementaciones nuevas o bugs no corregidos. Esto es común cuando queremos instalar un software que además necesita actualizar bibliotecas compartidas y no queremos, ya que otros programás necesitan de esas, en este caso, una versión anterior podría solucionar esto.

Bibliotecas compartidas o dinámicas

Son "fragmentos" de los programás más utilizados, referenciadas por el ejecutable principal de un programa, el cual omite la mayoría de las rutinas de bibliotecas.

Las bibliotecas compartidas son identificadas por su extensión **.so** o **.so.versión**, a diferencia de las *estáticas* que son incluidas dentro del mismo programa, y tienen la extensión **.a**.

Ventajas del uso de las bibliotecas compartidas o dinámicas:

- Le facilita la vida al programador.
- El programa final tiene un tamaño mucho menor, lo que significa que necesita menos espacio en disco y consume menos RAM y si encima son utilizadas por muchos programás de forma paralela, el uso de RAM o el espacio en discos no es redundante.
- Como regla general, los programás que dependen de bibliotecas compartidas se benefician de la actualización de estas de forma inmediata. No se necesita una actualización del programa ni volver a compilarlo, referenciando a una nueva biblioteca.

Inconvenientes del uso de las bibliotecas compartidas o dinámicas:

- Los cambios en una biblioteca pueden ser incompatibles con uno o todos los programás que hacen uso de ella. Para esto, Linux utiliza sistemás de numeración, para que se puedan mantener varias versiones de una misma biblioteca.
- Para que un programa se ejecute bien, necesita encontrar las bibliotecas, esto se hace mediante archivos de configuración y variables. El archivo de configuración general es */etc/ld.so.conf*, este a su vez puede contener una línea como "*include /etc/ld.so.conf.d/*.conf*" que indica que cargue también los archivos de configuración de dicho directorio, ya que esos archivos pueden contener rutas de otras bibliotecas referenciadas.

- Al existir multitud de bibliotecas compartidas, esto puede producir una maraña de referencias a bibliotecas que puede resultar tedioso.
- El sobre-escribir, eliminar, o cualquier otra razón por la que una biblioteca quede inaccesible, puede producir desde que un programa no inicie, hasta que el mismo sistema no arranque.

Nota: Algunas distribuciones, como por ejemplo *Gentoo*, poseen características diferentes. Para modificar *ld.so.conf*, habría que añadir o editar los archivos del directorio */etc/env.d* y luego utilizar la herramienta **env-update**, que lee los archivos de este directorio creando una "ruta final" para referenciar las bibliotecas. Estos métodos cambian las referencias a las bibliotecas de forma permanente y global.

Podemos configurar de forma no global, rutas de referencias a bibliotecas gracias a la variable de entorno **LD_LIBRARY_PATH**, por ejemplo, si quisiéramos probar el efecto de unas nuevas bibliotecas sin tener que definirlas de forma global, las almacenaríamos en un directorio y añadiríamos dicha ruta a esta variable, de forma que para esa sesión o futuras sesiones (en caso de que la configurásemos en un script de inicio de usuario) el sistema busque en determinado path archivos de configuración de bibliotecas.

```
$export LD_LIBRARY_PATH=/usr/local/pruebas:/opt/otraslib
```

Puede ocurrir que al ejecutar un programa nos muestre un mensaje de error, en el que se informa de que la biblioteca no se ha encontrado. Lo más normal es que la biblioteca no esté instalada, o bien el nombre o ruta de esta, difiera con el que el binario del programa pide. Para solucionar estos problemás lo normal es seguir el rastro del paquete y comprobar que librería está pidiendo y luego comprobar si está instalada en otro directorio no referenciado, o tiene otro nombre. Si directamente no está instalada, la buscaremos por la web, repositorios, etc… si está instalada con otro nombre o en otro directorio, bastará con crear un enlace simbólico desde la que tenemos hacia la que pide.

Linux posee un par de programás para la administración de bibliotecas.

- **ldd** : Muestra las bibliotecas de las que depende un programa, si le pasamos el nombre de una biblioteca, entonces mostrará la(s) biblioteca(s) de las que esta depende.
- **ldconfig** : Lee los archivos de configuración (*/etc/ld.so.conf*) e implementa todos los nuevos cambios (actualiza la caché) y como trabajo secundario actualiza todos los enlaces simbólicos de librerías.

Nota: Los programás *ld.so* y *ld-linux.so* son los que gestionan la carga de bibliotecas, y estos no leen los archivos de configuración, leen directamente el archivo **/etc/ld.so.cache** que es un binario que contiene toda las rutas de directorios y archivos (un formato mucho más eficaz que en texto plano). El inconveniente es que cada vez que se añade un archivo o ruta hay que refrescar esta caché, para eso el comando *ldconfig*.

Compilación de paquetes

Lo primero que podemos preguntarnos si somos usuarios nóveles en *GNU/Linux* es ¿Que es compilar?, pues bien, la definición de compilar podría ser algo como "*reunir una serie de objetos necesarios en una misma unidad con el fin de poder hacer que esta sea totalmente funcional*". En *GNU/Linux* cuando hablemos de compilar un paquete nos estaremos refiriendo a la transformación de un programa escrito en código fuente (el cual es legible por nosotros, los humanos) en código máquina, de manera que pueda ser interpretado por el sistema haciéndolo este funcionar.

Vamos a explicar el procedimiento de compilación mediante un ejemplo: Supongamos que queremos instalar un programa/paquete para el cual no encontramos el respectivo *.rpm* o *.deb* de nuestra distribución o sistema específico (hardware, arquitectura, etc..) pero en cambio si tenemos la opción de bajar el código fuente del mismo. Lo primero que deberemos de hacer será descargarlo a nuestra máquina. Una vez descargado comprobaremos que se trata de un archivo *.tar*, *.tar.gz* o *tar.bz2* (aunque es posible igualmente encontrarlos en *.zip* o en otros sistemás de compresión).

Una vez tengamos el código fuente (archivo *.tar*, *.tar.bzp2*, etc..) en nuestra máquina deberemos de comprobar si tenemos algún compilador instalado en el sistema. En *GNU/Linux* es común utilizar **GCC** (compilador para C, C++, Java, Fortran y otros códigos de programación que se distribuye como software gratuito bajo licencia *GPL* (*General Public License*)) que podremos comprobar mediante la siguiente orden en consola:

```
# gcc --version
```

Si no obtenemos resultados deberemos de proceder a su instalación mediante alguna de las herramientas de instalación de paquetes anteriormente estudiadas.

Ahora que ya tenemos instalado *gcc*, vamos a descomprimir el código fuente. Dependiendo de la extensión del paquete deberemos de utilizar una u otra herramienta, por ejemplo si es un *.zip* utilizaremos:

```
# unzip nombre_paquete.zip
```

Si es un *tar.gz* o *tar.bz2* utilizaremos la herramienta **tar** (esto lo veremos en capítulos posteriores, no obstante vamos a indicar como descomprimir exclusivamente este paquete) tal que así:

```
# tar -zvxf nombre_paquete.tar.gz
```

o

```
# tar -jxvf nombre_paquete.tar.bz2
```

Esto nos creará un nuevo directorio donde se almacenarán los archivos que contenía el paquete antes de ser descomprimido. Accederemos a este directorio mediante el comando **cd**. Dentro de este directorio buscaremos los archivos *README* e *INSTALL* donde se nos presentará una serie de instrucciones para la instalación y requisitos necesarios que deberemos de instalar en caso de no encontrarse instalados en nuestro sistema.

Ahora con "todos" los requisitos supuestamente instalados y una idea de lo que vamos a realizar para instalar el paquete o haber obtenido información sobre este mediante los archivos anteriormente citados, ejecutaremos el comando "**./configure**" con el que procederemos a verificar los requisitos (nos alertará si faltase alguno) y preparar la compilación.

Nota: Aunque por regla general "todos" los paquetes suelen llevar una ruta predefinida válida (la cual suele estar definida bajo el directorio */usr*), podremos utilizar el parámetro *–prefix=/ruta/de/instalación* para especificar una ruta alternativa en la que instalar el nuevo programa. Es importante que esta ruta este en nuestra variable *PATH* de manera que podamos ejecutar el nuevo programa sea cual sea nuestro directorio de trabajo.

Con todo listo y todos los requisitos cumplidos procederemos a compilar. Para ello utilizaremos el comando **make**. Esto producirá una salida en la que iremos comprobando el proceso de compilación. Esta tarea podrá tardar desde segundos a horas (depende de numerosos factores).

Nota: Si tenemos un procesador con más de un núcleo podremos utilizar la opción *-j <num>* para especificar el número de hilo (*thread*) a utilizar durante el proceso de compilación.

Bueno ya tenemos compilado nuestro programa ahora queda instalarlo. Es importante saber si tenemos permisos en el directorio de instalación, bien en el que viene por defecto (que se puede determinar dependiendo del paquete que estemos compilando e instalando) o en el que hayamos prefijado nosotros mediante la opción *–prefix=* de *./configure*. De no tener los permisos suficientes deberemos de ingresar como *root* (**su –**), para lo que necesitaremos conocer su *password* o bien utilizar **sudo** (si es que el archivo */etc/sudoers*, algo que veremos en otros capítulos, no lo permite). Sin más ejecutamos el comando **make install**.

Si todo ha ido bien, ya tendremos nuestra aplicación instalada la cual podremos ejecutar desde cualquier directorio (recordemos que el directorio en el que se encuentra el binario de la aplicación debe estar añadido como valor a nuestra variable *PATH*). Para finalizar no estaría de más hacer un poco de limpieza, algo que hará fácilmente el comando **make clean**.

Nota: Este comando es necesario ejecutarlo, si tras la compilación obtuvimos errores y queremos volver a compilar. Antes de continuar deberemos de ejecutar *make clean*.

Si por cualquier motivo queremos desinstalar la aplicación, utilizaremos el comando **make unistall**, pero antes deberemos de movernos al directorio en el que se encuentran los archivos de compilación y el código fuente (sí, el que se creo cuando descomprimimos al principio el paquete comprimido del código fuente) y autentificarnos como *root*.

Controlando los procesos

Para controlar los procesos antes, deberemos de saber algunas características básicas sobre estos, como por ejemplo los estados que un proceso puede adoptar.

Los procesos pueden encontrarse en diferentes estados, por ejemplo, **D** se identifica con un proceso que se encuentra en estado *ininterrumpible* (proceso Input/Output), **S** referencia a un proceso dormido (*sleep*), **R** que se encuentra ejecutándose (*running*), **T** parado, **X** muerto y con **Z** se indica que un proceso se encuentra en estado '*zombie*' o '*defunct*' (en proceso de defunción).

Nota: Un proceso en estado *zombie* es un proceso que ha completado su ejecución pero aún tiene una entrada en la tabla de procesos, permitiendo al proceso que lo ha creado (su proceso padre) leer el estado de su salida. Al padre se le envía una señal **SIGCHLD** indicando que el proceso ha muerto, esta señal ejecuta la llamada al sistema **wait**, que lee el estado de salida y elimina el proceso *zombie.* Los procesos *zombie* pueden existir por un corto período de tiempo, que típicamente significa un error en el programa padre. La presencia de muchos procesos zombie puede indicar problemás en el sistema, y puede acarrear una alta carga del sistema, lentitud y respuestas lentas. Con la opción '-**Z**' del comando **ps** (comentado en breves) podremos localizar procesos zombie en nuestro sistema.

Aclaración: Un proceso *zombie* no es lo mismo que un proceso huérfano. Los procesos huérfanos no se convierten en procesos *zombies*, sino que son adoptados por *init* (PID 1)

Una de las herramientas más importantes para la administración de procesos es **ps**, que muestra el estado de los procesos, por lo que es útil para la monitorización del sistema. Aparentemente *ps* es un programa de fácil uso, su sintaxis es:

```
$ ps -opciones
```

La complejidad llega cuando descubrimos que las opciones de *ps* son en realidad un grupo de otras opciones, es decir, *ps* admite opciones de tipo **Unix98**, las cuales van precedidas de (**–**), son de un único carácter, pero se pueden agrupar (*ps -e-f = ps -ef*), admite opciones de **BSD**, opciones de un único carácter que también se pueden agrupar, no van precedidas del caracter (-) y por último las opciones **GNU** largas, precedidas por (**—**) y son multicaracter. Podemos cambiar el comportamiento de *ps* a través de la

variable **PS_PERSONALITY** con los valores *posix, old, linux, bsd, sun, digital*, entre otros.

ps genera una salida separada en columnas, cuyo encabezado define el significado de cada una de ellas, como por ejemplo:

- El nombre de usuario
- ID del proceso (**PID**)
- ID del proceso principal (**PPID**)
- El teletipo **TTY** (código que se utiliza para identificar una terminal o acceso remoto, los programás GUI no tienen ese número)
- El tiempo de CPU (**TIME**=cantidad total de tiempo consumido de CPU y **%CPU**=porcentaje de tiempo de CPU que utiliza el proceso cuando se ejecuta *ps*)
- Prioridad de la CPU (**NI** muestra el código de prioridad, *números* **+** para prioridad reducida y *números* **–** para prioridad aumentada)
- Uso de memoria (**RSS**=memoria utilizada por un programa y sus datos, **%MEM**=porcentaje de memoria que utiliza el programa y **SHARE**=memoria compartida con otros procesos (como las bibliotecas compartidas)
- La columna Comando muestra el comando ejecutado

Nota: Podemos usar el comando **pstree** para mostrar una salida de *ps* en árbol

En ocasiones usamos *grep* junto a *ps* para imprimir solo aquellos procesos que coinciden con un patrón. Existe una herramienta con la que poder realizar la misma acción y es **pgrep**. Programa que se escribió originalmente para *Solaris 7*. *P*osteriormente fue implementado para *Linux* y *OpenBSD*. La función principal de *pgrep* es devolver el número de ID de los procesos cuyos nombres coinciden con lo del patrón pasado por una expresión regular.

Nota: Podemos decir que **$ps ax | grep mate | grep -v grep | awk ‘{print $1}’** es igual que **$pgrep mate**, de hecho el segundo es más preciso que el primero.

Otro comando útil relacionado con los procesos es **fuser**. El comando *fuser* nos permite identificar que procesos están usando un cierto archivo, *socket* o directorio, y si queremos, podremos matarlos. Usar *fuser* por ejemplo para ver que proceso está usando nuestro directorio actual:

```
$ fuser .
```

Veremos una salida en la que se mostrará un número de proceso acompañado del caracter ‘**c**‘ (es posible que en vez de un solo proceso, veamos una serie de procesos, significará entonces que varios procesos están haciendo uso de nuestro directorio). El caracter ‘*c*‘ tras el número de proceso representa que es el directorio actual de trabajo (*Current Directory*) por lo que dependiendo del archivo que usemos podremos ver una ‘*c*‘ o ‘**e**‘ si es un ejecutable en activo, ‘**f**‘ para un fichero abierto, ‘**F**‘ si el fichero está

abierto para escritura, ‘**r**‘ para el directorio raíz o incluso ‘**m**‘ si se trata de una biblioteca compartida o archivo mapeado.

Podemos ejecutar *fuser* contra un servicio simplemente pasando el nombre del servicio como parámetro y opcionalmente **-v** para obetener más información:

```
$ fuser -v httpd
```

Si se trata de un *socket* usamos la opción **-n** (con -*v* si queremos) el nombre del protocolo (por ejemplo *tcp*) y el puerto. Podemos hacer que *fuser* sea interactivo con la opción **-i** útil si estamos matando procesos con **-k**

Para hacer un seguimiento de los procesos activos en el sistemás, comentamos antes una buena forma de hacerlo con el comando ps, pero también contamos con el utilitario **top**. *top* es una herramienta de línea de comandos, aunque también posee su equivalente gráfico como **kpm** o **gnome-system-monitor**, útil sobre todo para saber cuanto tiempo de CPU consumen unos procesos con respecto a otros. Por defecto *top* ordena su salida en función de aquellos programás que consumen más tiempo de CPU. Uno de los datos importantes que proporciona top es la carga media (*load average* : x.xx y.yy z.zz, donde x.xx es la carga media actual, y.yy la carga media desde hace 5 minutos y z.zz la carga media desde hace 15 minutos) que indica cuantos programás están compitiendo por tiempo de CPU (también se puede averiguar la carga media con el comando **uptime**).

Un sistema que tiene un programa que hace uso intensivo de CPU tendrá una carga media de 1 (suponiendo que tenga un solo núcleo o CPU, ya que de tener 4 la medía podría ser 4.00).

Un programa puede crear una carga media de 1.0, sin embargo programás con múltiples procesos pueden generar cargas más elevadas.

El comando *top*, puede recibir opciones como parámetros e incluso nos permite interaccionar con él, una vez ejecutado.

Al igual que comprobamos la carga media del sistema, podemos comprobar la memoria RAM disponible o utilizada por el conjunto de procesos que corren en el sistema. Un comando útil para tal fin es **free**. Las opciones de *free* son pocas y sencillas, pero las veremos en el apartado de comandos para el manejo de procesos.

Si queremos imprimir una información mínima sobre los procesos asociados con la sesión actual, podemos usar el comando **jobs** (su demonio *jobd*). *jobs* muestra el ID de las tareas, útil si queremos conocer cuantos programás continúan abierto antes de cerrar sesión, etc…

Al existir procesos en primer y segundo plano, *jobs* puede ser una buena alternativa para manejar esto.

Ejemplo:

Imaginemos que estamos trabajando con una terminal y ejecutamos un solo comando el cual nos va a bloquear el uso de la terminal debido a la ejecución de un programa, si pulsamos *Control-Z* suspenderemos ese programa retomando el control de la terminal, ahora para volver a tener el control del programa escribimos **fg** (*foreground*, primer plano) o bien si queremos que el programa siga ejecutándose pero en segundo planos escribiremos **bg** (*background*, segundo plano).

Ahora es cuando hacemos uso de *jobs*, imaginemos en que vez de 1 programás ya tenemos 2 (el anterior que ahora corre en segundo plano y uno nuevo que hemos mandado a segundo plano también), supongamos que deseamos enviar a primer plano el primer programa, bastaría con escribir *fg*, pero al tener 2 programás necesitamos conocer su ID, para ello **$jobs** nos mostrará el ID y el nombre del programa (empezando desde 1), suponiendo que es 1 el ID, escribimos **fg 1** (también $fg %1), ahora con el control de ese programa en primer plano, escribimos *Control-C* si queremos finalizarlo.

El uso del metacaracter '***%***' es importante, ya que permite enviar/reanudar un trabajo concreto. Suponiendo que tenemos 2 programás con IDs 1 y 2 en segundo plano y detenidos, podríamos reanudar por ejemplo uno de ellos con:

```
$bg %2
```

Otra forma de iniciar un programa en segundo plano es mediante el uso de **&**:

```
$nedit archivo.txt &
```

Nota: inicia el editor en segundo plano, permitiéndote el uso de la terminal.

Si quisiéramos esperar a que todos los trabajos en *background* terminaran para retomar el control de la terminal, podríamos usar el comando **wait** si ningún parámetro, esperar a que termine el trabajo con ID 2 (pasandole el parámetro %2) o bien, en vez de pasar un ID de job, podemos pasar directamente el PID de un proceso.

Si lo que queremos es ejecutar un programa desde una terminal (ya sea en primer o segundo plano) pero que siga ejecutándose una vez cerrada esta, podemos usar los comandos **nohup** o **screen**.

El comando **nohup** lo que hace es ignorar la señal *SIGHUP*, es decir la producida al cerrar la terminal, por lo que el comando/programa se mantiene en ejecución una vez cerrada esta. Su uso es básico, se le pasa como argumento el comando/programa que queremos ejecutar.

Screen trabaja de forma diferente a *nohup* pero igualmente nos permite ejecutar programás aun habiendo "cerrado" la terminal. Esto no es del todo cierto, pues en

realidad la terminal desde la que lanzamos el comando no se cierra o mejor dicho su proceso no se mata. Supongamos que estamos trabajando en un servidor *Linux*, desde un equipo remoto (ya sea a través de putty en un equipo Windows o una terminal en un cliente Linux) o desde una terminal del mismo servidor, y queremos realizar una copia de seguridad de una parte pesada del servidor (lo que llevará horas…) el problema es que no queda más de 30 minutos para terminar nuestro día laboral, es el momento de crear una terminal virtual con *screen*, por lo que en la terminal que tenemos abierta tecleamos *screen* y esto producirá un leve parpadeo, emitirá un mensaje o simplemente no hará nada, pero tenemos que saber que desde ese momento tenemos abierta una terminal virtual. Ahora lanzamos el respaldo contra el servidor y llega el momento de irse a casa, pues nada, cerramos nuestra terminal (no con exit, si no simplemente cerramos las ventanas), apagamos nuestra máquina cliente y nos marchamos a casa. A la mañana siguiente nos conectamos a una terminal del servidor, bien de forma remota o local y tecleamos $*screen -ls*, lo que nos mostrará cuantas terminales virtuales tenemos abiertas y sus id, para conectarnos a una en concreto escribimos $*screen -r la_ID* y accederemos a la terminal virtual que abrimos el día anterior para lanzar la copia de seguridad. Si sabemos que solamente tenemos una terminal abierta bastará con escribir *$screen -r* e igualmente tomaremos el control de la terminal en la que ejecutamos nuestro respaldo, ahora tendremos el control sobre esa terminal, de modo que podemos comprobar como ha ido la copia de seguridad. Una vez finalizado nuestro trabajo con la terminal virtual escribiremos *exit* y cerraremos así la terminal de forma definitiva.

Prioridad de los procesos

- Para tener un control sobre la prioridad de los procesos en cuanto a la demanda de tiempo en CPU usamos los comandos **nice** y **renice**.
- El comando *nice* permite ejecutar un programa con una prioridad especificada, mientras que *renice* permite la modificación de esa prioridad mientras el programa se encuentra en ejecución.
- La prioridad de los programás va desde el **-20 al 19**, siendo los números negativos los de mayor prioridad o prioridad incrementada (solo *root* puede usar la prioridad incrementada) y los positivos para una prioridad baja.
- La prioridad de ejecución de un programa por defecto es 0 y la prioridad por defecto utilizando *nice* (en caso de no pasar ningún valor de regulación) es 10, por el contrario si *renice* no recibe ningún argumento de prioridad asume que el número es un PID.

renice además puede recibir como parámetros un número de PIDs (**-p**), varios GIDs de grupos (**-g**) o varios UIDs (**-u**), solo root puede alterar la prioridad para los grupos y usuarios.

Destruir procesos.

A veces, no solo basta con reducir la prioridad de un programa, si no que necesitamos pararlo, por ejemplo por que se haya quedado bloqueado. Para tal fin podremos usar algunas herramientas como: **kill, killall** o **pkill**.

Antes de entrar en detalle con las herramientas, vamos a comentar el funcionamiento de "matar un proceso".

Linux utiliza una serie de señales para comunicarse con los procesos (*SIGKILL*, *SIGHUP*, *SIGTERM*…), siendo el *kernel*, el usuario o el propio programa el encargado de enviar algunas de estas señales.

Para finalizar un proceso será *kill* el programa que envíe esa señal, señal que podremos pasar como parámetro, bien por su nombre largo (**-s** señal), por su nombre pero sin "SIG" (–señal, -KILL) o directamente por su número (–num). Si no especificamos una señal, kill envía la señal por defecto *SIGTERM (15)* que permite cerrar el programa de la forma más limpia, esto es, cerrando sus archivos, cerrando conexiones, limpiando su búfer,etc…

Vamos a ver algunas de estas señales:

```
$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 16) SIGSTKFLT 17) SIGCHLD 18) SIGCONT 19) SIGSTOP 20)SIGTSTP 21) SIGTTIN 22) SIGTTOU 23) SIGURG 24) SIGXCPU 25)SIGXFSZ 26) SIGVTALRM 27) SIGPROF 28) SIGWINCH 29) SIGIO 30)SIGPWR 31) SIGSYS 34) SIGRTMIN 35) SIGRTMIN+1 36) SIGRTMIN+2 37)SIGRTMIN+3 38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42)SIGRTMIN+8 43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47)SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52)SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9 56) SIGRTMAX-8 57)SIGRTMAX-7 58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62)SIGRTMAX-2 63) SIGRTMAX-1 64) SIGRTMAX

1) SIGHUP: Cierra los programás interactivos y hace que muchos demonios vuelvan a leer su archivo de configuración. Es la que se genera por ejemplo cuando cerramos la terminal.

2) SIGINT: Interrupción. Se produce cuando el usuario pulsa la combinación de teclas *Control-C*, Puede ser ignorada por un manejador de señales.

3) SIGQUIT: Igual a *SIGINT*, pero se produce al pulsar *Control*-\ y además se crea un archivo .core (volcado).

7) SIGBUS: Se produce cuando se intenta acceder de forma errónea a una zona de memoria o dirección inexistente. Su acción es terminar el proceso que la recibe.

9) SIGKILL: No puede ser ignorada por el manejador de señales, y ni su rutina puede ser modificada. Provoca irremediablemente la terminación del proceso sin realizar las tareas de finalización de rutinas.

13) SIGPIPE: Intento de escritura en una tubería en la que nadie está leyendo. Suele ocurrir cuando un proceso de lectura termina de forma anormal. Su acción es terminar el proceso.

15) SIGTERM: Terminar un proceso de forma "limpia", es decir cerrando conexiones, archivos, limpiado búfer…

18) SIGCONT: Continúa un proceso

19) SIGSTOP: Para un proceso

31) SIGSYS: Se envía cuando se produce un argumento erróneo en una llamada al sistema.

Ahora que conocemos algunas de las señales, vamos a nombrar alguna de las herramientas útiles:

- **kill** : Envía una señal de finalización de proceso
- **pkill**: Es idéntico al comando *pgrep* (busca PIDs de procesos según patrón) y además una vez encontrado los mata. Es posible pasarle señales como argumento, al igual que se hace con *kill*.
- **killall**: Este comando destruye un programa en función de su nombre y no de su PID. Se le puede pasar una señal a través de la opción **-s** o **–signal**. Existe una variante de *killall* en versiones de Unix, que lo que hacen es matar todos los procesos iniciados por un determinado usuario. *$killall alberto*

Nota: Como siempre, para ampliar la información sobre los comandos acuda a la página de comandos LPIC-1

Notas para el exámen

- Comandos para la gestión de paquetes (rpm, yum, dpkg, apt-get, apt-cache, dselect y aptitude)
- Compilación de paquetes: Uso, compilación en diferentes arquitecturas, compatibilidades entre arquitecturas y distribuciones…
- Gestión de librerías: instalar, eliminar, añadir nuevas rutas de librerías…

LPIC-1 Capítulo 3

EN ESTE CAPÍTULO SE ABARCAN LOS SIGUIENTES OBJETIVOS DEL EXAMEN:

- 101.1: Determinar y configurar parámetros de hardware (2)
- 102.1: Diseñar la estructura de discos duros (2)
- 104.1: Crear particiones y sistemás de archivos (2)
- 104.2: Mantener la integridad de los sistemás de archivos (2)
- 104.3: Controlar el montaje y desmontaje de sistemás de archivos (3)

Instalar, configurar y administrar el hardware del equipo

IRQ

Petición de interrupción o interrupción (**IRQ**): Señal enviada a la *CPU* que le indica que suspenda su actividad y se encargue de eventos externos, como por ejemplo el teclado, existencia de datos a procesar por parte de la gráfica, etc…

El diseño del bus ISA (*Industry Standard Architecture*) hace complicado compartir una interrupción entre dos dispositivos, algo que puede llevar a conflictos (por ejemplo en la IRQ5 de la siguiente tabla) este bus está obsoleto desde 2001, pero si tenemos una máquina con este tipo de bus, debemos de asegurarnos de que no comparte *IRQ*.

El bus PCI (*Peripheral Component Interconnect, Interconexión de componentes periféricos*) es un bus de recurso compartido (esperan su turno para comunicarse por el bus) con limitación de latencia (ancho de banda) que hace más fácil compartir las interrupciones, pero solo entre dispositivos PCI. Existen ciertas limitaciones a la hora de compartir *IRQ* entre dispositivos PCI, por ejemplo, quizás la *BIOS* y la placa base permitan compartir *IRQ* pero no el propio dispositivo PCI, que exige su propia *IRQ* para un correcto funcionamiento.

IRQ y sus funciones habituales en sistemás x86:

0 Reloj del sistema (uso interno)
1 Teclado

2 Gestión *IRQ* 8-15 *(El circuito interno de administración IRQ en x86 solo puede ocuparse de 8 IRQ. La IRQ2 está asociada con la gestión de las 16 IRQ, aunque solo debe usarse para gestionar las IRQ 8-15).*
3 Segundo puerto serie RS-232: COM2 en Win. Puede compartirse con un cuarto puerto RS-232
4 Primer puerto serie RS-232: COM1 en Win. Puede compartirse con un tercer puerto RS-232
5 Tarjeta de Sonido o segundo puerto paralelo LPT en Win.
6 Controlador de discos flexibles
7 Primer puerto paralelo: LPT1
8 Reloj de tiempo real (usado por el reloj del sistema)
9 Interrupción abierta
10 Interrupción abierta
11 Interrupción abierta
12 Ratón PS/2
13 Coprocesador matemático (uso interno)
14 Controlador ATA primario para dispositivos ATA: hda y hdb
15 Controlador ATA secundario para dispositivos ATA: hdc y hdd

NOTA: En sistemás x86_64 las *IRQ* se suelen asignar como hemos mencionado arriba, pero a las *IRQ* superiores se les puede asignar hardware adicional

Para comprobar nuestra tabla *IRQ* y ver si existen conflictos o simplemente varios dispositivos comparten *IRQ*, podemos hacer uso del sistema de archivos virtual (*/proc*), el cual hace referencia a datos del *kernel*, hardware, procesos en ejecución, etc. Precisamente el archivo */proc/interrupts* (**$cat /proc/interrupts**) es el encargado de enumerar las *IRQ* que están en uso en Linux, es decir, que no aparezca la *IRQ* 7 (puerto paralelo) no quiere decir que no esté disponible, si no que ningún componente hardware está haciendo uso de ella en ese momento.

Podemos resolver problemás con las *IRQ* desde la *BIOS*, con paquetes como ***isapnptools***, a través de los *jumpers* del dispositivo, etc..

Algunos pasos para configurar las *IRQ* de dispositivos PCI/AGP:

1. Desde la *BIOS* deshabilitar todos los dispositivos que no estemos usando, para permitir asignar a otros su propia *IRQ*.
2. En la *BIOS*, localizar la configuración PnP y asegurarnos de que nuestra *BIOS* está configurada para ejecutar un sistema operativo “non-PnP” y además seleccionamos la opción “Reset ESCD data” (o similar), para forzar que tras un reinicio del sistema la *BIOS* reasigne de nuevo las *IRQ*

Si tras los pasos anteriores seguimos teniendo problemás de suplantación de *IRQ* entre dispositivos, podemos probar con otras dos opciones:

1. Existen *BIOS* que nos permiten asignar nosotros mismos las *IRQ* (opción menos frecuente)
2. Si tenemos slots PCI aun libres: Apagamos el equipo, lo desconectamos de la toma eléctrica y pulsamos varias veces el botón de encendido para descargarlo por completo, y movemos las tarjetas PCI de slot hasta encontrar que todas o al menos las que no permiten compartir *IRQ* han encontrado su propia *IRQ*, esto habrá que comprobarlo con el comando *$cat /proc/interrupts*, es un poco "gipsy" pero funciona.

Nota: Linux cuenta con una suit de tools pci para facilitar su configuración (pci-utils). La herramienta de configuración avanzada de hardware PCI es setpci, pero también podemos ayudarnos con lspci para ver la configuración.

Podemos emplear la utilidad **setpci** para consultar y ajustar directamente las configuraciones de los dispositivos PCI a bajo nivel, como la latencia, algo muy útil (aunque suelen venir configurado para un funcionamiento optimo) cuando añadimos o cambiamos de hardware. Una latencia de 0 indica que el dispositivo se detendrá en el momento en que otra tarjeta PCI necesite su atención, una latencia de 248 (máxima) demorará su parada.

Con el comando **lspci** podemos comprobar la configuración de los dispositivos PCI. *lspci* muestra información sobre todos los buses del sistema y los dispositivos conectados a estos.

Otros comandos como **lshw**, **dmidecode**, **lsusb** y **biosdecode**, nos permitirán tener un mejor conocimiento sobre el hardware que el sistema tiene instalado.

Desde el BIOS podemos desactivar/activar dispositivos hardware de nuestro sistema, como la tarjeta de sonido, usar la gráfica nativa de la placa o bien activar alguna que hayamos instalado, etc..

Nota: Aunque existan controladores para la administración de dispositivos, la mayoría de estos están diseñado para cargar, configurar y actualizar nuevas características de estos, pero lo normal a la hora de desactivar componentes hardware es hacerlo a través del firmware

Direcciones de E/S

Las direcciones de entrada/salida (conocidas como puertos **E/S**), son ubicaciones exclusivas en memoria, reservadas para las comunicaciones entre la CPU y dispositivos hardware específicos. Como pasa con las *IRQ*, lo normal es no compartir una misma dirección para 2 o más dispositivos.

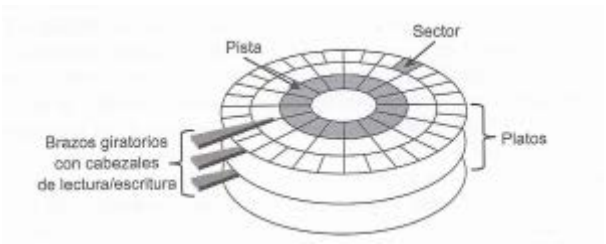
Podemos explorar que dispositivos E/S están en uso, desplegando el contenido del archivo virtual **/proc/ioports**.

Direcciones DMA

Un método alternativo para la comunicación de los puertos E/S es la **DMA** (*Direct Memory Addressing, Direccionamiento directo de memoria*), que permite a los dispositivos E/S enviar directamente la información sin que medie en esto la CPU, mejorando el rendimiento global del sistema. Este método utiliza canales de comunicación, que al igual que pasa con las *IRQ* y los puertos E/S, conviene no compartir un canal. Para conocer los canales que están en uso podemos desplegar el contenido del archivo virtual **/proc/dma**.

Geometría de un disco duro

Geometría **CHS** (*Cilindros/Cabezales/Sectores*): Estructura tradicional de un disco duro que consiste en un número fijo de cabezales de lectura/escritura, encargados de delimitar las pistas al girar sobre los discos, las cuales en su conjunto forman los cilindros (organizados verticalmente). Cada pista esta formada por cientos de sectores. Un sector puede ser identificado por tres números: un número de cilindro, un número de cabezal y un número de sector. Para ello la *BIOS* debe conocer el número de cabezales, cilindros y sectores que tiene el disco. El tamaño de un sector suele ser de **512 bytes**



La geometría **CHS** tiene sus inconvenientes, como por ejemplo determinar el número de sectores por pistas en los discos más recientes, ya que estos tienen más sectores en las pistas externas que en las internas, por consiguiente la geometría que el disco duro presenta a la *BIOS* es realmente falsa, alcanzando además un máximo de 540MB, sin el uso de parches para la traducción de geometría *CHS*, con el que alcanzaríamos un total de 8GB.
Para dar solución a esto, actualmente se utiliza el modo **LBA** (*Logical Block Addressing*) por el cual se asigna un número exclusivo a cada sector del disco, permitiendo al firmware del disco leer del cabezal y cilindro correctos cuando se le pasa un número determinado de sector.

Dispositivos de conexión en frio (coldplug) y en caliente (hotplug)

Los dispositivos de conexión en frio (RS-232, LPT, ISA…) están diseñados para conectarse y desconectarse físicamente sólo cuando el ordenador está desconectado. Intentar conectar y desconectar estos dispositivos en plena ejecución del ordenador puede dañar el dispositivo o el ordenador. Precisamente por esto, se desarrolló una variante de PCI de conexión en caliente para utilizarla en equipos donde la interrupción o apagado no es una opción (como por ejemplos servidores). También existen dispositivos SATA para la conexión en caliente, y dispositivos como los USB, Ethernet e IEEE-1394 que son dispositivos puramente de desconexión en caliente, basados en un software especializado de Linux que permite detectar los cambios del sistema según se conecten y desconecten.

Algunas utilidades para administrar los dispositivos en caliente podrían ser:

- **Sysfs** : El sistema de archivo virtual **/sys**, exporta información sobre los dispositivos, de forma que los programás del usuario puedan acceder a ellos. Antiguamente */dev* era el encargado de proporcionar los medios principales de interfaz entre los programás del usuario y el hardware.
- Demonio **HAL** (*hald*) : Suministra de forma constante información sobre el hardware disponible a los programás de usuario.
- **D-Bus** : Demonio que permite a los procesos comunicarse entre si en lo referente a las notificaciones de eventos, por ejemplo la disponibilidad de un nuevo disco externo USB
- **udev** : Es el gestor de dispositivos que usa el *kernel* de Linux en su versión 2.6. Su función es controlar los ficheros de dispositivo en /dev, lo que significa que maneja el directorio /dev y todas las acciones del espacio de usuario al agregar o quitar dispositivos, incluyendo la carga de firmware. Se creó para corregir ciertas limitaciones de *devfs* (su predecesor) como el de poder acceder a un dispositivo con un nombre siempre fijo. *devfs* ya mejoró en ciertos aspectos la gestión de dispositivos conectados al sistema. Antiguamente el directorio */dev* contenía un nodo para cada dispositivo conocido, estuviese o no conectado al sistema, formando un conjunto de archivos estáticos. Esto tiene sus limitaciones e inconvenientes como la dificultad de administrar el directorio */dev* debido a la gran cantidad de archivos, el uso limitado de números asociados a dispositivos, etc…
  udev mantiene en /dev sólo las entradas correspondientes a los dispositivos que hay conectados al sistema, solucionando así el problema de cantidad de archivos estáticos en el directorio /dev. *udev* no usa el número *mayor* y *menor* para reconocer a cada dispositivo además avisa mediante mensajes *D-BUS* para que cualquier programa del espacio de usuario pueda enterarse cuando un dispositivo se conecta o desconecta (esto es útil para HAL). También permite a los programás consultar la lista de dispositivos conectados y la forma de acceder a cada uno.
  El proceso ***udevd*** ocupa poca memoria y no necesita ejecutarse siempre. Esto también favorece a los sistemás embebidos y equipos poco potentes.

Módulos del kernel

El hardware en Linux es administrado por los controladores (módulos) del *kernel*, que son archivos independientes que suelen estar bajo el directorio */lib/modules*. Estos módulos suelen cargarse al iniciar el sistema, aunque es posible que necesite cargar otros módulos una vez el sistema haya iniciado. Existe diferentes herramientas que nos ayudarán a gestionar estos módulos.

Por ejemplo vamos a suponer que queremos ver los módulos que se encuentran actualmente cargados en nuestro sistema, para esto podremos usar **lsmod**, que nos desplegará una lista con los nombres de todos los módulos cargados, así como la memoria que consume, el número de procesos o otros módulos que están haciendo uso de él y que proceso o modulo es este.

Nota: El comando lsmod solo muestra los módulos cargados y no los controladores que se compilan directamente en el kernel. Esto quiere decir que para utilizar un hardware concreto quizás debamos cargar manualmente nosotros el módulo (si es que no aparece en el listado de lsmod) o bien se cargará al inicio y aparecerá en el listado de lsmod, pero a lo mejor en otro sistema que usemos ese mismo hardware no aparece el módulo y si funcione el hardware, eso es debido a que el controlador venga directamente compilado en el kernel.

Si encontramos un módulo que indica estar siendo utilizado por otro, no podremos eliminarlo hasta que previamente se elimine el que está haciendo uso de este. Podemos obtener información sobre un determinado módulo con el comando **modinfo**.

Para descargar un módulo podemos usar el comando **rmmod** (solo descarga un módulo que no se encuentra en uso) o **modprobe -r**, que descargará el módulo y todos aquellos de los que depende (a menos que estén siendo utilizado, que entonces no se eliminarán). Por el contrario si quisiéramos cargar un módulo recurrimos al comando **insmod** o **modprobe**. La diferencia es que *insmod* solo carga el módulo en cuestión mientras que **modprobe** carga el módulo y aquellos de los que posiblemente dependa. Si vamos a cargar un módulo con *insmod* y este depende de otro, el módulo no será cargado hasta que antes carguemos del que depende. El comando **nmmod** es el opuesto a *insmod*, es decir solo descarga el módulo especificado y no aquellos de los que depende, se esté haciendo uso de ellos o no.

A *modprobe* basta con pasarle el nombre del módulo a diferencia de *insmod* que necesita conocer la ruta completa y el nombre del módulo.ko, por ello *modprobe* depende de un archivo de configuración **/etc/modprobre.conf** o varios archivos bajo **/etc/modprobre.d/**

Nota: El tener módulos cargados que no se estén usando no tiene más inconveniente que la memoria de la que esta haciendo uso, que suele ser mínima.

Controladores USB de Linux

El **USB** es el sistema de interfaz preferido para el uso de muchos dispositivos externos, es tanto un protocolo como un puerto de hardware para transmitir datos hacia y desde dispositivos. Existen diferentes especificaciones USB. El USB 1.0 y 1.1 admiten un total de 127 dispositivos y velocidades de transferencias de hasta 12Mbps, mientras que USB 2.0 transfiere a un máximo de 480Mbps y USB 3.0 tiene una velocidad teórica de 4,8Gbps, aunque en la práctica se aproxima a 3,2Gbps.

El motivo de que al conectar un disco duro externo por USB aparezca como /dev/sdX (aparentemente bajo conexión SCSI), es porque en las primeras implementaciones de USB para Linux, este necesitaba un controlador independiente para cada uno de ellos, controladores que aun siguen en el kernel y además hay programás que los utilizan. Estos controladores USB hacen de "puente" pareciendole a Linux que se trata de una conexión SCSI. Además Linux ofrece un sistema de archivos USB bajo **/proc/bus/usb**, donde accede el software para controlar el dispositivo.

Como USB se diseñó como hotplug, el kernel de Linux no fue concebido originalmente con este tipo de actividad, por lo que la administración de dispositivos USB se apoya en utilidades externas, como por ejemplo **usbmrg** y **hotplug**.

- **usbmgr** es un programa que se ejecuta en segundo plano y que al detectar cambios, carga o descarga los módulos del kernel necesarios para controlar los dispositivos. *usbmgr* a perdido relevancia con el tiempo debido al cambio de controladores USB específicos a sistemás de archivos de dispositivos USB (/proc/bus/usb).
- **hotplug** es la nueva herramienta con la que podremos tener un control sobre los dispositivos USB conectados al sistema. *hotplug* utiliza archivos almacenados en **/etc/hotplug** para controlar la configuración de dispositivos USB. El archivo **/etc/hotplug/usb.usermap** contiene una base de datos ID de dispositivos USB y punteros a los scripts de **/etc/hotplug/usb** que se ejecutan cuando se conectan o desconectan dispositivos USB. Con estos scripts podrían cambiarse los permisos de los archivos de los dispositivos USB para poder ser usados por otros usuarios, ejecutar comandos para detectar nuevos dispositivos USB, etc…

Nota: Aunque los USB y los IEEE-1394 sean dispositivos de conexión en caliente, es recomendable desmontar antes el sistema de archivos, para asegurar una extracción correcta del dispositivo.

Configurar discos duros

Son habituales tres tipos de interfaces de disco duro: **PATA** (*ATA en Paralelo, Advance Technology Attachment*), **SATA** (*ATA en Serie*) y **SCSI** (*Small Computer System Interface*).
SCSI suele ser un bus en paralelo aunque su variante **SAS** (*Serial Atachment SCSI*) es un bus en serie.

Los discos *PATA* suelen tener nomenclatura **hda** (maestro del 1º controlador) , **hdb** (esclavo del primer controlador), **hdc** (maestro del segundo controlador), **hdd** (esclavo del segundo controlador) y asi…
Según la partición tendrá un numero u otro; hda1, hda2, hda3, hdb1, hdb2, hdc1, etc…
Los discos *PATA* solían usarse en PC del 2005 o anterior y se configuraban (maestro/esclavo) mediante un *Jumper* o bien indicando en la *BIOS* que su configuración sea mediante cable. El cable para discos *PATA* tiene 3 conectores, el de la placa base, el del disco esclavo (el conector intermedio) y el del maestro (último conector). Para un correcto funcionamiento, lo mejor es conectar un disco *PATA* en cada cable, ya que el controlador *PATA* posee un rendimiento limitado, que puede verse superado por la suma de los dos discos.

Los discos *SATA* a pesar de ser conexión en serie, envían más bit/s que sus predecesores *PATA*. La configuración de *SATA* es prácticamente nula, los cables son más finos y la placaba base contiene un conector para cada disco (normalmente 4), la posición es independiente. La mayoría de los controladores Linux para discos *SATA*, tratan a estos como discos *SCSI (/dev/sdX)*, aunque algunos controladores antiguos, los nombran *hdX* como si de discos *PATA* se tratase.

SCSI es tradicionalmente un bus *PATA* aunque bastante más costoso. Su variante *SAS* es un bus en serie como *SATA*. *SCSI* en teoría admite de 8 a 16 dispositivos, aunque en la práctica esto depende de diversos factores, como por ejemplo la longitud del cable. Cada dispositivo *SCSI* tiene su propio ID, asignado normalmente a través del *jumper*. Para arrancar un sistema desde un disco *SCSI* se precisa de un dispositivo costoso, que tenga su propio firmware compatible con el arranque, de lo contrario, su placa reconocerá los discos y podrá hacer uso de almacenamiento, pero no de arranque.
Los ID *SCSI* no se utilizan para identificar el archivo de dispositivo, que siguen la nomenclatura /dev/sd[abc..] para los discos, /dev/st[0123..] y /dev/nst[0123…] para las cintas y /dev/scd[0123…] para dispositivos *CDROM* y *DVDROM SCSI*, si no que identifican a los propios discos, por lo que es recomendable usar IDs bajos para los primeros discos e ir subiendo consecutivamente para nuevas unidades, aunque esto depende de en parte del diseño del adaptador de host *SCSI*, como **Wide SCSI** que enumera del ID 14 al 8. Los discos *SCSI* no son de conexión en caliente y a menos que no sea un disco extraible, como un disco ZIP, es primordial desmontarlo antes, para evitar posibles daños en el sistema de archivo.

Particionar un disco

Definición breve de partición: Dividir un disco físico, en varios discos "*lógicos*" en los que poder almacenar diferentes sistemás operativos, o usar como "cajones" para almacenar nuestra información digital.

Particionar un disco nos ofrece diferentes ventajas, como por ejemplo: el soporte multi-OS, el uso de diferentes sistemás de archivos, una mejor forma de administrar el espacio

en disco, protección frente a errores de disco (al tener diferentes particiones podemos mover los datos de una a otra e incluso instalar un nuevo sistema en otra partición y gestionar los datos exitente en otra partición a través de este nuevo sistema), seguridad (por ejemplo empleando diferentes opciones de montaje para cada partición) y por supuesto copias de seguridad (aisladas del sistema de archivo de uso ordinario), o bien para una mejor gestión de las particiones de menor tamaño a la hora de crear una copia de la misma.

Cuando creamos un esquema de particiones, este, se guarda en una parte del disco duro. Existen diferentes sistemás para guardar este esquema aunque hay dos que destacan por su popularidad, como es **MBR** y **GPT**, sistemás de los que se hablará más detalladamente en el tema 5.
Existe la alternativa a estos sistemás de particionado, y es **LVM** (*Logical Volumen Management*), el cual nos permitirá, crear grupos de volúmenes, con varios sistemás de archivos (volúmenes lógicos) que pueden ocupar incluso varios discos físico, simulando hasta cierto punto un *RAID 0*. Una de las grandes ventajas de *LVM* es el crear/destruir/redimensionar particiones de forma no muy complicada, por ejemplo, si tenemos un volumen lógico creado para /home, y nos estamos quedando sin espacio, pero disponemos de otro volumen, por ejemplo /usr y tenemos suficiente espacio libre, podemos reducir este, usando el espacio para asignárselo a /home o incluso podríamos añadir un nuevo disco físico y dar parte o la totalidad de este disco al volumen lógico /home.

Los bloques que constituyen un *LVM* son: Volumen Lógico (**LV**) formado por "extensiones lógicas" (**LE**), uno o varios *LV* se encuentran en uno o varios Volumenes Físicos (**PV**) formados por Extensiones Físicas (**PE**) y uno o un grupo de estos forman los grupo de volúmenes (**VG**)

- **Volumen Lógico** (Logical Volume, **LV**): Podemos pensar que estos son particiones normales, las cuales pueden contener sistemás de archivos diferentes montados en */home*, */opt*, */usr*, etc. Están formados por *Extensiones Lógicas* (**LE**). Todos Volúmenes Lógicos tendrán el mismo tamaño para sus Extensiones Lógicas.
- **Volúmenes físicos** (*Physical Volume*, **PV**): Son particiones en discos físicos o bien directamente discos físicos completos o archivos *loopback* donde almacenamos uno o varios volúmenes lógicos. El *PV* posee una cabecera especial y está dividido en Extensiones Físicas (Physical Extents,**PE**) que tienen el mismo tamaño que las Extensiones Físicas del Grupo de Volúmenes. Mientras no agreguemos un PV a un *VG*, este no podrá ser utilizado.
- **Grupo de Volúmenes** (*Volume Group*, **VG**): Grupos de volúmenes físicos. Podemos pensar en que un Grupo de volúmenes puede hacer referencia al conjunto de discos físicos o particiones de estos, es decir podemos tener un Grupo de Volumenes con un espacio de 500GB que en realidad es la suma de 2 discos de 200GB cada uno más 2 particiones de 500MB cada una de las cuales pertenecen a discos diferentes (sda + sdb + sdc1 + sdd1, por poner un ejemplo…)

<u>Ventajas del uso de LVM:</u>

- Usar varios discos físicos como uno solo gracias a la agrupación de estos, mediante el VG (Volume Group)
- Tener una partición (Lógical Volume) que ocupe el tamaño de varios discos físicos
- Facilidad para añadir, eliminar y redimensionar particiones, ya que estás no dependen de su posición como es el caso de las particiones normales.

Desventajas del uso de LVM:

- Su configuración es algo más complicada que la administración de particiones normales.

Técnicas de particionado de un disco

Antes de comenzar vamos a ver una estructura básica de un disco particionado para Linux.

Punto de montaje	Tamaño típico	Uso
/boot	100-500 MB	Al crearlo como una partición independiente podremos evadir las limitaciones de BIOS y cargadores de arranque antiguos que por lo general no pueden arrancar un kernel que se encuentre por encima de los valores comprendidos entre 504MB y 2TB. Esta partición contiene los archivos básicos de arranque
/home	200 MB – 3 TiB	Partición para almacenar los archivos de usuario, como sus fotos, documentos, música y películas. Una partición independiente para el home de los usuarios nos da bastante flexibilidad, como por ejemplo tener varios sistemas Linux en el equipo (incluso otros no Linux) y poder compartir esta partición, de manera que lo que está disponible en Linux, lo estará para otro Linux o incluso Windows.
/mnt		Mas que una partición para contener archivos estáticos, sirve para crear directorios que sirvan como punto de montajes para otras unidades, dispositivos USB, CDROMs, etc.
/media		Igual que /mnt, pero suele ser el destino preferido para montajes multimedia
/opt	100 MB - 20GB	Ideal para almacenar programas de terceros, como los comerciales, algo así como "C:/Archivos de programa" de Windows.
/tmp	100 MB – 20 GB	El espacio para esta partición es dependiente del uso que vayamos a hacer de este, por ejemplo para el uso de base de datos, quizás necesitemos darle mas, que para un uso normal de usuario de escritorio
/usr	200MB - 20GB	Contiene la mayoria de los programas de Linux y sus archivos de datos. En 2012 hubo un cambio que dificulta la creación de esta partición de forma independiente
/var	500 MB – 25 GB	El tamaño de esta partición variar bastante ya que si tenemos un servidor weo de correo que haga uso de /var para contener sus archivos, colas de correos, etc.... puede que necesitemos un gran espacio para almacenar todo esto
swap	RAM*2	Es una partición de intercambio, que actua como una ampliación de ram, solo que es mas lenta, al estar situada en el propio disco. Tiene un formato especial y es una partición independiente al resto. El tamaño que se le suele asignar es el doble de lo que tenemos instalado en RAM.

Nota: Los directorios **/etc**, **/bin**, **/sbin**, **/lib**, **/dev** e incluso **/usr** no deben de colocarse en particiones independientes, puesto que contienen archivos de configuración del sistema y programás ejecutables que posibilitan el inicio del sistema como por ejemplo el fichero **/etc/fstab**, el cual contiene los dispositivos a ser montados. El directorio **/dev** puede residir en un sistema de archivo independiente pero no en una partición independiente ya que la utilidad *udev* controla el sistema de archivos */dev* en las versiones más recientes de Linux.

El particionamiento de un disco conlleva dos tareas: crear las particiones y prepararlas para su uso (darles formato). En Linux suelen utilizarse herramientas diferente para cada operación, aunque también las hay que realizan ambas cosas.

Herramientas particionado **MBR**:

- **fdisk** (fixed disk): Es la herramienta tradicional en modo texto utilizada para el particionado *MBR* del disco. Para utilizar *fdisk* solo hay que escribir el comando seguido del disco a particionar (**#fdisk /dev/hdX** o sdX) y se desplegará un menú donde se nos presentan una serie de opciones. Podemos ver las particiones de un disco directamente desde línea de comandos, sin entrar al programa con **#fdisk -l**. *fdisk* no admite tabla de particiones *GPT*.

  Nota: La herramienta FDISK de DOS es totalmente independiente de fdisk de Línux, quizás sirvan para el mismo propósito pero su uso y funcionamiento, poco tienen que ver.

- **cfdisk**: Editor de particionamiento con una interfaz diferente a *fdisk* y que además, a diferencia de esta, *cfdisk* nos permite redimensionar particiones extendidas cuando existe espacio libre tras ellas. Es fácil moverse por la interfaz mediante los cursores y además nos ofrece varias opciones obvias como *new* (para nuevas particiones), *type* (para escoger el tipo de sistema de archivo), *bootable* (si queremos hacer arrancable una partición), entre otras.
- **sfdisk**: Una de las principales diferencias de *sfdisk* con respecto a *fdisk* es que no es interactiva, por lo que nos permite automatizar mediante scripts el particionado de un disco o crear un clon de nuestro disco como medida de seguridad. *sfdisk* no alinea las particiones y tampoco está diseñado para particiones de gran tamaño. Los principales uso de *sfdisk* son: mostrar el tamaño de las particiones, ver las particiones de un disco, checkear las particiones de un disco y reparticionar un disco.

Herramientas de particionado **GPT**:

- **parted**: Se inicia de un modo similar a *fdisk*, escribiendo su nombre seguido del dispositivo que se desea modificar, **#parted /dev/sdX**. Podemos desplegar una lista de comandos usando el caracter **?**. Algunos de los comandos de *parted* son: **print** que muestra la tabla de particiones actual, **mkpart** para crear una partición, **rm** elimina una partición y algunos más que veremos en la sección de comandos. *Parted* soporta particiones *MBR*, *GPT*, *Acceso Raw* (útil para *RAID/LVM)* y mapa de particiones

Macintosh, entre otros.
Uso de *parted*:

– **Detecta** sistemás de ficheros: ext2, ext3, FAT16, FAT32, linux-swap, hfs/hfs+, jfs, ntfs, reiserfs, ufs y xfs.

– **Crea** particiones ext2, FAT16, FAT32, linux-swap y reiserfs (necesitaremos tener instalada la librería *libreiserfs*).

– **Aumentar** particiones ext2/ext3

– **Reducir** particiones FAT16 (debido al tamaño del cluster no podremos aumentarlas), FAT32, linux-swap, hfs/hfs+, reiserfs (el inicio de la partición debe de permanecer fijo y además deberemos de tener la libreria libreiserfs).

– **Copiar** particiones ext2/ext3 (la partición de destino tiene que ser igual o mayor que la de origen), FAT16/FAT32 y reiserfs.

– **Comprobar** particiones ext2/ext3, FAT16/FAT32, linux-swap y reiserfs

- **GNOME Partition Editor** o **gparted**: Digamos que es la herramienta gráfica por excelencia para escritorios *GNOME*, formada principalmente por la librería *GNU parted*. Según que sistema de archivo manipule, *gparted* nos permite una infinidad de cosas tanto en particiones *MBR*, *GPT*, *APM* y *BSD* (entre otros), como por ejemplo, crear, eliminar, mover, redimensionar particiones, seleccionar el tipo de sistema de archivos, etc.. Habrá opciones que solo podamos usar si arrancamos desde un disco de rescate o las particiones a tratar no están en uso. Tiene soporte para *CHS* y *LBA*.
- **GPT fdisk** es un set de herramientas de modo texto para el particionado del disco, tanto para *MBR* como para *GTP*. En este set se encuentran las herramientas:

– **gdisk**: Es el “hermano” de *fdisk* pero para particiones *GPT*, tanto para discos rígidos como SSD, donde se encarga por defecto de que sus particiones estén perfectamente alineadas. Su uso es puramente desde línea de comandos con permisos de administrador. Algunos de sus comandos útiles son:

– **m**: Volver al menú principal

– **d**: Para eliminar una partición

– **n**: para crear una partición

– **o**: Para crear una nueva tabla de particiones *GPT*

– **b**: Bakcup de la tabla de particiones

– **t**: para cambiar el tipo de partición

– **cgdisk** (interfaz interactiva) y **sgdisk** (útil en scripts) : que son válidas tanto para Linux como para Windows, FreeBSD y OS X.

Los programás de particionamiento actuales comienzan las particiones en límites de 1MB (2048 sectores) debido a que muchos discos modernos requieren que las particiones se alineen en limites de ocho sectores o más para obtener un rendimiento óptimo.

A la hora de crear una partición de arranque *GPT*, existen dos opciones para dar formato al sector de arranque:

1. El ordenador soporta arranque **UEFI** mediante la *BIOS* con tabla de partición *GPT*:
   Es necesario tener una primera partición especial (**ESP**) formateada con *FAT* donde estarán los o el cargador de arranque.
   #gdisk /dev/sda
   n (nueva partición), **1** (primera partición), **+200M** (espacio de la partición), **L** (listar códigos de tipo de particiones) y escribimos **ef00 >** *[Enter]*
2. El ordenador no soporta **UEFI** por lo que arranca con *BIOS* y un disco con particionado *GPT*:
   Se va a crear la partición que se sitúa entre el primer sector (*MBR*) y la primera partición que se le dará un tamaño de 200M para poder convertirla en **EFI** si así quisiéramos, ya que *EFI* necesita mínimo 200M).

   Nota: No aplicamos formato sobre esta partición!!!

Referencias:
http://www.ubuntu-es.org/node/173252#.VKmb0l237tQ
http://es.wikipedia.org/wiki/Formato_de_disco

Tipos y Creación de sistemás de archivos

¿Que es un sistema de archivos?

"Un sistema de archivos son los métodos y estructuras de datos que un sistema operativo utiliza para seguir la pista de los archivos de un disco o partición; es decir, es la manera en la que se organizan los archivos en el disco. El término también es utilizado para referirse a una partición o disco que se está utilizando para almacenamiento, o el tipo del sistema de archivos que utiliza."

Definición dada en el libro "Guía para administradores v 0.8"

La mayoría de los sistema de archivos UNIX tienen una estructura general parecida, aunque los detalles

exactos pueden variar un poco. Los conceptos centrales son **superbloque**, **nodo-i**, **entradas de directorio** y **bloque de datos.**

- El *superbloque* contiene metadatos críticos del sistema de archivos como información acerca del tamaño, cantidad de espacio libre y donde se encuentra los datos. Si el superbloque es dañado, y su información se pierde, no podría determinar que partes del sistema de archivos contiene información y lo más seguro es que no podamos ni montar el sistema de archivos por este motivo, se guardan varias copias del superbloque en determinados bloques del sistema de archivos.
  Por defecto estas copias se almacena cada 8.192 bloques, por lo que la primera copia se encontrara en el bloque 8.193 (se suma 8192 al primer bloque), la segunda en el bloque 16.385, 24577, etc… Podremos usar el comando *# dumpe2fs <dispositivo>* para verificar que es este el tamaño de nuestro '*Block group*' en concreto la línea que nos dará esta información será '*Blocks per group:* ' por lo que podremos buscarla con grep (ya que la salida de *dumpe2fs* será muy extensa). Se nos mostrarán varias líneas indicando donde se encuentran las copias de seguridad. Una vez localizadas (supongamos que una de ella se encuentra en el superbloque 32768, se lo pasaremos al comando **e2fsck** junto con las opciones **-f** y **-b** (man e2fsck o apartado de comandos de este tema)
- Un *nodo-i (inode, Nodo de indice)*: Contienen información sobre un archivo, salvo su nombre. Un inodo guarda la siguiente información:
  - **Identificador** del dispositivo que alberga al sistema de archivos.
  - **Número de inodo** que identifica al archivo dentro del sistema de archivos.
  - **Longitud** del archivo en bytes (tamaño del archivo).
  - Identificador del **usuario propietario** del archivo con derechos diferenciados.
  - Identificador del **grupo propietario** del archivo con derechos diferenciados.
  - **Modo de acceso** al archivo.
  - **Marcas de tiempo**: última modificación (*mtime*), acceso (*atime*) y de alteración del propio inodo (**ctime**).
  - **Número de enlaces** (*hard links*) es decir, entradas de directorio asociados con este inodo.

  Nota: Podemos conocer el número de nodo de indice de un archivo mediante el comando **ls -i** o el comando **stat**

- *Entradas de directorios*: Contienen el nombre de un archivo y el número de nodo-i que representa a este.
- *Bloque de datos*: Conjunto de sectores contiguos que componen la unidad de almacenamiento más pequeña de un disco. El número de bloques es determinado al crearse el sistema de archivos, y depende del tipo elegido, así como de las opciones que use en su creación. Un bloque solo puede contener un archivo, o parte de un archivo, es decir, si el archivo es más pequeño que el bloque, el espacio restante del bloque no será utilizado. El tamaño por defecto de bloque es de 4KB.

Nota: Existen sistemás de archivos como Reiserfs que usan técnicas para encajar los finales de los archivos en los espacios sin utilizar de otros.

Fuentes:

- Libro "Guía para administradores v 0.8"
- http://mural.uv.es/ *(Web oficial de la Universidad de Valencia, España)*
- http://www.makeinstall.es

Tipos de Sistemás de archivos:

1. Nativos de Linux:

- **ext2** : extendido 2 es el tradicional sistema de archivos nativo de Linux, es un sistema fiable y perfecto para particiones **/boot** o discos de menos de 1GB. No posee respaldo de transacciones.
- **ext3** : Es el evolutivo del segundo sistema de archivos nativo para Linux (ext2), incluye respaldo de transacciones por lo que es un sistema de archivos igual de fiable que ext2 y además se recupera mucho más rápidamente de los cortes de alimentación y las caídas del sistema.
- **ext4** : Es la nueva generación de la familia "extendidos", pensado para trabajar con archivos grandes (+2TB), discos muy grandes (+16TB) y extensiones que mejoran el rendimiento.
- **Reiserfs** : Diseñado para trabajar con un gran número de archivos pequeños (menos 32KB) ya que utiliza varios trucos para encajar los finales de los archivos en los espacios sin utilizar de otros. *Reiserfs* hasta su versión 3.x viene incluido en el *kernel*. Tiene varias características que no todos los sistemás de archivos poseen, como *journaling* que previene el riesgo de corrupción del propio sistema de archivo, reparticionamiento tanto con el filesystem montado (online, solo es posible aumentarlo) como desmontado (offline, permite aumentar o disminuir el tamaño) o el *Tail packing* que reduce la fragmentación interna.
- **JFS** (*Journaling File System*): Sistema de archivos de 64bits con respaldo de transacciones creado por IBM para sistemás de altos rendimientos y donado a Linux (su versión OS/2) aunque existen versiones para otros sistemás como AIX, HP-UX… *JFS* utiliza un método para organizar los bloques vacíos estructurándolos en un árbol y además al ser 64bits soporta bastante bien el manejo de grandes ficheros y particiones *LFS* (*Large File Support*). Podríamos resaltar algunas características como el *Journaling* que al igual que r*eiserfs* sigue el principio de *metadata only* que en vez de una comprobación completa, solo se tienen en cuenta las modificaciones en los metadatos provocados por las actividades del sistema, eficiente administración de directorios, mejor utilización de la memoria mediante adjudicación dinámica de inodos, etc.
- **XFS** (*Extents File System*): Al igual que *JFS*, es un sistema técnicamente sofisticado aunque se ganó su reputación por su robustez, velocidad y flexibilidad en *IRIX*, no todas estas características fueron bien encajadas en Linux. *Red Hat Enterprise Linux* incorporó en su versión 7 *XFS* como su sistema de archivos por defecto, destacando su capacidad de manejar una partición de hasta 500TB
- **Btrfs** (*B-tree FS*): Es un sistema de archivos copy-on-write anunciado por *Oracle Corporation* para *GNU/Linux*. El objetivo de *Btrfs* es sustituir a *ext3*, eliminando el mayor número de sus limitaciones como el tamaño máximo de los ficheros o la compatibilidad con nuevas tecnologías. *Btrfs* tiene la intención de centrarse en la

tolerancia a fallos, reparación y fácil administración. Algunas de sus características más importantes son:

– Asignación dinámica de inodos (no se crea un número máximo de archivos al crear el filesystem)

– Snapshot escribibles y snapshot de snapshot

– Comprobación de datos y metadatos (alta seguridad de integridad)

– Comprobación del sistema de archivos sin desmontar (aunque desmontado es aun más rápida su comprobación)

– Desfragmentación sin desmontar

– Modo optimizado para SSD

– Mejora opcional en el formato de discos incompatibles con el objetivo de acelerar el sistema

– Optimizaciones de rendimiento

2. Otros sistemás de archivos:

- **FAT** (*File Allocation Table*): Antiguo y primitivo pero omnipresente ya que los principales OS entienden a *FAT* lo que lo convierte en un buen file system para discos extraibles con los que intercambiar datos. Existen dos variantes de *FAT* que principalmente difieren en el tamaño de la estructura de datos de *FAT* (FAT16 bits y FAT32 bits). Para utilizar los nombres de archivos tradicionales de *FAT* (limitados a 8 caracteres para el nombre y 3 para la extensión lo que se conoce como nombres de archivos 8.3) utiliza el código de tipo de sistema de archivos *msdos*, mientras que para usar el tipo de nombres de archivos extensos al estilo de Windows, utilice VFAT.
- **NTFS** (*New Technology File System*): Es el preferido por los principales OS Windows (desde más antiguos hasta los más nuevos). Desafortunadamente el soporte de *NTFS* para Linux es bastante rudimentario no pudiendo escribir nuevos datos en particiones *NTFS*. Si necesitásemos tener un buen soporte de lectura/escritura podríamos probar con el controlador *NTFS-3G* que reside en el área del usuario en vez de en el del kernel.
- **HFS/HFS+** (*Hierarchical File System*): Es el sistema de archivos para usuarios de Mac OS, aunque si quisiéramos cambiar archivos con ellos, lo mejor sería utilizar *FAT*. Apple ha ampliado *HFS* con *HFS+* conocido como *HFS extendido* (plus) para dar un mejor soporte a discos duros de gran capacidad y a diversas funcionalidades Unix. El *kernel* de Linux desde su versión 2.6.x incorpora un soporte limitado para *HFS+*, pero la compatibilidad de escritura solo funciona si se desactiva las transacciones *HFS+. HFS+* soporta archivos mucho más grandes a parte de usar Unicode en vez de Mac OS Roman para el nombre de los archivos, lo que además permitió nombres de archivos de hasta 255 letras. *HFS+* utiliza tabla de asignación de 32bits y no de 16bits como *HFS*

limitación sería que suponía que ningún disco, independientemente de su tamaño pudiese apoyar más de 65.536 archivos.

- **ISO-9660**: Estándar para *CD-ROM*, presente en varios niveles. El Nivel 1 es similar a FAT, por lo que solo soporta nombres de archivos 8.3. Los niveles 2 y 3 soportan nombres de archivos de hasta 32 caracteres. El código para este sistema de archivo en Linux es iso9660. El soporte de *ISO-9660* de Linux también funciona con extensiones *Rock Ridge* (permite nombres largos, permisos, enlaces simbólicos, etc..) y *Joliet*.
- **UDF** (*Universal Disc Format*): Siguiente generación para discos ópticos. Suele usarse en discos regrabables y DVDs. La compatibilidad lectura/escritura de *UDF* con Linux aun es precaria.

Crear una partición:

Para crear un sistema de archivos usamos el comando **mkfs** o **mkisofs** (para *ISO-9660*). Existen otras herramientas como **mke2fs** y **mkdosfs** que no son más que referencia a *mkfs.ext2* o *mkfs -t ext2* (para *mke2fs*) y referencias a *mkfs.msdos* (*mkfs -t msdos*) y *mkfs.vfat* (*mkfs -t vfat*).

El uso de *mkfs* es simple y suele usarse sin opciones aunque hay dos a las que debemos de prestar atención como son:

1. La opción **-c** hace que mkfs compruebe los bloques defectusos sector a sector, invalidando estos en caso de que se encuentren bloques defectuosos en ellos. Esto producirá una demorá en el tiempo de ejecución de mkfs
2. La opción **-m** nos permite eliminar el espacio libre que deja *mkfs* al sistema de archivos, espacio reservado para poder restaurar el sistema si fuese necesario, permitiendo a root crear nuevos archivos o directorios de rescate en ese espacio. El problema es que para discos grandes, quizás se desperdicie demásiado espacio ya que el % por defecto de *mkfs* es del **5%**. Con *-m nuevo_%* podemos reducir este tamaño, ideal para discos extraibles (que podemos dejarlo incluso a 0) o dicos de mayor tamaño. Tambien tiene sentido establecer a 0 este espacio, en sistemás de archivos de almacenamiento de datos, como por ejemplo /home

Nota: Una vez creado un sistema de archivos podemos revisar los bloques malos o sectores defectuosos con el comando **badblocks**. En realidad la opción -c hace este trabajo durante la creación del sistema de archivos, pero no estará de más conocer *badblocks* y revisar nuestro sistema de archivos una vez creado.

Partición de intercambio:

Una partición de intercambio es una partición/archivo que Linux utiliza como una extensión de la memoria (no por ello es tan rápida ya que se encuentra en disco físico). El código que identifica a esta partición es **0x82**

Nota: Solaris para x86 también utiliza el código de tipo de partición MBR 0x82, pero en solaris hace referencia a una partición estándar. Si tenemos un arranque dual con Linux y Solaris deberemos de prestar especial atención a esto, ya que quizás sea necesario cambiar el código de partición con fdisk, para cambiar temporalmente el código evitando que Linux utilice una partición estándar de Solaris como partición de intercambio o al contrario, sea Solaris quien use la partición de intercambio Linux como una de sus particiones.

Para crear un espacio de intercambio utilizamos el comando **mkswap**, por ejemplo *#mkswap /dev/sda5* y lo activaremos con **swapon**, *#swapon /dev/sda5*.

Nota: Para activar de forma permanente el espacio de intercambio deberemos de crear una entrada en /etc/fstab

Puntos de montaje

Los puntos de montajes son directorios que nos permitirán montar sobre ellos un sistema de archivos. Supongamos que creamos en el directorio /mnt un nuevo directorio llamado USB, ahora si tuviésemos un disco duro externo que contenga por ejemplo 2 carpetas “backups” y “multimedia” o en vez de un disco duro externo, una partición independiente de nuestro disco, podremos montarla en /mnt/USB, quedando una vez montados, de tal forma: /mnt/USB/backups y /mnt/USB/multimedia. Si accedemos a estos directorios y vemos su contenido con ls, veremos que contienen todos los datos del disco duro externo. Podemos crear una entrada para cada directorio en /etc/fstab de manera que se monten automáticamente o no y con restrinciones o opciones por defecto.

Montar sistemás de archivos de forma permanente

Para montar sistemás de archivos de forma permanente hacemos uso del archivo /etc/fstab (filesystem table) que consiste en una serie de líneas con seis campos cada una separados por uno o más espacios o tabulaciones.

Un ejemplo de archivo podría ser:

```
#device          mount
point    filesystem    options                dump    fsck
/dev/sda1              /              ext4          defaults
  1          1
LABEL=/home        /home          reiserfs      defaults
  0          0
/dev/sda5              /mnt/USB         vfat          uid=500,umásk=0,auto
  0          0
//winsrv/shared        /media/shared    cifs          users,credentials=/root/cred
  0          0
```

Descripción de cada campo:

- **device** : Indica el dispositivo de montaje. Suele usarse nombres de dispositivos (/dev/sda1) aunque para evitar errores en caso de cambiar el nombre de dispositivo es válido e incluso mejor usar LABEL o UUID para identificar el dispositivo o partición. También se puede usar algo como *server:/home* (para exportar el sistema de archivos /home del equipo "server") o *//winsrv/shared* (que identifica un recurso compartido a traves de un equipo Windows o Samba llamado winsrv)
- **mount point** : Indica el punto de montaje.
- **filesystem** : Indica el tipo de sistema de archivos.
- **Opciones de montaje** : Son las opciones con las que es posible montar un dispositivo. Estas las vimos junto al comando mount
- **dump** : Si recibe un 1 es que se generará una copia de seguridad. El programa dump está en desuso por lo que aun teniendo un 1 carezca de sentido si no se usa el programa dump.
- **fsck** : Indica la revisión del sistema de archivos durante el arranque o montaje del sistema de archivo. Un 0 indica que no se debe revisar, un 1 (lo normal para /) indica que será el primero en ser revisado y un 2 (para el resto de dispositivos) indica que se revisarán en segunda posición. El sistema de archivos reiserfs no debe ser revisado automáticamente por lo que se le asignará un 0.

Nota: La entrada //winsrv/shared contiene la opción "credentials=/root/cred" es útil cuando se comparten archivos mediante Windows o Samba y el recurso compartido está protegido por usuario y contraseña. Aunque se podría haber usado las opciones username=usuario y password=contraseña, estas no son la mejor opción ya que cualquiera con acceso a fstab vería las credendiales, por lo que es mejor utilizar un archivo (cred) en este caso que se encuentre protegido de cualquier usuario distinto de root y en su interior contenga las credenciales:
username=usuario
password=contraseña

Poner a punto un File System

Hay tres herramientas especialmente importantes para poner a punto los sistemás de archivos ext2, ext3 y ext4 que son: *dumpe2fs*, *tune2fs* y *debugfs*

- Con **dumpe2fs** podemos depurar un sistema de archivos (ext2, ext3 y ext4) y obtener información de como se encuentra configurado, como por ejemplo cuando se realizó el último fsck al dispositivo, que cantidad de nodos indices (inodo) tenemos disponible (lo que se traduce en cantidad de archivos, ya que cada inodo guarda información sobre un archivo concreto), características del sistema de archivos, etc… *dumpe2fs* lo podemos ejecutar mientras el sistema de archivo se encuentra montado por lo que podemos ver su configuración y en caso de no agradarnos en varios aspectos podemos reconfigurarlo. La sintaxis de *dumpe2fs* es muy sencilla, en realidad basta con pasarle

como argumento el nombre del dispositivo (/dev/sdaX), aunque tambien admite varias opciones que estudiaremos en el apartado de comandos de este mismo tema.

- Para reconfigurar las opciones del filesystem podemos usar la utilidad **tune2fs**, la cual nos permite cambiar muchos de los parámetros del sistema de archivos de los que informa *dumpe2fs*. Debemos usar *tune2fs* con opciones, ya que cada una de estas ajustará un parámetro diferente. La sintaxis de *tune2fs* es sencilla pero tiene multitud de opciones que estudiaremos en el apartado de comandos, algunas de ellas son: Ajustar el número máximo de montajes para un dispositivo, añadir respaldo de transacciones, ajustar el tiempo entre comprobaciones, definir bloques reservados…

Nota: No se recomienda utilizar herramientas de bajo nivel como tune2fs mientras el sistema de archivos a configurar se encuentra montado. Si necesitásemos configurar "/" podríamos arrancar desde un LiveCD o disco de emergencias.

- Existen herramientas que nos permiten depurar un sistema de archivos interactivamente como **debugfs** (para abarcar las posibilidades de *dumpe2fs* y *tune2fs*) o **xfs_db** para sistemás de archivos *XFS*. El uso de *xfs_db* está recomendado para expertos en sistemás de archivos *XFS*, no así *debugfs* que si que va incluido en el exámen. *Debugfs* proporciona las capacidades de *dump2fs*, *tune2fs* y muchas de las herramientas habituales de manipulación de archivos Linux (ls, rm, ln…). Para utilizar debugfs basta con escribir *debugfs* seguido del nombre de archivo del dispositivo (/dev/sdaX). Algunos de los comandos para ejecutar una vez estemos interactuando con *debugfs* son: **stats** (mostrar información de los super bloques del sistema), **stat nombre-archivo** (mostrar información de nodos de indice), **help**…, estudiaremos debugfs más a fondo en la sección de comandos.

Otros muchos sistemás de archivos carecen de un equivalente a *dumpe2fs* y *tune2fs* pero no es el caso de *XFS* para el que existe la herramienta **xfs_info** que comparte ciertas similitudes con *dumpe2fs* y **xfs_admin** que es el equivalente a *tune2fs*.

Para utilizar *xfs_info* basta con pasarle el nombre del dispositivo o punto de montaje como argumento y que el sistema de archivos se encuentre montado. La información devuelta por *xfs_info* es bastante técnica, relacionada en gran parte con el tamaño de los bloques, tamaño de sectores, uuid del dispositivo, etc…
Otra herramienta de *XFS* es **xfs_metadump** la cual copia los metadados del filesystem en un archivo (nombres y tamaños de los archivos, etc…) para posteriormente estudiarlo e intentar deducir problemás. *xfs_metadump* no se utiliza para realizar copias de seguridad, ya que no copia los archivos sino su información. *xfs_metadump* es una herramienta de depuración del sistema de archivos su sintaxis es: **#xfs_metadump /dev/sdaX ~/dump_my-fs**
Podemos obtener un listado de todos los inodos y rutas de archivos del sistema de archivos con el comando **xfs_ncheck**

Como ya hemos dicho, *xfs_admin* es el equivalente de *tune2fs* y algunas de sus opciones son: Utilizar la versión 2 del formato del respaldo de transacciones, obtener la etiqueta y UUID del sistema de archivos (aunque para esto tambien se puede utilizar el comando

blkid que aporta este tipo de información y no solo de la partición *XFS*), definir la etiqueta y UUID del sistema de archivos…

Respaldo de transacciones

Los sistemás de archivos con respaldo de transacciones mantienen un registro (**.journal**) con una estructura de datos que describen las operaciones pendientes de lo que se va hacer, de forma que si el sistema se cae o falla la alimentación, examinará el registro revisando las estructura de datos pendientes. Los sistemás de transacciones Linux no cuentan con una opción "deshacer" ilimitada. En caso de encontrar inconsistencias, el sistema puede revertir los cambios, devolviendo al disco un estado consistente. Al revisar solo las estructuras de datos pendientes de cambios se acelera enormemente la comprobación del disco tras un fallo, caída o corte de alimentación. Linux posee 5 sistemás de archivos con respaldo de transacciones que son: ext3, ext4, Reiserfs, XFS y JFS. No obstante ext2 puede convertirse a ext3 con tune2fs y su opción **-j**, como se describe en las opciones del comando *tune2fs*.

Revisar sistemás de archivos

Errores, fallos de alimentación o problemás mecánicos pueden afectar a las estructuras de datos de un sistema de archivos lo que puede derivar en pérdidas de datos o un rendimiento precario, por lo que es conveniente revisar periodicamente el filesystem. Linux incluye herramientas para tal fin, como por ejemplo **fsck**, que en realidad es una interfaz de usuario, front-end de **e2fsck** que a su vez este lo es para *fsck.ext2, fsck.ext3 y fsck.ext4* o **xfs_check** y **xfs_repair** de *XFS*.

El exámen se centra más en *e2fsck* que en *fsck* pero al ser *fsck* una herramienta más general y util en otros sistemás de archivos, la comentaremos de forma detallada. Aun así, en esta sección nos preocuparemos de las herramientas *fsck, e2fsck, xfs_check y xfs_repair* de igual manera.

- **fsck** : Es una herramiento para comprobar y opcionalmente reparar uno o varios sistemás de archivos, normalmente dañados o inconsistentes a causa de fallas de la red eléctrica, que apaga el sistema sin la posibilidad de que **sync** vacíe la caché del buffer en el disco que puede resultar en bloques de datos marcados como usados cuando en realidad se encuentran vacios. Otros errores se dan cuando se escribe directamente a un dispositivo en un área que ya contiene datos.
  Para usar *fsck* la mejor opción es desmontar el sistema de archivos a revisar y ejecutar el comando pasando como argumento el nombre del sistema de archivos (etiqueta), nombre de archivo de dispositivo como */dev/sda4,* un punto de montaje como */home* o un *UUID*, acompañado de alguna de sus opciones. Por regla general *fsck* intentará comprobar sistemás de archivos de diferentes discos de forma paralela para reducir el

tiempo de comprobación de todos ellos, tiempo que se incrementaria al hacerlo en serie. Si no se especifica un filesystem o la opción **-A**, *fsck* intentará revisa en serie (**-s**), todos los sistemás de archivos encontrados en el fichero */etc/fstab*.
Algunas de las acciones más utiles de *fsck* podrian ser: revisar todos los sistemás de archivos (**-A**), indicar el progreso de la operación (**-C**), mostrar una salida detallada (**-V**), realizar una simulación de lo que haría fsck (**-N**), ejecutar en modo no interactivo (**-a**), ejecutar en modo interactivo (**-r**) o revisar si existen bloques defectuoso (**-c**)

- **e2fsck** : No es más que la utilidad *fsck* pero para sistemás de archivos ext2, ext3 y ext4. Por regla general no es seguro ejecutar el comando *e2fsck* con los sitemás de archivos montados a excepción de que pasemos la opción **-n** (solo lectura y asume como 'no' la respuesta a todas las preguntas que nos realice *e2fsck*) y no vaya acompañada de **-c**, **-l** o **-L**, aun así se recomienda que solo expertos en sistemás de ficheros de la familia de extendidos ejecuten *e2fsck* con sistemás de archivos montados.
- **xfs_check** : Debe usarse cuando haya evidencias de que el sistema de archivos esta comprometido o inconsistente. Por supuesto se recomienda desmontar la unidad a comprobar o al menos montarla en solo lectura. Se aconseja usar el comando **xfs_repair -n** para mayor escalabilidad y velocidad.
- **xfs_reparir** : Util para reparar un sistema de archivos *XFS* corrupto. Este debe de estar desmontado de lo contrario los resultados no serán del todo fiables.

Monitorizar el uso del disco

Es primordial llevar un control del uso y espacio libre o consumido de un disco. Para ello disponemos de dos herramientas como son df y du.

Por particiones:

La herramienta **df** en su uso más simple (es decir sin pasar opciones ni argumentos) nos mostrará información sobre el consumo del espacio de disco para cada una de las particiones y sistemás de archivos montados en el sistema, así como su punto de montaje. *df* es util para localizar particiones en peligro de desbordarse, pero una vez obtenida dicha información será momento de especificar un poco más, el porque y en donde se está produciendo dicho desbordamiento. Para tal fin contamos con la herramienta *du*.

du busca los directorios especificados e informa de cuanto espacio consumen cada uno. La búsqueda es recursiva, aunque podemos modificar la salida de *du* mediante sus opciones como por ejemplo indicar que queremos mostrar los archivos, tener una suma total, contar enlaces de referencias, resumir la salida, etc…

Un ejemplo del uso de *du* sería ver que usuarios están haciendo un mayor uso de consumo de disco, podemos entonces crear una búsqueda simplificada (es decir que solo muestre el total de espacio para cada directorio que cuelga directamente de /home, esto sería algo como **#du -s /home/***

LPIC-1 Capítulo 4

EN ESTE CAPÍTULO SE ABARCAN LOS SIGUIENTES OBJETIVOS DEL EXAMEN:

- 103.3: Administración básica de archivos (4)
- 104.4: Administrar cuotas de disco (1)
- 104.5: Administrar los permisos y la propiedad de los archivos (3)
- 104.6: Crear y cambiar enlaces de referencia y enlaces simbólicos (2)
- 104.7: Localizar archivos del sistema y ubicar archivos en el sitio correcto (2)
- 103.8: Edición básica con Vi (3)

Filesystem y Administrar archivos

Reglas de nomenclatura de archivos y expansión de comodines

Los nombres de archivos *Linux* pueden contener letras en mayúsculas o minúsculas, números e incluso, la mayoría de caracteres de control y puntuación. Para evitar confusiones recomendamos restringir caracteres no alfanuméricos como el punto (**.**), el guión (**–**), el guión bajo (**_**), el asterisco (*****), el signo de interrogación (?), la barra (/), la barra invertida (\) y las comillas (**“**).
Algunos programás crean archivos de copias de seguridad que terminan en virgulilla (**~**), como editores de texto.
En la línea de comandos será necesario escapar los caracteres de espacio anteponiendo la barra invertida (\) o rodeando el nombre del archivo con comillas (**“**nombre archivo**“**).

Nota: Los nombres de archivos de DOS están restringidos a 8 caracteres para el nombre, seguidos de una extensión opcional de 3 caracteres, conocido esto como nombres 8.3. En Linux no es obligatorio el uso de extensiones pero es habitual ver extensiones de 1 hasta 4 caracteres, y sus nombres de archivos pueden empezar por punto “.nombre” (archivos ocultos)

En Linux existen dos nombres de archivos particularmente especiales. El archivo consistente en un solo punto (**.**) que hace referencia al directorio actual y el de dos puntos (**..**) que hace referencia al directorio principal o “padre” del directorio en el que nos encontramos.

Linux distingue minúsculas de mayúsculas en sus nombres de archivos (al contrario que Windows) por lo que entonces no es lo mismo *File.txt* que *file.txt*.

Comodines o Wildcards: Es un símbolo o conjunto de símbolo que representan a otros caracteres. Hay tres clases de comodines habituales en Linux:

- **?** : Representa un solo caracter. En expresiones regulares puede representar la opcionalidad de existencia de un caracter.
- ***** : Coincide con cualquier caracter o conjunto de caracteres, incluyendo la ausencia de caracteres.
- **[]** : Caracteres incluidos entre los corchetes representan la opcionalidad de existencia de uno de ellos en una cadena. Si por el contrario se separan dos caracteres por un guión (-), especificará que se trata de un rango de valores. Por ejemplo: Bo[a-z]t podrá hacer referencia a Boat, Boot, Bolt, etc..

Nota: El proceso de expansión de comodines se conoce como *file gobbing* o *globbing*

Comandos de archivos

Si queremos ver el contenido de un directorio mostrando así los nombres de los archivos, podemos usar el comando **ls** (*list*) con el que podemos usar diversas opciones entre ellas -F (*–file-type*) con el que podremos ver además del nombre, de que tipo de archivo se trata (***** ejecutable, / directorio, **@** enlace simbólico, = socket o | para un pipe). No confundir esta opción del comando *ls*, con el propio comando **file** con el que además de conocer el tipo de archivo de una forma más precisa, sabremos su también su formato.

Para copiar archivos usamos el comando **cp**, para mover archivos, directorios o simplemente cambiarles el nombre usamos **mv**. Tanto *cp* como *mv* comparten muchas de sus opciones. Una característica importante de *mv* es que cuando movemos un archivo de un directorio a otro dentro de un mismo sistema de archivo, *mv* actuará de una forma rápida ya que *mv* realizará la tarea reescribiendo las entradas del directorio, por lo que no habrá que leer y reescribir los datos del archivo, por el contrario si ambos directorios pertenecen a particiones o discos distintos, *Linux* debe de leer el archivo original, reescribirlo en la nueva ubicación y borrar el antiguo, lo cual ralentiza a *mv*.

Si queremos deshacernos de un archivo de forma permanente usaremos el comando **rm**. *Linux* no proporciona por defecto ningún tipo de funcionalidad de tipo “papelera” para su comando *rm*. Una vez eliminado un archivo con *rm* habrá desaparecido y no habrá forma de recuperarlo a menos que realice un mantenimiento de disco a bajo nivel con la utilidad **debugfs**.

Nota: Muchos administradores GUI de Linux si implementan una función de tipo papelera para que pueda recuperar fácilmente los archivos eliminados (a menos que haya vaciado la papelera).

Podemos modificar las marcas temporales de un archivo o crear uno vació en caso de que no exista con el comando **touch**. *touch* asigna la hora actual como hora de modificación y acceso .Los archivos nativos de *Linux* mantienen 3 marcas temporales para cada archivo:

- Hora de última modificación
- Hora de último cambio de nodo de índice
- Hora de último acceso

Nota: Hay varios programás que se basan en estas marcas temporales, como por ejemplo la utilidad make que las utiliza para determinar qué archivos del código fuente se deben de recompilar si ya existe un archivo de objeto para un archivo particular.

Comandos para la compresión de archivos o paquetes de archivos

Existen diferentes utilidades para comprimir o descomprimir archivos individuales (tratamos también un paquete de archivo *tar* como un único archivo) que usan diferentes algoritmos de compresión, herramientas como **zip**, **gzip**, **bzip2** o **xz**. *Zip* por ser antiguo y de los peores compresores de los citados no lo estudiaremos en el apartado de comando, basta decir que para comprimir un archivo con *zip* podemos hacerlo tal que así: *$zip fichero.zip <archivo_a_comprimir>* y podremos descomprimirlo de la siguiente manera: *$unzip fichero.zip*

De las tres herramientas restantes *gzip* es la más antigua y la que menor compresión ofrece, *bzip2* ofrece mejor compresión y *xz* es la más moderna y la que mejor compresión ofrece.
Gzip y *bzip2* son dos buenas herramientas de compresión pero como es lógico dependiendo de que busquemos podremos utilizar una u otra. La compresión tanto con *gzip* como con *bzip2* son muy buena, llegando incluso a reducir tamaños de binarios en un 40% – 50%. Ahora bien, si lo que buscamos es velocidad a la hora de comprimir/descomprimir deberíamos de usar *gzip*. Esto tiene un pero, y es que si usamos *gzip* a su mínima compresión será muy rápido pero a penas notaremos diferencia de compresión, mientras que si lo usamos a su máximo, será más lento y comprimirá menos que si usamos *bzip2* en su mínimo valor (ya estaría la cosa en ajustar el valor de *gzip* para que no fuese ni el mínimo ni el máximo y quizás consigamos una buena compresión en menos tiempo de lo que *bzip2* podría hacerlo). Esto deja claro que *bzip2* es más lento que *gzip* pero que comprime mejor a cambio de un mayor uso de

memoria.
Comprimir un archivo con *gzip* y *bzip2*:

$gzip <fichero>*** y ***$bzip2 <fichero>

Ambos comandos eliminan el archivo de origen.

Descomprimir un archivo gz y bz2:

$gzip -d fichero.gz y ***$bzip2 -d fichero.bz2***

Nota: También podemos usar *$gunzip fichero.gz* para descomprimir archivos .gz y *$bunzip fichero.bz2* para descomprimir archivos bz2

El funcionamiento básico de *xz* es parecido al resto de compresores. Por ejemplo si queremos comprimir o descomprimir un archivo bastará con pasarle como argumento el nombre del archivo; ***$xz archivo*** (para comprimir 'archivo' eliminándose el origen y dando como salida 'archivo.xz') y ***$unxz archivo.xz*** (para descomprimir 'archivo.xz' eliminándose este y quedando 'archivo' como resultado).
Con *xz* podemos tomar la salida de otros comandos como entrada de este, como por ejemplo: ***$tar baz.tar baz* | xz > baz.tar.xz***
Podemos cambiar el formato de compresión a *lzma* usando su opción ***-F*** <formato> (*$xz -F lzma archivo*) y usar *unlzma* para descomprimir a este.
xz nos permite modificar la salida estándar con su opción ***-c*** tanto para comprimir como para descomprimir. **xzcat** y **lzcat** nos permitirán descomprimir archivos (.xz y .lzma respectivamente) redirigiendo su salida. Esto es útil si por ejemplo tenemos varios archivos de texto, algunos comprimidos con *xz* o *lzma* y otros no comprimidos, desplegando todo su contenido en un solo archivo.

```
$xz -dcf arch.txt compri.xz otro-texto.txt unlz.lzma > unidos.txt
```

Nota: La opción -f forzará la descompresión eliminando primeramente el archivo unidos.txt ya existente y creando este último. Además con esta opción podremos usar xz con enlaces simbólicos, archivos con varios enlaces de referencia (hard links), archivos con SUID, SGID o sticky bit

Tras la compresión o descompresión *xz* utilizará el propietario, grupo de propietario, los permisos, el tiempo de acceso y la hora de modificación del archivo fuente.

Nota: xz no admite aún la copia de otros metadatos como las listas de control de acceso o atributos extendidos.

El uso de la memoria para la compresión y descompresión variará en función de los parámetros utilizados, por ejemplo según que nivel de compresión utilicemos, se necesitará más o menos memoria, siendo el nivel 6 el valor por defecto. Para descomprimir se usará de un 5% a 20% de la memoria que fue utilizada a la hora de comprimir el archivo. El uso de memoria y nivel de compresión podremos ajustarlo según las necesidades. En el apartado de comandos aprenderemos más sobre *xz*.

Comandos de archivo de archivos: tar y cpio

Las herramientas de archivo de archivos recopilan un grupo de archivos en un único paquete (archivo). Linux incluye varios comandos de archivos siendo los más destacados **tar** y **cpio**.

Nota: También se incluye dd que aunque no es técnicamente un comando de archivo nos permite copiar una partición o disco completo en un archivo o viceversa.

tar

La utilidad *tar* como comentábamos, nos permite agrupar varios archivos en un único paquete (*tarball*), que además podemos comprimir usando opciones de compresión. Según que opción pasemos, se usará una u otra herramienta para comprimir el paquete resultante: **-z** para compresión con *gzip* cuyo paquete tendrá la extensión *.tar.gzip* o *tgz*, **-j** para comprimir con *bzip2* con extensión *.tar.bz2* o *.tbz* y **-J** para compresión *xz* (extensión *.tar.xz*).

Nota: La compresión del paquete tarball reduce más el tamaño total, que si hubiésemos comprimido uno por uno cada archivo del paquete y posteriormente haber creado un tar sin compresión. La desventaja de esto es que al comprimir un paquete tarball, este se vuelve más susceptible ya que un error de un solo byte al inicio del archivo puede hacer que resulte imposible recuperar los datos.

Un *ejemplo* de agrupación y compresión con *tar* y su posterior descompresión y desempaquetado en un directorio diferente del que se encuentra el paquete *tarball*

```
$tar -cpvzf union.tar.gz *.txt
$tar -xpvzf union.tar.gz -C otrodir/
```

Nota: A la hora de descomprimir para luego desempaquetar podríamos usar también una herramienta de descompresión y pasar la salida por pipe como entrada de tar, tal que así:
$gunzip -c union.tgz | tar -xvf –

Recordatorio:

- La opción **-c** indica que envíe su salida a la salida estándar indicada (en este caso a la pipe)
- A *tar* no se le ha pasado el nombre del *tarball*, pero si, el guión (-), que asume que el nombre del archivo es el que se le pasa a la entrada de *tar* mediante la pipe que en este caso sería *union.tar* (ya que ha sido descomprimido por *gunzip* previamente)

Como siempre encontraremos más información sobre los comandos en su respectivo apartado de comandos del tema en cuestión.

cpio

En principio la utilidad *cpio* es similar a *tar* pero los detalles de su funcionamiento difieren en algunos aspectos. *cpio* fue inicialmente creado para el almacenamiento de copias de seguridad en cintas magnéticas aunque su uso es el de copiar ficheros en o desde el árbol de directorio o archivo cpio.
Un archivo *cpio* (.cpio) consiste en una serie de ficheros y directorios, con los encabezados utilizados por *GNU CPIO* para extraer el archivo, y otros encabezados extras que contienen el nombre, fecha de creación, permisos y propietarios de cada archivo o directorio.
cpio tiene tres modos de operación: **copy-out** (-o, –create), **copy-in** (-i, –extract) y **copy-pass** (-p, –pass-trough)

Nota: Antes de explicar estos modos, aclararemos que tenemos que tener en cuenta que "dentro" será nuestra posición en el sistema de archivo, es decir, en el árbol de directorio, y "fuera" será el sitio que ocupa un archivo o directorio cuando se encuentra en el interior de un archivo .cpio.

Copy-out: Aclarado "fuera/dentro" en la nota anterior, entendemos que "copiar fuera" será copiar archivos/directorios desde el filesystem a un archivo .cpio (lo que llamaríamos en tar, empaquetar). Para llevar a cabo esto el comando *cpio* leerá una lista de nombres de archivos/directorios los cuales serán añadidos al archivo .cpio. Para pasar esta entrada es muy normal usar el comando *find* y redireccionar su salida mediante una pipe al comando *cpio*.

```
$find /home/user/desktop/ | cpio -o > /tmp/miescritorio.cpio
$find /home/user/desktop/ | cpio -o | gzip /tmp/miescritorio.cpio.gz
```

(comprimiremos el archivo .cpio resultante)

Nota: Esto copiará todos los directorios y ficheros que tenemos en nuestro Escritorio en el archivo /tmp/miescritorio.cpio. Podríamos pasar la opción -depth para minimizar los problemás con directorios/subdirectorios a los que no podremos acceder por ejemplo, por no tener permisos.

Copy-in: Sabiendo que hace *Copy-out*, podemos imaginar lo que hará esta opción, "extraerá" o mejor dicho copiará el contenido de un archivo .cpio en nuestro árbol de directorio. Podemos "extraer" solo determinados ficheros o todo su contenido (si no pasamos un patrón).

```
$cpio -i < /tmp/miescritorio.cpio (todo el contenido)
$cpio -i solo-este-archivo < /tmp/miescritorio.cpio
```

Copy-pass: Básicamente lo que hace esta opción es copiar un árbol de directorios en otro directorio, sin usar un archivo .cpio como origen o destino. *Copy-pass* combina las dos opciones anteriormente explicadas. Si por ejemplo queremos tener un respaldo de

un árbol de directorio en otro dispositivo, sistema de archivo remoto o una partición montada en el equipo, *Copy-pass* puede ser una buena opción.

```
$find . -print | cpio -pvd /mnt/destino
```

Recordatorio:

- Imprimiremos los nombres de archivos con **-print**
- La opción **-p** indica que será una Copy-pass y **-d** creará los directorios en el destino para ir alojando los archivos respaldados.

Podemos crear backups de determinados sistemás de archivos o directorios en un dispositivo de destino:

```
$find /home /opt /usr -xdev | cpio -oF /dev/st0
```

Referencias:

http://manpages.ubuntu.com/manpages/oneiric/es/man1/cpio.1.html

http://osr507doc.sco.com/en/OSUserG/_Creating_a_cpio_backup.html

dd

La utilidad **dd** puede ser un buen modo de crear copias exactas de particiones completas (ya que incluye incluso todos los espacios vacíos), hacer una copia exacta de un disco extraible (incluido los discos ópticos), copiar un disco para el que Linux carece de controladores del sistema de archivos, crear varias instalaciones idénticas de Linux en varios ordenadores (siempre que estos tengan discos duros del mismo tamaño) o crear un archivo vacío de un tamaño particular que pueda manipular posteriormente dándole formato con *mkfs*.

Los dispositivos ópticos requieren procedimientos de copias de seguridad especiales. Una buena forma es crear un archivo *tar* o *cpio* en disco, después utilizar *mkisofs* para dar formato *ISO-9660* o *UDF* y posteriormente grabar el archivo de imagen en el disco óptico. El resultado será un disco que podrá montar y que contendrá un archivo que podrá leer con *tar* o *cpio*.

Otra opción es crear un archivo del archivo y grabarlo en el disco óptico con *cdrecord*. Un disco de este tipo no se podrá montar, pero podría acceder directamente al archivo utilizando el archivo del dispositivo CD-ROM.

Algunos inconvenientes del comando *dd*:

- Una partición de 5GB que contiene sólo 5MB de archivos, requerirá 5GB de espacio de almacenamiento. Esto es debido a que *dd* copia los espacios en blanco

- Para restaurar archivos individuales, a menos que sea un dispositivo de acceso aleatorio que se pueda montar, deberá restaurarlo todo para recuperar un único archivo
- No resulta fácil restaurar la información en una partición que es más pequeña que la original y en caso de ser una partición mayor, desperdiciará parte del espacio disponible

En el siguiente ejemplo, se realiza la copia de la partición 3 del dispositivo sda, en un dispositivo extraible (sdb).

```
$dd if=/dev/sda3 of=/dev/sdb1
```

Nota: Para restaurar la copia bastará con invertir el nombre de if (InputFile) y of (OutputFile). Podemos usar el tamaño completo de sdb si no especificamos partición.

Los enlaces

Un enlace es un medio de proporcionar varias identidades a un comando, archivo o directorio (similar a un acceso directo de Windows) facilitándoles la accesibilidad, por ejemplo, permitir a los programás, buscar los mismos archivos en distintas ubicaciones o, referenciar a un ejecutable que se encuentra "escondido" tras una larga ruta, desde un enlace creado en el directorio en el que trabajamos.

Hay dos tipos de enlaces en Linux: Enlace de referencia (*hard links*) o enlaces simbólicos (*soft links*).

Los **enlaces de referencia** normalmente no se pueden crear a directorios, solo root puede intentarlo si pasa como argumento las opciones -d, -F o –directory. En la práctica es improbable que esto funcione, ya que la mayoría de los filesystem no lo soportan, en cambio, esto no supone ningún problema para los enlaces simbólicos.
Los enlaces de referencias creados por defecto con **ln** (sin opciones) produce dos entradas de directorio que apuntan al mismo archivo (más concretamente, al mismo nodo de índice). Ninguno de los dos nombres es más auténtico que otro por lo que para borrar el archivo, deberemos de eliminar ambos enlaces de referencia. Otra restricción de los enlaces de referencia es que deben referenciar a archivos dentro del mismo sistema de archivos a bajo nivel, por ejemplo, no podríamos crear un enlace duro o de referencia (*hard link*) desde / (raíz) hacia /home, suponiendo que /home esté en una partición independiente.

Por el contrario, **los enlaces simbólicos** son un tipo especial de archivo (archivos que contienen nombres de archivos) cuyo contenido apunta al archivo enlazado. Linux accede al archivo enlazado (archivo original) cada vez que intenta acceder al enlace

simbólico, por lo que acceder a un enlace simbólico es como acceder al archivo original. Esto nos permite crear punteros entre sistemás de archivos de bajo nivel.

En la teoría acceder al archivo original desde el enlace simbólico consume una cantidad mínima de tiempo, por lo que es más lento acceder a los enlaces simbólicos que a los enlaces de referencia, aunque no lo suficiente como para percatarnos de ello.

En la práctica, los enlaces simbólicos son más comunes que los de referencia; sus desventajas son menores y la capacidad de enlazar entre sistemás de archivos independientes y directorios puede ser importante.

Nota: Cada enlace simbólico consume un único ínodo de índice.

La sintaxis para crear un enlace simbólico es básica:

```
$ln [opciones] <archivo-origen> <nombre-enlace>
```

Propiedad, Permisos y Comandos de directorios

La mayoría de comandos que se aplican a archivos, sirven para directorios: ls, mv, cp, rm, touch…

Para crear un directorio usamos el comando **mkdir** (*make directory*) y para eliminarlo **rmdir** (*remove directory*). Aclarar que para eliminar un directorio que contiene archivos usaremos **rm -r** (*recursive*) ya que *rmdir* no elimina archivos, deberíamos de eliminar primero todos los archivos y luego usar *rmdir -p* para borrar también los subdirectorios.

Administrar los permisos y la propiedad de los archivos

Podemos saber quien es el propietario de un archivo con el comando *$ls -l* (list), que genera un listado largo, que incluye la información de propiedad y permisos.

$ls -l

```
-rwxrwxrwx 1 nebul4ck tripas 5059518 dic  6  2013 1. Virginia Alexandre - Lady M
-rwxrw-r-- 1 nebul4ck tripas     432 ene 18 14:43 abcd.sh
-rw-r--r-- 1 nebul4ck tripas   10240 ene 18 04:05 arch.tar
drwxr-xr-x 2 nebul4ck tripas      80 ene 19 10:46 descom
lrwxrwxrwx 1 nebul4ck tripas      23 ene 16 21:35 logo -> dire1/este/logo-web
```

Usemos este listado para estudiar algunas de las propiedades. Por ejemplo el primer dígito hace referencia al tipo de archivo (**–** : archivo regular, **d** : directorio y **l** : enlace), existen además otros tipos como: **p** para pipes, **s** para Socket (similar a pipe pero

permite enlaces de red y comunicaciones bidireccionales), **b** para dispositivo de bloque (archivo que corresponde a un dispositivo de hardware que transfiere información en bloques de más de un byte de forma bidireccional, como los discos duros, disquetes, CD-ROM, etc..) y **c** para dispositivos de caracteres, que se corresponden con un dispositivo de hardware, hacia y desde el que se transfieren los datos en unidades de un byte (por ejemplo dispositivos con puertos paralelos y RS-232 en serie).
Los siguientes 9 dígitos engloban a los **permisos** del usuario **propietario** del archivo (los 3 primeros dígitos), los permisos del **grupo** primario del usuario propietario del archivo (los 3 siguientes dígitos) y los últimos 3 dígitos para el resto de usuarios, llamado también permisos generales o **globales**.
Linux codifica esta información en formato binario (cada dígito o permiso individual se le suele denominar bit de permiso).

Nota: Cada grupo de 3 bits de permisos se expresa en base 8 por lo que se usan los dígitos del 0 al 7. Siendo el valor 4 para el bit de lectura (r), 2 para el bit de escritura (w), 1 para el bit de ejecución o acceso (x) y 0 para la ausencia de permiso que se representa con un guión (-).

Conociendo el formato de los bits de permisos podremos deducir la representación numérica de los permisos. Ahora cojamos la 2ª línea del listado para su estudio:

```
-rwxrw-r-- 1 nebul4ck tripas     432 ene 18 14:43 abcd.sh
```

Esta línea indica que el usuario propietario del archivo (*nebul4ck*) tiene permiso total sobre el archivo **r** (4), **w** (2) y **x** (1) que numéricamente sería 7, los usuarios pertenecientes al *grupo primario* de nebul4ck (*tripas*) tendrán acceso de lectura (4) y escritura (2) (suman un total de 6) y el resto de usuarios solo de lectura (4) por lo que la representación numérica definitiva sobre el archivo *abcd.txt* sería de **764**

Volviendo a la imagen del listado del directorio vemos que la primera línea del listado muestra la existencia de un directorio (**d**) y un enlace simbólico (**l**), hay que resaltar dos cosas sobre esto: la primera es que nunca (o no sería normal) veremos un directorio en el que el bit de ejecución o acceso no se defina junto al de lectura, ¿Por que? porque si tiene el bit de acceso es para que el usuario acceda al directorio y pueda listar su contenido (necesario el permiso de lectura 'r'), de lo contrario para saltar a otro directorio dentro de este, deberá de conocer el nombre del siguiente directorio (ya que al no poder listar su contenido, desconocemos los directorios que pudiese haber dentro, algo "útil" si queremos esconder un subdirectorio dentro del directorio y que "solo" nosotros podamos acceder conociendo el nombre de este). La segunda nota a resaltar es que cuando creamos un enlace simbólico este se crea con permisos 777 (rwxrwxrwx) para que todos los usuarios puedan leer el contenido del enlace para saber el nombre del archivo al que apunta, pero los permisos del archivo enlazado son los que determinarán finalmente el acceso a este.

Nota: El cambio de los permisos del enlace simbólico afectará al archivo enlazado, pero el enlace seguirá con permisos 777

Si un usuario puede escribir en un directorio (el directorio tiene permiso w), dicho usuario podrá crear, borrar o renombrar los archivos del directorio, incluso si el usuario no es el propietario de dichos archivos y además no tiene permisos de escritura sobre estos. Es el permiso w del directorio el que tiene preferencia sobre su contenido y no los permisos individuales de los que cuelgan de este.

Nota: El superusuario puede leer o escribir en cualquier archivo del ordenador, incluso en aquellos que tienen permisos 000

Bits de permisos especiales y ACL

Existen opciones adicionales para los permisos, que se pueden indicar mediante cambios en la cadena de permisos:

- Definir el ID de usuario (set user ID, **SUID**): Se utiliza en archivos ejecutables indicándole a Linux que ejecute el programa con permisos del propietario del archivo y no con los permisos del usuario que en realidad ejecuta el programa. El permiso *SUID* está indicado por una "**s**" en la posición del bit de ejecución del propietario del archivo (rw**s**r-xr-x).
- Definir el ID de grupo (set group ID, **SGID**): Similar a la opción *SUID*, pero asigna el bit "**s**" en la posición del bit de ejecución del grupo propietario del programa. (rwxr-**s**-r-x). Cuando se define el bit *SGID* en un directorio, los nuevos archivos o directorios creados bajo este heredarán el grupo propietario del directorio y no el grupo principal del usuario que ha creado realmente el archivo.
- Definir el **Sticky bi**t: En las versiones más modernas de Unix se utiliza el *Sticky bit* para evitar que los archivos que se encuentran dentro de un directorio con permisos de escritura (w) puedan ser borrados por usuarios que no son su propietario. Cuando este bit está presente en un directorio, solo podrán eliminar los archivos del directorio los propietarios de estos, el propietario del directorio o root. El *Scticky bit* se indica con una "**t**" en la posición del bit de ejecución de los permisos generales (rwxr-xr-**t**). Si aplicamos *sticky bit* sobre archivos ejecutables, provocará que estos programás permanezcan en el área *swap* lo cual resulta en programás que cargan más rápido.

Advertencia: Los programás con **SUID** y **SGID** (y en particular los programás root *SUID*) suponen riesgos potenciales para la seguridad ya que al ser ejecutados por un usuario normal, este tomará privilegios de root (o del usuario propietario del programa) durante la ejecución de ese programa. Si el programa ejercer tareas de eliminación de archivos críticos o acceso a sitios donde solo *root* podría acceder, dicho usuario no tendrá problemás a la hora de realizar estas tareas. Es decir el usuario conseguirá una elevación de privilegios.

Nota: Cuando estudiemos a continuación los comandos para el cambio de propietario, grupo y modos de un archivo o directorio, veremos como se asignan los bits SUID, SGID y Sticky bit, pero de momento decir que el bit SUID es asignado por un 4, el SGID por un 2 y el Sticky bit por un 1.

ACL

Una *Lista de Control de Acceso* es una lista de usuario o grupo y los permisos que tienen asignados sobre determinados archivos o directorios. Una *ACL* consta de 3 bits al igual que los permisos ordinarios (r, w y x), pero difiere en que no está limitada, bien a un grupo propietario (en el que deberíamos de añadir a todos los usuarios que quisiéramos distinguir del resto de usuarios) o a permisos generales (es decir a todos los usuarios) si no que nos permite que el propietario del archivo pueda asignar diferentes permisos a diferentes grupos o usuarios.

Nota: Para crear una ACL el sistema de archivo debe de tener soporte para esta. Actualmente los principales sistemás de archivos de Linux soportan ACL, pero puede que tenga que recompilar su kernel (o, al menos, el módulo pertinente del kernel) para activar dicha compatibilidad.

Para definir o visualizar *ACL* estás cuentan con sus propios comandos. Con **setfacl** podremos definir una *ACL* y con el comando **getfacl** podremos mostrar la *ACL* de un archivo.

Importante crear una *ACL* por defecto, así cada vez que creemos contenido dentro de un directorio en el que hemos definido la *ACL* heredará los permisos de este. En el apartado de comandos, tratamos el parámetro adecuado para tal fin.

Cambiar el propietario, grupo, modos y atributos de los archivos

El superusuario puede cambiar el propietario de un archivo o directorio mediante el comando **chown**.

La sintaxis del comando *chown* es:

```
$sudo chown [opciones] nuevpropietario:nuevogrupo <archivos a modificar>
```

Nota: El comando chown acepta el punto (.) como separador de nuevousuario y nuevogrupo, no obstante el uso del punto es obsoleto por lo que esta característica podrá acabar desapareciendo.

Para cambiar el grupo propietario de un archivo se usa **chgrp**, el cual acepta mucha de las opciones del comando *chown*. Al igual que *chown* solo root podrá cambiar el grupo

propietario de un archivo, aunque para *chgrp* existe una excepción y es que un usuario normal podrá cambiar el grupo principal de un archivo de su propiedad siempre y cuando el usuario pertenezca al nuevo grupo.

Los permisos de los archivos o directorios los pueden cambiar el superusuario y el propietario del archivo, se cambian con el comando **chmod** y su sintaxis es:

```
$sudo chmod [opciones] modo <nombre archivo>
```

Las opciones de *chmod* son similares a las de *chown* y por tanto a las de *chgrp*. Podemos especificar el modo de un archivo de dos formás básicas: *como un número octal* o *como un modo simbólico.*

- **Modo octal**: *$sudo chmod **644** archivo.txt*

Nota: Si no usamos 4 dígitos, chmod limpiará los bits SUID, SGID y Sticky Bit. Esto en ocasiones no funciona por lo que deberemos de limpiar los dígitos de forma simbólica: chmod u-s archivo.txt

Modo octal indicando el bit *SGID*: *$sudo chmod 2644 archivo.txt*
Modo octal indicando el *Sticky bit*: *$sudo chmod 1644 archivo.txt*
Modo octal indicando los bits *SUID*, *SGID* y *Sticky bit*: *$chmod 7644 archivo.txt*

- Un modo simbólico consta de tres componentes:

1. Código que indica el conjunto de permisos que desea modificar: propietario (**u**), el grupo (**g**), globales (**o**) o todos (**a**)
2. Símbolo que indica la acción a realizar; añadir (**+**), quitar(**–**) o definir el valor indicado (**=**)
3. Por último, el código que especifica cuál debería ser el permiso: rw, rwx, rx, etc…

Nota: Estos códigos diferencian entre mayúsculas y minúsculas

Ejemplo 1: Dar permisos **rw** al propietario, **r** al grupo y quitar **w** al grupo:

```
$sudo chmod u+rw,g+r,g-w report.txt
```

Nota: Si el grupo anteriormente ya tenía permisos de escritura y el resto de usuarios por ejemplo permisos de lectura, este comando no anulará lo que ya había. Para especificar el valor concreto, anulando posibles permisos dado anteriormente usamos (=)

Ejemplo 2: Dar permisos concretos (anula lo que ya había):

```
$sudo chmod u=rwx,g=rx,o=r archivo.txt
```

Al igual que otorgamos permisos de forma simbólica para r,w y x, podemos hacerlo para *SUID* y *SGID* (s), y *Sticky bit* (t). Además existe la opción **X** que permite ejecutar solamente si el archivo es un directorio o si el archivo tiene permisos de ejecución.

Por lo general usamos modos simbólicos para dar permisos sencillos, como u+x (para que el usuario tenga permisos de ejecución tras crear un script) y modo octal para dar permisos más específicos como rw-r-xr–

Importante: Debemos de saber que los permisos de archivos se almacenan como parte del nodo de índice del archivo, que no forma parte de la entrada del directorio. El acceso de lectura y escritura a la entrada del directorio o, incluso, al propio archivo, no le da al usuario el derecho de cambiar las estructuras de nodos de indice (a menos que lo haga de un modo indirecto, como por ejemplo, cuando al escribir se cambia el tamaño del archivo o cuando un borrado de archivo elimina la necesidad del nodo de índice).

Umásk y el grupo por defecto

Podemos definir el modo y el grupo de un archivo por defecto. Cuando un usuario crea un archivo, dicho archivo tiene un propietario predeterminado (por defecto su creador y el grupo principal de este) y unos permisos predeterminados que vienen definidos por la máscara del usuario. Esta máscara se puede modificar con el comando **umásk** (*umásk* es una operación a nivel de bits).

El comando *umásk* recibe como entrada un valor octal que representa los bits a eliminar de los permisos 777 de los directorios, o 666 de los permisos de archivos.
Pongamos un ejemplo: Supongamos que tenemos una *umásk* de 022 o 002 (suelen ser las *umásk* por defecto), si creamos un directorio este se creará con los permisos 755 o 775 (777 – 022 y 777- 002), si creamos un archivo los permisos serán 644 y 664.

Nota: Si tenemos una umásk de 247, los permisos de los nuevos directorios serán 530 y 420 para archivos, como vemos no importa que se reste 7 al valor máximo de un archivo que es 6, pues en este caso umásk no afecta al bit de ejecución (que es el que eleva a 7 la suma total y que por defecto no viene en los archivos). Esto es porque como dijimos, umásk es una operación a nivel de bits, por lo que si un bit no está definido (como es el caso del bit de ejecución en ficheros) no habrá donde operar.

Los usuarios normales pueden introducir el comando *umásk* para cambiar los permisos de los nuevos archivos que creen. El superusuario también puede cambiar la configuración por defecto para todos los usuarios, modificando un archivo de configuración del sistema (*/etc/profile*).

Advertencia: La configuración de *umásk* en el archivo */etc/profile* puede ser invalidada por los archivos de configuración de perfil de inicio de cada usuario (*~/bash_profile*).

Podemos ver la configuración de *umásk* en octal con el comando *umásk* y de forma simbólica con *umásk -S*. Al igual que los modos de los archivos, podemos definir la

umásk por defecto de forma simbólica, pero de hacerlo así deberemos de indicar los bits que si queremos definir, por ejemplo,

```
$umásk u=rw,g=rx,o=rx (644)
```

Además de la máscara, los usuarios pueden cambiar su grupo predeterminado para nuevos archivos con el comando **newgrp**. Para tal cambio el usuario deberá de pertenecer a dicho grupo. Para reinicializar el entorno y hacer que los cambios tengan efectos usamos *newgrp -l*

Por último decir que algunos sistemás de archivos nativos de Linux admiten varios atributos que se pueden ajustar con el comando **chattr,** veremos de que atributos se trata en el apartados de comandos de este mismo capítulo.

Administrar las cuotas de disco

Las cuotas de disco son límites que implementa el SO para controlar el número de archivos o la cantidad de espacio en disco que puede consumir un único usuario. El sistema de cuotas de Linux admite cuotas tanto para usuarios individuales como para grupos. Estas cuotas son independientes unas de las otras, esto quiere decir que el total de cuota en disco de los usuarios de un grupo no está relacionado con la cuota aplicada al grupo. Por ejemplo, imaginemos que tenemos 5 usuarios con una cuota en disco individual de *500MB* cada uno, y todos son miembros del grupo '*sistemás*', el total de cuota en disco de los usuarios sería de *2,5GB*, pero no por ello la cuota de disco del grupo '*sistemás*' tiene que tener tal tamaño, puede que tan solo tenga *1GB*. Cuando un usuario crea un archivo, el archivo tendrá como propietario a ese usuario y su grupo principal que puede que no sea el grupo '*sistemás*' por lo que ese archivo consumirá espacio de la cuota del usuario (*2,5GB*) pero no de la del grupo '*sistemás*'. En cambio si la cuota de disco del grupo '*sistemás*' se llena y un usuario tiene como grupo principal a '*sistemás*' este usuario tendrá que cambiar de grupo para poder crear archivos ya que la cuota de disco del grupo '*sistemás*' al estar llena le impedirá crear nuevos archivos.

Importante: Cuando un usuario crea un archivo nuevo, primero se revisa la cuota de disco del grupo propietario del usuario, si esta está llena el usuario no podrá crear archivos, aunque su cuota de disco individual este vacía. Además una vez creado el archivo el tamaño de este será sumado a la cuota de grupo y no a la de usuario. Por ello es aconsejable que se creen cuotas para usuarios individuales y grupos no principales para ninguno de los usuarios.

Las cuotas requieren soporte tanto en el kernel como en varias utilidades del espacio de usuario. Los sistemás de archivos ext2, ext3, Reiserfs, JFS y XFS admiten cuotas de disco.
Para el uso de cuotas debemos activar explícitamente el soporte mediante la opción "*Quota Support*" del *Kernel* en el área del sistema de archivos, cuando recompilemos el

kernel aunque muchas distribuciones vienen con este soporte precompilado.
Hay dos sistemás de soporte para cuotas disponibles: **v1** (kernel 2.4.x) y **v2** (serie de kernels 2.6.x +). Ambas funcionan de modo similar.
Se necesitarán herramientas de soporte para emplear cuotas. Para cuota v2 el paquete se suele denominar "*quota*" e instala varias utilidades, archivos de configuración, scripts de inicio del sistema,etc..
Deberemos de modificar las entradas de */etc/fstab* para cualquier partición con la que vayamos a usar cuotas, en particular añadir la opción de montaje del sistema de archivos *usrquota* (para cuotas de usuario) y *grpquota* (para las cuotas de grupo), tal que así:

/dev/sda5 /home ext3 usrquota,grpquota 1 1

Por regla general los pasos necesarios a dar son: cargar el módulo del kernel para el uso de cuotas con modprobe, si fuera necesario ejecutar el script de inicio (que usa el comando *quotaon* para activar el soporte de cuotas) y re-montar los sistemás de archivos implicados en el uso de cuota (una vez definido como comentamos anteriormente en */etc/fstab*) con el comando *mount -o remount /sistema/de/archivo*.

Nota: Si hemos tenido que recompilar el kernel para añadir directamente el soporte para cuotas, una vez definido en fstab el uso de cuotas, y habiéndonos asegurados que los script de inicio se van a encargar de ejecutar el soporte con quotaon, deberemos entonces de reiniciar.

LLegados a este punto el soporte para cuotas debería de estar totalmente activo, ahora llega el momento de establecer las cuotas de disco para los usuarios o grupos.

El comando **edquota** <usuario> inicia el editor Vi con un archivo de configuración temporal (*/etc/quotatab*, que proporciona información sobre el número de bloques de disco y de nodos de índice en uso) que controla las cuotas para el usuario especificado (podemos usar la opción -g <grupo> para indicar un grupo). Al finalizar, *edquota* utilizará este archivo de configuración temporal para escribir la información de cuota en las estructuras de datos de disco de bajo nivel.

Existen límites de **cuota estrictos** (*hard*) y **límites flexibles** (*soft*).

- Los límites estrictos son el número máximo de bloques o nodos de índice que puede consumir el usuario. El *kernel* no permitirá a un usuario sobrepasar estos límites.
- Los límites flexibles son en cierto modo menos restrictivos; los usuarios pueden exceder temporalmente sus valores, pero en tal caso el sistema emitirá un aviso. Estos límites trabajan con un "periodo de gracia", si su límite de cuota se excede durante un tiempo mayor al establecido, el *kernel* comenzará a tratarlos como límites estrictos y no permitirá al usuario crear más archivos. Podemos definir este "periodo de gracia" con el comando *edquota* y su opción **-t**.

Nota: Los periodos de gracia se definen para todo el sistema de archivos.

Podemos anular un límite de cuota indicando un valor de 0 y además ajustar los límites de cuota de forma independiente para cada sistema de archivo.

El comando **quotacheck** verifica y actualiza la información de las cuotas en los discos con cuotas activadas, además de asegurar que la contabilidad de las cuotas sea la adecuada.

El comando **repquota** resume la información de las cuotas del sistema de archivos especificado o de todos los sistemás de archivos si le pasa la opción **-a**. Esta herramienta puede ser muy útil para seguirle la pista de uso del disco a un usuario o filesystem.

El comando **quota** posee un efecto similar. *quota* ***-g*** muestra las cuotas del grupo, con **-l** omite los montajes NFS y **-q** limita la salida a los sistemás de archivos cuyo uso sobrepasa el límite.

Aprender a localizar archivos y las herramientas útiles

Linux proporciona varias herramientas para ayudarnos a localizar archivos, pero antes de usarla es recomendable conocer como está estructurado el árbol de directorios de un sistema Linux y que podemos encontrar en cada uno de ellos, esto nos facilitará o nos permitirá hacer búsquedas más concisas.

Con la implantación de **FSSTND** (*Filesystem Standar*, Estándar de Sistema de Archivos) se estandarizó varias funciones específicas que debía de cumplir la jerarquía del árbol de directorios Linux

- Estandariźo los programás que se encontraban en */bin* y */usr/bin*
- Especificó que los archivos ejecutables no debían de estar en */etc*
- Eliminó los archivos cambiables del árbol de directorios */usr*, permitiendo así que este se montara en modo de sólo lectura (una útil medida de seguridad)

Existen tres versiones principales de *FSSTND*: 1.0, 1.1 y 1.2, no obstante hacia 1995, las limitaciones de *FSSTND* se hicieron evidentes, por lo que se desarrollo un nuevo estándar: **FHS** (*File Hierarchy Standar*, Estándar de Jerarquía de Sistema de Archivo), basado en *FSSTND*, pero que lo amplia sustancialmente.

FHS hace una importante distinción entre archivos compartibles y no compartibles, y archivos estáticos y dinámicos. La mayoría de los programás ejecutables son claro ejemplo de archivos estáticos mientras que los directorios y las colas de correo están compuestos por archivos variables.

FHS intenta aislar cada directorio en una celda de esta matriz de 2×2 (compartible/no compartible x estático/variable)

	Se pueden compartir	No se pueden compartir
Estáticos	/usr /opt	/etc /boot
Variables	/home /var/mail	/var/run /var/lock

Nota: Algunos directorios son mixtos pero en estos casos, FHS intenta especificar el estado de los subdirectosios concretos, por ejemplo, /var es variable y contiene algunos subdirectorios compartibles y otros no compartibles.

Algunos distribuidores de Linux, en especial Fedora, iniciaron cambios que se alejan de *FHS*, como por ejemplo, desde Fedora 17, todos los binarios se incluyen en */usr/bin* y */usr/sbin*, siendo ahora los directorios */bin* y */sbin* enlaces simbólicos. Este sistema complica algunos tipos de configuraciones como las que requieren una partición */usr* independiente.

Los directorios más comunes definidos por *FHS* o utilizados por convención son:

- **/** : Directorio raíz. El resto de directorios cuelgan de este. Algunos directorios como /etc o /sbin deben de residir en la misma partición de raíz.
- **/etc** : Archivos de configuración y scripts de *shell* como por ejemplo los de inicio SysV
- **/etc/skel** : Contiene los archivos de configuración estándar para ser copiados a los directorios home cuando una nueva cuenta de usuario es creada.
- **/etc/X11** : Archivos de configuración de X Window
- **/Lost+found** : Directorio usado para los archivos recuperados.
- **/boot** : Contiene archivos estáticos y no compartibles relacionados con el arranque del ordenador. Algunos sistemás imponen límites a /boot, por ejemplo, las antiguas BIOS x86 y versiones antiguas de LILO pueden necesitar que /boot se encuentre por debajo del cilindro 1024 del disco. Algunos métodos de arranque EFI, funcionan mejor con /boot como partición independiente bajo ext2fs o Reiserfs
- **/bin** : Contiene los comandos que son accesible por todos los usuarios. En sistemás como Fedora, suele ser un enlace simbólico a /usr/bin.
- **/sbin** : Contiene programás que por regla general solo son ejecutados por root (herramientas de gran impacto como por ejemplo e2fsck y fdisk)
- **/lib** : Contiene las bibliotecas de los programás, y bajo /lib/module se encuentran los módulos del kernel

Nota: /bin, /sbin y /lib, contienen archivos estáticos y pueden ser compartidos, aunque en la práctica esto no suele tener mucho sentido.

- **/usr** : Aloja la mayoría de los programás Linux. Su contenido es estático y compartible y como medida de seguridad puede ser montado como solo lectura. Muchos administradores separan /usr en una partición independiente. Algunos directorios que cuelgan de /usr como bin y lib, aun siendo importantes, no lo son lo suficiente como para impedir el funcionamiento de un ordenador.
- **/usr/local** : La idea de este directorio es tener un área que esté a salvo durante las actualizaciones automáticas del software, por lo que puede ser un buen sitio para alojar archivos que instala localmente el administrador de sistemás, de echo muchos suelen crearle su propia partición para mantener a salvo estos archivos/programás en futuras actualizaciones del sistema o migración de OS.
- **/usr/local/bin** : Archivos binarios de software instalados después de la instalación del sistema operativo.
- **/usr/X11** : Hace tiempo este directorio se utilizaba frecuentemente para albergar los archivos relacionados con el sistema de ventanas X, pero las nuevas distribuciones han movido el contenido de este a otros directorios como /usr/bin
- **/usr/bin** : Aquí residen la mayoría de las aplicaciones del usuario. Debe de formar parte del valor de la variable PATH de los usuarios.
- **/usr/sbin** : Binarios no esenciales y ejecutados por root.
- **/usr/include** : Los archivos headers estándares de C/C++
- **/usr/lib** : Librerías estáticas, así como los subdirectorios para las librerías de lenguaje no C/C++
- **/usr/src** : Archivos fuentes de la mayoría de los paquetes instalados en el sistema
- **/usr/src/linux** : El código fuente de Linux
- **/opt** : Este directorio es similar a /usr/local solo que suele almacenar archivos y programás desarrollado por terceros, como procesadores de textos, juegos, etc… Por ello muchos administradores creen conveniente crear una partición independiente de manera que su contenido pueda ser respaldado, migrado y compartido.
- **/home** : Directorio compartible y variable que contiene los datos de los usuarios. Aunque para el estándar FHS se considera opcional, realmente lo opcional es su nombre, por ejemplo al añadir un nuevo disco para crear un segundo home (/home2) para separar los home de unos usuarios y los de otros. Por esta razón suele estar en una partición independiente.
- **/root** : Es el directorio home del superusuario
- **/var** : Contiene archivos efímeros de varios tipos: registros del sistema (logs), cola de impresión, correo (spool), noticias, etc… Al igual que /home, /opt y en ocasiones /usr/local, /var suele estar en una partición independiente, en especial, en sistemás que registran gran actividad como servidores de impresión, de correos, sistemás que suelen generar una cantidad importante de log de servicios, etc…
- **/tmp** : Aloja los archivos temporales. La mayoria de las distribuciones incluyen rutinas que limpian este directorio periódicamente (a veces al inicio del sistema). Colocar a /tmp en una partición independiente puede ser una buena idea, ya que puede ocurrir que procesos no controlados provoquen problemás en el sistema de archivos raíz al crear archivos temporales demásiado grandes.

- **/mnt** : Ubicación tradicional para montar dispositivos extraibles, aunque está siendo reemplazado por el directorio /media, que es opcional en FHS.
- **/dev** : Contiene un gran número de archivos que hacen de interfaz de hw. Como ya vimos es controlado por el kernel y herramientas de soporte (udev), que crean sobre la marcha los archivos de dispositivos conectados al sistema. Un usuario con privilegios suficientes podrá acceder (escribir/leer) al hardware directamente a través de estos archivos de dispositivo. Algunos archivos de dispositivos importantes son: **/dev/hdX** y **/dev/sdX** (para discos rígidos), **/dev/fdX** (para discos flexibles), **/dev/md0** (primer grupo de meta-discos, dispositivo RAID), **/dev/stX** o **/dev/nstX** (para cintas magnéticas), **/dev/loopX** (para montar CD-ROM), **/dev/lpX** (puertos paralelos), **/dev/null** (dispositivo interpretado como un cubo donde almacenar bits que nunca más se volverán a ver, es como un agujero negro donde poder, por ejemplo, enviar la salida de comandos que no queremos que sean mostradas), **/dev/psaux** (puerto para el ratón PS/2, aunque existen otros como **/dev/usb/usbmouse** o **/dev/input/mice**), **/dev/zero** (dispositivo del que poder obtener ceros), **/dev/ttySX** (puertos serie).
- **/proc** : Es un sistema de archivo virtual que Linux crea dinámicamente para proporcionar acceso a ciertos tipo de información del hardware: info de cpu, IRQ, direcciones de E/S, DMA, procesos, etc…

Nota: Colocar un programa en la ubicación equivocada puede provocar problemás a largo plazo, por ejemplo, si coloca un archivo binario en /bin cuando este debería de ir en /usr/local/bin, puede que dicho programa se sobreescriba o incluso se elimine al realizar una actualización del sistema.

La lista anterior de directorios esta incompleta, si quieres saber más (**obligatorio para administradores de sistemás**) puedes acceder a la página oficial de FHS http://www.pathname.com/fhs/ , aun así haz -> clic <- aquí para ver un listado con los directorios/archivos más importante o utilizados. Es un extracto del libro "Guía para administradores GNU/Linux v0.8″ (Traducción en TLDP-ES, http://es.tldp.org/)

Ahora que conocemos de alguna forma el árbol de directorios vamos a intentar localizar archivos de una forma más precisa. Para encontrar los archivos del ordenador Linux emplea comandos como find, locate, whereis…

La utilidad **find** implementa un sistema de fuerza bruta para localizar archivos. Este programa localiza los archivos buscando por todo el árbol de directorios especificado, revisando los nombres de los archivos, sus fechas de creación, etc… *find* tiende a ser lento pero es más flexible y tiene una mayor probabilidad de éxito que otros programás de búsqueda, si a esto le añadimos que sabemos en que directorios buscar determinados archivos, supondrá una optimización en el tiempo de búsqueda. La sintaxis de *find* es sencilla y admite multitud de opciones que veremos más detalladamente en el apartado de comandos de este mismo tema.

Nota: El comando find no sobrepasa los permisos de archivos, por lo que si carecemos de permisos para mostrar el contenido de un directorio, find devolverá el nombre del directorio acompañado de un mensaje de error.

Otra herramienta de búsqueda es **locate**, que funciona de un modo muy similar a *find* en los casos en que queremos buscar un archivo por su nombre. La herramienta *locate* normalmente se utiliza para buscar sólo por el nombre del archivo (devuelve todos los archivos que contienen la cadena especificada). La utilidad *locate* trabaja con una base de datos que en muchas distribuciones suele ser actualizada mediante una tarea cron en la que se invoca al comando *locate* con opciones que hacen que esta se actualice. Para actualizar la base de datos de *locate* tambien podemos usar el comando **updatedb** en cualquier momento. Al trabajar con una base de datos, *locate* suele ser mucho más rápido que el comando *find* en particular las búsquedas que realizamos por todo el sistema, pero hay que tener en cuenta que si no actualizamos ciertos directorios a la hora de actualizar la base de datos, *locate* omitirá estos directorios en su búsqueda. Para utilizar *locate* basta con escribir:

```
$locate cadena-busqueda
```

Importante: Algunas distribuciones de Linux utilizan **slocate** en lugar de *locate*. El programa *slocate* incluye características de seguridad que impiden que los usuarios vean los nombres de los archivos de los directorios a los que no tienen acceso. En la mayoría de los sistemás que utilizan *slocate*, el comando *locate* es un enlace a *slocate*, por lo que *locate* implementa las características de seguridad *slocate*. Algunas distribuciones, por defecto, no instalan *locate* ni *slocate*.

Si queremos localizar ejecutables de programás y archivos relacionados, como archivos de documentación o configuración, podremos usar la herramienta **whereis**.
El programa *whereis* devuelve los nombres de archivos que comienzan por aquello que escriba como criterio de búsqueda, inconluso en archivos con extensiones. Con frecuencia esta funcionalidad halla los archivos de configuración de /etc, páginas man y archivos similares. Podemos usar *whereis* de este modo:

```
$whereis ls
```

Si por el contrario lo que queremos es conocer la ruta a un determinado comando (por ejemplo para incluir esta en un shell script que estamos programando y que necesitamos para llamar a un programa) usamos **which** que devolverá la ruta completa de la primera coincidencia que encuentre (podemos mostrar todas las coincidencias con el parámetro **-a**)

Otro comando que aunque no sea una utilidad de búsqueda, nos puede venir bien si queremos saber como interpreta Linux un determinado comando (si es nativo, es un comando externo o alias, etc…) es la herramienta **type**.

Edición básica con Vi

Primer editor de texto a pantalla completa creado para *UNIX*, diseñado para ser pequeño y sencillo. Podemos abrir el editor **Vi** de diferentes formás:

1 . Abrir un archivo para editar con *Vi* en modo "*full screen*":

```
$ vi /etc/fstab
```

2 . Recuperar la última versión guardada de un fichero. Esto es útil si no hemos cerrado correctamente Vi y se ha quedado guardado el archivo swap del fichero:

```
$ vi -r /home/nebul4ck/listas.txt
```

3 . Abrir un archivo con Vi situándonos en la última línea:

```
$ vi + /etc/samba/smb.conf
```

4 . Abrir un archivo para editar con Vi posicionando el cursor en una línea concreta (en este caso en la 24):

```
$ vi +24 /etc/apt/source.list
```

5 . Editar múltiples archivos. Iremos abriendo y guardado un archivo cada vez. Podemos pasar de un archivo a otro desde el modo comando de *Vi* con **:n** (guardamos cambios) o **:n!** si no queremos guardar:

```
$ vi archivo1 archivo2 ...archivoN
```

6 . Abrir un archivo con Vi situando el cursor en la primera ocurrencia de patrón:

```
$ vi +/patrón /etc/fstab
```

Modos de Vi

Vi podrá ser ejecutado en los siguientes modos:

- **Modo comando**: Acepta letras independientes que son aceptadas como comandos, como por ejemplo '**i**' o '**a**' que pasan a modo inserción (de manera distinta) o el caracter '**o**' que crea una nueva línea bajo la actual
- **Modo ex**: Utilizado para manipular archivos; guardar archivo y ejecutar programás externos (por ejemplo), bastará escribir '**:**' para pasar a este modo
- **Modo inserción**: Es el modo en el que vi nos permite introducir texto en el archivo. Para regresar al modo comando pulsamos '**Esc**'

Cuando accedemos a Vi por defecto nos encontramos en el modo comando. Podremos pasar al modo inserción con las siguientes teclas:

- Insertar después del carácter donde se encuentra actualmente el cursor: **a**
- Insertar antes del carácter donde se encuentra actualmente el cursor: **i**
- Añadir al final de la línea en la que se encuentra el cursor: **A**
- Añadir al principio de la línea en la que se encuentra el cursor: **I**
- Entrar en modo reemplazo de caracter: **R**
- Añadir una línea en blanco debajo de la línea en la que se encuentra el cursor y pasa a modo inserción: **o**
- Añade una línea en blanco encima de la línea en la que se encuentra el cursor y pasa al modo inserción: **O**

En el modo insercción solo podremos insertar texto y movernos con las teclas *RePág* y *AvPág*.

Opciones que acepta Vi

En el modo comando además de las teclas anteriormente vistas (con las que pasamos a modo inserción) podremos realizar otras acciones:

Opciones de movimientos

- **j** ó **RETURN**: Siguiente línea
- **k** : Línea anterior
- **h** : Carácter anterior
- **l** : Siguiente carácter
- **[[** , **H** ó **:0** : Ir al inicio del archivo
- **]]** , **L** , **G** ó **:$** : Ir al final del fichero
- **:**n : Donde '*n*' es la línea en la que nos queremos posicionar
- **:=** : Muestra la línea actual
- **Ctrl+F** : Ir a la siguiente pantalla
- **Ctrl+B** : Ir a la pantalla anterior
- **Ctrl+D** : Avanzar media pantalla
- **Ctrl+U** : Retroceder media pantalla

Operar con el archivo

- **:e** : Edita un archivo indicado si no hay cambios en el archivo actual
- **:e!** : Edita el archivo indicado, perdiendo las cambios en el archivo actual (en caso de que los hubiese)
- **:r** : Copia/añade el contenido de un archivo en el actual (después de la línea en la que se encuentra el cursor)
- **:nr** : Igual que el anterior pero lo añade a partir de la línea indicada con 'n'
- **:sh** : Ejecuta un shell sin salir de Vi. Para finalizar el shell deberemos de escribir "exit"
- **:!comando** : Ejecuta el un comando externo pasado por '*comando*'. Por ejemplo **:!ls** muestra los archivos del directorio actual.
- :**n,mw!** : Guarda solo las líneas indicadas por el rango '*n,m*'
- **:n,mw>>archivo** : Añade en '*archivo*' las líneas comprendidas por el rango '*n,m*'
- **:q** : Sale de un archivo si los datos han sido guardados o no han habido cambios

- **:q!** : Sale del archivo sin escribir cambios
- **:w** : Guardar cambios
- **:w <nombre_archivo>** : Guarda el fichero con un nombre específico
- **:wq** , **:x ó ZZ** : Guarda los cambios y sale del editor

Copiar, pegar y borrar texto dentro de Vi

Antes de mostrar los comandos debemos de concocer una característica importante. Con gran parte de los siguientes comandos podremos utilizar un dígito precediendo al comando con lo que conseguiremos una repetición del comando, por ejemplo, el comando **dd** elimina una línea, pues **2dd** borraría dos.

- **yy** : Copia la línea en la que se encuentra el cursor. Admite dígitos de repétición. Se copiaría la línea del cursor y las 'n' siguientes. **3yy** Copia la línea del cursor y las dos siguientes
- **p** : Pega el contenido del buffer (las líneas copiadas) debajo de la línea del cursor
- **P** : Igual que '*p*' pero en la línea superior a la que se encuentra el cursor
- **x** : Borra el carácter donde se encuentra el cursor. Admite *n* dígitos por lo que se borrarán '*n*' carácteres incluido el del cursor.
- **dd** : Elimina (corta) la línea en la que se encuentra el cursor. Igualmente podemos aplicar '*n*' líneas
- **dw** : Borra la palabra en la que se encuentra el cursor. Admite *n* dígitos, por lo que se borrarán '*n*' palabras incluyendo a la del cursor.
- **D** : Borra desde la posición del cursor hasta el final de la línea. Admite *n* dígitos, por lo que borraremos desde el cursor hasta el final de la línea y tantas líneas como hayamos especificado.
- **~** : Cambia el carácter de minúscula a mayúscula (virgulilla)
- **u** : Deshacer último cambio

Buscar texto y reemplazar

- **/patrón** : Busca la cadena '*patrón*' desde la posición del cursor. Podemos continuar con la búsqueda (en caso de que exista más de una coincidencia) con la letra **n**, o **N** si queremos ir a la coincidencia anterior.
- **?patrón** : Igual que el comando anterior pero invertido, es decir, busca desde la posición del cursor hacia arriba (hacia el principio del archivo)
- **:set ic** : Ejecutar esto antes de comenzar una búsqueda (por ejemplo con */patrón*) hará que no se tengan en cuenta las mayúsculas y las minúsculas.
- **:set noic** : Es igual que el anterior pero si se tendrán en cuenta las mayúsculas y minúsculas.
- **:%s/original/sustituto/** : Cambia todas las entradas que coincidan con original por sustituto. Si cambiamos **%** por un numero de línea, solo se modificarán las palabras de esa linea. Si escribimos *numero,numero* se modificaran todas las palabras de esas lineas que coincidan con original. Si anulamos '%' y cualquier número, solo modificaremos las ocurrencias de la línea en la que se encuentra el cursor.
- **:g/original/s//sustituto/**: Igualmente cambiará todas las coincidencias de '*original*' por '*sustituto*'. Sin la '**g**' solo modificaremos la primera ocurrencia.

LPIC-1 Capítulo 5

EN ESTE CAPÍTULO SE ABARCAN LOS SIGUIENTES OBJETIVOS DEL EXAMEN:

- 101.2: Proceso de arranque del sistema (3)
- 101.3: Runlevels / Target y los sistemás de inicialización: Sysv, Upstart y systemd (3)
- 102.2: Los administradores de arranque (2)

Inicializar sistemás Linux: SysV, Upstart y systemd

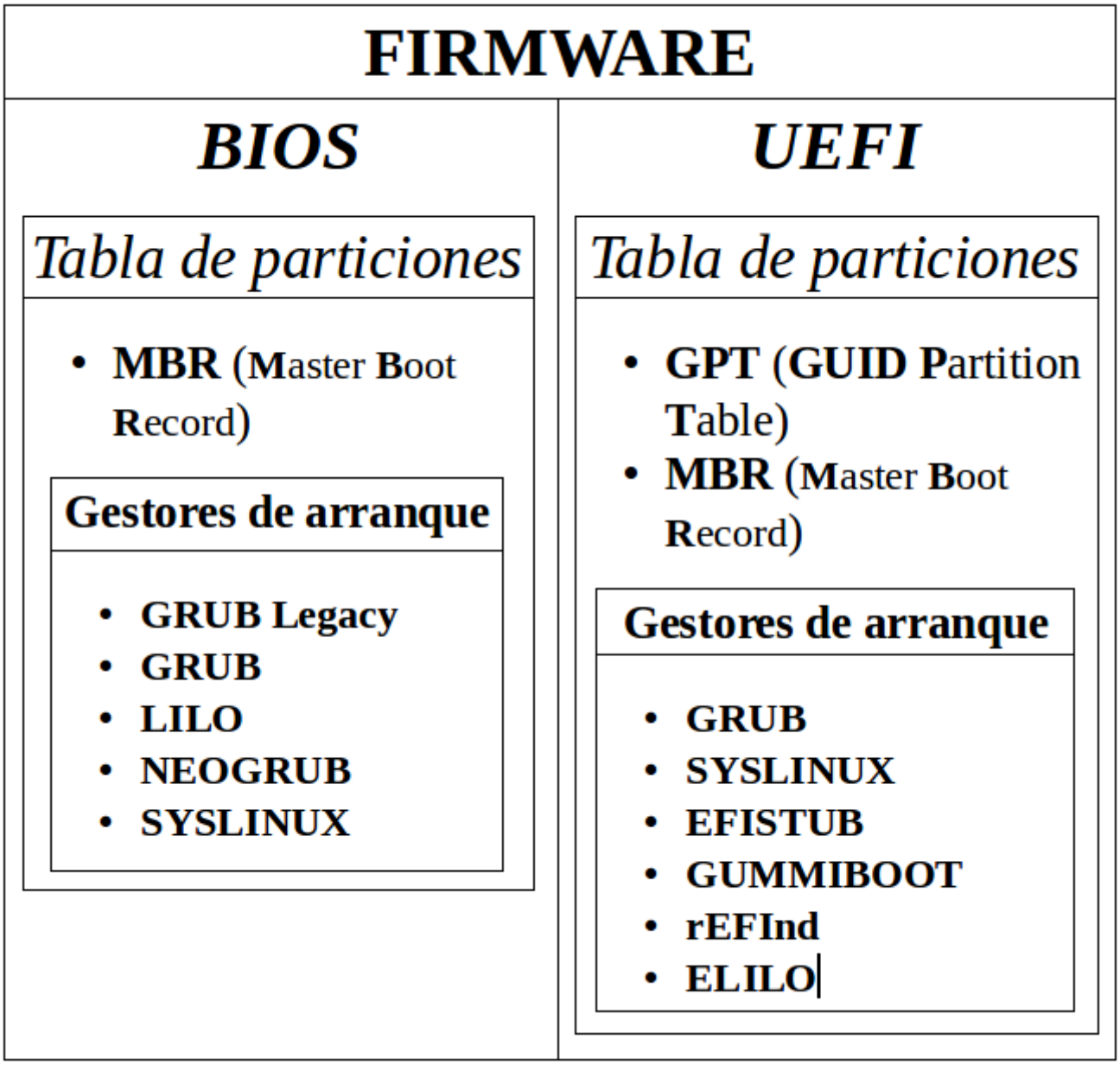
FIRMWARE: BIOS y EFI (UEFI)

Antes de hablar de los cargadores de arranque es necesario hablar de dos de los firmware más usados en equipos x86-64, estos son BIOS (Basic Input/Output System) y EFI (Extensible Firmware Interface) o su versión posterior UEFI (Unified EFI).
De una u otra forma, el firmware de su ordenador lee el cargador de arranque en memoria desde el disco duro y lo ejecuta. Por su parte, el cargador de arranque se encarga de cargar el kernel de Linux en memoria y lo empieza a ejecutar, por tanto, la configuración del cargador de arranque en el disco es clave para que el firmware pueda localizarlo y comenzar con el proceso de inicio del sistema.

Nota: Las distribuciones Linux ofrecen métodos semiautomáticos para configurar un cargador de arranque durante la instalación del sistema, aunque puede que necesite saber más, en especial si recompila su kernel o tiene que configurar opciones avanzadas, por ejemplo para seleccionar diferentes OS.

A la secuencia completa de arranque de un sistema (es decir la delegación de BIOS en el cargador de arranque y de este en el kernel del OS) se le conoce como *Bootstrapping* o *Bootstrap Loader*.

Tabla de los 2 principales firmware y sus cargadores de arranque:



Nota: Se puede usar una tabla de particiones **GPT** en sistemás con firmware **BIOS**, en cuyo caso *GRUB* se almacena parcialmente en una partición especial conocida como partición de arranque *BIOS*.

BIOS

BIOS (Basic Input/Output System): Firmware instalado en memoria ROM (EEPROM) o memoria Flash de la placa base de un ordenador, encargado de realizar una serie de pruebas e inicios (Power-On Self-Test,POST) como la cantidad de RAM presente, los dispositivos PCI, periféricos, etc…

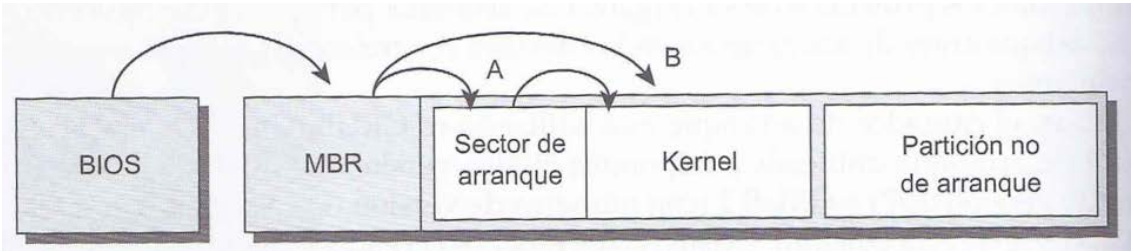
Existen dos tipos de arranque, el arranque en frio (el ordenador se encuentra apagado) y el arranque en caliente (tenemos el ordenador encendido). La diferencia es que si arrancamos en caliente (Ctrl+Alt+Spr, botón de reset/reinicio) la BIOS salta la tarea de POST, por el contrario el arranque en frio hace que la BIOS siga su curso de chequeo e inicialización.

Una vez realizada la tarea POST, si todo ha ido bien sonará un "bip" (en ordenadores más antiguos, indicando que todo está OK) y la BIOS leerá desde el **MBR** (sector de arranque especial situado en los primeros *512 bytes* del primer disco duro) la tabla de particiones.

Podemos indicar mediante el menú de *BIOS* que el disco de arranque sea una unidad USB, un disco flexible, una unidad de CD-ROM, uno u otro disco rígido, etc…

El *MBR* contiene la tabla de partición (16 bytes por cada partición, con un máximo de cuatro), un centinela (0xAA55) y el código ensamblador (Máster Boot Code, de 446 bytes), leído por la BIOS para continuar con el proceso de arranque. Este código (cargador de arranque principal) busca en la tabla de particiones aquella partición que está marcada como activa, lee de su sector de arranque y ejecuta el cargador de arranque "secundario" para que continué con el inicio del sistema (fase 1). Dependiendo del tamaño del cargador de arranque, se instalará en el mismo *MBR* o bien en una partición de arranque (/boot), que será la partición activa. De estar instalado en el *MBR* se pasará a fase 2, fase en la que se delega la carga del OS al cargador de arranque principal, finalizando así el inicio mediante BIOS, por el contrario de existir un cargador de arranque "secundario" se delegará a este el arranque (fase intermedia o fase 1.5).

Nota: Tener el cargador de arranque en el *MBR* tiene varias desventajas por ejemplo que el tamaño es limitado, o que si instala un OS Windows después de un OS Linux, este escribirá en el *MBR* por lo que machacará el cargador de arranque Linux.



Si es necesario una fase intermedia o fase 1.5, es porque GRUB está dividido en 2 partes; por un lado **boot.img**, imagen que se instala en el *MBR* para aprovechar los servicios de la *BIOS* y que será la encargada de cargar una nueva imagen con más funcionalidades (**core.img**) que se encuentra instalada en */boot*.
BIOS no tiene soporte para leer filesystems por lo que *boot.img* necesitará saber en que sectores se encuentra la partición activa para poder leer *core.img*, junto a los módulos y archivos de configuración de *GRUB*. Con esto se permite la lectura de cilindros por encima del 1024 y unidades de tipo **LBA**. En fase 2 se mostrará el menú de *GRUB* en el

que se nos permitirá iniciar desde uno u otro sistema. Ya es *GRUB* el encargado de localizar la partición en la que se encuentra dicho sistema.

Nota: Si tenemos un esquema como este, es decir *boot.img* en *MBR* y el resto del contenido de *GRUB* en una partición por ejemplo /boot, puede ocurrir que al instalar Windows después de Linux, este escriba en el *MBR* e invalide el arranque de Linux. Podremos solucionar esto con **FDISK** de *DOS*, marcando como activa la partición /boot, para que en el próximo arranque se nos muestre el menú de selección de los diferentes OS.

Existen distintos cargadores de arranque compatibles con *BIOS*. De los cargadores de arranque hablaremos una vez tengamos entendido el funcionamiento de los principales firmware, así como la compatibilidad y uso de las tabla de particiones que hacen cada uno de ellos. Algunos de los cargadores de arranque compatibles con *BIOS* son:

- GRUB Legacy
- GRUB
- LILO
- NeoGRUB
- SYSLINUX

Particiones de tipo MBR:

El sistema de particiones *MBR* está limitado a particiones de 2 tebibytes (1 TiB = 2^40 bytes), al menos si se utiliza el tamaño de sector universal de 512 bytes. *MBR* utiliza tres tipos de particiones: **particiones primarias**, **partición extendida** (solo puede haber una y sirve de contenedor para las particiones lógicas) y **particiones lógicas** (estas se incluyen dentro de la partición extendida). No existe diferencia de rendimiento entre particiones primarias y particiones lógicas.

Nota: Sistemás como *DOS*, *Windows* y *FreeBSD* necesitan arrancar desde una partición primaria. Linux, aunque siendo una práctica inusual, puede arrancar desde particiones lógicas.

Para las particiones principales (primarias y extendidas) la numeración va desde sda1 a sda4, mientras que para las lógicas se usa del 5 en adelante. Pueden existir huecos entre las particiones primarias pero no en las lógicas, es decir, puede existir sda1, sda3, sda5, sda6 y sda7, pero no sda1, sda4, sda5, sda8 (si existe sda8, tiene que existir sda6 y sda7

Truco: Podemos crear copias de seguridad de nuestra tabla de particiones, de una forma sencilla (aunque existes otros métodos, de momentos vamos a quedarnos con este:

 # sfdisk -d /dev/sda > /ruta/donde/guardar/sda-backup.txt

Podemos guardar esta copia en un USB y a continuación restaurarlo de la siguiente manera:

 # sfdisk -f /dev/sda < /ruta/del/backup/sda-backup.txt

Las particiones *MBR* usan números hexadecimales de dos dígitos, de uno o dos bytes, para identificarlas. Alguno de los códigos más habituales son:

- FAT: 0x0c
- NTFS: 0x07
- Sistema de archivos Linux: 0x83
- Intercambio de Linux: 0x82
- Partición extendida: 0x0f

El *MBR* está siendo reemplazado por la **GUID Partition Table** (*GPT*), ya que esta no puede superar los discos de 2TiB

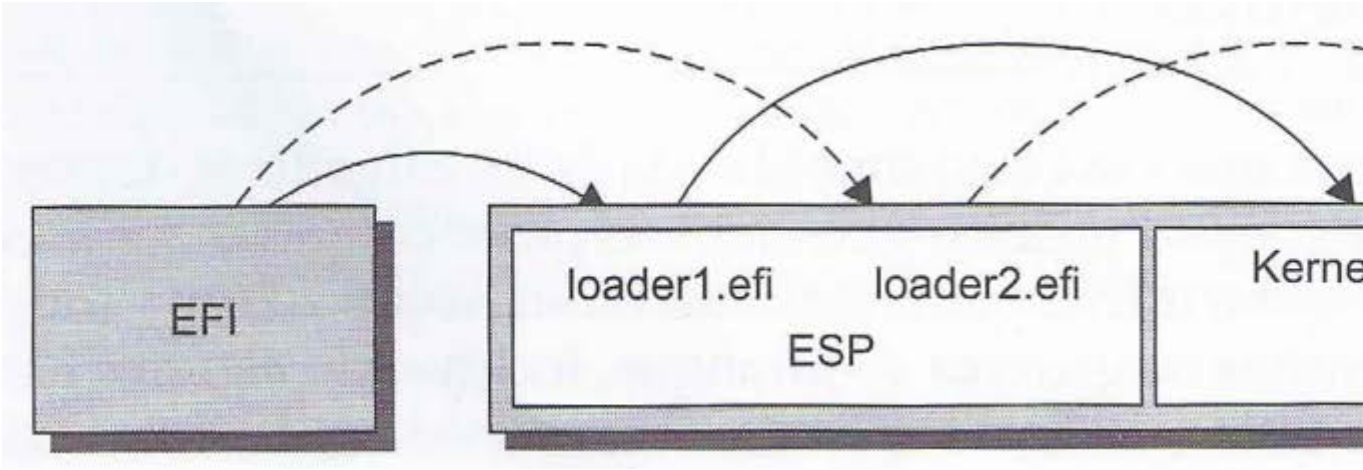
EFI

EFI (Extensible Firmware Interface, propiedad de Intel) o **UEFI** (Unified Extensible Firmware Interface, Alianza de varias compañías líderes en tecnología: AMD, HP, IBM, Intel, Dell, Apple…)

Desarrollada por *Intel* para reemplazar al *MBR* de IBM. *UEFI* es la versión mejorada de *EFI*. Algunas mejoras son el cifrado, la autenticación de red, la interfaz de usuario,etc…

EFI hereda las nuevas características avanzadas de *BIOS* como ACPI (*Interfaz Avanzada de Configuración y Energía*) y el SMBIOS (*Sistema de Gestión de BIOS*) además, cabe destacar la capacidad de arranque desde unidades de almacenamiento grandes, arquitectura y controladores de la CPU independientes, diseño modular, etc. EFI permite el arranque tanto desde el *MBR* como desde *GPT*, el cual solventa las limitaciones técnicas de *MBR*. EFI posee su propio gestor de arranque minimalista que permite también la selección y carga directa del OS, eliminando la necesidad de recurrir a gestores de arranque externos.

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



EFI debe de conocer los cargadores de arranque instalados en el *ESP*, para ello se suelen registrar los cargadores en el firmware, bien mediante una interfaz de usuario del propio firmware o con una herramienta como el programa **efibootmgr** de Linux.

Nota: Muchas implementaciones *EFI* de x86-64 usan el cargador de arranque *EFI/boot/bootx64.efi* del *ESP* como predeterminado si no hay otros registrados.

Resumen sobre firmware: Para arrancar un sistema, la *BIOS* accede a un dispositivo configurado para ser el primero en detectar y lee el primer sector, en dicho sector se encuentra el *MBR* el cual contiene la tabla de particiones y el gestor de arranque, que será el encargado de continuar el arranque del OS. Si el administrador de arranque tiene un tamaño considerable, puede que este esté instalado en una partición primaria marcada como activa (por ejemplo /boot) y en cuyo sector de arranque se encuentra este administrador, quien cargará el kernel del sistema. En cambio, *EFI* lee un archivo de cargador de arranque que se encuentra en una partición especial del sistema de archivos *ESP* (EFI System Partition, Sistema de partición EFI) formateada normalmente con FAT. *EFI* ofrece una amplia variedad de opciones de arranque como arrancar desde distintos dispositivos a arrancar desde determinados sectores de arranque de otros disco del equipo. Los equipos con *BIOS* permiten un máximo de 4 particiones primarias (o 3 primarias y una extendida en la que crear más particiones lógicas) y *EFI* nos permite hasta un máximo de 128 particiones (x86_64) todas primarias.

CARGADORES DE ARRANQUE

Bootloader (*cargador de arranque*): El gestor de arranque o cargador de arranque es el software inicializado por la BIOS o UEFI para llevar a cabo la carga del kernel con los parámetros apropiados y la imagen RAM inicial (**initramfs**), antes de iniciar el proceso de arranque del OS. Se pueden utilizar diferentes gestores de arranque como GRUB o **Syslinux**. Algunos gestores de arranque solo admiten arrancar desde la BIOS o UEFI y otros desde ambos.

Los cargadores de arranque EFI más populares para Linux se basan en cargadores de arranque BIOS, por lo que ofrecen funciones que no suelen necesitar, como por ejemplo, permitir la concatenación con otro cargador de arranque.

Gestores de arranque que soportan UEFI y BIOS

- **GRUB** (GRand Unified Bootloader): Derivado (*fork*) de **PUPA** (*proyecto de investigación destinado a mejorar GRUB Legacy*). *GRUB* necesita crear una partición /boot separada para instalarse, a menos que se utilice el sistema de archivos **Btrfs** (compatible con los algoritmos de compresión *zlib* o *LZO*) para la partición root. *GRUB* es ahora modular y ya no requiere de la fase 1.5 o fase intermedia, en consecuencia los módulos para soporte de **LVM** o **RAID** deben ser cargados desde el disco duro. Los discos duros pasan de ser */dev/sda1* (GRUB Legacy) a *hd0,msdos1* (disco 1 partición 1 para sistemás MBR) o *hd0,gpt1* (para los sistemás que usan GPT), en *GRUB.*
- **SYSLINUX**: Es una colección de cargadores de arranque, capaz de iniciar desde discos duros, CDs o Red (utilizando PXE). Veremos su instalación en artículos destinados a tal fin en este mismo blog (https://nebul4ck.wordpress.com). De entre todos los cargadores de arranque incluidos en *SYSLINUX* quizás el más importante sea **ISOLINUX**, un cargador de arranque para los discos ópticos.

Nota: *Syslinux* no es capaz de acceder a los archivos contenidos en una partición diferente de aquella en la cual se ha instalado. Tal característica (llamada multi-fs) no está aún implementada. Para un gestor de arranque alternativo con características multi-fs use *GRUB.*

Gestores de arranque para BIOS

- **GRUB Legacy** (GRUB de legado): Es el cargador por defecto de las distribuciones de Linux más antiguas. Este cargador de arranque quedó eclipsado por su sucesor GRUB. Veremos como configurar *GRUB Legacy* en el apartado de "*Configurar cargadores de arranque*"
- **LILO**: Es un gestor de arranque más sencillo, por lo que ofrece menos posibilidades a la hora de arrancar distintos OS. *LILO* cayó prácticamente en desuso con la llegada de *GRUB.*
- **NeoGRUB**: Es una implementación de *GRUB2DOS* proporcionado por el configurador de cargador de arranque *EasyBCD* para Windows. *NeoGRUB* permite arrancar Linux desde el cargador de arranque Windows, que para hacer esto posible incorpora una implementación de *GRUB Legacy* dentro del propio cargador de arranque Windows. *NeoGRUB* soporta los siguientes sistemás de archivos: FAT16/32, MINIXfs, ext2fs, Reiserfs, JFS y XFS.

Gestores de arranque para UEFI

- **EFISTUB**: A partir de la versión del kernel 3.3 Linux permite arrancar con *EFISTUB*. Esta característica permite que el firmware *EFI* cargue el kernel

como un ejecutable EFI. Digamos que es un gestor de arranque minimalista que permite arrancar desde el propio firmware EFI.

- **rEFIt**: Es un administrador de arranque (no un cargador), se puede usar en PC basados en *UEFI*. Dispone de una atractiva interfaz gráfica. Actualmente es un administrador de arranque obsoleto
- **rEFInd**: Derivado de *rEFIt* para PCs basados en *UEFI* y para ampliar sus prestaciones. Al igual que *rEFIt* es un administrador de arranque con una bonita interfaz, útil para equipos con *EFI* que ofrecen administradores de arranque incompletos. Puede utilizarse junto con *EFISTUB* para ejecutar las imágenes mediante su menú.
- **Gummiboot**: Similar a *rEFInd* solo que su interfaz es en modo texto y cuenta con menos opciones.
- **ELILO**: Se utiliza en algunas distribuciones (en especial en *OpenSUSE*) como cargador de arranque predeterminado para equipos basados en *EFI*

Referencias:

Firmware:
http://es.wikipedia.org/wiki/BIOS
https://wiki.archlinux.org/index.php/Unified_Extensible_Firmware_Interface_%28Espa%C3%B1ol%29

Tabla de partición:
https://wiki.archlinux.org/index.php/Máster_Boot_Record_%28Espa%C3%B1ol%29
https://wiki.archlinux.org/index.php/GPT

Cargador de arranque:
https://wiki.archlinux.org/index.php/Boot_loaders_%28Espa%C3%B1ol%29
https://wiki.archlinux.org/index.php/Syslinux_%28Espa%C3%B1ol%29
https://wiki.archlinux.org/index.php/GRUB_%28Espa%C3%B1ol%29
https://wiki.archlinux.org/index.php/UEFI_Bootloaders

Sobre Secure Boot: Con *Secure Boot* un equipo basado en *EFI* solo iniciará un cargador de arranque si se ha firmado criptográficamente con una clave cuyo duplicado se almacena en el firmware del equipo, evitando así que programás malintencionados accedan al ordenador al inicio del arranque.
Desde Linux el problema es que el uso de *Secure Boot* requiere la firma de un cargador de arranque de Linux con la clave de *Microsoft* (ya que es la única cuya presencia se garantiza en la mayoría de los equipos). En la práctica puede que tengamos que desactivar *Secure Boot* o generar su propia clave para arrancar una distribución arbitraria de Linux o un kernel personalizado.

CONFIGURAR LOS CARGADORES DE ARRANQUE.

Vamos a estudiar de una forma “breve” como configurar los principales cargadores de arranque *GRUB Legacy*, *GRUB*, *SYSLINUX* y *UEFI* con la herramienta ***efibootmgr***.

Nota: Solo *GRUB Legacy* y *GRUB* son abarcados por el examen, será por ello por los que realicemos una descripción más exhaustiva de estos cargadores de arranque, aun así, podrás encontrar aquí una entrada en este mismo blog sobre un uso eficiente de *syslinux* a la hora de instalar un sistema **Fedora**, del cual tenemos *vmlinuz* e *initrd* y arrancando desde una unidad externa. Sobre *efibootmgr* daremos una breve explicación de su uso.

Configurar GRUB Legacy.

La ubicación normal para el archivo de configuración de GRUB Legacy en un equipo basado en BIOS es */boot/grub/menu.lst* (o *grub.conf* para algunas distribuciones como RedHat, Fedora o Gentoo). GRUB utiliza (hd0) o (hd0,0) por ejemplo para señalar la primera partición del primer disco en lugar de los archivos de dispositivos Linux (/dev/hda o /dev/sda), además no distingue entre dispositivos PATA, SATA, SCSI y USB, por lo que en un sistema que solo tiene SCSI, la primera unidad de disco será igualmente (hd0).

Nota: GRUB Legacy no arrancará desde una unidad USB si utiliza un ordenador antiguo que no admita unidades USB como discos duros.

Las asignaciones de unidades de GRUB Legacy se pueden encontrar en el archivo */boot/grub/device.map*

GRUB Legacy interpretará como partición raíz aquella en la que se encuentre su archivo de configuración, por lo que si existe una partición independiente (/boot) para GRUB será está la partición que tome como partición raíz y no /, algo que deberemos de tener en mente a la hora de referenciar el kernel y la RAM inicial.

```
# grub.conf/menu.lst
#
# Opciones globales:
#
default=0
timeout=15
splashimage=/grub/bootimage.xpm.gz
#
# Opciones de imagen del kernel:
#
title Fedora (3.4.1)
    root (hd0,0)
    kernel /vmlinuz-3.4.1 ro root=/dev/sda5 me
    initrd /initrd-3.4.1
title Debian (3.4.2-experimental)
    root (hd0,0)
    kernel (hd0,0)/bzImage-3.4.2-experimental
#
# Otros sistemas operativos
#
title Windows
    rootnoverify (hd0,1)
    chainloader +1
```

Estudiaremos las opciones (globales y de imágenes) de GRUB Legacy mediante la imagen anterior.

- La opción **default=0** indica que arranque por defecto (si no pulsamos ninguna tecla antes del timeout) el primer OS de la lista, en este caso Fedora (3.4.1)

- **Timeout=15** indica que tenemos 15 segundos para seleccionar un OS diferente al de por defecto.
- La línea **splashimage=/grub/bootimage.xpm.gz** hace que aparezca una imagen de fondo en el menú de selección. Como dijimos GRUB Legacy toma como directorio raíz aquel en el que se encuentra su archivo de configuración que en este caso es /boot por lo que la imagen realmente se encuentra en */boot/grub/bootimage.xpm.gz*

Nota: Si la imagen que deseamos se encuentra en otra partición podremos especificarla con (hd0,2) si por ejemplo se encontrase en la 3º partición del primer disco.

- **title Fedora (3.4.1)** hará que en el menú de selección aparezca un OS con este nombre (Fedora …) en nuestro caso, además será el primero que aparezca del menú ya que se encuentra en primera posición.
- **root (hd0,0)** como imaginaremos, referencia a la primera partición del primer disco, recordemos que para nosotros es /boot (ya que es una partición independiente marcada como activa en la tabla de particiones)
- El parámetro **kernel** describe la ubicación del kernel de Linux a cargar, y sus respectivos parámetros. Una vez más, la ruta es relativa a su partición primaria, es decir el kernel se encuentra dentro de /boot/

Nota: Si tuviésemos el kernel en otra partición o incluso en otro disco podríamos especificarlo así: *kernel (hd0,4) /vmlinuz…*

- Utilice la opción **initrd** para especificar un disco RAM inicial que contiene un conjunto mínimo de controladores, herramientas y archivos de configuración, empleados por el kernel en un punto del proceso de arranque para poder montar su sistema de archivos raíz antes de que pueda acceder al disco duro. Además esto disminuye considerablemente el tamaño del archivo kernel principal.
- Con la opción **rootnoverify** GRUB Legacy no intentará acceder a los archivos de esta partición debido a que no puede cargar directamente por ejemplo, un kernel como el de DOS o Windows.
- Por ultimo la opción de cargar en cadena: **chainloader** indica a GRUB que pase el control a otro cargador de arranque. Normalmente se pasa la opción **+1** para cargar el primer sector de la partición del SO de destino, que para este ejemplo es (hd0,1). La carga en cadena funciona en equipos BIOS. Una versión de GRUB Legacy compatible con EFI, puede realizar la carga en cadena, pero debe indicar a GRUB que use el ESP, indicando la partición como siempre (hd0,0) y después pasar el nombre de un archivo de cargador de arranque EFI a través de la opción *chainloader*, como *chainloader /EFI/Microsoft/boot/bootmgfw.efi*

Nota: Las opciones de las imágenes de GRUB Legacy suelen presentar sangrías tras la primera línea, pero esto es una convención y no un requisito del formato del archivo.

Para instalar GRUB Legacy usamos el comando **grub-install** acompañado del nombre de dispositivo en el que se encuentra el sector de arranque:

- **# grub-install /dev/sda** o **# grub-install '(hd0)'**

Nota: Ambos comandos instalarán el cargador de arranque en el primer sector del disco 1 (en el MBR). Si quisiéramos instalarlo en el sector de arranque de una partición concreta podríamos indicarlo de igual modo …./sda2 o '(hd0,1)'

En Fedora: Para instalar GRUB Legacy compatible con EFI copie el archivo **grub.efi** y **grub.conf** en */boot/efi/EFI/redhat*. Si realiza una instalación con **rpm grub-efi** el archivo .efi se incluirá en esta ubicación de forma predeterminada. Puede que tenga que usar **efibootmgr** para añadir el cargador de arranque a la lista de EFI de la siguiente manera:

- **# efibootmgr -c -l \\EFI\redhat\grub.efi -L GRUB**

Nota: UEFI utiliza la barra invertida `\` como separador de ruta (similar a las rutas de Windows), pero el paquete oficial **efibootmgr** proporciona soporte a las rutas de estilo unix con barras diagonales `/` como separador de ruta para la opción `-l`. *Efibootmgr* convierte internamente `/` a `\` antes de codificar la ruta de acceso del cargador. Hubiese valido igualmente */EFI/redhat/grub/.efi -L GRUB*

Utilizar GRUB como cargador de arranque

GRUB está diseñado para funcionar en equipos basados en BIOS y EFI.

La configuración de GRUB2 es muy similar a la de GRUB Legacy, aunque difieren en varios aspectos como el nombre o ruta del archivo de configuración, la compatibilidad con módulos de carga (el comando **insmod** del archivo de configuración de GRUB2 permite cargar módulos), la admisión de instrucciones lógicas en la configuración, la forma en la que manipulamos el archivo de configuración, etc…

El archivo de configuración pasa a llamarse **grub.cfg** (y no grub.conf o menu.lst) y la ruta puede contener un 2 en grub (*/boot/grub2/..*) lo que permite instalar los dos cargadores de arranque (GRUB Legacy y GRUB2 o GRUB).

Un aspecto importante a tener cuenta es el cambio de nombre de algunos de sus parámetros en el archivo de configuración y la forma en la que manipulamos a este.

Por ejemplo la palabra **title** pasa a llamarse **menuentry**, el título de menú se incluye entre comillas, además tras este aparece una llave (**{**) y cada entrada termina con una llave de cierre (**}**), **set** precede a la palabra clave root y se usa el signo (=) para separar a root de la partición, la palabra clave **rootnoverify** se ha eliminado y en su lugar se utiliza **root**, las particiones (no lo discos) se empiezan numerando desde 1 y no desde 0, y destacar que las versiones más recientes de GRUB también admiten un sistema de identificación de particiones para especificar el tipo de tabla de partición como (**hd0,gpt2**) para indicar la 2º partición del primer disco de una tabla de particiones GPT o (**hd0,mbr3**) para referenciar a la 3º partición del primer disco de una tabla MBR.

Como comentábamos anteriormente la forma en la que se manipula ahora el archivo de configuración de GRUB difiere de GRUB Legacy y es debido a que GRUB utiliza una serie de scripts para el mantenimiento automático del archivo de configuración. Estos scripts buscan en el sistema versiones de kernel y añaden entradas a GRUB. El objetivo de esto es que los administradores del sistema nunca tengan que editarlo de forma explícita. En su lugar se editan los archivos de **/etc/grub.d** y el archivo **/etc/default/grub** que controla los elementos predeterminados creados por los scripts de configuración. Por ejemplo si desea ajustar el tiempo de espera, podría cambiar la siguiente línea:

GRUB_TIMEOUT=10

y tras realizar los cambios, deberemos de volver a generar el archivo *grub.cfg* con los comandos **update-grub**, **grub-mkconfig > /boot/grub/grub.cfg** o **grub-mkconfig -o /boot/grub/grub.cfg**, como ya comentamos algunas instalaciones usan el 2 tras la palabra grub, habrá que tener esto presente.

Al instalar Linux por primera vez, el instalador debe configurar GRUB correctamente y usar *grub-install* de la misma forma que GRUB Legacy.

La recuperación de GRUB en caso de sufir daños, la dejaremos para artículos independientes de este mismo blog, pero como nota, mencionaremos la herramienta **supergrubdisk** (http://supergrubdisk.org), que es una imagen de disco de arranque con varias opciones para localizar y usar el archivo de configuración de GRUB en su disco duro.

Configurar UEFI con efibootmgr

Como comentamos anteriormente UEFI posee un gestor de arranque minimalista mediante el cual podremos añadir entradas de arranque en la placa base (siempre y cuando esta y sus implementaciones no lo permitan) usando el comando **efibootmgr**.

efibootmgr requiere que el kernel tenga soporte para acceder a las variables no volátiles de EFI (*/proc/efi/vars* en kernels 2.4 y */sys/firmware/efi/vars* en kernels 2.6). Podemos ayudarnos de **lsmod |grep efivars** y **modprobe** para comprobar el soporte.

Primero vamos a comprobar que el soporte para las variables de UEFI en el kernel funcionan correctamente:

```
# efivar -l
```

Si todo ha ido OK y *efivar* nos ha enumerado las variables, creamos la entrada de arranque:

```
# efibootmgr -c -d /dev/sdX -p Y -l /EFI/refind/refind_x64.efi -L
"label_de_entrada"
```

Podemos verificar la entrada con:

```
# efibootmgr -v
```

Nos devolverá un número por cada entrada. Utilice estos números para ordenar las entradas con el comando:

```
# efibootmgr -o <num_entrada>,<otra_entrada>,<etc...>
```

Advertencia: Algunas combinaciones de kernel y efibootmgr podrían negarse a crear nuevas entradas de inicio. Esto podría ser debido a la falta de espacio libre en la memoria NVRAM. Podría tratar de eliminar cualquier archivo de volcado de EFI:

```
# rm /sys/firmware/efi/efivars/dump-*
```

Podemos encontrar documentación extra, en caso de tener problemás con los requisitos del kernel por ejemplo en:

http://wiki.archlinux.org/index.php/Unified_Extensible_Firmware_Interface_%28Espa%C3%B1ol%29#efibootmgr

http://wiki.archlinux.org/index.php/Unified_Extensible_Firmware_Interface_%28Espa%C3%B1ol%29#Requisitos_del_soporte_de_las_variables_UEFI_para_que_funcione_correctamente

PROCESO DE ARRANQUE

Los pasos que sigue un ordenador para arrancar un sistema operativo son los siguientes:

1. Se enciende el sistema y un circuito de hardware especial hace que la CPU busque y ejecute el firmware (BIOS o UEFI).

2. El firmware realiza tareas de comprobación del hardware, de configuración (POST) y por último si todo va bien, busca el cargador de arranque.

3. Cuando el control pasa del firmware al cargador de arranque carga un kernel o enlaza la carga de otro cargador de arranque.

kernel: Es el núcleo de un sistema operativo. Funciona en un nivel bajo (*kernelspace*) que interactúa entre el hardware de la máquina y los programás que utilizan los recursos del hardware para funcionar. Para hacer un uso eficiente de la CPU, el kernel utiliza un sistema de programación para arbitrar qué tareas tienen prioridad en un momento dado, creando la ilusión (para la percepción humana) de que varias tareas se están ejecutando simultáneamente.

4. Cuando el kernel de Linux toma el control, este descomprime la imagen **initramfs** (sistema de archivos RAM inicial), que se convierte en el sistema de ficheros root inicial, y comienza la carga de dispositivos, monta la partición raíz y por último carga y ejecuta el programa inicial de su sistema (por defecto /etc/init).

initram o RAM inicial: El propósito de *initramfs* es arrancar el sistema hasta el punto donde se puede acceder al sistema de archivos raíz, esto significa que los módulos que se requieren para dispositivos como IDE, SCSI, SATA, USB, etc… deben ser cargados desde *initramfs* si no están compilados en el kernel. Una vez que los módulos necesarios se cargan (ya sea de forma explícita a través de un programa o script, o implícitamente a través de **udev**), el proceso de arranque continúa.

Nota: initramfs sólo debe contener los módulos necesarios para acceder al sistema de archivos raíz, no tiene por qué contener todos los módulos que sirven al completo funcionamiento de la máquina. La mayoría de los módulos se cargarán más tarde por udev, durante la fase init.

5. El programa inicial recibe el ID de proceso 1 (PID1) que es el primer programa que se ejecuta en el sistema de arranque tradicional de Linux. **/sbin/init** lee el archivo **/etc/inittab** para determinar que otros programás ejecutar. En sistemás que utilicen Upstart o systemd, /sbin/init lee otros archivos de configuración e incluso su nombre se ve reemplazado como es el caso para systemd donde init pasa a ser systemd

init lanza las **getty**, una para cada terminal virtual (normalmente seis de ellas), las cuales inicializan cada tty pidiendo el nombre de usuario y contraseña. Una vez que se proporcionan el nombre de usuario y la contraseña, getty los compara con las que se encuentran en */etc/passwd*, llamando a continuación al programa **login**, para que, una vez iniciada la sesión de usuario, lance la **shell** de usuario de acuerdo con lo definido en /etc/passwd. Alternativamente, getty puede iniciar directamente un gestor de pantallas si existe uno presente en el sistema, de haberlo este se ejecutará en la tty configurada, reemplazando el prompt de inicio de la getty.

Proceso de arranque:
http://es.wikipedia.org/wiki/Registro_de_arranque_principal
http://es.wikipedia.org/wiki/Proceso_de_arranque_en_Linux
http://es.wikipedia.org/wiki/Arranque_%28inform%C3%A1tica%29

Una vez que el kernel es cargado:
https://wiki.archlinux.org/index.php/Arch_boot_process_%28Espa%C3%B1ol%29#Tipos_de_firmware

Initramfs:
https://wiki.archlinux.org/index.php/Mkinitcpio_%28Espa%C3%B1ol%29

SISTEMA DE INICIALIZACIÓN SYSTEM V (SysV)

Linux se basa en modos de ejecución para determinar que funciones hay disponibles, enumerados del 0 al 6. Cada uno de ellos tiene asignado un conjunto de servicios. Los modos de ejecución 0,1 y 6 están reservados para fines especiales; los modos de ejecución restantes están disponible para los objetivos que desee o los que decidan los proveedores de las distintas distribuciones Linux.

Nota: Puede haber modos de ejecuciones fuera del rango 0-6, pero es algo poco habitual.

Es importante saber cuales son estas funciones y cómo gestionar los modos de ejecución para poder controlar el proceso de arranque de Linux y las operaciones en curso. Para ello se debe conocer la finalidad de cada modo de ejecución, identificar los servicios activos en cada uno de ellos, gestionarlos, revisar sus modos de ejecución actuales y poder cambiarlos si fuese necesario.

Funciones de los modos de ejecución

El archivo **/etc/inittab** define y describe los distintos modos de ejecución.

- **0** : Se emplea para apagar el sistema. Es un modo transitorio, es decir se emplea para pasar de un estado a otro. En este caso de iniciado a cerrado. En equipos con hardware moderno el sistema se apaga por completo, no así en equipos más antiguos donde tiene que intervenir el usuario.
- **1** : Modo **monousuario**, también representado por **s**, **S** o **single**. Se suele emplear para realizar un mantenimiento a bajo nivel, como por ejemplo el redimensionamiento de particiones, configuración del servidor X, etc…
- **2** : En **Debian** y sus derivados es un modo **multiusuario** completo con X en ejecución. Muchas otras distribuciones dejan este modo de ejecución sin definir.
- **3** : Para una gran mayoría de distribuciones como **Red Hat** o sus derivados, se define este modo como un modo **multiusuario** mediante **terminal**
- **4** : Disponible para configuraciones personalizadas
- **5** : En Fedora, Mandriva, Red Hat y la mayoría de las demás distribuciones tienen el mismo comportamiento que el modo 3 pero con servidor X.
- **6** : Se emplea para **reiniciar el sistema**. Al igual que el modo 0, este es un modo transitorio, donde el ordenador se apaga por completo y vuelve a iniciar.

Si configura su sistema con los modos 0 y 6 su sistema se apagará o reiniciará en cuanto termine el proceso de arranque. En la mayoría de distribuciones, lo normal es tener el sistema configurado para que arranque en el modo de ejecución 2,3 o 5.

Nota: Las distribuciones que usan sistemás de inicio más moderno (Upstart o systemd) no suelen utilizar modos de ejecución de forma nativa, pero ofrecen herramientras de compatibilidad que hacen que parezca que el ordenador los usa.

Identificar los servicios de un modo de ejecución.

Existen varias maneras de identificar que programás se ejecutan cuando se pasa a un modo de ejecución concreto con **SysV**, como por ejemplo a través del archivo */etc/inittab* o bien con la herramienta **chkconfig**. Con la herramienta **update-rc.d** podremos activar y desactivar servicios para runlevels específicos. Tenemos detallada estas dos herramientas en el apartado de comandos de este mismo capítulo.

Nota: Para identificar programás que son ejecutados en un determinado modo de ejecución, existe una forma más rudimentaria que es localizar el directorio del modo de ejecución concreto y listar su contenido. En el listado aparecerán los enlaces simbólicos hacia scripts principales de los programás que serán ejecutados en ese modo. Esto lo veremos más adelante cuando estudiemos los scripts de inicio SysV

Para identificar o definir programás para un cierto modo de ejecución mediante el archivo */etc/inittab*, antes tendremos que conocer su estructura.

Las entradas de /etc/inittab siguen un formato sencillo:

id:modosdeejecución:acción:proceso

Donde:

- **id** es una secuencia de uno a cuatro caracteres que identifica su función
- **modosdeejecución** es una lista de los modos de ejecución para los que se aplicará la entrada. Puede definirse un solo modo (por ejemplo 3) o varios de ellos (2,3,5)
- **acción** indica a init cómo tratar el proceso. Por ejemplo: ***wait*** inicia el proceso una vez cuando pase a un modo de ejecución y espera a que este finalice, ***respawn*** reinicia el proceso cada vez que finalice, ***powerwait*** indica a *init* que apague el sistema antes una falla en el sistema eléctrico (se precisa un UPS), ***ctraltdel*** reinicia el sistema cuando el usuario presione esas teclas (podemos cambiar las funciones o anularlas), ***sysinit*** comando ejecutado cuando inicia el sistema (por ejemplo limpiar /tmp), etc… podremos conocer más acciones a través de *man inittab*. Para la acción **initdefault** se ignora el campo proceso.
- **proceso** es el proceso que ejecutar para una determinada entrada, incluyendo las opciones y argumentos que sean necesarios.

Una vez entendida la estructura del archivo */etc/inittab* podremos saber como introducir nuevas entradas para nuevos procesos o eliminar en caso de que no queramos que un programa sea ejecutado en un determinado modo.

Antes de comentar la segunda forma de identificar programás ejecutados en un determinado modo de ejecución vamos hablar sobre los scripts de inicio Sysv.

El script **/etc/init.d/rc** o **/etc/rc.d/rc** realiza la tarea crucial de ejecutar todos los scripts asociados con el modo de ejecución. Estos scripts se encuentran almacenados en */etc/rc.d/rc?.d*, */etc/init.d/rc?.d*, */etc/rc?.d* o en ubicaciones similares, donde **?** es el modo de ejecución. Cuando se pasa a un determinado modo de ejecución, **rc** le pasa el parámetro *start* a todos los scripts que se encuentran dentro del directorio del modo especificado cuyos nombres comienzan con **S** y el parámetro *stop* a los archivos cuyo nombre comienza con **K**. El programa *rc* ejecuta los scripts en orden numerico debido a que algunos servicios dependen de otros, es por esto que los nombres de los scripts además de empezar por *S* o *K* tienen un número. En realidad estos scripts no son más que enlaces simbólicos a los scripts principales que se suelen almacenar en */etc/rc.d*, */etc/init.d* o */etc/rc.d/init.d* (la ubicación depende de la distribución).

Sabiendo esto podremos determinar que servicios están activos o se pararán, en un modo de ejecución específico, buscando los scripts cuyos nombres de archivos comiencen por *S* o por *K* en el directorio de scripts de inicio SysV apropiado. También podemos usar la herramienta **chkconfig** de la siguiente manera:

```
# chkconfig --list
```

O si estamos interesado en un servicio concreto:

```
# chkconfig --list <servicio>
```

Gestionar los servios de los modos de ejecución

Al igual que existen dos formás para identificar servicios en un determinado modo de ejecución (o 3 si separamos el uso de *chkconfig* y la búsqueda de scripts por sus directorios) podemos gestionar los programás para que sean ejecutados o parados según el modo de ejecución de la misma manera.

Mediante el archivo */etc/inittab*:

Bastará con añadir o eliminar entradas del archivo

Con chkconfig:

Modificar los modos de ejecución para un determinado servicio

```
# chkconfig --level 235 <servicio> on|off|reset
```

Nota: Con *reset* le asignaremos su valor por defecto

Si hemos añadido un script de inicio al directorio principal de los scripts (*/etc/init.d*, por ejemplo) podemos hacer que *chkconfig* lo registre y añada los enlaces de inicio y fin apropiados en los directorios de modo de ejecución adecuados.

```
# chkconfig --add <servicio>
```

Cuando lo hagamos, *chkconfig* inspeccionará el script en busca de comentarios especiales para indicar los modos de ejecución por defecto. Puede que este método no funcione si el script carece de las líneas de comentario necesarias con los números de secuencia de los modos de ejecución que utiliza *chkconfig*.

Podemos hacer este trabajo de forma manual que sería creando nosotros mismos el enlace simbólico del script principal en los directorios de los modos de ejecucion en los que queremos que se inicie o detenga un programa concreto. Tenemos que tener en mente los nombres de archivos con S, K y los números de orden.

Importante: Podemos iniciar, detener o comprobar el estado de los servicios mediante la ejecución del comando '*service*':

```
# service samba status|start|stop
```

o con la ejecución del script principal

```
# /etc/init.d/samba status|start|stop
```

Existe la herramienta **ntsysv** la cual nos permitirá marcar aquellos servicios que queremos ejecutar durante el arranque. Es una herramienta 'gráfica' de línea de comandos, su uso es sencillo y puede resultar útil incluso para ver que servicios tenemos marcados en el arranque. También puede ser usada para especificar un runlevel concreto.

Comprobar y cambiar el modo de ejecución

Podemos comprobar el modo de ejecución por defecto y el modo de ejecución actual.

Para comprobar el modo de ejecución por defecto basta con desplegar el contenido del archivo */etc/inittab* y localizar la entrada que contiene como acción **:initdefault:**, esto no suele ser válido para equipos que usen *Upstart*, *systemd* u otra herramienta de inicialización.

Si por el contrario el sistema está en funcionamiento, puede ver su modo de ejecución actual como el comando runlevel:

```
# runlevel
N 2
```

El primer caracter es el modo de ejecución anterior, en este caso **N**, lo cual indica que el sistema no ha cambiado de modo de ejecución desde el arranque. El segundo caracter **2** es su modo de ejecución actual.

Se puede cambiar a modos de ejecución diferentes en un sistema operativo en funcionamiento con los programás **init** (o telinit), **shutdown**, **halt**, **reboot** y **poweroff**.

Por ejemplo para cambiar al modo monousuario o reiniciar el sistema podemos usar:

 # init 1 o **# init 6** (respectivamente)

init no es el comando más lógico para reiniciar, apagar o pasar a modo monousuario, ya que lo hace de forma inmediata y podríamos molestar a otros usuarios conectados al sistema, pero si que sirve por ejemplo para que el sistema vuelva a leer el archivo */etc/inittab*

```
El comando telinit es una variante de init, normalmente es un enlace
simbólico aunque la páginas man de ambos comandos presentan una
sintaxis ligeramente distinta. Si quisieramos por ejemplo que se
volviese a leer el archivo /etc/inittab, podriamos usar telinit o init
de la siguiente manera:
 # telinit q
 # telinit Q
```

Nota: Las herramientas *Upstart* y *systemd* ofrecen los comandos *init* y *telinit* que funcionan de una forma similar.

Cuando desee reiniciar (**6**), apagar (**0**) o pasar a modo monousuario (**1**) en un entorno multiusuario, lo mejor será emplear el comando *shutdown*.

Pasar a modo monousuario de forma inmediata:

```
# shutdown now
```

Reiniciar el sistema en 5 minutos

```
# shutdown -r +5
```

Apagar el sistema a las 13:45

```
# shutdown -P 13:45
```

Suspender el equipo (no lo apaga) a las 1:30 AM avisando a los usuarios.

```
#shutdown -H 1:30 "A las 1:30 de la madrugada el sistema va a ser
detenido"
```

Nota: La opción **-h** puede detener o apagar el equipo, aunque normalmente acaba apagándolo.

Detener el apagado de un equipo por un cambio de ultima hora (por ejemplo):

```
# shutdown -c "Se pospone el reinicio del sistema, se avisará
nuevamente. Disculpen"
```

Nota: *Upstart* y *systemd* ofrecen la herramienta *shutdown*

Otros tres comandos útiles para detener, reiniciar o apagar el sistema respectivamente tanto en SysV, Upstart como con systemd son: **halt**, **reboot** y **poweroff**

SISTEMA DE INICIALIZACIÓN UPSTART

Básicamente, **Upstart** hace lo mismo que las secuencias de comandos *Sysvinit* (*SysV*) pero está mejor diseñado para admitir el hardware actual y su continuo dinamismo, lo que permite conectarlo y desconectarlo de un ordenador en ejecución. Mientras *SysV* trabaja de una forma síncrona, el modelo basado en eventos de *Upstart* le permite responder a los eventos de una forma asíncrona cuando estos son generado, resolviendo así alguno de los problemás que presenta *SysV*, como el bloqueo de futuras tareas hasta que la actual se haya completado.

Para sistemás con *Upstart* el PID 1 sigue siendo para *init*, padre de todos los procesos en el sistema (incluso de los procesos huérfanos) y responsable de iniciar el resto de procesos.

Upstart ofrece funciones compatibles con *SysV*, no obstante también cuenta con sus propios scripts y difiere en algunos aspectos importantes, en concreto *Upstart* ignora el archivo */etc/inittab*, en su lugar proporciona un conjunto integrado de scripts de arranque que en principio pueden reemplazar totalmente a */etc/inittab* y a los scripts de arranque específico del modo de ejecución de SysV, que en vez de encontrarse bajo directorios como */etc/init.d/rc?.d*, */etc/rc.d/rc?.d* o */etc/rc?.d* se encuentran en el directorio **/etc/init**, con nombre '**servicio'.conf**, donde servicio es el programa que *init* tratará como un job. Los scripts de *Upstart* ofrecen más acciones que los de *SysV*, por ejemplo iniciar un servicio siempre que se conecte un determinado dispositivo de hardware.

Al igual que usábamos el comando **#service <servicio> start|stop|status** para iniciar un script *SysV*, en *Upstart* tenemos la opción de utilizar el comando **initctl** con el que podremos iniciar, parar, ver el estado, etc… de los servicios del sistema o directamente usar **start** y **stop**

$sudo initctl stop smbd o **$sudo stop smbd**

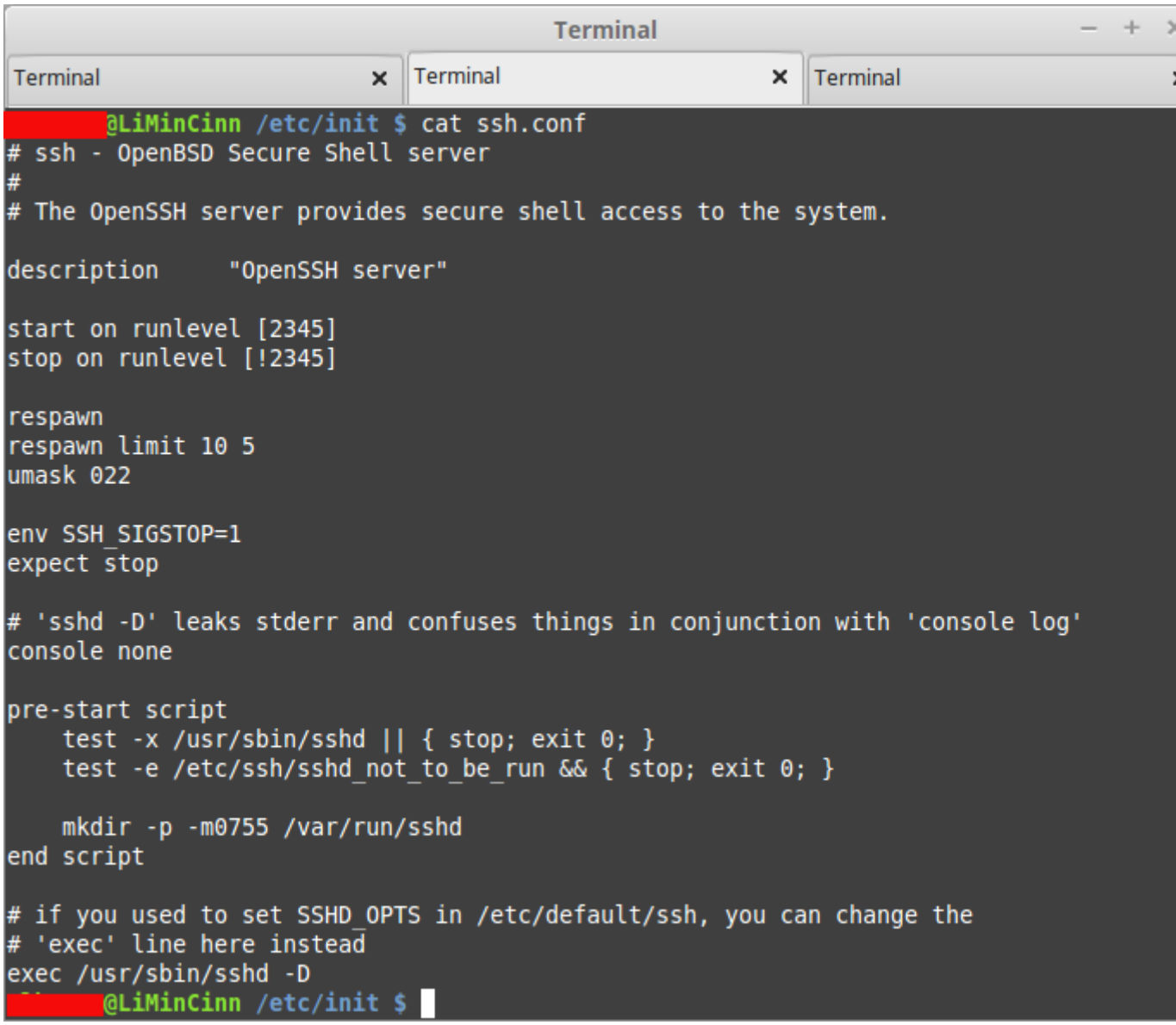
Nota: Para ver un listado de estos servicios podemos listar el contenido de */etc/init* donde se encuentran como ya sabemos todos archivos de configuración de servicios *Upstart*, por lo que el nombre del servicio es igual a los del directorio pero sin .conf

El comando *service* podremos seguir usándolo para servicios SysV instalados en el sistema. Anteriormente hemos comentado que los scripts de *Upstart* sustituyen de forma completa a los scripts de SysV y a su archivo */etc/initta*b, pero esto no es del todo

cierto ya que *Upstart* concede en cierto modo la compatibilidad con SysV. *Upstart* continua ejecutando scripts de SysV (ubicados en los mismos directorios como si de un sistema SysV se tratase) debido a que existen programás que aún no incluyen los scripts de configuración *Upstart* y necesitan ser arrancado de forma habitual.
Si ha instalado utilidades como *chkconfig* en sistemás que inicializan con *Upstart* deberá poder usarla para gestionar sus servicios *SysV*. No obstante *chkconfig* y estos servicios que aun se mantienen en *SysV* acabarán desapareciendo debido a que el progreso apunta a *Upstart* y *systemd*.

Gracias a la **API inotify** no será necesaria la recarga de un archivo de configuración (archivo con el que se configura un servicio por ejemplo *smb.conf*) para un determinado servicio, ya que esta proporciona un mecanismo para el seguimiento de eventos del sistema de archivos, lo cual supervisa tanto archivos individuales como directorios, por lo que si se diera el caso de que un archivo a sido modificado este será recargado automáticamente

Para cambiar los modos de ejecución en los que se ejecuta un servicio concreto, tendrá que editar su archivo de configuración *Upstart* (**/etc/init/'servicio'.conf**), localice el script del servicio que quieres modificar, ábrelo para editar, y encuentre las líneas que contengan el texto *start on* y *stop on*.
Estos archivos suelen tener un formato similar al siguiente:



Vemos la cadena **start on** y **stop on runlevel**, seguidas de **[2345]** y **[!2345]**, en este caso se indica que el servicio **ssh** se inicie en los modos 2345 y sea detenido para cualquier modo distinto (**!**) de los indicados, también 2345.

Podemos ver cadenas de start on más complejas en las que se hace uso de **mountall**, es un demonio propio de Debian y Ubuntu que se encarga de montar los sistemás de ficheros que se especifican en */etc/fstab* y */lib/init/fstab*, y que además emite una serie de eventos útiles cuando estos son o no montados como por ejemplo:

- **mounting**: Emitido cuando un sistema de archivos en particular está a punto de ser montado.
- **mounted**: Emitido cuando un sistema de archivos en particular ha sido montado correctamente.

- **all-swaps**: Se emite cuando se montan todos los dispositivos de intercambio
- **filesystem**: Una vez que todos los sistemás de archivos han sido montados (o intentado)
- **virtual-filesystem**: Emitida después de que se haya montado el último sistema de archivos virtual
- **local-filesystem** y **remote-filesystem**: Cuando el último sistema de archivos local o remoto (respectivamente) hayan sido montado

Por lo tanto podemos encontrar entradas como esta:

```
start on (local-filesystems
      or (mounting
      or (mounted
      or (virtual-filesystems
      or (remote-filesystems
      or (all-swaps or filesystem))))))

script
  echo "\n`env`" >> /dev/.initramfs/mountall.log
end script
```

Existen situaciones en las que nos interesa que un proceso se restaure automáticamente como por ejemplo en casos de fallos y cierres inesperados del servicio, para ello *Upstart* (al igual que SysV) ofrece el comando **respawn** con el que además podemos especificar límites como en el caso de la imagen (**respawn 10 5**) que viene a significar que si el proceso se reinicia más de 10 veces en 5 segundos, finalice definitivamente.

Nota: Para un proceso, un 'cierre inesperado' será siempre que no se detenga con la orden stop, mientras que para una tarea es cuando el código de retorno sea diferente a 0.

Se pueden seleccionar secciones del archivo **.conf**, como por ejemplo en nuestro caso aparece **pre-start script** que indica que los siguientes comandos sean ejecutados antes de que el servicio sea arrancado

Podemos habilitar los trabajos de usuarios para permitir que los usuarios tengan acceso a los métodos y propiedades de *Upstart*, el archivo para esto es **/etc/dbus-1/system.d/Upstart.conf**, los trabajos de usuario se encuentran en **~/.init**

Para ver los comandos que podemos utilizar dentro de un archivo *.conf* viste la web: http://upstart.ubuntu.com/wiki/Stanzas

Para más información podemos acudir a la siguiente página, donde aprenderemos incluso a crear nuestros propios archivos de configuración de servicios: http://upstart.ubuntu.com/cookbook/

SISTEMA DE INICIALIZACIÓN SYSTEMD

Sobre systemd

systemd ha sido creado para ofrecer un inicio más rápido y flexible que SysV, permitiendo el arranque paralelo de servicios y su inicio basado en la detección de conexión de nueva unidad externa.

Nota: Hasta ahora el **PID1** era para el programa **init**, cosa que ha cambiado en systemd a favor de **/usr/lib/systemd/systemd** y además *systemd* al igual que *Upstart* deja de utilizar el archivo */etc/inittab*

Algunas de las mejoras que ofrece systemd:

- Se ha mejorado sustancialmente la velocidad de inicialización del sistema
- systemd asume que cualquier dispositivo puede ser conectado o desconectado en cualquier momento (**hotplug**)
- systemd utiliza la activación de daemons por medio de sockets, aportando capacidades de paralelización
- Una de sus características es el registro (**journal**) mediante **cgroups** de todos los servicios y procesos iniciados
- systemd es *modular*, esto quiere decir que se han elaborado una seríe de "paquetes" en los que varios servicios son administrados de forma conjunta

Según definición oficial systemd es: Un sistema y administrador de servicios para Linux, compatible con scripts de inicio (init) SysV y LSB. systemd proporciona capacidades de paralelización agresiva, utiliza socket y activación D-Bus para iniciar los servicios, ofrece la puesta en marcha de demonios bajo demanda, realiza el seguimiento de procesos utilizando Linux cgroups, soporta copia instantánea de volumen y la restauración de estado del sistema, mantiene puntos de montaje y automontaje e implementa un elaborado servicio lógico de control transaccional basado en la dependencia.

Vamos a desglosar un poco esta definición:

- **Capacidades de paralelización agresiva usando socket**: systemd crea de una misma vez todos los sockets para todos los demonios acelerando así el arranque completo e iniciar más procesos en paralelo. En un segundo paso systemd ejecutará a la vez todos los demonios.
- **Activación D-Bus para iniciar servicios**: Utilizando la activación D-Bus, un servicio puede ser iniciado la primera vez que es accedido.
- **Seguimiento de procesos utilizando Linux cgroups**: cgroup también llamados grupos de control, es una característica del kernel para crear límites, políticas e incluso explicar el uso de los recursos de ciertos grupos de procesos. cgroup asocia un conjunto de tareas con un conjunto de parámetros, para uno o más subsistemás, proporcionando un control de servicios y tareas, así como todos sus

futuros 'hijos' en grupos jerárquico. Un subsistema es un módulo resultado de la agrupación de diversas tareas con el fin de mantener un mejor control sobre estas de forma particular.

Nota: La herramienta **systemd-cgls** nos muestra recursivamente el contenido del árbol de jerarquías de un determinado grupo de control de Linux.

- **Mantiene puntos de montaje y automontaje**: Puede utilizarse para montar o desmontar los puntos de montaje, quedando */etc/fstab* como una fuente de configuración adicional a la que podremos llamar para su supervisión con la opción "comment=" de fstab para marcar las entradas controladas por systemd.

Unidades de servicios (units)

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

Nota: Los archivos bajo el directorio */etc/systemd/system/* prevalecerán en caso de duplicados.

Existen siete tipos diferentes de unidades:

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

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

Nota: La unidad **default.target** activa servicios y otras unidades dependientes, durante el arranque de systemd. Esto lo comentaremos en el apartado *target*

Compatibilidad de systemd con SysV

systemd al igual que Upstart ofrece compatibilidad con SysV (comando *service* y *chkconfig*) para aquellos servicios que aun soportan o funcionan únicamente con scripts de inicio SysV (actualmente en 2015, son pocos los servicios que corren bajo SysV). Upstart pese a mantener compatibilidad con los comandos *service* y *chkconfig* de SysV implementó su propia utilidad para la administración de servicios, de igual modo systemd lo hace con su herramienta **systemctl**

#systemctl stop nombreservicio.service

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

Habilitar el servicio httpd al arranque del sistema (*chkconfig httpd on* , para SysV):

#systemctl enabled httpd.service

Listar todas las unidades de servicios instaladas (algo parecido a *chkconfig –list*)

#systemctl list-unit-files

O solo aquellas que se encuentran en activadas:

#systemctl list-units o **#systemctl**

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

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

Nota: Consulte man *systemd.unit* para más detalles.

Uso de systemctl

La utilidad de administración de las unit de systemd es **systemctl**, la cual combina las herramientas *service* y *chkconfig* de SysV, por lo tanto podremos arrancar, parar, recargar servicios, activar o desactivar servicios en el arranque, listar los estados de los servicios,etc…

Creo que con la siguiente tabla veremos el uso básico de systemctl además de sus "iguales" en SysV.

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

1* Puede que un servicio este "**enable**" pero no tiene porqué estar activo cuando iniciemos sesión ya que puede que ese servicio este configurado para ejecutarse solo en determinados runlevels (o target en nuestro caso), a diferencia de *chkconfig –list de SysV* que mostraba todos los servicios con todos los runlevels posibles y para cada uno indicaba si estaba on o off. Para conseguir algo parecido en *systemd*, tendríamos que listar los target disponibles y veríamos que servicios se ubican dentro de estos, así

sabremos con que target (o runlevel) un servicio será iniciado. Podemos emplear el siguiente listado

```
#ls /etc/systemd/system/*.wants/httpd.service
/etc/systemd/system/multi-user.target.wants/httpd.service
```

Este comando nos devuelve que el servicio httpd se encuentra bajo *multi-user.target* lo que viene siendo el runlevel 3 de SysV. En los siguientes apartados conoceremos los distintos target que de algún modo tienen similitud con los runlevels de SysV

Nota: Podríamos haber omitido el subdirectorio */httpd.service* para conocer la lista completa de targets y sus unit (mount, service, socket…)

Podemos hacerlo de una forma inversa (para lo cual hay que conocer el target), es decir ver que servicios están ejecutándose para un target concreto:

#systemctl show -p "Wants" multi-user.target

Además de los comandos de la tabla también podemos ver las units de servicios que tenemos cargados en el sistema con el comando:

#systemctl -t service list-units –all

Con este comando veremos que servicios están cargados y además si están activos o muertos a parte de una pequeña descripción.

Nota: Podemos cambiar service por mount, socket, device… para listar otros tipos (**-t**) de units.

Si en vez de ver las units cargadas queremos ver cuantas hay instaladas (es decir aquellas que tienen archivos de configuración instalados en nuestro sistema y por lo tanto están disponibles):

#systemctl -t service list-unit-files –all

Nota: Podemos cambiar el tipo de unit o directamente omitir el parámetro -t y listar todas.

Aprenderemos más sobre el comando **systemctl** en el apartado de comandos para este tema (Tema-5)

Sugerencia: Puede utilizar las siguientes órdenes **systemctl** con el parámetro **-H usario@host** para controlar una instancia de *systemd* en una máquina remota. Esto utilizará SSH para conectarse a la instancia systemd remota.

Target

systemd utiliza *target* en vez de runlevels (0123456) que reciben un nombre (en vez de un número) para identificar un propósito específico, con la posibilidad de realizar más de una acción al mismo tiempo. Algunos *targets* pueden heredar todos los servicios de otro *target* e implementarse con servicios adicionales. La siguiente tabla muestra la similitud entre algunos de los target con los runlevels de SysV:

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

Nota: Actualmente no existen *target* similares a los runlevels 2 y 4 de SysV, pero podríamos definirlos nosotros mismos.

Existen otros target que podremos ver con el comando:

#systemctl list-units –type=target

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

 # systemctl isolate graphical.target

Nota: Esto podría ser equivalente a la orden *#telinit 5 de SysV*

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

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

Nota: Los archivos unit de */etc/systemd/system* tienen una mayor precedencia sobre archivos unidad de */lib/systemd/system*

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

Si quisiéramos podríamos crear un target desde 0 a nuestro gusto y darle un nombre (*mylevel.target*) y a continuación habilitarlo:

 # systemctl enable mylevel.target

El efecto de esta orden crea un enlace simbólico (*/etc/systemd/system/default.target*) que apunta a nuestro *mylevel.target*. Esto solo funciona si hemos añadido la etiqueta **[Install]** al archivo, de la siguiente manera:

[Install]
Alias=default.target

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

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

Journal

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

Hasta Fedora 18 (e inclusive a día de hoy, Enero del 2015, en Centos7) el registro de *journal* se almacena de forma volátil en **/run/log/journal** por lo que será eliminado tras un reinicio. La intención es que venga configurado por defecto de manera que sea un registro persistente y con directorio principal **/var/log/journal** (que no venga por defecto no quiere decir que no lo podamos configurar de tal modo).

Una de las razones por las que *syslog* comenzaba a quedarse “anticuado” es la necesidad por ejemplo de mostrar las últimás líneas del registro de un determinado servicio en el momento de preguntar por su estado (con *systemctl*), esto necesita que archivos de

registro sean descomprimido en tiempo real y a la vez mostrado, algo que con *syslog* se hacía casi imposible y además, ineficiente y poco seguro.

Nota: Estudiaremos más sobre los log de registros (*syslog*, *rsyslog*, *syslog-ng*, y *journal*) en el tema dedicado a tal fin de este mismo libro.

Hasta ahora nos haremos la idea de que journal es el sustituto de *syslog*, el cual nos permite filtrar la salida del registro por campos, dejar el registro de forma volátil o volverlo persistente mediante su archivo de configuración (**/etc/systemd/journald.conf**), darle un tamaño máximo al fichero de registro, permite la compatibilidad con el antiguo *syslog* mediante el *socket /run/systemd/journal/syslog* (tendremos que asociar *syslog* a este *socket* y no a */dev/log*, el paquete *syslog-ng* proporciona automáticamente la configuración necesaria), reenviar la salida del registro a una terminal, etc…

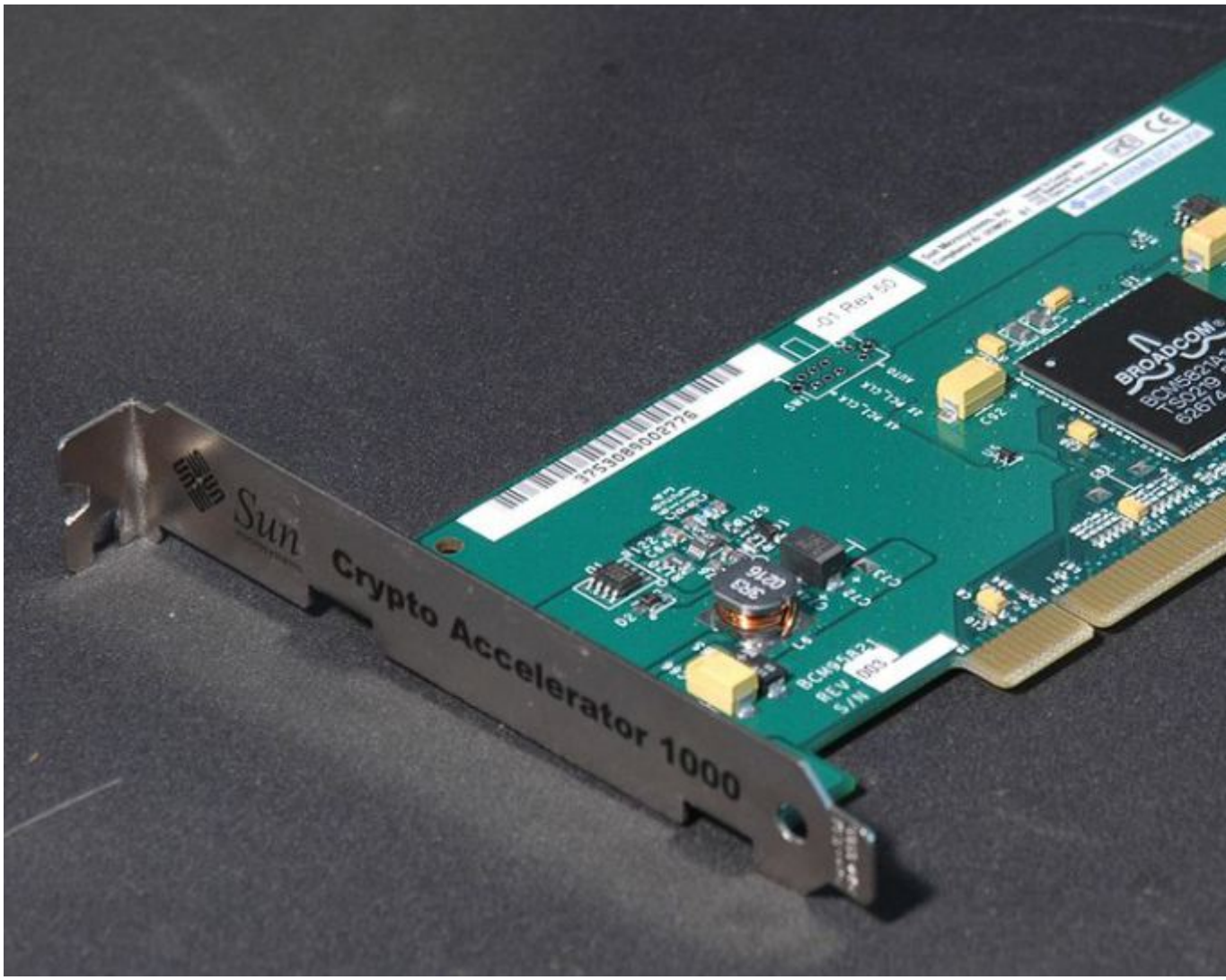
Solucionar problemás

Este apartado lo podemos encontrar igualmente en: https://wiki.archlinux.org/index.php/Systemd_%28Espa%C3%B1ol%29 (*Solución de problemás*), he creído conveniente introducirlo en el tema para que sirva como ejemplo práctico.

Nota: Intenté cambiarlo por un error personal, pero el comando *systemctl –state=failed* en mi *Centos7* solo me proporcionaba el error:

```
rngd.service loaded failed failed Hardware RNG Entropy Gatherer Daemon
```

RGN es un Generador de números aleatorios a partir de un componente hardware, en vez de crearlos mediante software. Al no usar este tipo de software pues evidentemente me da error. (Mira si a alguno de ustedes le ocurre lo mismo, ya llevan 2 problemás solucionados, este y el que se explica a continuación, jeje)



Hardware RGN

Podría haber creado mi propio problema, pero creo que el ejemplo de la wiki de arch puede ahorrarme un maravilloso tiempo.

1. Lo primero es comprobar si existen servicios que están fallando en el incio:
   $systemctl –state=failed
   systemd-modules-load.service loaded failed failed Load Kernel Modules
2. Vamos a indagar sobre este error. Miraremos su estado.
   $systemctl status systemd-modules-load
   systemd-modules-load.service – Load Kernel Modules
   *Loaded: **loaded** (/usr/lib/systemd/system/systemd-modules-load.service; static)*
   *Active: **failed** (Result: exit-code) since So 2013-08-25 11:48:13 CEST; 32s ago*
   Docs: man:systemd-modules-load.service(8).
   man:modules-load.d(5)
   *Process: **15630** ExecStart=/usr/lib/systemd/systemd-modules-load (code=exited, status=1/FAILURE)*

3. Tenemos el PID, investiguemos sobre este:
   $journalctl -b _PID=15630— *Logs begin at Sa 2013-05-25 10:31:12 CEST, end at So 2013-08-25 11:51:17 CEST. —*
   Aug 25 11:48:13 mypc systemd-modules-load[15630]: Failed to find module 'blacklist usblp'
   Aug 25 11:48:13 mypc systemd-modules-load[15630]: Failed to find module 'install usblp /bin/false'
4. Este mensaje -> Failed to find module 'blacklist usblp' <- la palabra module y blacklist juntas mmmm, vamos a revisar si por casualidad tuviesemos una blacklist configurada…
   $ ls -Al /etc/modules-load.d/
   …
   -rw-r–r– 1 root root 79 1. Ene 2015 blacklist.conf
   -rw-r–r– 1 root root 1 2. Mar 2014 encrypt.conf
   -rw-r–r– 1 root root 3 5. Dic 2014 printing.conf
   …
5. Al abrir el archivo blacklist, encontramos precisamente las líneas 'blacklist usblp' y 'install usblp /bin/false', por lo que vamos a comentarlas (añadiendo # delante de cada una de ellas) para que no las tenga en cuenta y reiniciaremos el servicio a ver que tal.
6. Si no hay señales de error en el prompt vamos a comprobar cual es su estado:
   $ systemctl status systemd-modules-load
   systemd-modules-load.service – Load Kernel Modules
   Loaded: loaded (/usr/lib/systemd/system/systemd-modules-load.service; static)
   Active: active (exited) since So 2013-08-25 12:22:31 CEST; 34s ago
   Docs: man:systemd-modules-load.service(8)
   man:modules-load.d(5)
   Process: 19005 ExecStart=/usr/lib/systemd/systemd-modules-load (code=exited, status=0/SUCCESS)
   Aug 25 12:22:31 mypc systemd[1]: Started Load Kernel Modules.

Otra forma de detectar arranques en el inicio es mediante su **depuración**, para lo que tendremos que añadir los parámetros **systemd.log_level=debug systemd.log_target=kmsg log_buf_len=1M** a la línea del kernel en el gestor de arranque

NOTAS:

Podremos descargar *systemd* desde la siguiente *página oficial*: http://www.freedesktop.org/software/systemd/ o bien consultar su documentación oficial http://freedesktop.org/wiki/Software/systemd/

Puedes ampliar la información aquí dada, e incluso conocer otras funcionalidades o aspectos de *systemd* como:

- Características de systemd (ampliadas)
- Línea de comandos de arranque del Kernel
- Cambiar el número de gettys por defecto
- Personalizar un archivo unidad o agregar uno personalizado
- Inicio de sesión automático en una terminal
- Implementación readahead (lectura anticipada)
- Escribir archivos .service personalizados
- Archivos temporales y temporizadores

a través de algunos sitios:

https://fedoraproject.org/wiki/Systemd/es
https://fedoraproject.org/wiki/SysVinit_to_Systemd_Cheatsheet/es
https://wiki.archlinux.org/index.php/Systemd_%28Espa%C3%B1ol%29
https://wiki.ubuntu.com/SystemdForUpstartUsers
https://wiki.archlinux.org/index.php/Systemd

Examen 102

LPIC-1 Capítulo 6

EN ESTE CAPÍTULO SE ABARCAN LOS SIGUIENTES OBJETIVOS DEL EXAMEN:

- 106.1: Instalar y configurar X11 (2)
- 106.2: Configurar un administrador de pantalla (1)
- 106.3: Accesibilidad (1)
- 107.3: Localización e internacionalización (3)
- 108.1: Mantener la hora del sistema (3)
- 108.4: Administrar impresoras e impresión (2)

Configurar el sistema de ventanas X, localización e impresión

Sobre X Windows System

"Parte de la siguiente información está extraída de https://www.freebsd.org/doc/es_ES.ISO8859-1/books/handbook/x11-wm.html"

En Linux la principal GUI se denomina sistema de ventanas X o "*X Windows System*" (X para abreviar). Fue diseñado desde el principio pensando en redes y adoptó un modelo **cliente-servidor**. *X Windows System* es independiente del protocolo y de la red, por lo que existen muchas posibles configuraciones para ejecutar el modelo *cliente-servidor* en diferentes equipos conectados a una red. El *servidor* es responsable de manejar la pantalla, la entrada de datos a través del teclado, el ratón, etc. Cada aplicación X (como una *XTerm*, *Firefox*, *Gedit*, etc…) es un *cliente*. Para un sistema local el servidor y cliente se ejecutarán en la misma máquina, no obstante la gran ventaja radica si por ejemplo tenemos un servidor potente en la oficina o casa que haga de cliente (es decir ejecutará las aplicaciones que vayamos a utilizar) mientras son administradas y accedidas desde distintos servidores X como pueden ser el resto de ordenadores personales de la familia o compañeros de trabajo.

Importante: Recuerde que el servidor X es la máquina con el monitor y el teclado, mientras que los clientes X son los programás que muestran las ventanas (las propias aplicaciones).

X no impone la manera en la que deben de gestionarse las ventanas (cómo moverlas con el ratón, cómo deben ser las barras de título en cada ventana, si tienen o no botón de cierre, etc…) sino que delega esta responsabilidad en una aplicación llamada **gestor de ventanas** (*Mutter*, *Kwin*, *Muffin*, *compiz*, *fluxbox*, Window Maker, etc…) que nos ofrecerá un aspecto diferente en función de cual usemos.

Nota: Algunos entornos de escritorio como **KDE**, **GNOME** y **Cinnamon** tienen sus propios *gestores de ventanas* integrados con el escritorio. *Mutter*, *Kwin* y *Muffin* serán los gestores de ventanas actuales para *GNOME*, *KDE* y *Cinnamon* respectivamente.

Además de los entornos de escritorio (que veremos en breves) y sus gestores de ventanas, es importante conocer el papel del **gestor de pantalla X** (*Session Manager* o *X Display Manager*) y los **gestores de archivos** (*File Manager*).

- El *gestor de pantalla X* es una parte opcional del sistema *X Window* que se usa para la gestión de sesiones. **XDM** dispone de un interfaz gráfica para elegir a qué pantalla se quiere conectar y pedir la información de autentificación como el nombre de usuario y le contraseña. Esto quiere decir que se encarga de las entradas al sistema en la pantalla a la que está conectado y arranca el gestor de sesiones para que lo utilice el usuario (normalmente un gestor de ventanas X). Al finalizar sesión *XDM* envía la señal de que el usuario ha terminado y que se le debería desconectar de la pantalla. En este punto *XDM* puede mostrar las pantallas de entrada al sistema y de selección de pantalla para que el siguiente usuario intente acceder al sistema.

Nota: Veremos más sobre los gestores de pantalla en la sección "*Configurar servidores XDMCP*" de este mismo capítulo.

- El **gestor de archivos** proporciona un punto de acceso integrado a los archivos y aplicaciones. Mediante un gestor de archivos podremos crear, mostrar, gestionar y personalizar las carpetas y documentos, además de permitirnos navegar por el sistema de archivos. Existen multitud de *gestores de archivos* para *GNU/Linux* por ejemplo:
  - **Dolphin**: Administrador de archivos por defecto en *KDE*. Cuenta con una interfaz fácil de usar.
  - **Nautilus**: Gestor de archivos por defecto en *GNOME*, *Nautilus* tiene una interfaz intuitiva y se puede ampliar mediante plugins.
  - **Thunar**: Es el administrador de archivos por defecto en el escritorio *Xfce*. Su objetivo es mantenerse simple ofreciendo únicamente las funciones básicas. No tiene soporte para el uso de pestañas pero cuenta con una barra lateral izquierda para acceder rápidamente a los lugares comunes
  - **Nemo**: Es el gestor de archivos por defecto en el entorno de escritorio *Cinnamon*. Es un fork de *Nautilus* pero ofrece algunas ventajas extras.
  - **Double Commander**: Ofrece dos interfaces, **GTK** y **Qt**. *GTK* combina bien con *GNOME*, mientras que *Qt* se adapta mejor en *KDE*. Viene con

un estilo de doble panel e incluye un gran número de opciones de configuración.
  - **Midnight Commander**: Es un popular administrador de archivos de doble panel para la terminal, basado en la biblioteca *ncurses*.

Entornos de escritorio

Un *entorno de escritorio* puede ser cualquier cosa que vaya desde un simple *gestor de ventanas* hasta una completa *suite* de aplicaciones de escritorio como *KDE* o *GNOME*, en la que se integra su propio gestor de ventanas, gestor de archivos y multitud de utilidades y aplicaciones que hacen la vida más fácil al usuario del sistema.

Los principales entornos de escritorios son:

- **GNOME**: Es un entorno de escritorio amigable que permite a los usuarios una configuración fácil de sus sistemás gracias a su panel con el que poder iniciar las aplicaciones y mostrar su estado, el escritorio y un conjunto de herramientas y aplicaciones de escritorio estándar. La manera más fácil de arrancar *GNOME* es con *GDM*, el gestor de pantalla de *GNOME*, que se instala con el escritorio *GNOME* al ser parte del mismo, aunque venga desactivado por omisión. Podremos activarlo añadiendo *gdm_enable=”YES”* a */etc/rc.conf*. Tras reiniciar, *GNOME* arrancará automáticamente al acceder al sistema; no es necesario configurar nada más.
- **KDE**: Es un entorno de escritorio contemporáneo y fácil de usar el cual ofrece al usuario un bonito escritorio y transparente en red, un sistema de ayuda integrado que facilita un acceso adecuado, un gran número de útiles aplicaciones como su famoso navegador web *Konqueror*, menús y barras de herramientas estándares, teclas programables, esquemás de color, etc
- **XFce**: Es un entorno de escritorio basado en el *toolkit* GTK utilizado por *GNOME* pero es mucho más ligero y está pensado para aquellos que quieran un escritorio sencillo, eficiente y fácil de utlizar y configurar. *XFce* ofrece interesantes características como su fácil configuración utilizando el ratón (con arrastrar y soltar), menú y accesos directos a aplicaciones, gestor de ventanas integrado, rápido, ligero y eficiente (ideal para máquinas viejas o con escasos recursos.
- **Cinnamon**: Es un fork (derivación) de Gnome Shell, basado en Gnome 3. Tiene una apariencia minimalista pero bastante bonita y la funcionalidad de Gnome 2 mezclada con algunas cosas de Gnome 3 como el área de aplicaciones y escritorios ya bien conocida en Gnome Shell. La distribución con la que fue desarrollada fue Linux Mint la cual está bastante orientada al usuario Novel y proveniente de otros sistemás como Mac o Windows

Servidores X

Principales servidores X

- **XFree86**: Servidor X dominante hasta 2004, la mayoría de los sistemás modernos ejecutan XFree86 4.x o X.org-X11. Con algunas tarjetas antiguas es posible que tenga que recurrir a este servidor X e incluso a versiones anteriores a la 4.x
- **X.org-X11**: Es el servidor incluido en las distribuciones más modernas, se basa en XFree86, ambos son virtualmente idénticos en la mayoría de los aspectos importantes. Una diferencia significativa es el nombre del archivo de configuración y la ubicación de las fuentes.

Nota: Tanto *XFree86 4.x* como *X.org-X11* admiten una arquitectura modular de controladores, lo que significa que puede hacerse con un módulo de controladores específicos para su tarjeta y usarlo con cambios mínimos en su configuración de X. Si tienes problemás con los controladores estándar de vídeo de X, puede descargar de la página del fabricante aquellos que sean para su modelo de gráfica y tengan soporte para Linux.En ocasiones son estrictamente necesarios como por ejemplo para funciones de aceleración 3D en tarjetas modernas. Quizás tenga que reinstalar los controladores de fabricantes como *Nvidia* o *ATI* cuando actualice su kernel.

- **Accelerated-X**: Es un servidor comercial que sirve como alternativa a los de código libre mencionados anteriormente. Su base de controladores son independientes y las herramientas y los archivos de configuración son completamente diferente a los de *XFree86* y *X.org-X11*.

Herramientas, configuración y opciones del servidor X

En ocasiones tendrá que recurrir a los archivos de configuración del servidor X para modificar ciertos aspectos de su configuración, como el teclado, ratón, tarjeta de vídeo, etc.. podemos hacer uso de determinadas utilidades para tal fin como por ejemplo:

- El servidor X posee la capacidad de consultar el hardware y generar un archivo de configuración mediante las herramientas **xfree86 -configure** o **xorg -configure** que cargarán todos los drivers y módulos del hardware y lo escribirán en un archivo *xf86config.new* o *xorg.conf* que nos valdrá como punto de partida. Es indispensable ejecutar estos comandos en modo *monousuario* o *runlevel 3*, donde la interfaz gráfica no esté configurada para iniciarse. Si hemos realizado algunos cambios podremos arrancar nuevamente X para comprobarlos con **startx** o pasando a un modo de ejecución donde X este activa. Si seguimos sin conseguir los objetivos podremos pasar nuevamente a modos de consola. Una vez hayamos configurado en gran medida nuestro servidor X podremos ejecutar **telinit 5** para activar el modo gráfico. En sistemás *Debian* y derivados lo mejor será matar el servidor X mediante el comando **kill** o **Control+Alt+Retroceso** y volviendo a iniciar con el script de inicio.

- Cada distribución suele contar con sus propias herramientas para configurar el servidor X como es el caso de *RedHat* que cuenta con **system-config-display** o la distribución SUSE con sus herramientas **YaST** o **YaST2**.
- Otras dos herramientas que actualmente se encuentran en desuso son **xf86cfg** y **xorgcfg** que si su distribución cuenta con ellas pueden serle de ayuda.
- Para versiones anteriores a *XFree86 4.x* estas herramientas mencionadas no funcionarán ya que para esta versión del servidor de X se cuenta con herramientas como **xf86config**, **xconfigurator** o **xf86setup**.

Los nombres de los archivos de configuración y su ubicación suelen variar según el servidor de X que utilicemos, por ejemplo para xorg es posible que encontremos el archivo de configuración en */etc/xorg.conf* o */etc/x11/xorg.conf*, mientras que para XFree86 4.x es posible que se oculte con el nombre de *XF86config-4* o *XF86config* en */etc/* o */etc/X11*.
Estos archivos de configuración suelen estar divido por **secciones**, una por cada característica principal y que comienzan con una línea *Section* y el nombre de la sección y finalizan con *EndSection*. Veremos algunos ejemplos en los siguientes apartados.

Secciones principales de los archivos de configuración

Para *XFree86 4.x* y *X.org-X11* las principales secciones son: *Module*, *InputDevice*, *Monitor*, *Device*, *Screen* y *ServerLayout*.

- La sección **Module**, controla la carga de los módulos del servidor X que son los controladores de las características específicas del hardware. Su sección duele tener el siguiente aspecto:

```
Section "Module"
 Load "dbe"
 Load "glx"
 ...
Endsection
```

- La sección **InputDevice** como su nombre indica referencia a los dispositivos de entrada, como por ejemplo al teclado y al ratón. La gran mayoría de secciones cuentan con la línea *Identifier "nombre"* que sirve para identificar el dispositivo del que trata y una serie de opciones o características del dispositivo.

```
Section "InputDevice"
 Identifier "Keyboard0"
 Driver "kbd"
 option "xkbmodel" "pc105"
 option "Autorepeat "500 200"
 ...
Ensection
```

Las líneas *Identifier* suelen utilizarse posteriormente en la sección *ServerLayout*, que digamos que es donde se unen diferentes dispositivos para realizar una función conjunta

o en la sección *Screen* por ejemplo, si se tratase de monitores, para agrupar varios dispositivos de monitores en un mismo conjunto.

La línea *Driver* deja claro que indica el driver a cargar para hacer posible el uso del teclado.

Algunas opciones interesantes con respecto al teclado y al ratón son:

Autorepeat "500 200″ : Indica que tiempo debemos de dejar pulsado una tecla (*500 milisegundos*) para que comience a repetirse el caracter y el tiempo que pasará hasta que vuelva a ser repetido, suponiendo que tenemos la tecla aun pulsada (*200ms*)

Device: Es una opción para el *Identifier "Mouse0"* y con ella indicamos el archivo de dispositivo por el cual se accede al ratón, por ejemplo */dev/input/mice*, */dev/psaux* (para el puerto de ratón PS/2), */dev/usb/usbmouse* (un antiguo identificador de ratones USB), */dev/ttyS[0|1]* (primer o segundo ratón RS-232).

Protocol: Indica el protocolo usado para el dispositivo. El protocolo *"Auto"* puede adivinar el protocolo que es necesario utilizar para el dispositivo en cuestión.

Nota: *PS/2* es tanto una interfaz de hardware como un protocolo de software; muchos ratones USB utilizan el protocolo de ratón *PS/2* aunque no usen el puerto de ratón *PS/2* necesariamente.

Emulate3Buttons: En ratones sin bola, al pulsar ambos botones a la vez produce el efecto de apretar la ruleta, como si fuese un 3º botón.

ZaxisMapping: Asocia el efecto de "ruleta hacía arriba" y "ruleta hacia abajo" con los botones 4 y 5.

- La sección **Monitor** tiende a ser bastante extensa y define alguno de los aspectos más delicados de la configuración de X, como la frecuencia de refresco y los posibles modos de resolución. La resolución deseada a emplear se define en la sección *Screen*. Algunas líneas de esta sección importantes podrían ser las que ajustan los modos de resolución (modeline) y las velocidades de refresco del monitor (horizontal y vertical).

```
Section "Monitor"
 ...
 HorizSync 30.0-83.0
 VertRefresh 55.0-75.0
 Modeline "1920x1080" 138.50 1920 1968 ....
 ...
Ensection
```

Las líneas **HorizSync** y **VertRefresh** definen el rango de velocidades de refresco horizontal y vertical que puede aceptar el monitor en Kilohercios (*Khz*) y hercios (*Hz*),

respectivamente. Quiere decir esto que cuando el manual del fabricante del monitor especifica que el monitor puede trabajar en rangos de (por ejemplo) 27 a 96 kHZ en valores horizontales y de 50 a 160 Hz en valores verticales, el monitor estará preparado para admitir una velocidad de refresco vertical de hasta 160Hz dependiendo de la resolución, por ello estos valores determinan la resolución máxima. La velocidad máxima de refresco del monitor vendrá dada por los valores de refresco horizontal y vertical, y la resolución máxima.

Las líneas de modo (modeline) definen las combinaciones de tiempos horizontales y verticales que pueden producir una resolución y velocidad de refresco dadas. Si se le pide que genere una resolución concreta, X buscará todas las líneas de modos y empleará aquellas que ofrezcan la velocidad de refresco más alta a dicha resolución. Muchas archivos de configuración omiten estás líneas y se debe al uso de la característica **DDC**.

XFree86 4.x y *Xorg-X11* suelen incluir la característica **DDC** (*Data Display Channel*): Protocolo que permite a los monitores indicar al ordenador sus velocidades máximás de refresco horizontal y vertical y las líneas de modo (modeline) apropiadas.

- La sección **Device** define varias opciones especificas para servidores X como por ejemplo la tarjeta gráfica. En esta sección al igual que con el resto de dispositivos, se referencia a los drivers. Los nombres para los drivers de la tarjeta gráfica podemos deducirlos de los nombres de sus controladores que suele encontrarse en */usr/lib/xorg/modules/drivers* (o lib64) o */usr/X11R6/lib/modules*. Sus nombres suelen acabar en *'_drv.0'*
- La sección **Screen** informa a X de la combinación de monitores y tarjetas de vídeo que se están utilizando, por medio de las líneas *Monitor "Monitor0"* y *Device "Videocard0"* ambas correspondientes a los identificadores (*Identifier*) de sus correspondientes secciones. Además es posible que la sección *Screen* esté dividida en sub-secciones que definen los diferentes modos de vídeo que puede utilizar X (resoluciones) con los que iniciar el monitor y la profundidad de color, que viene definida por la línea **Depth**. Por el contrario fuera de estas sub-secciones, es decir en la sección general de *Screen* aparecerá la línea **DefaultDepth**, que ya podemos imaginar para que es.

```
Section "Screen"
Identifier "Screen0"
Device "Videocard0"
Monitor "Monitor0"
DefaultDepth 24
SubSection "Display"
Depth 24
Modes "1920×1080" "1280×1024" "1024×768"
EndSubsection
SubSection "Display"
Depth 24
```

Modes “1024×768″ “800×600″ “640480”
EndSubSection
EndSection

Nota: La línea Modes especifica la resolución que deseamos emplear

- Por ultimo, la sección **ServerLayout**, es la sección que reúne el resto de componentes de la configuración de X. Normalmente esta sección identifica una sección *Screen* o varias, dependiendo del número de monitores y dos secciones *InputDevice* uno para el teclado y otro para el ratón.

Nota: Muchos entornos de escritorio y otras utilidades de usuario ofrecen herramientas para definir opciones de dispositivos, por lo que los valores indicados en los archivos de configuración del servidor X pueden ser invalidados por los ajustes del usuario.

Para obtener información de la pantalla usamos la herramienta **xdpyinfo**. Esta herramienta nos ayudará a conocer el número de versión de X, la resolución y la profundidad de color de todas las pantallas actuales, información de extensiones de X (módulos de software que proporcionan capacidades ampliadas a X), etc..

Configurar las fuentes de X

El sistema de fuentes principal de X se puede definir desde el archivo de configuración de X. Otra opción es configurar un servidor de fuentes para gestionarlas de forma centralizada.

Las tecnologías de fuentes se pueden separar en dos categorías:

- **Fuentes de mapa de bits**: El tipo más simple de formato de fuente. Representa las fuentes mediante los pixeles individuales de un vector que pueden estar activos o inactivos (apropiadas para ordenadores con poca potencia). El problema principal de estas fuentes es que deben de optimizarse cada fuente para mostrarse con una resolución concreta.
- **Fuentes escalables**: Corresponde a la mayoría de las fuentes modernas, son también conocidas como fuentes perfiladas. Este tipo de formato representa cada carácter como una serie de líneas y curvas en una matriz de alta resolución permitiendo al ordenador escalar cualquier tamaño de fuentes o resolución. Las fuentes escalables se suelen ver peor que las de mapa de bits.

Los dos formatos principales son **PostScript Type 1** o **Type 1** de *Adobe* y **TrueType** de *Apple*.

Configurar las fuentes básicas de X

Las fuentes básicas son aquellas gestionadas directamente por X. Para configurarlas, debe de hacer dos cosas: preparar un directorio de fuentes que las contenga y añadirlo a la ruta de fuentes de X.

Preparar un directorio fuente:

Los directorios para instalar las fuentes suelen ser */usr/X11R6/lib/X11/fonts* para *XFree86*, */usr/share/fonts/* o */usr/share/X11/fonts* para *Xorg* y */opt/fonts* o */usr/local/fonts* para fuentes externas. Las extensiones para las fuentes *Type 1* son **.pfa** o **.pfb** y **.ttf** para fuentes *TrueType*

Una vez copiadas las fuentes a un directorio, deberá de preparar un archivo de resumen que las describa (*fonts.dir* o *fonts.scale*). El modo más sencillo para crear estos archivos es utilizando **mkfontscale** y **mkfontdir**.

mkfontscale lee todas las fuentes del directorio actual y crea el archivo *fonts.scale* que es como el archivo *fonts.dir* pero describe solo fuentes escalables.
mkfontdir combina el archivo *fonts.scale* con el archivo *fonts.dir*, creando este si no existe ya.

Nota: Es preferible utilizar el programa *mkfontscale* ya que gestiona ambos tipos de fuentes.

Si vas a utilizar una distribución antigua puede probar con los programás alternativos **ttmkfdir**, para fuentes *TrueType* (fonts.dir) o **type1inst** para fuentes *Type1*

Nota: Las fuentes básicas presentan inconvenientes importantes:

- La integración entre la visualización por pantalla e impresión no es fácil, por lo que son poco adecuadas para procesadores de texto o aplicaciones que generan salidas impresas.
- Si están facilitadas por un servidor, se puede dar el caso que estén inaccesible por aplicaciones remotas no capaces de acceder al servidor (incluso que el servidor aparezca como caído en la red)
- Tienen un soporte limitado o nulo para características tipográficas avanzadas.
- No admiten suavizado.

Añadir fuentes a la ruta de fuentes de X:

Cuando haya configurado las fuentes en un directorio y creado el archivo correspondiente, deberá de añadir las fuentes a la ruta de fuentes de X editando la sección *Files* del archivo de configuración X.

Si su sección *Files* contiene las líneas *FontPath* que hacen referencia a *unix:/7100*, *unix:/-1* o *tcp/nombreservidor.dominio.edu:7100* su sistema estará configurado para obtener fuentes básicas de un servidor de fuentes X.

Para añadir su nuevo directorio de fuentes a la ruta de fuentes, duplique una de las líneas *FontPath* existentes y cambie la especificación del directorio para que apunte al nuevo. Si desea que sus nuevas fuentes invaliden las fuentes existentes, coloque el nuevo directorio al principio de la lista. Podremos añadir **:unscale** tras el nombre del directorio para indicar a X que utilice la fuente solo si coincide exactamente con el tamaño de fuente solicitado, es decir que no intente escalar las fuentes de ese directorio.

Para probar las nuevas fuentes el modo más fiable de hacerlo es cerrar X y reiniciarlo. Hay métodos más rápidos pero pueden generar algún error, que es añadir la ruta de fuentes a un sistema en ejecución con el programa **xset**.

```
$ xset fp+ /el/directorio/fuentes
$ xset fp rehash
```

Donde:

fp+ indica que se añada el directorio al inicio del listado y **rehash** que se genere una nueva lista de directorio.

Configurar un servidor de fuentes

Algunos sistemás siguen usando servidores de fuentes, pero la mayoría han abandonado esta práctica. Para añadir un servidor de fuentes, primero deberemos de instalar las fuentes en el sistema como indicamos anteriormente y después modificaremos el archivo de configuración del servidor de fuentes (*/etc/X11/fs/config* o similar) donde indicaremos mediante la palabra '**catalogue=**' todos los directorios de fuentes, uno por cada línea y separados por coma '**,**' menos el último. A continuación reiniciaremos el servidor:

```
$ /etc/init.d/xfs restart
```

Configurar fuentes Xft

Xft está basado en parte en la biblioteca *FreeType*, biblioteca usada para renderizar fuentes *TrueType* y *Type 1*. Es un sistema basado en el cliente, admite el suavizado de fuentes y otras funcionalidades avanzadas.
Permite compartir los mismos directorios de fuentes para ambos sistemás es decir si ha preparado un directorio de fuentes como describimos anteriormente, puede añadirlo a *Xft*. Abra el archivo */etc/fonts/local.conf*, */etc/fonts/fonts.conf* o similar y busque la línea *<dir>/directorio/de/fuente</dir>*, duplíquela y modifique el directorio, apuntando al nuevo.

Nota: Si no existen líneas *<dir>directorio</dir>* tendrá que crearla justo antes de la línea *</fontconfig>*

Escriba **fc-cache** como root para que Xft recorra sus directorios de fuentes y cree archivos de índice similares a *fonts.dir*.

ADMINISTRA ACCESOS GUI

X es una GUI con acceso a red. Utiliza el protocolo **XDMCP**, por lo que podría administrar tanto la pantalla del servidor X local, como escuchar las conexiones de un servidor X remoto.

Existen varios métodos para iniciar el servidor *XDMCP*. Los dos más comunes son iniciarlo de manera más o menos directa desde init, a través de una entrada en /etc/inittab o sus archivos de configuración auxiliares, o bien iniciarlo como parte de un conjunto de scripts de inicio de un modo de ejecución. Podríamos iniciar o parar un servidor *XDMCP* con los scripts SysV */etc/init.d/xdm*, *kdm*, *mdm* o *gdm*.

La mayoría de las distribuciones definen un servidor *XDMCP* de dos modos:

1. Selección mediante archivo de configuración (*/etc/sysconfig/desktop*), definiendo la variable **DISPLAYMANAGER=/etc/xdm** o **DISPLAYMANAGER=xdm**
2. Selección mediante Script de inicio: SysV (*xdm*, *kdm*, *gdm*…), Upstart o Systemd

Nota: Linux Mint define en su archivo de arranque */etc/init.d/mdm* la variable **DEFAULT_DISPLAY_MANAGER_FILE** que apunta al archivo */etc/X11/default-display-manager* cuyo contenido referencia al binario **/usr/sbin/mdm**

Configurar servidores XDMCP

En Linux son habituales cinco servidores *XDMCP* (*X Display Manager Control Protocol*):

- **XDM** (*X Display Manager*): Más antiguo y con menos funcionalidades. Acepta nombres de usuario y contraseñas pero no permite a los usuarios realizar otras acciones, como por ejemplo escoger que entorno de escritorio ejecutar (esto se configura a través de los archivos de acceso del usuario). Su archivo de configuración principal es */etc/X11/xdm/xdm-config*. Podemos configurar el sistema para que responda a las peticiones de acceso remoto prestando atención a la línea *DisplayManager.requestPort: 0* (en este caso, configurado para NO aceptar conexiones remotas). Para configurar *xdm* como servidor de acceso remoto, deberíamos cambiar 0 por 117 (puerto de *XDMCP*) y reiniciar *xdm*. Podemos controlar quien accede de forma remota al servidor xdm y de que modo configurándolo en el archivo */etc/X11/xdm/Xaccess*, por ejemplo,

```
*.mydomain.es
miequipo.otrodominio.es
*mydomain.es CHOOSER BROADCAST
```

En las líneas anteriores estamos indicando que permitimos el acceso a todos los equipos del dominio "*mydomain.es*", al equipo "*miequipo.otrodomonio.es*" y solo los equipos de "*mydomain.es*" podrán solicitar un selector (una visualización de los sistemás locales que aceptan conexiones *XDMCP*)

Existen diferentes archivos de configuración importantes relacionados con **xdm**. Cuando ejecutamos **startx*** este pasa a *xinit* el shell script **/etc/X11/xinit/xinitrc**, en el cual se pueden definir aspectos globales de configuración X, como poner un fondo, hacer que se lancen determinadas aplicaciones al inicio, iniciar el controlador de ventanas o incluso simplemente referenciar a otro archivo de configuración. El contenido de *xinitrc* puede ser anulado si existiese el archivo **~/.xinitrc**, en que se configuran los aspectos de X a nivel local (de usuario). Podría ocurrir como comentábamos que *xinitrc* referencie a otro archivo como **/etc/X11/Xsession**. El archivo *Xsession* es otro script que realizará tareas de configuración e inicializará variables (similares a las variables de entorno pero que solo son aplicadas a programás basados en X) que apuntarán a otros archivos, realizándose así una configuración modular de X. Estos archivos de configuración pueden ser algunos como:

Xaccess: Donde se controla el acceso al servidor remoto

Xsession.options: Opciones globales de configuración

Xsession-errors: Normalmente se encuentra oculto en el home del usuario

Xservers: Servidores locales de *xdm*

Xsetup: Script que se ejecutará cuando se inicia *xdm*

Xstartup: Script que se ejecutará cuando un usuario acceda a XWindows.

Xresource: Se usa normalmente para cambiar el tamaño de las fuentes, colores e incluso conseguir bonitos efectos gráficos. Por ejemplo, si quisiéramos modificar el texto mostrado por *XDM* podríamos hacerlo modificando el recurso *xlogin*greeting*, el fondo de aplicaciones X, como el de la terminal con *XTerm*Background: black*, la barra de desplazamiento de la terminal *Xterm*scrollBar True*, etc…

(*****) Puede ser una buena idea echarle un vistazo a este script de inicio, en él además de llamarse al script *xinitrc*, se definen variables de importante valor como, la terminal por defecto para el usuario (*defaultclient=/usr/bin/term*), el servidor por defecto (*/usr/bin/X*), argumentos por defecto para el cliente o servidor si se necesitasen (*defaultclientargs=* y *defaultserverargs=*, respectivamente), el display por defecto (*defaultdisplay=":0"*) o la habilitación de autenticación de X (*enable_xauth=1*)

Nota: Estos archivos de configuración, como ya vimos con el archivo *Xaccess*, nos permiten el uso de comodines, por ejemplo '*****' para referenciar a '*todos*'.

- **KDM**: Asociado con el proyecto KDE. KDM está basado en XDM por lo que comparte muchas de sus opciones de configuración. La ubicación de los archivos de configuración de KDM es impredecible, en ocasiones utiliza archivos de configuración de XDM (/etc/X11/…) y otras se encuentran en ubicaciones extrañas como /etc/kde//kdm o /usr/share/kd4/. KDM amplia XDM permitiendo a los usuarios seleccionar un tipo de sesión cuando acceden. La mayoría de estas opciones se encuentran definidas en el archivo kdmrc. Este archivo puede encontrarse seccionado con etiquetas como por ejemplo:**[General]**: Contiene las opciones globales como, *ConfigVersión* (no modificar, limpia las actualizaciones de versión automática), *ConsoleTTYs* (cuando pasamos al modo consola **kdm** monitorizará todas las líneas aquí presente), *LockPidFile* (por defecto en *True*, kdm utiliza el bloqueo de archivo para guardar múltiples administradores de pantallas en ejecución).**[Xdmcp]:** Aquí se ajustan opciones relacionadas con el trabajo en red. De estar en en *Enable* se escucharán peticiones remotas, *Port* (indica el número de puerto, **117** para *XDMCP*).**[Shutdown]:** Contiene las opciones globales referentes al apagado del sistema.**[X-*-Greeter]:** Contiene las opciones relativas a la pantalla de 'Bienvenida'. *ColorSheme* (configura el color de la pantalla de Bienvenida), *GreetString* (cabecera de la pantalla de Bienvenida), *GreetFont* (la fuente utilizada para la cabecera de Bienvenida), en definitiva, todo lo relacionado con la pantalla de bienvenida.

- **GDM**: Gestor de pantallas de *GNOME*. Se ejecuta en 2º plano, dirige las sesiones de X y el inicio de sesión de usuario. No utiliza ningún código de *XDM* y es compatible con *XDMCP*. Puede ser instalado mediante el paquete **gdm** de los repositorios oficiales. Deberá de ser activado por *systemd*, mediante su script de inicio SysV o archivo de configuración *Upstar* (*.conf*). Desde la versión 2.28 no dispone de su herramienta de configuración '*gdmsetup*', debido a que ha sido integrado con el resto de *GNOME*. Su archivo de configuración se encuentra en */etc/gdm/custom.conf* en el que podremos efectuar cambios como iniciar sesión automáticamente, iniciar sin contraseña, cerrar sin contraseña, cambiar la sesión predeterminada de inicio… Se encuentra seccionado como el de kdm.

Iniciar sesión automáticamente:

```
[daemon]
 # activar acceso automático para el usuario
 AutomaticLogin=nombredeusuario
 AutomaticLoginEnable=True
```

Añadir una demora en el ingreso:

```
TimedLoginDelay=1
```

Para cambiar la sesión predeterminada de GDM podemos crear el archivo **~/.dmrc** con el siguiente contenido para pre-establecer cinnamon:

```
[Desktop]
 Session=cinnamon
```

- **MDM**: Es un fork de *GMD2* utilizado por Linux Mint, por lo que su configuración está basada en *GMD*.
- **LightDM** (*Light Display Manager*): Compacto y compatible con varios entornos de escritorio. Es el gestor de sesión que aspira a ser el estándar para el sistema de ventanas de *X11* y de *Wayland*. Se puede instalar desde los repositorios oficiales. Será necesario instalar tras este un *greeter* (*interfaz de usuario para LightDM, login screen*), se recomienda **lightdm-gtk-greeter** o **lightdm-kde-greeter**. Una vez instalado es necesario activarlo mediante *systemd*.

La instalación de lightDM se ubica bajo el directorio **/etc/lightdm**, donde encontraremos gran parte de los archivos de configuración como */etc/lightdm/lightdm.conf.d/*.conf* o */etc/lightdm/lightdm.conf* aunque es posible encontrar archivos bajo */usr/share/lightdm/lightdm.conf.d/*.conf* (no se recomienda editar estos últimos, serán anulados por los del directorio /etc/lightdm). Podremos entonces modificar la configuración como por ejemplo habilitar el *autologin*, *cambiar iconos*, *cambiar la imagen* o el *color de fondo*, *deshabilitar la sesión de invitados*, etc… mediante los archivos de configuración de /etc/lightdm/ o los de configuración del propio *greeter* **/etc/lightdm/lightdm-gtk-greeter.conf**

Imaginemos que deseamos anular la configuración de inicio de sesión por defecto en Ubuntu la cual es proporcionada desde */usr/share/lightdm/lightdm.conf.d/50-ubuntu.conf* . Deberemos entonces de crear un nuevo archivo de configuración */etc/lightdm/lightdm.conf.d/50-myconfig.conf* con el siguiente contenido:

```
[SeatDefaults]
user-session=mysession
```

Existe un archivo adicional para configuraciones de inicio de sesión de los usuarios que es **/etc/lightdm/users.conf**, pero deberemos de tener en cuenta que este será anulando si el demonio '**accountsservice**' se encuentra en ejecución, podemos comprobarlo de la siguiente manera:

```
ps -aef | grep accountsservice
```

Algunos ejemplos:

- Vamos a suponer que queremos deshabilitar el inicio de sesión para la cuenta de invitados (*guest*):

```
[SeatDefaults]
```

```
allow-guest=false
```

- Ocultar las cuentas de inicio de sesión de los suarios del sistema en la pantalla de login:

```
[SeatDefaults]
greeter-hide-users=true
```

- Permitir introducir un nombre de usuario manualmente para iniciar sesión:

```
[SeatDefaults]
greeter-show-manual-login=true
```

- Usar un usuario predeterminado para iniciar sesión:

```
[SeatDefaults]
autologin-user=username
```

- Cambiar la sesión por defecto:

```
[SeatDefaults]
user-session=name
```

Nota: El nombre de la sesión tiene que ser alguno de los instalados en: */usr/share/xsessions/*.desktop*

- Igualmente podríamos usar el archivo de configuración **/etc/lightdm/lightdm-gtk-greeter.conf** para cambiar la imagen de fondo de inicio de sesión. Deberemos de modificar la variable *background* del archivo tal que así:

```
background=/usr/share/pixmaps/black_and_white_photography-wallpaper-
1920x1080.jpg
```

Nota: Para modificar el color, bastará con darle a esta variable un valor hexadecimal de color.

- Si quisiéramos modificar la configuración para XDMCP, debemos de hacerlo desde el archivo de configuración **/etc/lightdm/keys.conf**

Lightdm tiene su propia herramienta de configuración de línea de comandos:

$ dm-tool –help

Alguna de las opciones que nos permite este comando son: cambiar entre sesiones de usuarios (incluyendo a una sesión de invitados), bloquear la sesión, listar las sesiones iniciadas, etc…

Igualmente podemos hacer uso del comando ***$ lightdm [opciones]***

Nota: Como siempre en la sección de comandos, obtendremos mejor información.

Referencias:

https://wiki.ubuntu.com/LightDM

https://wiki.debian.org/LightDM

https://wiki.ubuntu.com/LightDM

Utilizar clientes X remotos

X es un servidor que se ejecuta en el ordenador del usuario. Los clientes X son los programás que ejecuta el usuario como xterm, kmail, libreoffice, etc… Cuando se utiliza X para un acceso remoto, el servidor X se ejecuta en el ordenador del usuario mientras que los clientes X se ejecutan en el sistema remoto. Normalmente, Linux está configurado de manera que su servidor X responda solo a las peticiones de acceso local como medida de seguridad. Para solucionar este problema deberemos de realizar cambios que variarán dependiendo de la distribución y en servidor X, por ejemplo:

Si usamos **GMD** revisaremos el archivo de configuración (*gmd.conf*, para versiones antiguas) y buscaremos la línea *DisallowTCP=true* cambiándola a **false**. Para versiones más modernas de *GDM* editaremos su archivo de configuración (*custom.conf*) y añadiremos la línea *DisallowTCP=false* en la sección **[Security]**

Para los servidores **KDM** y **XDM** localizaremos el archivo *Xservers* y lo editaremos, eliminando el parámetro **-nolisten tcp** de la línea de configuración que comience con **:0** permitiendo así la escucha de conexiones remotas.

Para sistemás **openSUSE** deberemos de editar el archivo '*displaymanager*' ubicado normalmente bajo /etc/sysconfig y estableceremos en '**yes**' la opción **DISPLAYMANAGER_XSERVER_TCP_PORT_6000_OPEN**

Nota: Puede que incluso en el propio script de inicio '**startx**' debamos de eliminar el valor **-nolisten tcp** de alguna variable, puede que de *defaultserverargs*. Esto no suele ser muy común pero deberemos de tenerlo en cuenta.

Tras realizar estos cambios será necesario reiniciar X

Ejemplo de conexión remota a X:

Ordenador **A** aloja programás importantes como *eclipse, SQL Developer, procesadores de texto*,etc.. El ordenador **B**, es un PC 'estándar' con un teclado y un monitor. Será *B* el PC en el que el usuario esté trabajando, es decir los programás se ejecutarán en el PC *A*,

pero serán mostrados por la pantalla de *B*. A efectos visuales podriamos decir que es el PC *B* el que está realizando el groso del trabajo, pero no es así, realmente es *A* quien ofrece las aplicaciones. Esto podemos conseguirlo a través de varias formás, entre ellas podríamos destacar, la conexión *SSH* con X, *xhost* o *VNC*.

- El protocolo **SSH** es una herramienta útil de acceso remoto para realizar conexiones de red seguras, es decir, transportar otro protocolo a través de su propia conexión codificada. La codificación de *SSH* quizás ralentice el acceso a X, aunque si activa las funciones de compresión de *SSH* puede que reduzca la gravedad de este problema.
  Este método requiere que se configuren ciertas opciones, concretamente deberemos de pasar la opción **-X** o **-Y** al programa cliente ssh o establecer la opción *ForwardX11* o *ForwardX11Trusted* en 'yes' en el archivo de configuración del cliente ssh */etc/ssh_config*. Además de esto, deberemos de establecer la opción *X11Forwarding* en 'yes' en el archivo de configuración del servidor */etc/sshd_config*. Estas funciones activarán el reenvío de X por ssh.
- El programa **xhost** se utiliza para agregar y eliminar nombres de hosts o de usuarios a la lista de 'permitidos' para realizar conexiones con el servidor X. En el caso de los hosts, proporciona una forma pobre de control de privacidad y seguridad.

Para usar este método necesitaremos indicar al cliente, que acepte los datos que el servidor proporcionará:

```
$ xhost +nombreservidor
```

Lo siguiente será comprobar la variable **DISPLAY** en el servidor, por lo que accederemos por ssh a 'nombreservidor' y la setearemos correctamente:

```
$ ssh nombredservidor
$ export DISPLAY=pccliente:0.0
```

Con este comando estaremos indicando a la máquina servidora que utilice a pc cliente para mostrar sus X

A continuación podremos ejecutar desde esa misma terminal los comandos que inicien los programás clientes y estos, serán mostrado por nuestra pantalla, con lo que delegaremos el mayor consumo de CPU al servidor.

Una vez terminemos, lo mejor será quitar el permiso para el host servidor en el pc cliente con la siguiente orden:

```
$ xhost -nombreservidor
```

- **VNC**

Otra opción para ejecutar los programás X es utilizar el sistema *VNC* (*Virtual Network Computing*, Cálculo virtual de red). *VNC* ejecuta un servidor X en el ordenador servidor, mientras que el cliente tendrá su propia aplicación *VNC* cliente que contactará de forma remota con el servidor *VNC*. Esta forma de conexión puede ser beneficiosa en determinadas situaciones, como cuando se intenta acceder a un sistema desde detrás de ciertos tipos de corta-fuegos. Al ser un protocolo independiente de la plataforma se puede controlar un sistema *Windows* o *Mac* desde *Linux*.

ACCESIBILIDAD DE X

La utilidad **AccessX** es un programa antiguo (actualmente en desuso) que funciona en cualquier entorno de escritorio. *AccessX* y los paneles de control de los entornos de escritorio proporcionan opciones de accesibilidad. Este paquete no está disponible en muchas distribuciones y sus funciones se han integrado en paneles de control de los entornos de escritorio.

Teclado en pantalla

GOK (*GNOME On-Screen Keyboard*, Teclado en pantalla de GNOME), como su propio nombre indica es un teclado virtual, pero además proporciona herramientas que crean accesos directos para varias funciones de ratón, menús y barras de herramientas de otros programás.

Herramientas de aumento

Agrandan partes de la pantalla (normalmente el área que hay alrededor del ratón). Una de las lentes para pantallas es **KMag**.

Sintetizadores de voz

- **orca**: lector de pantalla integrado en *GNOME* 2.16 y posteriores
- **Emacspeak**: Similar a Orca en muchos aspectos pero amplia sus funcionalidades

Pantallas de Braille

BRLTTY es un demonio de Linux que redirecciona la salida de la consola en modo texto a una pantalla de Braille. Incluye características como desplazamientos, terminales virtuales e incluso un sintetizador de voz.

AJUSTAR LA HORA Y LA CONFIGURACIÓN LOCAL

Definir su zona horaria

Cuando nos comunicamos con otros ordenadores es aconsejable la definición de la zona horaria en cada equipo o incluso en cada sesión de cada equipo para '*situarnos*' mejor en el ambiente de trabajo, por ejemplo cuando estamos trabajando sobre un servidor localizado en Chicago y nosotros nos encontramos en nuestra oficina de España, podría resultar cómodo ver la hora en nuestro sistema y posteriormente en el servidor y comprobar que estamos bajo la misma franja horaria, es más, será conveniente incluso utilizar la hora **UTC**, ya que es la utilizada para las marcas de tiempo en los nuevos archivos y directorios. De este modo se establecerán correctamente las marcas horarias de sus archivos y evitaremos extraños problemás temporales al intercambiar datos con estos servidores.

¿De donde surge el estandar UTC?

Cuando la comunicación entre distintos puntos distantes de la tierra se hizo algo elemental y necesario, apareció la necesidad de tener una hora común, por lo que se definió un estándar mundial de medición horaria. Este estándar se llama hora universal UT o **UTC** (*Coordinated Universal Time*) formalmente conocido como '*Greenwich Mean Time*' o **GMT**, debido a que se utiliza como hora local en *Greenwich*,Inglaterra. Cuando personas con diferentes horas locales necesitan comunicarse, pueden expresar el tiempo en hora universal, para que no exista confusión acerca de cuando deben suceder las cosas.

¿Porque existe el horario local o 'time zone'?

El mediodía es el momento del día en el cual el Sol se encuentra en la posición más alta debido a la rotación de la Tierra. El momento del mediodía sucede en diferentes momentos en diferentes lugares. Esto nos conduce al concepto de hora local. Existen 24 zonas horarias en el planeta.
Es posible pensar que todos los lugares en donde el mediodía sucede en el mismo momento tienen la misma zona horaria, políticamente no siempre esto es posible. Por diversas razones, varios países adoptan el "horario de verano" (*daylight savings time*) para atender las demandas económicas.

Linux tiene un paquete de zonas horarias que reconoce todas las zonas horarias existentes, y puede ser fácilmente actualizado cuando las reglas cambian. Todos los usuarios pueden seleccionar la zona horaria apropiada.

Reloj de Hardware y reloj de Software

Una computadora tiene un *reloj de hardware* alimentado por una batería y que puede ser configurado directamente desde su *BIOS* o *UEFI*. La batería asegura que el reloj siga funcionando incluso cuando la computadora se encuentra sin suministro eléctrico. Este reloj suele configurarse con la hora *UTC*.

Por otra parte el kernel de Linux mantiene la fecha y hora de manera independiente al reloj de hardware, ya que acceder continuamente al reloj de hardware es lento y complicado. Durante el arranque, el kernel de Linux fija su hora basándose en la hora *UTC* del hardware, de manera que *Linux* posteriormente pueda aplicar sobre esta, el horario de verano o **DST** (*Daylight Savings Time*) y configurar así su hora local o *zone time*.

Nota: Otros OS como Windows, asumen que la hora del reloj de hardware es la hora local, por lo que puede existir un desfase horario si tenemos un arranque dual con ambos OS.

Configurar y visualizar la hora en un equipo Linux

En Linux, la zona horaria del sistema es determinada por */etc/localtime*, que en ocasiones es un enlace simbólico que apunta a un archivo de datos de zona horaria que describe la zona horaria local. Los archivos de datos de zonas horarias están ubicados bajo el directorio */usr/lib/zoneinfo* o */usr/share/zoneinfo*. Para cambiar la zona horaria, vale con enlazar */etc/localtime* y el archivo de configuración */usr/share/zoneinfo/UE/Amsterdam* (por ejemplo) o */usr/lib/zoneinfo/UE/Amsterdam*.

En caso de que sea un archivo binario y no un enlace simbólico podemos renombrar el actual:

```
$sudo mv /etc/localtime /etc/localtime.original
```

Y a continuación crear el enlace simbólico:

```
$sudo ln -s /usr/share/zoneinfo/UE/Amsterdam /etc/localtime
```

De esta manera ajustaremos la zona horaria para todo el sistema.

Si por el contrario lo que queremos es cambiar la zona horaria para una determinada sesión (usuario) podemos usar herramientas como: **tzsetup**, **tzselect**, **tzconfig** o similar. Por ejemplo en nuestro caso (*Linux Mint 17.1 Rebecca*) tenemos *tzselect*. Si invocamos este comando nos aparecerá un menú en el que primeramente aparecerán los continentes y unos números para seleccionar uno de ellos, a continuación igualmente aparecerá una lista con los países, y finalmente una breve aclaración de lo que va a ser modificado:

```
The following information has been given:
Iraq
Therefore TZ='Asia/Baghdad' will be used.
 Local time is now: Fri Jan 30 13:00:52 AST 2015.
```

```
Universal Time is now: Fri Jan 30 10:00:52 UTC 2015.
Is the above information OK?
1) Yes
2) No
```

Una vez seleccionado la opción 1) 'yes' nos aparecerá el contenido de la variable **TZ**, que podremos pegar en los archivos de inicio de sesión de los usuarios '*~/.profile*'

```
TZ='Asia/Baghdad'; export TZ
```

A parte de la variable *TZ*, nos aparece el *timezone*, en nuestro ejemplo anterior '*Asia/Baghdad*'. Si hemos cambiado la zona horaria para todo el sistema, sería interesante cambiar el valor de */etc/timezone* (Debian) o */etc/sysconfig/clock* (RedHat) por Asia/Baghdad, ya que este es útil para detectar rápidamente que timezone tenemos definido y para evitar confundir a las herramientas de configuración superior.

También podemos ver los *timezone* disponibles con el comando **timedatectl**. Por ejemplo si usamos:

```
$ sudo timedatectl list-timezones
```

Nos listará todos los *timezones* disponibles. Si queremos setear uno en concreto:

```
$ sudo timedatectl set-timezone [timezone]
```

Veremos como configurar el *tiempo*, habilitar o deshabilitar *ntp*, ajustar *RTC* o ver la configuración de tiempo actual, con este mismo comando en el apartado de comandos de este capítulo.

Para visualizar la hora actual usamos el comando **date**. Este comando desplegará una línea similar a

```
vie ene 30 11:15:15 CET 2015
```

Es comveniente cuando cambiamos de zona horaria comprobar que el conjunto de caracteres 'CET, *Central European Time*' ha cambiado por el actual. Podemos conocer el resto de códigos en la página http://www.timeanddate.com/time/zones/

Aclaraciones: Esté capítulo contiene extractos de información del libro "*Guía Para Administradores de Sistemás GNU/Linux: Versión 0.8*"

Mantener la hora en el sistema con servidores NTP

Tanto el reloj de hardware como el de software son poco fiables en *x86* y *x86-64* estándar, ambos tienden a desajustarse levemente, por lo que no es extraño que en un mes o dos después de ser ajustados acaben con una diferencia de varios minutos con respecto al tiempo correcto. Para salvaguardar esta situación podremos emplear varios

métodos, como por ejemplo configurar la hora manualmente cada cierto tiempo o incluso automáticamente a través de tareas *cron* (veremos el uso de *cron* en el *Capítulo 7*).

Podremos configurar la hora del reloj de software con el comando **date**, cuya sintaxis es la siguiente:

```
date [-u | --utc | --universal] [MMDDhhmm[[CC]AA] [.ss]]
```

El formato para *date* debe contener como mínimo un *mes*, un *día*, una *hora* y un *minuto* (**MMDDhhmm**), pudiendo añadir un año de dos o cuatro dígitos e incluso los segundos. La hora debe de ser configurada en formato 24 horas.

- Ajustar la hora a las 5:45 AM del 15 de Enero del 2015:

```
# date 011505452015
```

date asume que esta hora es especificada para hora local por lo que si en realidad lo que deseábamos era ajustar la hora universal *UTC*, deberíamos de haber incluido alguna de las 3 opciones disponibles: **-u**, **–utc** o **–universal**.

Para ajustar el reloj de hardware podremos hacerlo por ejemplo con la utilidad **hwclock** o bien desde el firmware del equipo.

Con el comando *hwclock* podremos ajustar el reloj de hardware a partir de la hora del reloj de software o viceversa. *hwclock* permite un argumento como opción, con el que podremos ajustar la hora de varias maneras o imprimir la hora actual.

- Ajustar el reloj de hardware a partir del reloj de software:

```
# hwclock --systohc
```

- Ajustar el reloj de software a partir del reloj de hardware:

```
# hwclock --hctosys
```

Esta opción suele ser común en los script de inicio para ajustar el reloj al inicio del sistema.

- Mostrar la hora del reloj de hardware

```
# hwclock -r | --show
```

- Configurar de forma manual la hora del reloj de hardware:

```
# hwclock --set --date=011505452015
```

Usamos el mismo formato que para el comando date

- Guardar la hora como UTC o como hora local: opciones **–utc –localtime**

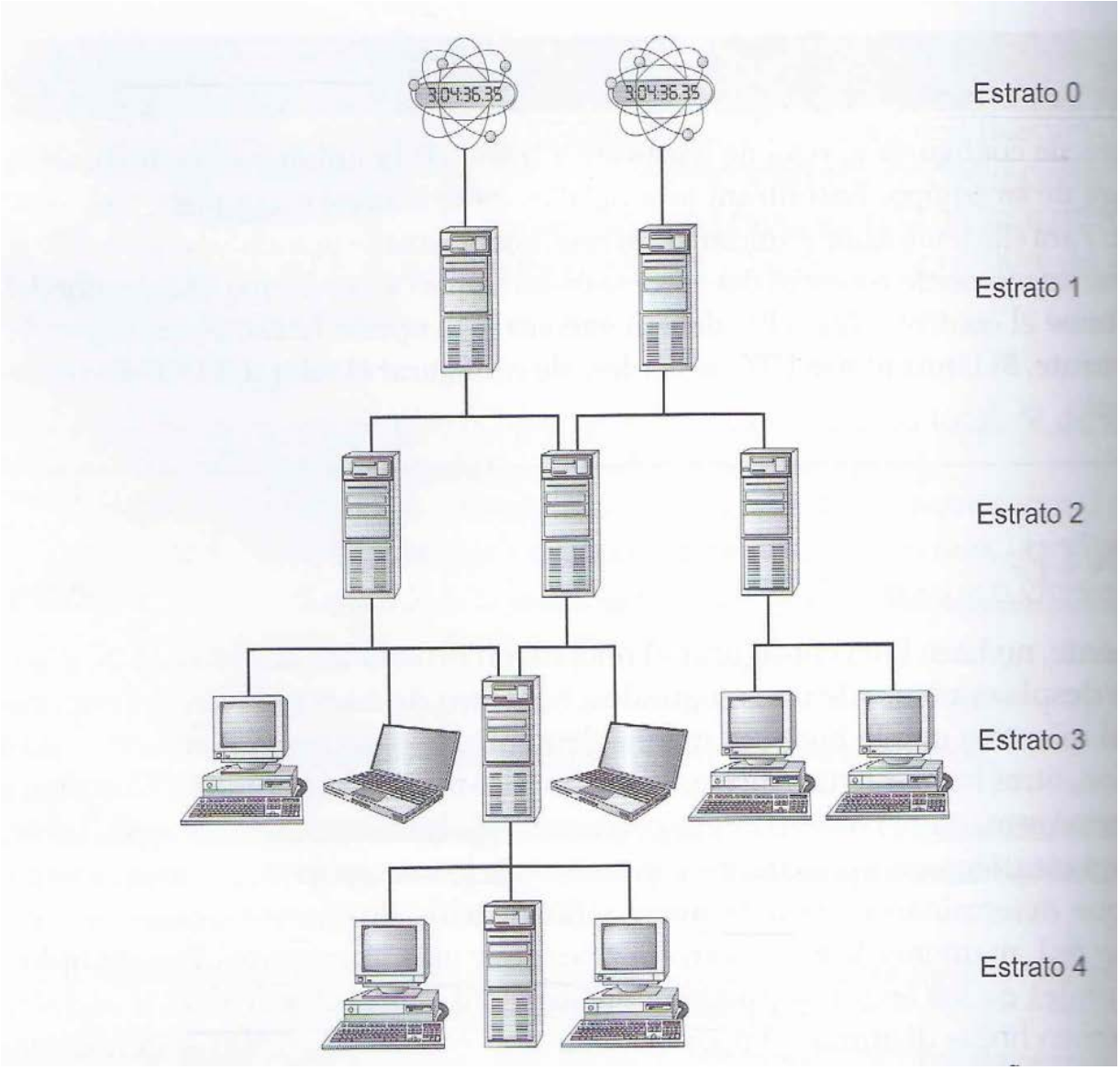
No es habitual utilizar el comando *hwclock*, quizás de vez en cuando para evitar que el reloj de hardware se aleje demasiado de la hora exacta, aunque esto lo hacen algunas distribuciones de manera automática como parte del procedimiento de cierre del sistema.

Hasta aquí hemos visto como ajustar la hora de los relojes de hardware y software mediante el uso de los comandos *date* y *hwclock*, pero es importante mantener totalmente la precisión de la hora en el sistema, especialmente en aplicaciones científicas, empresariales e industriales, donde las marcas temporales de los archivos pueden volverse confusas si los servidor de archivos y sus clientes tienen horas distintas. Existen protocolos de seguridad y autenticación que incrustan marcas temporales en sus paquetes y se basan en éstas para el funcionamiento normal del sistema. Podemos encontrarnos con que un servidor de *Directorio Activo* o la suite de seguridad *Kerberos* no funcionen, impidiendo el acceso a archivos o directamente la autenticación dentro del sistema.

Para evitar estos problemas se recomienda ajustar la hora a través de la red, de manera que todos los servidores y ordenadores de una empresa mantengan sus relojes actualizados al 'unísono'. Podremos conseguir esta configuración a través del protocolo **NTP** (*Network Time Protocol*). Para un cliente o equipo concreto, bastará con tener el servicio *ntp* instalado y en ejecución, siempre y cuando este debidamente configurado. En la siguiente sección veremos como configurar el servicio *ntp*.

Fundamentos de NTP

Una de las herramientas más populares para configurar la hora de la red es **NTP**. Este protocolo crea una jerarquía por capas de fuentes horarias. En la parte superior de la estructura, se encuentran una o varias fuentes horarias que normalmente son relojes atómicos o receptores de radio que obtienen sus horas de señales horarias basadas en relojes atómicos. Esta parte superior de la jerarquía la forman los llamados *servidores temporales de estrato 0* (*stratum 0*). No podremos acceder directamente a estos servidores , excepto desde los *servidores temporales de estrato 1*. Estos servidores de estrato 1 ejecutan servidores *NTP* que distribuyen la hora a *servidores de estrato 2*, que a su vez distribuyen la hora a *servidores de estrato 3* y así sucesivamente.



Cada incremento en el número de estrato reducirá ligeramente la precisión de la señal horaria, pero no demasiado. Un reloj de un servidor temporal de estrato 4 tendría una precisión con una desviación inferior al segundo.

NTP funciona midiendo el tiempo de ida y vuelta de los paquetes entre el servidor y el cliente. Los dos sistemas intercambian paquetes con marcas temporales incrustadas; el cliente ajusta su hora para que esté sincronizada con la marca temporal del origen pero añade una pequeña cantidad de tiempo para tener en cuenta el retardo del trayecto de ida y vuelta.

La diferencia entre ajustar la hora de forma manual o hacerlo de forma automática mediante *NTP*, es que como ya sabemos el reloj tiende a desfasarse con el tiempo por lo

que de configurarlo manualmente deberemos de estar pendientes cada cierto tiempo de que el desfase no se haga mayor. Aquí es donde destaca *NTP* el cual posee un archivo normalmente **/var/lib/ntp/ntp.drift** o **/etc/ntp.drift** que alberga información sobre las imprecisiones del reloj de software y, por tanto, se puede utilizar para corregirlas. Un servidor *NTP* aunque funcione solo como cliente, consultará periódicamente sus sistemás fuentes para mantener correctamente configurada la hora del sistema y para actualizar el archivo *ntp.drift* impidiendo con esto, que el desfase aumente de forma considerable.

Nota: El principal programa de servidor *NTP* de *Linux* funciona tanto de servidor como de cliente.

Configurando un servidor NTP

Si tiene una red de trabajo, por ejemplo una empresa en la que coexisten servidores de archivos, servidores de autenticación, de control de versiones, etc… y ordenadores clientes, lo mejor será configurar un servidor temporal *NTP* para que el resto de servidores y ordenadores de la red obtengan la hora a través de este. Para configurar este servidor temporal deberemos de conocer sus archivos de configuración y algunas características de *NTP* que harán más efectiva la sincronización horaria dentro de la empresa. Por ejemplo, cuando configure un servidor *NTP*, lo primero que deberá de hacer es sincronizar este servidor con alguno de los servidores de estrato o fuentes temporales. Puede que piense que localizar un servidor *NTP* con un número de estrato bajo (estrato 1) es lo ideal. Quizás sea lógico y cierto pensar que cuanto más cerca estemos del reloj atómico, mejor será la precisión horaria dentro de la oficina, pero en la mayoría de los casos el mejor método es sincronizar con un sistema de estrato 2 o inferior, ya que con esto reduciríamos la carga en los servidores de estrato 1, lo que mejoraría el rendimiento global de la red *NTP*.

Antes de comenzar con la configuración deberemos de elegir el candidato para servir la hora en la oficina. Este ordenador/servidor no necesariamente tiene que ser potente, pero si que debe de tener acceso permanente a la red. Una vez seleccionado el equipo, instalaremos (en caso de que no lo esté) el paquete **ntp** o **ntpd** de los repositorios oficiales. Cuando el sistema esté equipado con este paquete, pasaremos a editar su archivo de configuración para determinar las características y opciones de nuestro servidor *NTP*. El archivo de configuración suele ser **/etc/ntp.conf** y las líneas a las que mayor atención deberemos de prestar serán a las que empiezan por la palabra ‘**server**‘ ya que son las que indicarán con que servidores temporales queremos sincronizar. Lo normal es tener entre 2 y 3 líneas con servidores temporales, ya que de estar uno caído u ofreciendo una hora desfasada, automáticamente *NTP* sincronizará con otro servidor de la lista. Podremos localizar servidores temporales para usarlos como servidores de estrato superior al nuestro, es decir con los que sincronizaremos nuestro servidor *NTP* de varias formás, por ejemplo:

- **Mediante nuestro ISP**: Muchos proveedores de servicios de Internet (*ISP*) trabajan con servidores *NTP* que suelen estar muy cerca en términos de red, lo que los convierte en buenos candidatos.
- **El servidor *NTP* de la distribución que usemos**: Si nos encontráramos cerca de estos servidores (en términos de red), podrían ser una buena opción. A decir verdad esto casi nunca ocurre pero igualmente es bueno que lo tengamos en mente.
- **Listas de servidores NTP públicos**: Podremos encontrar estos servidores en http://support.ntp.org/bin/view/Servers/WebHome. Tendremos que localizar el más cercano en termino de red y además es posible que necesitemos permisos para utilizarlos.
- **Banco de servidores NTP públicos**: El subdominio *pool.ntp.org* está dedicado a servidores que funcionan voluntariamente como servidores *NTP* públicos. Para utilizar el *pool* de servidores bastará con utilizar el nombre del subdominio '*pool.ntp.org*' o un host dentro de ese dominio como '*0.pool.ntp.org*'. Suele ser la opción más extendida junto a la del *ISP*.

De los puntos anteriores podrán surgir preguntas tipo *¿Que quiere decir un servidor cercano en términos de red?* o *¿Como compruebo si un servidor esta cerca o lejos en términos de red?*

Un servidor cercano en términos de red es aquel que tiene unos buenos tiempos de respuesta, o más concretamente de ida y vuelta de los paquetes. Podremos comprobar esto con el comando **ping** <*ip del servidor*>. Si obtenemos tiempos elevados lo mejor será descartar el servidor en cuestión.

Una vez hayamos localizado y comprobado los servidores a utilizar, los agregaremos al archivo de configuración.

Es recomendable inspeccionar detalladamente el archivo de configuración para eliminar o modificar entradas no deseables. A continuación vamos a detallar una configuración *ntp.conf* en la que comentaremos muchas de las opciones disponibles para *NTP*, ya sea como cliente o como servidor.

```
$ cat /etc/ntp.conf
# Archivo para el control de desfase horario
driftfile /var/lib/ntp/ntp.drift
# Políticas de acceso para cualquier servidor de tiempo utilizado.
# Suele comenzarse con políticas restrictivas y se va siendo más
# permisivo en líneas posteriores.
# Aceptar todas las conexiones 'default' pero a la vez invertimos
# el significado con 'ignore' que denegará cualquier tipo de paquete
# incluyendo consultas con comandos ntpq y ntpc
restrict default ignore
# Nota: Podríamos haber usado -4 para especificar IPv4 o -6 para IPv6.
# Aceptar todas las conexiones, pero prohíbe modificar la
configuración
# del servidor 'nomodify', en cambio si permite las consultas, aunque
# luego se prohíben estas con el comando 'noquery'.
```

```
# 'notrap' niega el modo 6 de control de mensajes para servicios de
# control de registro remotos. 'nopeer' deniega nuevas asociaciones
# no autentificadas, propias de mensajes broadcast.
restrict default nomodify notrap nopeer noquery
# Nota: Los comandos ntpq y ntpc de poco servirán con una conexión
# como esta, ya que 'noquery' invalida las consultas. Esta línea
# es típica para una configuración cliente.
# Permitir conexiones entre la red local. Pero deniega reconfigurar
# la hora del servidor.
restrict 192.168.0.0 másk 255.255.255.0 nomodify
# Máxima permisividad. En este caso para el equipo local
restrict 127.0.0.1
# Añadir un reloj indisciplinado utilizado para cuando ninguna
# de las fuentes temporales están disponibles.
fudge 127.0.0.1 stratum 10
server 127.0.0.1
# Archivo de claves si acaso fuesen necesarias para realizar
# consultas
keys /etc/ntp/keys
# Lista de servidores de tiempo de estrato 1 o 2.
# Se recomienda tener al menos 3 servidores listados.
server 0.pool.ntp.org iburst
server 1.pool.ntp.org iburst
server 2.pool.ntp.org iburst
# Nota: Cuando un servidor es inalcanzable, iburst hace que se
# envíen 8 paquetes en lugar de 1 (por defecto) con un intervalo
# de tiempo de 2s entre paquetes, algo que se puede modificar.
# Esta opción es solo válida para el comando server y es altamente
# recomendable.
# Permisos que se asignarán para cada servidor de tiempo.
# En los ejemplos, se impide a las fuente consultar o modificar
# el servicio en el sistema, así como también enviar mensaje de
# registro.
restrict 0.pool.ntp.org másk 255.255.255.255 nomodify notrap noquery
restrict 1.pool.ntp.org másk 255.255.255.255 nomodify notrap noquery
restrict 2.pool.ntp.org másk 255.255.255.255 nomodify notrap noquery
# Se activa la difusión hacia los clientes y además estos no
necesitarán autenticación
broadcastclient
disable auth
```

Además de las opciones anteriormente definidas para el comando **restric** existen otras como:

- **kod**: Si el acceso es denegado a un cliente, se envía un paquete '*kiss-of-death*'. Para informar de ello
- **noserver**: El servicio de tiempo es denegado, aun que se puedan realizar consultas al servidor.
- **notrust**: Ignora todos los paquetes *NTP* que no están autenticados criptográficamente
- **limited**: Limitar el número de clientes para un servidor *NTP* de la misma red. Deberemos de usar además
  las opciones **clientlimit** <*limite*> (por defecto 3) y el intervalo de tiempo

considerado como inactivo en
el caso de que sea sobrepasado **lientperiod** <*segundos*> por defecto 3.600s

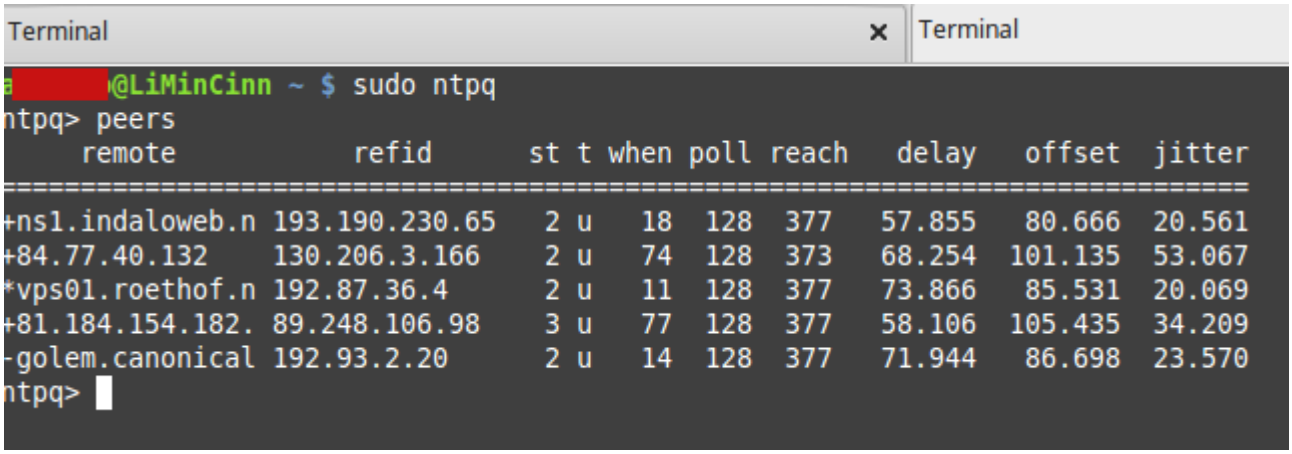
Nota: Podemos obtener más información a través de la página man: ***$ man ntp_acc***

Una vez hayamos configurado el servidor *NTP* será necesario reiniciarlo para que los cambios se hagan presentes. No obstante *NTP* tarda un tiempo en determinar cuáles de las fuentes proporcionan la mejor señal, por lo que es posible que los servidores fuentes no aparezcan como sincronizados hasta pasados unos minutos.

Para verificar que *NTP* funciona, podemos utilizar el comando **ntpq**. La herramienta *ntpq* puede ser utilizada de forma interactiva o bien podemos pasarle opciones directamente. Si pasamos la opción **-i** indicamos que queremos iniciar *ntpq* de forma interactiva (opción por defecto), con la opción **-p** listaremos los servidores de tiempo con los que estamos intentando sincronizar y la opción **-c** nos permite pasarle un comando a *ntpq* (*$ sudo ntpq -c peers*). Si queremos ver las IP de los sistemás con los que sincronizamos en lugar de su nombre *DNS* podremos acompañar la opción *-p* con **-n** (*-np*)

Nota: Podemos acceder a ntpq de forma interactiva e introducir el comando '*help*' para desplegar los comandos aceptados por *ntpq*

Si invocamos a *ntpq* directamente sin opciones, se desplegará un *prompt* (**ntpq**>)
donde podremos teclear los comandos aceptados por *ntpq*, por ejemplo:



La imagen anterior muestra una salida al comando **ntpq> peers** . Hubiésemos conseguido esta misma salida escribiendo directamente *$ sudo ntpq -p.*

Interpretación de la salida ntpq :

- **remote**: Muestra los servidores configurados con los que sincronizar la hora. Dependiendo del metacaracter que preceda al nombre del host tendrá uno u otro significado. El asterisco '*****' indica con que servidor estamos sincronizado, el

resto de servidores con horas adecuadas serán indicados con '+' mientras que aquellos servidores descartado por diferentes razones tendrán caracteres como **–**, **x** o **#**

- **refid**: Referencia los servidores a los que sincroniza cada sistema '*remote*'
- **st**: Muestra el estrato del servidor, en nuestro caso 2 y 3.
- **t**: Los tipos disponibles (**u**: unicast, es el más común. **l**: local, **m**: multicast, **b**: broadcast, **–**: netaddr)
- **when**: Número de segundos transcurridos desde la última respuesta
- **poll**: Intervalo de sondeo en segundos para las fuentes.
- **reach**: Código que indica si ha fracasado o ha tenido éxito (377) la localización de la fuente.
- **delay**: Indica el tiempo de ida y vuelta del paquete de respuesta (en milisegundos)
- **offset**: Indica el desfase de tiempo entre el cliente y el servidor
- **jitter**: Indica la diferencia de tiempo entre dos muestras (en milisegundos)

Nota: Si aparece el *host LOCAL(0)* sabremos entonces que la fuente de referencia de último recurso es el propio reloj del sistema.

Podremos comprobar cuan desviado está el reloj del sistema en segundos basándose en la última vez que contacto con el servidor remoto, con el comando **ntpdc -c loopinfo**

Configurar clientes NTP

La configuración de los clientes es similar a la de los servidores con un par de excepciones:

- Se configuran de modo que hagan referencia al servidor o los servidores *NTP* que se hayan configurado en un entorno local y no a fuentes externas.
- Como medida de seguridad es recomendable asegurarse de que no se puede acceder a sus clientes *NTP* como si se tratase de servidores. Esto se puede hacer bien con una regla de cortafuegos o con la línea '*restrict default ignore*' de *ntp.conf* o con ambas.

Cuando haya configurado un cliente, reinicie su demonio *NTP*. Entonces, podrá utilizar *ntpq* para verificar el estado. En algunos casos, es más sencillo configurar la hora en un cliente utilizando el comando **ntpdate**. Este programa forma parte de la suite *NTP* y configura el reloj en un solo paso. El comando *ntpdate* ha caído en desuso y podría desaparecer del paquete *NTP* en cualquier momento.Aún así podemos probar con la siguiente sintaxis:

```
# ntpdate servidor.dehora.com
```

Como alternativa, podemos iniciar *ntpd* con la opción **-g**, que le permite configurar el reloj en un solo paso.

En condiciones normales, *ntpd* finalizará si la hora del servidor horario difiere de la hora local en algo más que unos pocos minutos.

Consultar y establecer su configuración local

La configuración local es un modo de especificar el idioma, el país del ordenador o usuario y la información relacionada relativa a la personalización de la visualización del entorno.

Código de configuración local:

```
[idioma[_territorio][.conjuntodecódigos][@modificador]
```

Idioma: **en** (inglés), **fr** (francés), **es** (español), etc..

Territorio: **FR** (Francia), **JP** (Japón), **US** (Estados Unidos)…

Conjunto de códigos: Son los métodos de codificación.

- **ASCII**: Método de codificación más antiguo y primitivo, emplea codificación de 7 bits, suele utilizarse para el inglés. No puede procesar los caracteres que se emplean en la mayoría de los otros idiomás.
- **ISO-8859**: Fue uno de los primeros intentos de ampliar *ASCII*, empeando un octavo bit para ampliar *ASCII* a 128 caracteres. Se divide en muchos subestándares, por ejemplo *ISO-8859-1* que abarca los idiomás de Europa del este
- **UTF-8**: *8-bit Unicode Transformation Format* o *Formato de transformación Unicode de 8 bits*. Es el conjunto de códigos de idiomás más reciente. Se inicia con *ASCII* pero lo amplia incluyendo extensiones de tamaño variable, de modo que la codificación de un único carácter puede ocupar desde uno hasta cuatro bytes. Ofrece la posibilidad de codificar exto en cualquier idioma aceptando **Unicode**, que es un conjunto de caracteres diseñado para admitir todos los lenguajes posibles. *UTF-8* gestiona todos sus sistemás de escritura automáticamente.

¿Cual es su configuración local?

El código de configuración local se puede asignar a una o varias variables de entorno. Ejecute el comando **locale** para saber cómo están configuradas en su sistema.

Cuando los programás consultan estas variables, realizan automáticamente ajustes en función de sus valores. La mayoría de estas variables de configuración local definen características obvias y específicas, como *LC_PAPER* (tamaño del papel), *LC_MEASUREMENT* (unidades de medidas), etc…

La variable *LC_ALL* se emplea para realizar lo que podríamos denominar invalidaciones maestras, es decir, invalida el resto de las variables *LC_**

Otra variable relacionada es *LANG* que define la configuración local en caso de que las variables *LC_** no se hayan definido. ***locale -a*** identidfica todas las configuraciones locales disponibles.

Modificar su configuración local

Para cambiar su configuración local el método más sencillo es definir la variable de entorno LC_ALL, por seguridad debería de definir también LANG en sus archivos de script de inicio de bash, como ~/.bashrc o /etc/profile

```
$ export LANG=es_ES.UTF-8
$ export LC_ALL=es_ES.UTF-8
```

Nota: Algunos programás y conjuntos de programás pueden requerir que defina el idioma independientemente de la configuración local global del sistema

Hay parámetros que requieren mención especial: **LANG=C**

Cuando establece *LANG=C* los programás que ven esta variable de entorno muestran una salida que no ha pasado por el filtro de las traducciones de las configuraciones locales, útil en casos en los que la configuración local daña la salida de un programa; por ejemplo, cuando las conversiones a *UTF-8* cambian los caracteres que se deberían preeservar como entidades de 8 bits.

A veces ocurre que un editor de texto admita *UTF-8* pero no *ISO-8859*, si trabajas con archivos *ASCII* no habrá problemás, pero si recibe un archivo de texto *ISO-8859-1* con caracteres no romanos, como las diéresis, su editor podría mostrarlos de una manera extraña. La utlidad **iconv** puede resolver este problema ya que realiza conversiones entre conjuntos de caracteres. Su sintaxis es:

incov -f codificación [-t codificación] [archivo de entrada…]

Las opciones **-f** y **-t** especifican la codificación de origen y de destino. Podremos obtener una lista de codificaciones con la opción **–list**.

Por defecto iconv envía la salida a la salida estándar, si deseas almacenar los datos en un archivo podrías redireccionar la salida:

```
$ iconv -f iso-8859-1 -t UTF-8 foobar.txt > foobarcodeutf.txt
```

CONFIGURANDO IMPRESORAS EN LINUX

Sobre las impresoras en Linux

lpr es el programa encargado de envíar las tareas de impresión (la envía a una cola especificada, que suele encontrarse en un subdirectorio de (*/var/spool/cups*) a la impresora, que puede ser invocado por el usuario o la aplicación desde la que se ejecuta la orden de imprimir.

Nota: El sistema de impresión acepta tareas de impresión de *lpr* o de ordenadores remotos.

Existen programás que pueden enviar la tarea de impresión (a la cola) en formato **PostScript** (lenguaje de programación orientado a la impresión que heredó Linux de Unix) el cual es legible por impresoras de alto precio (normalmente). Ante este problema (el que linux antiguamente no pudiese usar impresoras domésticas), se creo **Ghostscript** que es un traductor de *PostScipt*. *Ghostscript* es un demonio que se ejecuta en 2º plano y que espera a que se envíe algún documento en formato *PostScript* para traducirselo a la impresora. Lo que hace *Ghostscript* realmente es tomar la entrada enviada en formato *PostScript*, la analiza y genera una salida en varios formatos de mapas de bits diferentes, incluyendo formatos válidos para muchas impresoras que no admiten *PostScript*. Una de las desventajas de *GhostScript* es que genera archivos de salida de un gran tamaño, por lo que algunas impresoras (en particular las láser) pueden necesitar una ampliación de memoria para operar de manera fiable bajo Linux.

La cola de impresión es gestionada por un software conocido como **CUPS** (*Common Unix Printer System*) que filtra el documento determinando a una cola de impresión concreta. *CUPS* puede funcionar como cliente, enviando tareas de impresión a otros ordenadores que ejecuten los mismos protocolos. Pueden existir varias colas, que imprimirán de maneras distintas en la misma impresora, por ejemplo, podemos definir una cola para que imprima a una sola cara y otra cola para que imprima a doble cara, siempre y cuando la impresora admita está función. En casos en los que el documento vaya en formato *PostScript*, será *CUPS* quien lo mande antes a *Ghostscript* para su traducción. Los administradores podrán examinar, alterar o establecer el contenido de la cola de impresión.

CUPS dispone de una herramientra web a la que podemos acceder via *http://localhost:631* o por el puerto *639* si se trata de *https*, en la cual podremos definir las impresoras, características,etc… *CUPS* se ejecuta mediante script de inicio SysV (**cupsd**) o de disponer de otro sistema de inicialización deberemos de buscar su archivo de arranque como ya hemos estudiado en el TEMA – 5. La herramienta GUI de CUPS puede modificar los archivos de configuración de la impresora de forma automática, pero si queremos hacerlo de forma manual, ya sea por que queremos compartir una impresora o para realizar una configuración más detallada, deberemos de acudir al directorio donde se encuentran estos: **/etc/cups**. Por ejemplo si queremos añadir o eliminar una impresora podremos hacerlo mediante el archivo */etc/cups/printers.conf*, si queremos configuirar opciones adicionales, como por ejemplo la configuración de una cola, podremos modificar el archivo */etc/cups/nombrecola.ppd* (*PostScript Printer*

Definition). Este tipo de archivos (**PPD**) son el "pegamento" perfecto para unir la impresora con *CUPS*, digamos que son los drivers, su definición en sí. Para realizar una configuración al detalle tendremos el archivo */etc/cups/cupsd.conf*, archivo que tiene una estructura parecida a *httpd.conf*, por lo que nos permitira ajustar incluso la seguridad, denegar acceso a hosts, acceder a impresoras IPP remotas, etc…

Si *CUPS* no nos muestra el modelo de nuestra impresora, desafortunadamente tendremos que buscar el archivo PPD de nuestra impresora en nuestra propia distribución, en Foomatic, Gutenprint, CUPSDDK, el fabricante o cualquier fuente que lo pueda tener.

Configurar y compartir la impresora

Podemos compartir impresoras mediante un servidor *samba* para redes mixtas (*Windows* y *Linux*). Windows reconoce las impresoras compartidas mediante el protocolo *SMB/CIFS* (*Server Message Block/Common Internet File System*). Si tenemos una impresora compartida mediante *samba* y el demonio *cupsd* corriendo en nuestra máquina Linux, podremos añadir la impresora mediante la ruta:

```
smb://nombredeusuario:contraseña@SERVIDOR/RECURSO_COMPARTIDO
```

Si en cambio vamos a imprimir en un equipo Linux o UNIX, que utilice el antiguo formato **LPD** (*BSD Line Printer Daemon*) utilizaremos la ruta:

```
lpd://nombredehost/cola o ipp://nombrehost/cola
```

Linux podrá trabajar con **LPD**, **IPP** o **SMB/CIFS**, pero por regla general *IPP* es el más sencillo de configurar, necesitará que ambos sistemás (cliente/servidor de impresión) estén usando *CUPS*, de lo contrario la segunda opción sería *LPD* ya que no requiere credenciales y por ultimo *SAMBA*.

La forma por defecto de añadir una impresora con *CUPS* es:

1. Acceder a CUPS GUI *http://localhost:631*, escribir usuario y password de root (si la solicita) y clicar en la pestaña '**Administración**' > '**Add printer**'
2. Podremos añadir una impresora a partir de tres categorías: *local*, *impresora de red detectada* o *otra impresora de red*.
3. Si por ejemplo usamos '*impresora de red*', introduciremos la ruta completa del dispositivo como mencionamos anteriormente, supongamos

```
lpd://nombreservidorimpresion/nombreimpresora
```

Nota: Se usará '*nombreimpresora*' como nombre de cola de impresión

4. Ahora estaremos en la página en la que podremos introducir el *nombre de la impresora*, *su descripción* y *ubicación*. Conviene que el nombre de la impresora sea

corto y entendible, ya que lo usaremos para lanzar impresión desde línea de comandos o para localizarla desde GUI. Igualmente, en esta página podremos marcar la opción '**Share This Print**'
5. A continuación seleccionaremos un *fabricante* y clicaremos en '**Continue**' o bien hacer referencia directamente a un archivo **PPD** que tengamos en el PC.
6. Si eligió un archivo *PPD*, puede saltarse este paso, de lo contrario deberá de seleccionar un *modelo de impresora* y continuar.
7. Aquí podremos seleccionar las opciones específicas para la impresora como el tamaño del papel, la resolución, etc… clicaremos '**Set Printer Options**' para terminar.

Herramientas para el uso de la impresora

Podemos usar varias herramientas para imprimir y gestionar las colas de impresión desde la línea de comandos como por ejemplo, si queremos enviar trabajos a la cola de impresión o modificar esta, podremos usar **lp** o **lpr**, si queremos ver el estado de la cola de impresión **lpq** o **lpc** que ofrece opciones extras, con **lprm** o **cancel** podremos cancelar trabajos en la cola de impresión o moverlos con **lpmove.** El comando **lpadmin** nos permite una administración de las impresoras a través de CUPS.

Otras herramientas podrian ser: **lpstat, lpoptions**, **cupsaccept**, **cupreject**, **cupsenable** y **cupsdisable**. Estos y los anteriores serán estudiados a fondo en la sección de comandos de este mismo capítulo.

Algunos ejemplos:

- Imprimir un archivo y eliminarlo a su finalización:

```
$ lpr -r <nombrearchivo>
```

- Enviar una notificación por correo:

```
$ lpr -m <dir-email>
```

- Ver las colas existentes:

```
$ lpq -P <nombreimpresora>
```

- Eliminar una tarea en concreto:

```
$ lprm <num-tarea>
```

LPIC-1 Capítulo 7

EN ESTE CAPÍTULO SE ABARCAN LOS SIGUIENTES OBJETIVOS DEL EXAMEN:

- 107.1: Administrar cuentas de usuarios y grupos, y los archivos del sistema relacionados (5)
- 108.2: El registro del sistema (3)
- 107.2: Automatizar las tareas de administración del sistema mediante la programación de tareas (4)

Administrar el sistema

Sobre usuarios y grupos

Usuarios

Linux nos permite dar casi cualquier nombre a una cuenta de usuario. Las reglas más "liberales" sobre nombres de usuario nos permiten usar caracteres en minúsculas y mayúsculas (para *Linux*, Andrea, andrEa y andrea serán cuentas diferentes) , tener una longitud de hasta 32 caracteres y usar algunos caracteres especiales como el guión bajo (**_**), el punto (**.**) o el signo del dólar (**$**) al final del nombre. Se recomienda que las cuentas de usuarios comiencen con una letra, aunque se pueden emplear números incluso caracteres de puntuación como el '**.**' y el '**_**'. Algunas utilidades usadas para la creación de cuentas de usuarios son más restrictivas, por lo que nos podemos encontrar problemás a la hora de crear un nombre con espacios, más de 8 caracteres o incluso que no nos permitan comenzar el nombre con caracteres especiales, números o incluso usar letras mayúsculas.

Nota: Si empleamos como primer caracter para el nombre de la cuenta el '.' (punto) tendremos que tener en cuenta que el directorio home del usuario aparecerá como oculto bajo */home*

Cada usuario que creemos en el sistema tendrá una línea que lo identifique en el archivo **/etc/passwd** y su correspondiente contraseña encriptada más algunas características de la cuenta en el archivo **/etc/shadow**.

Podemos ver los usuarios logueados en el sistema a través del comando **finger**.

Nota: El comando *finger* está en desuso por su alto riesgo de inseguridad. Existen otros comandos como **who** o **w** con los que podemos ver los usuarios logueados en el sistema

Sobre el archivo /etc/passwd

El formato de */etc/passwd* es el siguiente:
usuario**:**x**:**UID**:**GID**:**nombrecompleto**:**directoriohome**:**consola

```
nebul4ck:x:1000:1000:Gonzalo Sánchez,695-235-
526:/home/nebul4ck:/bin/bash
```

El segundo campo '**x**' indica que la contraseña no se almacena en este archivo, si no que se encuentra encriptada en */etc/shadow*. Los campos tercero y cuarto indican el **UID** (*ID de usuario*) y **GID** (*ID del grupo principal del usuario*) del usuario. El quinto campo es utilizado para proveer información acerca del usuario, como su nombre y teléfono en nuestra línea de ejemplo. Este campo no es un requisito para la creación de un usuario, es por ello que con frecuencia lo encontremos vacío. Si un campo no tiene valor, aún así deberá de ser definido por dos puntos '**:**' tal que así:

```
nebul4ck:x:1000:1000::/home/nebul4ck:/bin/bash
```

Por último el sexto campo, cuyo objetivo es ejecutar un comando. La mayoría de las veces se utiliza este campo con dos objetivos principales:

- Ejecutar un *shell* para el usuario, **bash** en nuestro ejemplo
- Negar el acceso de sesión a un usuario. Esto es utilizado normalmente para cuentas del sistema a las que no se le tiene permitido iniciar sesión como por ejemplo '*daemon*', '*man*', '*nobody*'… Esto se suele conseguir de dos formás distintas. Si despliegas el contenido del archivo */etc/passwd* podrás observar que existen líneas que poseen como sexto campo el comando */usr/sbin/nologin* mientras que otros (hablamos de valores diferentes a una consola) tienen como valor */bin/false*. Ambos niegan el acceso a la cuenta, la diferencia es que el comando *nologin* en realidad '*te permite acceder a la cuenta*' pero acto seguido te muestra un mensaje diciendo que el usuario no tiene permitido el acceso y te expulsa. Esta opción suele ser la preferida por el hecho de mostrar como poco un mensaje. /bin/false es un binario que finaliza directamente devolviendo el código de retorno '1', por lo que tras escribir la contraseña lo único que notaremos será un pequeño parpadeo en la pantalla y de nuevo estaremos ante el prompt pidiéndonos unas credenciales o bien en el prompt del usuario en el que nos encontrábamos antes de ejecutar la orden '*su – nuevousuario*', estudiaremos los comandos **su** y **sudo** en otra sección de este mismo capítulo.

Sobre el uso de UID y GID

Todos los archivos y directorio que cree el usuario, así como cualquier programa ejecutado por él, se asociarán a su *UID* y *GID*.

Algunas distribuciones dan como primer **UID** y **GID** el identificador **1000** y otras el **500**. Si tenemos un usuario con *UID* y *GID* 1000 unicamente y creamos otro, este tomará por defecto el *UID* y *GID 1001* y así secuencialmente, a menos que indiquemos nosotros mismos el *UID* y *GID* mediante opciones de la utilidad de creación de cuentas, al igual que usando estas, podremos seleccionar la consola, directorio home principal y nombre o comentario que nosotros decidamos. *Linux* sigue el rastro de los usuarios y grupos por sus respectivos *UID* y *GID* en vez de por sus nombres, por lo que accederá a */etc/passwd* o */etc/group* (veremos el formato de este archivo en el apartado de grupos) para localizar el número asociado al nombre. El primer centenar de *ID* de usuarios y grupos (0-99) suelen reservarse para uso del sistema, como por ejemplo para cuentas como *root* (0), *daemon* (1), *bin* (2), *mail* (8), etc..

Importante: Los *ID* de cuentas del sistema suelen variar de una distribución a otra, menos el *UID* y *GID* de **root**, que siempre será **0**. Deberemos de prestar especial atención a esto, ya que un intruso podría crear una cuenta con *UID* 0 para obtener privilegios de *root*. Si detecta una cuenta sospechosa en su archivo */etc/passwd* con un *UID* 0, es probable que su sistema esté comprometido.

Se pueden crear varios cuentas de usuario y grupos con los mismos *UID* y *GID*, siendo estas, cuentas totalmente diferentes, por lo que existirá una entrada en */etc/passwd* y */etc/group* para cada una de ellas, con diferentes contraseñas, directorios principales, etc… No obstante, *Linux* tratará a estas cuentas de la misma manera en lo referente a los permisos de archivos, ya que estos son aplicados a los *UID* y *GID*, por lo que debería de evitar crear varios usuarios o grupos de compartan *ID*.

Reutilizar un número *ID* puede provocar problemás si no elimina los archivos del antiguo usuario ya que el nuevo usuario se convertirá en el propietario de los archivos del anterior.

Los límites de numeración de cuentas se definen en el archivo **/etc/login.defs** con las entradas **UID_MIN** y **UID_MAX**

Sobre el archivo /etc/shadow

Como comentábamos anteriormente el archivo */etc/shadow* contiene la contraseña encriptada del usuario y algunos otros parámetros configurables con la utilidad **chage**, **passwd**, **useradd** o **usermod** que veremos en apartados posteriores.

Vamos a estudiar este archivo mediante una línea de ejemplo:

```
nebul4ck:$8$km3ZmQpk9mJLlq3bVGU6ssPwa:15988:0:-1:7:-1:-1:
```

- **Nombre de usuario**: El nombre de usuario enlaza las entradas de este archivo con las de */etc/passwd*.
- **Contraseña**: Formato encriptado. Un asterisco '*****' o signo de admiración '**!**' al comienzo de este campo denota una cuenta sin contraseña, por lo que esta se encontrará bloqueada. Si eliminamos este signo se desbloqueará la cuenta y se restaurará la contraseña original.
- **Ultimo cambio de contraseña**: Fecha en la que se realizó el último cambio de contraseña. Se identifica por el número de días transcurridos desde el 1 de Enero de 1970, **15988** en nuestro ejemplo.
- **Días que han de pasar para permitir un cambio de contraseña**: Es el número de días que han de pasar para que se pueda cambiar la contraseña. En nuestro ejemplo un **0**, que indica que cualquier día será bueno para cambiar la contraseña e incluso cambiarla varias veces al día.
- **Periodo de días tras el cual habrá que modificar la contraseña**: Es el número de días que deberán de pasar desde que se cambió la contraseña hasta que se requiera un nuevo cambio. En nuestro ejemplo se indica un **-1** (equivalente a 99999) que normalmente indica que se ha desactivado esta función.
- **Avisar con antelación de la necesidad de cambiar la contraseña**: En nuestro ejemplo se nos avisará **7** días antes de que nuestra contraseña nos caduque. 7 suele ser un número muy común en este campo, indica una semana antes.
- **Días que pasarán entre la caducidad de la contraseña y la desactivación de la cuenta**: Cuando una cuenta se desactiva (bloquea) por que su contraseña a caducado el usuario no podrá usar su cuenta, por ello se ofrece un plazo en días en los cuales el usuario podrá cambiar su contraseña justo después de acceder a la cuenta (siempre que se encuentre dentro del plazo), de lo contrarío la contraseña anterior será eliminada y será el administrador quien tenga que volver a activar dicha cuenta.
- **Fecha de caducidad:** Esta fecha se expresa de la misma manera que la fecha del último cambio de contraseña es decir los días transcurridos desde el 1 de Enero de 1970 hasta el día en que la cuenta caducará.
- **Indicador especial**: Este campo no suele usarse por lo que en nuestro ejemplo lo tenemos vacío

```
nombre:contraseña:último-cambio:cambiar-pass-no-antes-de:cambiar-
contraseña:días-de-aviso:periodo-de-gracia:caducidad:indicador
```

Nota: Cuando hablemos sobre la modificaciones de las cuentas de usuario veremos que es posible modificarlas a través de los archivos */etc/passwd* y */etc/shadow*, pero se recomienda que las modificaciones se hagan con las utilidades destinadas a tal fin como son *passwd*, *chage*, *usermod*,etc…

Advertencia: El archivo */etc/shadow* se suele almacenar con permisos más restrictivos (*600*) que los usados en */etc/passwd* (*644*) evitando de esta manera que los usuarios distintos a *root* lean el archivo y obtengan la lista de contraseñas encriptadas.

Grupos

Los grupos son medios para organizar colecciones de cuentas. Un grupo puede ser el grupo principal de uno o varios usuarios y además contener a más usuarios aunque para estos no sea el grupo principal. La pertenencia a los grupos se controla mediante el archivo **/etc/group**, el cual contiene una lista de grupos y los miembros que pertenecen a cada uno de ellos. Podemos comprobar a que grupos pertenecemos y otras características de la cuenta como los *ID* de usuario y grupos con el comando **id** o bien listar todos los grupos a los que pertenecemos con el comando **groups**. El comando *groups* no indicará cual es nuestro grupo primario ni los *GID*, simplemente desplegará una lista con todo aquel grupo del que formamos parte.

Cuando un archivo es creado se le asigna un grupo que será el grupo principal del usuario que ha creado o ejecutado el archivo/directorio o el programa en cuestión. Sin embargo, para ejecutar programás o crear archivos con un grupo distinto al grupo primario del usuario creador, este deberá de ejecutar el comando **newgrp** para cambiar la pertenencia del grupo principal al nuevo grupo. Para que esto sea posible el usuario debe de pertenecer antes al nuevo grupo o si el nuevo grupo tiene definida una contraseña de grupo esto no será necesario, pero si que el usuario podrá convertirse en miembro temporal del grupo siempre y cuando conozca dicha *password* que será solicitada tras la ejecución de *newgrp*.

Nota: En algunas distribuciones no se permite la modificación temporal del grupo principal aún conociendo la password del grupo (este recibirá un mensaje del tipo "*access denied*"), a menos que previamente el usuario pertenezca al grupo.

Ejemplo de newgrp:

Supongamos que el usuario '*dante*' tiene como grupo principal '*dante*' y que además pertenece a otros grupos, entre ellos al grupo '*sistemás*', y quiere cambiar a este último para que nuevos archivos sean creados con el *GID* del grupo *sistemás* y además ejecutar unos programás bajo los permisos asignados al grupo.

```
$ newgrp sistemás
```

Una vez ejecutado este comando el nuevo grupo principal para *dante* será *sistemás* y los archivos creados se asociarán a este.

Debemos de conocer algunas peculiaridades del uso del comando *newgrp*:

- El cambio al nuevo grupo principal es temporal, es decir, al iniciar una nueva sesión o cerrar y abrir una terminal, volveremos a tener nuestro grupo principal por defecto.
- Cuando ejecutamos *newgrp* veremos los efectos a la hora de crear archivos o ejecutar programás, pero no si desplegamos el contenido de */etc/passwd* o

/etc/group para los cuales nuestro grupo principal seguirá siendo el que teníamos por defecto.

- Si hemos ejecutado *newgrp* para pasarnos a otro grupo temporalmente y queremos volver a nuestro grupo de origen sin cerrar sesión e iniciar nueva sesión bastará con volver a ejecutar *newgrp* sin argumento alguno

Configurar cuentas de usuario

En esta sección del capítulo vamos a repasar las herramientas con las que podremos crear, modificar y eliminar cuentas de usuario. Linux posee herramientas *GUI* para tal fin, pero las que vamos a estudiar son las que están basada en modo texto, ya que son más flexibles y por ello nos permitirán realizar una configuración más detallada. Como veremos en las próximás secciones las cuentas de usuarios y grupos pueden ser creados mediante comandos destinados a tal fin, o bien editando una serie de archivos de forma manual. En la sección de creación de cuentas de usuarios y grupos hablaremos sobre las herramientas destinadas a estas tareas ya que son la forma conveniente de hacerlo. No obstante en la sección "*Modificando cuentas de usuarios y grupos directamente desde archivos de configuración*" estudiaremos como hacerlo de tal forma.

Creando cuentas de usuarios

Cuando creamos una cuenta de usuario en el sistema realmente lo que se está haciendo de forma automática es lo siguiente:

- Crear una línea de identificación y especificación de ciertas características de la cuenta en el archivo **/etc/passwd**
- Crear una contraseña con el comando **passwd**. Esto suele hacerse de forma manual (por seguridad) tras crear la cuenta y lo que hace es añadir una contraseña encriptada en el archivo **/etc/shadow**
- Se crea una línea en **/etc/group** para identificar al grupo primario del usuario
- Se crea un directorio */home/<usuario>* en el que se copian una serie de archivos de configuración de perfil desde el directorio **/etc/skel**

Todo lo automatizado o configurable del comando destinado a crear un usuario **useradd,** es definido en el archivo de configuración /etc/default/useradd y por otra parte el comportamiento de una cuenta viene determinado por */etc/login.defs* por lo que ambos ficheros son de importancia cuando se crea o administra una cuenta de usuario. Veremos que existen en algunas distribuciones una interfaz sencilla llamadas *adduser* la cual se configura en otros archivos independientes.

Para crear cuentas de usuarios utilizaremos la utilidad de bajo nivel *useradd*, para la cual existe en muchas distribuciones (como por ejemplo las derivadas de *Debian*) una interfaz más sencilla denominada **adduser** que automáticamente selecciona valores basados en la línea de comandos y en el archivo de configuración */etc/adduser.conf*, como por ejemplo el *ID* de usuario y grupo, la creación de un directorio personal (*/home/usuario*), ejecución de un script personalizado y algunas otras funcionalidades.

En este caso, aunque la herramienta que abarca el examen es *useradd*, vamos a estudiar también *adduser*.

Cuando invocamos a *useradd* es su forma más simple, es decir con tan solo el nombre de login de usuario como parámetro no crearemos ningún directorio home, los UID y GID serán definidos por defecto, no asignaremos un shell, ni tampoco añadiremos información complementaria como su nombre o teléfono. Esto no quita que la cuenta tenga unas característica o comportamiento determinado como por ejemplo un directorio mail a utilizar (*/var/mail*), un valor de variable *PATH* predeterminado para la cuenta, una *umásk*, valores de *UID*/*GID* mínimos y máximos, los intento de *login fallidos*, el *TIMEOUT* para el login, método de *encriptación para la clave* (SHA512 por defecto) cuando esta es creada con el comando *passwd*, y algunos otros muchos parámetros incluso relacionados con los grupos que son definidos por variables en el archivo */etc/login.defs*.

En cambio si utilizamos el comando *useradd* con parámetros para crear *shell* o directorio home, este se basará en los valores de unas determinadas variables definidas en el archivo */etc/default/useradd*. Podremos ver los valores de estas variables con la opción *-D* de este comando e incluso modificarlas.

Estudiaremos la sintaxis y opciones de ambos comandos en el apartado de comandos de este mismo capítulo, pero mencionaremos algunos ejemplos para ir familiarizándonos con ellos.

- Crear un usuario con **useradd** sin utilizar parámetros y opciones adicionales:

```
$sudo useradd cursolpi
```

La línea que se generará en el archivo */etc/passwd* tendrá el siguiente aspecto:

```
cursolpi:x:1007:1007::/home/cursolpi:
```

- Crear un usuario prácticamente al detalle utilizando el mismo comando **useradd**:

```
sudo useradd -d /home/cursolpi -m -c "Usuario para curso LPIC-1" -e
2015-04-20 -u 1024 -g 1001 -G 1002,104 -s /bin/bash -p norecomendedado
cursolpi
```

Lo que hace este comando es crear con '**-m**' el directorio indicado por el parámetro '**-d**', con un comentario '**-c**', indicando que la cuenta expire '**-e**' en una fecha determinada, se asigna un *UID* y *GID* específico y además se le hace miembro de otros dos grupos '**-G**', utilizará el shell *bash* y para acceder a su cuenta necesitará su nombre de usuario '*cursolpi*' y su contraseña '*norecomendado*'.

Advertencia: Como medida de seguridad no se recomienda utilizar el parámetro **-p** <*contraseña*> desde la línea de comandos. Es preferible crear la cuenta y a continuación utilizar el comando *passwd* para tal fin

Ahora la nueva línea del archivo */etc/passwd* tendrá el siguiente aspecto:

```
cursolpi:x:1024:1001:Usuario para curso LPIC-
1:/home/cursolpi:/bin/bash
```

- Crear un usuario con los valores por defectos configurados en */etc/adduser.conf* con el comando **adduser** que desplegará unas líneas que servirán para que interactuemos con la utilidad:

```
[redacted]@LiMinCinn ~ $ sudo adduser dottore
Adding user `dottore' ...
Adding new group `dottore' (1007) ...
Adding new user `dottore' (1007) with group `dottore' ...
Creating home directory `/home/dottore' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:  En estas lineas se nos pedirá que
Retype new UNIX password: introduzcamos la contraseña
passwd: password updated successfully
Changing the user information for dottore
Enter the new value, or press ENTER for the default
        Full Name []: Patricio Calamonte
        Room Number []: 12
        Work Phone []: 666-666-666
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
```

Tras esto la línea creada para este nuevo usuario en el archivo */etc/passwd* tendrá el siguiente aspecto:

```
dottore:x:1007:1007:Patricio Calamonte,12,666-666-
666,:/home/dottore:/bin/bash
```

Nota: La utilidad *adduser* nos permite además crear grupos pasándole el parámetro **–group**, pero mejor utilizar las herramientas destinadas a la creación de grupos.

Implementar contraseñas shadow y crear una o varias contraseñas.

Las distribuciones actuales traen por defecto la implementación de contraseñas *shadow*, esto no es más que una forma de implementar algo más de seguridad para guardar las contraseñas de nuestros usuarios. En versiones anteriores de distribuciones las contraseñas solían guardarse de forma encriptada en el campo de contraseñas del archivo */etc/passwd*. Esto tenía la vulnerabilidad de que cualquier usuario podía leer el archivo y con ello la contraseña encriptada. Si desplegamos el contenido de nuestro archivo */etc/passwd* lo más normal será tener una **:x:** en el campo de contraseña, esto indica que tenemos implementado contraseñas *shadow*, con lo que estas serán guardadas en el archivo */etc/shadow* el cual solo puede ser leído por *root*. No obstante podemos implementar o igualmente dejar de utilizar estas contraseñas con los comandos **pwconv** y **pwunconv** respectivamente. Si usamos el comando *pwconv* lo que hará será copiar el campo de contraseña de */etc/passwd* en el campo de contraseña de */etc/shadow* y añadir el caracter **x** en dicho campo de */etc/passwd*. El uso de *pwunconv* realizará la acción inversa.

Tengamos o no implementado contraseñas *shadow* en nuestro sistema, podremos usar el comando **passwd** o **gpasswd** para crear contraseñas de cuentas de usuario o grupos respectivamente. Además tenemos la opción de contraseñas para múltiples cuentas de una sola vez, gracias al comando **chpasswd** que leerá desde su entrada estándar una lista de *usuario:contraseña* (una por cada línea).

El comando *passwd* nos permite pasar una serie de opciones. El uso de la opción **-p** <*contraseña*> establece una contraseña desde línea de comando algo que es desaconsejado, mejor crear la cuenta y posteriormente asignar una contraseña con ‘*$ sudo passwd cursolpi*‘. Los usuarios normales pueden utilizar *passwd* para cambiar sus contraseñas, pero muchos de sus parámetros como **-l**, **-u**, **-f** y **-d** solo los puede utilizar *root*. Un usuario corriente podrá cambiar su contraseña simplemente invocando a la utilidad *passwd* sin ningún parámetro. Como medida de seguridad se le preguntará por la antigua contraseña antes de proceder al cambio. Solo *root* puede especificar tras la orden *passwd* el nombre del usuario al que se le va a crear/modificar la contraseña.

Nota: Hay que tener en cuenta a la hora de crear/modificar la contraseña que *Linux* distingue entre mayúsculas y minúscula.

Como comentábamos anteriormente, cabe la posibilidad de cambiar la contraseña a varios usuarios de una misma vez mediante el comando *chpasswd* que solo *root* podrá ejecutar. *chpasswd* lee de la entrada estándar una serie de líneas cada cual debe estar formada por un usuario y una contraseña (en texto plano) separado por dos puntos ‘**:**‘ (*usuario:contraseña*). Será *chpasswd* a través de *PAM* quien codifique las contraseñas, aunque este comportamiento puede ser cambiado mediante las opciones **-e**, **-c** o **-m** del comando, algo que no es recomendado. Este comando suele utilizarse en sistemás que contienen multitud de cuentas de usuarios donde realizar la tarea de cambiar contraseñas una a una resulta tedioso.

Crear grupos

Al igual que ocurre con la herramienta de creación de cuentas de usuario, para los grupos disponemos de la utilidad de bajo nivel **groupadd** y la interfaz sencilla para algunas distribuciones **addgroup**. Lo normal en el momento de crear un grupo es pasar como argumento el propio nombre del grupo al comando *groupadd*. El sistema le asignará el *GID* que cree conveniente, normalmente sumándole 1 al último *GID* existente.

La interfaz de creación de grupos *addgroup* funciona de una forma similar, es decir bastará con pasar como argumento el nombre del grupo. Será el archivo */etc/adduser.conf* el que predetermine la funcionalidad del comando *addgroup*.

- Creando un grupo con *groupadd*:

```
$ sudo groupadd -g 1003 -o -r -p vulnerable newbie
```

Con estos parámetros estaremos actuando de una forma '*imprudente*', pero servirá para nombrar algunas opciones importantes. Por ejemplo como ya vimos, pasar el parámetro **-p** seguido de la *password* no es lo aconsejable. Ya existe un grupo con el *ID 1003* en el sistema, pero con **-o** le estaremos indicando a la utilidad que cree el grupo '*newbie*' compartiendo *GID* con el grupo ya existente, es decir, existirán 2 grupos con diferentes nombres pero mismo *GID*. La opción **-r** indica que será un grupo del sistema, en realidad no es algo muy grave porque simplemente estaremos diciendo que use uno de los *GID* reservados (normalmente del 0-99) para grupos del sistema, aún así no es aconsejable crear un grupo '*ordinario*' de esta forma.

Modificar cuentas de usuarios

Tanto las cuentas de usuarios como los grupos pueden ser modificados de varias maneras, bien editando directamente los archivos */etc/passwd*, */etc/group*, */etc/shadow* y */etc/gshadow* (forma menos aconsejable) o la modificación mediante herramientas destinadas a tal fin con las que evitaremos errores típicos. Un error al editar el archivo */etc/shadow* suele ser el no tener en cuenta los años bisiestos a la hora de contar el número de días que hay desde el 1 de Enero de 1970 hasta el día en el que queremos que una contraseña caduque.

Además de estas dos formás, podremos personalizar los entornos de las cuentas de usuarios a través de sus archivos propios de configuración (*~/.profile*, *~/.bashrc* o *~/.bash_profile*) o los archivos de configuración global para cuentas como */etc/profile* y /etc/bash.bashrc.

Modificando las cuentas de usuarios y grupos directamente desde los archivos de configuración

En apartados anteriores describimos los formatos de los archivos */etc/passwd*, */etc/shadow* y */etc/group*. Conociendo los formatos, podremos modificarlos mediante un editor de texto e indicar en cada uno de sus respectivos campos aquellos valores con los que queramos que las cuentas sean configuradas.

Ala hora de modificar estos archivos directamente habrá que tener en cuenta por ejemplo la creación de una cuenta encriptada, siempre será mejor utilizar la utilidad *passwd* para tal fin. No es aconsejable editar manualmente de forma general el archivo */etc/shadow* ya que contiene campos demásiado específicos que podremos configurar de forma más segura con herramientas como *passwd*, *useradd* o *usermod*.

Los archivos */etc/passwd* y */etc/group* son más fácil de configurar que */etc/shadow*, aun así volvemos a hacer hincapié en el uso de herramientas de línea de comandos o *GUI*. No obstante si hemos decidido finalmente crear o modificar usuarios y grupos desde sus archivos de configuración deberemos de tener en cuenta una serie de aspectos:

- Hacer una copia de seguridad de los archivos a editar. Si metemos la pata durante la edición, lo mejor será tener una copia del archivo original.
- En el archivo */etc/passwd* deberemos de dejar en blanco el campo de la contraseña para posteriormente generarla con el comando *passwd*.
- Será importante mantener la secuencia de los identificadores de usuarios y grupo en los campos *UID* y *GID*. Para ello podremos ver el número *ID* asignado al último usuario y grupo que creamos y sumarle +1.
- Una vez generada la línea que identifica a un usuario en el archivo */etc/passwd*, guardaremos el archivo, generaremos la clave, y entonces crearemos el directorio */home*, en el que copiaremos el contenido del directorio esqueleto (*/etc/skel*) del cual hablaremos más adelante. A continuación le cambiaremos los permisos a este mediante el comando *chown* para que el nuevo usuario pueda acceder al sistema ya que de no tener permisos para acceder a su directorio no se le permitirá el acceso al sistema.
- Por último antes de que el usuario acceda al sistema deberemos de crear su grupo principal, añadiendo en el archivo */etc/group* su correspondiente línea, a menos que hayamos indicado un grupo principal en */etc/passwd* ya existente.

Modificando las cuentas de usuarios desde la línea de comandos

Las dos herramientas por excelencia para la modificación de cuentas de usuario ya creadas son **usermod** y **chage**.

Con el comando *usermod* podremos modificar características de una cuenta como por ejemplo cambiar el directorio principal de un usuario **-d** y además mover todo el contenido del antiguo directorio al nuevo **-m**. Otra de las funciones que nos facilita *usermod* es el cambio del nombre de acceso del usuario (incluso el nombre de *root* podrá ser cambiado), esto será posible gracias a su parámetro **-l**. Si hemos modificado el nombre del login para un usuario, tendremos que tener en cuenta que los nombres de su directorio principal y de existir el de *mail* no serán automodificados por lo que deberemos de renombrarlos nosotros. Del mismo modo que el comando *passwd,* nos permite bloquear y desbloquear una cuenta (en realidad bloquea la contraseña) podremos hacerlo con los parámetros **-L** y **-U** respectivamente.

Podemos también modificar los valores *UID* y *GID* de una cuenta, pero de hacerlo es muy importante saber que cuando cambiamos a un nuevo identificador, los archivos ya existente no cambiarán por lo que es posible que el usuario o incluso los usuarios pertenecientes al grupo dejen de tener acceso a estos. En este caso podremos recurrir a la utilidad *find* para localizar archivos en el sistema con un determinado *UID* y *GID* y ejecutar el comando *chown* para cambiarlos al nuevo usuario.

```
$ sudo find / -uid 1001 -gid 1001 -exec chown nebul4ck
```

Importante: Cambiar las características de una cuenta mientras el usuario está dentro del sistema puede tener consecuencias desagradables, en particular con el uso de los parámetros **-d** y **-m** de *usermod*. Algo a tener en cuenta por ejemplo cuando se cambian opciones como la consola por defecto de un usuario, es que este cambio no tendrá efectos hasta que el usuario salga y vuelva a entrar.

Nota: Con el comando *usermod* también podremos modificar características de los grupos. Esto lo veremos en la sección de grupos.

Las cuentas de *Linux* se pueden configurar para que caduquen automáticamente. Será el comando **chage** el que nos permita modificar ciertos parámetros relacionados con la caducidad. Una cuenta de usuario podrá caducar siempre y cuando se cumpla alguna de las siguientes condiciones:

- La cuenta está configurada para cambiar la contraseña cada cierto tiempo y esta, tras pasar el tiempo especificado e incluso un tiempo de gracia, no ha sido modificada.
- La fecha del sistema ha sobrepasado un tiempo predeterminado.

Con la utilidad *chage* podremos hacer cosas como mostrar la información sobre el periodo de caducidad de la cuenta y la contraseña **-l**, definir el tiempo mínimo en días para poder volver a cambiar la contraseña **-m** (*0* indica que puede cambiar la contraseña varias veces al día, *1* que puedes cambiarla una vez al día, *2* que podrás hacerlo 1 vez cada 2 días y así…) o por lo contrario establecer un periodo máximo entre cambios de contraseñas **-M**.

Nota: Si el usuario cambia una contraseña antes de la fecha límite, el contador se reinicializará, por lo que dispondremos nuevamente del margen de tiempo, por ejemplo, si hemos dado 30 días como tope entre cambios de contraseña, si al día que hace 25 modifico la contraseña, contaremos nuevamente con 30 días para volver a realizar el cambio y no con 5 días.

Otras opciones útiles para gestionar la caducidad pueden ser:

- Definir la fecha del último día en que se cambió de contraseña, que aunque este valor lo controla *Linux* de forma automática, podremos alterarlo con **-d** *AAAA/MM/DD* o con el número de días transcurrido desde el *1 de Enero de 1970*
- Podremos definir la fecha de caducidad absoluta de una cuenta con el parámetro **-E** *2015/09/21*, con el número de días transcurridos desde el *1 de Enero de 1970* o incluso que no caduque nunca pasándole como argumento al parámetro **-1** en vez de una fecha o días.
- Si antes de que caduque una cuenta definitivamente queremos disponer de un "*periodo de gracia*" comprendido entre el día en que caducó la contraseña y el día en que la cuenta será bloqueada podremos hacerlo con el parámetro **-I** *díasdeinactivo*. Una cuenta con contraseña caducada pero que aún no ha sido bloqueada pedirá al usuario que cambie la contraseña justo después de acceder a esta.
- Una cuenta podrá pasar varios días en estado inactivo tras haber caducado una contraseña e incluso podrá llegar a bloquearse definitivamente si el periodo de gracia se termina. Para evitar esto lo mejor será definir un número de días previos a la caducidad de la cuenta para que seamos informados y poder modificar nuestra contraseña y evitar el bloqueo, por ejemplo **-W** *7* nos avisará desde el modo consola una semana antes de que nuestra cuenta caduque. Los usuarios que estén en modo Gui no serán avisados.

Normalmente solo *root* puede utilizar el comando *chage*, la única excepción es cuando se emplea la opción *-l*, que permite a los usuarios consultar la información de caducidad de su propia cuenta.

Para terminar con la modificaciones de cuentas de usuario, nombrar el comando **chsh** el cual nos permitirá cambiar el *shell* con el que inicia la cuenta de usuario y el comando **chfn** que permite cambiar la información (comentarios **-c**) del usuario.

Modificar grupos

Como ya comentábamos anteriormente las características de las cuentas de usuarios y los grupos pueden modificarse con utilidades destinadas a ello, o bien modificando directamente sus archivos de configuración. Para la modificación de grupos mediante utilidades además de la herramienta *usermod* disponemos también de otras como

groupmod y **gpasswd** que serán las que se cubran en esta sección del tema "*Sobre usuarios y grupos*".

Con el comando *groupmod* podremos modificar los parámetros de un grupo ya existente. Su sintaxis es bastante clara y sencilla al igual que sus opciones.

Sintaxis del comando groupmod:

groupmod [opciones] grupo

Básicamente las opciones de *groupmod* nos permitirán seleccionar un *GID* concreto para el grupo con su opción **-g** o **–gid** e incluso elegir uno ya existente con la opción **-o**:

```
$ sudo groupmod -go 1004 migrupo
```

Cambiar el nombre del grupo:

```
$ sudo groupmod -n nuevogrupo miantiguogrupo
```

Asignar una *password* con su opción **-p** (se desaconseja utilizar esta opción):

```
$ sudo groupmod -p mipasswd
```

La password podremos crearla mejor con el equivalente a *passwd* que es en este caso *gpasswd*, simplemente pasando como argumento al comando gpasswd la contraseña deseada. Como ya vimos en secciones anteriores esta contraseña además de implementar seguridad a la hora de agregar usuarios al grupo, permitía que un usuario que no es miembro de un grupo pudiese pasar a tener a este como grupo principal de forma temporal con el comando *newgrp*, aunque también se comentó que esto no funcionaba en todas las distribuciones de *Linux*. Podremos eliminar la password de un grupo con la opción **-r** de *gpasswd*. El comando *gpasswd* además de permitirnos crear o eliminar una contraseña de grupo nos permite realizar otras personalizaciones en los grupos con opciones como **-a** y **-d** que añadirá o eliminará respectivamente un usuario al grupo. Podremos agregar más de un usuario al grupo, es decir crear una lista de miembros de grupo con la opción **-M**. Por último si queremos implementar algo más de seguridad al grupo, a parte de tener una contraseña, sería bueno que el grupo fuera administrado por uno o varios usuarios, algo que conseguiremos con la opción **-A**. Con **-R** podremos restringir el acceso al grupo de forma temporal (*newgrp*) solo los usuarios que conozcan la password podrán acceder al grupo. Si la opción *-R* es definida la contraseña del grupo será seteada con el signo de admiración '**!**'.

Algo muy común en la administración de grupos es el añadir miembros a un grupo. A continuación vamos a nombrar diferentes formás de hacerlo.

- Añadir un usuario que ya existe a un grupo que también existe:

```
$ sudo adduser <usuario> <grupo>
```

- Añadir un usuario que ya existe a un grupo que también existe. Es similar al comando anterior y además es el que deberemos de acostumbrar a usar:

```
$ sudo gpasswd -a <usuario> <grupo>
```

- Añadir a uno o varios grupos existentes usuarios de nueva creación:

```
$ adduser | useradd [opciones de creación de usuario] -G
grupo1,grupo2,grupoN...
```

- Añadir un usuario que ya existe a uno o varios grupos a la vez:

```
$ sudo usermod -G grupo1,grupo2,grupoN <usuario>
```

Nota: En este ultimo caso deberemos de especificar la lista de todos los grupos a los que ya pertenece el usuario además de los nuevos grupos de los que queremos que forme parte. De no especificar todos sus grupos anteriores, aquel que no indiquemos será eliminado, por lo que este comando también nos puede servir para eliminar a un usuario de uno o varios grupos.

- Acabamos de ver en la nota anterior como eliminar a un usuario de uno o varios grupos, pero la mejor forma de hacerlo es la siguiente:

```
$ sudo gpasswd -d <usuario> <grupo>
```

- O igualmente con la interfaz para userdel de algunas distribuciones Debian:

```
$ sudo deluser <usuario> <grupo>
```

Eliminar cuentas de usuario

Eliminar una cuenta de usuario no es complicado, pero deberemos de estar atentos para no dejar rastros de los usuarios, como son sus archivos o directorios creados, cuentas de correo, directorio home, etc… Si no queremos eliminar todo su contenido deberemos de hacer algo con él, para evitar que nuevos usuarios creados en el sistema puedan adoptar su *UID* (algo que a menos que se fuerce no suele ser muy normal) y tengan acceso completo a todos los archivos. A continuación vamos a ver como eliminar una cuenta de usuario y ver que hacer con todo aquello que queremos o no conservar.

Para eliminar cuentas disponemos de la utilidad **userdel** y de su interfaz sencilla *deluser* (en teoría solo para algunas distribuciones Debian). Ambas herramientas son sencillas de utilizar por lo que vamos a explicar el funcionamiento de de las dos.

- Eliminar la cuenta de usuario de *e.torres* eliminando a demás su directorio */home* y con ello todos los archivos en él creados, además de su directorio *mail*.

```
$ sudo userdel -r e.torres
```

- La misma acción pero más agresiva (eliminará no solo el directorio home del usuario, si no cualquier archivo con su *UID* en el sistema) podremos realizarla con *deluser* y su opción *–remove-all-files* o *–remove-home* si queremos dejar intacto todo archivo creado por el usuario fuera de su directorio home

```
$ sudo deluser --remove-all-files e.torres
```

Nota: Ambos comandos aceptan la opción **-f** con la que podremos eliminar un usuario que se encuentre dentro del sistema e incluso al mismísimo *root*.

- Si queremos eliminar un usuario con *userdel* y dejar intactos sus archivos bastará con no pasar la opción **-r**
- En este caso el comando *deluser* ofrece una ventaja importante y es que nos permite realizar copias de seguridad que por defecto se crearán en el directorio raíz */nomusuario.tar.gz* o */nomusuario.bz2* (opción **–backup**) o incluso redirigir la copia a un directorio concreto con **–backup-to** <*directorio*>

Nota: *deluser* nos permite eliminar un grupo (no el primario de un usuario) con su opción **–group** y podemos asegurarnos de que eliminamos un grupo que se encuentra vacío si lo comprobamos antes con la opción **–only-if-empty**

Una vez eliminada la cuenta con *userdel -r* podremos comprobar con el comando **find / -uid <UID>** si el usuario a dejado rastro por algún directorio del sistema y eliminarlos o decidir que hacer con ellos.

Algunos servidores entre los que destaca **Samba**, guardan su propia lista de usuarios. Si emplea un servidor de este tipo, es preferible eliminar la entrada del usuario de la lista de usuarios del servidor al eliminar la cuenta principal del usuario. En el caso de *Samba* esto se suele hacer editando manualmente el archivo *smbpasswd*, que suele encontrase en */etc/*, */etc/samba* o */etc/samba.d* y eliminar la línea correspondiente al usuario en cuestión o bien utilizando el comando **smbpasswd** y su opción **-x**

Eliminar grupos

Los grupos se eliminan con el comando **groupdel** o como siempre dependiendo de la distribución podremos usar su interfaz sencilla **delgroup** e incluso **deluser –group**. Igualmente podremos eliminar un grupo editando el archivo */etc/group* y */etc/gshadow* si el grupo tiene contraseña. El comando *groupdel* verifica si el grupo es el grupo primario de algún usuario, en tal caso rehusará eliminarlo; tendrá que cambiar primero el grupo primario del usuario o borrar su cuenta. Como ocurre al eliminar las cuentas de usuario, puede que existan archivos y directorios en el sistema con el *GID* del grupo, lo mejor será buscarlos con **find** y modificar el grupo con **chown**.

Poner a punto entornos de sistemás y usuarios

Como ya comentamos al inicio del tema de creación de usuarios, los entornos de tipo texto de los usuarios se controlan a través de los archivos de configuración de la consola. En **bash** estos archivos son */etc/profile*, */etc/bash*, */etc/bash.bashrc*, *~/.profile* o *~/.bash_profile* y *~/.bashrc*.

Los archivos de */etc* son archivos de configuración global, los del directorio principal de los usuarios afectan a las cuentas de los usuarios de manera individual y son copiados en el en home a la hora de crear la cuenta. Estos archivos controlan las distintas opciones de *bash*, incluyendo las variables de entorno. Podemos ver todas las variables de entorno definidas para una cuenta concreta ejecutando la orden **env**.

Bash mantiene algunas variables de entorno automáticamente como **PWD**, por lo que no debería intentar definirla en un script de configuración adicional. La modificación de los archivos de configuración de *bash* solo afectan a *bash* por lo que si un usuario no utiliza una consola en modo texto o usa otro *shell* distinto de *bash* definir las variables de entorno en archivos de configuración de *bash* no le será de ayuda.

Algo de información extra

Truco para recuperar la contraseña de root

Existen varias formás de hacerse con el control de *root* incluso cuando hemos sido olvidadizos y no recordamos la contraseña de nuestro *superusuario*. Vamos a explicar una forma sencilla pero eficaz de acceder a *root* sin saber su contraseña. Para esto necesitaremos un *LiveCD* o herramientas de rescate como *SystemRescueCd*, otro usuario en el sistema y además conocer la contraseña de este. Suponiendo que contamos con lo anterior, comenzaremos iniciando con el sistema de recuperación de emergencia y localizaremos el archivo */etc/shadow*. De este archivo copiaremos la contraseña encriptada del otro usuario y la pegaremos en el campo de contraseña de *root*. A continuación podremos iniciar con *root* y teclear la contraseña del usuario. Una vez dentro se recomienda cambiarla con el comando *passwd*.

En una situación en la que ni siquiera contamos con otro usuario podremos vaciar el campo de contraseña de *root* para dejar el sistema si *password*. Una vez que accedamos lo mejor será crear una nueva contraseña.

Sobre los archivos esqueletos

Proporcionan un conjunto central de archivos de configuración que deben estar presentes en los directorios principales de los usuarios cuando estos se creen. Proporcionan un punto de partida para que los usuarios modifiquen sus archivos de consola y características de su perfil (~/.bashrc) e incluso algunas aplicaciones de terceros pueden exigir que definamos ciertas variables en alguno de estos archivos para su correcto funcionamiento (normalmente en ~/.profile o ~/.bash_profile), otros son usado para guardar los últimos comandos usados en la consola (~/.bash_history) o realizar tareas al finalizar la sesión en una terminal, como puede ser limpiar la pantalla y mostrar el prompt de nuevo (~/.bash_logout).

Nota: El archivo *~/.bashrc* es ejecutado cuando invocamos un *shell* desde consola por lo que ajustará la configuración para este nuevo *shell* mientras que el archivo *~/.bash_profile* o *~/.profile* es ejecutado cuando el usuario inicia sesión por lo que será quién controle la configuración principal del *shell* de inicio del usuario. La configuración de estos archivos solo afectan a *bash*.

Podremos añadir al directorio /etc/skel archivos de configuración, de presentación (tipo README) e incluso un árbol de directorios.

Bases de datos de cuentas de red

Muchas redes emplean bases de datos de cuentas de red como por ejemplo **NIS** (*Network Information System*, Sistema de información de la red), su predecesor **NIS+**, **LDAP** (*Lightweight Directory Access Protocol*, Protocolo Ligero de Acceso a Directorios), dominios de **Kerberos**, dominios de **Windows NT4.0** o dominios de **Directorio Activo** de Windows (*AD*).

Todos estos sistemás tienen como función principal trasladar la gestión de la base de datos de las cuentas de usuarios a un mismo servidor centralizado. La ventaja de estos sistemás es que administradores y usuarios no tienen por que preocuparse de la gestión individual de cada cuenta. El uso de estos sistemás o semejantes implica que la mayoría de las cuentas no aparecerán en */etc/passwd*, */etc/group*, */etc/shadow* o */etc/gshadow* y digo la mayoría porque siempre estarán como mínimo las cuentas del sistema local (*daemon*, *man*, *mail* o incluso la de algún usuario no perteneciente al dominio pero que tiene una cuenta concreta en el sistema local).

Implementar un sistema de este tipo suele ser complejo. Entre otras cosas deberemos de instalar el software apropiado, modificar el archivo **/etc/nsswitch.conf** y los archivos de configuración **PAM** (*Pluggable Authentication Module*, Módulo de autenticación conectable) de */etc/pam.d*.

Tras implementar un sistema de autenticación basado en red, el comportamiento de utilidades como *passwd* o *usermod* quizás dejen de sernos útiles para crear estas cuentas de dominio.

Conclusiones sobre la administración de usuarios y grupos en Linux

Durante las secciones relacionadas con la creación, modificación y eliminación de cuentas de usuarios y grupos hemos estudiado algunos comandos y formás destinadas de realizar estas tareas. Hemos vistos comandos y sus similares en algunas distribuciones y hemos desaconsejado formás de realizar tareas de mantenimiento de usuarios de forma manual. A continuación vamos a ver un listado de las herramientas '*estándar*' para realizar el mantenimiento de usuarios y grupos y posteriormente las opcionales que funcionarán solo para algunas distribuciones.

Herramientas estándar:

- Crear, Eliminar y Modificar usuario: *useradd, userdel y* (*usermod, chage, passwd, chpasswd, chfn* y *chsh*), respectivamente.
- Crear, Eliminar y Modificar grupos: *groupadd, groupdel y* (*groupmod* y *gpasswd*), respectivamente
- Añadir o Eliminar usuarios a un grupo: *gpasswd -a* ó *gpasswd -d*
- Cambiar a grupo principal temporal: *newgrp*

Nota: El comportamiento de estos comandos puede ser modificado a través de lo archivos de configuración */etc/login.defs* y */etc/default/useradd*

Símiles de las herramientas estándar:

Nota: Son válidas solo en algunas distribuciones y pueden cambiar el comportamiento de una a otra distribución.

- Crear y Eliminar usuarios: *adduser* y *deluser*
- Crear y Eliminar grupos: (*addgroup* ó *adduser –group*) y (*delgroup* ó *deluser –group*), respectivamente.
- Añadir y Eliminar usuarios a un grupo: *adduser* <*usuario*> <*grupo*> y *deluser* <*usuario*> <*grupo*>

Nota: Estos comandos suelen ser más sencillos de utilizar, puesto que llevan configuradas opciones para pasar pocos argumentos por línea de comando y realizar muchos cambios e incluso son interactivo, es decir se realizan una serie de preguntas sobre el usuario cuando este es creado. Los archivo de configuración para estas herramientas son */etc/adduser.conf* y */etc/deluser.conf* aunque pueden existir otros como */usr/local/sbin/adduser.local* o */usr/local/sbin/deluser.local* que de existir serán ejecutados tras crear o eliminar las cuentas.

Como último de los comandos para la administración de grupos y usuarios estudiados en este tema mencionaremos el comando **getent** el cual nos permite imprimir por

pantalla (o redireccionar a un archivo) el contenido de las bases de datos de gestión de usuarios como lo son */etc/passwd*, */etc/group* y */etc/shadow*;

```
getent passwd | group | shadow
```

además de permitirnos ver el contenido de otros archivos como */etc/hosts*, los servicios con su puerto y protocolo, los protocolos o redes;

```
getent hosts | services | protocols | networks
```

Nota: Estos últimos serán estudiados más a fondo en el "*Capítulo 8 – Configuración básica de redes*"

SU y SUDO

Tanto el programa **su** como el programa **sudo** nos permiten ejecutar ordenes con privilegios de *root* o en el caso de *su*, ingresar al sistema con la cuenta de otro usuario. Dicho esto, las situaciones típicas para usar estos comandos serán para el caso de *su*:

1. Estamos logueados con nuestro usuario estándar y necesitamos ejecutar una o varias tareas como *root*, ya sean ejecutar binarios, modificar archivos crearlos o eliminarlo
2. Estamos con un usuario cualquiera, incluido *root*, y nos vemos en la necesidad de pasarnos a otro usuario

Para el uso de *sudo*, tendremos que vernos en la necesidad de obtener privilegios de *root* de forma temporal para realizar tareas administrativas.

El nombre de *su* deriva de la abreviatura **S**uper**U**ser o en otros casos leeremos que viene de **S**ubstitute **U**ser. El nombre de *sudo* es la abreviatura del Inglés **S**ubstitute **U**ser **do**.

Ambos comandos nos facilitan o simplifican la tarea de tener que salir de nuestra sesión y volver a iniciar con la de un determinado usuario incluido *root*. Imaginemos que estamos con nuestro usuario estándar y necesitamos modificar el archivo */etc/passwd* para el que solo *root* tiene permisos de escritura. Sin tener la posibilidad de ejecución de estos dos comandos deberíamos de salir de la sesión y volver a loguearnos con los credenciales de *root*, pero se nos facilita la tarea con tan solo escribir **$ su –** y posteriormente editar el archivo, bastará escribir **# exit** para volver a nuestra sesión anterior o bien ejecutar el comando de edición directamente tras *sudo*, tal que así: **$ sudo vi /etc/passwd**

Para ambos comandos existen algunas peculiaridades. Por ejemplo podríamos haber ejecutado '**su**' sin el guión '**–**' pero de hacerlo así, habríamos accedido a la cuenta de *root*, sin ejecutar sus archivos de configuración de pérfil y sin cambiar de directorio de

trabajo al directorio de trabajo por defecto de root (*/root*). Si pasamos de una cuenta de usuario a otra, ya sea a la cuenta *root* o a la cuenta de otro usuario estándar necesitaremos introducir su contraseña, solo *root* podrá acceder mediante '*su*' a cualquier cuenta del sistema sin precisar de la contraseña de la cuenta de usuario a la que se mueve. Para el caso de '**sudo**' necesitaremos que nuestro usuario estándar cuente con el derecho de poder ejecutar *sudo* y además de editar el archivo, es decir, los permisos de quien usa *sudo* y como lo hará se definen en el archivo **/etc/sudoers**. Una línea demásiado permisiva de este archivo sería:

```
%sudo ALL=(ALL:ALL) ALL
```

Esta línea indica que cualquier usuario perteneciente al grupo *sudo* podrá ejecutar cualquier comando de *root* bajo la utilidad sudo.

Hablaremos sobre este archivo en el capítulo destinado a la seguridad (*Capítulo – 10*)

Nota: El comando *su -c <comando>* suele ser muy utilizado en scripts de inicio de servicios para iniciar un servicio bajo el nombre de otro usuario.

Monitorizar el sistema mediante sus archivos de registro

Linux mantiene archivos de registros (*logfiles*) a los que se reportan sucesos relevante sobre el funcionamiento del sistema, ya sea como del propio *kernel* o de *servicios de red* que se encuentran ejecutados de forma permanente en segundo plano (*demonios*). Estos archivos de registros son especialmente útiles para detectar fallas en el sistema, o incluso prever estas mediante la monitorización de archivos claves o concretos de un servicio. Una de las obligadas tareas que tiene un administrador de sistemás es precisamente revisar estos archivos de registro periódicamente para poder anticiparse a problemás venideros o por ejemplo conocer las causas de porque un servicio no se ejecuta de forma correcta.

Una particularidad común de todos los sistemás de registro y los archivos en los que se acumula la información es que crecen, de hecho pueden crecer tanto que dejen el sistema operativo '*no operativo*' dependiendo de donde se estén guardando estos archivos. Por ejemplo si tenemos una partición '/' (*raíz*) en la que se encuentra el directorio */var* y bajo este lo normal es encontrar los archivos de registro (*/var/log/**) puede hacer que el espacio asignado para la partición raíz se llene impidiendo incluso el acceso de nuevos usuarios al sistema o trabajando de una forma extraña. Por ello el administrador del sistema está obligado a definir una política de seguridad ante tal problema. La política que se implemente estará definida de acuerdo al esfuerzo de administración que conlleve o a la importancia que la organización le preste a tal problema, entre otras. Algunas de las políticas suelen ser:

- **No almacenar archivos de registro**. Evidentemente esta opción es muy poco recomendable, aunque evidentemente se establecerá o no, en función de la criticidad del sistema.
- **Resetear los archivos periódicamente**. Puede ser una medida a corto plazo contra el llenado de espacio de un disco o partición, pero a largo plazo puede ser desastrosa, ya que no contaremos con los archivos de registro que quizás necesitemos en función del tiempo que haya pasado.
- **Rotar los archivos en función del tiempo transcurrido**. Esta suele ser una de las medidas más adoptadas por administradores de sistemás. Es una política configurable desde los propios archivos de configuración del sistema de registro que estemos usando. Una vez rotado el archivo, el antiguo es comprimido (ahorrando espacio en disco) y quizás llegue el momento en el que sea automáticamente eliminado dependiendo del tiempo de rotación y cantidad de archivos rotados almacenados en el disco que hayamos definidos. Algunos sistemás de registro como *Journal* comprimen directamente los archivos de registros.
- **Almacenar los archivos de registro**. Si disponemos de una unidad externa de almacenamiento o partición destinada a copias de seguridad, podemos guardar nuestros archivos de registro en dicha unidad y acceder a ellos cuando fuese necesario.

Nota: Podemos jugar con las opciones de rotar y almacenar para encontrar una solución definitiva.

Existen diferentes sistemás de registro para *Linux*. Vamos a estudiar el demonio **syslogd**, **syslog-ng** (*syslog de nueva generación*), **rsyslog** y **Journal**

Syslog

Es un sistema que procura centralizar el manejo de los registros de eventos que generan los diversos programás que corren bajo un sistema *Linux*. Por un lado facilita a los desarrolladores de aplicaciones la generación y el manejo de mensajes a registrar, y por otro lado facilita a los administradores de sistema el manejo de forma centralizada de los mensajes provenientes de diferentes aplicaciones. *Syslog* clasifica los mensajes por origen e importancia, y los envía a diferentes destinos como pueden ser archivos, la terminal o eventualmente a un comando que lo reenvíe a direcciones de correo electrónico o paginador.
Syslog permite manejar mensajes originados en diferentes sistemás de la red.

Los componentes más importantes de **syslog** son:

- **syslogd**: el daemon que recibe los mensajes y los almacena de acuerdo a la configuración del archivo */etc/syslogd.conf*

- **openlog()**, **syslog()** y **closelog()**: rutinas de la biblioteca C estándar para permitir la comunicación entre *syslog* y el programa.
- **logger**: comando de usuario para agregar un mensaje a un archivo de registro

Instalación y configuración de syslogd

El demonio **syslogd** es uno de los primeros que se lanza cuando el sistema se incia, para comenzar a recibir mensaje desde los diferentes servicios de red y registrarlos en sus respectivos archivos de registro de acuerdo con lo especificado en su archivo de configuración.

En ocasiones se suelen confundir o comparar los demonios *syslogd* y **klogd**, este ultimo registra los eventos del *kernel*. Ambos demonios son instalados mediante el mismo paquete **sysklogd** desde los repositorios oficiales.

Una vez tengamos instalado el sistema de registro tendremos que configurarlo, esto como mencionamos anteriormente lo haremos desde el archivo de configuración **/etc/syslogd.conf**. Este archivo tiene un formato sencillo pero ofrece un gran potencial. Las líneas adoptan la siguiente forma:

```
recurso.prioridad acción
```

En algunos documentos, libros, wikis, etc… podremos leer *facility.level* en vez de su traducción *recurso.prioridad*, al final ambos identifican a un **selector**. Es decir un *selector* estará formado por un **recurso** que no es más que el código del tipo de programa que generó el mensaje y la **prioridad**, que será igualmente un código que identifique la importancia que tendrá ese mensaje. El campo **acción** decide el que se hará con todos los mensajes que se identifiquen con un determinado selector (*recurso.prioridad*).

Vamos a ver los posibles valores de cada campo y seguidamente expondremos un ejemplo para ver esto de una forma más clara.

- **recurso**: Suelen ser valores prefijados por el sistema e identifican a uno o varios servicios como *auth* (mensajes relacionado con la seguridad), *mark* (reservado para uso interno), *mail*, *cron*, *daemon*, *lpr*, *ftp*, *news*, *syslog*, *uucp* y desde *local0* hasta *local7* (usados con cierta libertad por el usuario para diferentes aplicaciones). No todos los recursos se encuentran aquí enumerado. Si quisiéramos especificar más de un recurso para una *prioridad* y *acción* concreta en una sola línea del archivo se utila el caracter ‘**,**‘ (*coma*) y si queremos definir todos los *recursos* para una *prioridad* y *acción* ‘*****‘ (*asterísco*)
- **prioridad**: Con este campo seleccionaremos que mensajes queremos incluir en uno u otro archivo de registro para uno o varios recursos. Los códigos de *prioridad*, listados de menor a mayor prioridad son: *debug*, *info*, *notice*, warning

(warn), *err* (error), *crit*, *alert* y *emerg* o *panic* (este último en desuso, al igual que warn y error). La prioridad *debug* registra la mayor parte de la información (pensado para depurar programás) y en el extremo opuesto *emerg*, que registrará los mensajes para problemás muy serios. Un aspecto importante a tener en cuenta es saber que mensajes serán guardados en los archivos de registro. Todos los mensajes emitidos por los recursos se acompañan de un código de prioridad, y serán registrados por defecto siempre y cuando el código sea igual o superior (esto es modificable) al indicado en el archivo de configuración para ese determinado recurso. A continuación veremos esto con un ejemplo.

- **Acción**: Existen varias opciones para este campo, la más utilizada es la de guardar los registros en un archivo, el cual deberá de estar creado de antemano y bastará con indicar el path completo. Podemos enviar los registros al demonio *syslogd* de otra máquina escribiendo '*@<nombre máquina o IP>*' o reenviar a la terminal de un usuario siempre y cuando este esté logueado indicando como acción un archivo de dispositivo de consola (/dev/console). En esta última opción podremos separar diferentes usuarios con el caracter '**,**' o marcar a todos con un asterisco '*****'. Algunos sistemás permiten enviar el mensaje a la entrada estándar de un comando mediante un pipe '|'.

Los espacios entre los selectores (*recurso.prioridad*) y la acción suelen ser tabulaciones. Vamos a clarificar con algunos ejemplos las situaciones descrita en los puntos anteriores.

- Registrar todos los mensajes (*****) del recurso *mail* al archivo */var/log/mail*:

```
mail.*          /var/log/mail
```

- Enviar solo los mensajes con prioridad *notice* para los recursos *news* y *mail* a la consola principal del sistema:

```
news,mail.=notice          /dev/console
```

Nota: Podemos separar varios recursos son '**,**'. Si en el archivo de configuración existe también la entrada del ejemplo anterior, los mensajes de *mail* para prioridad *notice* serán enviados a la consola y registrados en */var/log/mail*. Podríamos haber cambiado */dev/console* por '*****' (asterisco) para enviar los mensajes a todas las consolas de modo texto abiertas en el sistema.

- Registrar todos los mensajes con prioridad *crit* o superior (es decir; *crit*, *alert* y *emerg*) del recurso *daemon* en */var/log/error* y los mensajes con prioridad *warn* o *menor* (es decir; *warn*, *notice*, *info* y *debug*) para el recurso *lpr* en */var/log/lpr-info*. Si queremos especificar los mensajes con prioridad igual o menor a una determinada prioridad lo haremos con el caracter de admiración '**!**':

```
daemon.crit          /var/log/error
lpr.!warn          /var/log/lpr-info
```

- Ahora vamos a ver como manejar diferentes mensajes según su prioridad para un recurso con acciones diferentes. En este ejemplo vamos a enviar con la primera línea todos los mensajes generados por el *kernel* al archivo */var/log/kernel*, con la segunda línea indicaremos que además de registrarse en el archivo, aquellos que tengan una prioridad *crit* o superior se enviarán a *syslogd* de otro sistema (el otro sistema tiene que estar configurado para tal fin) y además esos mismos mensajes también serán impresos por terminal gracias a la tercera línea. Por último haremos que todos los mensajes comprendidos entre la prioridad *info* y *err*, además de ser enviados al archivo */var/log/kernel* gracias a la primera línea, serán escritos en el archivo */var/log/kernel-info*:

```
kern.*        /var/log/kernel
kern.crit        @unamaquina.enundominio.es
kern.crit        /dev/console
kern.info;kern.!err        /var/log/kernel-info
```

Nota: Vemos como hemos seleccionado dos selectores separados por el caracter ‘;’ para ser comprendidos por una misma acción

Añadir registros manualmente

Cuando mencionamos anteriormente los componentes más importantes de *syslog* vimos que **logger** es la herramienta que nos permite crear registros de forma manual.

```
$ logger ejemplo del funcionamiento de logger
```

El resultado sería seguramente una entrada en el archivo */var/log/message* con el siguiente contenido:

Feb 11 18:02:27 LiMinCinn nebul4ck: ejemplo del funcionamiento de logger

Rotar archivos de registro

La rotación del registro se controla a través del archivo */etc/logrotate.conf* en el que se suele incluir la referencia a los archivos bajo */etc/logrotate.d/*. Las entradas de estos archivos le indican al sistema si debe rotar los registros a intervalos fijos o cuando estos almacenen un tamaño concreto. Cuando un registro rota, se renombra y dependiendo de la configuración se comprimirá o no, se creará uno nuevo e incluso puede que se borre el archivo de registro comprimido más antiguo de los existentes.

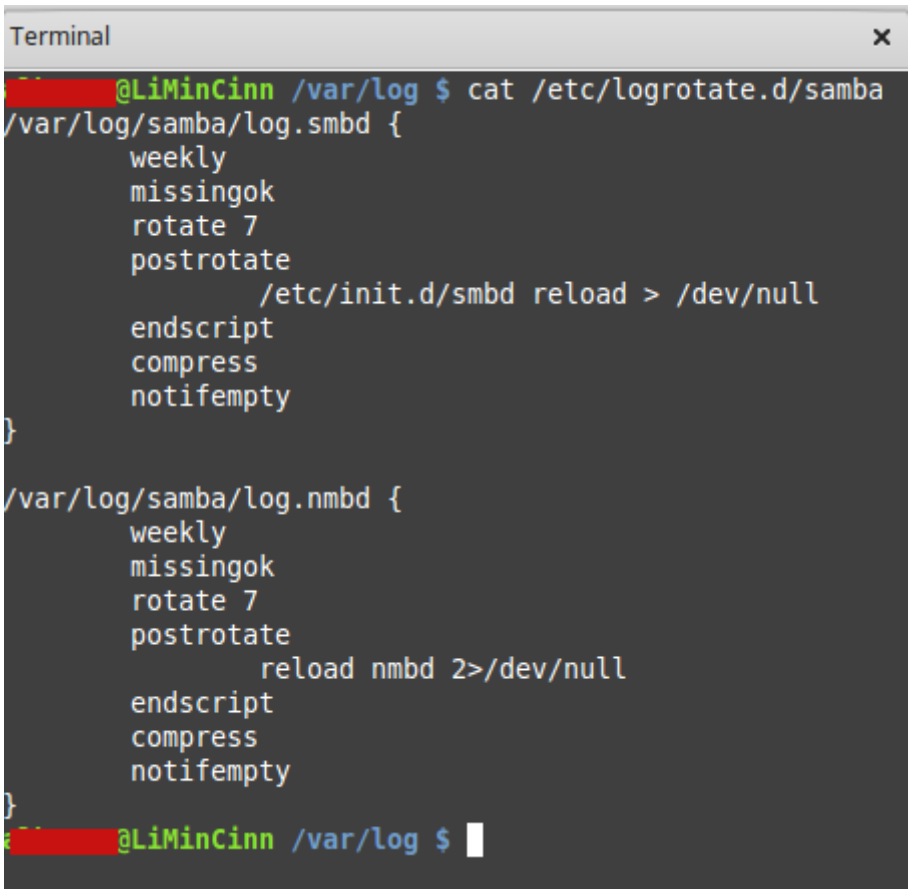
Para rotar archivos de registros necesitaremos tener instalado el paquete **logrotate**, tener una buena configuración y ser lanzado periódicamente, algo de lo que se encargará *cron* o en su defecto (ya que esta tarea es ejecutada por las noches y la mayoría de los pc de

usuario duermen en estas horas) *anacron*, pero de esto hablaremos en las próximás secciones.

Cuando se invoca *logrotate* este consulta su archivo de configuración (o archivos en caso de que este referencie a otros como */etc/logrotate.d/**) y actuará en función de los ajustes que en él o ellos encuentre. Un archivo de configuración para *logrotate* podría tener el siguiente aspecto:

```
# Rotar los archivos de registro de forma semanal
weekly
# Usar el grupo syslog por defecto
su root syslog
# Conservar los registros antiguos durante 4 semanas
rotate 4
# Crear nuevos archivos de registro tras la rotación
create
# Comprimir los archivos de registros antiguos
compress
# Incluir configuraciones para archivos de registros de servicios
específicos
include /etc/logrotate.d
# Algunas otras opciones
notifempty
nomail
noolddir
# Rotar archivos de registro para wtmp el cual no está controlado por
un archivo de
# configuración individual
/var/log/wtmp {
       missingok
       monthly
       create 0664 root utmp
       rotate 1
}
```

Si se especifica la inclusión de archivos de configuración bajo */etc/logrotate.d* significará que habrá servicios que configuren sus propios archivos de registro de los que además serán los propietarios. Algunos de los servicios que tienen tal comportamiento podrían ser: *apt*, *aptitude*, *yum*, *dpkg*, *samba*, *ufw* y *upstart*. A continuación pondremos uno de estos archivos a modo de ejemplo.



El archivo de configuración para samba incluye dos servicios: **smbd** y **nmbd** ambos con sus respectivos archivos de configuración bajo el directorio */var/log/samba* y con su propia configuración.

Algunas de las características más importante de los archivos de configuración individuales para *logrotate* están contemplados en la imagen anterior, aun así vamos a nombrar algunos más.

- **Nomenclatura de los archivos rotados**: Por defecto a los archivos rotados se le asigna un digito como extensión al nombre del archivo por ejemplo: *smbd.log.3* (indica que este archivo a rotado un total de tres veces). Podemos cambiar este comportamiento con la opción *dateext*
- **Opciones de compresión**: Definida en nuestro ejemplo con la palabra *compress*. Por defecto se comprimen con *gzip*, si queremos usar por ejemplo *bzip* podemos definirlo así: *compresscmd bzip2* e indicar opciones del comando, como la tasa de compresión con *compressoptions*
- **Crear un nuevo archivo de registro tras la rotación**: Podemos hacer que se cree un nuevo archivo con la opción *copytruncate* que lo que realmente hace es copiar el archivo antiguo y vaciar su contenido, o por ejemplo pasar un usuario propietario y unos permisos (esto no siempre funciona): *create 644 samba samba*

- **Opciones temporales**: Con *daily*, *weekly* y *monthly* provocaremos que los registros roten *diariamente*, *semanalmente* o *mensualmente* respectivamente.
- **Opciones de tamaño**: Si no queremos rotar en función del tiempo podremos hacerlo especificando un tamaño máximo de archivo de registro, por defecto el valor es en *bytes* aunque podemos cambiar esto con los sufijos **k** o **M**. *size 100k* para que el archivo rote cuando haya llegado al tamaño de 100Kilobytes
- **Opciones de rotación**: Con *rotate* '*número*' indicaremos cuantos archivos de registros antiguos queremos conservar. Si indicamos **rotate 3**, tendremos *smbd.1.gz*, *smbd.2.gz* y *smbd.3.gz*. Cuando el que actualmente está registrando los mensajes llegue a su tamaño de cota o al tiempo límite de rotación pasará a llamarse *smbd.1.gz* y *smbd.3.gz* será eliminado. Los otros dos archivos pasarán del 1 al 2 y el otro del 2 al 3.
- **Opciones de correo**: Utilizando *mail* '*dircorreo*' podremos enviar el archivo que rota por correo electrónico. Con *nomail* no enviaremos ningún correo.
- **Scripts**: Las palabras claves *prerotate* y *postrotate* indican que se ejecute una acción (comando) antes o después de haber rotado un archivo de registro. En nuestra imagen tenemos el ejemplo. Para finalizar la ejecución de los comandos lo haremos con *endscript*.

Revisar el contenido de los archivos de registro

Para revisar un archivo de registro primero deberemos de encontrar el archivo y luego buscar la información deseada dentro de este. Lo normal es que los archivos de registro se guarden bajo el directorio */var/log*, que a su vez como hemos visto en el ejemplo de *samba* puede haber un directorio para un servicio concreto y dentro de este encontremos el archivo de registro. Utilizaremos normalmente las herramientas *cat*, *tail* y *head*, o los paginadores *less* y *more*.

La mayoría de mensajes enviados por el sistema suelen guardarse en el archivo */var/log/messages*, archivo que podremos desplegar con el comando **dmesg**. Otros archivos de registros importantes son *auth.log*, *utmp*, *wtmp* y *lastlog*. Linux mantiene un registro en los archivos **utmp** y **lastlog** sobre información de los usuarios como por ejemplo si un usuario está logueado en el sistema, en que terminal y desde cuando. Estos archivos son actualizados en cada login y logout guardando además esta información en el archivo **wtmp** que mantendrá entonces un histórico sobre los inicios de sesión de los usuarios. Al ser una archivo que va creciendo deberemos de tenerlo contemplado en la configuración de logrotate.
Estos archivos están en formato binario pero podrán ser desplegados con los comandos **who**, **w** y **lastlog**. El comando *who* desplegará el contenido del archivo *utmp* por lo que veremos los registros de los usuarios actualmente dentro del sistema, en que terminales se encuentran *(pts/<número>*, si iniciaron una terminal virtual desde el entorno gráfico o *tty<número>* si se lanzó la consola al inicio de sesión. Podremos ver también la terminal actual con el comando **tty**) y la hora de inicio de sus sesiones. El comando *w* se comporta de la misma manera pero nos ofrece algo más de información como por

ejemplo el *display* utilizado. Si queremos desplegar el contenido del archivo de registro *lastlog* bastará con invocar a *lastlog*, el cual depslegará una lista de todos los usuarios de */etc/passwd* mostrando su último inicio de sesión (para los usuarios nunca logueados aparecerá: ***Never logged in***). Para desplegar el contenido del archivo *wtmp* (histórico de login y logout de usuarios) pasaremos como parámetro el nombre del archivo *wtmp* al comando *who:*

```
$ who /var/log/wtmp
```

o bien utilizando directamente el comando *last* o *lastb* para ver el histórico de logueos e incluso logueos fallidos, respectivamente.

Nota: Los comandos *w*, *who*, *last* y *lastb* nos muestran como mínimo el *usuario*, la *terminal*, el *display* y la *hora* de inicio. El comando *w* además muestra la *carga del sistema* y el *número de usuarios* conectados. *who* y *last* muestra además la *fecha*.

La ubicación de estos archivos varía de un sistema a otro. El archivo *utmp* habitualmente está en el directorio */etc* o */var/run*, el archivo *wtmp* suele estar en */var/log* o */var/adm* al igual que el archivo *lastlog*.

Una vez identificado el archivo de registro necesitado podremos desplegar su contenido con uno de los paginadores conocidos (*more* o *less*), buscando contenido específico con *grep*, o *tail* y *head* que podrán sernos igualmente útiles y si queremos dejar monitorizando un archivo de registro específico ya sabemos que la opción *-f* de *tail* nos permite tal acción.

Existen herramientas avanzadas de análisis del registro como **Logcheck** que forma parte del paquete *Sentry Tools* o Whatlogh

Syslog-ng

Syslog-ng (*Syslog de nueva generación*) es una implementación del antiguo *syslog* en la que se mejoran algunos aspectos como la capacidad de filtrado mediante contenido, su flexibilidad de configuración, uso de *TCP* en lugar de *UDP* y la opción de crear túneles de conexión cifrada bajo *SSL/TLS*.

El funcionamiento de *syslog-ng* es básicamente como el de *syslog*, es decir, los mensajes generados por una *fuente* ‘***source***‘ son enviados a una dirección ‘***destination***‘ a través de una serie de filtros ‘***filter***‘ basados en el contenido. *Source*, *filter* y *destination* forman el **logpath** cuyo equivalente en *syslog* sería el *Selector* (*recurso.prioridad*) + *Acción*. El archivo de configuración de *syslog-ng* es */etc/syslog-ng/syslog-ng.conf*. No obstante como en otros servicios *Linux*, contamos con un directorio en el que almacenar diferentes configuraciones individuales para

determinados servicios, en este caso para separar los distintos *filtros* y *plantillas* para el procesamiento de reglas.

Antes de pasar al archivo de configuración vamos a ver los distintos '*source*' y '*destination*' así como la manera de poder *filtrar* los mensajes.

Algunos de los *source* desde los que podremos reenviar los mensajes de registro son: tuberías '*pipe*', archivos '*file*', '*internal*' (solo escucha los mensajes de la máquina local), o los protocolos *TCP* y *UDP*.

Para filtrar los mensajes podremos usar los siguientes filtros:

- **facility**: Servicio que origina el log. Al igual que con *syslog* pueden ser algunos como *kern*, *user*, *mail*, *lpr*, *daemon*, etc…
- **level**/**priority**: Prioridad del mensaje. Las mismás que para *syslog*. *none* (no envía mensajes), *debug*, *info*, *notice*, *warning*, *err*, *crit*, *alert* y *emerg*.
- **program**: Filtrará los mensajes de aquellos programás que coincidan con una expresión regular dada.
- **host**: Filtra los mensajes de aquellos ordenadores cuyo nombre coincida con una expresión regular dada.
- **filter**: Con este filtro podremos invocar a otro filtro.
- **match**: Aplica una expresión regular

Los destinos serán los que especifiquen a donde y como serán enviados los mensajes.

Logpath: *Source, filter* y *Destination*

No vamos a entrar en muchos detalles del archivo de configuración, pero si que conoceremos los detalles básicos que nos permitirán leer el archivo con cierta claridad. Lo primero a tener en cuenta es saber que el archivo **syslog-ng.conf** definirá unas opciones de orden general. Un ejemplo:

```
options {
    sync (0);
    stats (0);
    chain_hostnames(no);
    create_dirs (yes);
    dir_perm(0755);
    dns_cache(yes);
    keep_hostname(yes);
    log_fifo_size(2048);
    long_hostnames(no);
    perm(0644);
    time_reopen (10);
    use_dns(yes);
};
```

Una vez especificadas las opciones generales casi a gusto de consumidor, tendremos que definir las fuentes de las que queremos obtener mensajes, los filtros y los destinos. Existe una sintaxis concreta para definir una *fuente* y es la siguiente:

```
source <identificador> { source-driver_1(parámetros); source-
driver_2(parámetros); ... };
```

Algunos ejemplos de fuentes podrían ser:

- Permitir que las aplicaciones se comuniquen con el sistema de logs. Linux utiliza **unix-stream:**

```
source src { unix-stream("/dev/log"); internal(); };
```

Nota: Con *internal* estamos diciendo que solo filtre mensajes de nuestra propia máquina.

- Permitir que el *kernel* muestres sus mensajes:

```
source kernsrc { file("/proc/kmsg"); };
```

Nota: Podríamos haber unido ambas líneas en una sola y utilizar el driver de fuente *pipe* en vez de *file*. La diferencia es que la *pipe* abre el archivo en modo de *lectura-escritura* por lo que puede suponer una vulnerabilidad.

```
source src { unix-stream("/dev/log"); internal(); pipe("/proc/kmsg");
};
```

- Otras definiciones de fuentes podrían ser:

```
source s_net { udp(); };
source s_net { tcp(); };
source s_apache_ssl_access {
 file("/var/log/apache2/ssl_access.log" follow_freq(1) flags(no-
parse));
};
```

Al igual que las fuentes, los filtros y destinos tienen su propia sintaxis.

La sintaxis para los filtros es la siguiente:

```
filter <identifier> { expression; };
```

y algunos ejemplos son:

- Filtrar mensajes para el recurso *auth*:

```
filter f_auth { facility(auth); };
```

- Filtrar mensajes que vayan desde la prioridad *info* a *warning* y no incluya para estas prioridades los recursos *mail* y *news*:

```
filter f_messages { level(info..warning) and not facility(mail, news);
};
```

Para definir un destino utilizaremos la siguiente sintaxis:

```
destination <identifier> {destination-driver(params); destination-
driver(params); ... };
```

Y algunos ejemplos de destinos son:

```
destination authlog { file("/var/log/auth.log"); };
destination console { usertty("root"); };
destination remote_server { udp("10.0.0.2" port(514)); };
```

Ahora que ya conocemos como definir un recurso, un filtro y un destino vamos a formar el *logpath* que los conectará para formar un 'objeto' concreto. La sintaxis de *LogPath* se va a ver claramente con el siguiente ejemplo:

```
log { source(src); filter(f_mail); filter(f_info);
destination(mailinfo); };
```

Cuando tengamos listo el archivo de configuración reiniciaremos *syslog-ng* de acuerdo al sistema de inicialización que tengamos ejecutándose es decir, **service** (o script *SysV*), **initctl** (*Upstart*) o **systemctl** (*systemd*).

Rsyslog

Rsyslog es el remplazo de *syslog-ng* en muchas distribuciones por defecto. Es un eficiente y rápido sistema de procesamiento de registros del sistema que ofrece un diseño modular y niveles de seguridad que lo hacen ideal para muchos programás modernos. Suele incorporarse en las distribuciones más modernas (aquellas que aun no implementan *systemd*) como sistema de registro por defecto al ser suficientemente versátil y robusto para ser utilizado tanto en entornos empresariales como sistemás de escritorio. *rsyslog* mantiene compatibilidad con *sysklog*. Es posible utilizar el archivo de configuración *syslog.conf* para una configuración básica de *rsyslog*, pero no al revés, debido a que *rsyslog* implementa mejoras en su archivo de configuración que hacen posible por ejemplo crear plantillas que permitan almacenar los registros en BBDD *Mysql (MariaDB)* o *Postgresql*, seleccionar un formato específico de mensajes, crear archivos de nombre dinámico para almacenar los registros de un programa, etc…

Nota*: rsyslog* usa *logrotate* para su rotación de archivos.

El archivo de configuración principal de *rsyslog* es **/etc/rsyslog.conf** y por regla general suele tener una estructura básica compuesta de las siguientes secciones:

- **Configuración global**: Establece las propiedades globales del demonio *rsyslog* como el tamaño de la cola de mensajes (*$MainMessageQueueSize*) o la carga de módulos externos (*$ModLoad*). Cada línea contiene una propiedad de *rsyslog* y debe comenzar con el siglo del dólar '**$**'.
- **Plantillas**: Usadas para dar formato a los mensajes de salida o por ejemplo crear nombres de archivos de registro dinámicos en función de unas reglas definidas.
- **Canales de salida**: Son un nuevo concepto introducido en *rsyslog* que permiten definir una serie de propiedades específicas para uno o varios registros. Primero se define un canal y luego este es usado para la salida de un selector. La sintaxis de un canal es parecida a:

```
$canal de salida <nombre>,<nombre-archivo>,<tamaño-máximo>,<comando-
tamaño-máximo>
```

Donde:

– **nombre**: nombre del canal de salida

– **nombre-archivo**: nombre del archivo de registro

– **tamaño-máximo**: tamaño máximo para el archivo de salida

– **comando-tamaño-máximo**: nos permite ejecutar un comando cuando el archivo haya alcanzado el tamaño máximo. El comando va acompañado de un parámetro separado por un espacio

Para llamar a un canal de salida se usa una sintaxis como por ejemplo

```
*.* :omfile:$micanal
```

- **Reglas** (*Selector + Acción*): Las reglas definen en gran parte el comportamiento de los mensajes de registros. La sintaxis y sus opciones son las ya estudiadas anteriormente en *syslog.*

El archivo de configuración *rsyslog* es complejo y permite una gran variedad de acciones adicionales a *syslog*, no obstante podremos hacer funcionar un sistema de registro *rsyslog*, prácticamente con el formato del archivo de configuración *syslog.conf*

Un ejemplo de *rsyslog.conf* podría ser:

```
# /etc/rsyslog.conf Archivo de configuración para rsyslog.
# Podremos encontrar la definición de las reglas por defecto en
# /etc/rsyslog.d/50-default.conf
```

```
#### MODULES ####
$ModLoad imuxsock # Proporciona soporte para el sistema de registro
local
$ModLoad imklog # Soporte para el registro del kernel
# Si descomentamos las 2 siguientes líneas estaremos dando soporte al
registro
# de log remoto por UDP.
#$ModLoad imudp
#$UDPServerRun 514
# Recepción de registro remoto pero por TCP
#$ModLoad imtcp
#$InputTCPServerRun 514
# Habilitar los "non-kernel facility klog messages"
$KLogPermitNonKernelFacility on

#### GLOBAL DIRECTIVES ####
#
# Usar el formato tradicional de timestamp.
# Para habilitar los timestamp de gran precisión, descomenta la
siguiente línea.
#
$ActionFileDefaultTemplate RSYSLOG_TraditionalFileFormat
# Filtrar los mensajes duplicados
$RepeatedMsgReduction on
#
# Setear los permisos por defecto para todos los archivos de registro.
#
$FileOwner syslog
$FileGroup adm
$FileCreateMode 0640
$DirCreateMode 0755
$Umásk 0022
$PrivDropToUser syslog
$PrivDropToGroup syslog
#
# Directorio de trabajo para las colas rsyslog
#
$WorkDirectory /var/spool/rsyslog
#
# Incluir los archivos de configuración de /etc/rsyslog.d/
#
$IncludeConfig /etc/rsyslog.d/*.conf
```

Journal

Desde la incorporación del sistema de inicialización *systemd* en las principales distribuciones de Linux como *Red Hat* y *CentOS 7* y a las que le seguirán *Debian*, *Ubuntu* y *derivadas*, no es necesario el sistema de registro *syslog*, ya que *systemd* incorpora su propio sistema de registro llamado **journal**, aún así en algunas distribuciones de Linux coexisten ambos sistemás de registros, algo que es compatible y veremos a continuación.

journal recopila una serie de mensajes a partir de una serie de fuentes como:

- Los mensajes del *kernel* vía **Kmsg**
- Los mensajes básicos de *syslog* proporcionados por las llamadas de la librería *libc* de **syslog**
- Mensajes del sistema estructurados, proporcionados por la **API** nativa de *journal*
- Mensajes que provienen de la salida estándar (***stdout***) y salida de error (***stderr***) de los diferentes servicios
- Registros de auditorias entregados mediante el subsistema de auditoría.

Por defecto el registro de *journal* es volatil, es decir perderemos los datos una vez el sistema haya sido reiniciado. Este aspecto de *jounal* así como muchos otros pueden ser configurables desde sus archivos de configuración.

Configurando Journal

Journal es implementado mediante la unidad *systemd-journald.service.*

Los archivos de configuración de *journal* son leídos en el siguiente orden de preferencia: */etc/systemd/journald.conf*, */etc/systemd/journald.conf.d/.conf*, */run/systemd/journald.conf.d/*.conf y */usr/lib/systemd/journald.conf.d/*.conf*. Los parámetros y opciones que configuremos en estos archivos serán los que determinen el funcionamiento de *systemd-journald.service*.

Cuando se instala *journald*, por defecto sus paquetes y archivos se instalan sobre */usr/lib/systemd* pero serán los archivos de */etc/* los que use el administrador del sistema para sobreescribir los del paquete de origen, es decir, estos tienen preferencia sobre el resto. Los archivos de estos directorios (incluidos los de */run/systemd/*) están ordenados por orden alfabético por lo que si una misma opción con diferente valor es indicada en diferentes archivos, será el valor del último archivo el que tome *journald*. Se recomienda (al igual que en muchos archivos de configuración **.conf.d* de *Linux*) prefijar los nombres de los archivos con dos dígitos y un guión para mantener de forma más sencilla el orden de los archivos de configuración, algo como *20-archivo.conf*. Para deshabilitar un archivo de configuración, basta con que el administrador del sistema cree un enlace simbólico hacia */dev/null* con el nombre del archivo a deshabilitar en el directorio */etc/systemd/journald/*

Opciones para la configuración de journald

Opciones de disco:

- **storage**: El valor por defecto para esta opción en la mayoría de las distribuciones es '*auto*' que provocará que el log sea guardado de forma no persistente en */run/systemd/journal*. Podremos cambiar este comportamiento con la opción '*persistent*' que hará que el registro se guarde de forma persistente en */var/log/journal/*. No obstante si este directorio no existe está opción quedará invalidada y el registro continuará guardándose de forma no persistente en */run/systemd/journal*. Aun existiendo el directorio */var/log/journal/* podemos hacer que los registros sean volatiles con la opción '*volatile*' o incluso no guardarlos con '*none*'.
- **compress**: Esta opción nos permitirá comprimir los registros. Es una característica por defecto de *journal* y es por ello que *journal* utiliza su propia herramienta de visualización (descomprimidos en tiempo real) para ver los registros.
- **RateLimitBurst** y **RateLimitInterval**: Estas opciones permiten definir el número máximo de mensajes a guardar de cada servicio en un intervalo de tiempo dado (respectivamente). En caso de que el número de mensajes sobrepase el límite, estos no serán guardados pero podremos ver el número de mensajes perdidos mediante un registro automático, algo que nos permitirá ajustar mejor el número de mensajes por intervalo de tiempo. Si indicamos los valores *0* para estas opciones estaremos eliminando el límite de mensajes por intervalo de tiempo.
- **SystemMaxUse**, **SystemKeepFree** y **SystemMaxFileSize**: Definirán en gran medida el uso del disco destinado a el almacenamiento de registros del sistema.
- **RuntimeMaxUse**, **RuntimeKeepFree** y **RuntimeMaxFileSize**: Son idénticos a los anteriores pero cuando se está usando un registro volatil. Es decir afecta al uso de la memoria *RAM*.

Rotación y sincronización de registro:

- **MaxFileSec**: Definir un tamaño para realizar una rotación del archivo de registro
- **MaxRetentionSec**: Tiempo máximo de almacenamiento para archivos de registros antiguos
- **SyncIntervalSec**: Tiempo de espera antes de sincronizar los archivos de registro con el disco. Por defecto 5 min y solo se aplicará a las prioridades *debug*, *info*, *notice*, *warning* y *err*

Compatibilidad con syslog y wall:

- Si tenemos activado *syslog* y *wall* podremos redirigir los mensajes a estos demonios con opciones como **ForwardToSyslog** (reenvía a *syslog*), **ForwardToKMsg**, **ForwardToConsole** (reenvía a la consola principal) y **ForwardToWall** (reenvía a la consola de todos los usuarios logueados en el sistema), y además controlar los mensajes por nivel con **MaxLevelStore**, **MaxLevelSyslog**, **MaxLevelKMsg**, **MaxLevelConsole**, **MaxLevelWall**.

Nota: La compatibilidad con *syslog* se proporciona a través del socket */run/systemd/journal/syslog* por donde pasan todos los mensajes. Tendremos que asociar *syslog* a este socket y no a */var/log*.

Visualizando los logs

Cuando usamos un sistema de registro como *syslog*, *rsyslog* o *syslog-ng*, acostumbrábamos a visualizar el contenido de los registros con herramientas como *cat*, *head*, *tail*, *more* o *less*, esto es porque el contenido es guardado en texto plano. Esto ha cambiado con *journal*. Como comentábamos anteriormente *journal* guarda sus registro en formato comprimido por lo que necesitaremos usar la herramienta que el mismo sistema de registro nos proporciona, llamada **journalctl**. Para invocar *jounalctl* deberemos de contar en la mayoría de los casos con un usuario con privilegios, ya sea perteneciendo a un grupo específico del sistema o con permisos a través de *sudo*.

Para visualizar todos los registros del sistema ordenados por fecha bastará con invocar a *journalctl* sin parámetro alguno. *Journalctl* cuenta con un sistema de filtrado por campos que nos permitirá ajustar mucho mejor la salida para dar caza a registros concretos. Para acceder a estos 'campos' utilizaremos algunas de las siguientes opciones.

- Mostrar los mensajes del arranque actual:

```
journalctl -b
journalctl -b 0
```

- Mostrar los mensajes del arranque anterior o incluso del anterior al anterior (y sucesivos):

```
journalctl -b -1
journalctl -b -2
...
```

- Mostrar los mensajes de error de un arranque anterior:

```
journalctl -b -p err
```

- Mostrar los mensajes de un binario concreto:

```
journalctl /usr/bin/<binario>
```

- Mostrar los mensajes de un proceso concreto:

```
journalctl _PID=<proceso>
```

- Mostrar todos los mensajes para un servicio concreto:

```
journalctl _SYSTEMD_UNIT=httpd.service
```

- Mostrar todos los mensajes de unidades que contengan un patrón específico:

```
journalctl -u <patrón>
```

- Ver el búfer circular del kernel:

```
journalctl _TRANSPORT=kernel
```

- Usar operadores AND (espacio) y OR (+) para filtrar mensajes:

```
journalctl _SYSTEMD_UNIT=httpd.service + -u http _PID=3412
```

- Mostrar registros para un dispositivo específico:

```
journalctl /dev/sdb
```

- Podremos especificar un intervalo de tiempo con los parámetros **-since** y **-until**. El intervalo puede ser especificado de manera explicita para días como '*yesterday*', '*today*' o '*tomorrow*', o indicando la fecha mediante el formato "*YYYY-MM-DD hh:mm:ss*"

Ejecutar tareas automáticamente en el futuro

Con *Linux* podremos crear tareas para que sean ejecutadas en un futuro. Para ello contamos con la utilidad **cron**, estrechamente relacionada con **at**. La diferencia entre ambas es que *cron* nos permite ejecutar tareas de forma periódicas mientras *at* es utilizada para ejecutar un comando en un momento dado de forma exclusiva. Las tareas *cron* relacionadas con la administración del sistema suelen ejecutarse en horarios en los que la mayoría de ordenadores clientes o de escritorio suelen estar apagado. Esto se programa así para evitar una sobrecarga en el sistema a horas en los que quizás el usuario este trabajando. En servidores donde el apagado total del sistema es infrecuente este mecanismo funciona de forma efectiva, pero para solucionar el problema con los ordenadores de escritorio se implementó **anacron**, el cual se asegura de que las tareas de sistema programás por *cron* sean ejecutadas eficientemente cuando el ordenador del cliente es encendido, más concretamente cuando el demonio *anacron* es ejecutado.

Cron

El demonio *cron* parece que está '*dormido*' en nuestro sistema y que solo es '*despertado*' en un momento concreto especificado en algunos de sus archivos de configuración para ejecutar una tarea concreta y luego volver a su estado de '*descanso*'. Pero lo que en realidad hace *cron* es comprobar cada minuto los archivos de los directorios */var/spool/cron* y */etc/cron.d/* además del archivo */etc/crontab*, ejecutando los comandos especificados en estos archivos, siempre y cuando la hora de comprobación coincida con alguna de las que posee las tareas configuradas.

Existen dos tipos de tareas *cron*: las **tareas cron del sistema** y **tareas del usuario**. Las tareas *cron* del sistema se ejecutan como *root* y por lo general llevan a cabo tareas de mantenimiento del sistema, como por ejemplo eliminar antiguos archivos temporales del directorio */tmp*, rotación de archivos de registro, etc. En cambio las tareas *cron* del usuario ejecutan ciertos programás que el usuario quiere ejecutar de forma regular. Es posible crear tareas de usuario que sean ejecutadas por *root*, por ejemplo si fuese necesario ejecutar ciertos comandos en momentos no contemplados por las tareas del sistema.

Advertencia: Las tareas *cron* se ejecutan de forma automática y sin supervisión, por lo que no se recomienda ejecutar programás o scripts interactivos.

Formato para la definición de tareas cron

Existen diferentes formás de crear una tarea *cron*, pero por lo general el formato de la tarea tendrá la siguiente sintaxis.

```
min  hora  diadelmes  mes  diadelasemana  [usuario]  /ruta/al/script
```

- **Minutos**: 0 – 59
- **Horas**: 0 – 23
- **Día del mes**: 1 – 31
- **Mes**: 1 – 12 o *Ene*, *Feb*, *Mar*, etc.
- **Día de la semana**: 0 – 7 (tanto el 0 como el 7 pertenecen al Domingo, por lo que podríamos concretar con 1 – 7) o *Lun*, *Mar*, *Mie*, etc.
- **Usuario**: Las tareas cron de usuarios no incluyen la especificación del nombre de usuario en esta línea. No obstante suele ser corriente encontrarlo en tareas cron del sistema, donde el encargado de dicha ejecución es root.
- **Programa**: Es el binario o script que vamos a ejectar el cual realizará las funciones pertinentes. Puede contener varios comandos o cadenas de ellos acompañado de directorios como en el caso de tareas cron de sistema.

Una vez que conocemos la sintaxis para definir una tarea, será necesario conocer algunos 'trucos' para especificar rangos o momentos concretos, por ejemplo:

- Un asterisco '*****' coincide con todos los valores posibles.

```
30 * * * 1 root /usr/sbin/<programa>
```

Esta tarea se ejecutará todos los días Lunes de todos los meses del año en el minuto 30 de cada hora.

- Un asterisco '*****' y una barra inclinada '/' precediendo a un dato indica un valor escalonado

```
*/30 * * * 1 root /usr/sbin/<programa>
```

Esta tarea se ejecutará cada 30 minutos. La diferencia con la anterior es que esta se ejecutará 2 veces en una hora, mientras que la anterior solo cuando sea el minuto 30′ de cualquier hora.

- Una lista separada por comás (0**,**6**,**12**,**18) coincide con cualquiera de los valores especificados

```
30 0,4,8,12,16,20 * * 1 root /usr/sbin/<script>
```

Ejecutará en el minuto 30′ de las 12, 4 y 8 AM y 12,16,20 PM la tarea especificada.

- Dos valores separados por un guión '**–**' indican un rango en el que los valores de los extremos son incluidos.

```
15 16-22 * * * /home/nebul4ck/bin/script
```

Se ejecutará la tarea del usuario *nebul4ck* los minutos 15′ de las horas 16,17,18,29,20,21 y 22

Crear tareas cron del sistema

Una vez que conocemos los archivos de configuración de *cron* y el formato para definir una tarea, vamos a aprender a programar una tarea *cron* del sistema.

Lo primero es saber que el archivo **/etc/crontab** es el encargado de las tareas del sistema, suele comenzar con algunas líneas donde se definen variables como *$PATH* y *$MAILTO* y a continuación una serie de líneas que invocarán determinadas tareas. La mayoría de las distribuciones incluyen tareas *cron* del sistema que se ejecutan cada *hora (hourly)*, *día(daily)*, *semana(weekly)* y *mes(monthly)*. La forma en la que *Linux* ejecuta estas tareas se simplifica enormemente con el uso de comandos como **run-parts**, **cronloop** o una utilidad similar y unos directorios en los que se encuentran los script a ejecutar. Lo que hacen estos comandos es ordenar que se ejecuten todos los scripts que se encuentren dentro del directorio especificado, de esta manera no tendremos que crear una línea por cada script/tarea a ejecutar.

```
17 * * * * root cd / && run-parts --report /etc/cron.hourly
```

Esta línea llegada las *17:00* se desplazará como *root* al directorio *raíz* y desde allí ejecutará el comando **run-parts** el cual a su vez ejecutará todos los scripts que se encuentren dentro del directorio */etc/cron.hourly*

Entonces si sabemos que todos los script, por ejemplo bajo el directorio */etc/cron.daily* o */etc/cron.d/daily* van a ser ejecutados diariamente, podremos crear nuestro propio script y copiarlo dentro de este directorio, así nos aseguraremos***** que *root* ejecutará la tarea de forma diaria.

(*****) En realidad seguro no hay nada, pues dependiendo de la hora de ejecución puede que la tarea se ejecute o no, ya que muchos sistemás, sobre todos los de escritorio suelen apagarse por las noches, así que quizás una tarea programada para ejecutarse diariamente a las *5:00 AM* nunca llegue a ejecutarse si el ordenador a esta hora siempre esta apagado. Para solucionar esto se implemento **Anacron**, una utilidad que veremos en la próxima sección.

Advertencia: El propietario de los directorios que almacenen tareas *cron* del sistema debe ser *root* y solo este debe de tener permisos para escribir en dichos directorios.

Importante: Antes de enviar un script como tarea *cron* se recomienda encarecidamente que sea probado, de lo contrario puede que verse ante un sistema no funcional tras la ejecución de este.

Crear tareas cron de usuario

En la sección anterior hemos aprendido a crear tareas *cron* del sistema y para ello hemos hecho uso del archivo */etc/crontab*. Tenemos que dejar claro que cuando nos refiramos al archivo de configuración de tareas *cron* del sistema deberemos de indicar la ruta completa ya que existe un programa con el mismo nombre '***crontab***' con el que podremos mantener los archivos de configuración de tareas *cron* del usuario y además también nos podemos referir como *crontab* a los archivos de configuración de tareas *cron* de los usuarios, para el que deberemos entonces de nombrar la ruta completa (al igual que con el archivo de configuración de tareas *cron* del sistema) cuyo *path* será alguno de estos: */var/spool/cron/*, */var/spool/cron/crontabs* o */var/spool/crontabs* y recibirá el nombre del usuario.

Dicho esto vamos a crear una tarea *cron* de usuario, que ya sabemos entonces que deberemos de utilizar la herramienta *crontab* cuya sintaxis es la siguiente:

```
crontab [-u usuario] [-l | -e | -r] [archivo]
```

Si no proporcionamos el parámetro **-u** <*usuario*> modificaremos el archivo de configuración de tareas de usuario del usuario desde el que es invocado el comando. De lo contrario si queremos modificar el archivo *cron* de otro usuario, algo que deberemos de hacer como *root*, tendremos que especificar la opción *-u*. Si queremos listar el contenido del archivo *cron* del usuario empleamos **-l**, **-r** elimina y **-e** edita el archivo. Para editar se empleará un editor de texto predeterminado que podremos modificar cambiando el valor de la variable *$VISUAL* o *$EDITOR*.

Advertencia: Si creamos una tarea con '***$ crontab -e***' tendremos que tener en cuenta que los archivos *cron* de usuario serán creado automáticamente bajo los directorios */var/spool/cron*, */var/spool/crontabs* o */var/spool/cron/tabs* con el nombre del usuario y que se recomienda no volver a modificar a mano si no con el propio comando *crontab*.

La forma anterior es una manera rápida de editar el archivo *cron* del usuario creando así tareas *cron*. Pero también podremos crear nosotros mismos un archivo en nuestro directorio home (por ejemplo) con el mismos formato que tiene el archivo */etc/crontab*, es decir un comando precedido de los cinco campos de dígitos o comodines para indicar el momento de ejecución de la tarea y si es necesario, las primeras líneas del archivo en las que se definían ciertas variables de entorno. Una vez creado el archivo se lo pasaremos como argumento al comando *crontab*.

```
$ crontab -u <usuario> mi-archivo-cron
```

Nota: Recordar que si estamos instalando una tarea para nuestro propio usuario con el que además estamos logueado en el sistema, no será necesario el parámetro *-u*

Restringir el acceso a cron

El acceso al recurso *cron* se puede restringir de varias maneras:

- **Permisos de ejecución**: Podremos modificar los permisos de ejecución para *cron* y *crontab*, de manera que solo puedan ser ejecutados por usuarios pertenecientes a un determinado grupo como por ejemplo *cron*.
- **Lista de usuarios permitidos**: El archivo */etc/cron.allow* contiene una lista de los usuarios a los que se permite el acceso a *cron*. Si este archivo está presente, sólo podrán utilizar *cron* aquellos usuarios cuyos nombres figuren en el archivo.
- **Lista de usuarios rechazados**: El archivo */etc/cron.deny* contiene una lista de usuarios que tienen denegado el acceso a cron.

Anacron

En sistemás que se apagan con frecuencia, como portátiles o PC personales, es frecuente que las tareas *cron* del sistema no se ejecuten nunca, sobre todo aquellas que son ejecutadas a altas horas de la madrugada. Un comportamiento como este (no ejecutarse dichas tareas) podría derivar en problemás graves como el llenado de una partición a causa del no-borrado de archivos bajo */tmp*, saturar los archivos de registro, u otros problemás. Por ello se implementó el uso de **anacron**, concebido como suplemento de *cron*, con el fin de asegurar que las tareas de mantenimiento regular se ejecuten a intervalos razonables. De hecho *anacron* se utiliza en algunas distribuciones para ejecutar tareas *cron*, por lo que el efecto es tomar el control de las obligaciones de *cron******.

(*****) *Anacron* mide sus intervalos de tiempo en días por lo que no servirá para ejecutar tareas *cron* que se ejecuten con intervalos de horas. Por consiguiente no deberíamos de eliminar las tareas *cron* de sistema.

Anacron guarda un registro de los programás que se deberían de ejecutar y la frecuencia en días con la que deberían de hacerlo. Cuando se ejecuta, *anacron* comprueba en su configuración (*/etc/anacrontab*) que programás deben de ejecutarse y determina si ha transcurrido un periodo superior al intervalo de ejecución. Si el periodo ha sido superado, *anacron* ejecutará el programa. De este modo se podrán reconfigurar las tareas *cron* normales de su sistema como tareas *anacron* con la seguridad de que se ejecutarán incluso en sistemás que se apagan con regularidad durante largos periodos de tiempo.

Configuración de anacron

El archivo de configuración de *anacron* consta de tres tipos de líneas: líneas de comentarios (comienzan con #), asignaciones de variables de entorno (*SHELL=/bin/bash*) y líneas de definición de tareas, las cuales están formadas por cuatro campos:

```
periodo retardo identificador comando
```

- El **periodo** es la frecuencia en días con la que se debe de ejecutar el comando.
- El **retardo** es el periodo en minutos que transcurre entre el inicio de *anacron* y el momento en que se ejecuta el comando. Esta función está pensada para evitar que el sistema de sobrecargue si *anacron* tiene pendiente de ejecutar muchas tareas a su inicio.
- El **identificador** identifica al comando.
- El **comando** es el programa a ejecutar seguido opcionalmente de los parámetros que pudiera recibir.

Nota: La variable *PATH* es particularmente importante si algún script invoca programás sin especificar sus rutas completas.

Un archivo anacron podría tener el siguiente aspecto:

```
# /etc/anacrontab: configuration file for anacron
# See anacron(8) and anacrontab(5) for details.
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
HOME=/root
LOGNAME=root
# These replace cron's entries
1 5 cron.daily run-parts --report /etc/cron.daily
7 10 cron.weekly run-parts --report /etc/cron.weekly
@monthly 15 cron.monthly run-parts --report /etc/cron.monthly
```

Invocar a anacron

Para que todo lo anterior funcione obviamente hay que invocar a *anacron* que suele hacerse de dos maneras:

- **Mediante un script de inicio**: Puede crear un sencillo script de inicio *SysV* que ejecute *anacron* desde su modo de ejecución normal o colocar la llamada a *anacron* en un script de inicio local como */etc/rc.d/rc.local* (Fedora y Red Hat) o */etc/boot.d/boot.local* (Suse). Esto es preferible en sistemás que se apagan e inician con frecuencia, como portátiles o sistemás de escritorio.
- **Mediante una tarea cron en /etc/crontab** que remplace a las entradas normales de las tareas *cron* de su sistema. Esto puede presentar los mismos problemás que *cron*, es decir, que el PC este apagado cuando sea el momento de ejecutar *anacron*, pero podremos sino invocar a *anacron* más de una vez al día, por ejemplo, una vez cada seis horas asegurándonos que se ejecutará durante una típica jornada diaria de ocho horas.

Importante: Independientemente de como ejecute *anacron*, asegúrese de desactivar de *cron* cualquier tarea que sea controlada ahora por *anacron*, de lo contrario esta tarea será ejecutada dos veces en el caso de que el PC se encuentre encendido cuando *cron* ejecute sus tareas.

At

El programa **at** nos permite ejecutar una sola vez un único comando en un momento futuro determinado, útil cuando el uso de *cron* o *anacron* suponen un despliegue excesivo. El comando *at* depende del demonio *atd* por lo que tendremos que asegurarnos de que *atd* está ejecutándose en segundo plano. Dependiendo del sistema de inicialización que tengamos podremos ejecutarlo de una u otra manera, pero seguramente dependa de un script de inicio que tendremos que ajustar o iniciar con nuestro *runlevel* o *target* por defecto.

En su uso normal *at* recibe como única opción una hora, aunque como casi siempre este comportamiento es configurable mediante otras opciones. La hora puede adoptar varias formás:

- **Una hora del día** pasada con el formato *HH:MM* donde además podemos añadirle *AM* o *PM* para utilizar el formato de 12 horas. Si la hora marcada ya ha pasado, el comando se ejecutará al día siguiente.
- Las siguientes palabras pueden ser utilizada para ejecutar un comando a las 12 PM '***noon***', a las 00:00 '***midnight***', a las 16:00 o 04:00 PM '***teatime***', el día de hoy '***today***' o el día de mañana '***tomorrow***'
- **Un determinado día**: Si especificamos un día querrá decir que estamos planificando una tarea para ser ejecutada con más de 24 horas de antelación. Deberemos de añadir el día tras la hora tal que así; '*hh:mm MMDDYY*', '*hh:mm MM/DD/YY*' o 'hh:mm *DD.MM.YY*'
- Concretar **un futuro específico**: Por ejemplo, queremos ejecutar una tarea dentro de 3 horas, independientemente de la hora actual. Usaremos la palabra *now* (ahora) y le sumamos las horas que deberán de transcurrir: '***now + 3***'. Igualmente podemos indicar que a partir de ahora (now) espere por ejemplo hasta el próximo Sábado y reinicie '***at now sat reboot***'

Cuando ejecutemos el comando *at* con su hora (no importa el formato utilizado), el programa responderá con su propio *prompt*

```
at >
```

Podremos utilizar el *prompt* de *at* como si de *bash* se tratara, es decir escribiremos los comandos que queremos ejecutar, con la diferencia que no serán ejecutados en el momento si no cuando sea la hora especificada. Una vez hayamos terminado de escribir comandos finalizaremos con la combinación de teclas **Control+D**. Podremos realizar la misma función pero pasando un archivo con los comandos en vez de escribirlos en el *prompt*. Un ejemplo de esto podría ser:

```
at -f comandos teatime
```

Algunos programás que ofrecen soporte a *at* son:

- **atq**: Enumera las tareas pendientes. Es igual que ***at -l***
- **atrm**: Elimina una tarea de la cola. Es igual que ***at -d***
- **batch**: Interesante en equipos que suelen tener indices de carga elevados. *batch* funciona de modo muy similar a *at*, pero solo ejecuta las tareas cuando el *nivel de carga del sistema baja de* ***0,8***. Igual a ***at -b***

Restringir el acceso a at

Podemos implementar restricciones en el uso de *at* al igual que hacíamos con *cron*. Los archivos */etc/at.allow* y */etc/at.deny* funcional de manera análoga a los archivos de restricciones de *cron*. Si ninguno de los 2 archivos existe solo *root* podrá ejecutar *at*. La lista de usuarios permitido concederá derecho de ejecución de *at* a aquellos usuarios cuyo nombre aparezca en la lista y de la misma forma serán denegados aquellos que estén en la lista de usuarios denegados (*/etc/at.deny*)

LPIC-1 Capítulo 8

EN ESTE CAPÍTULO SE ABARCAN LOS SIGUIENTES OBJETIVOS DEL EXAMEN:

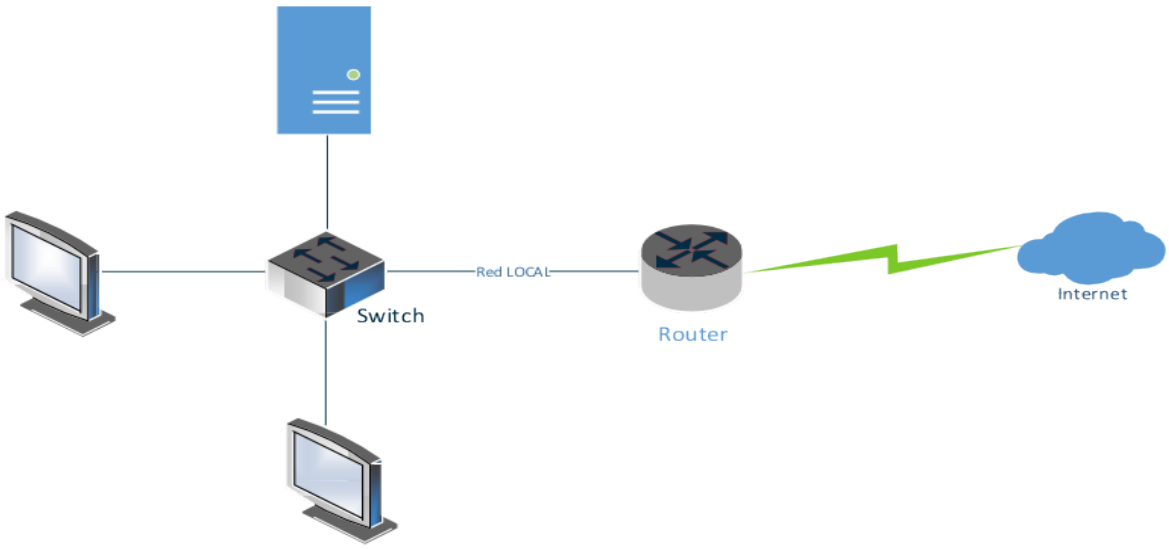
- 109.1: Fundamentos de los protocolos de Internet (4)
- 109.2: Configuración básica de redes (4)
- 109.3: Resolución de problemás básicos de red (4)
- 109.4: Configuración DNS en el lado del cliente (2)

Configuración básica de redes TCP/IP

La mayoría de los sistemás *Linux* están conectados a una red, ya sea como máquinas clientes, servidores o ambos a la vez. Es por ello necesario saber como configurar las herramientas básicas de red, de manera que sea posible mantener comunicaciones con el exterior.

Una red a parte de estar formada por máquinas clientes y servidoras, consta de una serie de dispositivos que deben estar interconectados como son los *routers*, *switch*, *hubs*, *modems*, *hardware*, *radioenlaces* y/o *telefonía IP*, además de *servicios* y *programás*.

Un esquema de red básico donde se usan *PCs* clientes, servidor, *switch* o *hub* y *router* con conexión al exterior, podría ser el siguiente:



Nota: En la red de la imagen se ha utilizado un *switch* y no *hub*. Los *hub* redirigen todo el tráfico entre todos los ordenadores, mientras que los *switch* son lo bastante

inteligentes como para enviar paquetes solo al destino pertinente, además los *hub* sólo permiten una transmisión *semidúplex* (como la usada por los *walkie-talkies*, mientras uno envía mensaje el otro extremo solo escucha), por el contrario, los *switch* usan conexiones *dúplex*, en la que ambas partes pueden enviar información al mismo tiempo. Estas características hacen superiores a los *switch* frente a los *hubs.*

Para que todos estos dispositivos, servicios y programás puedan mantener '*conversaciones*' inteligibles y fluidas por todos ellos, debe de existir una serie de '*normás*' y estándares de construcción y comunicación. Serán los protocolos las reglas que utilice la red para hacer posible las comunicaciones.

Actualmente, el estańdar de la industria en redes es un conjunto o pila de protocolos denominado **TCP/IP**
(*Transmission Control Protocol/Internet Procotol, Protocolo de control de Transmisión/Protocolo de Internet*) formado por una serie de capas o niveles que resuelven tareas relacionadas con la transmisión de datos y proporcionan un servicio bien definido a los niveles más altos. Los niveles más altos son los más cercanos al usuario y tratan con datos más abstractos, dejando a los niveles más bajos la labor de traducir los datos de forma que sean físicamente manipulables.

Supongamos la transferencia de datos de un ordenador a otro, un archivo por ejemplo. Lo primero será disponer de una aplicación cliente con la que enviar el archivo, esta aplicación puede encontrarse en la capa superior de la pila de protocolos. Las capas consecutivas harán entre otras cosas que el archivo sea dividido en bloques de datos conocidos como paquetes a los que se le añadirá cierta información y serán encapsulados y enviados por separado permitiendo por ejemplo que si un paquete se pierde o corrompe se pueda reenviar solo el paquete específico en vez del archivo completo. El sistema receptor deberá de desemcapsular estos paquetes para recomponer el archivo de origen.

Nota: Es algo normal que en las transferencias se retrasen o pierdan paquetes, es por ello que muchos protocolos de red incluyen procedimientos para la detección de errores.

Existen varios tipos de paquetes e incluso algunos se pueden almacenar dentro de otros. *Ethernet* incluye su propio tipo de paquete conocido como *frame*, por lo que aquellos paquetes generados por protocolos de red que se ejecuten sobre *Ethernet* se almacenarán dentro de estos *frames* para ser transportados.

En las próximás secciones estudiaremos la pila *TCP/IP* de una forma más detallada. De momento será indispensable que conozcamos el hardware de red así como sus funciones básicas.

El hardware de red

El hardware de red está diseñado para permitir que dos o más ordenadores se comuniquen entre sí, es por ello que en cada punto de la comunicación será necesario un dispositivo que permita la entrada y salida de mensajes. La mayoría de ordenadores cuentan con interfaces de red integradas en su placa base, tarjetas de red internas (*PCI* o *PCIe*) o externas como las *USB*.

Linux admite varios tipos habituales de hardware de red, entre los que destacan las tarjetas *Ethernet* y las utilizadas para las conexiones inalámbricas (*Wi-Fi*).

Ethernet

Las conexiones vía **Ethernet** se realizan mediante cableado de par trenzado (lo que evita las interferencias) llamado *UTP*. Las conexiones *Ethernet* se presentan en varias velocidades de transmisión siendo las más comunes en redes locales las **10Base-T** o **100Base-T** (*10 Mbits/s* o *Mbps* y *100Mbps*) para las que se suelen utilizar el cableado *UTP* de categoría 5 o 5e (*UTP Cat 5* o *5e*). Otros estándares emergente de gran velocidad son los de **1000Base-T** (*1.000Mbps* o conocido como *Gigabit Ethernet*) y **10 gigabits**, para los que se suele utilizar cableado *UTP Cat 5e* o *cables ópticos*.

Wi-Fi

Para las conexiones **Wi-Fi** se utilizan los protocolos inalámbricos siendo los más habituales:

- **802.11b** cuya velocidad para la cual está diseñado es de *11Mbps*
- **802.11a** y **802.11g** que trabajan a una velocidad de *54 Mbps* teórica
- **802.11n** el cual es el más rápido con una velocidad de hasta *300Mbps*

A excepción del protocolo *802.11a* raramente utilizado, el resto son compatibles entre ellos, utilizándose siempre el más lento de entre los detectados en la red.

Advertencia: Si utiliza redes inalámbricas, debemos de asegurarnos de usar encriptación sobre la clave de acceso a la red mediante los protocolos **WPA** (*Wi-Fi Protected Access*, *Acceso protegido a red inalámbrica*) o preferiblemente **WAP2**. La encriptación **WEP** (*Wired Equivalent Privacy*) es más débil y superada con facilidad.

Otros tipos de hardware de red

- **Token Ring**: Anteriormente usado en redes IBM
- **LocalTalk**: Diseñado para la comunicación entre los primeros Macintosh

- **FDDI** (*Fiber Distributed Data Interface, Interfaz de datos distribuidos por fibra*), **HIPPI** (*High-Performance Parallel Interface, Interfaz en paralelo de alto rendimiento*) y **Fibre Channel**: Interfaces de alta velocidad utilizadas en aplicaciones de alto rendimiento, por ejemplo, para interconectar edificios que están a muchos metros de distancia e incluso Kilómetros como pueden ser las centrales de telefonía y armarios de subrepartición.

Pilas de protocolos de red

Una pila de protocolo es un conjunto de programás que convierten y encapsulan datos entre capas de abstracción. La pila puede ir desde una capa superior que por ejemplo trabaja con un programa cliente, como podría ser *Filezilla* (un programa para transferencias mediante el protocolo *FTP*) hasta capas más inferiores donde podríamos encontrar el controlador de la propia tarjeta de red.

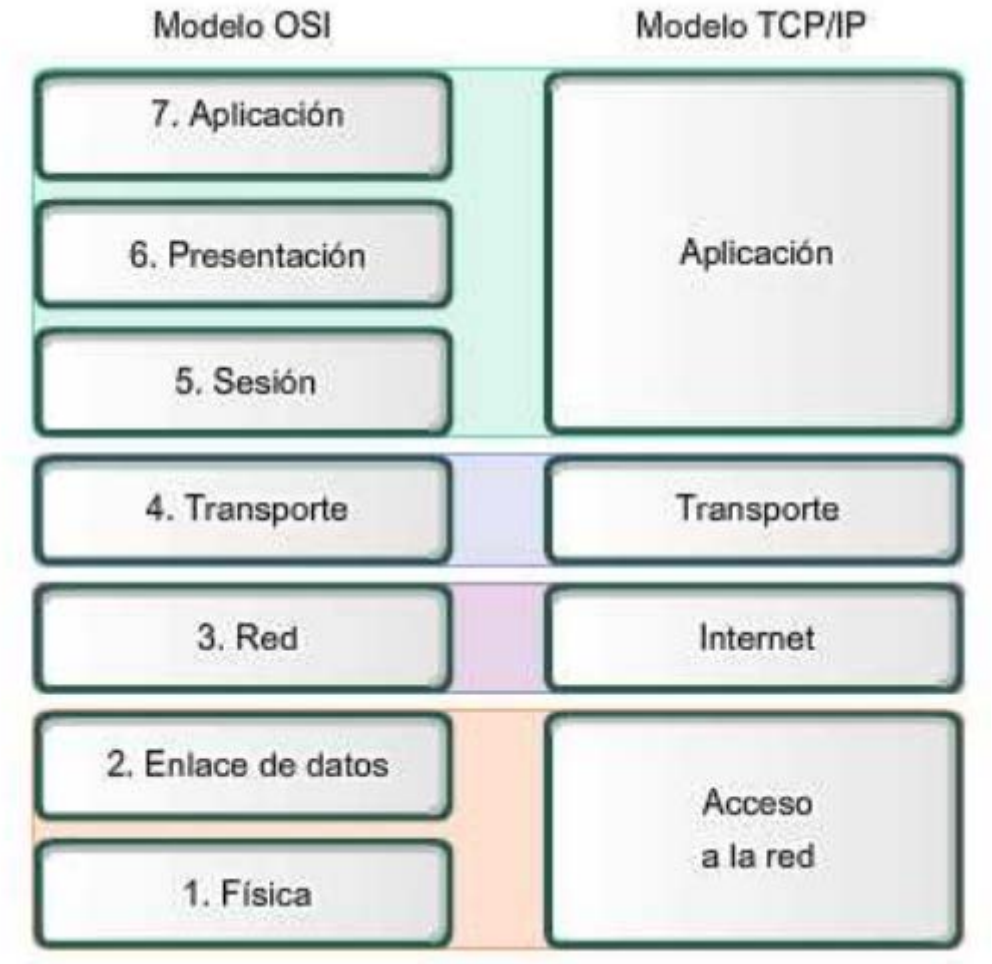
Para que existan transacciones entre diferentes dispositivos se precisa de una pila de protocolos compatibles en cada uno de ellos, de manera que un dispositivo receptor pueda trabajar con los paquetes recibidos de manera homóloga pero a la inversa de como lo hizo el emisor.

Las pilas de protocolos se suelen representar gráficamente mediante diagramás.

La pila TCP/IP

La pila de *TCP/IP* es un conjunto de protocolos de red en los que se basa Internet permitiendo la comunicación y transferencia de datos entre ordenadores instalado en distintos puntos.

Podemos describir por analogía la pila *TCP/IP* con el modelo **OSI** (*Open System Interconnection*) propuesto como una aproximación teórica y como una primera fase en la evolución de la redes de ordenadores. El modelo *OSI* es más fácil de entender pero el que se utiliza es *TCP/IP* sucesor de **NCP** (*Network Control Program*). El modelo *OSI* esta formado por un total de *siete* capas mientras que *TCP/IP* 'absorbe' alguna de ellas en varias capas reduciendo el número a *cuatro*, es por ello que *OSI* se entiende mejor. La siguiente imagen muestra ambas pilas de protocolos:



A continuación vamos a resumir muy brevemente lo que realiza cada capa de la pila *TCP/IP*

1. La capa de *Acceso a la red* controla los dispositivos del hardware y los medios que forman la red, por ejemplo la tarjeta de red del usuario, asegurando que existe conexión y está OK
2. La siguiente capa "*Internet*" determina la mejor ruta a seguir a través de la red.
3. La capa de transporte admite las comunicaciones entre distintos dispositivos de distintas redes
4. Representa los datos para el usuario, además del control de codificación, compresión y diálogo entre usuarios

Se le denomina *TCP/IP* en referencia a dos de los más importantes protocolos que la componen y que además fueron los primeros en definirse: **TCP** (*Transmission Control Protocol, Protocolo de control de Transmisión) y* **IP** (*Internet Procotol/Protocolo de Internet*). Algunos otros protocolos que formar esta pila son (clasificados por capa dentro de la pila):

Capa 4 – Aplicación:

- **DNS** (*Domain Name Service*): Utilizado para resolver nombres de direcciones *IP* en Internet
- **HTTP** (*Hypertext Transfer Protocol*): Se usa para transferir y acceder a las páginas web
- **SMTP** (*Simple Mail Transfer Protocol*): Se utiliza para la transferencia de mensajes de correo y adjuntos
- **FTP** (*File Transfer Protocol*): transferencia interactiva de archivos entre sistemás
- **Telnet**: Un protocolo de emulación de terminal, utilizado para proporcionar acceso remoto a servidores y dispositivos de red

Capa 3 – Transporte:

- **UDP** (*User Datagram Protocol*): Es protocolo más sencillo de la pila *TCP/IP* pero rápido. No proporciona sofisticados procedimientos para corregir los paquetes desordenados, garantizar el envío ni, en general, mejorar las limitaciones de *IP*. Entre los protocolos de la capa de Aplicación que se construyen sobre *UDP* se incluyen: *DNS*, *NFS* y muchos otros protocolos de medios en *streaming*.
- **TCP** (*Transmission Control Protocol*): Seguramente el protocolo de transporte más ampliamente utilizado en la pila *TCP/IP*. *TCP* crea conexiones completas con comprobación de errores y otras funciones que simplifican la creación de protocolos de red que deben de intercambiar grandes cantidades de datos. Por ello *TCP* aplica una pequeña penalización de rendimiento. La mayoría de los protocolos de la capa de Aplicación se construyen bajo *TCP* y destacan algunos como: **SMTP**, **HTTP** y **FTP**.

Capa 2 – Internet:

- **IP** (*Internet Protocol*): Protocolo central de la pila *TCP/IP*. Proporciona un método de mejora para transferir paquetes entre ordenadores. Esto no garantiza que los paquetes alcancen su destino. Entre otras cosas más *IP* es el protocolo es encargado de asociar direcciones *IP*
- **ICMP** (*Internet Control Message Protocol*): Sencillo protocolo para la comunicación de datos. Se suele utilizar para enviar mensajes de error entre ordenadores, algo que se suele hacer modificando un paquete y devolviéndoselo al emisor.

Otras pilas de protocolos

Otras pilas de protocolos para la transferencia de datos entre ordenadores pueden ser:

- **NetBEUI**: Pila de protocolos original de *Microsoft* e *IBM* para *Windows*. En desuso
- **AppleTalk**: Pila inicial de *Apple*
- **IPX/SPX** (*Internet Packet Exchange/Sequence Packet Exchange, Intercambio de paquetes entre redes/Intercambio de paquetes en secuencia*): Pila de protocolos preferida por *Novell*

Nota: Cada capa de las que se compone el sistema emisor equivale a una capa del sistema receptor, pero no tienen porque ser absolutamente idénticas, por ejemplo, puede tener distintos modelos de tarjetas de red o, incluso, puede utilizar tipos de hardware de red totalmente diferentes como Ethernet y Wi-Fi. Los ordenadores pueden ejecutar distintos OS y por tanto utilizar pilas de protocolos diferentes (pero lógicamente equivalentes).

IPv6. Mejoras sobre IPv4

- Mientras *IPv4* alcanza un máximo teórico de "*cuatro mil millones*" de direcciones *IP*, que debido al mal reparto de estas son algunas menos, *IPv6* eleva esta cifra a "*seiscientos setenta mil billones*" (670 mil billones) de direcciones *IP* por cada milímetro cuadrado de la superficie de la tierra.
- **Multicast**, habilidad para enviar un único paquete a múltiples destinos. Forma parte de la especificación base de *IPv6* y no como en *IPv4*, donde es opcional aunque usualmente implementado. *IPv6* no implementa *broadcast* (envió de un paquete a todos los nodos conectados). El *multicast IPv6* comparte protocolos y características comunes con *IPv4*, pero también incorpora cambios y mejoras.
- **SLAAC** (*Stateless Address Autoconfiguration*): La primera vez que un dispositivo es conectado a una red, este envía una solicitud de router (*RS: router rolicitation*) usando *multicast* pidiendo los parámetros de configuración. Si los router están configurados para responder a esta petición, enviarán un *RA: router advertisement* con los datos de configuración. Si una aplicación no soporta *SLAAC* usará entonces *DHCP*.
- **IPSec** (*Internet Protocol Security*): Protocolo para cifrado y autenticación de *IP*. Obligatorio en *IPv6 ya que viene como base del protocolo, aunque actualmente no se está utilizando.*
- IPv6 ofrece mejoras en el rendimiento relativo al procesamiento de datos a través de los router.

Direcciones de red

Para que los dispositivos puedan comunicarse necesitarán algún medio al que hacer referencia a la hora de transmitir los paquetes, este medio no es otro que una dirección y

un puerto. Las direcciones pueden adoptar formás diferentes dependiendo del tipo de hardware de red, pila de protocolos, etc… e igualmente podrán necesitar un puerto, usado para dirigir un programa específico a un ordenador remoto.

Para conectar ordenadores dentro de redes en las que pueden estar usándose hardware de bajo nivel diferentes, como pueden ser *Ethernet*, *Token Ring*, *Wi-Fi*, etc… se necesitarán diferentes direcciones, de entre las que deberemos de conocer al menos las direcciones de hardware de red, las direcciones IP numéricas y los nombres de hosts de tipo texto.

Direcciones de hardware

El hardware de red dedicado como las tarjetas de *Ethernet* llevan programadas direcciones de red únicas conocidas como direcciones **MAC** (*Media Access Control, Control de acceso a medios*). Tienen una longitud de seis bytes y se expresan como números hexadecimales separados por dos puntos. Podemos informarnos sobre las direcciones de hardware para las tarjetas *Ethernet* instaladas en un sistema utilizando el comando **ifconfig**.

Algunas utilidades y el hardware de red de bajo nivel utilizan las direcciones de hardware para dirigir paquetes de datos. Por ejemplo, el *switch* detecta que una dirección en particular está conectada a un cable concreto, y en consecuencia, envía los datos dirigidos a dicha dirección sólo a través del cable asociado. Otro ejemplo es el uso que hace de estas direcciones el servicio *DHCP* asignando direcciones *IP* fijas a direcciones de hardware específicas. Existen herramientas de diagnostico avanzado que permiten examinar paquetes que proceden de o están dirigidos a direcciones de hardware específicas. En la mayoría a de los casos, no tendrá que preocuparse de las direcciones de hardware de un ordenador.

Direcciones numéricas IPv4/IPv6

Acabamos de ver algunos motivos del porque sobre la existencia de direcciones *MAC* o direcciones de hardware. Estas direcciones no guardan relación matemática alguna con las direcciones *IP* numéricas de *IPv4* y *IPv6* pero si que existen protocolos de la pila *TCP/IP* como **ARP** (*Address Resolution Protocol, Protocolo de resolución de direcciones*) o **NDP** (*Neighbor Discovery Protocol, Protocolo de detección de vecinos*) que realizan conversiones entre direcciones *IPv4 – MAC*, y *IPv6 – MAC*, con el fin por ejemplo de permitir a un ordenador que envíe un paquete *Broadcast* (mensaje que se envía a todos los ordenadores de la red) en cuya respuesta se incluirá la dirección hardware del receptor del paquete y así la pila *TCP/IP* puede dirigir el tráfico para una *IP* dada a la dirección hardware obtenida.

Las direcciones *IPv4* se suelen expresar con cuatro números en base decimal, donde cada uno puede llegar al máximo de 255 (*192.168.10.255*) separados por puntos o bien, mediante notación "***dotted quad***". *dotted quad* viene a ser la representación de una IP mediante una secuencia de *32 bits* separados por puntos (*dots*) en grupos de cuarto (*quad*) bytes cada uno. Por ejemplo podríamos representar la IP de máscara de red *255.255.128.0* de la siguiente forma:

```
255.255.128.0 = 11111111.11111111.10000000.00000000
```

Las direcciones *IPv6* constan de ocho grupos de números hexadecimales de cuatro dígitos separados por dos puntos y tienen un tamaño de 16 bytes (*128 bits*). Si uno o varios grupos de cuatro dígitos son 0000, éste o estos se pueden omitir, indicándolo con "**: :**". Una IPv4 podría ser:

```
fed1:odb8:85a3:08d3:1319:8a2e:3070:7334
```

La máscara de red como ya adelantábamos (también llamada máscara de *subred* o *netmásk*) es un número que define las 2 partes de las que está formada una *IP*, la parte de dirección de red y la de dirección de host. Empleando el ejemplo de netmásk podríamos apreciar ambas partes de la siguiente manera, donde con azul representamos la dirección de red y con rojo el número de hosts que podemos anidar en esa red. Evidentemente para esto necesitamos una dirección *IP* de host, pero esto se explicará más detalladamente en las siguientes secciones:

```
11111111.11111111.10000000.00000000
```

Nota: Las direcciones *TCP/IP* se combinan con una máscara de red para aislar la dirección de red.

Además de las formás con las que hemos representado anteriormente una máscara de red (bien en decimal o bien en binario) podremos identificar la *netmásk* de una red con una serie de sufijos colocados en la dirección de red. Igualmente, usando la *netmásk* del ejemplo anterior (*255.255.128.0*) y la dirección de red *192.168.10.0*, podremos entonces definir la red como:

```
192.168.10.0/17
```

Nota: Si contamos el número de bits 1 que forman la máscara de red en binario, deduciremos que el */17* representa entonces a la *netmásk* para esa red. Además esta porción de la *netmásk* (la del grupo de 1, define la porción de red, algo que veremos a continuación).

La máscara de red de *IPv6* funcionan igual que las de *IPv4* con la diferencia de que los números son más largos y que en *IPv6* se tiende a emplear la notación hexadecimal.

Clases de redes

Dentro de *IPv4* nos encontramos con bloques de direcciones que definen redes, estas se han dividido tradicionalmente en varias clases, cada una de ellas destinadas a un tamaño de red, de manera que podamos estructurar las subredes y dispositivos conectados de una forma más eficiente, al menos antes de la llegada del estándar *CIDR* (veremos este estándar a continuación). En la actualidad sirven principalmente como medio para definir máscaras de red por defecto (sufijos /8, /16 y /24). Las clases de direcciones *IPv4* son las siguientes:

Clase	IP Inicial	IP Final	Redes	Host
A	1.0.0.1	126.255.255.255	126	2^24=16777214
B	128.0.0.1	191.255.255.255	16384	2^8 - 2^16=de 256 a 6
C	192.0.0.1	223.255.255.255	2.097.152	2^1 - 2^8=de 2 a 256
D	224.0.0.1	239.255.255.255		
E	240.0.0.1	255.255.255.255		

Las clases A, B y C son para uso general de la red. Normalmente utilizaremos una clase de red y su rango de direcciones privadas reservado en función del tamaño que le vayamos a dar a la red. Dependerá si es para una corporación, gran empresa o campus universitario (clases A y B) o una pequeña o mediana empresa e incluso las redes de uso doméstico (clase C). Los rangos de direcciones privadas reservados en cada clase son los siguientes:

- **Clase A**: 10.0.0.0 – 10.255.255.255
- **Clase B**: 172.16.0.0 – 172.31.255.255
- **Clase C**: 192.168.0.0 – 192.168.255.255

Estos rangos de *IP* reservadas se conocen como direcciones **RFC1918** ya que es el documento que las define. Se usan para que cada dispositivo dentro de una red privada (*LAN*) pueda tener su propia dirección. No sirven para conectar directamente con el exterior. Para conectar con el exterior u otras redes privadas, se dispondrá de una puerta de enlace (normalmente un dispositivo de traducción de dirección de red o *NAT*, o un servidor *proxy*).

El cometido de **NAT** (*Network Address Translation*) es 'ocultar' una serie de dispositivos de una red tras un único sistema o *IP*. Por ejemplo, si contamos con una red privada de *clase C 192.168.1.x* con 5 ordenadores, *NAT* tendrá que traducir las conexiones de estos 5 ordenadores para que puedan tanto enviar como recibir datos a

través de una sola *IP* pública ofrecida por nuestro **ISP** (*Internet Service Provider, Proveedor de Servicios de Internet*), mediante la cual conectamos a Internet.

Las **clases D** y **E** son especiales. La primera (*clase D*) es utilizada para el envío simultáneo de datos a varios ordenadores (*multidifusión),* mientras que la *clase E* es utilizada para propósitos experimentales unicamente.

Importante: Las direcciones *IP* deben ser asignadas a ordenadores individuales para evitar que dos sistemás utilicen la misma dirección en Internet o red local.

Direcciones IPv4 especiales

- La dirección *IP* 0.0.0.0 es usada como red **default**.
- Las direcciones *IP 127.x.y.z* en concreto *127.0.0.1* son utilizadas como direcciones **loopback**, es decir el ordenador la usa para enviarse información a él mismo.
- Para el direccionamiento **Broadcast** (enviar los datos una sola vez desde un emisor a todos los destinos posibles) se usa la dirección *255.255.255.255*, dirección que el protocolo *IP* tiene limitado a uso local ya que de lo contrario saturaríamos Internet. Podremos hacer un *Broadcast* directo e igualmente limitado localmente (en términos de red) a un grupo de ordenadores concretos, o mejor dicho a una subred específica, utilizando el prefijo de dirección de red y la porción de host que identifica la conexión *Broadcast* de dicha red (algo que veremos en la próxima sección cuando hablemos del *subnetting*), por ejemplo, para la red con sufijo *192.168.1* y con una máscara de red *255.255.255.0* podríamos usar la dirección *192.168.1.255* para realizar una transmisión *Broadcast* en esa sola red.
- Una dirección **multicast** esta asociada con un grupo de receptores interesados. El emisor envía un único *datagrama* desde su dirección *IP* hacía la dirección *multicast*, cuyo rango es de la *IP 224.0.0.0* a la *239.255.255.255* (*clase D*) definido por la *RFC 3171*, y el *router* será el encargado de entregar este mensaje a aquellos emisores que hayan informado sobre su interés en el contenido.

El estándar CIDR y la técnica VLSM

El estándar **CIDR** (*Classless Inter-Domain Routing, Direccionamiento entre dominios sin clases*) aplicado empleando la técnica **VLSM** (*Variable Length Subnet másk, Máscara de Subred de Longitud Variable*), se implantó con el fin de aportar más flexibilidad y, en resumidas cuentas poder crear máscaras y definición de redes que no dependieran de una clase como las anteriormente estudiadas.

Originalmente las *IP* se dividían en las clases *A,B,C,D* y *E,* lo que daba lugar entre otras cosas a que los *routers* utilizaran un encaminamiento con clases (*classfull routing*). Las clases eran divididas según su máscara de red la cual además se definía a partir de bloques enteros de unos y ceros binarios dando lugar a poder utilizar los sufijos naturales *8, 16* y *24* para definir una red, por ejemplo:

Una red con *IP 192.168.1.0* y una *netmásk 255.255.255.0* puede ser definida mediante *192.168.1.0/**24*** donde el 24 indica el número de bits 1 que componen la porción de red dentro de la máscara (*255.255.255.0 = 11111111.11111111.11111111.00000000*)

Lo que se pretendió con *CIDR* fue permitir una mayor flexibilidad al dividir rangos de direcciones *IP* en redes separadas lo que aportó un uso más eficiente de las cada vez más escasas direcciones *IPv4* y disminuyó la sobrecarga de los *routers* principales de Internet para realizar el encaminamiento, utilizando ahora un encaminamiento sin clases (*classless routing*).

Información: El término *VLSM* se usa generalmente cuando se habla de redes privadas, mientras que *CIDR* se usa cuando se habla de Internet.

Por ejemplo, vamos a utilizar una máscara de red de longitud variable y no una con una clase predefinida como la del ejemplo anterior. Esto puede ser útil si queremos crear una red en la que solo puedan anidarse 6 dispositivos, por poner un ejemplo: Un *router*, un PC cliente, un Servidor y un dispositivo móvil (aun tendremos dos *IP* libres). Para lograr esto, tendremos que adaptar la máscara de red:

La *IP* de la red seguirá siendo *192.168.1.0* pero en cambio la máscara pasará a ser *255.255.255.248* (11111111.11111111.11111111.11111000) o *192.168.1.0/**29***. Para conocer el número de host basta con ver la porción de host de la *netmásk* (aquella que solo contiene ceros) que está en base dos por lo que en nuestro caso será *2^3-2=6*

Nota: Restamos 2 debido a que no podemos usar la primera *IP* la cual es para la red y ni la última que se utiliza para *Broadcast*. Si hubiésemos elegido una máscara más elevada como por ejemplo */30* solo podríamos anidar 2 hosts, así que nos hubiésemos quedado cortos.

Subnetting

Las direcciones *IP* están divididas en dos bloques, definidos por su máscara de red. El primer bloque define la dirección de la red a la que esta pertenece y es representada con valores 1 binarios en su máscara de red, mientras que la segunda parte, porción de host, identifica a un host específico dentro de la red y es determinada en función de los valores 0 binarios de la máscara de red, esta porción determina el número máximo de host que podremos anidar a la red.

Vamos a seguir un ejemplo a modo de explicación. Supongamos que tenemos la siguiente *IP* y *netmásk.*

Nota: Hemos añadido una segunda *netmásk* (*B*) de tipo CIDR para duplicar el ejemplo y afianzar los conocimientos.

- Dirección IP: *172.30.9.102*
- Máscara de red: *255.255.0.0*
- Una máscara de red *CIDR* (B): *255.255.255.224*

Importante: Las máscara de red pueden representarse igualmente de la siguiente manera:

- Dirección IP y netmásk: 172.30.9.102/**16**
- Dirección IP y netmásk *CIDR* (B): 172.30.9.102/**27**

Lo primero que deberemos de hacer será pasar a binario estos números.

- Dirección IP: *10101100.00011110.00001001.01100110*
- Máscara de red: *11111111.11111111.00000000.00000000*
- Máscara de red CIDR (B): *11111111.11111111.11111111.11100000*

Nota: La parte Azul representa la porción de red mientras que la parte roja indica la porción de host.

De momento se ve claro el porque del **/16** o **/27**, solo hay que contar los números 1 de las máscara de red anteriores. Ahora para conocer la IP de red y el número máximo de host deberemos de igualar de la siguiente manera estos números binarios:

11111111.11111111.00000000.00000000 -> Netmásk: 255.255.0.0
011010101100.00011110.~~00001001.01100110~~ -> Red: 172.30.0.0

El número de hosts será de **2^16-2** , pues existen 16 bits con valor 0 que indican la porción de host menos dos que se usan: una para la dirección de red (la primera) y otra para *broadcast* (la última). Un total de 65534 dispositivos podrán ser anidados a la red *172.30.0.0*

El ejemplo B sería algo más complicado pero captaremos pronto el concepto. Quedaría algo así:

11111111.11111111.11111111.11100000 -> Netmásk: 255.255.255.224
10101100.00011110.00001001.011~~00110~~ -> Red: 172.30.9.96

El número de hosts en este caso será de **2^5-2**. Esto define una red más reducida, útil si queremos mantener el control de redes pequeñas para evitar así el acceso a un gran número de dispositivos para los que realmente no necesitamos acceso.

Ahora que ya sabemos realizar *subnetting* sobre una red, esto es, crear redes en función del número de hosts que tengamos pensado anidar a una red, jugando con los bits de la porción de red y porción de host y sabemos determinar el valor *IP* de nuestra red a partir de una *IP IPv4* y *netmásk*, vamos a ver como identificar la *IP* de *multidifusión* de nuestra red.

Para identificar IP *broadcast* de nuestra red tendremos que igualar la IP binaria con la *netmásk* binaria y cambiar los 0 binario por 1 en la *IP* en función de los 0 existentes en la *netmásk*, tal que así.

- Para el primero de los ejemplos teníamos una máscara de red con clase *255.255.0.0* que definía una *IP* de red 10101100.00011110.00000000.00000000 (172.30.0.0). Bastará con convertir los dos últimos octetos a 1 binario dándole así su máximo valor (.255.255), resultando: **172.30.255.255**
- Para el ejemplo (B), es decir IP de tipo *CIDR* haremos lo mismo pero con matices, ya que no solo existen 0. En este caso modificaremos los bits de la dirección IP que coincidan con los bits 0 de la *netmásk* cambiándolos de 0 a 1 dejando intactos los que ya tienen como valor 1:

Netmásk: 11111111.11111111.11111111.11100000
Dirección IP: 10101100.00011110.00001001.011**00**11**0**

Broadcast: 0101100.00011110.00001001.011**11111**, resultando: **172.30.9.127**

Conversión binaria-decimal: 01111111 = 127

Interés: Existe una web (http://www.aprendaredes.com/cgi-bin/ipcalc/ipcalc_cgi) donde podemos calcular de forma automática todo estos valores, algo que nos ayudará a la hora de encontrar conclusiones o corroborar nuestros resultados.

Resolución de nombres de hosts

Los nombres de host se componen de dos partes: nombre del equipo y el nombre del dominio, separados por un punto '**.**'. El primer campo hace referencia al nombre del propio ordenador mientras que el segundo a una colección de ordenadores. Por ejemplo vamos a suponer que tenemos una red local doméstica o empresarial que recibe el nombre de *4sysadmins* y que nuestro equipo de trabajo y un servidor web que gestionamos y que se encuentra dentro de la red de la oficina se llaman *nebul4ck* y *freeser*, respectivamente. El nombre de host de estos equipos serán *nebul4ck.4sysadmins* para nuestro equipo de trabajo y *freeser.4sysadmins* para el servidor.

TCP/IP ofrece la posibilidad de vincular nombres de hosts con direcciones *IP* de varias maneras. Antes de aprender a relacionarlos, vamos a conocer un poco más sobre los dominios.

Nota: Un conflicto de nombres podría impedir contactar con un sistema, a parte de causar probablemente otros problemás, como redireccionamientos de correo, registro en formularios webs, etc…

Los dominios se estructuran jerárquicamente. Para los dominios de Internet de nivel superior o **TLD** (*Top-Level Domain*) encontramos los *.com*, *.edu*, *.org*, etc.. que normalmente corresponden a países o grandes corporaciones. Dentro de cada *TLD* hay dominios que identifican a otras organizaciones más específicas como empresas, negocios y en definitiva a casi cualquier cosa para la que se haya registrado dicho dominio. Por ejemplo para un banco (*mibanca.es*) o para un usuario anónimo que quiere tener su propio dominio (*jesuslafuentemedina.net*). Estos dominios aún pueden dividirse en subdominios, un ejemplo de ello puede ser la dirección de este blog *nebul4ck.wordpress.com*, donde la plataforma *wordpress.com* presta un subdominio a cada usuario en ella registrada. Esta estructura puede continuar dividiéndose aunque no es lo habitual.

Cada uno de estos dominios y subdominios incluyen a ordenadores físicos o virtuales específicos.

Información: Podemos registrar nuestro propio dominio en Internet desde 11€ al año.

Ahora que conocemos algo más sobre los dominios y la estructura que estos siguen podemos aprender algunas técnicas de resolución de nombres, esto es, convertir un nombre en una dirección *IP* o viceversa.

Cuando accedemos por ejemplo a *google.es* no escribimos la dirección *IP* del servidor con el que está enlazado el nombre, si no directamente el nombre (*google.es*). Esto es posible gracias a los **DNS** (*Domain Name Server*) que no es más que una base de datos distribuida la cual realiza las conversiones oportunas entre direcciones *IP* y nombres de hosts.

Para que nuestro equipo o dominio pueda trabajar de esta manera es necesario incluir como mínimo un servidor *DNS*. Esto puede hacerse a nivel de configuración de *router* o del propio dominio o host. Lo ideal es contar con al menos dos servidores *DNS* que puedan resolver los nombres de Internet o si no, que se deleguen entre ellos esta tarea en caso de que uno deje de estar operativo o no pueda encontrar la relación IP-nombre. El proceso de resolución de nombres es transparente a nosotros ya que hay empresas encargadas de ofrecer servidores *DNS* para que una vez configurados en nuestra red hagan el trabajo de preguntar a otros *DNS* y así sucesivamente. No obstante podemos crear nuestros propios servidores *DNS* dentro de nuestra red doméstica o empresa, para que albergue en su base de datos distribuida todos los nombres y direcciones *IP* que queramos que sean conocidas dentro de nuestra red o dominio, de manera que entre los

ordenadores del dominio exista la comunicación vía nombre de host y no necesariamente por IP. Esto no entra en los límites del examen pero es bueno saber que esta es una forma de trabajar con nombres y no con direcciones *IP* dentro de nuestro dominio o red.

Existen otros métodos para trabajar con nombres dentro de nuestro dominio (que abarca tanto a equipos en local como en remoto, siempre y cuando pertenezcan a nuestra red) que no necesariamente implica un despliegue como el de los servidores *DNS*. Por ejemplo a través del archivo */etc/hosts* podremos crear asociaciones entre direcciones *IP* y nombres. Supongamos una red local cuyo dominio es *4sysadmins* y en la que existen varios equipos y dispositivos (*móviles*, *impresoras*, *servidores*, *portátiles*, etc..). Tenemos un ordenador de trabajo desde el que normalmente accedemos al resto de dispositivos (*LinMinCinn*), ya sea por *ssh*, *ftp*, *samba*, etc.. y estamos cansados de tener que memorizar las *IP*. Podremos configurar el archivo */etc/hosts* de la siguiente manera:

```
127.0.0.1    localhost
127.0.1.1    LiMinCinn.4sysadmins LiMinCinn

# IPS de la red

10.5.50.12    hprinter.4sysadmins hprinter
10.5.50.21    xiaomi.4sysadmins xiaomi
10.5.50.30    zopo.4sysadmins zopo
10.5.50.39    tab.4sysadmins tab
10.5.50.48    freeser.4sysadmins freeser
```

Nota: La segunda línea del archivo quizás no la encontréis en vuestros sistemás, es un bug de Debian que asigna esta IP al host al que no se le ha especificado una IP fija. Nada preocupante. Como sabemos está dentro del rango de IP de *loopback*.

Con un archivo de configuración como este podremos acceder desde el equipo *LinMinCinn* a cualquiera de los dispositivos listados mediante su nombre. La estructura es básica:

```
ip        nombre.dominio        nombre
```

La ventaja de resolver nombres mediante el archivo */etc/hosts* es que es de fácil configuración, pero como desventaja, al ser un archivo local solo afecta al equipo en el que se ha configurado.

Antes de emplear el servicio *DNS*, *Linux* busca en el archivo */etc/hosts*, comportamiento que se puede modificar desde la configuración del archivo */etc/nsswitch.conf*. Este archivo configura el servicio **NSS** (*Network Switching Subsystem*, *Subsistema de conmutación de red*). Para modificar el comportamiento de orden de búsqueda de relaciones IP-nombre deberemos de modificar la línea '*hosts:*'. Por defecto la línea hace que prevalezcan los archivos (*files*) sobre el servicio *DNS* (*dns*)

```
hosts:        files dns
```

Linux además del archivo */etc/hosts* incluye un archivo llamado */etc/networks* que funciona de un modo similar pero que aplica solo a direcciones de red. Además invierte la línea de formato, que pasan a ser:

```
nombre        ip
```

Otros tipos de configuraciones existentes para resolución de nombres pueden ser los del servicio **LDAP** (*Lightweight Directory Access Protocol, Protocolo ligero de acceso a directorios*) o utilizando invocaciones NetBIOS de *Windows*.

Herramientas para buscar manualmente información DNS

A veces, necesitaremos buscar la información proporcionada por *DNS* de manera manual. Existen varios programás para realizar esta tarea:

- **nslookup**: Realiza búsquedas *DNS*, por defecto en ordenadores concretos y devuelve los resultados. Este programa está oficialmente obsoleto.
- **host**: Sustituto para las funciones más sencillas de *nslookup*. Carece de un modo interactivo. En el caso más simple podemos escribir host nombre.dominio o la dirección *IP* que deseamos consultar.
- **dig**: Realiza búsquedas *DNS* más complejas que host. Podemos utilizar para localizar la dirección *IP* de un único nombre de *host* o viceversa. Es más flexible que host.
- **whois**: Puede buscar la información de un dominio en conjunto. Su uso pasa por invocar el nombre del comando seguido del nombre del dominio (*4sysadmins.com*, por ejemplo) y mostrará información como el propietario del dominio, a nombre de quién se ha contratado, en que empresa de dominio se ha registrado, personas de contacto, etc… Muchos dominios no ofrecen esta información

Puertos de red

Cuando un cliente envía una petición a un servicio concreto de un servidor debe de especificar un número de puerto, pues no solo bastará con la dirección *IP* del servidor. Una analogía con el mundo '*real*' sería el tener que entregar un paquete a una vivienda concreta dentro de un edificio. Si tan solo tienes el número de edificio (lo que podría ser el *servidor*) pero no número de puerta (*puerto*) de la vivienda dentro de ese edificio no podrás entregar el paquete al destinatario.

Es por esto por lo que se agrega un número (*número de puerto*) a la dirección *IP*, de manera que el sistema pueda redirigir el tráfico a un programa/servicio específico dentro del servidor. Cuando iniciamos un servicio dentro de un servidor, por ejemplo un

servicio **FTP** (*File Transfer Protocol*) por convención el servidor asignará un número concreto de puerto (el 21 para *FTP*) para este servicio. El archivo de configuración */etc/services* es el archivo que asocia a cada servicio un número de puerto. Esto viene definido por convención, por lo que para un funcionamiento básico no será necesario modificar estas relaciones, a menos que usemos un servicio poco usual y queramos crear una relación entre el servicio y un número de puerto por el que redirigir los paquetes para ese servicio en concreto.

Si tenemos un servidor en el que corren diferentes servicios, como por ejemplo un servidor web apache, un servidor de correos y un servidor ssh (*Secure Shell*), cuando el cliente envíe un paquete a uno de estos servicios, el sistema asignará el puerto de entrada al destino en función de la petición realizada. Si lo que se ha enviado a sido un paquete de solicitud para mostrar una página web, esto irá dirigido por el puerto 80 del servidor.

Los clientes no poseen números fijos de puertos de salida. Esto se asigna de forma dinámica lo que permite que en un mismo ordenador se puedan ejecutar diferentes instancias de un mismo cliente. De este modo no competirán entre ellos para usar un único puerto de salida.

Nota: Un cliente es un programa que inicia una conexión de red para intercambiar datos mientras que un servidor es quien escucha dichas conexiones y responde a éstas. En una conexión web el cliente podría ser *Firefox* y el servidor *Apache.*

Números de puertos, uso y servicios típicos de Linux

Los números de puerto son características de los protocolos *UDP* y *TCP*. Algunos protocolos como *ICMP* no emplean números de puerto.

Una diferencia clave de los puertos *TCP/IP* es la existente entre puertos con y sin privilegios. Los puertos con privilegios indican que son usados por servicios configurados por el administrador (*root*) y usan números por debajo del 1.024.

A continuación una tabla con los puertos más comunes utilizados por los principales servicios:

Puerto	Protocolo	Uso	Servidores Linux
20	TCP	Datos FTP	ProFTPd, vsftpd
21	TCP	FTP	ProFTPd, vsftpd
22	TCP	SSH	OpenSSH, Dropbear
23	TCP	Telnet	in.telnetd
25	TCP	SMTP	Sendmail, Postfix, Exim, qmail
53	TCP y UDP	DNS	BIND (*Berkeley Internet Name Domain, Dominios de Nombres de Internet*) también conocido como named, dnsmasq.
67	UDP	DHCP	ISC (*Internet Software Consortium, Consorcio de software de Internet*), DHCP (*dhcpd*), dnsmasq
80	TCP	HTTP	Apache, Nginx, Roxen
110	TCP	POP-3	Dovecot
111	TCP y UDP	Asignador de puertos	NFS, NIS y otros servicios basados en RPC
113	TCP	Auth/ident	identd
119	TCP	NNTP (*Network News Transfer Protocol, Protocolo para la transferencia de noticias en red*)	InternetNews (*INN*), Diablo
123	UDP	NTP (*Network Time Protocol, Protocolo de hora de red*)	ntpd
139	TCP	NetBIOS Session (*archivos compartidos de Windows*)	Samba
143	TCP	IMAP (*Internet Mail Access Protocol, Protocolo de acceso a mensajes de Internet*)	Dovecot, Cyrus IMAP
161	UDP	SNMP (*Simple Network Management Protocol, Protocolo simple de administración de red*)	Net-SNMP
162	TCP	SNMP-trap	Usado por servicios que reportan incidencias asíncronas a través de la red
177	UDP	XDMCP	XDM, KDM, GDM, MDM
389	TCP	LDAP	OpenLDAP
443	TCP	HHTPS (*HTTP sobre SSL*)	Apache, Nginx, Roxen
445	TCP	DS (*Microsoft Directory Services, Servicios de directorio de Microsoft*)	Samba
465	TCP	Registrado para URD (*URL Rendezvous Directory*) aunque suele utilizarse para el envío seguro de correos (*SMTP sobre SSL*)	Sendmail, Postfix, Exim, qmail o router de red
513	TCP	Rlogin (*login remoto*)	rlogin, rdesktop
514	UDP	Syslog (*Usado para logs del sistema*)	syslogd, rsyslog
631	TCP	IPP (*Internet Printing Protocol, Protocolo de impresión de Internet*)	CUPS (*Common Unix Printing System, Sistema de impresión común de Unix*)
636	TCP	LDAP sobre SSL	OpenLDAP
993	TCP	IMAP sobre SSL	Dovecot, Cyrus IMAP
995	TCP	POP-3 sobre SSL	Dovecot
5900+	TCP	RFB (*Remote FrameBuffer*)	VNC (*Virtual Network Comp*

diferentes utilidades para comprobar incluso de manera remota que puertos tenemos abiertos en nuestro ordenador, cuales están siendo utilizados y porqué servicios, y además aquellos que se encuentran a la escucha esperando conexiones remotas o cerrados. Algunas de estas herramientas pueden ser *netstat*, *lsof*, *nmap* o *nessus*, de las que hablaremos en la sección "*Diagnóstico de red*"

Configurar la red en Linux

Durante la instalación de una distribución de *Linux*, lo más normal es que se pida que realicemos la configuración de la tarjeta de red de forma casi automática, con una serie de parámetros que nos permitan conectarnos a Internet o bien mantener comunicación con los dispositivos de nuestra red local. Normalmente basta con marcar que queremos recibir unos parámetros de forma automática vía DHCP. No obstante podremos introducir de forma manual y personalizada esos valores si indicamos que queremos conservar una configuración estática.

Los parámetros mínimos para que una red pueda tener comunicación con el resto de dispositivos son: la dirección *IP* del host, la máscara de red, la puerta de acceso y un servidor *DNS*. Veremos estos y otros parámetros en la sección de "*configuración de red con parámetros estáticos*"

Para aprender a configurar la red desde 0 vamos a suponer que no la hemos configurado ya durante la instalación y que además no vamos a usar las herramientas *GUI* que se nos ofrecen para tal fin. Dicho esto comencemos.

Instalando el hardware de red

Lo primero que deberemos de hacer en caso de que nuestro equipo no cuente con un hardware de red interno, será instalar una tarjeta de red en la placa base o vía USB. Una vez tengamos la tarjeta instalada y el sistema en funcionamiento, deberemos de asegurarnos de que el módulo de la tarjeta está cargado. Esta operación de carga normalmente la lleva a cabo el propio sistema mediante scripts de inicio, pero si nuestro sistema no ha detectado nuestro hardware, podremos cargar nosotros mimos el controlador. Para cargar el módulo de la tarjeta será necesario contar con dicho módulo y conocer su nombre. A continuación pasaremos a cargarlo con el comando **modprobe**:

```
# modprobe <módulo>
```

Con la tarjeta ya instalada y cargado so módulo e incluso configurado para ser cargado al inicio pasaremos a configurarla.

Nota: De compilar un *kernel* deberemos de asegurarnos que incluimos el módulo para la tarjeta de red.

Configurando el hardware de red

Podemos configurar la tarjeta de red para que acceda a Internet o se comunique con el resto de dispositivos de la red local de varias maneras. Si descartamos como ya dijimos anteriormente las herramientas *GUI*, podremos configurar la red vía *DHCP* o con parámetros estáticos a través de los archivos de configuración.

Configurar la red con DHCP

Es la forma más sencilla de configurar un ordenador para que utilice una red *TCP/IP*. Básicamente el funcionamiento con *DHCP* es el siguiente: En la red existirá un ordenador que ejecuta un servidor *DHCP* o bien un *router*, el cual tiene configurado los parámetros necesarios para que los ordenadores de la red puedan acceder a ella. Estos parámetros son normalmente una *IP* de red, un rango de *IP* dentro de la red, un servidor *DNS*, una máscara de red y una puerta de acceso. Es posible añadir otros parámetros como nombres de hosts o servidores *DNS* secundarios.

Todas las distribuciones Linux tienen como mínimo un cliente *DHCP* por defecto que inicia al arranque del sistema a través de su script de inicio o desde el archivo principal de configuración de la red. Cuando este arranca, el cliente *DHCP* envía una multidifusión en busca de un servidor *DHCP*. El servidor responde utilizando únicamente la dirección de hardware del cliente (es decir la *MAC* de la tarjeta de red) con la información de configuración que el cliente necesita para poder comunicarse con los demás ordenadores. De esta manera el cliente tendrá asignada una *IP* temporal (en caso de no ser renovada, esta puede pasar a utilizarse por otro dispositivo de la red) una puerta de enlace y aquellos parámetros configurados en el servidor o *router*.

Nota: En Linux hay tres clientes *DHCP* de uso habitual que son *pump*, *dhclient* y *dhcpcd*. Los tres pueden ser ejecutados además de por sus archivos de inicio o de configuración de red, como herramientas corrientes del sistema bajo el usuario *root* (*# dhclient eth0*). Por defecto en la gran mayoría de distribuciones el nombre de la interfaz es *eth0*, aunque esto puede cambiar

Ahora que conocemos el funcionamiento de *DHCP*, vamos a configurar nuestro ordenador para que haga uso de este servicio. Para esto solo tendremos que acudir al archivo de configuración de la interfaz de red y modificar una línea. Para sistemás *Red-Hat* y *Fedora* este archivo suele encontrarse en */etc/sysconfig/network-scripts/ifcfg-<nombre_interfaz>*, mientras que para sistemás *Debian* y derivados puede que el

archivo en cuestión sea */etc/network/interfaces* cuyo objetivos son similares pero los detalles quizás difieran.

Una vez localizado el archivo de configuración de interfaz en nuestro sistema deberemos de modificar la siguiente línea:

- Para Red-Hat y Fedora:

```
BOOTPROTO=dhcp
```

- Para Debian y derivados:

```
iface eth0 inet dhcp
```

Nota: Como ya avisamos antes, el nombre de la interfaz puede cambiar.

Configurar la red con valores estáticos

Si una red carece de servidor *DHCP* o simplemente queremos que nuestro ordenador que va a funcionar como un servidor web mantenga su *IP* de forma estática, deberemos de configurar los valores *IP* mediante su archivo de configuración, uno a uno.

- Archivo de configuración de *Red-Hat* y *Fedora* (otros como *CentOS*, derivado de *Red-Hat* usan el mismo formato y archivo):

```
$ sudo cat /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE="eth0"
BOOTPROTO="static"
IPADDR="192.168.1.32"
NETMÁSK="255.255.255.0"
NETWORK="192.168.1.0"
BROADCAST="192.168.1.255"
GATEWAY="192.168.1.1"
ONBOOT="yes"
```

Los parámetros de configuración son ya conocidos (si hemos seguido el estudio de secciones anteriores), pero vamos a destacar *DEVICE* el cual es el identificador del dispositivo, *BOOTPROTO* que como vemos a diferencia de la configuración anterior, en este caso es estática, la *GATEWAY* que además de configurarse en este archivo podrá ser configurada igualmente mediante el comando **route** o de forma permanente en otro archivo como */etc/sysconfig/network/routes* (*Red-Hat* y *Fedora*) y *ONBOOT* que al estar en '*yes*' lo que hará será arrancar la configuración al momento del inicio del sistema. Podemos preguntar que donde se encuentran configuradas las *IP* de los *DNS*, la respuesta es en el archivo */etc/resolv.conf*, el cual admite hasta tres servidores y son definidos uno por línea con el siguiente formato:

```
namserver <ip-servidor>
```

También podemos definir en este archivo el nombre de nuestro dominio local utilizando la línea:

```
domain <nombre-de-nuestro-dominio>
```

Nota: La *gateway* no es necesaria en sistemás que no se vayan a conectar a redes más grandes, es decir que sólo se encuentren trabajando en una red local que no contiene *router*.

- Archivo de configuración *Debian* y derivados:

```
$ sudo cat /etc/network/interfaces
iface eth0 inet static
address 192.168.1.23
netmásk 255.255.255.0
broadcast 192.168.1.255
gateway 192.168.1.1
hwaddress 81:ab:1c:59:aa:7b
up <comando>
```

Nota: Como curiosidad debemos mencionar la posibilidad de ejecutar comandos desde este archivo de configuración. En este ejemplo se a utilizando *up <comando>* para ejecutar un comando al '*levantar*' la tarjeta de red, pero existe la posibilidad de ejecutarlo al '*bajarla*' o antes y después de *levantarla*/*bajarla* con *down*, *pre-up*, *pre-down*, *post-up* y *post-down* respectivamente.

Importante: Es importante saber que si estamos utilizando herramientas *GUI* como *NetworkManager* para la configuración de nuestros hardware de red, nos encontremos que los archivos anteriormente mencionados están vacios o sus líneas comentadas. Esto se debe a que estas aplicaciones guardan sus configuraciones en otros archivos. Por ejemplo el archivo de configuración para *LinuxMint* (derivado de *Debian*) es */etc/NetworkManager/system-connections/<nombre-conexión>* y su contenido puede ser parecido al siguiente:

```
[802-3-ethernet]
mac-address=88:AE:1D:69:5A:5A

[connection]
id=wired-11
uuid=212274c7-08bc-4586-9h49-c1u218p9239f
type=802-3-ethernet
timestamp=1424866869

[ipv6]
method=auto

[ipv4]
method=manual
```

```
dns=8.8.8.8;8.8.4.4;
address1=192.168.1.50/24,192.168.1.1
```

Mostrar información sobre la configuración de la red

Con el comando *ifconfig* ya mencionado en secciones anteriores podremos mostrar la configuración de la red. Basta con escribir *$ ifconfig* y será mostrada la información de todas las interfaces configuradas. Si queremos mostrar solo una, deberemos de pasar su identificador como parámetro de *ifconfig*:

```
$ ifconfig eth0
```

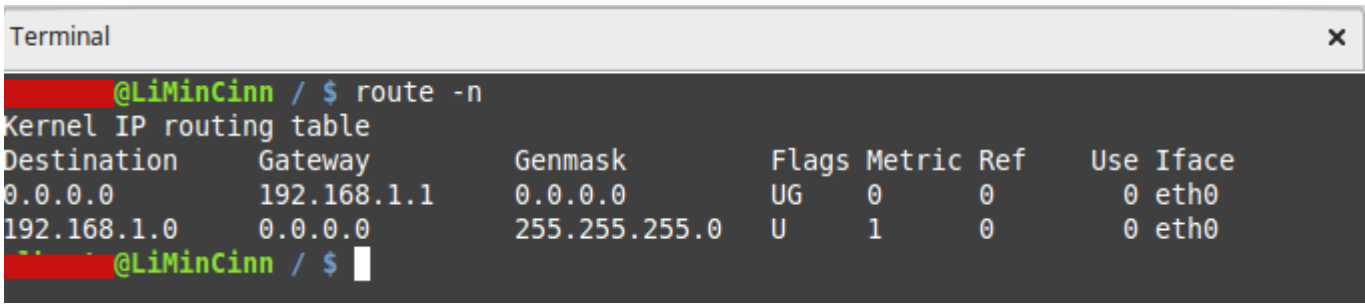
El comando *ifconfig* nos permite además configurar una interfaz de red directamente desde la línea de comandos, siempre y cuando se siga su sintaxis y especifiquemos en ella los valores pertinentes:

```
# ifconfig eth0 up 192.168.50 netmásk 255.255.255.0
```

Para añadir la puerta de acceso podemos usar *route*:

```
# route add default gw 192.168.1.1
```

Además, podemos utilizar *route* para efectuar un diagnóstico de la configuración de red, por ejemplo:



La primera línea indica que todo el tráfico saliente hacia Internet desde *eth0* saldrá por la puerta de acceso *192.168.1.1* mientras que la segunda línea indica que los datos destinados a cualquier ordenador dentro de nuestra red local pasan directamente por *eth0.*

Una vez configurada la tarjeta de red y con esto la conexión a Internet o red del equipo, podremos hacer uso del comando **ifup** para levantar la red. *ifup* es también útil para comprobar la configuración, ya que de no levantar la red podríamos sospechar de que algo anda mal y posteriormente usar *ifconfig* o *route* para depurar la configuración. De la misma manera podremos usar **ifdown** para cerrar las conexiones. Ambos comandos

se implementan como scripts que consultan los archivos de configuración y ejecutan entre bastidores comandos relevantes de bajo nivel.

Configurar el nombre de un equipo

Es importante dar un nombre a nuestro equipo, ya sea de forma local o dependiendo de servidores *DNS* de nuestra red. Podemos dar un nombre a nuestro equipo en la red para que sea reconocido por el resto de dispositivos de dos formás diferentes, bien a través de una entrada en la base de datos *DNS* o bien modificando los archivos */etc/hosts* de todos los dispositivos de la red. Los equipos con *Windows* también disponen de este archivo solo que evidentemente en otra ruta.

Si por el contrario queremos dar un nombre a nuestro equipo de forma local podremos usar la herramienta **hostname**, la cual sirve tanto para cambiar el nombre de forma temporal, es decir al reiniciar volveremos al nombre anterior o, mostrar el nombre actual del equipo. Para mostrar el nombre bastará con escribir el comando sin parámetro alguno. Para cambiar el nombre deberemos hacer lo siguiente:

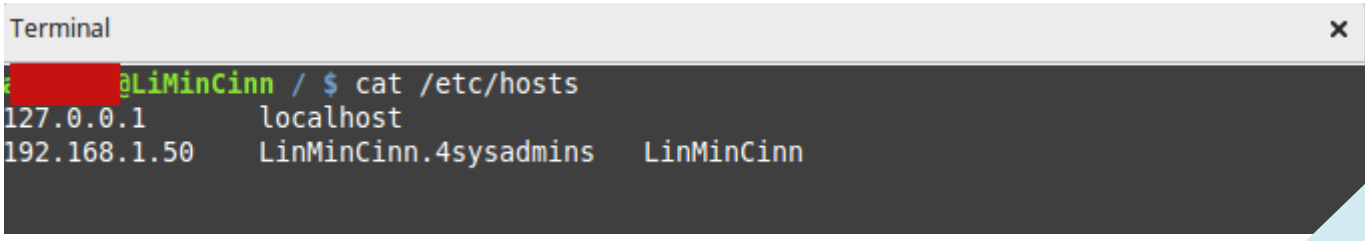
```
# hostname <equipo.dominio>
```

Si lo que queremos es nombrarlo de forma permanente deberemos de editar el archivo */etc/hostname* o */etc/HOSTNAME* (dependiendo de la distribución) y añadiendo una línea con el nombre del equipo y dominio.

Nota: Tanto en el uso de *hostname* como en la edición del archivo, el nombre del dominio es opcional.

Ya que nos hemos decidido a cambiar el nombre de forma permanente, una vez modificado el archivo */etc/hostname* o */etc/HOSTNAME* es recomendable cambiar los siguientes archivos:

- **/etc/hosts**: Añadir una línea con nuestra *IP* y nombre:



- **/etc/sysconfig/network**: En distribuciones como Fedora o CentOS puede ser interesante cambiar el nombre de cara a la red. Podremos modificar el valor de la variable HOSTNAME por nuestro nombre

Existen dos comandos con los que mostrar o cambiar el nombre del dominio del ordenador. Si queremos modificar el nombre del dominio del ordenador utilizado por **NIS** (*Network Information System, Sistema de información de red*) utilizaremos el comando **domainname**, mientras que con **dnsdomainname** definiremos el nombre del dominio utilizado por *DNS*.

Advertencia: Estos comando no afectan a los servidores remotos, solo al nombre dato a los programás que emplean llamadas a estos servidores.

Configuración del enrutamiento

Como regla general los encargados de redirigir el tráfico en una red son los *routers*. Estos *routers* disponen de una tabla de enrutamiento o tabla de reglas compleja en la que se definen las rutas a tomar por los paquetes. No obstante nuestro *Linux* dispone de una tabla de reglas básica como ya vimos anteriormente con el comando *route* por lo que podremos redirigir tráfico a determinadas redes. Antes de nada, debemos de saber que esto no es una opción que venga por defecto en el sistema si no que deberemos de editar un archivo para que Linux comience a aceptar paquetes que van a otra red y sea el quien los redirija a esta. Podremos hacer esto de la siguiente forma:

```
# echo "1" > /proc/sys/net/ipv4/ip_forward
```

Si queremos hacer este cambio permanente deberemos de localizar el archivo */etc/sysctl.conf* o */etc/sysconfig/sysctl* (dependiendo de la distribución esto puede cambiar). Una vez localizado lo editaremos con nuestro editor de texto favorito y cambiaremos el valor del parámetro '*net.ipv4.ip_forward = 0*' por 1.

Nota: Si tenemos problemás para localizar este archivo en nuestra distribución, recodemos que disponemos de *grep* para localizar cadena de texto. Podemos usar como patrón *ip_forward* o similar.

Ahora que ya tenemos el sistema configurado para redirigir tráfico, tendremos que utilizar una herramienta para configurar una nueva puerta de acceso y ruta. Para esto usaremos el comando *route*.

Por defecto nuestra red tiene su propia puerta de enlace, la cual es la dirección del *router* (normalmente *192.168.1.1*). Por esta puerta pasarán todos los paquetes proveniente de nuestra red con destino al exterior. Pero como queremos que ciertos paquetes destinados por ejemplo a la red *172.20.8.0/16* sean filtrado por otra puerta de acceso, configuraremos esta ruta tal que así:

```
# route add -net 172.20.8.0 netmásk 255.255.0.0 gw 172.20.9.1
```

Donde:

- **-net 172.20.8.0**: Indica el destino. En este caso una red (-net) con IP. Podríamos haber usado -host si el destino fuese un equipo concreto
- **172.20.90.1**: Es el *router* (puerta de acceso) por el que deberán de pasar los paquetes.

El archivo /etc/sysctl.conf y el comando sysctl

Este archivo es especialmente importante y aunque no se abarca en los propósitos del *LPIC-1* no está de más que sepamos ahora que lo hemos modificado para hacer el reenvío de paquetes, que función es la que desempeña y como poder cambiar los parámetros que se encuentran definidos en él. El archivo */etc/sysctl.conf* es usado para pasarle parámetros de configuración al *kernel* en tiempo de ejecución, es decir, se utiliza para configurar o cambiar el comportamiento del *kernel* que se está ejecutando. Equivale a modificar los valores de los archivos del sistema de archivos virtual */proc* y más concretamente a los del directorio */proc/sys* solo que de hacerlo a través de los archivos, al reiniciar estos cambios no serán conservados, mientras que si modificamos los parámetros directamente desde */etc/sysctl.sys* los cambios permanecerán. Para cambiar los valores de este archivo una vez que el sistema se encuentra arrancado usaremos el comando **sysctl**, el cual sirve tanto para visualizar parámetros como para modificarlos. Los parámetros a modificar se encuentran como ya hemos comentado en */proc/sys* y los más significativos suelen ser:

- **dev** : Establece los parámetros de configuración de los dispositivos conectados.
- **fs** : Parámetros relacionados con los sistemás de ficheros como los *inodes*, *quotas*, etc..
- **kernel** : Comportamiento general del kernel
- **net** : Los relacionados con la configuración de la red.
- **vm** : Utilizados para configurar la memoria virtual

La sintaxis de *sysctl* es sencilla, como casi todo comando este acepta una serie de opciones y se le pasa la variable/parámetro a modificar junto con su valor:

sysctl [opciones] variable=valor

Alguna de las opciones más habituales para *sysctl* son: **-a** (muestra todos los parámetros y sus valores asignados que se encuentran configurados en el sistema), **-w** (establece un valor indicado, no será permanente pero nos ayudará a comprobar sus resultados. En caso de ser favorables podremos hacer posteriormente los cambios permanentes modificando *sysctl.conf*) y **-p** (carga los valores de *sysctl.conf*)

El comando IP

Para terminar esta larga sección sobre "*Configuración del hardware de red*" vamos a comentar sobre el comando **ip**. El comando *ip* es el comando que pretende sustituir a

ifconfig y *route*, y por supuesto a *ifup* y *ifdown*, vamos que es la navaja suiza con respecto a la configuración de la red. A día de hoy seguimos utilizando estos comandos pero sin duda con el paso del tiempo iremos haciéndonos a *ip*.

El comando *ip* forma parte de la *iproute2 suite*. Su uso está destinado a mostrar información sobre la configuración de los dispositivos de red, configurar rutas, políticas de rutas y túneles. Veremos este comando más a fondo como siempre en la sección de comandos correspondiente a este capítulo – 7. No obstante y como venimos haciendo a lo largo de la guía de estudios, vamos a nombrar sus principales funciones.

- Mostrar la configuración de las interfaces de red (parecido a *ifconfig*):

```
$ ip addr [list|show] [dispositivo]
```

- Activar o desactivar una interfaz:

```
$ ip link set eth0 up | down
```

- Configurar un dispositivo de red:

```
$ ip addr add 192.168.1.25/24 broadcast 192.168.1.255 dev eth0
```

- Eliminar la configuración *IP* de un dispositivo:

```
$ ip addr del 192.168.1.25/24 dev eth0
```

- Activar el modo *multicast* para eth0:

```
$ ip link set dev eth0 multicast on
```

- Informar sobre la tabla de rutas:

```
$ ip route [list|show]
```

En realidad la sintaxis de *ip* es sencilla, admite pocas opciones tras el comando y luego indicamos el objeto (*link*, *addr*, *route*, *tunnel*, etc…) sobre el que realizaremos una determinada acción (*set*, *show*, *on*, etc..) para un dispositivo (*dev*) concreto (*eth0*). Existen casos donde detrás del dispositivo se indica un parámetro como en el caso de activar (*on*) una interfaz. Podremos recurrir a la man de *ip* o también podemos obtener información sobre un objeto concreto mediante *help*:

```
$ ip link help
```

Diagnóstico de red

Cuando configuramos una red en ocasiones las cosas no siempre funcionan como se planean, es por ello que necesitamos recurrir a ciertas herramientas de diagnóstico o depuración de red para encontrar posibles fallas o simplemente porque deseamos auditar una red en busca de errores o funcionamientos poco fiables. *Linux* ofrece gran variedad de utilidades destinadas a este tipo de tareas. Vamos a estudiar en las siguientes secciones aquellas que son requisito imprescindible para hacernos con la certificación *LPI 1*.

- Verificar la conectividad con un host remoto: **ping** (*ping6*)
- Rastrear una ruta entre dos puntos: **traceroute** (*traceroute6*) y **tracepath** (*tracepath6*)
- Auditar la red: **netcat** (o *nc*), **netstat**, **iptraf**, **lsof** y **nmap**
- Sniffer, husmeador de red: **tcpdump** y **Wireshark**
- Herramientas de diagnóstico adicionales (aunque no estén diseñadas para tal fin): **Telnet** y **FTP**

Nota: Las variantes "*nombre***6**" están diseñadas para trabajar con *IPv6*. Por lo general suelen funcionar casi idénticamente a las de *IPv4*.

Verificar la conectividad básica

Uno de los primeros pasos que podemos dar para verificar si nuestra red tiene una buena configuración, o al menos la suficiente como para llegar a un destino especificado es usar el comando **ping** o **ping6**, este último es el análogo de ping pero para direcciones *IPv6*

Información: A partir de este momento cuando hablemos de *ping6*, *traceroute6* o *tracepath6* estaremos refiriendonós a los comandos usados para direcciones *IPv6*.

Con *ping* podremos verificar la conectividad con otros puntos de la red. Para ello el comando envía un paquete *ICMP* al sistema remoto y espera una respuesta. Esto lo hace de forma reiterada y sin cesar pero podremos limitar el número de envíos mediante la opción **-c** <*num*>, donde *num* es el número de envíos que queremos realizar o bien detener los envíos pulsando **Control-C**, lo que cancelará la ejecución del comando.

```
# ping -c 4 80.25.32.18 | unequipo | unequipo.sudominio |
www.unservidor.edu
```

Podremos usar cualquiera de las opciones anteriores para indicar el host, pero solo una. En este caso ping enviará cuatro paquetes. Una respuesta al comando *ping* podría ser la siguiente:

```
PING google.es (216.58.211.195) 56(84) bytes of data.
64 bytes from mad01s25-in-f3.1e100.net (216.58.211.195): icmp_seq=1
ttl=55 time=37.5 ms
```

```
64 bytes from mad01s25-in-f3.1e100.net (216.58.211.195): icmp_seq=2
ttl=55 time=37.7 ms
64 bytes from mad01s25-in-f3.1e100.net (216.58.211.195): icmp_seq=3
ttl=55 time=38.1 ms
64 bytes from mad01s25-in-f3.1e100.net (216.58.211.195): icmp_seq=4
ttl=55 time=37.7 ms
--- google.es ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 37.571/37.829/38.164/0.213 ms
```

Podemos ver que los 4 paquetes fueron recibidos por el host y que ninguno se perdió y todo en un tiempo de *3004ms*, lo que indica que tenemos conectividad con el host remoto.

Otras opciones como *-i*, *-f*, *-n*, *-l*, *-p* o *-s* pueden ser usadas tanto con ping como con ping6. La opción **-i** nos permite indicar un intervalo de tiempo entre el envío de cada paquete *ICMP* al host remoto, con **-f** podemos hacer un envío másivo de ping, esta opción es incompatible con la opción *-i* y lo que hace por defecto es envíar los paquetes con la misma velocidad que los recibe o cien veces por segundo, dependiendo de cual de ellos es mayor. El parámetro **-n** muestra solo salidas numéricas, es decir no hace ningún intento por buscar nombres simbólicos para los host remotos. Con **-l** *ping* intentará envia tantos paquetes tan rápido como le sea posible antes de volver a su comportamiento normal. La opción **-p** resulta útil para diagnosticar problemás de red relacionados con los datos, nos permite mediante un patrón *-p <patrón>* se puede completar un paquete con más 16 bytes. Por su parte **-s** especifica el número de bytes que se van a enviar. La cantidad por defecto es *56* que sumando los *8* de la cabecera de paquetes *ICMP* pasan a ser *64 bytes*. Existen otras opciones pero son poco usuales.

Una de las conclusiones que podemos sacar de la respuesta de *ping* es que si podemos hacer ping a máquinas dentro de nuestra red local pero no a máquinas externas a esta, es que tenemos un problema de configuración hacia el exterior, o si podemos hacer ping a una máquina mediante su *IP* pero no a su nombre, quizás el servicio *DNS* esté experimentando problemás.

Rastrear una ruta

Una vez comprobada la conectividad con un host remoto, quizás nos interese conocer la traza o ruta que sigue el paquete hasta llegar al destino o bien determinar si existe un problema en la conectividad de la red. Esto podemos hacerlos con los comandos**tracepath** (**tracepath6**) o **traceroute** (**traceroute6**).

El comando *traceroute* envía una serie de tres paquetes de prueba a cada ordenador entre nuestro sistema y el host remoto especificado, mostrando todos los *routers* que existen de por medio y los tiempos de respuesta. Su uso es básico, aunque admite algunas opciones. Podemos ejecutar *traceroute* con su opción **-n** para mostrar las direcciones *IP* de los ordenadores de destino en vez de sus nombres, lo que puede

acelerar el proceso, aunque en ocasiones es igualmente imporante conocer los nombres de estos. Aún así *-n* hace la salida más legible en la mayoría de los casos

```
$ traceroute -n 82.123.78.50
```

Con *traceroute* podemos localizar también problemás de conectividad en la red, por ejemplo una alta variabilidad en los tiempos de respuestas o la ausencia de algunos de estos, lo cual se indica mediante asteríscos ‘‘, puede indicar que el router en el que nos encontramos el ‘‘ está sobrecargado o que tiene una conexión poco fiable. Un salto considerable en los tiempos, suele significar que existe una enorme distancia entre los dos router (habitual en conexiones entre continentes). No obstante si los dos sistemás están lo bastante cerca como para que este salto enorme carezca de sentido deberemos de realizar comprobaciones sobre la red, pues lo más seguro es que existan problemás sin determinar.

El comando *traceroute6* es equivalente a *traceroute -6* y con ellos podremos usar opciones como **-i** que nos permite especificar por cual interfaz de red queremos enviar los paquetes, especificar el número máximo de saltos con **-m** <*hops*> (por defecto es 30), indicar una gateway alternativa con **-g** <*gateway*> o indicar el tiempo límite de espera para una respuesta, **-w**

Una alternativa a traceroute es *tracepath*. A diferencia de *traceroute*, *tracepath* genera una línea de salida por cada paquete de prueba, por lo que su salida es más extensa. *tracepath* admite menos opciones, aun así coinciden en su opción **-n**, dándonos además la posibilidad de mostrar tanto la dirección *IP* como el nombre de host mediante la opción **-b**. De la misma manera podemos especificar el número de saltos máximos con **-m** e indicar el puerto inicial de destino con **-p**, algo que también nos permite hace *traceroute* y que no hemos comentado. El puerto de destino se va incrementando en uno a medida que se envían paquetes de pruebas.

Nota: Igualmente *tracepath6* admite las opciones de *tracepath* solo que en este caso, debemos de escribir *tracepaht6* y no *tracepath -6*.

Auditar la red

Existen multitud de redes para diagnosticar el estado de la red pero aquí solo abarcaremos las siguientes:

- **netcat** o **nc**: Es una de las herramientas de diagnostico y seguridad más populares dentro de la comunidad *GNU/Linux*. Trabaja tanto con conexiones *TCP* como *UDP* y permite hacer cosas como por ejemplo abrir puertos en host remoto quedándose a la escucha de posibles conexiones, rastrear puertos abiertos o realizar transferencias de archivos bit a bit entre dos equipos. Funciona tanto para *IPv4* como para *IPv6*.

Para dejar a *nc* como servidor a la escucha (*-l, listen*) en un determinado puerto. *nc* aceptará la conexión de un único cliente y se cerrará:

```
# nc -l <unpuerto>
```

Para abrir una conexión como cliente del servidor del ejemplo anterior:

```
# nc <equiposervidor> <unpuerto>
```

Abrir una conexión en el servidor para recibir datos desde un cliente:

```
# nc -l <unpuerto> > <archivo-donde-se-vuelva-el-contenido>
```

Enviar desde el cliente un archivo para que sea volcado en la conexión del ejemplo anterior:

```
# nc <equiposervidor> <unpuerto> < <archivo-que-será-volcado-en-el-
servidor>
```

Revisar que puertos *TCP* (opción por defecto) están abiertos dentro de un ranto de puertos determinado (para nuestro ejemplo usaremos el rango 10-50) en un sistema:

```
# nc -vz <unequipo> 10-50
```

Netcat acepta algunas opciones entre las que destacamos (a parte de **-l**, *listen* y **-vz**) **-p** para especificar un puerto, **-k** para cambiar el comportamiento de escucha del primero de los ejemplos, es decir aceptar infinitas conexiones y evitar que la conexión se cierre, cambiar el protocolo de puerto por defecto *TCP* a *UDP* con **-u**, especificar un retraso (*delay*) en el envío o recepción de mensajes **-i**.

- **netstat**: Puede considerarse la herramienta por excelencia para el diagnóstico de red. Dependiendo de los parámetros que le pasemos podrá utilizarse en lugar de muchas otras. Si queremos recibir información sobre nuestras interfaces de red (similar a *ifconfig*) podemos pasar el parámetro **-i**, *–interface*. Para obtener un listado de la tabla de enrutamiento (similar a route) usamos la opción **-r**, *–route*. La opción **-N** o *–másquerade* nos informará sobre las conexiones en las que intervenga las funciones de *NAT*. Un uso de *netstat* sin opciones puede mostrarnos gran información, como por ejemplo los puertos que tenemos abiertos con conexiones establecidas o cerrados y además los hosts y servicios que se conectan a ellos. Podremos usar la siguiente línea para indicar valores específicos:

```
# netstat -punta <host>
```

Donde:

-p muestra el *PID* y el servicio que hace uso del puerto, **-u** informa de los puertos *UDP* (además de los *TCP*, opción por defecto), **-n** expresa los host y puertos mediante números (útil si queremos redireccionar la salida con una pipe para buscar un puerto específico con *grep*), **-t** muestra el estado de la conexión (*LISTEN*, *ESTABLISHED*, *CLOSE*, *CLOSE_WAIT*, etc..) y **-a** muestra todas las conexiones y puertos de escucha. Si queremos encontrar directamente servidores que escuchan conexiones podemos agregar **-l** a *-p*

Importante: El uso de la opción **-p** suele desplegar líneas de más de 80 caracteres, podemos abrir una terminal adecuada para estas líneas o redirigir la salida a un archivo, para su comodo estudio.
Algo que conviene comentar sobre la salida del comando *netstat* es el significado por ejemplo de las columnas "*Local Address*" y "*Foreign Address*" que más o menos queda claro que refieren a la dirección local y dirección remota respectivamente de la conexión. Otro aspecto es el uso del asterísco '*****' en el campo de las direcciones *IP* o puertos. Este metacaracter representa que no hay activamente una conexión para un determinado servicio, el cual si que se encuentra en ejecución y es posiblre contactar con él.

Nota: Con el comando netstat podremos averiguar quien está conectado a un puerto determinado o si es necesario tener algunos puertos abiertos o servicios corriendo innecesariamente en nuestro sistema.

- **iptraf** : Herramienta basada en consola que proporciona estadísticas de red a partir del trafico *TCP* de nuestras interfaces de red. Es parecida a *netstat* pero posee ciertas ventajas y características que la hacen bastante útil, como por ejemplo la posibilidad de medir el ancho de banda utilizado en cada conexión, monitorizar el tráfico *IP*, posee un módulo de estadísticas sobre *LAN* que permite descubrir *hosts* en la red además de mostrar los datos de su actividad, admite la audición de múltiples protocolos como *IP*, *TCP*, *UDP*, *ICMP*, *ARP* y *OSPF* entre otros.

Podemos descargar *iptraf* desde los repositorios oficiales de nuestra distribución.

- **lsof**: La utilidad *lsof* no está diseñada precisamente para el diagnóstico de la red pero al ser una herramienta desarrollada para la monitorización del sistema podremos utilizarla igualmente para monitorizar ciertos aspectos de la red. En realidad lo que hace *lsof* es identificar que archivos están abiertos en el sistema: archivos regulares, directorios, tuberías, librerías o socket de red entre otros, quién accede a ellos, nombre e *ID* del proceso, la ruta absoluta del archivo, su descriptor, etc… útil por ejemplo si queremos desmontar un sistemás de archivos (incluso los de red) pero existen conexiones abiertas a determinados archivos y queremos tener información al respecto

Nota: En el apartado de comandos estudiaremos las opciones de *lsof*, a continuación solo mostraremos el uso de la opción **-i** que es la que nos interesa en este momento.

Podremos usar *lsof* en lugar de *netstat* para determinadas tareas, entre las que se incluye localizar los servidores que se encuentran en uso gracias a su opción **-i**. En su forma más básica este comando acompañado de está opción nos motrarán conexiones entre hosts. Estas conexiones se indican mediante la sintaxis **host:puerto->hostremoto:puerto**. Al igual que con *netstat* si encontramos líneas que contengan un asterísco en el nombre del host seguido de un nombre de puerto tal que así ***:ftp** indicará en nuestro caso que estamos corriendo el servicio *ftp* pero con ninguna conexión activa en ese momento.

Si queremos restringir la salida a una determinada dirección podremos hacerlo mediante la siguiente sintaxis:

```
# lsof -i [4 | 6] [protocolo] [@nombrehost | direcciónIP] [:servicio | puerto]
```

Nota: 4 o 6 indica si es *IPv4* o *IPv6*. Para indicar un protocolo servicio o puerto podremos guiarnos por la información del archivo */etc/services*

Para realizar una auditoria general de las conexiones de red del sistema, debería escribir *lsof -i*, sin restringir la salida. De esta manera podremos ver conexiones sospechosas a las que deberemos de prestaler especial atención.No obstante podremos restringir las salidas con comandos como:

```
# lsof -i :ftp
```

Este comando mostrará todas las conexiones entrantes o salientes del puerto 21. Si no obtenemos salida queda claro que no existen conexiones.

```
# lsof -i |grep LISTEN
```

Mediante el uso de la tubería del ejemplo anterior podremos localizar directamente que servidores se encuentran a la escucha.

Nota: Si ejecutamos lsof como usuario normal solo veremos las conexiones relacionadas con nuestra propia red. Lo ideal es ejecutarlo con *root*.

- **nmap**: El utilitario *nmap* es un escaner de puertos abiertos tanto del ordenador local como de un ordenador remoto. El comando *nmap* trabaja por defecto con paquetes **SYN**, paquetes que intentan abrir conexiones bajo *TCP*. Cuando escaneamos puertos con nmap podemos encontrarnos con tres estados diferentes del puerto: OPEN (*abierto*), CLOSE (*cerrado*) o FILTERED (*puerto no accesible, un cortafuego lo está filtrando*).

Como la gran mayoría de herramientas *Linux*, *nmap* admite varias opciones, permitiendo incluso exámenes "*secretos*" que probablemente no detecten la mayoría de los cortafuegos y exámenes mediante *ping* (*# nmap **-sP** <IP>*, si usamos la opción **-n** evitamos conversiones *DNS* por lo que el examen será más rápido) para detectar que

host están activos. Vamos a tratar las opciones de *nmap* como modificadores entre los que destacan **-sS** que realiza un escáner que no deja registro en el sistema (requiere usuario privilegiado), **-sT** realiza un barrido de puertos *TCP* (no necesita ser ejecutado con root), **-sU** provoca un barrido de puertos *UDP* útil si necesitamos conocer puertos de nivel superior que pueden estar tras un firewall (realiza exámenes más exactos), **-sA** usa paquetes *ACK* para lograr que el sistema responda (muchos firewall no filtran estos mensajes, por lo que podemos detectar puertos abiertos más sutilmente), **-sN** y **-sF** puedes pasar un *firewall* mal configurado y permitirnos ver que servicios se están ejecutando en el sistema. Por último **-sV** que intenta identificar los servicios en relación con los puertos abiertos (puede detectar vulnerabilidades).

Utilidades como **Nessus** (otro escaner de puertos montado sobre nmap) ofrecen una *GUI* y medios para realizar pruebas automatizadas y más sofisticadas que las que proporciona *nmap*. Pueden incluso localizar vulnerabilidades conocidas, de modo que pueden indicar si un servidor esta en riesgo. Nessus puede funcionar tanto como cliente como de servidor. Será el cliente quien controle al servidor, que es quien realmente hace el trabajo.

Nota: Cuando utilicemos un scaner de red, deberemos de tener presente que los puertos que vemos desde nuestro sistema pueden no ser los mismos que ve un atacante. Es un detalle a tener en cuenta si el sistema que estamos probando esta trás un cortafuego. Es probable que nuestro equipo de pruebas revele puertos que realmente no son accesibles desde el exterior.

Examinar el tráfico de red sin procesar

Para examinar el tráfico de red, lo normal es usar un sniffer (husmeador), que son herramientas avanzadas para la resolución y diagnóstico de problemás en redes, ya que proporcionan una gran cantidad de información. Linux nos ofrece muchas herramientas para tal fin, pero la que se abarca en el exámen es **tcpdump**, no obstante veremos alguna otra como **Wireshark** (antes conocido como **Ethereal**).

tcpdump puede interceptar paquetes de red para registrarlos o mostrarlos por pantalla o, incluso almacenar los resultados en un archivo para su posterior estudio, siempre que empleemos la opción **-w**.

Para ejecutar *tcpdump* necesitaremos ser el administrador del sistema (*root*) y basta con escribir el nombre del comando (*# tcpdump*) aunque podemos modificar la salida con algunas opciones. Si queremos mostrar el contenido de los paquetes en código *ASCII* deberemos de indicarlo con **-A**. Para mostrar una lista de interfaces en las que puede escuchar *tcpdump* usaremos **-D**. Como en muchos comandos de los anteriormente estudiados, si empleamos la opción **-n** obtendremos salidas con direcciones *IP* en lugar de nombres y al igual pasa con la opción **-v** (o **-vv**, **-vvv**) de otros comandos, con la que podremos ampliar la información.

Por lo general las líneas de salida del comando *tcpdump* suelen ser extensas e incluso pueden llegar a ocupar más de una línea de la pantalla. Suelen incluir una marca temporal, un identificador de pila, el nombre o la IP y el puerto del sistema de origen y destino (respectivamente) e información específica del paquete. Podemos usar la opción **-c** <*num*> para que muestre solo un número determinado de paquetes y luego finalice, al igual que hacemos con el comando *ping*, o pulsar directamente **Control+C** para finalizar la ejecución del comando.

Wireshark es un analizador de protocolos utilizado para realizar análisis y solucionar problemás en redes de comunicaciones. La funcionalidad que provee es similar a la de *tcpdump*, pero añade una interfaz gráfica y muchas opciones de organización y filtrado de información. Permite ver todo el tráfico que pasa a través de una red (usualmente una red *Ethernet*, aunque es compatible con algunas otras) estableciendo la configuración en modo promiscuo. También incluye una versión basada en texto llamada **tshark**.

Uso de herramientas adicionales

Existen otras herramientas que aunque no fueron diseñadas para la depuración de la red, pueden resultarnos útiles para deducir cierta información sobre la configuración de aspectos particulares de cierto ordenador, como por ejemplo comprobar si tiene un puerto específico abierto. Estas herramientas pueden ser **Telnet** y **ftp**.

Telnet es una herramienta de acceso remoto. No se recomienda su uso para tal fin ya que carece de encriptación. Por regla general debería de eliminar el servidor *Telnet* de su sistema y no utilizar nunca el programa cliente *telnet*. El programa **SSH** es una alternativa mucho más segura para conexiones remotas. Estudiaremos el programa *SSH* en el *capítulo 10*. No obstante vamos a utilizar *telnet* en este caso para comprobar si una máquina dispone de un puerto abierto, como ya adelantábamos. Por ejemplo vamos a comprobar si el equipo '*freeser*' permite el acceso por el puerto 25 (puerto utilizado por *SMTP*)

```
# telnet freeser 25
```

Si obtenemos el mensaje de error **Connection refused** (*conexión rechazada*) indicará que el servidor remoto no está en ejecución o es inaccesible (puede que un firewall esté filtrando el puerto).

Nota: Esta prueba sólo funciona con protocolos que utilizan *TCP*

De la misma manera, el programa **ftp** como ya mencionamos puede sernos util por ejemplo para comprobar si tenemos acceso a una máquina por el puerto 21, o incluso para comprobar el tiempo de transferencia y la velocidad de la red. Para esto tendremos que conectar primero con el servidor:

```
# ftp freeser
```

Nos aparecerá un prompt que aceptará comandos como (*cd*, *get*, *put*, *ls*, etc…) nosotros usaremos **get** y **put**. Una vez estemos en el directorio donde se aloja un archivo de peso (ideal para comprobar las tasas), lo descargaremos a nuestro sistema con el comando *get* <*archivo*>. Si lo que queremos es comprobar la subida usaremos *put* <*archivo*> para subir un archivo desde nuestro sistema.

Al igual que *Telnet*, *FTP* es una pobre elección como protocolo por motivos de seguridad.

LPIC-1 Capítulo 9

EN ESTE CAPÍTULO SE ABARCAN LOS SIGUIENTES OBJETIVOS DEL EXAMEN:

- 105.1: Personalizar y utilizar el entorno de la consola (4)
- 105.2: Programar shell scripts (4)
- 108.3: Funcionamiento y configuración del correo electrónico (3)
- 105.3: Administración básica de datos SQL (2)

El entorno de Consola, Shell Scripts, el Correo Electrónico y uso básico de SQL

Administrar el entorno de la consola

En el Capítulo 1 "*Herramientas básicas para la línea de comandos Linux*" vimos las consolas más comunes para entornos *Linux*, como se exportaba una *variable*, diferencia entre comandos internos y externos a la *shell*, y alguna que otra característica del empleo del modo comando. Durante el Capítulo 7 "*Administrar el sistema*" aprendimos como crear un entorno para un usuario y que archivos eran los que definirían sus características, archivos que dependían en gran medida del directorio */etc/skel* (comúnmente llamado directorio esqueleto). Vimos que este directorio contenía aquellos archivos con los que de algún modo queríamos preconfigurar o proporcionar una configuración local base para un entorno de usuario. Vamos a repasar brevemente estos archivos puesto que a continuación veremos en cual de ellos deberemos de modificar datos dependiendo de donde queremos que tomen efectos, bien de forma local y aislada para un usuario, o de forma global, lo que afectará a todos los usuarios del sistema.

Nota: Un usuario individual puede invalidar aquello que se configure de modo global, mediante la edición de los archivos que configuran su entorno, por lo que no es recomendable usar los archivos de configuración global para determinar aspectos sobre la seguridad del sistema.

Los archivos de configuración del entorno pueden clasificarse de la siguiente manera:

Archivos de configuración	Archivos que se ejecutan durante el proceso de acceso al sistema	Archivos ejecutados otros procesos, c ejemplo al abrir un Xterm
Global (afecta a todos los usuarios del sistema)	• */etc/profile* • */etc/profile.d/**	• */etc/bashrc* • */etc/bash.bas*
Individual (afecta únicamente al entorno de un determinado usuario)	• *~/.bash_login* • *~/.bash_profile* • *~/.profile*	• *~/.bashrc*

Entonces, explicando brevemente la tabla anterior, diríamos que si queremos configurar el diseño de un *prompt* que afecte a todos los usuarios del sistema, deberíamos de definir la variable *PS1* en el archivo */etc/bashrc* (o */etc/bash.bashrc*, dependiendo de la distribución) que es el archivo que contiene el valor para *PS1*. De este modo el *prompt* cambiará tanto si accedemos en un modo de ejecución de consola como si accedemos en un *runlevel* gráfico y posteriormente abrimos una terminal, pero… ¿Por qué también si accedemos directamente en un modo de solo consola si es */etc/profile* el que se ejecuta y no */etc/bashrc*? – Porque si nos fijamos en el contenido de */etc/profile* veremos que llama a */etc/bashrc* y entonces este setea el *prompt* para el usuario.De este modo a menos que comentemos las líneas de */etc/profile* en las que se llama a */etc/bashrc* en ambos casos el *prompt* cambiará y además de forma global.

Si lo que queremos es que el nuevo valor de la variable *PS1* solo afecte a un determinado usuario y además únicamente cuando abre una terminal desde el entorno de escritorio, es decir el *prompt* no cambiaría si accedemos en un modo de ejecución de modo consola, tendremos que definir la variable *PS1* en el archivo *~/.bashrc* del usuario en cuestión y además tendremos que comentar las líneas de *~/.bash_profile* (o *~/.profile* dependiendo de la distribución) en las que se llama a ~/.bashrc. De lo contrario el *prompt* cambiará esta vez solo para ese usuario, pero en ambos casos, es decir, tanto si el usuario accede directamente desde el modo consola como si lo hace desde el entorno gráfico a través de una terminal virtual.

Nota: Dependiendo de la distribución podremos o no encontrarnos con el archivo *~/.bashrc*. De todos modos podremos crearlo nosotros en */etc/skel* de manera que cuando creemos un nuevo usuario, este ya cuente con el archivo.

Al igual que las consolas poseen *scripts* de inicio, puede que también cuenten con *scripts* de desconexión o *logout*. Estos *scripts* como suponemos, se ejecutan cuando el usuario se desconecta. El *script* ejecutado normalmente es *~/.bash_logout* el cual puede ejecutar programás para limpiar los directorios temporales, eliminar claves de seguridad de la memoria o realizar otras tareas que incluso nosotros mismos hayamos programado para que sean ejecutadas.

Advertencia: Debemos de comprobar que es lo que realiza este *script* o si hemos programado algunos comandos para que sean ejecutados al desconectar, deberemos de tener en cuenta, que cabe la posibilidad (muy frecuentemente) que tengamos más terminales del mismo usuario abiertas, por lo que si cerramos una y se ejecuta el *script* puede que eliminemos archivos que estamos usando en otras terminales, o simplemente no queremos que sean eliminados hasta que la última sesión sea cerrada.

Otro archivo de configuración de *bash* es *~/.inputrc*. Este archivo sirve para personalizar la configuración del teclado. La sintaxis básica de las líneas del archivo es:

```
combinación_de_teclas: acción_a_realizar
```

Si tenemos la sensación de que determinadas combinaciones de teclas no funcionan como deberían funcionar en modo texto, es aconsejable investigar la configuración de este archivo.

Nota: La configuración de archivo *~/.inputrc* no afectará a los programás que se ejecuten en X, aunque sean programás en modo texto que se ejecuten dentro de ventanas *Xterm*.

Sobre la variable PS1

Hacer un pequeño inciso para comentar el valor de la variable *PS1*. La variable *PS1* como hemos mencionado, define el aspecto del *prompt* por ejemplo, para un *prompt* estándar el aspecto sería el siguiente:

```
nebul4ck@LiMinCinn ~ $
```

Lo que indica que el usuario es *nebul4ck*, el equipo se llama *LinMinCinn*, nuestro directorio de trabajo actual es nuestro home (**~**) y el caracter **$** nos identifica como usuario estándar (**#** para *root*). En este caso la variable *PS1* tendría el siguiente valor:

```
PS1='\u@\h \W \$ '
```

Disponemos de las siguientes variables del sistema para modificar esto:

- Indicar la fecha del sistema: **\d**
- El nombre del la máquina: **\h**
- El *shell* que utilizamos: **\s**
- El nombre del usuario activo: **\u**
- La versión del bash: **\v**
- El directorio de trabajo: **\w**
- El número de histórico del comando ejecutado: **\!**
- Indicar $ para usuario estándar y # para root: \$

Definir variables y alias de comandos

Los *scripts* de *bash* que acabamos de recordar y otros nuevos que hemos mencionados sirven como hemos visto para configurar un entorno, pero para ello se necesitarán comandos que ejecutar y construir una cierta lógica que permita al archivo realizar determinadas acciones. A continuación vamos a ver como declarar una variable de entorno y un alias de comando en un *shell script* y dejaremos para la próxima sección el estudio de funciones y estructuras de las que nos serviremos para programar un *script* en *bash*.

Recordemos también que las variables de entorno proporcionan los medios para pasar datos mediante nombres a programás iniciados desde una consola. Quizás un programa necesite conocer el nombre del ordenador, es por ello que existe una variable (y si no existe podremos crearla y exportarla para nuestro entorno en los archivos de inicio de consola anteriormente estudiados) llamada *$HOSTNAME*. Vamos a imaginar que esta variable no existiese en nuestro entorno y un programa de terceros señala en su documentación que el programa necesitará del valor de la variable *$HOSTNAME*, el cual debe de ser el nombre del equipo que ejecuta el comando. Pues bien para definir esta variable de entorno, editaríamos el archivo *~/.bash_profile* (o *~/.profile*) declarando la variable de la siguiente manera:

```
HOSTNAME=freeser
export HOSTNAME
```

Nota: Si el valor de la variable contiene espacios es obligatorio encerrarlo entre dobles comillas.

En la primera línea declaramos la variable *$HOSTNAME* y en la segunda convertimos a la variable local en variable de entorno de manera que en cuanto el *script* de inicio sea ejecutado, esa variable estará disponible para cualquier programa.

Cuando exportamos una variable directamente desde la línea de comandos, el valor de esta solo estará disponible para la terminal desde la que ha sido exportada, anulándose su valor en cuanto cerremos la sesión o en cualquier otra terminal virtual.

Podemos hacer que una variable sea de solo lectura anteponiendo la palabra '*readonly*' al nombre de la variable, de manera que su valor no pueda ser modificado pero si visualizado hasta que el proceso que hace uso de ella (un script, un determinado shell o sesión…) haya finalizado, es decir, en teoría una variable que se ha modificado para solo lectura no puede ser modificada ni eliminada hasta que el proceso que la utiliza finalice.

Cuando definamos una variable en un *shell script*, no será necesario exportarla, si no que la definiremos y posteriormente el programa irá haciendo uso de ella conforme se

vaya necesitando. Una variable debe de empezar por una letra o un guión bajo '_' pero no por un número. A este tipo de variables simples se les conoce como **escalares** (*scalar*)

Podemos ver el valor de una variable concreta desde la línea de comandos así:

```
$ echo "$HOSTNAME"
```

Nota: Fijémonos en la importancia de preceder el nombre de la variable con el signo '**$**', algo que no se debe de hacer cuando se declara o define.

Si lo que queremos es ver todas las variables definidas para un entorno utilizaremos el comando **env**. Este comando dispone de un par de opciones útiles pero para listar todas las variables bastará con invocarlo sin parámetro alguno:

```
nebul4ck@LiMinCinn ~ $ env
SSH_AGENT_PID=1872
TERM=xterm
SHELL=/bin/bash
USER=nebul4ck
USERNAME=n0name
LOGNAME=nebul4ck
PWD=/home/nebul4ck
HOME=/home/alberto
LANG=es_ES.UTF-8
...
```

Nota: La salida del comando *env* ha sido cortada ya que suele ser bastante extensa.

Podemos definir una variable para un único programa con *env* de la siguiente manera:

```
$ env DISPLAY=freeser.4sysadmins.es:0.0 nedit
```

Nota: En la mayoría de los casos podremos omitir el comando *env*

Esto provocará que el programa **nedit** intente iniciar usando la pantalla *:0.0* en *freeser.4sysadmins.es* en lugar de la pantalla local predeterminada o definida en la variable *$DISPLAY*

Si queremos comenzar con un entorno vacío es decir, ignorando el entorno heredado, acompañaremos a *env* del parámetro **-i** (*–ignore-environment*) y para anular el valor de una variable específica usamos **-u** <*nombre_variable*> o **–unset=***nombre_variable*.

Antes vimos como mostrar el valor de una variable mediante el comando *echo*. Otra forma de hacerlo con *env* sería así:

```
$ env | grep NOMBRE_VARIABLE
```

Para terminar con las variables de entorno vamos a presentar una lista de las que quizás sean las más conocidas o nos vayan a ser más útiles:

- **USER** o **USERNAME**: Nombre de usuario actual
- **SHELL**: Contiene la ruta a la consola de comandos en uso
- **PWD**: Esta variable la mantiene el sistema y su valor es el directorio de trabajo actual. Por ello puede ser utilizada por programás para buscar archivos cuando no se proporciona la ruta completa.
- **HOSTNAME**: Como ya vimos anteriormente, esta variable contiene el nombre del equipo de trabajo
- **PATH**: Es una variable cuyo valor es lo suficientemente importante como para abrir una brecha de seguridad. Su cometido es contener rutas de directorios separados por el caracter '**:**', por ejemplo *PATH=/bin:/sbin:/usr/bin:/usr/sbin:/home/nebul4ck/bin*, este valor indicaría donde buscar archivos ejecutables, de manera que el usuario pueda ejecutar aquellos programás que se encuentran bajo estas rutas (sea cual sea su directorio de trabajo), ahorrando la necesidad de escribir *./archivo_ejecutable*. Se recomienda no añadir la ruta '**.**' (directorio actual) al valor de esta variable, especialmente para el usuario *root* ¿Porqué? – Un usuario malintencionado podría crear en un directorio que solemos frecuentar un programa llamado por ejemplo '**ls**' de manera que cuando *root* invoque a *ls* en el directorio donde se encuentra el programa malicioso, este sea ejecutado en vez de el comando *ls* del sistema. En el caso de que sea necesario para un usuario normal, deberemos de colocar esta ruta al final del valor de la variable PATH
- **HOME**: El valor de esta variable es la ruta de nuestro directorio principal, y al igual que la variable *PWD*, puede ser utilizada por programás
- **MAIL**: Su valor es la ruta del buzón de correo del usuario en cuestión, que suele ser */var/spool/mail/nombreusuario*
- **LANG**: Utilizada para guardar el idioma actual
- **TZ**: Indica la zona horaria que tenemos configurada o podemos emplearla precisamente para eso, para configurar nuestra zona horaria en caso de que estemos usando un ordenador remoto
- **LD_LIBRARY_PATH**: Indica los directorios en los que los programás pueden encontrar archivos de bibliotecas. Es similar a *PATH* pero en vez de ejecutables son librerías
- **PS1**: También hemos visto ya la función de esta variable, la cual define el *prompt* del sistema
- **TERM**: Indica la terminal en uso.
- **DISPLAY**: Otra variable ya vista, identifica la pantalla utilizada por *X* que suele ser *:0.0*. Este valor puede ir precedido del nombre de un host remoto o cambiar a *:1.0* por ejemplo si tenemos dos sesiones de *X* abiertas (*:0.0* para la primera sesión y *:1.0* para la segunda).
- **EDITOR**: El editor de texto iniciado por defecto será aquel que conste en el valor de esta variable.

Igual que definimos las variables podemos definir los **alias** de comandos, pero… ¿Que son o para que sirve? – La principal ventaja de utilizar alias de comandos es la de crear una versión acortada de un comando, para que cada vez que queramos que un comando

actúe de determinada manera no tengamos que escribirlo de forma completa. Vamos a ver esto con un ejemplo sencillo:

Supongamos que cuando ejecutamos **ls -l**, el listado que nos devuelve el comando es un listado de directorios y archivos sin color, es decir el mismo color utilizado para las letras de la consola. Cabe la posibilidad de listar el contenido de un directorio de forma que los archivos sean mostrado en diferentes colores, para que a simple vista reconozcamos de que tipo de archivo se trata, por ejemplo *verde* para ejecutables, *azul* para directorios, *celeste* para enlaces simbólicos, *rojo* con fondo *negro* para enlaces rotos, *amarillo* con fondo *negro* para archivos de caracter, etc… ¿y como conseguimos esto? – Para hacer que cada vez que ejecutemos el comando 'ls' a secas despleguemos un listado en formato de una sola columna (con los permisos, propietario, fecha, etc..) en color y ordenado por fecha descendente, haremos lo siguiente:

```
$ alias ls='ls -lt --color'
```

Así de sencillo es crear un alias.

Nota: Para utilizar este alias de forma persistente podremos definirlo en alguno de los archivos de inicio de consola. Dependiendo si queremos que sea aplicado de forma global o local elegiremos uno u otros archivos.

Los Arrays

Un *array* o *vector* es una variable que tiene asignado un conjunto de valores a los cuales podemos acceder mediante su número de posición, el cual nos sirve como índice de valores. Podemos definir un *array* de la siguientes maneras:

- Múltiples valores de forma simultánea:

```
$ array=(ford renault "mercedes benz" dacia vw bmw)
```

- De uno en uno:

```
$ array[0]=ford
$ array[1]=renault
$ array[2]="mercedes benz"
...
```

Nota: Los indices no necesariamente tienen que ser numéricos, por ejemplo sería válido también definir los valores del *array* tal que así: *$ array[primero]=ford, $ array[segundo]=renault, etc…*

- Igualmente podemos añadir un nuevo valor al *array*:

```
$ array[6]=chevrolet
```

Para consultar todos o un valor específico del *array*:

- Consultar un valor concreto:

```
$ echo ${array[1]}
renault
```

- Mostrar todos los valores:

```
$ echo ${array[*]}
ford renault mercedes benz dacia vw bmw chevrolet
$ echo ${array[@]}
ford renault mercedes benz dacia vw bmw chevrolet
```

Programación de shell scripts

Ahora que ya sabemos ejecutar comandos desde la terminal o desde un *script* de inicio (la forma de hacerlo es la misma), sabemos declarar variables y crear alias, vamos a estudiar la forma de crear un *script* ejecutable que sea reconocido por el *shell* y realice una serie de tareas en base a estructuras lógicas (expresiones condicionales), bucles y funciones.

Antes de comenzar ha crear un *script*, los cuales son archivos de texto plano, por lo que podremos hacerlo mediante un editor de texto corriente, debemos de saber algunas características básicas. A estas alturas no es una sorpresa saber que la consola *bash* es la más utilizada en entornos *Linux*, al menos es la que viene por defecto en la gran mayoría de distribuciones, por lo que algo normal puede ser programar un *script* de consola para *bash*. No obstante un *script* de consola *bash* puede ser ejecutado en *tcsh* y otros lenguajes de programación de consola, a menos que el *script* utilice funciones muy específicas de una consola concreta (en este caso *bash*).

La línea shebang o hashbang

Shebang o **hashbang** es el nombre con el que se conoce a la primera línea por la que está formado un *script* de consola (*shell script* de ahora en adelante), aunque también recibe otros nombres como **hashpling** o **poundbang**. Antes de mostrar el formato de esta línea es interesante saber que a través del caracter '#' podremos escribir líneas dentro de un *shell script* las cuales no serán ejecutadas aunque si interpretadas por el kernel. Dicho esto pasamos a mostrar el contenido de esta primera línea del shell script:

```
#!/bin/bash
```

Está línea esta formada por el caracter de comentario, por el signo de exclamación '**!**' y por una ruta. Los dos primeros caracteres forman un código que le indica al *kernel* de *Linux* que se trata de un *script* y que utilice el resto de la línea como ruta al programa que va a interpretar el *script*, **bash** en nuestro ejemplo.

Nota: Es posible encontrar la ruta a otro interprete o incluso a */bin/sh* que en la mayoría de las veces no es más que un enlace simbólico a la consola de la distribución, por lo que de esta manera nos aseguramos de que el *shell script* va a ser interpretado. No obstante como comentamos antes, si tiene funciones muy específicas de una consola concreta, mejor indicar a esta.

Ejecutando un shell script

Cuando finalizamos la programación de un *shell script*, llega el momento de ejecutarlo para ver sus resultados. Existen diferentes formás de ejecutar un *script* cada cual con su debida implicación:

- Establecer con **chmod** los permisos de ejecución. Usaremos **u** para el usuario, **g** para el grupo o **a** para todos:

```
$ chmod a+x myshell.sh
```

Nota: Las extensiones como ya sabemos no son necesarias en *Linux* pero si identificativas.

- Utilizando el programa de la consola seguido del nombre del *script*. Se recomienda utilizar la forma del ejemplo anterior, aunque esto puede ser útil si no hemos conseguido modificar los permisos con *chmod*:

```
$ bash myshell.sh
```

- Ejecutar el *shell script* con el comando **exec**. Si ejecutamos un programa a través de *exec* este tomará el control del proceso de la terminal desde el que ha sido invocado y una vez finalice el *script* o programa ejecutado la terminal se cerrará. Esto puede ser útil en determinadas ocasiones aunque no es muy utilizado, por ejemplo cuando un *script* llama a otro y este a su vez a otro y así… De esta manera el nuevo toma el control del proceso del anterior. Puede ser un método útil para controlar los recursos dedicados a la ejecución de una seríe de *script*.

```
$ exec /opt/myshell.sh
```

Podemos hacer la prueba ejecutando por ejemplo nuestro editor de texto desde la consola. Veremos que cuando cerremos el editor la terminal virtual será igualmente cerrada:

```
$ exec nedit
```

El comando *exec* lo estudiamos en el Capítulo 1 concretamente entre los comandos "*Redirecciones y Canalizadores*"

- Otra forma de ejecutar un *script* es aplicarle **source**. Para ello se utiliza el propio comando *source* o el caracter '.' tal que así:

```
$ source myshell.sh
$ . myshell.sh
```

De este modo el *script* se ejecuta en la consola actual y no se inicia una instancia nueva de la consola, como sucede cuando se ejecuta un *shell script* introduciendo su nombre o utilizando el comando *exec* (que si que inicia una instancia nueva aunque esta controle a la terminal desde la que se ha ejecutado). Existen ciertas ventajas a la hora de ejecutar un *script* con *source*, por ejemplo, si queremos que el *script* haga uso de las variables de entornos definidas en la consola (aunque no se hayan exportado) desde la que se ha ejecutado, este deberá de ser llamado con *source*.

Nota: Por lo general, sólo las variables de entorno que se hayan exportado de forma explícita, estarán disponibles para los *script* que se ejecuten.

Esto pasa igualmente si un *script* que ha sido ejecutado de forma digamos '*convencional*' y en el se definen ciertas variables que se usarán a lo largo del programa. Si en un punto de este programa llamamos a otro *script* y queremos que este haga uso de esas variables, este nuevo *script* deberá de ser llamado con *source*.

Otra ventaja de ejecutar con *source* un *script* es que si este establece variables de entornos, estás estarán disponibles para la consola desde la que se ha invocado al *script*. Esto es lo que ocurre cuando se inicia una terminal y */etc/profile* o *~/.bash_profile* llaman a */etc/bashrc* o *~/.bashrc* respectivamente. Si nos fijamos en estos *script* de inicio de consola veremos que se ejecutan así:

```
if [ -f /etc/bash.bashrc ]; then
. /etc/bash.bashrc
fi
```

Esto produce que el entorno creado por *profile* pueda ser utilizado e incluso modificado por *bashrc*

También, ejecutar un *script* de esta manera evita la sobrecarga asociada a iniciar nuevas consolas, que aunque suele ser imperceptible, podría requerir de recursos si este *script*

llamara de forma normal a múltiples *scripts* secundarios generando cada uno una nueva instancia de consola.

Otro dato y como último apunte, es interesante saber que al ejecutar un *script* con *source* se ejecuta en el idioma de la consola que realiza la invocación, mientras que al ejecutar un *script* de la forma normal, se usa el lenguaje de consola especificado en la línea *hashbang*.

Uso de variables en los shell scripts

Ya hemos estudiado como definir una variable de entorno y exportarla para que esté disponible en el entorno local o global de los usuarios. Las variables de los *shell scripts* suelen utilizarse para facilitar la modificación de estos en un futuro. Los *shell script* pueden usar variables que hayan sido pasadas como parámetros, definidas internamente, extraídas del propio entorno del *script* o pasándoselas a petición del propio script

Cuando llamamos a un *shell script* bastará normalmente con escribir su nombre (si se encuentre dentro de alguno de las rutas que contiene el *path*), ejecutarlo con *./nombre_script.sh*, con el programa de consola (*bash nombre-script.sh*) o pasándoselo a *source* (*source nombre-script.sh* ó . *nombre-script.sh*). Sea cual sea la forma de ejecutarlo podremos escribir tras el nombre una serie de palabras a las que se le denominan argumentos de programa y de las que este hará uso durante su ejecución. Para el siguiente ejemplo el valor de la primera palabra podremos recuperarla desde dentro del *script* con **$1**, el valor de la segunda palabra con **$2** y así sucesivamente hasta **$9**. La variable **$0** indica el nombre del propio *script*:

```
$ ./myscript.sh Juán 24 Madrid
```

Hola me llamo Juán, tengo 24 años y vivo en Madrid.

Para recibir este mensaje tras la ejecución del *script* bastaría con que *myscript* tuviese el siguiente contenido:

```
#!/bin/bash
echo "Hola me llamo $1, tengo $2 años y vivo en $3"
```

Importante: Podemos recuperar el número de parámetros pasados al comando mediante la variable **$#**, con **$*** recuperamos todos los parámetros sin incluir a *$0* (muestra su valor, no el número), con **$$** podemos conocer el *PID* que ha creado nuestro *script* y con **$?** recuperamos el código de retorno del último comando ejecutado.

El comando **shift** desplaza los parámetros, de modo que lo que es *$2* se convierte en *$1*, *$3* pasaría a ser *$2* y así sucesivamente, menos *$0* que no es alterado. Podríamos usar *shift* junto con un *bucle* para examinar los parámetros que son pasados a un *script*.

Como siempre mediante un ejemplo lo veremos más fácil. Supongamos que hemos ejecutado la siguiente línea:

```
$ ./miscript.sh uno dos tres cuatro y cinco
```

Y que el contenido de *myscript.sh* es:

```
#!/bin/bash
echo "El número de parámetros es: $#" \\ Esta línea desplegará un 5
echo "Los valores de los parámetros son: $*" \\ El valor que despliega
esta línea es uno dos tres cuatro cinco
shift 3  \\ Con el comando shift indicamos que ahora el parámetro
primero sea $4, pues hemos hecho que empiece inmediatamente después de
la tercera posición
echo -e "Ahora que hemos usado shitf 3 el parámetro 1 es: $1 y el
parámetro 2 es: $2 \\ Esto devuelve: cuatro cinco
echo "Ahora el número de parámetros es: $#" \\ Pues el número de
parámetros ahora es: dos (cuatro cinco)
```

Para definir una variable dentro del *script* basta con escribir **nombre=valor**:

```
MYVAR="Este es mi valor"
$ echo "$MYVAR"
Este es mi valor
```

Podemos concatenar variables así:

```
YO="Soy Juán"
años=" y tengo 24 años"
Todo="$YO$años"
echo "$Todo"
Soy Juán y tengo 24 años
```

También podemos hacer que valga la salida de un comando. Para esto deberemos de rodear el comando con acentos graves tal que así:

```
COMANDO=`ls -l | cut -d" " -f1`
echo -e "$COMANDO\n"
total
lrwxrwxrwx
drwxr-xr-x
lrwxrwxrwx
...
```

Nota: El parámetro **-e** de echo como ya sabemos por el *Capítulo 1*, hace que se interpreten los caracteres especiales en este caso **\n** inserta un salto de línea.

Otras formás de utilizar variables es a través del entorno donde se está ejecutando el *script* o que sea este quien nos pida que la introduzcamos.

- Usando variables del propio entorno: Si sabemos que nuestro entorno (y si no lo sabemos lo comprobamos con *echo "$VARIABLE"*) está utilizando una variable (por ejemplo *$HOSTNAME*) podremos hacer uso de ella dentro del *script* y será reconocida, o bien si definimos como hemos hecho en el método anterior ciertas variables y luego llamamos con *source* a otro *script* dentro de este *script* que contiene a las variables, el segundo *script* (el que es llamado) podrá hacer uso de estas variables igualmente.
- El *script* nos pide que introduzcamos un valor: Podemos hacer que el propio *script* sea quien nos pregunte por algún valor que almacenará para utilizarlo como variable dentro de su contenido. Podríamos conseguir esto con el comando **read** de la siguiente manera:

```
#!/bin/bash
echo "Introduzca un nombre sin apellidos: "
read nombre
echo "Introduzca una edad: "
read edad
echo "Introduzca su ciudad de nacimiento: "
read ciudad
echo -e "\nHola me llamo $nombre, tengo $edad años y nací en $ciudad
\n"
```

Aritmética con variables

Si queremos realizar operaciones aritméticas en el *shell* (algunos *shell* antiguos cono *sh* no soportan esto) podemos utilizar el siguiente formato:

```
$((expresión))
```

Expresión estará formado por cualquier valor numérico específico, variables que contengan valores numéricos y operadores.

```
$ echo $((b=a*2))
$ b=$((a*2))
```

Esto desplegará el resultado de '**b**' que será el valor que contenga la variable '**a**' multiplicado por dos. Tenemos que prestar atención a como declaramos la variable dentro de la expresión, que como vemos no se antepone el signo '**$**', esto es porque el *shell* reconoce que al no ser un operador o valor numérico se trata de una variable. Si la variable no está previamente definida o su valor no contiene un valor numérico se le autoasigna el valor 0.

Algunos operadores aritméticos:

+ suma
* mutiplicación
– resta
/ división entera
% resto de la división entera

( ) agrupar operaciones. Al igual que en matemáticas esto fuerza a que el grupo sea resuelto en primera instancia

Uso de expresiones condicionales

La expresión condicional literal:

"*Si el valor de X es menor que el valor de Y, haz esto*"

"*Si el valor de X es igual a Y, entonces haz esto, o si no haz esto otro*"

"*Si esto es verdadero y esto otro no lo es, haz esto, si esto no es verdadero o esto si lo es, haz esto, y si no es ninguno haz esto otro*"

podríamos representarlo de la siguiente manera:

Si hablamos de dígitos:

```
if [ "$X" -lt "$Y" ]
then
        echo "El valor de X es $X y es menor que el valor de Y que es
$Y"
fi
```

Si lo que se comparan son cadena de caracteres:

```
if [ "$COCHE1" == "$COCHE2" ]
then
          echo "El coche1 es de la misma fábrica que coche2. La
fabrica es Ford"
else
          echo "La marca de los coches no coinciden"
          exit 0
fi
```

Podemos alargar la estructura así:

```
if [ "$a" -eq "$b" ] && [ "$COCHE1" != "$COCHE2" ]
then
          echo "A y B son iguales, y además coche1 y coche2 son
distintos"
elif [ ! -f "$archivo" ] || [ $(($1*$2)) -eq "$c" ]
then
          echo "No existe el archivo $archivo ó A es igual a B
else
          echo "Ninguna de las condiciones anteriores se cumplen"
          exit 0
fi
```

Nota: Podemos crear una estructura **if** a partir del comando **test** o incluso usarlo únicamente para recuperar el valor de retorno de un comando y utilizarlo luego:

```
if test $n -le $N
then
          echo "El número $n es menor o igual que $N"
fi
test -d /home/nebul4ck
if [ "$?" -eq "0" ]
then
          echo "El directorio principal existe"
fi
```

Para terminar con *if* debemos de saber que podemos anidar diferentes *if* siempre y cuando los cerremos todos debidamente, por ejemplo:

```
if [....]
then
      if [....]
      then
            .......
      else
            if [....]
            then
                  .......
            fi
      fi
else
      .........
fi
```

Test condicionales con if:

-d fichero : cierto si fichero existe y es un directorio
-e fichero : cierto si fichero existe, independientemente del tipo que sea
-f fichero : cierto si fichero existe y es un fichero normal
-r fichero : cierto si fichero existe y se puede leer
-s fichero : cierto si fichero existe y tiene tamaño mayor que cero
-w fichero : cierto si fichero existe y es se puede escribir sobre él
-x fichero : cierto si fichero existe y es ejecutable

Condicionales de dígitos:

n1 -eq n2 : cierto si los enteros n1 y n2 son iguales
n1 -ne n2 : cierto si los enteros n1 y n2 no son iguales
n1 -gt n2 : cierto si el entero n1 es mayor que n2
n1 -ge n2 : cierto si el entero n1 es mayor o igual que n2
n1 -lt n2 : cierto si el entero n1 es menor que n2
n1 -le n2 : cierto si el entero n1 es menor o igual que n2

Condicionales para cadena de caracteres:

s1 = s2 : cierto si las cadenas de texto s1 y s2 son idénticas
s1 != s2 : cierto si las cadenas de texto s1 y s2 no son idénticas

Nota: Podemos aplicar el signo de exclamación '**!**' a las condicionales de dígitos o de test de archivos para negar

s1 > s2 : cierto si la cadena de texto s1 es mayor que s2
s1 < s2 : cierto si la cadena de texto s1 es menor que s2
-n cadena : cierto si la longitud de la cadena de texto es distinta de cero

Condicionales con expresiones:

! expresion : cierto si expresion es falsa (negación)
expresion1 -a expresion2 : cierto si expresion1 y expresion2 son ciertas
expresion1 -o expresion2 : cierto si expresion1 o expresion2 son ciertas

Referencias:

http://blackshell.usebox.net/pub/shell/taller_sh/x214.html

Nota: Con **&&** indicamos que se ejecute el siguiente comando en caso de que la ejecución del anterior haya sido correcta (*ls -lt && echo "Listado del directorio"*) o bien implicamos a que la condición sea verdadera solo si se cumplen todos los requisitos (como ya vimos en uno de los ejemplos anteriores). Con || estamos diciendo que si el último comando dió error, se ejecute el siguiente (*mkdi /opt/notfound || echo "El comando no está bien escrito y falló"*) o bien la condición será cierta si uno u otro de los requisitos es verdadero. Esto también lo vimos en un ejemplo superior.

Existe otra instrucción con la que podemos usar el valor de una variable o expresión y ejecutar una acción solo en el caso de que el valor o expresión coincidan con un patrón. Esta estructura es **case**:

```
case palabra in
      patrón1) comando(s);;
      patrón2) comando(s);;
esac
```

Los patrones se pueden ampliar utilizando comodines y reglas de expansión. Además es posible incluir una condición por defecto, es decir, en el caso de que *palabra* no coincida con ningún *patrón*, se ejecute una determinada acción. Esto lo conseguimos utilizando un ***** como *patrón* por lo que deberá de ir como última opción, de lo contrario se ejecutaría siempre.

Los bucles for, while y until

Los bucles son estructuras que le indican al *script* que realice la misma tarea repetidamente hasta que se cumpla (o hasta que deje de cumplirse) una determinada condición.

for

El bucle **for** se ejecuta una vez para cada elemento que pasamos (variable, lista de nombres, salida de un comando…). Cada elemento se le asigna como valor en cada iteracción o turno a una variable con la que se realizarán diferentes acciones. Veamos esto con un ejemplo:

```
for i in negro blanco azul
do
          echo "El color es $i"
done
```

Existe un comando muy útil a utilizar con **for** que nos permitirá crear una lista de número en función del o los parámetros que le pasemos. Puede usarse para generar un número y compararlo con otro y actuar de una determinada forma o compararlo con un parámetro pasado al *script* etc… se trata del comando **seq** y su sintaxis es *seq(x [y z])*

- Si pasamos un solo parámetro a *seq* (**for i in `seq 10`**) se generará una lista de números que comenzará con *1* y terminará en el número indicado, *10* en nuestro caso.
- Si pasamos dos parámetros a *seq* (**for i in `seq 5 10`**) se generará una lista de números que comenzará en *5* y terminará en el *10* para nuestro ejemplo. Esto es útil si queremos ir creando un sumatorio de *$i* con respecto al valor anterior.
- Por último cabe la posibilidad de pasar tres parámetros como parámetros tal que así : **for i in `seq 2 4 14`** . El primer parámetro (*2*) indica el número por el que se comienza, el segundo parámetro indica el incremento (*4*) y el tercer y último parámetro indica el número final a mostrar. Nuestro ejemplo mostraría la siguiente columna de números (aquí la representamos en fila): *2 6 10 14*

Nota: En estos ejemplos hemos usado la variable **i** pero podemos pasar cualquier palabra para usar luego como variable.

while

While es otro tipo de bucle que se ejecuta mientras la condición que le pasemos sea cierta. Es útil para comprobar la veracidad de reiteradas condiciones. Su forma básica es la siguiente:

```
while [ condición ]
do
          comandos
done
```

Un ejemplo del bucle *while* en el que se ejecutará una acción mientras **n** sea menor o igual a 10:

```
n=1
while [ "$n" -le 10 ]
do
          echo "$n"
          ((n++))
done
```

until

Para terminar con los bucles comentar **until**, que es igual que *while* pero al contrario, es decir, se ejecuta hasta que la condición sea verdadera. Su sintaxis es igual a la de *while*.

En este caso, a diferencia de *while* vamos a ejecutar una acción hasta que **n** sea mayor o igual a 11. El resultado es el mismo al del ejemplo anterior con *while* pero se interpreta al contrario:

```
n=1
until [ "$n" -ge 11 ]
do
        echo "$n"
        ((n++))
done
```

Existen tres comandos que nos pueden resultar útiles cuando trabajamos con bucles y estos son: *break*, *continue* y *exit*.

- **break** : Termina el bucle. Si existe un bucle dentro de otro, *break* provoca que salga del bucle desde el que fue llamado pero no del bucle padre de este.
- **continue** : Salta o continua con la siguiente iteracción del bucle en el momento en el que es llamado, es decir, no termina el bucle.
- **exit** : Termina la ejecución del *script*. Suele usarse con el valor 0 o 1 como parámetro para devolver un código de retorno del propio *script* por si tuviese que ser evaluada su ejecución.

select

El bucle **select** nos permite mostrar un menú de forma fácil y poder interactuar con él. Es común el uso de *select* junto a *case* de manera que se nos presente un menú y en función de la opción elegida hagamos coincidir esa opción (patrón) con uno de las posibilidades de case y ejecutar una acción. Con el siguiente ejemplo veremos esto fácilmente:

```
select i in listado home backup quit
do
        case $i in
               listado) ls -lt /opt/archivos;;
```

```
                home) cd ~/; echo "Moviéndonos al directorio home...";;
                backup) tar -cpvzf ~/backup.tar.gz /opt/archivos/*;echo
"Realizando backup de /opt/archivos...";;
                quit) echo "Saliendo del programa...";exit 0;;
        esac
done
```

Las funciones

Las funciones son muy útiles en la creación de *shell script* debido a que pueden ahorrarnos muchas líneas de código repetidas a lo largo del *script*. Podría definirse función como una parte del *script* que realiza una *subtarea* específica y que puede ser invocada por su nombre desde otra parte del *script*. La condición para usar una función es que siempre preceda a la palabra que lo llama. Esto hace que en numerosas ocasiones tras definir las variables de un *script*, se definan las funciones y a continuación se desarrolle el *script*, de manera que al ir leyendo linea a línea, cuando se llame a una función por su nombre el *script* se desplazará hacía la parte superior para llamar a la función, realizará las acciones pertinentes y volverá a la línea donde fue llamada con el valor de retorno. Vamos a ver esto con un ejemplo. Vamos a llamar una función a la que además se le va a pasar un argumento, esto no es necesario pero si útil en muchos casos. Para crear un ejemplo sencillo bastaría con no pasarle ningún parámetro:

```
retorna_valor() {
       if [ -d "$directorio" ]
       then
                 echo "El directorio $directorio existe"
       else
                 echo "El directorio $directorio no existe"
       fi
}

echo "Vamos a comprobar si existen una serie de directorios"

for directorio in "/home/a.coslada" "/home/p.merino" "/home/r.garcia"
do
          retorna_valor "$directorio"
done
```

Correo electrónico

A diferencia de otros sistemás, en *Linux* el correo electrónico es uno de los servicios de red más importantes. *Linux* hace uso del correo electrónico para muchas de sus tareas. Además de estar estrechamente ligado a las cuentas de los usuarios, el correo en *Linux*

es utilizado por ejemplo para informar de sus actividades programadas con *cron*, o enviar notificaciones sobre operaciones realizadas por *shell scripts*, entre otras. El servidor de correo contiene los mensajes entrantes de cada usuario, normalmente en */var/spool/mail*, aunque como siempre, esto pueda variar en función de la distribución que estemos usando. Deberemos de tener un conocimiento básico sobre su funcionamiento, configuración y posibilidades (crear *alias* de correo, reenviar correos, configurar el correo en aplicaciones clientes, etc…).

Existen varios protocolos para administrar el correo electrónico, siendo el más común **SMTP** (*Simple Mail Transfer Protocol, Protocolo simple de transferencia de correo*). En la mayoría de ocasiones se utiliza *SMTP* como sistema de entrega de correo, por estar diseñado como sistema *push* (el sistema emisor inicia la transferencia). No obstante en la etapa final del envío del correo se utilizan protocolos de tipo *pull* (el sistema receptor inicia la transferencia) como **POP** (*Post Office Protocol, Protocolo de oficina de correos*) o **IMAP** (*Internet Message Access Protocol, Protocolo de acceso a mensajes de Internet*). Estos protocolos son útiles sobre todo cuando el sistema receptor es el equipo de un usuario, el cual suele permanecer apagado en muchas ocasiones. La diferencia básica entre ambos protocolos es que con *POP* descargaremos los mensajes a nuestro equipo y con *IMAP* podremos leerlos desde el propio servidor.

Funcionamiento del correo electrónico

El método tradicional en *Linux* para enviar correo, es hacer que los programás locales se comuniquen con el servidor de correos local, quién a través del protocolo *SMTP* reenviará el correo hacia el destino. Los servidores *SMTP* son conocidos también como **MTA**, *Agentes de Transferencias de Correos*. De esta manera en caso de que el servidor de envío *SMTP* se encuentre *offline*, el correo se pondrá en cola y será enviado a sus destinos una vez que este vuelva a estar operativo. La mayoría de los clientes de correo (**MUA**, *Agentes de usuario de correo*) ofrecen la opción de contactar directamente con un servidor *SMTP* remoto al enviar el correo.

Un servidor *SMTP* puede hacer tanto de servidor (es decir recibe el correo de otro sistema) como de cliente (envía un correo a otro sistema).

El protocolo *SMTP* permite que un mensaje pase por un número arbitrario de ordenadores. Un usuario puede redactar un *email* que será enviado a su servidor *SMTP* local. Este servidor buscará a través de los servidores *DNS* el destino y una vez localizado se procederá a la entrega.

Cada servidor de la cadena de transmisión alterará el correo añadiendo un encabezado que proporcionará información sobre la ruta que ha tomado el correo. Los *spammers* y similares, falsifican los encabezados del correo complicando enormemente el análisis de seguimiento.

Configurar el correo electrónico

Aviso: Esta sección queda fuera del alcance del examen *LPIC-1*, aún así se cree conveniente añadir algunos comentarios o parámetros de configuración que nos permitan entendernos con un servidor de correo local, bien para enviar y recibir correos a usuarios de la misma máquina o red, o incluso hacer *relay* a un proveedor como *gmail*, de manera que podamos enviar mensajes al exterior a través de una cuenta de correos existente en un proveedor externo.

En un sistema de correo *Linux* totalmente funcional, probablemente se incluyan otros software. Lo normal es tener un servidor de correo saliente (*push*) y uno de correo entrante (*pull*), además de utilidades que nos ayuden a reorganizar el correo, a filtrarlo, a detener correo *spam*, etc…

Existen diferentes alternativas para servidores de correos de tipo *push*. Algunos de estos con formatos de configuración más complejos, mientras que otros quizás estén más orientado a proporcionar seguridad, o una configuración rápida y efectiva. Los servidores de correo más populares en *Linux* son:

- **Sendmail**: Durante muchos años fue el servidor de correo dominante en Internet. Es muy potente pero también difícil de configurar. Sus archivos de configuración pueden parecernos algo arcaicos.
- **Postfix**: Diseñado como sustituto modular de *sendmail* ya que utiliza varios programás, controlando cada cual su pequeña tarea específica. Esto lo hace más seguro y además suele ser más fácil de configurar que *sendmail*. Es el servidor de correo por defecto en muchas distribuciones Linux. Será este el que usemos en los ejemplos de configuración que veremos en breves.
- **Exim**: Al igual que *sendmail*, es un servidor monolítico pero su formato de configuración es más sencillo por lo tanto más fácil de configurar. Algunas distribuciones de Linux lo emplean como servidor de correo por defecto.
- **qmail**: Es un servidor modular (como *postfix*) con la seguridad como principal objetivo. Es más sencillo de configurar que *sendmail*, pero debido a su licencia, la cual complica su distribución en *Linux*, no se ofrece como servidor de correo por defecto.

Podemos averiguar de diferentes formás que servidor de correo instala nuestra distribución por defecto como por ejemplo mediante el comando **ps** o incluso mirando dentro de los directorios */usr/bin* o */usr/sbin*. Para ambos podemos aplicar filtros con **grep** como ya vimos en Capítulos anteriores. Por motivos de compatibilidad la mayoría de servidores de correo *Linux* incluyen un programa *sendmail*, por lo que este podría servir de patrón en nuestra búsqueda.

De igual modo que con el servidor *push*, dependiendo del modo en el que vayamos a configurar nuestra forma de recibir correos podremos optar por algunas opciones entre las que destacamos:

- **Servidores pull de correo**: Los dos protocolos más populares para este tipo de sistema son *POP* e *IMAP* y ya vimos la diferencia básica entre ellos. Algunas de las herramientas más destacadas con las que podremos controlar la lectura del correo en nuestro *Linux* son: *cyrus*, *dovecot* y *courier* (los tres tienen su equivalente tanto para *POP* como *IMAP*).
- **Fetchmail**: Si dependemos de un *ISP* externo para la distribución de correo, necesitaremos un programa que obtenga el correo utilizando *POP* o *IMAP*, y que después lo inserte en una cola de correo *SMTP*. Esta es la función principal de *Fetchmail*.
- **Lectores de correos**: Son el último eslabón de la cadena de correo. Existen muchos y suelen venir incluidos con la aplicación cliente. Por nombrar algunos: *Evolution*, *KMail*, *Thunderbird* o *mutt*. Mientra los tres primeros son para *GUI*, *mutt* puede ser útil para leer el correo desde la línea de comando, posee una interfaz gráfica para la *shell*, es decir está basado en texto.

Alguna de las herramientas de las que podemos hacer uso para completar nuestro sistema de correos podrían ser:

- **d-push**: Útil si queremos recuperar mediante *push* el correo en dispositivos móviles. Requiere de un servidor web como *apache* o *nginx* y de soporte para *PHP*.
- **procmail**: Funciona como **MDA** (*Mail Delivery Agent, Agente de entrega de correo*). Sirve para filtrar y clasificar el correo de entrada. Permite a los usuarios definir reglas que filtren los *email* recibidos y realizar una serie de funciones como colocarlos en determinadas carpetas, clasificarlos, reenviarlos, etc…
- **spamássassin**: Identifica el correo sospechoso o másivo añadiendo cabeceras a los mensajes de modo que pueda ser filtrados por el cliente de correo electrónico

En las dos próximás secciones abarcaremos una configuración básica para en reenvío de correo electrónico dentro de la red local o el propio sistema (algo que en muchas distribuciones viene por defecto) y como reenviar el correo a un proveedor externo. No vamos a abarcar la configuración de un servidor de correo totalmente funciona pues son muchas y diferentes configuraciones las que entran en juegos, pero para que quede una idea, podremos montar una infraestructura con:

- **Postfix** quién hará de servidor *SMTP* enviando y recibiendo el correo
- **dovecot** para recuperar el correo *IMAP* en aplicaciones clientes de los usuarios
- **spamássassin** para controlar el *spam*
- **procmail** para filtrar y clasificar el correo
- **d-push** con el que recuperar los *email* en clientes móviles

Configuración básica local

Ahora que tenemos claro los conceptos vamos a proceder a configurar un servidor de correo de manera que el sistema pueda enviar y recibir *email* entre cuentas de usuarios, algo que en muchos sistemás viene configurado por defecto. Para ello vamos a usar *Postfix* como *SMTP* sobre la distribución *Linux Mint 17.1* (derivado de *Debian*)

Nota: En distribuciones como *CentOS* o *Fedora* (derivados de *Red Hat*) este sistema viene ya preconfigurado para un funcionamiento a nivel local.

En caso de que ya tuviéramos *postfix* instalado (de lo contrario deberemos de buscarlo por ejemplo con *aptitude search postfix*, en caso de distribuciones derivadas de *Debian*, y posteriormente instalarlo con *apt-get*) pero no configurado haremos:

```
$ sudo dpkg-reconfigure postfix
```

Esto volverá a ejecutar el programa de configuración posterior a la instalación de *postfix*. En el se realizan una serie de preguntas a las que deberemos de responder y con estos datos se creará el archivo de configuración */etc/postfix/main.cf* que en caso de tenerlo podemos ahorrarnos el pasar por esta interfaz de menú y editar el archivo directamente. Importante recargar el servicio tras su edición.

Otra manera de efectuar cambios sobre *main.cf* es a través de la herramienta **postconf** (herramienta de configuración de *postfix*) la cual además también nos sirve para ver el valor de los parámetros de configuración o modificarlos. Por ejemplo, para ver el valor de una directiva de *main.cf* bastará con pasársela como argumento a *postconf*, Si queremos cambiarle su valor, lo haremos así '*postconf* ***-e*** *<directiva=valor>*'. Si queremos ver el valor que trae por defecto cada directiva de *main.cf* bastará con pasarle el parámetro **-d**, o **-n** para listar todas aquellas que tenemos ya configuradas. Con **-p** listaremos todas las directivas de configuración de *postfix*, tanto aquellas a las que no se les ha cambiado su valor como las que si se han modificado. Sobre-escribiremos el valor de una directiva usando **-o** como opción. Existe otro archivo igual de importante (o incluso más) que configura el servidor *postfix* y es *máster.cf*. Para mostrar su contenido usaremos **-M**,

Las preguntas a contestar (y sus respuestas para mi configuración personal básica) son:

- Configuración base inicial: **solo correo local**
- FQDN: En mi caso no dispongo de dominio así que '***freeser***' a secas será el nombre de mi host.
- Alias para redireccionar el correo de *postmáster* y *root*: Yo lo redirecciono todo a mi usuario '***nebul4ck***'
- Lista de dominios separados por comás para los que actuará nuestro servidor: ***<ok>***. Yo la dejo tal cual, básicamente con el nombre de mi host y si tuviese dominio pues lo pondría igualmente.
- Sincronizar el correo de forma síncrona: ***<No>***. Esto ralentiza el procesado del servicio de correo. Si tenemos un sistema de archivos con *journal* no tiene por que hacernos falta.
- Configuración del bloque de red: Para enviar y recibir en equipos de la red sería algo parecido a *192.168.1.0/24* (podemos eliminar la IP *IPv6* si no la vamos a utilizar). Yo dejo la que viene por defecto que es para gestionar solo el correo local (***127.0.0.1/8***)
- Tamaño máximo del buzón de correo: *0* ilimitado, ***51200000*** el valor dado por el desarrollador. Esto no tiene que ver con la cuota de correo para cada usuario, para eso deberemos de usar el sistema de *Quota Disk* que ya hemos estudiado.

- Elegir un caracter que defina la extensión de dirección local: '***+***' por defecto. Yo lo dejo en blanco.
- Concretar el protocolo de IP a utilizar: ***IPv4*** en mi caso

Estas preguntas y respuestas lo que hacen es añadir/modificar las siguientes líneas del archivo *main.cf*, con las que tendremos un sistema igualmente funcional en caso de que queramos hacerlo editando el archivo directamente:

```
relayhost =
myhostname = freeser
alias_maps = hash:/etc/aliases
alias_database = hash:/etc/aliases
mydestination = nombredetuequipo
mynetworks = 127.0.0.1/8
mailbox_size_limit = 51200000
inet_interfaces = loopback-only
inet_protocols = ipv4
```

Una vez tenemos la configuración base para *postfix* vamos a crear un alias para *root*, recordar que en los pasos anteriores creamos un alias para nuestro usuario de manera que todo el correo de *postmáster* y *root* se redirigiera a nuestro usuario, pero al parecer, según el mensaje obtenido tras la configuración: **WARNING: /etc/aliases exists, but does not have a root alias**. *root* no tiene alias por lo que deberemos de redirigir todo el correo (incluido el de *postmáster*) hacia *root* y entonces el de *root* será redirigido a nosotros. Para ello editamos el archivo */etc/aliases*. Bastará con escribir '*root: nuestrousuario*'

Nota: En la sección "*Administrando el correo electrónico*" veremos como crear alias y redirigir el correo.

Ahora con el archivo */etc/aliases* editado, necesitaremos ejecutar el comando **newaliases** para crear un binario que pueda ser leído por el sistema.

Con esta configuración ya podremos enviar correos entre los usuarios del sistema local o red si hemos configurado el bloque de red de la siguiente manera:

```
$ mail -s "Asunto" usuariodelsistema@nombredehost
```

Nota: Si el *prompt* devolviera el típico "*mail: command not found*" deberemos de instalar el paquete *mailutils* que es donde suele encontrarse la utilidad *mail*.

Tras introducir esta línea nos aparece '**cc:**' esto es por si queremos poner en copia a alguien, si no queremos pulsamos *<Enter>* y a continuación el *prompt* esperará que introduzcamos texto. Una vez terminado pulsamos *Control+D* y el *email* será enviado.

En la sección "*Administrando el correo*" veremos más opciones de **mail** y aprenderemos a gestionar nuestro correo.

Configuración para reenviar correo a un proveedor como gmail

En la sección anterior hemos aprendido a configurar *postfix* para enviar correo desde una cuenta de nuestro sistema a otra cuenta del mismo sistema u otro dentro de nuestra red local. Lo que haremos ahora será configurar un *relay* en *postfix* de manera que podamos reenviar el correo desde nuestra cuenta del sistema a una cuenta propia en un servidor de correo externo como podría ser *gmail* (para nuestro ejemplo) y que sea esta la que envíe el correo al destinatario. Así podremos salir a Internet fácilmente a través de un proveedor externo.

Para pasar de la configuración de *correo local* a hacer *relay* contra por ejemplo *gmail* las líneas a tener en cuenta y que deberíamos de agregar en caso de no existir en *main.cf* serían las siguientes:

- relayhost = **[smtp.gmail.com]:587**
- smtp_sasl_auth_enable = **yes**
- smtp_sasl_password_maps = **hash:/etc/postfix/sasl_passwd**
- smtp_sasl_security_options = **noanonymous**
- smtp_tls_CAfile = **/etc/postfix/cacert.pem**
- default_transport = **smtp**
- relay_transport = **relay**

Las modificaciones han sido claras. Estamos indicando que queremos hacer *relay* contra el servidor *SMTP* de *gmail* por el puerto 587, hemos habilitado el mecanismo de autenticación *sasl* indicando que archivo es el que contiene las credenciales de la cuenta que vamos a usar para hacer el *relay* con la opción '*noanonymous*', se ha referenciado el *PATH* para el certificado de nuestra máquina y por último se han modificado las directivas *default_transpor* y *relay_transport*, ambas con el valor '*error*' en nuestra anterior configuración "s*olo correo local*".

Por lo general suelen existir las siguientes líneas en *main.cf*, de lo contrario (como por ejemplo en *CentOS* y *Fedora*) deberemos de añadirla nosotros para habilitar la capa *SSL/TLS*:

- smtpd_use_tls = **yes**
- smtpd_tls_session_cache_database = **btree:${data_directory}/smtpd_scache**
- smtp_tls_session_cache_database = **btree:${data_directory}/smtp_scache**
- smtp_tls_policy_maps = **hash:/etc/postfix/tls_policy**

Volviendo a las líneas anteriores, para crear el archivo de credenciales '*sasl_passwd*' y hacer que *postfix* lo reconozca haremos lo siguiente:

1. Crear el archivo con las credenciales:

```
$ sudo vi /etc/postfix/sasl_passw
```

2. Añadimos la línea a *sasl_passw* con los datos de la cuenta:

```
[smtp.gmail.com]:587 USERNAME@gmail.com:PASSWORD
```

3. Cambiamos el modo de acceso del archivo a solo lectura para *root*:

```
$ sudo chmod 400 /etc/postfix/sasl_passwd
```

4. Indicamos a *postfix* que utilice el archivo de credenciales:

```
$ sudo postmap /etc/postfix/sasl_passwd
```

Con el archivo de credenciales creado y reconocido por *postfix*, pasamos a generar un certificado. En *Linux Mint* contamos con bastantes certificados preinstalados. En concreto nosotros usaremos uno de los que son válidos para *gmail* '*Thawte_Premium_Server_CA.pm*':

```
$ ls -lt /etc/ssl/certs |grep Thawte
 Thawte_Premium_Server_CA.pem -> /usr/share/ca-
certificates/mozilla/Thawte_Premium_Server_CA.crt
```

Por lo que podremos generarlo de la siguiente manera:

```
$ cat /etc/ssl/certs/Thawte_Premium_Server_CA.pem | sudo tee -a
/etc/postfix/cacert.pem
```

Importante: Igualmente podríamos haber creado nuestro propio certificado, algo que no será útil por ejemplo en *CentOS* (entre otras) donde no contamos con estos certificados preinstalados. Para crear nuestro propio certificado haremos:

```
# cd /etc/pki/tls/certs
# make <nombre_host>.pem
```

Se nos hará una serie de preguntas a las que deberemos de responder, como el *País*, la *Ciudad* o *Estado*, *email*, etc… luego ya podremos generar nuestro certificado:

```
# cp /etc/pki/tls/certs/<nombre_host>.pem /etc/postfix/cacert.pem
```

Nota: Si hemos tenido que crear nuestro propio certificado normalmente en estas distribuciones tendremos que crear un archivo */etc/postfix/tls_policy* y seguir los mismos pasos realizados con el archivo */etc/postfix/sasl_passwd* solo que el contenido a añadir será:

[smtp.gmail.com]:587 encrypt

Ahora, como ya sabemos, cada vez que modificamos *main.cf* toca recargar el archivo de configuración o incluso reiniciar el servicio, así que:

```
$ sudo service postfix reload | restart
```

Hasta aquí debe de estar todo listo para poder hacer el reenvío. Antes comentamos alguna forma de enviar un correo desde la línea de comandos con *mail*, aquí mostramos esta otra:

```
$ echo "Este es mi tercer email" | mail -s "Mi tercer"
nebul4ck@gmail.com
```

Para ver lo que ha ocurrido como siempre visitaremos nuestro mejor amigo */var/log/mail.log* o */var/log/maillog*

Podremos encontrar mucha información sobre como configurar el servidor *postfix* en */usr/share/doc/postfix/*

Los servidores de correo proporcionan una amplia variedad de opciones avanzadas para priorizar el envío de correos, aceptar mensajes por la línea de comandos, borrar mensajes específicos de la cola, depurar conexiones de correos y otras muchas opciones que no se han abarcado en esta sección por no extender en demásía el Capítulo, además de no ser el propósito de esta guía de estudios para LPIC-1.

Proteger el servidor

En esta sección no se desarrollarán métodos sobre como securizar nuestro servidor de correos, pero si alguna información importante a tener en cuenta y herramientas que pueden hacernos la vida menos complicada en lo referente a nuestro servidor de correos. Por ejemplo algo con lo que estar familiarizado deben de ser los *bugs* del propio servidor de correos o herramientas que usemos para dar de alguna forma asistencia o complementación a nuestro servicio de correo. Los *bugs* pueden poner en peligro el sistema permitiendo a alguien obtener acceso y enviar correos, o como vimos en el Capítulo anterior sobre el diagnóstico de la red, conectándose al puerto 25 (*SMTP*) mediante un cliente *Telnet* y escribiendo comandos *SMTP* para aprovechar la vulnerabilidad. Es por esto que muchas distribuciones de *Linux* reducen el acceso exclusivamente al ordenador local. Es muy importante instalar los parches que puedan surgir a raíz de *bugs* encontrados y actualizar el sistema constantemente.

Otro aspecto a tener en cuenta es la configuración. Deberemos de ser conocedores de que parámetros son los que estamos configurando y que valor le estamos dando. Existen configuraciones defectuosas que permiten a *spammers* utilizar su servidor de correos para enviar *email* másivos por lo que no tardarán en bloquear nuestra IP o añadirla a listas negras (*blacklist*). A este tipo de configuraciones se las llama **open relay** (*transmisiones abiertas*). La mayoría de distribuciones *Linux* configuran el servidor de manera que no permita el *open relay* por lo que de cierta manera podemos andar tranquilos. No obstante deberemos de revisar de que nosotros tenemos la configuración apropiada. Existen varias maneras o sitios en Internet en los que podremos comprobar esto como por ejemplo desde la web *http://www.spamhelp.org/shopenrelay*, o bien

mediante conexión *Telnet*, conectando a nuestro servidor *SMTP* por su *IP* y puerto 25 y enviando comandos *SMTP*.

Algunas herramientas útiles de cara a la seguridad en sistemás totalmente funcionales de correo electrónico pueden ser: **spamássassin** (herramienta open source que utiliza una extensa serie de test o reglas tanto a nivel local como de red para identificar firmás de spam en los correos electrónicos.), **smapd** o **greylist** (ambas orientadas a reducir el correo *spam*).

Administrar el correo electrónico

Hasta aquí hemos visto algunos aspectos básicos sobre la configuración de un servidor *postfix SMTP*, de manera que podamos enviar y recibir correos en nuestra máquina local o aquellas que pertenecen a nuestra red, y además como reenviar correo desde nuestra cuenta del sistema hacia una cuenta en un proveedor externo (en nuestro caso *gmail*) para que sea esta quien entregue el correo del sistema a través de Internet. En las siguientes secciones aprenderemos un uso básico sobre la administración del correo de manera local, como el envío y lectura del correo a través de la línea de comandos, ver la cola de correo y eliminar alguno o todos los correos existente en ella, además de como redirigir el correo entre usuarios de una máquina mediante *alias* de correo.

Enviar y recibir correo desde la línea de comandos

Linux dispone de una amplia gama de aplicaciones de correo cliente o *MUA* (*Mail User Agent*) con las que poder administrar una cuenta. Alguna de estas pudiesen ser **Geary**, sistema cliente básico para principiantes, **Thunderbird**, **Evolution** y **KMail**, todas ellas disponibles para *GUI* y orientadas a usuarios más avanzados, **Zimbra**, para acceder vía web. Además *Zimbra* es un paquete *Open Source* con el que podremos crear y gestionar un servidor funcional completo de correos (*SMTP*, *POP*, *IMAP*…) que incluye otras funcionalidades como la *Agenda* y el *Calendario*. Otro cliente con el que podremos enviar y recibir correo pero esta vez desde la línea de comandos es **mutt**. *Mutt* es un *MUA* que soporta protocolos como *IMAP* y *POP* para la lectura del correo, así como *SMTP* para la transferencia. Es bastante configurable y su interfaz está basada totalmente en texto. No obstante en esta sección nos vamos a centrar en la ya mencionada herramienta **mail** con la que podremos envía correo desde la línea de comandos e incluso leerlos.

La sintaxis básica de *mail* es la siguiente:

- Para enviar un correo:

mai [-v] [-s “asunto”] [-c “dirección con-copia” | -b “dirección con copia oculta”] dirección destinatario

Nota: La opción **-c** no trabaja en todas las versiones de *mail*

Anteriormente ya vimos dos formás de enviar un *email*, bien escribiendo la línea del comando *mail* la cual nos desplegará un *prompt* donde introducir texto y finalizar con *Control+D* o bien mediante el comando *echo* redireccionado a *mail*. Otras dos formás de enviar un correo serían:

1. Enviar el contenido de un archivo como correo:

```
mail -s "Lista de invitados para la fiesta!!" <
/home/party/listainvitados.txt invitados@4sysadmins.es
```

2. Es igual a la opción básica pero en vez de usar Control+D para finalizar, utilizamos una palabra:

```
mail -s "Tareas por hacer" nebul4ck@4sysadmins.es << EOF
Aqui escribiremos todo lo que queramos
El correo terminará cuando repitamos la palabra con
la que abrimos el contenido de este email
EOF
```

Nota: Esta última forma es muy útil en correos enviados desde *shell scripts*.

- Para leer un correo:

mail [-v] [-f "/var/mail/<usuario>" | -u <usuario>]

Con **-f** indicaremos el nombre del archivo de correo de un determinado usuario o bien con **-u** pasamos directamente el nombre de ese usuario. Otras formás de leer el correo es desplegando el contenido del archivo de correo de un usuario mediante un paginador de archivo o *cat*, o bien utilizando *mail* de forma interactiva lo cual conseguiremos llamando a *mail* sin parámetros.

Cuando usamos *mail* de forma interactiva lo que hace es desplegar una serie de líneas de resúmenes de mensajes (un resumen de mensaje por línea) con el siguiente formato:

<nº de mensaje> <remitente> <fecha> <nº de líneas>/<nº de bytes> <Asunto>

Deberemos de prestar atención al número de mensaje pues es el que deberemos de introducir para desplegar su contenido. Posteriormente podremos eliminar el mensaje con **d** o responder a este con **r**.

Comprobar la cola de correo

El programa **mailq** es la principal herramienta para administrar la cola de correo. Originalmente este programa formaba parte del paquete *sendmail*, es por ello que usar

sendmail -bp es equivalente a *mailq*. El comando básico (sin opciones) muestra el contenido de la cola de correo de todos los sistemás.

Podremos vaciar una cola de correos con **sendmail -q**, **postqueue** (*postfix*) o **runq** (*exim*). Además en *postfix* contamos con la utilidad **postsuper** con la que podremos eliminar toda la cola de correos con '**-d** ALL' o un solo correo con '**-d** <*nº de correo*>'

Redirigir el correo

Por regla general los servidores de correo mantienen una cuenta propietaria que en la mayoría de los casos se nombra *postmáster* a la que se le es enviado el correo y cuyo contenido solo debería de ser leído por el administrador del sistema (*root*). Existe una manera con la que el correo es direccionado o redirigido a otro usuario con el fin de que este pueda leerlo. Para hacer posible esto se cuenta con el archivo **aliases** que suele encontrarse bajo */etc/* o bajo */etc/mail* y que posee un formato sencillo pero eficaz.

Si abriésemos este archivo veríamos una lista de líneas con un formato parecido al siguiente:

```
# Basic system aliases -- these MUST be present.
mailer-daemon: postmáster
postmáster: root
# General redirections for pseudo accounts.
bin: root
daemon: root
adm: root
...
```

Este sería el caso más simple en el que se nombra a *postmáster* como *mailer-daemon* y seguidamente se crea el primer *alias* en el que el correo de *postmáster* será redirigido a *root* (*postmáster: root*). Las siguientes líneas (y algunas más que faltan) por lo general lo que hacen es que el correo enviado a las cuentas de servicios o programás del sistema sea redirigido a *root*, de manera que el administrador cuente con la información de todos ellos. Al final de de este archivo podríamos añadir la siguiente línea:

```
root: nuestrousuario
```

con lo que estaríamos diciendo que una vez que todo el correo del sistema (servicios y programás) haya sido enviado a *root*, este sea su vez reenviado a nuestro usuario, con el fin de mantener un control de esta información sin llegar a ser *superusuario* o poseer los permisos de este.

Además de usar un nombre de usuario del sistema al que redirigir el correo, podremos utilizar un comando por el que canalizar los mensajes (|), el nombre de un archivo en el que se encuentran añadidos una serie de direcciones o incluso una cuenta de correo como *nebul4ck@4sysadmins.es*.

En la mayoría de servidores de correos es un requisito el compilar /etc/aliases a un archivo binario de manera que pueda ser procesado más rápidamente. Para esto usaremos el comando ya visto *newaliases*

Otra manera de redirigir el correo es hacerlo a nivel de usuario, es decir, editando el archivo *~/.forward* del directorio principal del usuario de manera que el correo que es enviado a este, sea reenviado a otra cuenta. Para hacer esto basta con editar el archivo con tan solamente el nombre de usuario al que queremos redirigir el correo o bien una cuenta de correo electrónico completa. Este sistema tiene varias ventajas como por ejemplo, el administrador del sistema podría crear por ejemplo una cuenta '*correos*' y avisar a todos los usuarios del sistema que hagan el reenvío de su correo a esta cuenta, de forma que el administrador pueda tener el correo de los usuarios de una forma controlada, o bien los usuarios podrán reenviar el correo de varias de sus cuentas a una sola cuenta de usuario o correo sin tener que molestar al administrador.

Administrar datos con SQL

En las siguientes secciones hablaremos sobre el *Lenguaje de consulta estructurado* o **SQL** (*Structured Query Languaje*) que como su nombre indica es utilizado para obtener datos de una base de datos. Para comenzar a utilizar *SQL* deberemos de disponer de algún paquete de software que lo implemente. En *Linux* disponemos de algunos, siendo los más habituales:

- **MySQL**: Este paquete nos ofrece un servidor de *BBDD*, un cliente para conectar con estas y herramientas de desarrollo. Es un software en propiedad de *Oracle* el cual se distribuye bajo *GPL*. La mayoría de las distribuciones de *Linux* incluyen a *MySQL* en sus repositorios oficiales.
- **PostgreSQL**: Disponible bajo licencia *BSD* y es una evolución del software *Ingres* inicial. El nombre de *PostgreSQL* es la compresión de *Post-Ingres SQL*. Al igual que *MySQL*, se encuentra disponible en los repositorios oficiales de las distribuciones Linux.
- **SQLite**: Es una biblioteca que implementa *SQL*. No se trata de un gestor de *BBDD* independiente, más bien un procedimiento o medio que permite a un programa almacenar datos utilizando una interfaz *SQL* del propio programa. Si desea escribir un programa que acceda a *BBDD* pero instalar un software pesado como MySQL o PostgreSQL resulta excesivo, quizás *SQLite* sea su solución.

Fundamentos de SQL

Desde *LPI.org* se está planteando la posibilidad de prescindir del contenido sobre gestión de bases de datos y más concretamente de la herramienta *SQL* con la que

acceder a *BBDD* relacionales. Es por ello que nos hemos extendido un poco más en el software del correo electrónico y lo haremos de forma más breve con respecto a *SQL*.

SQL permite la gestión de varias *BBDD*, a su vez estás están formadas por una o más tablas construidas por filas (*tuplas*) y columnas (*atributos* o *campos*).

Las columnas de una base de datos contienen tipos de datos específicos, ya que no es lo mismo indicar un color (rojo) que un precio (1.99€). Usaremos los tipos de datos específicos para crear restricciones relacionadas, llamadas "*dominio*" o "*tipo de datos*".

Algunos de estos tipos de datos son:

- **INTEGER** (*INT*): Valor entero de cuatro bytes.
- **SMALLINT**: Valor entero de dos bytes
- **DECIMAL** o **NUMERIC**: Almacenamiento preciso de valores decimales
- **FLOAT**: Número de coma flotante
- **DOUBLE PRECISION**: Número de coma flotante almacenado con el doble de precisión que con el que lo hace *FLOAT*
- **DATETIME**: Una fecha y una hora
- **DATE**: Una fecha
- **TIME**: Una hora en formato *HH:MM:SS*
- **CHAR**: Uno o más caracteres
- **VARCHAR**: Un número variable de caracteres
- **ENUM**: Una lista enumerada de datos como *pequeño*, *mediano* y *grande*
- **SET**: Datos que pueden tener cero o más valores. Como para un yogurt de macedonia: *fresa*, *plátano*, *melocotón* y *pera*.

Los datos de una tabla no están ordenados, pero podremos devolver el contenido de una consulta realizada con *SQL* de forma ordenada y filtrada como veremos en breves.

Primeros pasos en una base de datos

Lo primero que debemos de hacer para aprender a utilizar *SQL* es tener acceso a una base de datos *SQL*. Nosotros en nuestro ejemplo vamos a utilizar el motor *MySQL*. Aquí vamos a suponer que tenemos ya instalado el paquete *mysql* en el que se incluye servidor, cliente y herramientas, y además lo tenemos en ejecución (esto se hace de la misma forma que el resto de servicios Linux).

Para iniciar el cliente *mysql* basta con llamar a *mysql* sin parámetro alguno. Esto nos introducirá en un *prompt* (*mysql>*) que al igual que *bash* esperará a que introduzcamos comandos. Los comandos *SQL* se suelen escribir en mayúsculas. No es obligación pero si una forma de simplificar o clarificar la lectura de los comandos que normalmente ocupan más de una línea. Una vez en el *prompt* de *mysql* lo primero que debemos de hacer es **mostrar las bases de datos existentes**. Lo normal es que no haya ninguna

creada, pero *mysql* muy amigo nuestro nos tiene una preparada llamada *test* con la que podremos iniciarnos.

```
mysql> SHOW DATABASES;
```

Es importante fijarnos en como terminamos la línea de comandos. El caracter punto y coma '**;**' es el que indica que hemos terminado la línea de comandos, por lo que mientras que este no sea indicado se entenderá que la línea de consulta sigue abierta.

Si queremos **crear una base de datos** utilizaremos el comando *CREATE DATABASE* seguido por el nombre de la base de datos:

```
mysql> CREATE DATABASE animales;
```

Ahora podremos empezar a **usar la tabla** con el comando *USE*:

```
mysql> USE animales;
```

Como acabamos de crear la base de datos no existe ninguna tabla. Por ello vamos a **crear una tabla** con cuatro columnas o campos que servirán para diferenciar unas filas o tuplas de otras que iremos almacenando posteriormente. Por ejemplo:

```
mysql> CREATE TABLE gatos (nombre VARCHAR(30), raza  VARCHAR(20),
fech_nac DATE, precio DECIMAL (4,2));
```

Con este comando crearemos la tabla '*gatos*' como hemos dicho de cuatro campos **| id | nombre | raza | fecha de nacimiento | precio |**, en la que los dos primeros campos usan un tipo de datos de caracter variable, es decir podrán tener un máximo de 30 y 20 caracteres respectivamente. Si hubiésemos utilizado *CHAR* tanto los nombres de los gatos como sus razas deberían de tener exactamente 30 y 20 caracteres respectivamente. Para la fecha de nacimiento usamos el tipo *DATE* y algo a tener en cuenta en el campo '*precio*' es que podríamos haber usado el tipo *FLOAT* pero sin duda para un campo en el que vamos a definir algo tan importante y exacto como es un precio, es preferible utilizar el tipo DECIMAL (más exacto que FLOAT) con el que se indica los dígitos que irán delante de la coma (4 en este caso) y cuantos decimales tras la coma (2 para nuestro ejemplo). En otros motores de bases de datos existe el tipo *MONEY* pero *MySQL* no lo soporta.

Antes de pasar a almacenar datos en la tabla vamos a ver como **mostrar las tablas** que forma la base de datos:

```
mysql> SHOW TABLES;
```

Una vez localizada que tablas forman una base de datos podremos **ver los campos que construyen a cada tabla**. Vamos a ver por ejemplo que campos (nombre, raza, fecha de nacimiento y precio en nuestro ejemplo) son la base de la tabla gatos.

```
mysql> DESCRIBE gatos;
```

Bueno ahora que hemos localizado nuestra base de datos, hemos creado una tabla y hemos mostrado que campos son los que finalmente hemos añadido, vamos a pasar a **almacenar información** dentro de estos. Para insertar una *tupla* en una tabla usaremos el comando *INSERT INTO* de la siguiente manera:

```
mysql> INSERT INTO gatos
-> values ('rigodón', 'europeo', '2009-02-28', 23,4);
```

Bueno ahora supongamos que hemos insertado un dato erróneo, por lo que vamos a **actualizar los datos** de la tabla con el valor correcto:

```
mysql> UPDATE gatos SET precio=12.99 WHERE nombre='rigodon';
```

Esto buscará la *tupla* que tiene como nombre *rigodon* y volverá a setear el valor de precio pero esta vez con el valor 12.99. Si existiesen más de una *tupla* con nombre *rigodon* deberemos de añadir más criterios de filtrado. Podemos conseguir esto con **AND** y **OR**. Por ejemplo, vamos a suponer que existen dos filas con el valor *rigodon* en el campo *nombre* pero solo una con la fecha de nacimiento *2009-02-28*, para localizar esta *tupla* usaremos la siguiente línea:

```
mysql> UPDATE gatos SET precio=12.99 WHERE nombre='rigodon' AND
fech_nac='2009-02-28';
```

Como últimos comandos a estudiar en esta sección y no por ello menos importantes son **DELETE** y **DROP** con los que podremos eliminar algunos o todos los datos de una tabla o eliminar la tabla competa respectivamente.

Si queremos eliminar todos los datos de una tabla haremos uso del comodín '*****':

```
mysql> DELETE * FROM gatos;
```

Podemos eliminar solo datos específicos así:

```
mysql> DELETE FROM gatos
-> WHERE raza='persa' OR precio > 21,99;
```

Nota: Esto eliminará toda las tuplas pertenecientes a gatos cuya raza es '*persa*' o tienen un precio superior a *21,99*

En ocasiones lo que querremos será borrar una tabla completa (*tuplas* y *campos*) ya porque esta no sirva o porque haya tantos datos erróneos o desfasados que ya no nos interesen. En este caso usaremos *DROP* de la siguiente manera:

```
mysq> DROP TABLE gatos;
```

Entre las numerosas posibilidades de manipulación de datos con las que cuenta *SQL* tenemos la de recuperar datos de distintas tablas y combinarlos, dando como salida una única tabla con los datos necesitados. Para lograr esto, las tablas a combinar deben de tener todas ellas un campo común que pueda ser utilizado para conectarlas y además cada tabla igualmente deberá de tener un campo cuyo valor identifique inequívocamente a cada *tupla*. A este campo se le llama **PK** (*Primary Key*, *Clave Primaria*).

Antes de nada debemos de conocer el comando **SELECT**. *SELECT* nos permite realizar consultas en una base de datos. Será el comando que nos ofrezca la posibilidad de recuperar los datos de las distintas tablas para posteriormente manipularlos o simplemente obtener la información deseada.

La sintaxis básica para *SELECT* es la siguiente:

SELECT campo(s) FROM tabla [WHERE condiciones] [ORDER BY campo]

Vamos a recuperar todos los datos de la tabla *gatos*:

```
mysql> SELECT * FROM gatos;
```

Podemos seleccionar uno o varios campos de la siguiente manera:

```
mysql> SELECT nombre, raza FROM gatos;
```

Y de la misma manera que ya hemos visto podemos filtrar la consulta con *WHERE* y ser más específicos con *AND*, *OR* o *NOT*:

```
mysql> SELECT * FROM gatos WHERE raza='persa' OR fech_nac>'2008-11-29'
AND NOT raza='europeo';
```

Podemos incluso ordenar la salida por un campo en orden ascendente (**ASC**) o descendente (**DESC**). Igualmente podemos sumar (**SUM**) todos los valores de un campo. Esto último es útil por ejemplo si queremos sumar el precio de todas las máscotas.

```
mysql> SELECT raza,precio FROM gatos WHERE precio>15.99 GROUP BY
precio ASC
```

Una vez que conocemos el uso básico de *SELECT* será suficiente para poder combinar la salida de por ejemplo dos tablas. Para combinar tablas podemos usar el mismo comando *SELECT* aunque se suele emplear **JOIN**. A continuación veremos un ejemplo, pero antes vamos a hacer un uso sencillo de otro de los comandos que nos ofrece *SQL* y es **ALTER TABLE**. Este comando nos permite modificar la estructura de una tabla ya existente, por ejemplo, añadir nuevas columnas o eliminar otras o bien, crear y borrar restricciones. Para ello se permite el uso de palabras reservadas como *ADD* (añade o modifica una columna), *DROP* (elimina una columna o restricciones sobre ella), *UNIQUE* (los datos de cada *tupla* de la columna serán únicos), *PRIMARY KEY* (esta

columna hará de clave primaria para cada *tupla*) y *CONSTRAINT* (añade o elimina una restricción) . ¿ Porque comentamos esto aquí ? – pues porque si hubiésemos seguido los ejemplos que hemos ido realizando, nos daríamos cuenta (en caso de que no hayamos eliminado ya la tabla o base de datos) que cuando creamos la tabla *gatos* no creamos el campo/columna 'id' el cual nos permitirá combinar tablas con *SELECT* o *JOIN*. La sintaxis de *ALTER* es la siguiente:

- Añadir una columna (para eliminarla usamos la misma sentencia pero cambiamos *ADD* por *DROP*):

ALTER TABLE <tabla> ADD [COLUMN] <nombre_columna> <tipo>

- Añadir una columna con restricciones:

ALTER TABLE <tabla> ADD [COLUMN] <nom_column> <tipo> RESTRICCIONES

- Añadir restricciones a una columna ya existente:

ALTER TABLE <tabla> ADD [COLUMN] RESTRICCIONES <columna>

Nota: Para eliminar una restricción usamos la sintaxis anterior pero cambiamos *ADD* por *DROP*

Antes de pasar al ejemplo que comentábamos sobre la combinación de tablas con *SELECT* y *JOIN* vamos a crear definitivamente nuestro campo *id* en la tabla *gatos* con las siguientes restricciones: el campo no podra ser nulo (*NOT NULL*) y nos valdrá como clave primaria (*PRIMARY KEY*). Será de tipo entero y con un máximo de cuatro dígitos de longitud:

```
mysql> ALTER TABLE gatos ADD COLUMN id INTEGER(4) NOT NULL PRIMARY
KEY;
```

Aviso: El ejemplo anterior solo ha servidor para mostrar como crear una columna con un nombre y tipo y dos restricciones pero en la práctica obtendremos un error diciendo que no podemos hacer el campo id PK. Esto es porque al ser NOT NULL se le ha asignado el valor 0 a todos y al ser repetido es imposible que sea una clave primaria. Por ello anularemos el PRIMARY KEY, le daremos un valor diferente a cada tupla y a continuación usaremos la siguiente sentencia:

```
mysq> ALTER TABLE gatos ADD PRIMARY KEY (id);
```

Ahora que ya tenemos nuestro campo *id* en la tabla gatos volvamos al ejemplo de combinación de tablas:

Ejemplo: Un veterinario nos pide que saquemos los números de contacto de todos los gatos mayores de cinco años que tienen implantado el chip de reconocimiento y que además estén vacunados.

Lo primero que haremos será localizar las tablas que tengan campos en común y posean el campo *PK*. Además de esto las tablas deberán de tener la información de si se encuentran vacunados, si existe un número de contacto y la fecha de nacimiento. Vamos a suponer que ya hemos dado con dos tablas que nos serán útiles.

- La tabla *gatos* tiene los siguientes campos: **id | nombre | raza | fech_nac | precio |**
- La tabla *identificación* está formada por: **num_chip | nombre | vacunado | contacto |**

Sabemos que las *PK* de ambas tablas son '**id**' y '**num_chip**' y el campo común es '**nombre**'. Para obtener la información que nos ha pedido el veterinario haremos lo siguiente:

```
mysql> SELECT gatos.nombre, gatos.fech_nac, identificacion.vacunado,
identificacion.contacto
-> FROM gatos, identificacion
-> WHERE gatos.nombre=identificacion.nombre AND gatos.fech_nac<'2010-
01-01' AND identificacion.vacunado='si';
```

Si queremos combinar de la misma forma pero usando *JOIN* solo existe la diferencia de que una tabla es especificada con *FROM* y la otra con *JOIN*:

```
mysql> SELECT gatos.nombre, gatos.fech_nac, identificacion.vacunado,
identificacion.contacto
-> FROM gatos
-> JOIN identificacion
-> WHERE gatos.nombre=identificacion.nombre AND gatos.fech_nac<'2010-
01-01' AND identificacion.vacunado='si';
```

Y hasta aquí todo lo relacionado con la administración de bases de datos. Ha sido un mini tutorial pero como ya indicamos al principio de la sección, este contenido puede quedar obsoleto con respecto a la certificación *LPI-1*

LPIC-1 Capítulo 10

EN ESTE CAPÍTULO SE ABARCAN LOS SIGUIENTES OBJETIVOS DEL EXAMEN:

- 110.1 : Realizar tareas administrativas de seguridad (3)
- 110.2 : Configurar la seguridad del host (3)
- 110.3 : Proteger los datos mediante técnicas de encriptación (3)

Proteger el sistema

Hoy en día la mayoría de los ordenadores se encuentran conectados a Internet. Estar conectado a Internet aporta innumerables ventajas a nuestra sociedad, es como salir al exterior pero sin realmente dar un paso de donde nos encontramos. Esto, sin pensarlo, nos parece incluso seguro, nos da la sensación de ser menos vulnerables pero realmente es todo lo contrario, estar conectado a Internet significa prácticamente exponer en cierta forma nuestro '*yo*' más íntimo, a través de fotografías, vídeos, conversaciones, datos bancarios, etc… es por ello que cada día debemos de ser más conscientes y dedicar más tiempo, esfuerzo o economía a protegernos del exterior.

En las siguientes secciones veremos algunos puntos básicos a tener en cuenta cuando utilizamos un dispositivo tanto con acceso a red como si nos limitamos a un uso local. Tener un '*agujero*' de seguridad en nuestro dispositivo, en el mejor de los casos significaría que solo ese dispositivo se vería comprometido pero en la mayoría de las veces, solamente se trata de una puerta de acceso al interior de nuestra red, red en la que podemos tener servidores con importantes servicios ejecutándose o numerosa información personal que de ninguna de las maneras queremos que sea accedida por terceras personas. Existen muchos puntos a tener en cuenta para ser realmente eficaces con respecto a la posibilidad de parar a un atacante pero este curso abarcará solamente aquellos que se encuentran al nivel exigido para aprobar la certificación y que con serán los siguientes:

Administrar la seguridad local

- Crear contraseñas fuertes contra ataques de fuerza bruta
- Control de acceso al sistema
- Límites y cuotas de usuarios
- Permisos sobre archivos: *nosuid*, *nodev*, *noexec*
- Habilitación de *hosts*
- Módulos *PAM*

Administrar la seguridad de la red

- Técnicas utilizadas en los ataques y contramedidas
- Controlar los puertos de red que se están utilizando
- Comprobar las conexiones de red existentes y los servicios que las provocan
- Utilizar un súper servidor con el que poder limitar el acceso de una forma más controlada
- Protección mediante Firewalls
- Sistemás de detección de intrusos (*IDS*)

Administrar la seguridad local

El uso de contraseñas

Quizás el primer paso para acceder a un sistema, independientemente de la técnica utilizada para localizar el sistema en cuestión y sus vulnerabilidades (de ser un ataque en red) como el uso de la ingeniería social, un keylogger o cualquier otro método para tomar el control de un sistema de forma local, es acceder a la cuenta de un usuario de manera que se obtenga un punto de partida para continuar con un ataque, ya sea al propio sistema o equipos conectados a la red. Un equipo sin contraseña o con una contraseña lo suficientemente débil como para ser averiguada en poco tiempo mediante fuerza bruta allanaría el terreno aportado este punto necesario para tomar el control. Es por ello que debemos de seguir unas pautas para generar una contraseña de cuenta lo suficientemente robusta como para ser vencida rápidamente por un ataque de fuerza bruta, además no basta con generarla una vez si no que lo conveniente sería ir mortificándola cada cierto tiempo de manera que si un atacante obtuvo nuestra contraseña, se encuentre con que somos como mínimo una victima poco perezosa, en este caso deberá de volver a intentar tener acceso al sistema. Este será nuestro primer pequeño muro a construir. Algunas de las pautas a seguir para evitar riesgo con respecto a las contraseñas serían:

- **Utilizar contraseñas seguras**: Podemos utilizar alguna técnica para generar nuestra contraseña por ejemplo, crear una combinación aleatoria de letras mayúsculas y minúsculas, dígitos y signos de puntuación. Una buena forma de empezar sería escribir una frase que tenga (o no) que ver con la cuenta o programa. Imaginemos que vamos a crear una contraseña para el correo electrónico, podríamos hacer algo así; "*cuando abro mi correo solo tengo spam*". Vamos a trabajar sobre esta frase, de momento vamos a seleccionar las dos primeras letras de cada palabra quedando algo así "*cuabmicosotesp*". Ahora vamos a cambiar una vocal si y otra no por un dígito tal que así "cu4bmic0sot3sp". Con esto la contraseña tomaría suficiente fuerza pero si queremos podemos separar la contraseña en dos partes mediante un guión bajo o

signo de exclamación “cu4bmic_0sot3sp” o “cu4bmic!0sot3sp”. Con esto hemos conseguido una contraseña lo suficientemente segura. Podríamos seguir implementando seguridad a nuestra contraseña como por ejemplo invertir el orden o ampliando el pajar. La expresión “ampliar el pajar” viene de buscar una aguja en un pajar, y no es más que aumentar la longitud de la contraseña mediante la repetición de caracteres, en nuestro caso podría quedar definitivamente así (inversión + ampliación) “0sot3sp!!!!!!!!cu4bmic”. Parece paranoico pero nuestro nivel de paranoia será (o debería serlo) directamente proporcional a la criticidad de la información que queremos ocultar.

Información: Existen programás como *John the Ripper* que podemos aplicar sobre nuestra propia base de datos de contraseñas encriptadas para localizar contraseñas pobres.

- **Cambiar la contraseña con frecuencia**: Sería interesante a la hora de crear una cuenta en el equipo hacerlo de manera que cada cierto tiempo el sistema nos obligue a modificar la contraseña. Podemos hacer esto directamente a la hora de crear la cuenta mediante *useradd* o una vez creada mediante la herramienta *chage* o *usermod*.
- **Utilizar contraseñas ocultas**: En el *Capítulo 7* ya se comentó la importancia de usar contraseñas *shadow* (*/etc/shadow*) en lugar del antiguo mecanismo que permitía que un atacante pudiese leer el archivo /etc/passwd y localizar la contraseña directamente. Hoy en día todas las distribuciones utilizan por defecto contraseñas *shadow*. No obstante podemos modificar esto con los comandos *pwconv* y pwunconv que implementan o no contraseñas *shadow* respectivamente.
- **Mantener las contraseñas en secreto**: Aunque parezca arcaico no hay nada más ‘*seguro*‘ que mantener la contraseña escrita en un papel al que acudir cuando la necesitamos, aunque realmente lo mejor es memorizarla, pero…
- **Utilizar protocolos seguros de acceso remoto**: Más adelante hablaremos sobre protocolos o métodos de conexión remota, pero adelantamos que lo mejor es conectar a través de ***ssh***, ya sea a *ftp*, escritorio o consola remota, o bien copiar archivos (*sftp*, *ssh -X*, *ssh*, y *scp* respectivamente). Esto encriptará nuestra contraseña. Deberemos de desactivar *Telnet*, *FTP* y otros protocolos que emplean contraseñas en texto plano.
- **Tener cuidado con la ingeniería social**: práctica que implica engañar a personas fingiendo ser por ejemplo un administrador de sistemás o cualquier otra persona de responsabilidad en la materia, para que estas le entreguen sus contraseñas. Este sería el método sencillo pues esto puede llevar a conversaciones suficientemente elaboradas por aquel que emplea la ingeniería social de modo que nos esté sacando información valiosa sin ni siquiera darnos cuenta. Es posible que no digamos en ningún momento nuestra contraseña pero quizás si le estemos ofreciendo el método de llegar a ella. Una práctica relacionada es el ***phishing*** (o pesca telemática) en la que un atacante crea un sitio web falso o envía un correo electrónico fingiendo ser otra persona, de manera que la víctima se lo crea y revele algún dato sensible, como su número de tarjeta de crédito.

Controlando el acceso al sistema

Como se ha comentado en la sección anterior, el acceso al sistema mediante una cuenta de usuario puede ser el primer paso para comenzar un ataque hacia el interior de la red o del propio sistema. Según se mire, puede que tener el control de una cuenta sea el primer o segundo muro, dependiendo si el atacante a tenido que saltar diferentes obstáculos para llegar a través de la red hacia el sistema o bien estamos ante un ataque directo físico, como por ejemplo un compañero hostil de nuestro entorno que conoce nuestra contraseña y accesa al sistema. En cualquiera de los casos es de suma importancia mantener un control sobre quien o que usuario está iniciando sesión en nuestro equipo o servidor que administramos. Para ello *Linux* pone a nuestro disposición varias herramientas y archivos que mantienen un control sobre la entrada y salida de usuarios, herramientas que ya hemos estudiado en su correspondiente capítulo y que son algunas como *who*, users, *w*, *lastlog*, *last*, *lastb* o *finger*, este último en desuso por sus vulnerabilidades. Repasemos el uso de estas herramientas y los archivos implicados:

- **last** y **lastb**: Muestran un histórico de los usuarios que han conectado y desconectado al sistema así como intentos de sesión fallidos, respectivamente. Los archivos binarios que corresponden a estos comandos son (por defecto) */var/log/wtmp* y */var/log/btmp*
- **lastlog**: Otro comando que nos permite ver un histórico de entradas al sistema, pero esta vez de todos los usuarios del sistema hayan o no iniciando sesión alguna vez. Este comando despliega el contenido del archivo de registro binario */var/log/lastlog*
- **w**, **who** y **users**: El comando '*w*' muestra los usuarios conectados actualmente en el sistema y el uso que hacen de este (procesos ejecutados) mientras que *who* solo muestra los usuarios conectados en ese momento en la máquina. Ambos por defecto despliegan el contenido del archivo binario */etc/utmp*. Con *who* además podemos ver el usuario actual (como con ***whoami***) si usamos su opción **-m** o desplegar el contenido de */var/log/wtmp* (como hace *last* por defecto) si le pasamos el nombre de archivo (con ruta absoluta) como argumento. El comando *users* tiene un comportamiento similar a *who* pero ofrece menos información.

Como podemos comprobar, el uso de los archivos de registros de acceso al sistema son de vital importancia aunque no del todo fiables pues un atacante podría manipularlos de manera que oculte sus accesos. Estos archivos deberían de tener los permisos 644 sin afectar por ello al funcionamiento del sistema de registros. A continuación veremos como implementar una mínima seguridad sobre estos archivos.

Atributos de un archivo

Existen ciertos atributos para los archivos que pueden ayudar a incrementar la seguridad de un sistema. No vamos a estudiar todos los que existen pero si alguno de ellos que nos pueden ser de utilidad. La herramienta para implementar el uso de atributos de archivos es *chattr* y su sintaxis es la utilizada para la gran mayoría de los comandos *Linux*: '*# Comando [opciones] archivo(s)*', solo que en las opciones usaremos los signos +, – o =

para añadir, quitar o dar exactamente unos atributos a uno o varios archivos respectivamente.

Nota: Podemos ver los atributos de un archivo con *lsattr*.

Acabamos de ver la importancia de los archivos de registro de accesos al sistema y sabemos que pueden ser atacados por un intruso bien de forma manual o a través de *rootkits*. Para añadir un extra de seguridad a estos archivos podríamos hacer que tomaran el atributo '**a**' de manera que el archivo fuera abierto exclusivamente para añadir información, nunca para borrarlo. Solo *root* puede hacer uso de este atributo, de manera que si la seguridad del sistema se ha visto comprometida por un intruso a nivel de *root*, solo sería cuestión de tiempo que el atacante modificara el atributo con *chattr -a archivo* y modificase o eliminase el archivo. No obstante de no tener privilegios de *root*, estaríamos teniendo constancia de la entrada del intruso al sistema. Otro atributo interesante es '**i**' con el que haremos que un archivo sea inmutable, es decir, no se puede escribir en el, modificar permisos, enlazan con *ln*, eliminar, etc… Al igual que pasa con el anterior atributo solo *root* puede activar o desactivarlo. Podríamos usar este atributo para hacer inmutable ciertos archivos de interés que no suelen ser modificados con regularidad, como archivos de configuración de */etc/*, el archivo de contraseñas de usuario, etc…

Nota: Cuando activamos el atributo **i** debemos de tener en cuenta que ni siquiera *root* podrá modificar el archivo, a menos que desactive el atributo.

Otros atributos a destacar, aunque menos importantes que los dos anteriores son **s** y **S**. Si eliminamos un archivo que tenía activado el atributo *s*, el sistema rellenará sus bloques con ceros en lugar de efectuar un simple unlink. De esta manera dificultamos la tarea de recuperación de un archivo por parte de un atacante, aunque una vez más esto solo supondrá una pérdida de tiempo para un atacante experimentado. El atributo **S** lo que hace es escribir los datos directamente al disco sin esperar a *sync* de aquellos archivos que lo tengan activado. Esto no muy significante, pero de apagarse nuestro equipo en el momento en el que un atacante entra al sistema, mantendremos esa información en el archivo de registro.

Para terminar vamos a nombrar otros atributos quizás útiles en determinadas circunstancias:

- El atributo **A** hace que la fecha de último acceso en un archivo no sea modificada
- Cuando un directorio tiene activo el atributo **D** los datos serán escritos de forma síncrona en el disco. Idéntico a **S** solo que *S* trabaja sobre archivos.
- Un archivo con el atributo **u** activado podrá ser recuperado con alguna herramienta destinada a tal fin.

Limitar el acceso a root

Acceder como *root* directamente al sistema suele ser desaconsejable, por varias razones, entre ellas, que no deja ningún rastro en los archivos de registro de acceso. En el capítulo 7 se trato el uso de herramientas como su y sudo, mejores opciones para acceder como *root* al sistema. Recordemos que con **su –** podemos cambiar a *root* bajo su entorno (gracias a '**–**'), mientras que con sudo podemos ejecutar un programa con privilegios de *root* siempre y cuando tengamos correctamente configurado el archivo */etc/sudoers*. Hablaremos de como configurar este archivo al final de la sección.

También podemos usar **su -c "programa"** para ejecutar '*programa*' con privilegios de *root*, aunque esto también suele usarse en *scripts* que solo *root* puede ejecutar pero que posteriormente queremos que sea un determinado usuario el que sea tutor o propietario de dicha ejecución. Imaginemos que tenemos una tarea de sistema bajo *anacron* (tareas que solo *root* puede ejecutar) pero el programa que queremos ejecutar es de nuestro usuario particular y además (aunque *root* pueda hacerlo) queremos que lo ejecute nuestro usuario, para conseguir esto podremos crear una tarea *anacron* que llame a un *script* cuyo contenido sea similar a:

```
!/bin/bash
# Ejecutar 'programa' con usuario 'miusuario'
su - 'miusuario' -c "/home/'miusuario'/bin/'programa'
```

De esta manera *root* ejecutará la tarea *anacron* (en este caso este pequeño *script*) que a su vez llamará a nuestro programa con nuestro usuario.

El archivo **/etc/nologin** aplica un concepto de seguridad especialmente radical. Si este archivo está presente, sólo *root* podrá acceder al ordenador.

Configurando /etc/sudoers

Para comenzar a editar este archivo lo abriremos con el comando **visudo**:

```
$ visudo /etc/sudoers
```

Vamos a suponer que realizaremos una configuración avanzada. Lo primero que deberemos de hacer será crear grupos de comandos para luego asignárselos a uno, varios o todos los usuarios, de manera que puedan hacer uso de ellos mediante la herramienta sudo:

```
# Crear grupos de comandos (Alias) - También podemos crear Alias para
grupos, usuarios, equipos y "ejecutar con" (Runas)
 Cmnd_Alias ALMACENAMIENTO = /sbin/fdisk, /sbin/sfdisk, /bin/mount
 Cmnd_Alias PROCESOS = /bin/nice, /bin/kill, /usr/bin/killall

# Definir quien podrá usar los grupos de comandos
# El grupo sys puede usar todos los comandos de ambos grupos
(ALMACENAMIENTO y PROCESOS)
 %sys ALL=ALMACENAMIENTO,PROCESOS
```

```
# El grupo disk podrá utilizar todos los comandos de ambos grupos y
además sin requerir de la password
 %disk ALL=ALMACENAMIENTO NOPASSWD:ALL

# El grupo wheel podrá usar todos los comandos en todas las máquinas y
sin password
 %wheel ALL:(ALL) NOPASSWD:ALL

# El usuario nebul4ck podrá usar todos los comandos del grupo proceso
solo en la máquina freeser y con el uso de password
 nebul4ck freeser = PROCESOS ALL

# El usuario kafka podrá ejecutar unos determinados comandos en la
máquina freeser sin requerir de autenticación:
 kafka freeser = NOPASSWD: /bin/kill, /sbin/fdisk, /bin/chmod,
/bin/chown

# El usuario darkns podrá ejecutar ciertos comandos con permisos del
usuario nebul4ck y con el grupo kafka:
 darkns blouder = (nebul4ck:kafka) /bin/ls, /usr/bin/lprm, /bin/chmod
```

La configuración de este archivo puede llegar a ser bastante compleja por lo que no vendría mal acudir a las páginas man de **sudoers**. Podremos usar comodines o crear el particular Alias "**Runas**" para permitir a ciertos usuarios ejecutar unos determinados comandos como si fuese un usuario específico. Para ejecutar un comando con un usuario o grupo distinto especificado en un Alias *Runas* podríamos hacerlo así:

```
$ sudo -u <usuario> <comando>
$ sudo -u <usuario> -g <grupo> <comando>
$ sudo -g <grupo> <comando>
```

Restringir el acceso a root

Podríamos hacer que el acceso de *root* al sistema estuviese restringido, algo que no se suele hacer, pero que si trabajamos sobre el archivo */etc/sudoers* de manera que tengamos los permisos necesarios para realizar la mayoría de tareas cotidianas sin poner en riesgo la seguridad del sistema, podemos trabajar a diario con un usuario particular limitado a *root* en ciertos aspectos, evitando así que un intruso se haga con el control pleno a través de *root* y comprometa nuestro equipo. Lo primero que deberíamos de hacer sería adaptar el archivo de configuración de *sudo* (*/etc/sudoers*) de manera que nos otorguemos el derecho de uso de ciertas herramientas de administración del sistema a nuestro usuario particular, luego, desactivaremos el acceso a *bash* de *root* editando la línea de */etc/passwd* tal que así:

```
root:x:0:0:root:/root:/usr/sbin/nologin
```

A partir de este momento no podremos acceder a consola como *root* ni emplear la herramienta *su* –

Nota: Cuando estudiemos la herramienta **ssh** veremos como restringir el acceso remoto de *root* al sistema, de manera que primero debamos de acceder vía *ssh* con un usuario estándar pasando luego a *root*.

Definir límites y cuotas a los usuarios

En ocasiones una medida de seguridad puede ser la definición de una serie de límites a los usuarios que hacen uso del sistema de forma que se proteja en cierto modo el uso que hacen de este. Existen diferentes formás de definir límites en el sistema que varían en función del objeto que queremos limitar. Por ejemplo, si lo que queremos es imponer un límite sobre la explotación de recursos que realiza un usuario en el sistema, como el uso de memoria, tiempo de CPU, número de procesos podremos editar el archivo */etc/security/limits.conf* o utilizar la herramienta **ulimit**. Si lo que queremos es imponen un límite en el uso del espacio del disco duro del ordenador o servidor o el número de archivos abiertos simultáneamente por un usuario concreto, podremos utilizar el sistema de cuotas que nos ofrece Linux. Este último método ya se explicó en el capítulo – 4 en la sección "Administrar cuotas de disco", de todos modos volveremos a refrescar nuestra memoria en esta sección.

Definir límites de accesos, procesos y memoria

La imposición de límites como el número máximo de accesos permitidos, procesos ejecutados de forma simultánea, memoria utilizada o tiempo de CPU es preferible hacerla mediante un módulo **PAM** (*Pluggable Authentication Module, Módulo de autenticación conectable*) llamado **pam_limits**. Esto se hace modificando su archivo de configuración */etc/security/limits.conf* cuyo formato de línea es el siguiente:

```
dominio tipo elemento valor
```

donde:

- **Dominio** : Es la entidad a la que se le aplica el límite. Puede ser un **usuario**, un **@grupo** o un '*****' (lo que coincidiría con todo el mundo.
- **Tipo** : Se indica el tipo de límite a imponer. Se pueden definir límites estrictos (**hard**) o flexibles (**soft**) cuya diferencia radica en que los límites *hard* son impuestos por *root* y no se pueden sobrepasar, mientras que los límites *soft* pueden ser sobrepasados temporalmente. Para indicar que un límite es tanto *hard* como *soft* podemos hacerlo con '**–**'
- **Elemento** : Especifica que tipo de elemento es el que se limita, por ejemplo, **core** limita el tamaño de los archivos de volcado del kernel, **data** limita el tamaño del área de datos de un programa, **fsize** limita el tamaño de los archivos creados por el usuario o grupo limitado, **nofile** para limitar el número de archivos de datos abiertos, **rss** tamaño del proceso residente, **stack** para limitar el tamaño de la pila, con **cpu** limitaremos el número de minutos que un proceso puede hacer uso de la *CPU*, **nproc**

para el número de procesos concurrentes, **maxlogins** limita el número de accesos simultáneos y **priority** la prioridad de acceso.

Nota: Los elementos *data*, *rss* y *stack* están relacionados con la memoria consumida por un programa.

- **Valor** : Indica el valor que será aplicado al límite.

Un ejemplo en el que se limita el tiempo de *CPU* en 2 minutos de forma estricta para un determinado grupo podría ser:

```
@migrupo    hard    cpu   2
```

El tiempo de *CPU* se cálcula en función del tiempo que la *CPU* está procesando activamente datos del usuario. Existe otro límite que mide el tiempo de inactividad del usuario, como por ejemplo cuando la consola del usuario está activa pero no hay en ejecución ninguna tarea.

La otra manera de imponer estos límites es empleando la herramienta **ulimit**. Para ello haremos uso de la siguiente sintaxis:

ulimit [opciones [limite]]

En paralelo a los elementos de limitación anteriormente mencionados usaremos para ello la opción **-c** (para los volcados del kernel), **-f** (para el tamaño de los archivos creados por consola por un usuario), **-u** (para el número de procesos que puede ejecutar un usuario), **-t** (para limitar el tiempo de *CPU* en segundos en vez de minutos), **-v** (define el tamaño total de memoria virtual disponible para la consola), **-s** (define el tamaño máximo de la pila de memoria), **-m** (limita el tamaño máximo del conjunto residente), **-d** (limita el tamaño del conjunto de datos de los programás) y **-l** (define el tamaño máximo que se puede bloquear en la memoria).

Si lo que queremos es especificar un tipo de límite concreto utilizaremos **-H** o **-S** para especificar *hard* o *soft* respectivamente.

Para conocer los parámetros actuales de *ulimit*, pasaremos la opción **-a**

Advertencia: Como *ulimit* es un comando nativo de *bash*, su utilidad como herramienta de seguridad del sistema es limitada. Si los usuarios tienen acceso a herramientas *GUI* de inicio de sesión o pueden acceder al sistema saltándose *bash* de algún modo (por ejemplo mediante *SSH*, dependiendo de como esté configurado), las restricciones impuestas por *ulimit* carecerán de sentido.

Aplicar cuotas de disco

Como ya se estudió en el *capítulo – 4* podemos aplicar límites de cuotas de disco para determinados usuarios o grupo, de manera que cuando se rebase el límite se informe al usuario. Con esto podemos prevenir el desbordamiento de espacio de ciertas particiones que podrían resultar críticas para el sistema. Por ejemplo si tenemos el */home* de los usuarios o el directorio */tmp* en la misma partición que la raíz '/' y no tenemos un control sobre el uso que se está haciendo de esta, podemos acabar por dejar el sistema inutilizable al llenar la partición. Para prevenir esto podemos implantar el sistema de cuotas, por lo que nuestro kernel deberá de estar compilado con los módulos que ofrecen soporte para las cuotas y además deberemos de usar como *root* una serie de herramientas, tales como **edquota**, **repquota**, **quotacheck** y **quota**.

Al igual que con los limites impuestos en el módulo *pam_limits* también podemos crear *quotas* estrictas (**hard**) o flexibles (**soft**) con un periodo de gracia.

Nota: Para profundizar en el sistema de cuotas de disco volver al *Capítulo – 4* sección "*Administrando cuotas de disco*"

Permisos sobre archivos

Algo que deberemos de tener en cuenta si queremos hacer poco a poco nuestro sistema más seguro, y digo poco a poco porque nunca será suficiente y porque ya llevamos aprendidas algunas técnicas pero evidentemente quedan más, entre ellas por ejemplo el conocer los permisos con los que cuentan los archivos de nuestro sistema. Evidentemente es difícil mantener un control sobre todos los archivos o *scripts* del sistema pero si seguimos unos patrones podremos identificar gran parte de ellos. Algunas de las acciones que podemos realizar para detectar o prevenir vulnerabilidades son la detección de archivos o *scripts* con los bits especiales (*suid*, *sgid* y *sticky bit*), prestar especial atención a los parámetros con los que montamos una partición, ficheros con permisos de escritura cuando en realidad no existe motivo especial por el que deban de tenerlo, ficheros extraños como por ejemplo aquellos que no poseen un propietario o ficheros cuya configuración puede resultar peligrosa como los *.hosts* aunque a estos le dedicaremos una sección.

SUID, SGID y Sticky Bit

Seguir el rastro a estos archivos puede resultar una tarea sencilla y eficaz pues como ya sabemos, el que un archivo o en especial un *script* tenga alguno de estos bits activados puede resultar en un agujero de seguridad para el sistema, ya que un *script* con el bit *suid* activado puede hacer que un intruso eleve sus privilegios tomando el control del sistema. Para localizar los archivos con bits especiales podremos usar el comando **find** de la siguiente manera:

```
# find / -perm +7000 -type f
```

Nota: Usamos *-type d* si queremos localizar directorios o anulamos esta opción para localizara a todos. Igualmente podemos restringir la búsqueda a tan solo archivos con *suid* (+4000), *sgid* (+2000) o con *sticky bit* (+1000)

Cuando realizamos una búsqueda con *-perm +4000* es decir estamos buscando archivos con el *suid* activado, deberán de aparecer aquellos programás para los que un usuario corriente tiene acceso de ejecución por ejemplo, si un usuario quiere cambiar su contraseña necesitará acceder al comando *passwd*, o si necesita cambiar diferentes características de su cuenta con *chfn* o *chsh*. Igualmente necesitará acceso a programás como *mount*, *ping*, *su* o *sudo*. Hay que prestar especial atención a todos los de la lista por si hubiese alguno que no entendemos porque lo tiene, y tras buscar por Internet o las páginas man llegamos a la conclusión de que determinado archivo no debería de tenerlo, podemos empezar a pensar de que nuestro equipo quizás esté comprometido.

nosuid, nodev, noexec

En la mayoría de las ocasiones, o al menos es una buena práctica, el tener una partición para cada uno de los siguientes directorios */home*, */usr*, */var* y */tmp* por diferentes razones. Si creemos que son demásiadas particiones, podemos entonces aislar a */home* y */var* del resto (que entonces compartirán partición con '/') y si aún así tampoco estamos muy convencidos, se aconseja que como mínimo que el */home* se encuentre en una partición independiente por motivos de seguridad, comodidad y flexibilidad (entre otros).

Si hemos seguido este buen y ya conocido consejo y tenemos particiones independientes, deberemos entonces de controlar la configuración de */etc/fstab*. Normalmente salvo casos excepcionales no existe una buena razón que nos obligue a ejecutar *scripts* con los bits *suid* y *sgid* activados en los directorios */home* de los usuarios, o en aquellas particiones con permisos de escrituras para usuarios no *root*, por lo que una medida que podremos tomar será añadir la opción (si es que no viene ya por defecto) **nosuid** a estas particiones. Podemos también evitar la creación de archivos de bloques o caracter y ejecución de programás en estos directorios y en */var* con las opciones **nodev** y **noexec** respectivamente.

Archivos con permisos de escritura

Otra búsqueda que podemos realizar por el sistema es aquella que localice a archivos o directorios que contienen activados el permiso de escritura para todos los usuarios. Es evidente que no querremos dejar que un invitado, intruso, atacante, compañero hostil o cualquier otra persona *non grata* pueda modificar archivos que consideramos importantes, vulnerables o que simplemente no queremos perder su control, sin más y mucho menos que un directorio caiga en manos de estos dándoles directamente permisos para añadir o eliminar contenido en él. Para localizar estos archivos y directorios podemos usar el siguiente termino de búsqueda con **find**:

```
# find / -perm -2 [-type f|d] -print
```

Esto imprimirá una larga lista que seguramente resulte bastante tediosa de seguir, por lo que podemos limitar la búsqueda por aquellos directorios que más nos interesen. Los enlaces simbólicos serán añadidos a la lista pero como sabemos el permiso definitivo será el que tenga el archivo al que enlaza. También aparecerán bastantes archivos bajo */var*.

Archivos sin propietario

Los archivos sin propietario pueden ser un indicio de que alguien ajeno a nosotros a estado fisgoneando nuestro sistema, por lo que no estaría de más dar un repaso en busca de estos archivos.

```
# find / -nouser -o -nogroup -print
```

Habilitación de hosts

Durante el *capítulo – 6* cuando configuramos el equipo para permitir el acceso remoto de otras aplicaciones al servidor X del sistema, vimos la importancia del comando *xhost* con el que se indicaba un *host* o *usuario* al que permitir acceso remoto mediante X. En el sistema, dependiendo de a que o para quien otorguemos permisos para una determinada acción, como dar acceso al servidor de X a aplicaciones o usuarios remotos, se crean unos determinados ficheros que de tener una configuración incorrecta o poco cuidada pueden resultar en agujeros de seguridad en el sistema. Aunque siempre será mejor utilizar protocolos seguros como *ssh*, vamos a citar algunos archivos con los que a la hora de configurarlos si se necesitase, deberíamos de andar con cuidado.

- **~/.rhost** : Cuando este archivo está presente en el directorio */home* de un usuario significa que se ha creado una relación entre este usuario y otro usuario o máquina de la red. En otras palabras estaremos dando acceso mediante los comandos (*rcp*, *rlogin*, *rexec*, etc…) sin contraseña a los usuarios y máquinas especificados en este archivo. Por ello deberemos de tener bastante cuidado a quien ofrecemos nuestro sistema. Un intruso podría utilizar este archivo para crear relaciones con máquinas externas.
- **/etc/hosts.equiv** : Es exactamente igual que el archivo anterior pero más peligroso aún, ya que se crean relaciones a nivel de máquina especificando que servicios, usuarios y grupos pueden acceder sin contraseña a los servicios. Un sigo '**+**' otorgará permisos a "cualquier" máquina.

Nota: Normalmente estos archivos ya no suelen existen a consecuencia del uso de protocolos seguros como *ssh*.

Módulos PAM

Los módulos PAM son bibliotecas compartidas que implementan un método que permiten al administrador del sistema controlar cómo se realiza el proceso de autenticación de los usuarios en determinados programás. Las bibliotecas PAM se encuentran en */lib/security* mientras que los archivos de configuración para cada aplicación que los utiliza se instalan bajo el directorio */etc/pam.d*. Algunas de las aplicaciones que utilizan estos módulos para la autenticación de usuarios podrían ser: *cron*, *su*, *sudo*, *samba*, *login*, *mdm*, *xdm*, *gdm*, *kdm*…

Una línea típica de estos ficheros de configuración podría ser:

```
modulo flag ruta-al-modulo argumentos
```

donde:

- **Modulo** : indica el requisito a realizar como por ejemplo, que sea necesario la autenticación (**auth**), que se trate de un acceso restringido (**account**), tareas a realizar cuando un usuario entra o sale (**session**), indicar la contraseña (**password**)
- **flag** o bandera de control : especifican las necesidades. Que sea un requisito (**requisite**), necesario (**required**), suficiente (**sufficient**) o opcional (**optional**). Otra posibilidad para las banderas de control es utilizar la sintaxis '*valor y acción*'
- ruta al módulo o **path**
- **Argumentos** : dependiendo de cada módulo podremos pasar uno u otros argumentos.

A modo de ejemplo se van a mostrar solo tres líneas de uno de los archivos de configuración, algo que no será suficiente. Los módulos PAM son material que no se abarca en *LPIC-1* y que solo se han comentado como base de conocimiento. Si se desea profundizar más, como siempre se recomiendo las páginas *man*, *HOWTOs* de Internet, *foros*, etc…

```
auth required pam_unix.so nullok
account required pam_unix.so
session optional pam_lastlog.so
```

Administrar la seguridad de la red

Dejamos atrás de cierta forma todo lo que aquí se recoge con respecto a la seguridad local y damos paso al primer o segundo muro con el que chocará un atacante, y decimos primer o segundo por el mismo motivo que comentábamos en la sección de la seguridad local, puesto que dependerá desde donde se produzca el ataque. Lo que ya tenemos claro es que contra más piedras pongamos en el muro más difícil o más tiempo llevará su derribo y con ello su penetración.

En esta sección del capítulo vamos a estudiar aquellos aspectos a tener en cuenta con referencia a la red. Haremos una pequeña introducción sobre alguna de las técnicas más usadas para el *ciberataque* y algunas posibles contramedidas.

Técnicas utilizadas en los ataques y contramedidas

La *ciberseguridad* es algo que se encuentra en constante cambio, siempre existen mejoras y al poco tiempo se encuentran vulnerabilidades que explotar para esas nuevas mejoras. Se pueden encontrar vulnerabilidades a nivel de software o hardware por lo que es importante mantenerse siempre actualizado. A continuación se enumerarán algunas técnicas de ataque o tipos de vulnerabilidades y sus contramedidas o posibles métodos preventivos para la detección de lo que puede estar sucediendo en nuestro sistema:

- **Bugs**: También llamados *holes* o *agujeros*. Son fallos en el software, hardware, servicios, protocolos o directamente en el sistema operativo que se aprovechan para romper la seguridad del sistema y explotar los recursos, obtener información, elevación de privilegios, etc… Para ello se utilizan los llamados *bugs exploits*, que no son más que fragmentos de código malicioso y comandos, que realizan técnicas de ataques genéricos o particulares para el elemento afectado. Para aquellos *bugs* que aún no han sido conocidos y que están siendo explotados suele utilizarse el termino ***día cero*** (tanto para el *bug* como para el *exploit*). Para remediar en lo posible caer en un ataque por *bugs*, es aconsejable mantenerse actualizado a nivel de software, hardware y servicios del sistema, así como a la desactivación e incluso desinstalación de todos aquellos programás que han quedado obsoletos o no son utilizados. Como administrador de sistema es igualmente importante mantenerse actualizado con respecto a las vulnerabilidades encontradas, como mínimo de aquellos programás de los que hacemos uso.
- **Virus**: Son programás normalmente adjuntos a otros que utilizan técnicas de autocopiado y transmisión, para infectar un sistema. Suelen anexarse a programás que permiten lenguaje de macros (no verificados) como ejecutables, correo electrónico e incluso documentos. En *Linux* si no hacemos un uso abusivo de *root* o no ejecutamos programar que desconocemos, estamos casi totalmente a salvo ya que no se utiliza lenguaje de macros no verificados, ni en los documentos ni en el correo electrónico. No obstante, pudiesen aparecer a consecuencia de *bugs* en el sistema o incluso podemos llamar correo "*virulento*" a aquel que se pueda reenviar haciendo uso de nuestro sistema vía *email* como el ***spam***. Para frenar un virus la primera medida a adoptar es la de instalar un sistema de *antivirus*, anti *spam* y anti *malware*. Luego será cosa nuestra el no ejecutar programás sospechosos, sistemás macros no verificados, crear reglas de filtrado tanto a nivel de correo electrónico como de firewall, etc..-
- **Worm**: El termino *worm* o *gusano* es utilizado para identificar a aquellos programás que aprovechan algún agujero del sistema para realizar ejecuciones de código malintencionado sin permisos, con el fin de hacer uso de los recursos del sistema, como por ejemplo la *CPU*. Al igual que los virus usan técnicas de *autocopiado* y

transmisión. Una buena medida a adoptar para saber si podemos estar infectados por un gusano es monitorizar nuestro sistema, comprobar el uso de los recursos que se está haciendo, los usuarios que hacen uso de este y sobre todo la explotación de recursos de hardware a horas en las que no existe un porque para que la máquina este trabajando de forma constante. Igualmente importante es analizar el tráfico de entrada y salida a la red.

- **Trojan Horse**: *El caballo de Troya* o *troyano* son programás útiles pero que ocultan funcionalidades malintencionadas utilizadas para obtener información del sistema o comprometerlo de manera que por ejemplo pueda ser utilizado en un futuro por su creador. Suelen ser buenos métodos de transmisión programás aparentemente muy útiles y utilizados como por ejemplo los generadores de claves usados para crackear algún programa, video-juegos, etc… Las aplicaciones web desarolladas en *Java*, *JavaScript* o *ActiveX*, en caso de existir *bugs* en estas, pueden ser un buen método de transmisión una vez que el usuario acepta y consiente la ejecución de la aplicación. Si queremos comprobar la integridad de un programa de lo mejor que podemos hacer es verificarlo mediante los mecanismos de suma de verificación y firmado (md5 o gpg). Como con los gusanos, deberemos de prestar atención al uso de red que se está produciendo, filtrando mediante *firewall* posible tráfico sospechoso.
- **Backdoor**: *Puerta trasera* en Español y que referencia posibles métodos de entrada '*escondidos*' en una aplicación o sistema, con el fin de poder evitar la seguridad y acceder a él. En numerosas ocasiones las *backdoor* son creadas a propósito con el fin de poder acceder a un sistema o programa bajo circunstancias estrictamente necesarias. Lo que se hace básicamente es introducir una secuencia especial dentro del código de programación con el fin de poder evitar los sistemás de seguridad del algoritmo (autentificación) para acceder al sistema o herramienta. Existen herramientas como ***Netcat*** que pueden ser empleadas para abrir puertas traseras o defenderse de ellas. Para saber si un software que vamos a proceder a instalar contiene *backdoors*, es primordial obtener de los proveedores del mismo la certificación de que éste no contiene ningún tipo de puerta trasera escondida no documentada. Si tenemos un programa ya descargado que se ha verificado y probado que no contiene puertas traseras, estaría bien crear una suma de verificación o respaldo del mismo de manera que podamos evitar la creación de una *backdoor* a posteriori volviendo atrás o comprobando su integridad mediante la suma.
- **Keyloggers**: Utilizado para captar la interacción del usuario con el teclado de manera que pueda '*secuestrar*' la contraseña del usuario en cuestión. Pueden ser programás individuales o troyanos incorporados en otros programás.
- **Rootkits**: La palabra *rootkit* proviene de la unión de '*root*', es decir raíz o administrador en sistemás *Linux* y '*kit*', conjunto de herramientas que permiten a un atacante obtener acceso con privilegio a la máquina esquivando los métodos de autenticación. Los *rootkits* se han catalogado como *malware* aunque por si solo no tiene porque afectar el rendimiento de la máquina ya que su función principal es la de ***esconder*** la presencia del atacante en el sistema (a él, a los puertos que utiliza, sus conexiones de red, etc..) mediante un conjunto de herramientas (cada unas destinada a un fin concreto). Lo que está claro es que no se instala un *rootkit*, algo que hace el atacante una vez que ha accedido por primera vez al sistema objetivo (ya sea por Ingeniería Social, fuerza brutal o cualquier otra forma) para nada, es decir instalar un *rootkit*

implica que la máquina va a ser usada con un determinado fin por el atacante y en numerosos casos se trata de explotar sus recursos.

Para defendernos de los *keyloggers* y los *rootkits* en la manera de lo posible, conviene revisar con frecuencia o bajo sospecha, los archivos que mantenemos abiertos usando por ejemplo la herramienta que ya conocemos '***lsof***', los procesos que se encuentran ejecutándose en el sistema con '***ps***' o bien auditar nuestra red con herramientas como ***netstat*** o ***wireshark*** (entre otras) por si se diera el caso de que un *keylogger* estuviese realizando envíos externos.

Nota: Podemos comprobar el funcionamiento de un *keylogger* muy básico con el comando **script** o verificar los *rootkits* con **chrootkit**.

- **Scanner**: Es un paso previo a un ataque. Durante el *scanner* se recolecta información sobre posibles objetivos a través de herramientas que examinan la red en busca de máquinas con puertos abiertos (*TCP*, *UDP*, etc…). Una herramienta de uso común para tal propósito es ***nmap***.
- **Sniffers** o **husmeadores**: Este tipo de herramienta captura los paquetes que circulan por una determinada red, de modo que podemos recopilar información acerca de la activad de la red como por ejemplo capturar contraseñas, protocolos utilizados, posibles servicios y servidores, clientes que aceden, etc… Al igual que los *scanner* no son un ataque en si ya que sirven tanto para auditar nuestra propia red como para encontrar vulnerabilidades en otra. En el *Capítulo 8* vimos las herramientas propicias para tal fin y algunos de los puertos a los que deberíamos de prestar especial atención.
- **Hijacking**: Conocido como 'secuestro' ya que su pretensión es la de "pinchar" las comunicaciones de manera de que se pueda reproducir el funcionamiento o uso de una máquina remota. Esta técnica es comunmente utilizada para captar correos electrónicos, transferencias de ficheros o navegación web. Con respecto a la navegación web lo que se vendría a realizar es una captura de la sesión de manera que tengamos la información de navegación del usuario(páginas visitadas, tiempo de visita, interacción con formularios, etc…). Contra el *hijacking* la mejor herramienta es utilizar servicios seguros con encriptación y autentificación, a poder ser con renovación periódica.
- **Denial of Service**: *Ataque de denegación de servicio* o *ataque DoS* utilizados para dejar una máquina inaccesible a través de la sobrecarga de uno o varios de sus servicios. Básicamente el objetivo es crear muchas peticiones (envíos de miles de paquetes) hacia uno o varios servicios de una máquina hasta provocar su saturación y con ello la caída de la máquina. Una variante de este tipo de ataque es el ataque *Distribuido de Denegación de Servicio* o *ataque DDoS* en el que se utiliza una red de ordenadores distribuidos, conocida como ***botnet***, ampliando enormemente la potencia y eficacia de la sobrecarga remota. Estos tipos de ataques suelen estar dirigidos normalmente a servidores que utilizan servicios web como *apache* o DNS como *bind* en los que se han encontrado *bugs* o quizás no estén lo suficientemente actualizados dejando en evidencia sus vulnerabilidades al atacante. Por ejemplo si un atacante detecta que el servidor al que quiere atacar tiene abierto el puerto 80 y además tiene instalado un servidor web, el siguiente paso sería comprobar la versión del servidor web y si está desactualizada, podría irse a páginas del propio servicio en la que se informan de los

bugs conocidos para una determinada versión y a raíz de ahí comenzar el ataque concreto. Normalmente estos servicios suelen caer al recibir más conexiones o peticiones a conexiones de las que pueden soportar.

- **spoofing**: Es una de las técnicas más complejas y suele englobar varios métodos de ataque. Por lo general el fin del *spoofing* es la falsificación de datos, ya sean IP (falsifican la identidad de una máquina para enviar o recibir datos), interlocutores, correo electrónico falso con el que robar datos empleando Ingeniería Social, etc…
- **Ataque por fuerza bruta**: Consiste en averiguar la contraseña de un usuario mediante la combinación de palabras, frases, números, etc… que normalmente se encuentran en uno o varios diccionarios. Básicamente funciona así; conociendo el algoritmo de codificación de la contraseña, se recurre a la comprobación de esas palabras o combinaciones de ellas que posteriormente se codifican con el mismo algoritmo y se comprueba su coincidencia, si no coincide se pasa a otra combinación y así hasta que alguna combinación codificada resulte ser la de la contraseña que buscamos. Para intentar evitar en la medida de lo posible que nuestra contraseña sea averiguada deberemos de seguir una serie de pautas que veremos en secciones posteriores, pero básicamente hay que elegir una contraseña con un mínimo de caracteres, combinando números, letras mayúsculas y minúsculas y caracteres de puntuación de manera que generemos una combinación de caracteres difícil de adivinar. Además es aconsejable modificar la contraseña cada cierto tiempo.

CONCLUSIÓN: Aceptar el software procedente únicamente de fuentes verificadas y de confianza es algo que deberemos de tener siempre en cuenta para frenar de algún modo cualquiera de las técnicas de intrusión o infección anteriormente descritas, además de tenerlo siempre lo más actualizado posible y mantenernos informado acerca de vulnerabilidades (*bugs*) que vayan surgiendo. El uso de *firewalls* y auditores de red (*scanners*, *sniffers*, comprobadores de puertos, etc..) pueden sernos de gran utilidad para detectar un tráfico sospechoso entre nuestro sistema y la red, así como detectar puertos que no deberían de mantenerse en escucha, por ejemplo por que no los estemos utilizando. Es de suma importancia con respecto al entorno local, mantener contraseñas consistentes empleando métodos de fabricación de contraseñas que las hagan menos vulnerables, monitorizar el uso de los recursos de hardware, sobre todo en horas en los que el ordenador no debería de tener un funcionamiento constante, controlar los procesos que se están ejecutando, así como los archivos que mantenemos abiertos. Si llevamos a cabo estas estrategias de control seremos menos vulnerables, de lo contrario no esperemos estar seguro suponiendo que ‘*alguien*‘ o ‘*algo*‘ está haciendo el trabajo por nosotros, por que no será real.

Controlar los puertos de red que se están utilizando

Una buena herramienta con la que podemos hacer un seguimiento de los puertos que tenemos abiertos e incluso saber como se ve nuestra máquina (con respecto a los puertos) desde el exterior (esto quiere decir que también nosotros podremos ver los puertos “accesibles” desde el exterior para una determinada máquina remota) es **nmap**.

Nmap es capaz de realizar exámenes más sofisticados, incluyendo entre otros, exámenes "secretos" que probablemente no detecten la mayoría de cortafuegos, y exámenes mediante *ping* para detectar que *hosts* están activos.

Alguno de los usos que podemos hacer de *nmap* son los siguientes:

- Comprobar como se muestra nuestra máquina de cara a la red (*TCP*/*UDP*)

```
# nmap -sTU localhost
```

- Comprobar si el objetivo está "vivo" (examen de *ping*):

```
# nmap -sP <dirección-IP>
```

- Quizás el host remoto esté configurado para bloquear intentos de ping, en este caso:

```
# nmap -nP <dirección-IP>
```

- Comprobar únicamente un rango de puertos:

```
# nmap -p U:53,111,137,T:21-25,80,139,8080 <dirección-IP>
```

Nota: Se aconseja visitar la página *man* del comando y practicar con esta herramienta.

Ahora que ya sabemos en que estado se encuentran los puertos en nuestra máquina podremos hacer un análisis y posteriormente tomar precauciones, es decir, quizás existan puertos desconocidos que se encuentren escuchando conexiones, o servicios activos que jamás hemos utilizado o simplemente no son lo suficientemente seguros, por lo que deberemos de hacer algo con ellos. En estos casos lo más normal es recurrir a los archivos */etc/services* en el que podremos ver a que servicio corresponde un determinado puerto o */etc/protocols* si tenemos problemás a la hora de identificar un protocolo concreto. Seguramente debamos de utilizar acto seguido herramientas de estado como **netstat** con la que seguir el rastro de aquellos puertos que desconocemos, para comprobar la posible actividad de puertos sospechosos o de servicios que aparentemente no usamos. Cuando tengamos localizado a que servicio corresponde cada uno de los puertos y tengamos claro si es conveniente utilizarlo o no, podremos entonces desactivar aquellos servicios que nos sean innecesarios. Para desactivar un servicio deberemos normalmente de acudir a nuestra herramienta de administración de servicios (*service*, *initctl* o *systemd* para *sysV*, *upstart* y *systemd* respectivamente) y pasar la opción **stop** para pararlo. Sería conveniente también desactivarlo del arranque, esto según que sistema de inicialización tengamos podremos hacerlo de una u otra forma, por ejemplo para *sysV* podremos eliminar el enlace del servicio en cuestión que se encuentre en el directorio de nivel de arranque con el que iniciamos, o utilizar la herramienta **chkconfig**, para *upstart* podremos eliminar, renombrar o editar el archivo de configuración de inicio del servicio en */etc/init* o */etc/init.d* si conserva el modelo de *sysV*, si estamos utilizando *systemd* tendremos que trabajar con las *units* y *targets* correspondientes ayudándonos de la herramienta *systemd*.

Muchos servicios están gestionados por por un súper servidor. Los dos súper servidores más utilizados en *Linux* son **inetd** y **xinetd**, siendo este último el más utilizado a en la actualidad. Para las distribuciones derivadas de *Red Hat* es *xinetd* el súper servidor por defecto, mientras que en derivados de *Debian* de suele utilizar *inetd*, aunque podremos sustituirlo sin problemás por *xinetd*. En próximás secciones hablaremos sobre estos súper servidores y veremos como configurarlos para que tomen el control de ciertos servicios y poder así activarlos o desactivarlos según nuestra necesidad.

Comprobar las conexiones de red existentes y los servicios que las provocan

Para esta tarea podemos usar una gran cantidad de herramientas cada cual más sofisticada y útil. Algunas de ellas podrían ser *netstat*, *nessus* o *wireshark*. Cualquiera de estas tres herramientas nos puede informar de una manera u otra de que es lo que está ocurriendo en nuestra red. **netstat** es la que vamos a ejemplerizar aquí y digamos que es una herramienta que muestra el estado de la red en un momento concreto, **nessus** es un detector de vulnerabilidades que se basa en una seríe de plugings y tiene una estructura cliente-servidor, por último, **wireshark** que es un *sniffer* o *husmeador* de redes.

Un uso básico y bastante informativo sería utilizar *netstat* de la siguiente manera:

```
# netstat -punta
```

Con estas opciones estamos indicando que muestre el *PID* del proceso que está haciendo uso de la conexión (**-p**), informar sobre los puertos *UDP* además de los *TCP*,algo que ya hace por defecto (**-u**), expresar los nombres de hosts y puertos en dígitos (**-n**), mostrar el estado de las conexiones (**-t**), muestra todas las conexiones y puertos de escucha (**-a**)

Cuando anteriormente ejecutamos el comando *nmap -sTU* quizás obtuvimos alguno puerto desconocido (*unknow*) podríamos investigar más sobre este de la siguiente manera:

```
# netstat -punta |grep <num-puerto>
```

Los súper servidores inetd y xinetd

Muchos servicios de red que tenemos instalado en nuestro sistema abren puertos de conexión y permanecen en estado de escucha (*LISTENING*) de forma directa, sin intermediarios. No obstante existen otros muchos servicios que si que operan a través de otro programa el cual los controla de cierta forma, a este programa o servicio se le

conoce con el nombre de **súper servidor**. El súper servidor escucha las conexiones de red para estos otros programás y cuando detectan un inicio de conexión para alguno de ellos, es entonce cuando este (el súper servidor) le pasa el control al programa en cuestión. Este sería el procedimiento desde el punto de vista de los puertos de red, pero igualmente el súper servidor puede controlar quién puede o no acceder a un determinado servicio, arrancar o parar un servicio, configurar los parámetros de seguridad, etc… Estas medidas de seguridad se suelen conseguir gracias a los **TCP Wrappers**.

TCP Wrappers o simplemente *wrappers* es básicamente un servicio que actúa de intermediario llamado **tpcd** (*/usr/sbin/tcpd*) que se utiliza junto a los súper servidores (*xinetd* lo lleva ya implementado en su propio paquete) y que sustituye al servicio gestionado por el súper servidor, de manera que cuando *TCP Wrappers* recibe una petición de acceso al servicio, verifica el usuario y el origen de esta, para determinar si su configuración le permite o no utilizarlo. Además de esto permiten generar logs e informar por correo de los posibles intento de acceso.

Puede que una cierta aplicación venga ya compilada con la biblioteca de *wrappers*, por lo que podremos gestionar sus permisos directamente desde los archivos */etc/hosts.deny* en el que especificaremos que servicio denegamos y a quién, y */etc/hosts.allow* donde indicaremos los servicios que vamos a utilizar seguido de la lista de máquinas a las que se le permitirá el acceso. Existen dos comandos útiles para gestionar los *wrappers* que son **tcpdchk** con el que verificaremos la configuración y **tcpdmatch** al que pasaremos un determinado servicio y cliente y nos informará de como reaccionaría ante esta situación.

Podemos recurrir al archivo */etc/services* para identificar aquellos servicios que queremos indicar en los archivos de configuración. Se pueden utilizar comodines o ALL para especificar todos los servicios. Para indicar uno o varios clientes podremos utilizar el nombre de las máquinas, la dirección IP o dirección de red, o bien usar el nombre de la propia máquina o dominio. Podemos usar el comodín ALL para cubrir el acceso a todas las máquinas o EXCEPT para generar excepciones.

Nota: */etc/hosts.deny* y */etc/hosts.allow* son los archivos de configuración de *TCP Wrappers*. Si una máquina aparece en ambos archivos, tendrá preferencia */etc/hosts.allow*

Otra ventaja de utilizar un súper servidor es que podemos reducir la carga de memoria, ya que si este controla muchos servicios que raramente se utilizan solo permanecerán en memoria el súper servidor y uno o dos de los servicios que controla.

Nota: Si queremos saber si estamos utilizando algún súper servidor y en caso afirmativo de cual de ellos se trata, podemos comprobarlo con el comando **ps** de la siguiente manera:

```
ps ax | grep inetd
```

Configurar inetd y xinetd

Tanto la configuración de *inetd* como *xinetd* pueden contener varios archivos de configuración, es decir no se trata de una configuración monolítica en la que únicamente se depende de un archivo de configuración. Como en numerosos servicios, existe un archivo de configuración principal desde el que se puede referenciar a un directorio en el que encontrar otros archivos de configuración. Los archivos de configuración principal son */etc/inetd.conf* y */etc/xinetd.conf* para *inetd* y *xinetd* respectivamente, así como el directorio que puede contener archivos de configuración independientes para cada uno de los servicios controlados por estos súper servidores son */etc/inetd.d* y */etc/xinet.d*.

Nota: Mientras que el archivo de configuración principal de *inetd* puede contener líneas de configuración de servicios o referencia a un directorio en el que se almacenan archivos de configuración independientes, el archivo de configuración de *xinetd* contiene solo opciones globales para el súper servidor y una directiva para incluir los archivos almacenados en */etc/xinetd.d*. Estos archivos son creados automáticamente cuando se instala un servicio que es gestionado por *xinetd*.

Si quisiéramos configurar el servicio *ftpd* bajo *inetd* podríamos crear una línea como la siguiente en */etc/inetd.conf*:

```
ftp stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.ftpd -l
```

donde:

- **ftp** : Indica el nombre del servicio
- **stream** : En este segundo campo se indica el tipo de *socket*, es decir que tipo de conexión se espera. En este caso se trata de una conexión fiable en dos sentidos (*stream*). Otros tipos podrían ser **dgram**, conexión menos fiable pero con menos sobrecarga o conexión de bajo nivel a la red (**raw**). Este tipo de información podremos conocerla a través de la documentación de la propia aplicación.
- **tcp** : Protocolo utilizado, normalmente *tcp* o **udp**.
- **nowait** : En este cuarto campo se indica una posible acción, *espera/sin espera* (**wait/nowait**). *nowait* indica que el servidor se conectará a su cliente y liberará el *socket*, en cambio, *wait* procesará todos sus paquetes y, después se desconectará pasado un tiempo determinado.
- **root** : Las opciones más habituales para este campo son *root* o *nobody* y como se prevé, indica el usuario con el que se ejecutará el servicio. No obstante, siempre que fuese posible deberíamos de ejecutar el servicio con un usuario con poco privilegios.
- **tcpd** : En este penúltimo campo se suele referenciar al binario del servicio, pero como estamos utilizando como intermediario a *TCP Wrappers*, indicamos que se ejecute primero *tcpd* quién hará de intermediario recibiendo así las peticiones de conexión.
- **in.ftpd -l** : En el último campo se nombrará al servicio final al que van destinada las conexiones, es decir, al servicio que vamos a usar, en este caso ftp. Se aceptan parámetros.

Esto sería con respecto a la configuración de un servicio bajo *inetd* pero ¿Y para *xinetd*?. Ya sabemos que en el archivo principal de *xinetd* (*/etc/xinetd.conf*) lo que se configura son los parámetros y opciones absolutas, es decir, las que afectan al propio súper servidor, delegándose la configuración de los servicios al directorio */etc/xinetd.d*, donde existirá un archivo de configuración por cada servicio instalado que corra bajo *xinetd*. La configuración para un servicio que corre bajo *xinetd* no es muy diferente a aquel que lo hace con *inetd*, lo que ocurre es que el formato utilizado es muy diferente. A continuación vamos a ver un ejemplo, en el que usaremos el archivo de configuración del servicio *ftp* pero esta vez ejecutándose sobre *xinetd*.

```
service ftp
 {
 socket_type = stream
 protocol = tcp
 wait = no
 user = root
 server = /usr/sbin/in.ftpd
 server_args = -l
 }
```

Existe un parámetro adicional (**disable**) con el que poder activar o desactivar un servicio. De modo que si quisiéramos desactivar este mismo servicio podríamos haber añadido **disable = yes**. Si lo que queremos es desactivar varios servicios en una sola línea podremos añadir este mismo parámetro seguido de los nombres de todos los servicios a desactivar en lugar de '*yes*' en la sección **defaults** del archivo */etc/xinetd.conf*.

Esto mismo ocurre con la configuración de la seguridad de los servicios. Por ejemplo podremos implementar la seguridad de forma individual para cada servicio con parámetros como:

- **bind** : Indica a *xinetd* en que direcciones *IP* se escuchará en busca del servicio. Podemos restringir la escucha de manera que solo se disponga del servicio de forma local indicando la interfaz de circuito cerrado (*127.0.0.1*)
- **only_from** : Aceptará solo conexiones provenientes de las direcciones IP, direcciones de red o nombres que indiquemos. Es similar a la función que realiza el archivo */etc/hosts.allow* de *tcpd*.
- **no_acces** : Idéntico a *only_from* solo que en este caso rechaza conexiones. Parecido a la función que ejerce */etc/hosts.deny* de *tcpd*.
- **acces_time** : Define el rango horario en el que el servicio estará disponible para los usuarios. Se especifica en formado *hora:min-hora:min*.

Aviso: El rango horario indica las horas a las que se puede acceder al servicio, independientemente de las horas de las que se haga uso, es decir si el rango está marcado en 08:00-17:30 y un usuario comienza a hacer uso del servicio a las 17:29 podrá hacer uso de él desde ese minuto en adelante, es decir, que no se le va a cortar el acceso en cuanto pase al minuto 17:30, eso si, un nuevo usuario que intente hacer uso del mismo servicio a las 17:31 obtendrá un acceso denegado.

Al igual que ocurría con el parámetro *disable*, si queremos implementar opciones e seguridad de forma general deberemos de indicarlas en la sección *defaults* del archivo de configuración principal */etc/xinetd.conf*

Importante: Siempre que se realicen cambios en la configuración, tanto a nivel global como individual e independientemente del súper servidor que estemos gestionando, deberemos de recargar su archivo de configuración con **reload** o bien reiniciar el súper servidor con **restart**.

Protección mediante Firewalls

¿Que es un firewall?

Un **firewall** o **cortafuegos** es un sistema o dispositivo que se instala en una red con el objetivo de filtrar el flujo de datos (tráfico) que por ella discurre, de manera que sea capaz de separar los diferentes tipos de 'conversaciones' y tratarlas según nosotros, los administradores, hayamos decidido cuando creamos las reglas.

El filtrado actúa mediante la inspección de los paquetes, que representan la unidad básica de transferencia de datos entre ordenadores en Internet.

Al crear las reglas de filtrado en un cortafuegos, normalmente se tienen en cuenta diferentes aspectos, como el origen y destino de los paquetes (por ejemplo desde Internet hacia nuestra red privada), el protocolo utilizado y el puerto de destino. Una vez hemos considerado algunos de estos aspectos, provocaremos que se ejecute una acción. Algunas de ellas podría ser: permitir la entrada de los paquetes, redirigirlos, denegarlos o "perderlos".

Nota: Existe una diferencia clave entre denegar una conexión (**reject** rechaza los paquetes) a perderla (**drop**). Cuando se rechaza una conexión el emisor obtiene un "mensaje de respuesta" (normalmente un paquete *ICMP*) con el que se informa de la denegación, mientras que si hacemos que se pierdan los paquetes el emisor no recibirá ningana información al respecto.

Un cortafuegos correctamente configurado añade una protección extra a la red, pero en ningún caso debe considerarse suficiente. Cualquier tipo de ataque informático que use tráfico aceptado por el cortafuegos seguirá constituyendo una amenaza.

Un cortafuegos no puede protegernos de aquellos ataques cuyo tráfico no pase a través de él, ni de los fallos de seguridad de los servicios y protocolos cuyo tráfico esté permitido.

Algunas configuraciones

Existen diferentes formás de implementar el firewall en la red, pero sin duda dos de las más comunes podrían ser las siguientes:

- **Un firewall como muro de una red interna**: En este tipo de configuración suele utilizarse un *router* (el encargado de proporcionarnos el acceso a Internet y redirigir el tráfico de la red) conectado a un *switch* o directamente a un equipo (un pequeño ordenador portátil o de sobremesa valdría) en el que disponemos de dos interfaces de red. Una de las interfaces será la que conecte al *router* (nuestra puerta de acceso) mientras que la otra será conectada a un *switch* donde se comunicará con el resto de ordenadores y servidores de la red interna. Por esta segunda interfaz es por donde pasará todo el tráfico filtrado por nuestro firwall.
- **Un firewall como barrera protectora de un único sistema**: La mayoría de las distribuciones *Linux* traen (activado o no) un cortafuego, se trata del Netfilter **IPtables**. Con *IPtables* podremos crear reglas de filtrado de manera que podamos hacer más segura nuestra conexión con Internet. En ocasiones se suele utilizar para hacer *NAT* o redirigir conexiones según los puertos (*port forwarding*), por ejemplo cuando tenemos varios servidores como *smtp*, *web*, *ftp*, etc.. en diferentes ordenadores de la red privada.

NAT (*Network Address Translation*) : técnica que permite ocultar varios ordenadores (red privada) tras una sola IP pública. Esto se logra mediante el *NAT másquerading*. Los *routers* proporcionados por nuestro **ISP** (*Internet Service Provider*) emplean esta técnica además de implementar un *firewall* en la mayoría de los casos.

Netfilter: IPtables

IPtables permite realizar diferentes configuración mediante el uso de las reglas que afectan al sistema de filtrado, ya sea generación de logs, acciones de pre y post routing de paquetes, *NAT* o forwarding.

IPtables se constituye en tres tablas: **filter** (para el sistema de filtrado), **nat** (enmáscaramiento y reenvío) y **mangled** (configuración de opciones de paquetes). Estas tablas a su vez contienen una serie de cadenas que agruparán determinadas reglas según su función. Por ejemplo para la tabla **NAT** contamos con *PREROUTING* que permitirá modificar paquetes entrantes antes de su enrutamiento, *POSTROUTING* los modificará antes de que salgan al destino y *OUTPUT* que modifica los paquetes generados una vez se hayan enrutado, mientras que en la tabla **FILTER** podemos añadir reglas a las cadenas (*chains*) *INPUT* para los paquetes que entran, *OUTPUT* para los que salen y *FORWARD* si quisiéramos realizar un reenvío.

Importante: Cuando queremos hacer **forwarding** de paquetes, por defecto *GNU/Linux* no trae habilitada la opción por lo que deberemos de activarla nosotros. Podemos hacerlo de forma temporal (será volátil) a través del sistema de ficheros */proc* o bien de forma estática:

- Volátil:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

- Permanecerá a reinicios: deberemos de editar el archivo */etc/sysctl.conf* y añadir:

```
net.ipv4.ip_forward = 1
```

Los paquetes son analizados primeramente por la tabla *NAT*, a continuación la tabla *FILTER* decidirá si pasarán (en caso de que vayan destinado a nosotros) o si hay que reenviarlos a otro destino.

No podemos abarcar ni 1% de lo que supone *IPtables* por lo que se recomienda acudir a su página man. No obstante se va a explicar como crear algunas reglas básicas para las tablas *FILTER* y *NAT*.

Para comenzar a utilizar *IPtables* deberemos de tener el servicio corriendo. Esto ya sabremos como hacerlo, en función del sistema de inicialización que estemos usando. Para *sysV* bastará con */etc/init.d/iptables start* (*stop* para pararlo).

Sería interesante comprobar de que reglas dispone la configuración actual de *IPtables*:

```
$ sudo iptables -L
```

Crear reglas en la tabla *FILTER*

Para crear una regla básica en la tabla *FILTER* y en la cadena *INPUT*, podríamos utilizar la siguiente sintaxis:

```
$ sudo iptables -A INPUT -s 192.168.1.0/24 -j ACCEPT
```

Con **-s** <*ip*> indicamos el origen (*source*) con **-j** la acción (en este caso aceptar) y podríamos haber utilizado **-d** para indicar un destino.

El orden de las reglas es importante, por lo que deberemos de colocar siempre las menos restrictivas en las primeras posiciones, podemos hacerlo directamente desde la línea de comandos, tal que así:

```
$ sudo iptables -I INPUT 3 -s 192.168.1.0/24 -j ACCEPT
```

Esto haría que la regla se colocara en la tercera posición. La opción **-A** simplemente añade, **-I** inserta en una determinada posición.

Podremos eliminar por ejemplo esta misma regla si pasamos la opción **-D**

```
$ sudo iptables -D INPUT 3
```

o

```
$ sudo iptables -D INPUT -s 192.168.1.0/24 - ACCEPT
```

Una regla más elaborada podría quedar así:

```
$ sudo iptables -A INPUT -p tcp --dport 80 -s 0.0.0.0 -d 192.168.1.23
-j ACCEPT
```

Con esta regla indicamos que cualquier conexión proveniente (**-s**) de cualquier origen con destino (**-d**) al equipo con IP *192.168.1.23* concretamente por el puerto 80 (**–dport**, puerto destino) con protocolo *tcp* (**-p**) sea (**-j**) aceptada.

Y con alguna opción:

```
$ sudo iptables - A INPUT -p tcp --dport 80 -s 0.0.0.0 -d 192.168.1.23
-j REJECT --reject-with tcp-reset
```

Aquí rechazamos las conexiones (*REJECT*) y además emitimos una respuesta del tipo *tcp-reset*

Cuando modificamos las reglas deberemos de salvarlas. Para hacerlo en *Red Hat* bastaría con pasarle el parámetro '**save**' (*service iptables save* o */etc/init.d/iptables save* y estas se guardarán en **/etc/sysconfig/iptables**.

Si estamos usando una distribución *Debian* o derivado además de '*save*' deberemos de pasarle un nombre por lo que acabarán guardándose en **/var/log/iptables/<nombre-reglas>**

Aviso: Si el directorio no existe deberemos de crearlo.

Para cargar las reglas podremos pasar el parámetro '**load**' en *Red Hat* y '**load nombre-reglas**' en *Debian*.

Crear reglas en la tabla NAT

Podemos ver las reglas para una determinada tabla con el parámetro **-t** y en nombre de la tabla:

```
$ sudo -t nat -L
```

Si queremos ser más meticulosos y ver las reglas de una cadena concreta y además con detalles:

```
$ sudo -t nat -L PREROUTING -n -v
```

La opción **-n** da resultados numéricos y **-v** detalla información

Podemos eliminar todas las reglas de una tabla:

```
$ sudo -t nat -F
```

O concretar a solo las de una determinada cadena de una tabla concreta:

```
$ sudo -t nat -F OUTPUT
```

Podemos además de eliminar todas las reglas, poner los contadores a cero (útil cuando vamos a comenzar a definir reglas desde 0):

```
$ sudo -t nat -F
$ sudo -t nat -Z
```

Nota: Podemos omitir la tabla para que afecte de forma global.

Crear la regla de oro, es decir, conectar todos los ordenadores de una red privada con Internet a través de un dispositivo que posee dos interfaces de red, una conectada hacia Internet y la otra hacia el *switch* de la red privada. Esto recibe el nombre de *Source NAT* o **SNAT** (cambia la dirección *IP* de origen, en este caso para convertirla en la *IP* pública de nuestro *ISP*):

```
$ sudo iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -o eth0 -j
SNAT --to 88.23.144.210
```

Añadimos esta regla a la cadena *POSTROUTING* indicando el origen (**-s**) nuestra red privada. Solo se aplicará a los paquetes que salgan (**-o**, *output*) por la interfaz de red '*eth0*', la acción (**-j**) será *SNAT* y la nueva *IP* es *88.23.144.210* (la ofrecida por nuestro *ISP*). Existen diferentes formás de conocer nuestra *IP* pública, por ejemplo a través del *router* o de Internet, accediendo a direcciones como http://www.cualesmiip.com

Aviso: Este tipo de regla es configurada de esta manera cuando la *IP* que nos ofrece nuestro *ISP* es estática. Suele ser un servicio extra ofrecido por la compañía mediante pago.

Ahora vamos a crear la misma regla pero para una *IP* dinámica, es decir, lo común en cuanto a las *IP* ofrecidas por los *ISP*.

```
$ sudo iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -o eth0 -j
MÁSQUERADE
```

Como vemos solo cambia la acción, que en este caso usamos *MÁSQUERADE* que lo que hace es usar la *IP* que tiene en esos momentos configurada nuestra interfaz *eth0*

Para terminar con la configuración de reglas básicas de *IPtables* vamos a crear una regla común, que por lo general se suele hacer a nivel '*casero*' desde el propio *firewall* del *router* ofrecido por nuestro *ISP*, y es la de direccionar peticiones de acceso a un determinado servicio que escucha en un puerto concreto. El circuito típico sería, una conexión que entra desde la *IP* pública y que debe ser encaminada hacia una dirección

IP privada específica en la cual puede haber corriendo varios servicios pero en concreto queremos configurar el servidor web que por defecto escucha en el puerto 80. Para conseguir esto deberemos de aplicar la regla a la cadena *PREROUTING* ya que el paquete acaba de entrar y seremos nosotros quien le cree la ruta de encaminamiento.

```
$ sudo iptables -t nat -A PREROUTING -p tcp --dport 80 -i eth0 -j DNAT
--to 192.168.1.52
```

Nuevas opciones serían la **-i** (*input*) y la acción *DNAT* que es equivalente a *–to-destination* con la que se modifica la *IP* de destino.

ufw el front-end de IPtables

Algunas distribuciones como *Debian* y sus derivados, traen preinstalados el paquete **ufw** (*Uncomplicates Firewall*) que no es más que un *front-end* para *IPtables*, de manera que podamos crear reglas de una forma más sencilla. Para los fanáticos del GUI existe **GUFW** que igualmente es un *front-end* (esta vez gráfico) para crear reglas *IPtables*.

Un uso sencillo de ufw podría ser el siguiente:

- Denegar todo el tráfico entrante y permitir todo el saliente:

```
$ sudo ufw default deny incoming
$ sudo ufw default allow outgoing
```

- Abrir un puerto para un determinado servicio (bien por número o por nombre de servicio):

```
$ sudo ufw allow 139
$ sudo ufw allow samba
```

Nota: Para eliminar una regla añadimos **delete** entre *ufw* y *allow*

- Comprobar los nombres de servicio que maneja *ufw*:

```
$ sudo ufw app list
```

- Abrir un puerto y especificar el tipo de protocolo:

```
$ sudo ufw allow 443/tcp
```

Nota: Para abrir un rango usamos *xxxx:yyyyy/protocolo*

- Crear una regla algo más compleja (permitir todas las conexiones ssh a mi máquina cuyo origen se encuentra dentro de mi red)

```
$ sudo ufw allow from 192.168.1.0/24 to 127.0.0.1 app ssh
```

- Ver la configuración actual:

```
$ sudo ufw status verbose
```

- Resetear todas las reglas:

```
$ sudo ufw reset
```

- Habilitar o deshabilitar el servicio al inicio:

```
$ sudo ufw enable/disable
```

Sistemás de detección de intrusos (IDS)

Los sistemás de detección de intrusos (*IDS*) nos permiten mediante la utilización de nuestros propios recursos y la explotación de nuestro sistema, detectar posibles intrusiones en tiempo real. Lo normal es utilizar un detector de intrusos cuando hemos "terminado" aparentemente de configurar la seguridad de nuestra red y sistemás, de manera que los *IDS* que instalemos nos encuentren posibles vulnerabilidades que podamos intentar resolver, cerrando con ello agujeros de seguridad no deseados en nuestros sistemás. Igualmente su uso periódico nos informará de posibles ataques, uso malicioso de los recursos del sistema, modificación de ficheros, tráfico sospechoso, puertos que no deberían de permanecer abiertos, etc…

Podemos instalar y explotar un *IDS* destinado a localizar vulnerabilidades en el sistema local, denominado **IDS basado en host**, de forma que filtre los registros del sistema producidos por el demonio *syslogd* y los compare con una base de datos interna en la que se registran vulnerabilidades y ataques conocidos. Aquellos registros de log que hayan sido catalogados como anómalos por el *IDS* serán re-etiquetados con el nivel de criticidad que estime oportuno el detector de intrusos y se recopilarán en un nuevo log que posteriormente será enviado al administrador para su análisis. Para aquellos archivos de configuración típicos o servicios bastantes usuales, los *IDS* suelen crear sumás de verificación con *md5sum* o *sha1sum* cuyo resultado servirá para ir comparando periódicamente si los archivos han sufrido algún tipo de modificación. Con todo esto podremos encontrar archivos de configuración de servicios típicos que hayan sido modificados, servicios que trabajan de forma sospechosa, explotación indebida de los recursos del sistema, etcétera.

Igualmente podemos analizar la red en busca de servicios mal configurados, puertos abiertos que experimentan tráfico sospechoso, interfaz de red en modo promiscuo, etc…

Algunos de los sistemás de detección de intrusos más populares son:

- **Tripwire**: Es el *IDS basado en host* más popular. No está soportado en Red Hat. Mantiene una base de datos de sumás de verificación de los ficheros más importantes del sistema con la que poder comparar cada cierto tiempo en busca de posibles modificaciones. De esta forma sirve como medida preventiva ante posibles modificaciones en archivos de configuración del sistema provocados frecuentemente por los *rootkit*. La herramienta libre cuyo funcionamiento se asemeja al de *TripWire* es **AIDE**.
- **RPM** como *IDS basado en host*: Es posible usar el gestor de paquetes de Red Hat como un *IDS basado en host*. Para ello deberemos de importar la **GPG** de Red Hat a nuestro llavero de claves públicas de forma que se puedan comparar los paquetes instalados con la base de datos de Red Hat. Las opciones que nos ayudarán a esto serán **-V**, sola o junto con otras opciones como '**a**' que verifica todos los paquetes o '**f**' que verifica un único archivo dentro de un paquete. Por defecto *-V <nombre_de_paquete>* verifica todos los archivos que contiene el paquete y **-K** útil para comprobar la suma de verificación y la *GPG* de un determinado paquete *.rpm*. Esta última opción es útil si queremos comprobar que un paquete a sido firmado por Red Hat (veremos las firmás *GPG* en la última sección de este capítulo).
- **SWATCH**: Aunque no fue diseñado para ser un *IDS basado en host*, hace el trabajo de este al comprobar los registros del sistema y alertar de cualquier posible amenaza.
- **LIDS** (*Linux IDS*): Parche del *kernel* y herramienta de administración útil para controlar las modificaciones en archivos a través de *ACLs*.
- **Snort**: *IDS basado en red* de poco peso capaz de realizar análisis del tráfico de red en tiempo real. Permite filtrar por protocolos o contenido mediante la creación de reglas. *Snort* identifica gran cantidad de intrusiones y pruebas como *buffer overflows*, escaneos indetectables de puertos, pruebas de *SMB*, intentos de reconocimientos del sistema operativo, etc…

El servicio SSH

En la sección sobre la seguridad en la red vimos algunos servicios o protocolos de conexión remota (*Telnet*, *ftp*, *VNC*, escritorio remoto mediante X, etc..) los cuales compartían una vulnerabilidad, enviar los datos sin encriptar. Si estuviésemos utilizando algunos de estos servicios y existe en nuestra red un fisgón o intruso que esta monitorizando nuestras transferencias, este podría hacerse fácilmente con nuestras credenciales.

Para cerrar esta gran brecha de seguridad se diseñó el protocolo **SSH**, el cual utiliza técnicas de encriptación fuerte en todas las partes de la conexión de red, lo que lo convierte en un protocolo de conexión remota mucho más seguro. El protocolo *SSH* nos permite transferir archivos (**scp**), conexión remota con X (**ssh -X**), utilizar servicios como *ftp* de forma segura (**sftp**) e incluso la posibilidad de crear túneles por los que transmitir la información de otros protocolos que no encriptan sus datos.

Nota: Podemos usar el comando **scp** de manera similar a como lo hacemos con *cp*, de forma que podamos copiar uno o varios archivos a una máquina remota de forma segura. Para utilizar *scp* bastaría con utilizar la siguiente sintaxis:

```
scp <archivo_a_copiar>
usuario_remoto@ip_o_nombre_de_máquina_remota:/directorio/donde/copiaremos/el/archivo
```

Aviso: Es obligatorio conocer la contraseña del usuario remoto, a menos que tengamos activado un acceso sin contraseña, algo que aprenderemos en una sección posterior.

La principal desventaja de *SSH* es que la encriptación y desencriptación consumen tiempo de *CPU* lo que ralentiza las conexiones degradando incluso el rendimiento global del sistema.

El servidor más popular en Linux es **OpenSSH** (*sshd*), del que se dispone también de su versión cliente (*ssh*).

Configurando SSH

Podremos iniciar el servidor **sshd** desde un súper servidor o mediante su script de inicio. Siempre que modifiquemos su configuración será necesario el reinicio del servicio o recarga del archivo de configuración.

La mayoría de la configuración de *sshd* se realiza a través de su archivo de configuración **/etc/ssh/sshd_config** mientras que la configuración del cliente podremos llevarla a cabo mediante el archivo */etc/ssh_config*.

El archivo de configuración /etc/sshd_config se compone mayormente de líneas que adoptan el siguiente formato:

```
opción valor
```

Por regla general el servicio *ssh* (tanto el cliente como el servidor) vienen preconfigurado de manera que podemos hacer un uso seguro de él, no obstante conviene revisar algunas de las opciones, sobre todo de cara a la parte del servidor, como por ejemplo:

- **El nivel de protocolo que estamos usando**: *OpenSSH* admite los niveles 1 y 2, siendo el 2 el más seguro aunque en ocasiones se encuentra límitado con respecto a la comunicación con otros clientes. Podemos configurar el nivel de protocolo de manera que se use uno u otro o incluso los dos que según cual señalemos como primera opción será el que se use por defecto, por ejemplo 2,1.

- **Permitir el acceso directo de root mediante OpenSSH**: Una medida de seguridad sería establecer la opción **PermitRootLogin** en **no** de manera que para hacerse con privilegios de *root* no baste con conocer solo su contraseña, si no que deberá de acceder primero con otro usuario vía *ssh*, por lo que deberá de conocer a parte de la clave de *root* la de ese determinado usuario y acto seguido cambiar a *root* mediante *su*. Ya vimos como además cambiando el *shell* por defecto del usuario *root* en */etc/passwd* podemos incluso prohibirle el acceso a la consola.
- **Activar X remoto**: Con la opción **X11Forwarding** podremos indicar si deseamos activar la funcionalidad del túnel para X. Si deseamos usar *ssh -X usuario@host* para acceder de forma remota al servidor X, deberemos de cambiar el valor de esta opción a **yes**.

Lo ideal sería que configuráramos *SSH* para que solo aceptara conexiones de nivel 2 del protocolo y rechazara accesos directos como *root*. Si no es necesario el redireccionamiento de X, deberíamos de desactivar la opción *X11Forwarding*. En la medida de lo posible deberemos de utilizar un cortafuegos (filtrando el puerto 22) o *TCP Wrappers* (*/etc/hosts.allow* y */etc/hosts.deny*) para limitar los equipos que pueden contactar con el servidor *SSH*. Igual que podíamos restringir la conexión directa de *root* a través de *ssh*, podemos hacer que se permita solo su acceso. Si existe el archivo **/etc/nologin** *SSH* lo tendrá en cuenta y negará el acceso a todos los usuarios menos a *root*.

Como con todos los servidores deberemos de mantenerlo actualizado pues siempre existe la posibilidad de que un *bug* cause problemás.

Configurar túneles SSH

Como ya se comentó, *SSH* tiene la capacidad de crear túneles por donde otros protocolos conducirán sus datos de manera que estos sean encriptados, evidentemente esto precisa de configuraciones adicionales.

Nota: A la creación de túneles *SSH* para conducir el flujo de datos creados por otros protocolos se le conoce como **tunneling**

Para que veamos de una forma más o menos clara el funcionamiento del *tunneling* vamos a ejemplarizar una configuración en la que se desea que el tráfico de correo electrónico desde el servidor *IMAP* (en este caso) hacia el cliente, se realice bajo un túnel *SSH* de manera que se conserve la privacidad. Para ello el servidor de correos dispondrá tanto del servidor *IMAP* (protocolo que hará uso del túnel) como del servidor *SSH*, y de la misma manera el cliente deberá de tener ejecutándose un cliente *SSH* y un cliente del protocolo que hará uso del túnel (un cliente de correos en este caso). Con esto la comunicación será encriptada por *SSH* y enviada a través del túnel bajo el protocolo *IMAP*.

Para que esta comunicación sea posible deberemos de activar la opción **AllowTcpForwarding** en el archivo de configuración del servidor *SSH*

(*/etc/sshd_config*) y en el lado del cliente deberemos de crear una conexión especial con el servidor de correos como esta:

```
# ssh -N -f -L 143:mail.freeser.com:143 nebul4ck@undominio.es
```

donde:

- **-N** indica que no ejecute un comando remoto
- **-f** que se ejecute en segundo plano tras solicitar la contraseña
- **-L** indica el puerto local en el que se escucha (normalmente 143 para el protocolo *IMAP*, utilizado tanto en el cliente como en el servidor)
- Tras los primer dos puntos '**:**' encontramos la dirección del servidor de correos y el puerto en el que se escucha. El último parámetro el usuario y *host* con el que estableceremos el túnel.

Una vez establecido el túnel podremos utilizar nuestro programa cliente para conectarnos al puerto local especificado (143)

Uso de las claves SSH

Uno de los sistemás de seguridad que implementa el protocolo *SSH* es el uso de claves, especialmente dos, la **clave pública** (con la que podremos desencriptar mensajes) y la **clave privada** (la utilizada para encriptar mensajes). En una comunicación cifrada cada lado envía su clave pública al otro extremo de manera que solo con esa clave se puedan descifrar los mensajes que han sido encriptados por su respectiva clave privada.

Los datos encriptados con una clave pública solo podrán ser descifrados por la clave privada coincidente. Un ejemplo sería la conexión remota entre un cliente y un servidor bajo *ssh*, donde el cliente tendrá la clave pública del servidor al que conecta y cuando codifique información lo hará mediante esta clave de manera que solo el servidor bajo su clave privada pueda descifrar estos datos.

Importante: Si algún intruso se hiciese con la clave privada del servidor, correríamos el riesgo de que todos nuestros datos fueran descifrados por el atacante.

Las claves públicas de uso individual de los servidores, se guardan en el directorio **~/.ssh/known_hosts** del cliente, mientras que las de uso global se encuentran en el archivo **ssh_known_hosts** que suele colgar de */etc* o */etc/ssh*. Una modificación en una clave pública sería motivo de sospecha de falsificación.

En el pasado podíamos comprobar el contenido de estos archivos con editores de texto, en cambio, desde *OpenSSH 4.0* es posible habilitar la encriptación para los datos de este archivo de manera que la información se codifique mediante un algoritmo de encriptación de único sentido (*hash*). Así ni los intrusos ni nosotros mismos, podremos

identificar el servidor al que nos conectamos, únicamente el cliente y servidor compararán el *hash* y si coincide se aceptará la conexión.

Para que este sistema funcione hacen falta un total de cuatro a seis claves generadas automáticamente cuando se ejecuta el *script* de inicio de *sshd*. Estas claves necesarias son las correspondiente a las dos o tres herramientas de encriptación que admite *SSH* (**rsa**, **rsa1** y **dsa**) y se suelen almacenar bajo */etc/ssh*. Las más corriente suelen ser *ssh_host_**rsa**_key* y *ssh_host_**dsa**_key* a las que se le añadde la extensión .pub para sus coincidentes claves públicas. Quizás haya sistemás que cuenten con esta otra: ssh_host_**rsa1**_key

Si no disponemos de la privada y pública (**rsa1**) y no podemos poner el sistema en funcionamiento, podremos probar a utilizar el comando **ssh-keygen** de la siguiente manera:

```
# ssh-keygen -q -t rsa1 -f /etc/ssh/ssh_host_key -C '' -N ''
```

Esto generará tanto la privada como la pública y también es válido para *rsa* y *dsa*.

Aviso: Debemos utilizar permisos restrictivos para los archivos de las claves privadas, normalmente 0600 (rw——-) con propietario root. A las públicas debería de tener acceso todo el mundo.

Accediendo vía SSH

Para acceder a un servidor remoto bajo *ssh* es tan sencillo como instalar el paquete *OpenSSH client* y ejecutar el siguiente comando:

```
$ ssh usuario@maquina
```

A continuación, si es la primera vez que accedemos al sistema remoto se nos pedirá confirmación, de manera que tras introducir la clave del usuario, la clave pública de la máquina remota será guardada en nuestro llavero de claves públicas (*~/.ssh/known_hosts*). El resto de las veces no se nos pedirá confirmación pero si la clave del usuario remoto.

Este sería el método normal para acceder bajo SSH a un sistema remoto. No obstante, existen otras formás para acceder sin hacer uso de una password, de manera que podamos loguearnos a través de por ejemplo tareas automatizadas con shell scripting.

Uso de ssh-Agent

Otra opción para conectar es emplear el programa **ssh-agent** el cual solicitará una contraseña (*passphrase*) para iniciar las conexiones.

Para utilizar *ssh-agent* deberemos de seguir unos pasos sencillos pero necesarios:

1 . Desde el sistema cliente (el que utilizaremos para conectarnos al servidor remoto) generaremos una clave *SSH* versión 2 (*rsa y dsa, o rsa1* para versión 1):

```
$ ssh-keygen -q -t rsa -f ~/.ssh/id_rsa -C ''
```

2 . Se nos pedirá una frase (*passphrase*) de contraseña la cual será la que utilicemos para futuras conexiones:

3 . Ahora será el momento de acceder al servidor remoto:

```
$ ssh usuario_remoto@servidor
```

Escribiremos la contraseña del **usuario_remoto** accediendo ya al servidor. Ahora copiaremos el contenido de nuestro archivo *id_rsa.pub* (creado en el paso 1) al final del archivo *~/.ssh/authorized_keys* o *~/.ssh/authorized_keys2* (anillo de claves públicas) del directorio */home* de **usuario_remoto**. Si no existiese ninguno de estos dos archivos deberíamos de crear uno de ellos.

Nota: Puede que en algunos sistemás tengamos que cambiar los permisos del archivo *~/.ssh/authorized_keys* y directorios. Normalmente es suficiente con eliminar los permisos de escritura (0600) y asegurándonos de que el propietario es **usuario_remoto**.

4 . Ahora ya podremos usar *ssh-agent* desde el sistema cliente. Para ello ejecutaremos

```
$ ssh-agent /bin/bash
```

Esto abrirá un nuevo shell dentro del que hemos usado para ejecutar este comando. No notaremos nada, solo si vemos los procesos ejecutados podremos ver el nuevo proceso creado:

```
[redacted]  7813  7783  2 11:48 pts/2    00:00:00 /bin/bash
[redacted]  7814  7813  0 11:48 ?        00:00:00 ssh-agent /bin/bash
```

Desde esta '*sesión*' escribiremos primeramente **ssh-add ~/.ssh/id_rsa** para añadir nuestra clave *RSA* al conjunto gestionado por *ssh-agent*. Se nos pedirá que introduzcamos la frase/clave que introducimos al crear el par de claves en el paso 1.

Aviso: Cada vez que queramos volver a realizar accesos al sistema remoto al que hemos facilitado nuestra clave pública no deberemos de dar la contraseña del usuario remoto, pero si que tendremos que repetir el paso 4.

Si utilizamos este sistema con frecuencia desde nuestro usuario local, podemos hacer que cada vez que se inicia *bash* lo haga ejecutando *ssh-agent*. Esto podemos lograrlo modificando */bin/bash* por **ssh-agent /bin/bash** en la línea de nuestro usuario del archivo */etc/passwd*.

Acceder sin contraseñas

Hemos pasado de acceder a un host remoto bajo credenciales (usuario y contraseña) de un usuario específico de ese sistema, a acceder con un usuario del sistema pero sin que se nos solicite más clave que una propia creada cuando generamos nuestra clave pública. Lo siguiente es ya acceder directamente sin esa clave, es decir, conectaremos contra un host remoto sin que se nos solicite ningún tipo de password, ni la del usuario remoto, ni la nuestra propia.

Para conseguir esto, podremos recurrir a estas dos opciones:

A. Realizaremos el paso 1, y 3 vistos anteriormente cuando utilizábamos *ssh-agent*. En este caso el paso 1 debe modificarse añadiendo la opción **-N ‘ ‘** quedando así:

```
$ ssh-keygen -q -t rsa -f ~/.ssh/id_rsa -C '' -N ''
```

Nos hemos saltado el paso 2 puesto que con este comando no se nos solicitará ninguna clave. En el paso 3 todo igual, el usuario de la máquina remota (**usuario_remoto** en nuestro ejemplo) debe tener el contenido de nuestra clave pública en su archivo de autorización, normalmente *~/.ssh/authorized_keys*

Si ahora salimos del servidor remoto y volvemos a entrar no se nos pedirá ningún tipo de contraseña.

B. Esta forma es aún más sencilla. Haremos uso del comando **ssh-keygen** y de **ssh-copy-id** (con este último, evitaremos tener que entrar al servidor remoto y copiar el contenido de nuestra clave pública en el archivo de autorización del *usuario_remoto*, es decir, realizar el paso 3 de *ssh-agent* de forma automática)

1 . Creamos nuestro par de claves:

```
$ ssh-keygen -b 4096 -t rsa
```

donde:

-b genera una clave de 4096 bits, por defecto (si no usamos -*b*) se generará de 2048, que suele ser suficiente.

-t indica el tipo de clave, en nuestro caso *rsa* para protocolo 2

Nota: Deberemos de pulsar tres veces *[ENTER]*

2 . Copiamos de forma automática nuestra clave pública en un equipo remoto para un usuario específico, tal que así:

```
$ ssh-copy-id usuario@ip_del_equipo
```

En este paso se nos pedirá el password de “usuario”. Con esto habremos acabado. El resto de veces que queramos acceder a esa máquina con ese usuario, NO, se nos solicitará *password* ni *passphrase*. **Esto resulta muy útil cuando tenemos tareas programás con shell scripts que pretenden acceder a un servidor remoto de forma automática.**

Encriptar y firmar datos con GPG

GPG (*GNU Privacy Guard*) es una herramienta de cifrado y firmás digitales de código libre implementada de la herramienta propietaria **PGP** (*Pretty Good Privacy, Privacidad bastante buena*). Con *GPG* podremos por ejemplo, encriptar mensajes de correo electrónico, de manera que mantengamos una cierta privacidad en cualquier punto de la red que este atraviesa, además de poder firmarlo digitalmente. El firmar un mensaje implica que el destinatario del mismo pueda verificar mediante su herramienta *GPG* que el contenido no haya sido alterado.

Nota: *GPG* se utiliza principalmente para proteger y verificar correos electrónicos, por lo que la mayoría de clientes de correo de *Linux* proporcionan interfaces *GPG*

Generar, exportar, importar y recovar claves con GPG

Al igual que ocurre con las claves *SSH*, necesitaremos una clave privada y una clave pública *GPG*. Con la clave privada firmaremos los mensajes mientras que con la pública los lectores del mismo podrán realizar la verificación de la misma o bien, podremos encriptar un mensaje con la clave pública de un destinatario de manera que solo él mediante su clave privada pueda descifrarlo.

Generar claves

Para generar un par de claves utilizaremos el comando **gpg** con su opción *–gen-key* tal que así:

```
$ gpg --gen-key
```

Tras responder a una serie de preguntas relacionadas con la clave, como por ejemplo el nombre completo de la persona, dirección de correos, contraseña (muy importante recordarla y no perderla), etc… se crearán las claves en el directorio *~/.gnupg*.

Nota: Es posible que generar la clave tarde un poco, esto irá en función del tamaño de clave que hayamos puesto (por defecto 2048 bits) y la rapidez de generación de números aleatorios de nuestro sistema (*entropía*), podemos forzar esto moviendo continuamente por ejemplo el ratón. Sería interesante que se buscara información de como generar la *entropía*, no obstante tampoco es un requisito indispensable el saber esto.

Ahora podremos usar la opción **-k** de *gpg* para ver el *ID* de la clave, algo que no será útil para exportarla.

```
$ gpg -k
pub 2048R/BC9DAD91 2015-03-28 [expires: 2015-03-30]
uid 4sysadmins <nebul4ck@4sysadmins.com>
sub 2048R/24D3B0D1 2015-03-28 [expires: 2015-03-30]
```

Exportar las claves

Una vez generadas las claves podremos exportarlas de manera que podamos crear una clave pública para cada usuario que quiera descifrar o verificar nuestro correo, o bien encriptar sus mensajes con ella para que solo nosotros bajo nuestra clave privada podamos leerlo.

Para exportar una clave basta con utilizar el comando *gpg* pero esta vez con su opción **–export**. Le pasaremos como parámetro el *ID* del propietario, normalmente el nombre o correo electrónico que introducimos al crear la clave, o bien el número *BC9DAD91* que tenemos en la línea **pub** de la salida del comando anterior (este ID es solo un ejemplo, deberemos de reemplazarlo por el que generemos nosotros). Podremos utilizar el nombre o correo electrónico del usuario al que irá destinada nuestra clave publica o un identificador global si la vamos a compartir con un departamento, organización, etc… o bien utilizar nuestro nombre o correo de manera que el destinatario relacione la clave con nosotros.

```
$ gpg --export nuestro_ID > nebul4ck-gpg.pub
```

o

```
$ gpg --output nebul4ck-gpg.pub --export nuestro_ID
```

Con este comando estaremos generando una clave pública (nebul4ck-gpg.*pub*) la cual se creará en el directorio en el que estamos trabajando y que posteriormente distribuiremos para que puedan verificar nuestros correos/documentos o encriptar sus mensajes para que solo nosotros podamos descifrar.

Si teníamos pensado enviar nuestra clave pública por correo, podríamos haber añadido la opción **–armor** al comando *gpg –export* de manera que se generara una salida *ASCII*.

Otras formás de distribuir nuestra clave puede ser como adjunto en un *email*, subirla a un sitio Web o a través de un servidor de claves. Un servidor de claves es un servidor de red que funciona como un depósito donde se almacenarán las claves públicas a distribuir. Podemos almacenar nuestra clave pública en un depósito de claves de la siguiente manera:

```
$ gpg --keyserver pgp.rediris.es --send-keys ID_denuestraclave
  gpg: sending key BC9DAD91 to hkp server pgp.rediris.es
```

Este comando enviará nuestra clave al servidor de claves *pgp.rediris.es* (servidor de claves públicas) para que posteriormente pueda ser descargada por algún interesado.

Aviso: Para crear claves públicas anteriormente utilizamos el **uid** (nombre o correo electrónico) o bien el código de la línea **pub** de la salida del comando *gpg -k*. Para subir una clave pública a un depósito de clave solo servirá el ID de la linea **pub**.

Nota: Existen diferentes servidores públicos de claves *gpg* públicas. Es muy importante generar un certificado de revocación de clave si hemos decidido enviar nuestra clave a un servidor, ya que está será distribuida en pocos minutos al resto del mundo. Veremos esto en el aparatado "*Revocando claves en desuso*"

Importando claves de terceros

Ahora que hemos visto como exportar nuestra clave, es igualmente importante saber importar la clave de un usuario con el que queremos mantener una cierta privacidad, ya sea porque vaya a firmar documentos que nos interese corroborar que vienen emitidos por el sin ser alterados, o porque vayamos a encriptar información mediante su clave pública de manera que solo él con su clave privada pueda descifrar.

Podremos añadir la clave de un tercero a nuestro depósito de la siguiente forma:

```
$ gpg --import <nombre-archivo>
```

O bien, importarla desde un servidor de claves:

```
$ gpg --key-server <nombre_host> --recv-keys <id_clave>
```

De la misma manera que comprobamos la existencia de nuestra clave, podremos verificar el **uid** e **id pub** de la nueva clave agregada:

```
$ gpg --list-keys
```

Nota: Esto es igual que emplear la opción **-k**

Revocando claves en desuso

Cuando generamos un nuevo par de claves es importante el crear acto seguido un *certificado de revocación* de clave. Imaginemos que queremos dejar de usar nuestra clave pública por el motivo que sea, por ejemplo porque se encuentre en un depósito de claves público y nuestra clave privada se ha visto comprometida o simplemente hemos olvidado nuestra contraseña de clave y no queremos que se haga más uso de ella. Una clave pública revocada puede ser utilizada para verificar firmás realizadas en el pasado o descifrar mensajes que hayan sido cifrados antes de la revocación, siempre y cuando el usuario todavía tenga acceso a su clave privada. Lo que no será posible es cifrar datos una vez la clave sea revocada.

Para crear el certificado de revocación haremos lo siguiente:

```
$ gpg --gen-revoke <ID_clave>
```

Podemos añadir la opción *–output <nombre_archivo.gpg>* si en vez de mostrar el certificado por la *stdout* se añada a un archivo.

Una vez tengamos generado el certificado deberemos de copiar y pegar el texto mostrado por la *stdout* en un archivo (en caso de que no hayamos agregado la opción –output directamente). Este será nuestro certificado de revocación. Si llega el momento de revocar la clave deberemos de importar este certificado en nuestra clave, bastará con:

```
$ gpg --import <nombre_archivo.gpg>
```

Deberemos de distribuir esta revocación a los usuarios que estaban haciendo uso de nuestra clave. Si la teníamos subida a un depósito de claves público, deberemos de volver a usar el comando *gpg* con las opciones *–keyserver y –send-keys* para importar nuestra clave revocada.

Cifrar y descifrar documentos

Para **cifrar** un documento con la clave pública de un usuario (por ejemplo *jjgomez*) desde el usuario *nebul4ck* podremos hacer lo siguiente:

1 . Se supone que ta tenemos importada a nuestro anillo de claves la clave pública de *jjgomez* por lo que vamos a comprobar su ID:

```
$ gpg -k
/home/nebul4ck/.gnupg/pubring.gpg
------------------------------
pub 2048R/BCK2ADAK 2015-03-28 [expires: 2015-03-30]
uid nebul4ck <nebul4ck@freeser.com>
sub 2048R/2AD3B0D9 2015-03-28 [expires: 2015-03-30]
pub 2048R/5GJ945FD 2015-03-30 [expires: 2015-04-01]
uid Jose Julio Gómez (Mi mail) <jjmgomez@gmail.com>
sub 2048R/CB851754 2015-03-30 [expires: 2015-04-01]
```

2 . Ciframos de la siguiente manera (usaremos el *nombre*, *correo* o *ID* de *jjgomez*)

```
$ sudo gpg --output contabilidad_2015.dot.gpg --recipient 5GJ945FD --
encrypt contabilidad_2015.gpg
```

De esta manera obtendremos un archivo binario, por lo que no podremos pasarlo directamente por *email* a menos que sea como adjunto. Podremos añadir la opción **–armor** para codificarlo en *ASCII* y poder copiar y pegar el texto en un correo.

```
$ sudo gpg --output contabilidad_2015.dot.gpg --recipient 5GJ945FD --
armor --encrypt contabilidad_2015.dot
```

Ahora vamos a **descifrar** el documento desde el usuario *jjgomez*:

```
$ sudo --output contabilidad_2015.dot --decrypt
contabilidad_2015.dot.gpg
```

Nos pedirá la contraseña (la que introdujo en este caso *jjgomez* cuando generó su par de claves). Y listo! ya podremos leer el documento.

Firmar mensajes y verificar firmás

Ahora, vamos solamente a firmar el documento anterior (*contabilidad_2015.dot*) de manera que nuestro destinatario (*jjgomez* en nuestro caso) pueda verificar que se trata de un documento emitido por nosotros.

Para firmar un documento podremos usar las opciones **–sign** o **–clearsign** de *gpg*. La diferencia entre ambas opciones es que *–sign* además de firmar el documento, lo encriptará con la clave privada, de manera que el destinatario (*jjgomez*) podrá verificar que se trata de un documento procedente de nosotros mediante la opción **–verify <archivo>** pero no podrá leerlo a menos que lo descifre antes con *–decrypt*. Si utilizamos *–clearsign* el documento irá sin cifrar, por lo podrá leerse sin problemás y comprobar igualmente mediante *–verify* que el documento fue emitido por *nebul4ck*.

Utilizar *–sign* provocará que se cree un archivo encriptado con el mismo nombre que el original pero con extensión **.gpg**. Si utilizamos *–clearsign* el archivo resultando tendrá la terminación **.asc**

```
$ gpg --clearsign contabilidad_2015.dot
$ gpg --verify contabilidad_2015.dot.asc
```

Si queremos profundizar algo más en el uso de *gpg* podremos acudir a la siguiente dirección https://www.gnupg.org/gph/es/manual.html, y además leer la página man de *gpg*.

COMANDOS LPCI
COMPLEMENTARIOS

COMANDOS LPIC-1

Capítulo 1

Comandos para la ayuda de otros comandos

man : Es el paginador del manual del sistema. A menos que se especifique, man devuelve la primera página encontrada del comando argumentado. Por lo tanto también nos permite especificar la sección concreta de un comando, especificando su número.

man [opciones] [sección] comando

- Desplegar información sobre el uso del comando passwd:

```
$man passwd
```

- Desplegar información sobre el formato del archivo /etc/passwd

```
$man 5 passwd
```

[*Secciones*]:

- 1. Programás ejecutables y comandos de consola
- 2. Llamadas del sistema proporcionadas por el kernel
- 3. Llamadas de biblioteca proporcionadas por bibliotecas de programás
- 4. Archivos de dispositivos (almacenados generalmente en /dev)
- 5. Formato de archivos
- 6. Juegos
- 7. Miscelánea (paquetes de macros, convenciones,etc…)
- 8. Comandos de administrador del sistema (programás ejecutados en su mayor parte o exclusivamente por root)
- 9. Rutinas del kernel

[*Opciones*]:

- Presentar secuencialmente todas las páginas de manual para el comando especificado: **-a**
- Realizar una búsqueda de palabras clave pasada como patrón entre las descripciones breves de las páginas man y presenta las páginas que contienen dicho patrón (= *apropos*): **-k**
- Es igual a '*man -k*' pero hay que ser exactos en la elección del patrón, es decir pasar el nombre completo del comando a buscar, desplegará entonces todas aquellas páginas man que coincidan con ese comando (= *whatis*) : **-f**

- Mostrar la ruta para la página man del comando pasado como argumento: **-w**

info : El comando info utiliza un formato de hipertexto, lo que nos permite movernos entre las secciones de un comando.

- Mostrar información rápida sobre el comando cd:

```
$info cd
```

help : No todos los comandos disponen de una ayuda rápida, pero es útil, concreta y concisa.

```
$passwd --help
```

history : No es precisamente un comando con el que encontraremos la forma de utilizar un comando ni información sobre él, pero si que podremos comprobar los comandos que se han ido ejecutando en el *shell*. Sin opciones, *history* desplegará la lista de comandos ejecutados en *bash* por el usuario. Si cambiamos de usuario veremos que la lista es otra, ya que los archivos de historia son independiente a cada usuario, por defecto se usa el valor de la variable *$HISTFILE* que apunta a */home/<usuario>/.bash_history*. Podemos pasar como argumento el nombre de otro archivo o modificar el valor de *$HISTFILE* de manera que utilicemos otro archivo distinto por defecto. Si pasamos un número como argumento, *history* solo desplegara los '**n**' últimos comandos ejecutados. Existen diferentes variables con las que podremos configurar el historial de comandos de *bash*. Por ejemplo por defecto solo se muestra una lista de comandos ejecutados precedidos por un número que indica el número de comando, pero podemos hacer que además, nos muestre la fecha y hora a la que fue ejecutado el comando, algo muy útil en sistemás en los que se comparte un usuario. Para lograr esto podríamos exportar en el archivo *~/.bashrc* del usuario la siguiente variable:

```
export HISTTIMEFORMAT="%d/%h - %H:%M:%S "
```

Con lo que conseguiríamos un aspecto de líneas de *history* como esta:

```
304 23/feb - 14:29:55 info cp
```

Otras variables importantes son aquellas que definen el número de comandos y de líneas a guardar en el archivo del histórico, estas variable son *$HISTSIZE* y *$HISTFILESIZE* respectivamente. Como valor recibirán un único número que indicarán los comandos y el número de líneas a guardar.

Podemos desactivar el uso del historial de comandos si damos el valor 0 a la variable *$HISTSIZE*

La sintaxis del comando *history* es básica es decir solo acepta opciones tras el comando y alguna de ellas podrían ser:

- Los comandos ejecutados en la sesión actual no son añadidos al archivo de histórico hasta que salimos de la sesión pero si queremos hacerlo de inmediato podemos usar la opción **-a**
- Escribir la historia en curso en el archivo de histórico sobreescribiendo el contenido: **-w**
- Ejecutar un comando determinado concretando su número de línea en el archivo histórico: **!**<*num_línea*>
- Eliminar una línea concreta de nuestro archivo histórico: **-d** <*num_línea*>
- Limpiar el archivo de histórico: **-c**

fc : Otro comando que puede ayudarnos a leer el historial de comandos ejecutados y a ejecutar uno en concreto. Si ejecutamos *fc* sin opciones, se abrirá el editor de texto que tengamos seteado en la variable **FCEDIT** o **EDITOR** (si *FCEDIT* no está definida) mostrando la última orden ejecutada. En caso de no tener ninguna de las dos *Vi* será el editor ejecutado. Si queremos mostrar los últimos comandos ejecutados por *stdout* (hasta un máximo de 16) bastará con pasar la opción **-l**. Podemos invertir el orden acompañando *-l* de **-r**.

Otras opciones:

- Ver los últimos ejecutados a partir de la orden número 60 en adelante (hasta el último ejecutado):

```
$ fc -l 60
```

- Ver el rango de comandos ejecutados que va desde la línea número 300 a la 400 del archivo ~/.bash_history:

```
$ fc -l 300 400
```

- Ver los últimos 10 comandos ejecutados:

```
$ fc -l -10
```

- Ejecutar un comando pasando el número que a él corresponde en el archivo histórico:

```
$ fc -s <número>
```

watch: Este comando nos permite ejecutar un comando concreto de forma reiterada. Esto nos permitirá monitorizar la salida del comando cada cierto tiempo (es similar a **tail**). *watch* se ejecuta por defecto cada 2 segundos. Algunos ejemplos:

- Monitorizar el contenido del directorio actual cada 2 segundos (opción por defecto). Los archivos nuevos o eliminados se irán viendo como aparecen o desaparecen por la pantalla (*stdout*):

```
$ watch ls -l
```

- Resaltar los cambios ocurridos entre la ejecución anterior del comando y la actual: **-d**
- Monitorizar el sistema identificando los cambios entre ejecución y ejecución:

```
$ watch -d -n 10 vmstat –sSM
```

Redirecciones y canalizadores

> : Redirige la salida estándar a un archivo. Lo crea nuevo.

```
$echo “Crear archivo con este texto” > nuevo_archivo.txt
```

>> : Redirige la salida estándar a un archivo añadiendo la nueva info, al contenido ya existente.

```
$echo -e “\nAñade esto al texto anterior” >> nuevo_archivo.txt
```

2> : Crea un archivo que contiene el error estándar. Si el archivo existe, se sobrescribe.

```
$cat /home/nebul4ck/archvo.txt 2> ./stderr.txt
```

Nota: Suponemos que *archivo.txt* esta mal escrito, entonces no será encontrado, por lo que debería de mostrar algo como: *cat: archvo.txt: No existe el archivo o el directorio*, pero no es así ya que esto se encuentra ahora dentro del archivo *stderr.txt*

Truco: Podemos enviar los errores estándar a “ningún sitio” así:

```
$comando 2> /dev/null
```

2>> : Añade el error estándar al archivo existente

Con el caracter & podemos redirigir salidas de un tipo hacia otras, por ejemplo el uso más básico es **&>** *archivo* que crea un archivo nuevo que contiene tanto la salida estándar como el error estándar. Si el archivo existe, se sobrescribe. Otro ejemplo algo más complejo es **2&>1** *archivo* que redirige la salida de error hacia la salida estándar.

< : Envía el contenido del archivo especificado, para utilizarlo como entrada estándar.

<< : Acepta el texto de las lineas siguientes como entrada. Se inicia y finaliza el texto utilizando la misma palabra.

```
$echo << PATRON
> Este texto se mostrará por pantalla
> y finalizará cuando yo vuelva a escribir
> PATRON
```

<> : Utiliza el mismo archivo tanto para la entrada como para la salida.

Tee: Es la herramienta de redirección definitiva, divide la entrada estándar para que se muestre tanto por la salida estándar y en tantos archivos como especifiquemos.

```
$ls -lt /home/nebul4ck/listado.txt | tee salida.txt
```

| : El uso de pipes es importantísimo si tratamos de usar la salida de un comando como la entrada de otro (concatenación de comandos).

```
$ps -ef | grep term
nebul4ck 2420 2035 0 dic16 ? 00:00:16 mate-terminal
```

xargs : Con *xargs* podemos generar líneas de comandos, es decir podemos pasar un comando que despliegue una serie de líneas y ejecutar un comando por cada una de ellas.

```
$find ./ -type f -name “*.txt” | xargs -d “\n” rm
```

Nota: El comando *find* nos mostrara tantas líneas como *archivos.txt* encuentre en el directorio y a través de | canalizaremos esa salida para que *xargs* analice cada una de sus líneas (esto se consigue usando la opción **-d**, indica que queremos usar como delimitador, en nuestro caso el salto de línea) aplicándole el comando pasado a *xargs* que es **rm**, es decir se eliminarán todos los archivos *.txt* de ese directorio.

exec : El comando *exec* además de ejecutar programás externos (que pueden ser ejecutados, incluso escribiendo unicamente el nombre del programa) posee una funcionalidad adicional, y es que el nuevo proceso reemplaza a la consola, es decir *exec* se hace con el ID de la *shell* en la que ha sido ejecutado en vez de abrir un nuevo proceso, de manera que cuando el comando termine su ejecución, la consola se cerrará. Un uso común de *exec* es junto con otros comandos formando una cadena de comandos o dentro de scripts.

```
$find . -type f -name "*sed" -exec rm {} \;
```

find se encarga de encontrar un(os) archivo(s) basado en unos criterios y entonces se ejecuta **exec** que a su vez llama a **rm** para que sustituya los archivos encontrados dentro de **{}** y los elimine. Esto se podría haber realizado de forma más simple usando el metacaracter acento grave del *shell*, que devuelve el resultado de la ejecución de un programa, en este caso el resultado de *find* lo pasa como argumento a **rm**.

```
$rm `find . -type f -name “*sed”`
```

Otro uso que podemos hacer de *exec* es el siguiente:

```
exec &> /nombre/archivo
```

Si usamos esto dentro de un *shell script*, nos enviará la salida estándar y el error estándar de lo que se ejecute mediante ese *script* al archivo /nombre/archivo, “digamos que genera un log”.

Nota: Veremos otros usos de *exec* en el Capítulo 9 “*El entorno de Consola, Shell Scripts, el Correo Electrónico y uso básico de SQL*” a la hora de saber como ejecutar un *shell script* o programa.

Comandos Combinar archivos

cat : Nos permite combinar varios archivos en uno solo, además también sirve para mostrar el contenido de un determinado archivo si se le pasa como argumento el número de uno o varios archivos seguidos. El comando tac hace esta tarea pero invirtiendo el contenido.

```
$cat primero.txt segundo.txt tercero.txt
```

Nota: Imprimirá de forma secuencial el contenido de los archivos primero, segundo y tercero.

```
$cat primero.txt segundo.txt tercero.txt > todo_en_uno.txt
```

Nota: Combinará el contenido de primero segundo y tercero en todo_en_uno.txt. Se puede usar los otros elementos de redirección de igual moto, obteniendo resultados diferentes.

[*Opciones*]:

- Añadir el número al comienzo de cada línea : **-n**, *–number*
- No cuenta las líneas en blanco: **-b** , *–number-nonblank*
- Añade "$" al final de cada línea: **-E**, *–show-ends*
- Mostrar caracteres especiales: **-T** (TABs), **-v** (el resto de caracteres especiales.)
- Minimizar lineas en blanco sencuenciales a una: **-s**, *–squeeze-blank*

join : Combina archivos a partir de una columna en común. Los archivos deben de tener el mismo orden, podemos usar *sort* para tal fin. Similar al *join* de SQL. Imaginemos que en un archivo tenemos Nombres de personas en una columna y *DNIs* en otra, y otro archivo con números de Teléfono y Nombres de persona, la columna en común es el nombre de personas, con join podremos tener un archivo que contenga *DNI* Nombres de persona Teléfonos.

```
 $join fichero1.txt fichero2.txt
```

[*Opciones*]:

- Usar otro carácter separador distinto al espacio (por defecto): **-t** *char*
- Ignorar mayúsculas de minúsculas: **-i**
- Cambiar el campo común (por defecto el 1º campo del archivo):

```
$join -1 3 -2 1 archivo1.txt archivo2.txt
```

Nota: Usará el campo 3 del 1º archivo y el campo 1 del 2º

- Rellenar en blanco (empty) campos que faltan: **-e**

paste : Lista las columnas de archivo2 paralelas a las de archivo1, para que sean coincidentes, ambos archivos deben de tener el mismo número de líneas:

```
$paste archivo1.txt archivo2.txt
```

[*Opciones*]:

- Utilizar caracteres de *char* en vez de TABs (por defecto):

```
$paste -d char archivo1.txt archivo2.txt
```

- Mostrar ambos archivos de forma consecutiva en vez de en paralelo: **-s**

Comandos para transformar archivos

expand : Convierte tabulaciones en espacios, por defecto *expand* presupone que 8 espacios equivalen a 1 tabulación.

[*Opciones*]:

- cambiar el número de espacios por tabulación (por defecto 8): **-t** *num* (**–tabs=***NUM*)

unexpand : Es el opuesto a *expand*, es decir sustituye varios espacios por tabulaciones. Sirve para comprimir el tamaño del archivo que contenga muchos espacios, acepta las mismás opciones que *expand*

od : Muestra un archivo en octal (por ejemplo archivos de música que no son fáciles de mostrar en ASCII). od también permite representar archivos en hexadecimal o en decimal.

[*Opciones*]:

- Representar en octal: **-b**
- Representar en ASCII: **-c**
- Representar en decimal: **-d**

sort (ordenar) : Nos permite ordenar la salida de un archivo.

[*Opciones*]:

- Ignorar mayúsculas y minúsculas (por defecto sort distingue mayúsculas de minúsculas): **-f**
- Ordenar por la abreviatura de los meses (JAN-DEC): **-M** (*–month-sort*)
- Ordenar por números: **-n** (*–numeric-sort*)
- Comprobar si el archivo está ordenado: **-c**
- Combina los archivos listados en uno solo (deben de estar previamente ordenados): **-m**
- Invertir el orden por defecto: **-r** (*–reverse*)
- Especificar el campo por el que se quiere ordenar (por defecto el primer campo): **-k** *campo* (*–key=campo*)

Nota: Se puede ordenar por varios campos separándolos por comás

- Ignorar campos en blanco a la izquierda: **-b**
- Considerar solo espacios en blancos y caracteres alfanuméricos: **-d**
- Considerar solo caracteres imprimibles: **-i**

split : Divide un archivo en 2 o más archivos. Normalmente hay que indicar un nombre de salida al que se le añadirán los sufijos (bien numéricos o alfabéticos).

[*Opciones*]:

- Dividir un archivo por bytes: **-b** *tamaño* (*–bytes=tamaño*)

Nota: Puede cortar el archivo por la mitad de una línea.

- Dividir en bloques de bytes determinados por las líneas: **-C=***tamaño* (*–line-bytes=tamaño*)

Nota: Si el tamaño de la línea es superior al indicado, esta, se dividirá en varios archivos, anulando la acción de este parámetro

- Dividir por número de líneas: **-l** *num_lin* (*–lines=lineas*)
- Determina el máximo de caracteres para el sufijo (por defecto el sufijo es de 2 caracteres): **-a** *num_sufijo* (*–sufix-length=num_sufijo*)

Nota: $split -l 2 -a 3 arc1.txt partes , tendremos como resultado partes de 2 líneas con sufijos de 3 caracteres-> partesaaa partesaab…

- Utilizar sufijos números: **-d** (por defecto comienzo desde 0)

tr : cambia caracteres individuales por otros. Con el siguiente ejemplo se comprenderá fácilmente.

```
$tr BCJ bc < archivo1.txt
```

Nota: Los caracteres BC que aparezcan en *archivo1.txt* serán sustituidos por bc, por el contrario los que aparezcan como J, serán sustituidos por c. Esto ocurre porque GRUPO1 es mayor que GRUPO2.

[*Opciones*]:

- Si queremos cortar GRUPO1 en función de GRUPO2 (para que no se produzca la sustitución de la J como en el ejemplo anterior): **-t** (*–truncate-set1*)
- Reducir a 1 caracteres consecutivos encontrados en el texto y que son definidos en GRUPO1: **-s** (*–squeeze-repeats*)

Ejemplo:

```
$tr -s r R < archivo1.txt : Si en archivo1 existe esto: error, el
resultado será eRoR
```

- Para eliminar de *archivo1.txt* todos los caracteres (BCJ), usamos: **-d**

Nota: Podemos prescindir de GRUPO2, ya que no se tendrá en cuenta.

tr permite el uso de atajos y uso de caracteres especiales.

[*Atajos*]:

- **** : barra invertida
- **\n** : nueva linea
- **\t** : tabulación horizontal
- **\v** : tabulación vertical
- **A–M** : Caracteres de la A a la M
- **[:alnum:]** indicar este atajo en un GRUPO tendrá como consecuencia el incluir todos los números y letras
- **[:upper:]** todas las letras mayúsculas
- **[:lower:]** todas las letras minúsculas
- **[:digit:]** todos los dígitos
- **[:alpha:]** todas las letras
- **[:space:]** todos los espacios en blancos

uniq : Elimina las líneas duplicadas de un archivo. Útil si se usa como entrada (a *uniq*) desde la salida de otro comando (*sort* por ejemplo):

```
$sort archivo1.txt |uniq
```

[*Opciones*]:

- Preceder la línea con el nº de concurrencias: **-c**
- Mostrar solo las líneas repetidas: **-d**
- Ignorar mayúsculas de minúsculas: **-i**
- Mostrar solo las líneas que son únicas: **-u**
- Nota: uniq no detecta líneas repetidas a menos que sean adyacentes.

iconv: Realiza conversiones de codificación entre conjunto de caracteres o archivos.

```
$ iconv -f <codificación_de_origen> -t
<codificación_de_salida_deseada> archivo de entrada
```

Opciones:

- Listar los códigos de caracter conocidos: -l (*–list*)
- Especificar un archivo de salida diferente a la salida estandar: **-o** *<file>* (*–output <file>*)

mpage: Lee archivos de texto plano o PostScript y les aplica un formato que reduce el tamaño por página en caso de querer ahorrar papel.

```
$ mpage -P nombrecoladeimpresión archivo.txt
```

Opciones:

- Cambiar el número de páginas del archivo original a ajustar en cada página impresa (por defecto se reducen 4 páginas del original por 1 impresa): -1, -2, -4 y -8
- Ajusta el tamaño del papel de impresión: -b<tamaño-papel>

Tamaño papel puede ser:

- A3, A4, Letter (catas US) o Legal (estándar) podemos ver más tamaños con la opción **-bl**
- Imprime a doble cara, 1º y 4º página en una hoja y 2º y 3º en otra hoja: **-e**
- Usar una línea horizontal para separa la cabecera de la página con el resto del contenido: **-H**
- Impresión inversa. Comienza a imprimir desde la última página: **-r**

Nota: Existen otras opciones como por ejemplo, cambiar el formato de impresión (vertical por defecto) a horizontal -l, modificar los margenes -m, etc..

Comandos para dar formato a archivos

fmt : Da formato a los archivos que tienen líneas exageradamente largas, longitudes de líneas irregulares ,etc… El formato por defecto del nuevo párrafo es de 75 caracteres de ancho.

$fmt [OPCIONES] archivo

[*Opciones*]:

- Para cambiar el limite de 75 caracteres por linea: **–***ancho*, **-w** *ancho* o **–width=***ancho*
- Para preservar la sangría de las 2 primeras líneas: **-c**
- Para diferenciar la sangría de la primera línea con respecto a la segunda: **-t**

nl : Nos permite enumerar las líneas de un archivo de forma más sofisticada/compleja que con *cat -n*

$nl [opciones[ESTILO|FORMATO]] archivo

[*Opciones*]:

- Podemos hacer que nl identifique una nueva página: **-d=***código*, *–section-delimiter=código*
- Para enumerar las líneas del cuerpo del archivo: ***–bESTILO***, *–body- numbering=ESTILO*
- Para enumerar los pies de páginas: ***–fESTILO***, *–footer-numbering=ESTILO*
- Para enumerar las cabeceras: ***–hESTILO***, *–header-numbering=ESTILO*
- Incrementar el valor del número de línea (por defecto 1): ***–iNUMERO***, *–line-increment=NUMERO*
- No reajustar el número de líneas por página: **-p**
- Añadir una cadena de texto (si es posible) tras el número de línea: **-s***CADENA*, *–number-separator=CADENA*
- Añadir un tipo de formato: **-n**

[*Estilos*]:

- **a** : enumera todas las líneas
- **t** : enumera solo las líneas no vacías
- **n** : permite ocultar el número de líneas (muy útil en encabezados y piés de páginas)
- **p***HOLA*: enumera solo las líneas que coinciden con HOLA

[*Formatos*]:

- **ln** : justificado a la izquierda
- **rn** : justificado a la derecha
- **rz** : justificado a la izquierda anexando ceros

pr : Prepara un archivo para su impresión añadiendo encabezados (con fecha y hora actual, nombre del archivo y número de página), pies, etc…

$pr [OPCIONES] archivo

Nota: *pr* es muy útil cuando se usa pasándole como entrada la salida de otro comando.

[*Opciones*]:

- Eliminar el encabezado: **-t**, *–omit-header*
- Modificar el encabezado, anulando el nombre de archivo por CADENA: **-h** *cadena*, *–header=CADENA*
- Definir el margen izquierdo: **-o** *num_caracteres*, *–indent=num_caracteres*
- Definir el ancho de página (por defecto 72 caracteres): **-w** *caracteres, –width=caracteres*
- Definir la longitud de la página: **-l** *num_lineas*, *–length=num_lineas*
- Usar saltos de páginas en vez de números de líneas para marcar el final de la pagina: **-F**, **-f**, *–form-feed*
- Generar salida con espacios dobles: **-d**, *–double-space*
- Modifica el formato de fecha del encabezado: **-D** *FORMATO*, *–date- format=FORMAT*

Comandos para la mensajería desde línea de comandos

write : Comando que nos permite enviar mensajes a la consola de un determinado usuario, conectado al mismo sistema.

$write usuario < texto

```
$write usuario
Escribo aquí lo que
quiera que le llegue y luego cierro.
Control-D
```

mesg : Digamos que es el botón que '*mutea*' o activa los mensajes del comando *write* en una terminal (*pts*). Suponiendo que 2 usuarios conversan por terminal, el usuario A tiene abierta 2 terminales (*pts2* y *pts4*) y quiere dejar de recibir mensajes por pts4:

```
$mesg n < /dev/pts/4
```

- Si queremos volver a recibir mensajes por esa terminal utilizamos la opción: **-y**

talk : Conversación bidireccional mediante consola entre 2 usuarios conectados a la misma o diferente máquina.

$talk usuario [ttyn | pts/n]

```
$talk usuario@maquina
```

wall : Útil para enviar un mensaje a todos los usuarios conectados a un sistema Linux.

$wall < archivo_texto

```
$echo "Hola a todos" | wall
```

[*Opciones*]:

- Si queremos quitar el banner del mensaje (solo root puede hacerlo): **-n**
- Si queremos usar una cuenta a tras, por ejemplo para apagar el sistema: **-t** *tiempo*

echo : Muestra por pantalla o envía a un archivo aquello que se escribe/manda por la entrada estándar.

```
$echo -e "Esto se muestra por pantalla con salto de líneas\n"
$echo -n "Añado una línea a un archivo" >> archivo.txt
```

[*Opciones*]:

- Permitir la interpretación de caracteres con barra invertida, por ejemplo, un salto de página, una tabulación o un salto de línea : **-e**
- No inserta el carácter de nueva línea. Útil por ejemplo a la hora de codificar un texto desde consola: **-n**

```
$echo -n “contraseña” | openssl sha1
```

[*Caracteres de escape*]:

- **\n** : representa un salto de línea
- **** : Representa una barra invertida
- **\r** : Representa un retorno de carro
- **\t** : Representa una tabulación horizontal

Otras cosas que podemos hacer con echo:

- Listar el contenido de un directorio (siempre y cuando estemos dentro):

```
$echo /*
/bin /boot /cdrom /dev /etc /home /initrd.img /lib /lib64 /lost+found
/media /mnt /opt /proc /root /run /sbin /srv /sys /tmp /usr /var
/vmlinuz
```

- Enviar un mensaje a una determinada terminal utilizando como salida, la entrada de esa terminal (*pts*). El comando *w* o *who* nos serán de ayuda. Suponiendo que es a pts2 a quien queremos enviar el mensaje:

```
$echo “Esto es un msg para pts2” > /dev/pts/2
```

Nota: El mensaje aparecerá en la línea de comandos de pts2, como si hubiese sido tecleado por él.

Comandos para la visualización de archivos

head : Muestra por defecto las 10 primeras líneas de un archivo.

head [opciones] archivo

[*Opciones*]:

- Podemos cambiar el número de líneas: **-n** *numero*, *–lines=numero*
- Podemos acotar la salida por bytes en vez de por líneas: **-c** *num*, *–bytes=num*

tail : Es exactamente igual a *head*, solo que muestra las lineas finales del archivo. También es posible usar **-c** *num*, *–bytes=num* y **-n** *num*, *–lines num*

[*Otras opciones*]:

- Podemos dejar un archivo de registro abierto para ir viendo lo que se escribe en el: **-f**, *–follow*
- Podemos indicar a tail que finalice el seguimiento del archivo al terminar la ejecución del programa: **–pid=num_pid**

less : Es un visualizador ligero de archivos ya que no necesita cargar la información antes de visualizarla.

less [opciones] archivo

[*Opciones*]:

- Avanzar una página: **ESPACIO** / *Control V* / *Esc ESPACIO* (*full screen*)
- Retrocede una página: *Esc V* / **b** / *Control+B)* / *w [num líneas]*
- Avanzar media página: **d** / *Control+D*
- Retrocedes media página: **u** / *Control+U*
- Avanzar una línea: **ENTER** / **e** / *Control+E* / **j** / *Control+J* (Se pueden especificar N lineas)
- Retrocede una línea: **y** / *Control+P* / **k** / *Control+K*
- Desplazarse por los caracteres: **cursores** (*flechas*)
- (**/**) para buscar texto, **n** para la siguiente concordancia, **N** buscará hacia atrás
- (**?**) para buscar texto hacia atrás
- **g***NUM* iremos a la línea *NUM*
- **h** visualiza la ayuda interna de *less*
- **Q** para salir de *less*

more : Es una versión anterior de *less*, muchas opciones de *less* no funcionan con *more*.

Resumir archivos

cut : Extrae partes de una línea, en forma de fila (línea) y campos (columnas) delimitadas por un carácter

[*Opciones*]:

- Para extraer según los bytes: **-b** *lista*, *–bytes=lista*
- Para extraer por carácter: **-c** *lista*, *–characters=lista*
- Por campos (columnas): **-f** *lista*, *–fields=lista*
- Modificar el delimitador de campos (por defecto el tabulador): **-d** <*delimitador*>. **-d ” “** para utilizar el espacio como delimitador.

[*Lista*]:

Es un único carácter/número o bien un intervalo de estos: “4” , “4-8″ , “-6″ (de 0 a 6) , “2-” (de 2 en adelante)

Ejemplo: Mostrar la *MAC* de *eth0*

```
$ifconfig | grep eth0 | cut -d " " -f 11
```

wc : Nos permite hacer un recuento de: caracteres, palabras, líneas (en función del carácter de salto de línea), bytes y el tamaño máximo de línea.

[*Opciones*]:

- Para contar el número de líneas: **-l**, *–lines*
- Para contar las palabras: **-w**, *–words*
- Para contar los bytes: **-c**, *–bytes*
- Para contar los caracteres: **-m**, *–chars*
- Para ver la longitud máxima de línea: **-L**, *–max-linelength*

Comandos Expresiones Regulares

grep : Busca en archivos cadenas de texto y nos devuelve el nombre del archivo y si además es un archivo de texto nos devolverá la línea que contiene dicho patrón. La sintaxis de *grep* es:

grep [opciones] exp_regular [archivos]

Opciones:

- Si no queremos que nos muestre la línea que contiene el patrón pero si un número que indica cuantas líneas lo contienen: **-c**, **–count**
- Si no queremos hacer distinción entre mayúsculas y minúsculas: **-i**, **–ignore-case**
- Buscar de forma recursiva: **-r**, **–recursive**
- Si en vez de una exp_regular/patrón queremos pasar un archivo con un texto como patrón: **-f**, **–file** <*archivo*> .
- Para usar expresiones regulares extendidas (que contienen metacaracteres como ?, +, |, {},()) :**-E**, **–extended-regexp** . Esto es igual a *egrep*
- Si queremos hacer una búsqueda de varias cadenas de texto (opciones diferentes): **-F**

Nota: *grep -F* es igual a *fgrep*. No se permite el uso de expresiones regulares. Las opciones a buscar deben de ir en líneas separadas. El siguiente ejemplo busca trastero o garaje en un archivo de texto

```
$fgrep trastero
 >garaje archivo.txt
```

- Para negar, o buscar todo lo que no coincida: **-v**
- Mostrar el número de línea en la cual se ha encontrado el patrón (válido para *grep*, *egrep* y *fgrep*): **-n**
- Para mostrar líneas donde el patrón no es parte de una palabra si no una palabra completa: **-w**
- Devolver el nombre del archivo en el que se ha encontrado el patrón y no sus líneas: **-l**

Ejemplos:

Busca de forma recursiva en /etc/ todos los archivos que contienen la palabra eth0 o eth1

```
$grep -r eth[01] /etc/*
```

El uso de * implica que pueda haber 0 o más ‘i’ en palabras que contengan como mínimo ‘da’.

```
$grep dai* archivo.txt
```

Usamos la salida de ps como entrada de grep

```
$ps ax | grep inetd
```

En este ejemplo recalcaremos varias cosas, como por ejemplo, podríamos haber usado el comando “egrep” que es igual que usar “grep -E”, el entrecomillado de la expresión regular es vital, de lo contrario el shell podría hacer una mala interpretación de los metacaracteres, y para terminar la explicación de que hace en este caso grep: busca en /etc/ archivos que contengan o bien el nombre de host de alberto o de paula y que además contenga el número 15

```
$grep -E "(alberto\.nebul4ck\.com|paula\.nebul4ck\.es).*15" /etc/*
```

sed : Es un ‘Stream EDitor’ o editor de flujo que no modifica el contenido del archivo original si no una copia e imprime por pantalla las líneas procesadas o aquellas que han sufrido cambios (esto depende de la opción que le pasemos). La sintaxis de sed es:

sed [-n] [-e comando] [-f archivo-comandos] archivo [> guardar_cambios]

Explicación de la sintaxis:

- **-n** : Esta opción hace que no se impriman por pantalla las líneas procesadas. Podemos imprimir solo las que han experimentado modificaciones siempre y cuando utilicemos la bandera ‘flag’ p (print) que veremos a continuación.
- **-e** : Con esta opción indicamos que estamos pasando comandos por la línea de comandos y no a través de un archivo. Se puede omitir. Los comandos pueden ir acompañados de banderas que modifican su comportamiento
- **-f** archivo-comandos : Podemos indicar comandos desde un archivo (1 por línea)

Comandos sed:

- **d** : Elimina las filas
- **p** : Imprime las líneas que ha experimentado cambios.
- **q** : quit. Recorrer un archivo e imprimir una línea y sale.

Nota: *p* y *q* unidos es más efectivo, ya que no se recorre todo el archivo *$ sed -n ‘3{p;q}’ archivo.txt*

- **s** : sustitución.
- **a** : Agregar texto
- **i** : Insertar texto
- **c** : Cambiar texto
- r : Leer archivo
- **=** : Muestra el número de línea donde se encontró el patrón.

Podemos pasar varios comandos separados por '**;**' :

```
$sed -n 's/antiguos/MODIFICADO/g;p;=' archivo.txt
```

- O agrupados por '**{}**' cuyo uso es más habitual en archivos de comandos sed :

```
$sed -n '{ s/antiguos/MODIFICADO/g; {p; {=; }}}' archivo.txt
```

Pasar varios comandos desde un script sed:

```
$cat script.sed
s/antiguos/MODIFICADO/g
p
=
$ sed -n -f script.sed archivo.txt
```

Vamos a ver el uso de los comandos sed:

- Sustituir patron1 por patron2. Por defecto sed solo sustituirá la primera coincidencia de cada línea. Si añadimos una g final sustituirá todas las coincidencias de todas las líneas.

```
sed s/patrón1/patrón2/g archivo.txtx
```

- Si queremos solo sustituir la tercera coincidencia de una linea:

```
sed s3/patron1/patrón2 archivo.txt
```

- Si por el contrario queremos sustituir la primera coincidencia de patrón1 en la tercera línea:

```
sed 3s/patrón1/patrón2 archivo.txt
```

- Podemos sustituir todas las coincidencias de patrón 1 que ocurran desde la línea 5 a la línea 9:

```
sed 5-9s/patron1/patron2/g archivo.txt
```

Nota: Podríamos haber usado d y hubiésemos eliminado las líneas donde se hubiesen encontrado patrón 1, si además de la g añadimos la p (print) y la acompañamos con la opción -n de sed, hubiésemos imprimido por pantalla solo las líneas afectadas.

- Indicar líneas mediante patrones en vez de número:

```
sed /palabra_inicio/./palabra_final/s/patrón1/patrón2/g archivo.txt
```

En este caso **sed** busca '*palabra-inicio*' y empezará a sustituir *patrón1* por *patrón2* hasta que encuentre '*palabra_final*'. Tras leer la línea con '*palabra_final*' *sed* proseguirá en búsqueda de una nueva línea que contenga '*palabra_inicio*' repetirá su comportamiento sucesivamente hasta terminar el archivo.

Nota: Si a este comando le añadimos la opción **-n** pasará algo curioso, y es que si no se encuentra ninguna línea con '*palabra_inicio*' no se imprimirá nada, por el contrario si se encuentran líneas con '*palabra_inicio*' pero ninguna con '*palabra_final*' , se imprimirán todas. Esto es por el flujo de texto que sigue *sed* en su búfer, él sabe que ha encontrado, pero no lo que va a o no va a encontrar.

Cuando usamos los comandos sed dentro de un un archivo de comandos podemos usar la bandera **w** para guardar las líneas modificadas en un archivo:

```
s/patrón1/patrón2/w archivo_salida
```

- Añade la palabra Mundo a la 4º línea de archivo.txt después de la línea o patrón especificado:

```
$ sed '/3/ a\ Mundo' archivo.txt
```

Nota: Si empleamos la dirección **/3/**, *sed* añadirá el texto cada 3 líneas, si usamos 3 a secas, solo lo añadirá una vez en la cuarta línea. Pasa lo mismo con **i** y **c**

- Añade la palabra Mundo a la 4º línea de archivo.txt antes de la línea especificada:

```
$ sed '/5/ i\ Mundo' archivo.txt
```

- Esta línea va a sustituir a la línea que contiene a '*Antigua*' en archivo.txt

```
$ sed  '/Antigua/ c\' archivo.txt
```

- Agrega al final de cada archivo coincidente con '*fich*.txt*' el contenido de firma.txt

```
$ sed -s '$ r firma.txt' fich*.txt
```

- Mostrar el número de línea donde se encontró patrón

```
$ sed -n '/patron/=' archivo.txt
```

Otros ejemplos para limitar líneas:

- Eliminar las líneas 1,2,3,4 y 5 del archivo inventario.txt

```
$sed '1,5d' inventario.txt
```

- Efectúa cambios de la línea 23 a la última del archivo.

```
$sed '23,$s/patrón1/patrón2/g archivo
```

- Indicar que aquellas líneas que comiencen con '**#**' no las muestre

```
$sed '/^#/d' /ruta/mi/archivo.txt
```

- Negar una dirección o patrones, no imprimir las líneas 4,5 y 6:

```
$sed -n '4,6 !p' archivo.txt
```

- No imprimirá la línea que contenga el patrón.

```
$sed -n '/patron/! p' archivo.txt
```

El uso de &

El uso de & es extremadamente útil a la hora se sustituir texto, con el siguiente ejemplo quedará claro:

```
$sed 's/ GPL /& (General Public License)/g mi_texto.odt >
modificado.odt' : Esto es igual a
's/ GPL /GPL (General Public License)/g'
```

Algunos Trucos

- Mientras no se alcance la última línea ($!) imprime el archivo

```
$sed -n '$! p' archivo.txt
```

- Eliminar los espacios al comienzo de cada línea:

```
$sed '/^ *//g' archivo.txt
```

- Sustituir un tabulador por ; :

```
$sed 's/\t/;/g' archivo.txt
```

- Eliminar los saltos de línea (\n, líneas en blanco):

```
$sed '/./! d' archivo.txt ó sed '/^$/d' archivo.txt ó sed -n '/./ p'
```

Referencias:

https://www.gentoo.org/doc/es/articles/l-sed1.xml -> Parte 1

https://www.gentoo.org/doc/es/articles/l-sed2.xml -> Parte 2

http://es.kioskea.net/faq/3063-sed-introduccion-a-sed-parte-i -> Parte 1

http://es.kioskea.net/faq/3060-sed-introduccion-a-sed-parte-ii -> Parte 2

COMANDOS LPIC-1

Capítulo 2

Comandos para la administración de paquetes y dependencias

rpm : Permite la instalación de paquetes RPM.

rpm [acción] [opción] [archivos-paquetes | nombres-paquetes]

[Acciones]:

- Instalar un paquete: **-i** *[–force|-h|–no-deps|–prefix|–test]*
- Actualizar un paquete o lo instala si no existe: **-U** *[–force|-h|–no-deps|–prefix|–test]*
- Actualiza un paquete (solo si ya existe una versión anterior): **-F** *(–freshen)*
- Desinstalar un paquete: **-e** *[–no-deps]*
- Comprobar la suma de verificación de un paquete: **-K**
- Vuelve a generar la base de datos RPM (corrige errores): **–rebuilddb**
- Información sobre un paquete: **-q** *[-a(–all)|-f <archivo>|-R| -l(–list)|-p|-i]*
- Verificar un paquete: **-V** *[-a(–all)|-f <archivo>]*
- Crea un paquete binario a partir de su código fuente y archivos de configuración (ahora *rpmbuild*): **-b**
- Crea un paquete binario a partir de un RPM fuente (ahora *rpmbuild*): **–rebuild**

[Opciones]:

- Forzar la instalación o actualización de un paquete: **–force**
- Mostrar # al inicio de líneas a la hora de instalar o actualizar un paquete: **-h** *(–hash)*
- No realizar comprobaciones de dependencias de un paquete a la hora de instalar, actualizar o desinstalar: **–nodeps**
- Comprobar problemás de dependencias en un paquete (no instala ni actualiza, es una "simulación): **–test**
- Modificar la ruta de instalación o actualización de un paquete (no valido para todos): **–prefix** *new_ruta*
- Cuando se pide información de un paquete (*-q*) o se verifica (*-V*) podemos hacerlo para todos a la vez: **-a** *(–all)*
- Informa de todos los paquetes a los que pertenece un archivo cuando pedimos información (*-q*) o verificación (*-V*): **-f** *<archivo>* (*–file <archivo>*)
- Informa de todos los paquetes de los que depende un paquete en cuestión cuando pedimos información: (*-q*): **-R** (*–requires*)
- A la hora de pedir información sobre un paquete (*-q*) podemos hacerlo de un paquete desinstalado: **-p** *<archivo-paquete>*

- Cuando nos informamos sobre un paquete (*-q*) podemos desplegar una breve información secundaria: **-i**
- Cuando pedimos información de un paquete (*-q*) muestra también los archivos contenidos en ese paquete: **-l** (*–list*)

yum : nos permite descargar e instalar paquetes y sus dependencias desde la línea de comandos.

yum [comando] [paquete(s)]

[Comandos]:

- Instala los paquetes y sus dependencias: **install**
- Actualizar el sistema (si no se le pasa un paquete como argumento) o un paquete concreto a la versión más reciente: **update**
- Una variante de update pero que se ocupa de paquetes obsoletos mucho mejor: **upgrade**
- Instalar archivos RPM locales específicos, utilizando el repositorio de yum para resolver dependencias: **localinstall**
- Actualizar paquetes instalados con localinstall: **localupdate**
- Desinstalar un paquete y sus dependencias: **remove** o **erase**
- Buscar paquetes o semejantes, útil si no conocemos el nombre concreto de un paquete: **search**
- Comprueba si hay actualizaciones disponibles: **check-update**
- Limpiar el directorio caché de yum: **clean**
- Mostrar información sobre un paquete (similar a *$rpm -qi <nombre_paquete>*): **info**
- Mostrar información sobre un paquete (versión instalada, existencia de una nueva versión…): **list**
- Información sobre todos los paquetes y actualizaciones relativos a un programa: **provides** o **whatprovides** (Ej. *$yum provides samba*)
- Mostrar los paquetes que coinciden con la dependencia especificada: **resolvedep**
- Mostrar las dependencias de un paquete concreto: **deplist**
- Pasar al modo consola de *yum* (en él, se pueden pasar varios comandos a la vez): **shell**

yumdownloader : Descarga un paquete pero no lo instala

dpkg: se utiliza para instalar un paquete *Debian*

dpkg [acción] [opción] [nombre-paquete]

[Acciones]:

- Instalar un paquete: **-i** *[–ignore-depends|–no-act|-E|-G|–recursive]*
- Reconfigurar un paquete instalado (lanza el script de postinstalación): **–configure** = **dpkg-reconfigure**
- Borrar un paquete dejando intacto los archivos de configuración: **-r** *[–ignore-depends|–no-act|-B]*
- Elimina un paquete de forma completa: **-P** *(–purge)*
- Muestra información sobre un paquete instalado: **-p** *(–print-avail)*
- Muestra información sobre un archivo de paquete desinstalado: **-I** *(–info)*
- Mostrar el estado de instalación de un paquete: **-s**
- Localiza el(los) paquete(s) que incluyen el(los) archivo(s) especificados por PATRON: **-S** *PATRON* (*–search PATRON*)
- Mostrar todos los paquetes instalados en el sistema: **–get-selections** (si añadimos un patrón, solo mostrará los que coincidan).
- Enumerar todos los paquetes instalados, cuyo nombre coincida con PATRON (útil utilizarlo también con *grep* omitiendo el patrón) : **-l** *PATRON* (*–list PATRON*)
- Listar los ficheros contenidos en un paquete: **-L**
- Desempaquetar un paquete pero no instalarlo: **–unpack**
- Busca paquetes parcialmente instalados: **-C** (*–audit*)

[Opciones]:

- Al eliminar un paquete con *-r* desactiva los paquetes que se basan en el que se está eliminando: **-B** (*–auto-deconfigure*)
- Al instalar (*-i*) o eliminar (*-r*) ignora la información de dependencias del paquete: **–ignore-depends=**<nom_paquete>
- Simula la instalación o desinstalación de un paquete para verificar los errores de dependencias: **–no-act**
- Instala de forma recursiva, todos aquellos paquetes que coinciden con el nombre (comodín) del paquete: **–recursive**
- No instala el paquete si ya existe una versión más actualizada: **-G**
- No instala el paquete si ya se encuentra instalada la misma versión: **-E** (*–skip-same-version*)

apt-get : herramienta para instalación de paquetes y dependencias desde una ubicación especificada en el archivo */etc/apt/sources.list* o en los archivos individuales */etc/apt/source.list.d/*

apt-get [acción] [opción] [nombre paquete]

[Acción]:

- Instalar un paquete : **install** *[-d|-m|-q(-qq)|–no-upgrade|-f|-s|-y|-b]*

- Actualizar todos los paquetes que dispongan de una versión más reciente: **upgrade** *[-m|-s|-q(-qq)|-y]*
- Actualizar de forma inteligente los paquetes, es decir, evita actualizar un paquete si con ello se rompe una dependencia: **dist-upgrade**
- Desinstalar un paquete: **remove** *[-m|-q(-qq)|-s|-y]*
- Elimina un paquete por completo, incluyendo sus archivos de configuración: **remove –purge**
- Obtener información actualizada sobre los paquetes disponibles en las fuentes (*source.list*): **update**
- Revisa la consistencia de la bbdd de paquetes y las instalaciones defectuosas de paquetes: **check**
- Elimina la cache de apt-get: **clean**
- Eliminar la cache, pero solo de los paquetes que ya no se pueden descargar: **autoclean**
- Obtiene el archivo de paquete fuente, partiendo del nombre de archivo: **source**

[Opciones]:

- Descargar los archivos de paquetes pero no los instala: **-d** (*–download-only*)
- Cuando se instala o elimina un paquete se puede intentar corregir un sistema cuyas dependencias no se satisfacen: **-f** (*–fix-broken*)
- Cuando se obtiene, instala, elimina, actualiza… un paquete, es posible ignorar aquellos que por cualquier causa no se pueden obtener: **-m** (*–ignore-missing* ó *–fix-missing*)
- Si queremos omitir parte de la información de progreso (instalando, actualizando, desinstalando..): **-q** (si queremos omitir más aún: **-qq**)
- Realizar una simulación de la acción (instalar, actualizar…): **-s** (*–simulate*, *–no-act*, *–just-print*, *–dry-run* o *–recon*)
- Automatizar una respuesta afirmativa para un comando: **-y** (*–yes* ó *–assume-yes*)
- Compilar un paquete fuente tras obtenerlo (source): **-b** (*–build* ó *–compile*)
- Cuando instalamos, podemos no actualizar un paquete si ya existe una versión anterior: **–no-upgrade**

apt-cache : Ofrece información sobre la base de datos de paquetes *Debian*.

[Opciones]:

- apt-cache **depends** samba : Informa de todas las dependencias del paquete samba
- apt-cache **rdepends** samba : Informa de aquellos paquetes que dependen de samba
- apt-cache **show** samba : muestra información sobre el paquete samba
- apt-cache **pkgnames** sa : Devolverá nombres de paquetes que comiencen con sa
- apt-cache **stats** : Informa de cuantos paquetes hay instalados, cuantas dependencias hay grabadas, y otrasestadísticas
- apt-cache **unmet** : Devuelve información sobre las dependencias no satisfecha

dselect : Herramienta potente y con muchas opciones (nada obvias). Para ejecutarlo, basta con escribir *dselect* en la línea de comandos y seleccionar las diferentes opciones. Aunque mediante la sintaxis *$dselect [opciones] [comandos]* también es posible pasarle comandos y opciones como argumento. Algunas de estas opciones y comandos son:

[Opciones]:

- Cambiar el directorio donde se encuentran los ficheros de *dpkg* (por defecto */var/lib/dpkg*): **–admindir** *<ruta>*
- Activa el modo experto: **–expert**
- Activar el modo debug y enviar la info a un archivo: **–debug** *<archivo>*

[Comandos]:

- Instalar nuevos paquetes o actualizarlos a nuevas versiones: **install**
- Configurar cualquier paquete previamente instalado: **configure**
- Eliminar o purgar un paquete y sus archivos de configuración: **remove**
- Actualizar la lista de paquetes disponibles: **update**
- Ver, gestionar y seleccionar paquetes y sus dependencias: **select**
- Alterar la ruta en la que encontrar paquetes para su instalación: **access**
- Salir de dselect: **quit**

aptitude : A diferencia de *dselect*, *aptitude* muestra menús por los que poder movernos con *Control-T*

$aptitude [comando] [argumento]

[Comandos]:

- Instalar un paquete concreto: **install**
- Desinstalar un paquete: **remove** ó **$aptitude install nombrepaquete-**
- Actualizar software de forma segura: **safe-upgrade**
- Actualizar software de forma menos segura (puede generar errores): **full-upgrade**
- Buscar paquetes: **search**
- Actualizar la lista de paquetes de los repositorios APT: **update**
- Eliminar los paquetes descargados que ya no se encuentran disponibles: **autoclean**
- Eliminar todos los paquetes descargados: **clean**
- Obtener ayuda interna: **help**

Similitud de algunas opciones RPM

- Instalar un paquete: **rpm -ivh** == **rpm -Uvh** == **yum install** (este además instala las dependencias)
- Eliminar un paquete: **rpm -e** == **yum remove**
- Actualizar un paquete suponiendo que ya está instalado: **rpm -Uvh** == **rpm -Fvh**
- Información sobre un paquete: **rpm -qi** == **yum info**

Nota: Podemos ampliar la info. de rpm con **-Rl** (*R* paquetes dependiente y *l* lista de archivos del paquete)

- Buscar un paquete: **yum search** == **rpm -qa |grep** *<paquete>* (este además nos indica si está o no instalado)

Similitud de algunas opciones Debian

- Instalar un paquete: **dpkg -i** == **apt-get install** == **aptitude install** == **dselect install**
- Eliminar un paquete: **dpkg -P** == **apt-get remove** == **aptitude remove** == **aptitude install nom_paquete-** == **dselect remove**
- Actualizar de forma segura: **apt-get dist-upgrade** == **aptitude –safe-upgrade**
- Información sobre un paquete: **dpkg -p** == **apt-cache show** == **aptitude show**
- Buscar un paquete instalado: **dpkg -l** == **dpkg –get-selections** == **apt-cache pkgnames <patrón>** == **aptitude search**

Comandos para la administración de librerías

- **ldd** : Comprueba las bibliotecas de las que depende un programa o las bibliotecas de las que depende otra biblioteca. *ldd* muestra una lista de bibliotecas y su ruta completa (separadas por **=>**), o directamente la ruta completa de esta.

$ldd nombre_programa

[Opciones]:

– Mostrar información extensa (adicional): -v

- **ldconfig**: Actualiza la caché de configuración de librerías y como trabajo secundario, actualiza los enlaces simbólicos de referencia a librerías.

$ldconfig

[Opciones]:

– Mostrar información detallada: **-v**

– No reconstruir la caché (solo actualizar los enlaces simbólicos): **-N**

– Actualizar caché, pero no los enlaces: **-X**

– Actualizar solo los enlaces incluidos en los directorios especificados: **-n** */ruta/a/directorio*

– Utilizar un archivo de configuración diferente a */etc/ld.so.conf*: **-f** *nom_nuevo_archivo*

– Utilizar un nuevo archivo de cache: **-C** *nuevo_cache*

– Utilizar un directorio raíz diferente: **-r** *dir*

– Mostrar la caché actual: **-p**

- **uname** : Nos da información sobre el kernel de nuestro sistema**uname [opciones]**

  [Opciones]:

  - Mostrar el nombre del nodo (nombre del equipo en la red): **-n**, *–nodename*
  - Mostrar el nombre del kernel: **-s**, *–kernel-name*
  - Mostrar la versión del kernel: **-v**, *–kernel-version*
  - Mostrar la compilación del kernel: **-r**, *–kernel-release*
  - Ver la arquitectura utilizada: **-m**, *–machine*
  - Mostrar información sobre el fabricante, modelo, velocidad del reloj…: **-p**, *–processor*
  - Nombre del sistema operativo (GNU/LINUX, nomalmente): **-o**, *–operating–system*
  - Imprimir toda la información anterior: **-a**, *–all*

Compilación del kernel y paquetes

gcc –version : Comprobamos si tenemos el compilador *gcc* de *GNU/Linux* instalado

./configure : Verificará los requisitos para la compilación del paquete (alertará si faltase alguno) y preparará la compilación
Nota: Podemos usar la opción *–prefix=/directorio/de/instalación* para indicar un directorio alternativo en el que instalar la nueva aplicación

make : Determina automáticamente que '*piezas*' de un programa necesitan ser recompiladas y lanza las órdenes para su compilación. Se puede utilizar realmente para describir cualquier tarea donde algunos ficheros deban ser actualizados automáticamente a partir de otros en cualquier momento en que estos cambien. Antes de utilizar el comando *make* es necesario escribir un archivo llamado **makefile** que describe las relaciones entre los ficheros de nuestro programa y las órdenes necesarias paras actualizar cada fichero. Una vez escrito el *makefile* cada vez que cambiemos algún fichero, el comando *make* será suficiente para que se realicen todas las recompilaciones necesarias. Algunas opciones a utilizar con el comando *make* son:

- Cambiar el directorio de trabajo antes de leer los *makefiles*: **-C** *<directorio>*
- Mostrar información de depuración además del procesamiento normal: **-d**
- Dar preferencias a las variables de entorno sobre las que vienen definidas en los *makefiles*: **-e**
- Utilizar un archivo como *makefile*: **-f** *<archivo>*
- Especificar un directorio en el que buscar archivos *makefiles*: **-I** *<directorio>*
- Especificar un número determinado de hilos (si tenemos un procesador con más de un núcleo): **-j** *<num>*
- Comprobar las órdenes que se ejecutarían pero no hace nada, solo comprueba: **-n**
- No reconstruir el archivo aun siendo más antiguos que sus dependencias: **-o** *<fichero>*
- Modo silencioso. No muestra las órdenes que se están ejecutando: **-s**

make install : Utilidad que instalará el software una vez compilado

make clean : Útil para realizar una limpieza en el directorio del código fuente, bien por que hayamos obtenidos errores tras la primera compilación y queramos volver a compilar o porque hemos acabado la compilación e instalación del nuevo software. Este

comando se ejecuta desde el propio directorio (el que contiene el código fuente y los paquetes de compilación).

make unistall : Si queremos realizar la desinstalación del nuevo software compilado e instalado. Deberemos de ser *root* y encontrarnos en el directorio donde se encuentra el código fuente.

Comandos para la conversión de paquetes RPM, Debian, stampede y tarball

- **rpm2cpio**: Convierte un paquete *RPM* en un archivo *cpio*

rpm2cpio samba-client.3.6.5-86.fc17.1.x86_64.src.rpm **>** samba-client.3.6.5-86.fc17.1.x86_64.src.cpio

- **cpio** : Extrae archivos individuales de un paquete en formato *cpio*

$cpio -i –make-directories < samba-client.3.6.5-86.fc17.1.x86_64.src.cpio

Podemos unir estos 2 comandos con una tubería:

$rpm2cpio samba-client.3.6.5-86.fc17.1.x86_64.src.rpm **| cpio -i –make-directories**

Cubriremos mejor **cpio** en el *Capítulo 4*

Nota: Para un paquete fuente, el proceso de extracción seguramente sea un paquete *tarball* de código fuente (**.spec**), que contiene información que *RPM* utiliza para crear el paquete y quizás algunos archivos de parches

- **Alien** : Convierte paquetes *deb* en *rpm*, *tar*, *stampede* o viceversa.

alien [opciones] [archivo(s)]

[Opciones]:

– Convertir un paquete a formato deb: **–to-deb**

– Convertir un paquete a formato rpm: **–to-rpm**

– Convertir un paquete a formato stampede: **–to-slp**

– Convertir un paquete a formato tarball: **–to-tgz**

– Instalar el paquete convertido y a su vez eliminarlo tras la instalación: **–install**

Ejemplo: Convertir un *.deb* en *.rpm*

$alien –to-rpm paquete-1.2.3_i386.deb

Ejemplo2: Instalar un paquete *tarball* (en un sistema *Debian*), pero manteniendo un registro de los archivos que contiene en su base de datos de paquetes *Debian*:

$alien –install binary_tarball.tar.gz

Nota: Si se omite el formato de destino, *alien* interpreta que queremos convertirlo a *.deb*

Comandos para la administración de procesos del sistema

ps : Muestra el estado de los procesos. Por defecto *ps* solo muestra los procesos que se ejecutaron desde su propia terminal (xterm, acceso en modo texto o acceso remoto).

$ps [-|–] OPCIONES

[Tipos de opciones]:

- Opciones Unix98 : Opción de un único caracter, precedido por "-". Se pueden agrupar ($ps -e-f = ps-ef)
- Opciones BSD : Opción de un único caracter, pero no van precedido de "-". También se pueden agrupar
- Opciones GNU largas: Opciones multicaracter y precedidas de "–"

[Opciones Unix98]:

- Mostrar opciones básicas (PID, TTY, TIME, CMD): -A , -e
- Mostrar los procesos que pertenecen a un usuario concreto bien por su ID o por el nombre de login: -u usuario
- Mostrar información adicional: -f, -l . Podemos usar -L con -f para ver el número e ID de hilo
- Mostrar la jerarquía de procesos, útil para averiguar los parentescos de un proceso: -H -f ó –forest (agrupan los procesos y emplean sangrías)
- Mostrar una salida ancha, útil si queremos redireccionarla a un archivo para luego examinarla (por defecto ps recorta la salida a 80 columnas) : -w > archivo.txt

[Opciones BSD]:

- Mostrar todos los procesos del usuario que proporciona el comando : x
- Ver los procesos de un usuario concreto: U usuario
- Mostrar información adicional: j (formato de control de trabajos), l (formato largo de BSD), u (formato orientado al usuario) y v (formato de memoria virtual)
- No recortar la salida: w

[Opciones GNU]:

- Mostrar la ayuda de ps: –help
- Mostrar procesos de un usuario: –User usuario
- Mostrar la jerarquía de procesos: –forest

Algunos ejemplos:

- Ver todos los procesos del sistema:

```
ps -ef, ps -eF, ps -ely, ps ax, ps axu
```

- Imprimir el árbol de procesos:

```
ps -ejH, ps axjf
```

- Mostrar información sobre los hilos:

```
ps -eLf, ps axms
```

- Mostrar información de seguridad:

```
ps -eM, ps axZ
```

- Mostrar los números ID's de un programa concreto:

```
ps -C nombre_prog -o pid=
```

- Mostrar el nombre de programa de un proceso:

```
ps -p num_pid -o comm=
```

pgrep : Herramienta que combina los comandos *ps* y *grep*, para imprimir los ID de procesos que coinciden con el patrón pasado en la expresión regular.

$pgrep [opciones] expresión regular [archivo]

Ejemplo:

```
$ps ax |grep http | grep -v | awk '{print $1}' == $pgrep http
```

[Opciones]:

- Imprimir los procesos ejecutados por un determinado usuario: **-u**
- Mostrar el nombre del proceso: **-l**
- Mostrar todos los procesos que no coincidan con el patrón: **-v**
- Imprimir la ruta del binario del proceso: **-f**
- Mostrar el total de los procesos que coinciden con el patrón mediante un dígito: **-c**
- Mostrar los procesos de un determinado grupo mediante el nombre del grupo: **-G** (-g para el GID)
- Identificar el proceso más viejo: **-o**
- Identificar el proceso más nuevo: **-n**

Referencias:

http://es.wikipedia.org/wiki/Pgrep

http://systemadmin.es/2008/12/pkill-y-pgrep-buscar-o-mandar-senales-por-el-nombre-del-proceso

http://linux.floresdecerezo.com/el-comando-pgrep/

fuser: Nos permite identificar que procesos están haciendo uso determinados archivos, *sockets* o directorios, o dicho de otra manera por que procesos están siendo controlados determinados archivos. También nos permite matar procesos.

fuser [opciones] archivo

Opciones:

- Mostrar los procesos para todos los archivos pasados desde la línea de comandos: -a, –all
- Mata un proceso: -k
- Activa el modo interactivo: -i
- Muestra más información: -v
- Mostrar todas las señales conocidas -l
- Matar solo procesos que tienen acceso de escritura: -w
- Activar el modo silencioso: -s
- Añade el nombre del usuario propietario para cada PID: -u, –user

top : Muestra un resumen de la información del sistema y de los procesos del ordenador que más CPU consumen.

$top -opciones

[Opciones]:

- Especificar un retardo diferente al de por defecto (5s) para la actualización de la info: **-d**
- Monitorizar procesos específicos (hasta 20): **-p** *PID*
- Mostrar un número concreto de actualizaciones y luego cerrarse: **-n** *num_iteraciones*
- Utilizar un archivo para recoger el uso de CPU de determinados programás: **-b**

[Teclas para interaccionar con top]:

- Mostrar información de ayuda: **h**, **y**, **?**
- Destruir un proceso conociendo su PID: **k**
- Cambiar la prioridad de un proceso: **r**
- Modificar el intervalo de refresco de información: **s**
- Ordenar los datos mostrados por uso de CPU (top por defecto actúa así): **P**
- Ordenar los datos por uso de memoria: **M**
- Salir de top: **q**

uptime : Nos permite conocer el tiempo que lleva el sistema sin ser reiniciado, así como la carga media (load average).

$uptime

Nota: El comando *uptime* desplegará una línea con 4 campos: la hora actual, el tiempo que lleva el sistema iniciado sin ser reiniciado, el número de usuarios conectados y el promedio de carga del sistema en los últimos 1, 5 y 15 minutos.

También podemos ver el tiempo sin reinicio del sistema en segundos desplegando *$cat /proc/uptime* o el promedio de carga a través del archivo */proc/loadavg*

free : Muestra la cantidad de memoria física y de intercambio libre y usada en el sistema, así como el búfer utilizado por el kernel.

$free [opciones]

[Opciones]:

- Mostrar la memoria en bytes, kilobytes, megabytes o gigabytes: **-b -k -m -g** (*respectivamente*)
- Mostrar la cantidad de memoria en unidades (B, M, K, G o T) directamente abreviadas según la cantidad: **-h**
- Mostrar la salida del comando durante un tiempo deseado: **-s** *<num_seg>*
- Mostrar la salida de *free* tantas veces como deseemos. Es necesario acompañarlo de la opción s: **-c** *<num_veces>*

jobs : Identifica el número de trabajos en ejecución. Útil para conocer los ID de tareas y pasar estas de primer a segundo plano, suspenderlas o terminarlas

$jobs

& : Si lo añadimos al final de la línea de ejecución del comando nos permitirá que este se ejecute en segundo plano, dejando libre la terminal.

wait : Nos permite elegir cuando queremos retomar el control de la terminal en el caso de que haya varios programás en background.

$wait

[Opciones]:

- Si queremos esperar a que terminen todos los trabajos, no pasaremos ninguna opción.
- Si queremos retomar el control cuando termine el trabajo [2]: **%2**
- Si queremos retomar el control cuando termine el proceso 4563: <num_proc>

nohup : Permite la continuidad en la ejecución de un programa/comando aun habiendo cerrado la terminal desde la que se ejecutó.

$nohup [comando/programa] [opciones comando/programa]

screen : Crea terminales virtuales para poder ejecutar un comando pudiendo cerrar la ventana y que este siga ejecutándose, además poder abrir de nuevo esa misma terminal virtual y retomar el control.

$screen [opciones]

[Opciones]:

- Si escribimos 'screen' a secas, abriremos una nueva terminal virtual
- Si queremos listar las terminales virtuales existentes en el equipo: **-ls**
- Para retomar el control de una determinada terminal virtual: **-r** *ID*

Nota: Si solo hay una terminal virtual abierta, basta con escribir screen *-r*

nice : El comando *nice* nos permite ejecutar programás con una determinada prioridad en la CPU.

$nice [prioridad] [programa] [argumentos del programa]

```
$nice -12 [programa] [argumentos...]
$nice -n 12 [programa] [argumentos...]
$nice --adjustment=12 [programa] [argumentos...]
```

Nota: Estos 3 comando ejecutan "programa" con prioridad 12 positiva. Para prioridades negativas solo podemos usar las 2 últimás. Si no le pasamos prioridad a *nice*, inicia el programa con prioridad 10

[Prioridad]:

- La prioridad va desde -20 a 19, siendo los números negativos los de prioridad más elevada y solo pueden se utilizados por root.

renice : Permite modificar la prioridad de un programa en ejecución. La sintaxis es igual a la de *nice*, pero además nos permite modificar la prioridad de programa(s) por PIDs, por GUIDs y por nombres de usuarios

$renice [prioridad] [-p PIDs | -g grps | -u usuarios] [programa] [argumentos…]

Nota: Solo *root* puede modificar la prioridad por GUIs y nombre de usuarios. Si no se le pasa prioridad a *renice*, entiende que el primer número es un *PID*.

kill : Envía una señal al kernel para finalizar un proceso

$kill [parametro] [señal]

[Parámetro]:

- Para indicar la señal por su nombre completo: **-s**
- Para indica la señal por su nombre (sin SIG): **–***señal*
- Para pasarle el número de señal: **–***num*

Nota: El comando *kill* también acepta el parámetro *%id_job* para pasar una señal a determinado trabajo. **$kill %3**

pkill : Es igual que *pgrep*, pero además mata el proceso, es decir, se le pasa un nombre de programa como patrón, un usuario o grupo de usuarios y mata estos procesos. Además permite pasar señales a procesos.

$pkill [opciones] expresión regular

Ejemplo:

```
$pkill -HUP syslog
```

Nota: Este comando le pasa la señal de finalización a *syslog*

[Opciones]:

- Aquellas que se le pasen para identificar a un proceso, serán las que sirvan para identificar y matarlo.
- Enviar una determinada señal al proceso: **–**SEÑAL (sin SIG)

killall : Nos permite matar un proceso por su nombre y no por su *PID*. También podemos pasarles señales como a *kill*. Existe una variante de Unix que mata todos los procesos iniciados por un determinado usuario, en caso de pasarle como argumento el nombre del usuario.

```
$killall -i vi
```

[Opciones]:

- Para pedir confirmación de finalización de proceso: **-i**
- Todos los procesos de un determinado usuario: **-u**
- Todos los procesos de un determinado grupo: **-g**
- Incluir expresión regular para la búsqueda de los procesos: **-r**
- Mandar una señal de finalización: **-s**

COMANDOS LPIC-1

Capítulo 3

Comandos para la administración del hardware

- **biosdecode** : Información básica sobre el contenido de la memoria del BIOS
- **dmidecode** : Nos muestra información detallada sobre el hardware del sistema. Podemos usarlo sin opciones aunque para especificar un "área" concreto de la info podemos usar la opción **-t** con el tipo de DMI que queremos mostrar.

#dmidecode -t
dmidecode: option requires an argument — 't'
Type number or keyword expected
Valid type keywords are:
bios
system
baseboard
chassis
processor
memory
cache
connector
slot

#dmidecode -t memory

- **lshw** : Muestra información detallada del hardware conectado al equipo. Muestra más info que *dmidecode*, su salida es muy extensa por lo que en ocasiones conviene pasarla a formato html o xml, para que su lectura sea más cómoda.

$sudo lshw -html > /tmp/myhw.html

[*Opciones*]:

– Ver información sobre el disco y CDROM: **#lshw -class disk**
– Ver información sobre la memoria RAM: **#lshw -class memory**
– Generar un reporte compacto: **#lshw -short**

- **lspci** : Muestra información sobre todos los buses y dispositivos conectados a estos.

[*Opciones*]:

– Incrementar el nivel de detalle en la salida: **-v, -vv, -vvv**
– Mostrar el nombre del fabricante, el dispositivo y sus codigos numericos asociados: **-nn**
– Mostrar los IRQ además de otros datos según los ven los dispositivos en lugar de como los ve el kernel: **-b**
– Mostrar una vista en arbol de la relación que mantienen los dispositivos: **-t**
– Mostar los datos de un dispositivo concreto: **-d <dispositivo>**
– Mostrar dispositivos que se ocultan tras un puente PCI mal configurado: **-M**

- **lsusb** : Listar los dispositivos USB conectados al sistema.

[*Opciones]:*

– Mostrar información detallada: **-v**
– Restringir la salida en función del número de bus y dispositivo: **-s [[bus]:] [num_dispo]**
– Mostrar la salida en función del fabricante y el producto: **-d [ID_fabricante]:[ID_producto]**
– Mostrar dispositivos por nombres de archivos: **-D nombre_archivo_dispositivo**
– Mostrar la lista de dispositivos en forma de árbol: **-t**

Nota: El nombre del archivo del dispositivo suele colgar del directorio **/proc/bus/usb**, directorio el cual proporciona una interfaz de bajo nivel.

- **setpci** : Comando que nos permite consultar y configurar dispositivos PCI

[*Opciones*]:

– Mostrar información: **-v**
– Podemos complementar la opción -v con la siguiente, para realizar una demostración (es decir no escribimos cambios) y ver así que ocurriría: **-D**

[*Dispositivos*]:

– Para seleccionar un dispositivo podemos hacer así: **-S[[<bus>]:][.[<func>]][<slot>]**
– o así: **-D [<fabricante>]: [<dispositivo>]**

Ejemplos:

setpci -v -d *:* latency_timer=b0

(-d *:* indica a setpci que aplique este ajuste a todos los dispositivos PCI. La opción latency_timer=b0 ajusta el temporizador a 176 (b0 es en hexadecimal 176)

setpci -v -s 00:0f.0 latency_timer=ff

(00:0f.0 hace referencia a un dispositivo concreto. La opción -s indica que se va a especificar un dispositivo concreto según el bus/ranura y la función. El valor ff especifica un temporizador de latencia de 256, que setpci redondea a 248)

NOTA : Otra forma de obtener información de hardware es a través del sistema de archivo virtual /proc. Por ejemplo:

– less /proc/cpuinfo
– less /proc/meminfo

Comandos para la administración de los módulos del kernel

- **lsmod**: lista los módulos que se encuentran cargados en el sistema
- **modinfo**: Muestra información sobre un módulo
  $sudo modinfo [opciones] módulo

[*Opciones*]:
– Mostrar el directorio del que cuelga el módulo: **-b**
– Mostrar el autor, descripción, nombre de fichero: **-a, -d, -n** (respectivamente)

- **insmod**: Carga un único módulo. Hay que pasarle la ruta completa hacia el módulo
  $sudo insmod módulo [opciones del módulo]
- **modprobe**: Carga un módulo y todos aquellos de los que depende
  $sudo modprobe [opciones] módulo [opciones del módulo]

[*Opciones*]:
– Cambiar el archivo de configuración: **-C** <nueva_ruta> modulo
– Realizar una simulación de la inserción del módulo, se puede acompañar de **-v** para depurar: **-n** (–dry-run)
– Descargar un módulo y todos aquellos de los que dependen (si no están siendo usados): **-r**
– Forzar la carga de un módulo: **-f** (–force)
– Muestra todos los módulos de los que depende (no instala, solo informa): **–show-depends**
– Mostrar todos los módulos disponibles (admite comodines como patrón): **-l** (–list)

- **nmmod**: Es usado para descargar un solo módulo. Al igual que *insmod*, hay que pasarle la ruta completa
- **rmmod**: Descarga un módulo por su nombre. Si se intenta descargar un módulo que depende de otos que se encuentran en uso, dará error

$sudo rmmod [opciones] módulo

[Opciones]:
– Información detallada: **-v**
– Forzar la descarga de un módulo: **-f** (–force)
– Esperar a que deje de usarse para ser descargado: **-w** (–wait)

Comandos para el particionado de discos

MBR

- **fdisk**: Herramienta modo texto para el particionado MBR

#fdisk [opciones] dispositivo

[Opciones]
– listar las particiones de un dispositivo: **-l**
– Especificar el número de sectores: **-S**
– Especificar el número de cilindros: **-C**
– Especificar el número de cabezales: **-H**
– Modo de compatibilidad con DOS: **-c=mode** (mode=dos o mode=nondos, por defecto usa nondos)
– Imprimir ayuda: **-h**

Si usamos **fdisk** sin opciones indicando directamente el dispositivo (*$ sudo fdisk /dev/sdb*) entraremos al modo interactivo. Algunas opciones del modo interactivo son:

– **n**: Crear una nueva partición

Nota: fdisk mide los puntos de inicio y fin de una partición en cilindros o sectores, aunque podemos definir el tamaño con un sigo + un valor numérico y un sufijo (MB, GB..).

d : Elimina una partición
l : Enumera los códigos de tipo de particiones
t : Cambiar el tipo de una partición
L : Si hemos usado **t** para cambiar el tipo de partición, podemos usar **L** para que se despliegue una lista de tipos de particiones y elegir una de entre ellas
p : Muestra las particiones.
a : Marca una partición como activa para que se arranque desde ella
m o **?** : Desplegar el menú o ayuda
q : Salir sin guardar cambios
w : Salir escribiendo los cambios

- **sfdisk**: Herramienta de particionado no interactiva (a diferencia de *fdisk*).

#sfdisk [opciones] dispositivo

Si no utilizamos opción alguna con *sfdisk* querrá decir que queremos particionar el disco. Podemos indicar con **-O** <nombre_archivo> , que queremos realizar una copia de los sectores que vamos a modificar y en caso de accidente restaurarlo con **-I**

Ejemplo:

#sfdisk /dev/sdc -O sdc-partition-sectors.save

#sfdisk /dev/sdc -I sdc-partition-sectors.save

[*Opciones*]:
– Listar las particiones: **-l**
– Podemos chequear la partición con: **-v**
– Imprimir los tipos de sistema de archivos: **-T**
– Cambiar el ID de una partición: **-c** (–print-id y –change-id)
– Mostrar el tamaño del disco: **-s**
– Crear un volcado de las particiones de un disco: **-d** (#sfdisk -d /dev/sda > sda.out)
– Marcar una partición como activa o inactiva: **-A** <partición>

Nota: Podemos usar las opciones -l y -v juntas o bien -v y -q lo cual nos devolverá unicamente el estado de la partición.

- **cfdisk**: Administrar una tabla de particiones mediante un menú.

#cfdisk [opciones] dispositivo

Si no marcamos opciones entramos a un menú donde podremos crear, modificar, eliminar, mover, etc… particiones.

GPT:

- **parted**: Herramienta de particionado GPT.
  #parted [opciones] dispositivo [comandos [opciones_comandos]]

[Opciones]:
– Listar las particiones y su tamaño de un dispositivo: **-l** (–list)
– Pasar un script para que interactúe con parted y no el usuario: **-s**

*** partición = 1, 2, 3…*
*** tipo_de_tabla = bsd, dvh, loop, mac, msdos, pc98 o sun*
*** start end = suelen ser el número de MB inicial y final que abarcará la partición dentro del espacio de disco.*
– Chequear de forma simple una partición: **check**
– Copiar una partición de un dispositivo (por defecto el dispositivo actual) a otro dispositivo: **cp** <origen> <destino>
– Imprimir la ayuda para un determinado comando: **help**
– Crear un sistema de archivo determinado (FAT16/32, ext2, reiserfs o linux-swap): **mkfs** <partición> <fs-type>
– Crear una partición vacía con un sistema de archivos determinado (FAT16/32, ext2, reiserfs, ufs, Btrfs, linux-swap, etc…: **mkpart** <tipo-partición> <fs-type> start end

– Crear una partición dándole formato (se recomienda usar **mkpart** para crear la partición y **mke2fs** para darle formato): **mkpartfs** <tipo-partición> <fs-type> start end
– Crear una nueva tabla de particiones: **mklabel** <tipo_de_tabla>
– Mover una partición indicando el inicio (que queremos que tenga) y el final: **move** <partición> start end
– Darle nombre a una partición: **name** <partición> nombre
– Desplegar la tabla de particiones: **print**
– Rescatar una partición indicando el inicio y el final de la misma: **rescue** start end
– Redimensionar un sistema de archivos de una partición (por defecto en MB): **resize** <partición> start end
– Eliminar una partición: **rm** <partición>
– Seleccionar un dispositivo o partición RAID/LVM: **select** <dispositivo>

- **gdisk**: Es una herramienta de particionado a la que pasamos como argumento el dispositivo a particionar y podemos ejecutar una serie de acciones sobre ella. Estas acciones son listadas con los comandos **help** o **?** una vez dentro de *gdisk*. Algunas de estas opciones son similares a las de *fdisk*:

[Opciones]:
– Crear un backup de la tabla de particiones: **b**
– Modificar el nombre de la partición: **c**
– Eliminar una partición: **d**
– Mostrar información sobre una partición: **i**
– Listar los tipos conocidos de particiones: **l**
– Crear una nueva partición: **n**
– Crear una nueva tabla de particiones *GPT* vacía: **o**
– Imprimir la tabla de particiones: **p**
– Salir sin guardar: **q**
– Salir guardando: **w**
– Opciones de recuperación: **r**
– Comprobar disco: **v**
– Funciones extras: **x**

Nota: Cuando use el comando n , se debe pulsar ‘Enter’ para dar a la partición el último número libre y es necesario pulsar ‘Enter’ nuevamente para que acepte el sector de arranque por defecto para la nueva partición antes de establecer el tamaño que necesite para el último sector.

LVM:

Volúmenes físicos:

- **pvcreate**: Para poder usar un dispositivo de almacenamiento con *LVM* es necesario antes crear un volumen físico.

*#**pvcreate** /dev/sda3*

- **pvresize**: Aumentar el tamaño de un dispositivo físico (se puede hacer en caliente)
- **pvremove**: eliminar un volumen físico
- **pvmove /dev/sdbX /dev/sdfX**: Mueve un volumen físico
- **pvdisplay**: Ver los volúmenes físicos de un sistema

Grupo de Volúmenes

- **vgcreate**: Crea un grupo de volúmenes donde almacenar uno o varios volúmenes físicos.

*#**vgcreate** <nombre-grupo> <nom-pv>*

- **vgextend**: Nos permite extender el grupo de volúmenes ocupando más volúmenes físicos previamente creados.

*#**vgextend** <nom-grup-a-extender> <dispositivo-a-añadir>*

- **vgreduce**: Disminuye el tamaño de un grupo de volúmenes
- **vgdisplay**: Ver los grupos de volúmenes de un sistema.
- **vgchange -a <grupo_de_volúmenes>** : Desactiva un grupo de volúmenes

Volumen lógico:

- **lvcreate** : Crear un volumen lógico (como si fuese una partición) para luego asignarle un tipo de sistema de archivo. Lo bueno es que podremos redimensionar el tamaño de esta "partición" tanto como el grupo de volumen nos permita.

*#**lvcreate** [opción1 argumento] [opción2 argumento]….[opciónN argumento] <grupo_de_volúmenes> [volumen_físico]*

[Opciones]:
– Indicar el tamaño: **-L** <tamaño>
– Indicar el nombre: **-n** <nombre>
– Dar permisos de lectura (r) o ambos (por defecto rw): **-p** {r|rw}
– Crear un snapshot: **-s**

Ejemplo:

*#**lvcreate** -L 50GB -n "lvhome" -p r mi_vg_1*

(donde *mi_vg_1* es mi grupo de volúmenes anteriormente creado)

#*lvcreate* –size 100m –snapshot –name snapvol /dev/vg00/lvhome

Nota: Crea un *snapshot* de *lvhome* llamado *snapvol* para por ejemplo crear una copia de seguridad, mientras el volumen lógico *lvhome* sigue en ejecución.

- **lvconvert**: Convertir un volumen lógico
- **lvresize**: Redimensiona un volumen lógico
- **lvdisplay**: Muestra los volúmenes lógicos creados en el sistema

Nota: Los volúmenes lógicos suelen almacenarse bajo */dev/mapper* y */dev/nombre_grupo_volúmenes*, por lo que nuestro volumen tendrá el nombre o ruta completa */dev/mapper/vg1-lvhome* o */dev/vg1/lvhome*. En caso de no encontrarse los volúmenes necesitaremos cargar el modulo y escanear los volúmenes.

- Cargar módulo de *LVM*:

#***modprobe*** *dm-mod*

- Escanear *LVM*:

#vgscan
#vgchange -ay

- Si queremos dar formato a un volumen lógico usamos *mkfs*

#***mkfs.ext4*** */dev/mapper/vg1-lvhome*

Redimensionar un volumen lógico y su sistema de archivos:

Aumentar la capacidad de un *LV*:

- **lvresize**: Redimensiona un volumen lógico***#lvresize*** *-L +5GB /dev/vg1/lvhome* (aumentamos el tamaño en 5GB a *lvhome*, no es necesario tener desmontado el volumen)
- **resize2fs**: Redimensiona sistemás de archivo ext2,ext3 y ext4
- **resize_reiserfs**: Para sistema de archivo reiserfs
- **xfs_growfs**: Para *XFS* (no permite disminuir su tamaño solo aumentarlo)

#resize2fs */dev/vg1/lvhome*

Nota: Debemos de tener el volumen desmontado, podemos hacerlo pasando a nivel 1 (*telinit 1*) y desmontarlo.

Disminuir el tamaño de un *LV*:

Nota: En este caso, desmontaremos primero el volumen, reduciremos el sistema de archivos y luego el volumen. Supongamos que nuestro *lvhome* tiene una capacidad de 9GB y queremos dejarlo en 4GB.

1. ***#resize2fs*** */dev/vg1/lvhome 4G*
2. ***#lvreduce*** *-L -5G /dev/vg1/lvhome o* ***#lvreduce*** *-L 4G /dev/vg1/lvhome*

Nota: Ahora ya podemos montar nuestro volumen *lvhome*

Comandos para la creación de sistemás de archivos

- **mkfs** : Dar formato a una partición.

#mkfs [opciones] partición

[Opciones]:
– Indicar el tipo de sistema de archivo para dar formato: **-t** <tipo-fs>
– Comprobar bloques defectuosos durante el formato: **-c**
– Reducir el espacio libre que deja *mkfs* para tareas de recuperación (5% por defecto): **-m** <nuevo-%>

Nota: Podemos llamar a *mkfs* de distintas maneras, según que tipo de sistema de archivos utilicemos por ejemplo: *mkfs -t ext4* es igual a *mkfs.ext4*

Formatear la partición **/dev/sdb1** con el sistema de archivos **ext4**, con un espacio libre (desperdiciado, podemos incluso darle un 0, útil si no tenemos un sistema operativo en esta partición) reducido al **1%** y con una etiqueta "**Mi USB**":

```
$ sudo mkfs.ext4 -m1 -L "Mi USB" /dev/sdb1
```

- **mke2fs** : Es como *mkfs* pero exclusiva para la familia de los sistemás de archivos extendidos. Nos permite un mayor número de opciones ya que se centran unicamente en estos sistemás de archivos. Por ejemplo el equivalente a *mkfs.ext3* sería *mke2fs -j*, crea un sistema de archivo extendido con respaldo de transacciones. Este comando es útil si queremos conocer los bloques en los que se guardan los backups de superbloques, deberemos de pasar la opción **-n** (simulación) y **-b** <blocksize> para que sea más preciso.

#mke2fs [opciones] partición

[Opciones]:
– Crear la partición con una etiqueta: **-L** <etiqueta>
– Crear un sistema de archivos con un UUID determinado: **-U** <UUID>
– Crear el sistema de archivo ext2, ext3 o ext4: **-t** <tipo-fs>
– Utilizar mke2fs en un script: **-q**
– Simular un formateo: **-n**
– Pasarle el tamaño del bloque: **-b** <blocksize>

- **mkswap**: Crear una partición de intercambio
- **swapon**: Activar una partición de intercambio

Comandos para el montaje y desmontaje de unidades

- **mount**: Nos permite montar un sistema de archivo en un punto de montaje.

#mount [opciones] dispositivo <punto_montaje>

[Opciones]:
– Montar todos los sistemás de archivos indicados en */etc/fstab*: **-a**
– Montar en modo solo lectura: **-r**
– Montar en modo lectura/escritura: **-w**
– Especificar el tipo de filesystem a montar: **-t** <filesystem>
– Montar por UUID: **-U** <uuid>
– Montar por etiqueta: **-L** <label>
– Añadir opciones extras: **-o** <opciones>

Filesystem: ext2, ext3, ext4, reiserfs, jfs, xfs, vfat, msdos, iso9660, udf, nfs, cifs…

Nota: Si omitimos la opción **-t** , Linux intentará detectar de que sistema de archivos se trata. Aun pasandole la opción **-t** y el tipo de filesystem, si el kernel de Linux no tiene soporte para este tipo de sistema de archivos y tampoco precisamos de herramientas de 3º para ello, Linux no montará el sistema de archivos.

[Opciones extras válidas también para /etc/fstab]:
– Utiliza las opciones por defecto *(rw, suid, dev, exec, auto, nouser, async)*: **defaults**
– Interpretar caracteres o bloques especiales en el filesystem: **dev**
– Permitir ejecutar archivos binarios: **exec**
– Montar un archivo como si de una partición se tratase (por ejemplo una imagen.img): **loop**
– Montar o no el dispositivo de forma automática, bien durante el arranque o durante la ejecución de mount -a: **auto/noauto**
– Permitir o no, montar el dispositivo a usuarios indicando #mount /y_el_punto_de_montaje: **user/nouser**
– Permite a cualquier usuario desmontar el filesystem montado por otro usuario (user no permite esto): **users**
– Permitir o no, montar el dispositivo igual que user pero a diferencia de que debe de ser el propietario: **owner**
– Activar nuevas opciones sin tener que desmontar y volver a montar el dispositivo: **remount**
– Montar en solo lectura: **ro**
– Montar en lectura/escritura, solo para filesystem soportados: **rw**
– Definir un propietario para los archivos de la nueva unidad montada (ideal para filesystem que no reconocen los permisos Linux, como vfat, ntfs..): **uid=valor** (donde valor puede ser por ejemplo el id del usuario, 500, podemos ver los id en /etc/passwd)

– Igual que uid pero para el grupo: **gid=valor**
– Montar el dispositivo con una umásk específica. útil para los filesystem que no admiten permisos Unix. Funciona igual que para archivos Linux, es decir en binario y restando esos bits a la máscara 777. umásk=027 generará permisos 750: **umásk=valor**
– Igual que para la umásk de archivos pero solo para directorios: **dmásk=valor**
– Igual que dmásk pero solo para archivos: **fmásk=valor**
– Desactivar las extensiones RockRidge para CD-ROM ISO-9660: **norock**
– Desactiva las extensiones Joliet para CD-ROM ISO-9660: **nojoliet**

Nota: Como es lógico, no se encuentran aquí escritas todas sus opciones, sino aquellas que pueden sernos de mayor utilidad. Para más info: *$man mount*

- **umount**: Desmonta un sistema de archivos.

#umount [-afrnv] [-t file_system] dispositivo | punto de montaje

[Opciones]:
– Desmontar todos los dispositivos con entradas en /etc/mtab y que estén inactivos: **-a**
– Forzar el desmontaje: **-f**
– En caso de que umount no pueda desmontar el dispositivo lo vuelve a montar en solo lectura: **-r**
– Muestra la información: **-v**
– Desmonta sin escribir en /etc/mtab: **-n**
– Especificar el sistema de archivos: **-t <file_system>**

Nota: Al igual que ocurre con *mount*, los usuarios normales no pueden utilizar *umount* a menos que se haya indicado en su montaje alguna opción como *user*, *users* y *owner*.

Comandos para depurar, configurar y reparar sistemás de archivos

badblocks: Busca bloques defectuosos en un dispositivo, normalmente una partición de disco.

 # badblocks dispositivo [primer-bloque] [último-bloque]

[Opciones]

- Especificar el tamaño (en bytes) del bloque (por defecto 1024): **-b**
- Especificar el número de bloques defectuosos 'posibles' antes de finalizar el test automáticamente: **-e**
- Leer desde una lista de bloques malos ya conocidos para evitar que vuelvan a ser leídos y desperdiciar más tiempo: **-i** <fichero>
- Realizar una prueba no destructiva, es decir de solo lectura (no se puede combinar con -w): **-n**
- Escribir la lista de bloques defectuosos en un fichero: **-o** <archivo>
- Mostrar el porcentaje del proceso de escaneado: **-s**
- Usar modo destructivo, es decir de escritura: **-w**

dumpe2fs: Informa sobre la configuración y estado de un sistema de archivo. Aunque se puede ejecutar estando montado el sistema de archivos a diagnosticar, se recomienda que este permanezca desmontado durante la ejecución. Este comando es útil por ejemplo si queremos conocer el tamaño de bloque, el UUID del dispositivo, el tamaño de ínodo, las veces que se ha montado, y un largo etc ya que muestra bastante información útil

#dumpe2fs [opciones] dispositivo

[Opciones]:

- Imprimir bloques marcados como defectuosos: **-b**
- Usar el superbloque para examinar el dispositivo (casos en los que el sistema de archivo está bastante corrupto): **-o superblock=**<superblock>
- Usar un tamaño concreto de bloque (se usa para casos de filesystem corruptos): **-o blocksize=**<blocksize>
- Mostrar solo información relativa al superbloque: **-h**

tune2fs: Permite configurar muchos parámetros del sistema de archivos (ext2, ext3 y ext4) de los que informa dumpe2fs o usando la opción **-l**.

#tune2fs [opciones] dispositivos

[Opciones]:

- Ajustar el número de veces que un filesystem puede ser montado sin que fsck compruebe su estado: **-c**
- Engañar el conteo de número de veces que un sistema de archivos a sido montado sin ser revisado por fsck: **-C**
- Ajustar el intervalo de tiempo que puede transcurrir sin que un filesystem sea comprobado por fsck: **-i** <tiempo> (se añade el sufijo **d** (días), **w** (semanas) o **m** (meses)
- Añadir respaldo de transacciones a un sistema de archivos ext2: **-j**
- Definir un % de bloques reservados en el disco para root (esto es lo que hace mkfs -m) que por defecto es un 5%: **-m**
- Definir el espacio reservado en números de bloques en vez de %: **-r** <bloques>
- Forzar a *tune2fs* a terminar su ejecución aun cuando existan errores. Esta opción es útil sobre todo cuando hemos eliminado el registro .journal o es externo al sistema de archivos: **-f**
- Reservar bloques de un filesystem a un determinado grupo que podemos pasar a través de su guid o nombre de grupo: **-g** <grupo>
- Sobreescribir la configuración del journal creado por la opción -j: **-J size**=journal-size **location**=journal_location **device**=external_journal

Nota: Añadir el respaldo de transacciones a ext2 hará que se cree un archivo *.journal* donde se registraran los eventos pendientes. Si se añade un respaldo de transacciones a un filesystem no montado, el archivo *.journal* de registro será invisible.

Nota: Eliminar un archivo de registro de un filesystem el cual no se ha desmontado de una forma limpia, puede llevar a dejar el sistema inconsistente e incluso a una gran pérdida de datos.

debugfs: Depurador interactivo de filesystem. Puede ser utilizado tanto para comprobar la configuración de un sistema de archivo como para modificarla.

#debugfs [opciones] dispositivo

Nota: Una vez que estemos interactuando con *debugfs* nos permitirá pasarle comandos.

[Opciones]:

- Especificar que el sistema de archivos se abra en modo de lectura y escritura: **-w**
- Forzar al sistema de archivos ser abierto en modo de solo lectura, útil para casos precarios o inconsistencia del filesystem: **-c**
- Pasar a *debugfs* un archivo de comandos para ser ejecutados y finalizar: **-f** <archivo_de_comandos>

[Comandos]:

- Mostrar información sobre los superbloques del filesystem: **show_super_stats | stats** (similar a dumpe2fs)
- Mostrar el nodo de indice de un archivo o directorio: **stat** <nom_file>
- Deshacer la eliminación de un archivo: **(undel)eted** <inodo> <nombre>

Nota: Este recurso puede ser limitado ya que necesitamos el número de inode del archivo eliminado. Podemos intentar obtener una lista de los nodos de indices borrados con **lsdel** o **list_deleted_inodes**

- Extraer un archivo desde un filesystem a otro, útil si tenemos un filesystem dañado y no podemos o queremos montarlo: **write** <archivo-interno> <archivo-externo>
- Podemos manipular archivos con la mayoría de los comandos de Linux: **cd, rm, ln, cat, chroot, mkdir, pwd, quit…**
- Cerrar los filesystem abiertos: **close**
- Obtener ayuda: **list_request | help | lr | ?**
- Salir de debugfs: **quit**

fsck: Comprobar y reparar uno o más sistemás de archivos. En realidad es un front-end de varios comprobadores de filesystem (*fsck.fs_type*)

#fsck [opciones] [opciones-especificas] filesystem

[Opciones]:

- Revisar todos los sistemás de archivos marcados para revisión (1) en /etc/fstab: **-A**
- Indicador de proceso: **-C**
- Salida detallada: **-V**
- Simulación de lo que haría fsck: **-N**
- Especificar el tipo de sistema de archivo: **-t** <*fs_type*> (**-t no** <*fs_type*> para revisar todos los filesystem excepto los que sean del tipo especificado)
- Podemos pasar opciones especificas que *fsck* no entiende pero son válidas para un sistema de archivo concreto, entre ellas:

[Opciones-específicas]:

- Realizar una comprobación automática: **-a** o **-p**
- Realizar comprobación interactiva: **-r**
- Realizar una comprobación forzosa: **-f**
- Realizar ejecuciones de *fsck* en dispositivos en serie en vez de en paralelo: **-s**
- No comprobar sistemás de archivos montados y retornar 0 como código de salida: **-M**
- Escapar la comprobación del sistema de archivos ráiz (/) cuando se especifica la opción -A: **-R**

Nota: Ejecutar fsck solo en sistemás de archivos desmontados o montados en solo lectura

e2fsck: Usado para realizar comprobaciones en los sistemás de archivos ext2, ext3 y ext4. No es aconsejable usar el comando mientras los sistemás de archivos están montados, como excepción podemos usarlo si pasamos la opción **-n** no acompañada de **-l, -L** o **-c**

#e2fsck [opciones] dispositivo

[Opciones]:

- Corregir de forma automática un sistema de archivos siempre y cuando no se necesite la intervención del administrador de sistemás. Esta opción no podrá usarse junto a -n o -y: **-p** o **-a** (en desuso)
- Usar un superbloque diferente, útil cuando el primer bloque del dispositivo está dañado: **-b** <superbloque>
- Forzar a *e2fsck* a buscar un superbloque de tamaño específico: **-B** <tamaño>
- Localizar bloques defectuosos y añadirlos a la lista de bloques erróneos para que no sean reusados en nuevos archivos o directorios. Podemos especificar esta opción 2 veces para realizar una prueba de lectura-escritura no destructiva sobre el bloque: **-c**
- Abrir el sistema de archivo en modo de solo lectura y asumir como NO todas las preguntas interactivas: **-n**
- Conservar los bloques defectuosos en su lista de bloques malos y añadir a esta lista nuevos bloques defectuosos (hay que acompañarla con la opción -c): **-k**

xfs_info: Muestra información técnica sobre el filesystem. Es necesario que este esté montado (a diferencia de otras tantas herramientas de bajo nivel).

xfs_ metadump: Herramienta de depuración para sistemás de archivos XFS. Copia la información de un sistema de archivos XFS a un archivo, para ser analizado posteriormente.

#xfs_metadump [opciones] dispositivo archivo_dump

Nota: xfs_metadump no crea copias de seguridad!!

[Opciones]:

- Mostrar el progreso del volcado: **-g**
- Ignorar las lecturas de errores y copiar los metadatos que son accesibles unicamente: **-e**
- Especificar que el filesystem a procesar es guardado en un archivo regular: **-f**
- Mostrar por pantalla los errores encontrados: **-w**

xfs_admin: Es el equivalente en "bruto" a *tune2fs* pero para sistemás de archivos XFS. Permite cambiar parámetros de configuración del filesystem, para ello xfs_admin hace uso del comando xfs_db.

#xfs_admin [opciones] dispositivo

[Opciones]:

- Habilitar la versión 2 de respaldo de transacciones: **-j**
- Obtener el nombre y UUID del sistema de archivos: **-l** y **-u** respectivamente
  Nota: Para tal fin también podemos usar el comando **blkid**
- Definir la etiqueta o UUID del sistema de archivo: **-L** <Etiqueta> o **-U** <UUID>

Nota: La etiqueta tiene un máximo de 12 caracteres y el UUID puede utilizarse el que tiene antes de ser formateado o modificado el filesystem o bien utilizar el argumento "generate" como valor de UUID de modo que xfs_admin genere uno nuevo

Nota: Los sistemás de archivos deben de ser desmontado antes de ejecutar cualquier modificación

xfs_ncheck: Provee una lista de inodos de indice y rutas de archivos si se emplea sin opciones

xfs_check: Comprueba si un sistema de ficheros es consistente.

#xfs_check [opciones] dispositivo

[Opciones]:

- Especificar que se trata de una imagen de archivo regular **-f** <archivo.img>
- Especificar la ruta del log en un filesystem externo: **-l** <log-file>
- Mostrar solo aquellos errores de suma importancia: **-s**

xfs_repair: Repara un sistema de archivo corrupto. El sistema de archivo debe ser desmontado.

#xfs_repair [opciones] dispositivo

[Opciones]:

- No reparar, solo indicar que es lo que anda mal: **-n**
- Desactivar la obtención previa de inodos de indices y directorio. Útil si la ejecución de *xfs_repair* se atasca o deja de responder: **-P**
- Anular la cantidad de memoria RAM máxima que usa *xfs_repair* (un 75% por defecto): **-m** <maxmem>
- Permitir que *xfs_repair* repare un sistema de archivos montado en modo de solo lectura, típico para reparar el directorio raíz. Hay que reiniciar una vez terminado: **-d**

Comandos para monitorizar el uso del disco

du: Muestra el tamaño que ocupa un directorio, incluidos todos los que cuelgan de él.

#du [opciones] directorios

[Opciones]:

- Mostrar además del espacio total ocupado por un directorio, muestra también el espacio individual ocupado por cada uno de sus archivos: **-a** (–all)
- Mostrar la suma total en la última línea: **-c** (–total)
- Mostrar los tamaños acompañados de un sufijo (B, M, K, G): **-h**
- Mostrar el tamaño en Kb: **-k**
- Mostrar el tamaño en Mb: **-m**
- Contar los enlaces simbólicos: **-l** (–count-links)
- Limita la salida del informe a n niveles, es decir hasta que subdirectorio queremos que aparezca en el informe. Aunque los que estén por debajo de ese nivel no aparezcan, su espacio si que será contado por lo que se sumará al total: **–max-depth=n**
- Indicar en la última línea el tamaño total del contenido del directorio: **-c**
- Mostrar solo la línea con el tamaño total del contenido del directorio (es como –max-depth=0): **-s** (–summarize)
- Si no queremos incluir el tamaño de un sistema de archivo que está montado sobre un directorio del que queremos conocer su espacio ocupado: **-x** (–one-file-system)
- No incluir el tamaño de los subdirectorios del directorio especificado: **-S** (–separate-dirs)
- Excluir las entradas de menor tamaño que el especificado (si es positivo) o las de mayor tamaño (si el tamaño es negativo): **-t** (–threshold=size)
- Excluir archivos/directorios específicos según un patrón: **–exclude=pattern**
- Mostrar la fecha de ultimo acceso o creación: **–time=ctime|atime|access|use|status**
- Excluir de la lista los archivos/directorios que indiquemos mediante un archivo: **-X** <archivo_con_patrones> (–exclude-from=FILE

df: Informa sobre el uso del espacio de los diferentes sistemás de archivos y dispositivos montados en el sistema.

#df [opciones] [dispositivo]

[Opciones]:

- Incluir todos los pseudosistemás de archivos con tamaño 0 (/proc, /sys, /proc/bus/usb, etc..): **-a** (–all)
- Agregar una letra como sufijo para indicar la unidad de medida (por ejemplo K para Kb): **-h**
- Mostrar el espacio en Kb: **-k**

- Mostrar el espacio en Mb: **-m**
- En vez de mostrar el uso de espacio, mostrar los nodos de indices: **-i** (–inodes)
- Omitir los sistemás de archivos en red, es decir solo muestra los locales: **-l** (–local)
- Hacer una llamada a sync antes de desplegar la información: **–sync**
- Mostrar el tipo de sistema de archivo: **-T**
- Mostrar solo aquellos dispositivo que tengan un cierto sistema de archivo: **-t** <fs_type> (–type=fs_type)
- Mostrar solo los dispositivos que no tengan el tipo de sistema de archivo especificado: **-x** <fs_type> (–exclude=fs_type)

Nota: Está opción es útil para sistemás de archivos que crean un número fijo de nodos de indices como ext2, ext3, ext4, XFS y otros sistemás, pero no para sistemás como Reiserfs, que crean nodos dinámicamente.

blkid: Más que un comando para monitorizar el uso del disco, es un comando que nos proporcionará una buena información acerca del disco como por ejemplo: el nombre del dispositivo de bloques, el UUID de los dispositivos, etiquetas y sistemás de archivos.

$ sudo blkid [opciones] [dispositivo]

Opciones:

- Mostrar con un formato diferente: **-o** <formato>

Formatos:

– **list** : Muestra el dispositivo, tipo de sistema de archivos, etiqueta, punto de montaje y UUID

– **device**: Muestra solo los dispositivos (/dev/sda1, /dev/sdb1…)

– **export**:Recomendable para redireccionar la salida a un archivo. Muestra la etiqueta, el la siguiente línea el UUID y en la última línea (para ese dispositivo) el tipo de sistema de archivos. Los dispositivos están separados por una línea en blanco.

– **value**: Igual que export pero muestra solo los valores. Es decir omite el LABE=, UUID=,etc… Además no se separan los dispositivos con líneas en blanco

– **full**: Este es el formato por defecto

```
/dev/sda1: LABEL="MintCinn" UUID="6324422a8-5e6c-4bd3-b7a3-
f6d22906b9f9" TYPE="xfs"
```

- Mostrar todos los nombres de sistemás de archivos y RAID conocidos: **-k**
- Mostrar solo determinados tags (LABEL, UUID y TYPE) junto con el dispositivo: **-s** <tag>
- Mostrar el dispositivo que tiene una etiqueta determinada: **-L** <etiqueta>
- Mostrar el dispositivo que tiene un UUID determinado: **-U** <uuid>

COMANDOS LPIC-1

Capítulo 4

Comandos de archivos

ls : Lista el contenido de un directorio o muestra la información de un archivo (permisos, propietario, grupo, tipo de archivo, etc..)

ls [opciones] [directorio|archivo]

Opciones:

- Mostrar todas las entradas de un directorio incluyendo los archivos ocultos: **-a** (–all)
- Ordenar la lista de archivos más nuevos a más antiguos: **-t**
- Ordenar la lista de archivos más antiguos a más nuevos: **-c**
- Ordenar por el tamaño de archivos: **-S**
- Invierte el orden dado por -l : **-r**
- Añadir el indicador de archivo a cada entrada (*/=>@|): **-F**
- Mostrar el tamaño de los archivos en M,K o G: **-h**
- Imprimir el número de indice de cada archivo: **-i**
- Mostrar todas las entradas pero separadas por ‘,': **-m**
- Lista el contenido de forma recursiva: **-R**
- Listar un archivo por línea: **-1**

file : Determina el tipo de archivo

file archivo

Este comando tiene opciones pero poco habituales, quizás su opción **-z**, que intenta mostrar el tipo de archivo que se encuentra dentro de un archivo comprimido.

cp : Copia archivos

cp [opciones] archivo destino

Opciones:

- Forzar la copia (se ignora si se acompaña de la opción -n) : **-f**
- No permite sobreescribir un fichero: **-n**
- Interactúa con el comando al realizar una acción: **-i**
- Copia directorios de forma recursiva: **-r** o **-R**
- Crea un enlace simbólico a un archivo dentro del mismo directorio: **-s**
- Crea un hard link a un archivo en un destino diferente al directorio actual: **-l**
- Preservar los permisos y marcas temporales: **-p**

mv : Mueve un archivo, directorio y cambiarles el nombre

mv [opciones] origen destino

El comando *mv* admite las opciones **-f** , **-i**, **-n** de *cp* y otra opción útil sería la **-u** que solo mueve si origen es más nuevo que el destino

rm : Elimina un archivo o directorios con contenido

rm [opciones] archivo

Este comando acepta las opciones **-f**, **-i** y **-r** (**-R**). Podríamos usar **-d** (–dir) para eliminar directorios vacías. Para directorios con contenido usamos **-r** o **-R**

rm -rf directorio

Eliminará el directorio y todo su contenido sin preguntar. Muchas veces el mal uso de este comando podría provocar que quisiéramos cortarnos las venas!!!!

touch : Modifica las marcas de tiempo de un archivo o lo crea si no existe

touch [opciones] archivo

Opciones:

- Cambiar solo el tiempo de acceso: **-a**
- Cambiar la hora de ultima modificación: **-m**
- Usar la hora de un archivo en vez de la hora actual: **-r** <archivo>
- Actualizar la hora de un archivo a la actual: **-d**
- Pasar una fecha nosotros mismos: **-t** FECHA , donde fecha = [[CC]YY]MMDDhhmm[.ss]

Comandos para comprimir archivos

gzip : Comprime un archivo. Crea un nuevo archivo .gz y elimina el origen.

gzip [opciones] archivo

Opciones:

- Si no queremos eliminar el origen podemos usar la opción keep (**-k**) o redireccionar la salida, que además nos permite por ejemplo concatenar varios archivos.

```
$ gzip -k archivo
$ gzip -c archivo1.txt ab.txt ultimo.txt > unidos.txt.gz
```

Cuando descomprimamos *unidos.txt.gz* tendremos un único archivo *unidos.txt* que si desplegamos su contenido con *cat* veremos como se han concatenados los 3 archivos de textos (archivo1.txt, ab.txt y ultimo.txt). Si usamos **-c** con –**d** no se elimina el archivo comprimido al descomprimir su contenido.

- Para descomprimir usamos la opción **-d**:

```
$ gzip -d unidos.txt.gz
```

- Forzar la compresión o descompresión: **-f**
- Mostrar información sobre el archivo comprimido (tamaño comprimido, tamaño descomprimido, ratio de compresión y nombre del archivo descomprimido: **-l**
- No conserva la fecha ni el *timestamp* del archivo: **-n**
- Comprimir de forma recursiva sobre un directorio. No comprimirá un directorio en un solo *archivo.gz* si no que descenderá a dicho directorio, comprimirá todos sus archivos uno a uno y si encuentra un directorio, descenderá igualmente a él y comprimirá uno a uno todo los archivos que encuentre: **-r**

```
$ gzip -r directorio
```

- Para restablecer todos los archivos comprimidos por el comando anterior usamos **gunzip**:

```
$ gunzip -r directorio
```

- Realizar test a un archivo gz: **-t**

bzip2 : Comprime usando los algoritmos de compresión de *Burrows-Wheeler* y de codificación de *Huffman*. Ofrece mejor compresión que *gzip*. *bzip2* admite mucha de las opciones de *gzip* como: **-c**, **-d**, **-t**, **-f**, **-k** y **-q** (que omite warnings, no comentado en *gzip*, pero también válido para ambos)

- Para comprimir usamos la opción **-z**:

```
$ bzip2 -z archivo
```

- Para descomprimir también podemos usar **bunzip2**. Igualmente el origen se pierde a menos que usemos **-k** o redirecciones su salida con **-c**

xz : Comprime y descomprime archivos. Para muchos mejor herramienta que las 2 anteriores (*gzip* y *bzip2*). Admite gran parte de las opciones de los dos anteriores como por ejemplo: **-z** para comprimir (puede ser omitido), **-d**, **-c**, **-k**, **-f**, **-l** (muestra información), podremos agregar a **-l** una **v** para mayor información o incluso otra **v** (**-lvv**) para obtener más datos aún. Al igual que *gzip* y *bzip2* si no pasamos la opción **-k** o redirigimos la salida, el archivo de origen será eliminado. La opción **-f** (–force) de *xz* nos permite comprimir aun cuando el archivo es un enlace simbólico, tiene varios enlaces de referencia (hard links) o el archivo contiene uno o todos los permisos especiales como *SUID*, *GUID* y *Sticky bit*. Los permisos especiales no serán aplicados al nuevo archivo *.xz* creado.

xz conserva la sintaxis de los dos anteriores:

```
$ xz [opciones] archivo
```

Además de las opciones anteriormente comentadas, la herramienta *xz* nos brinda otras oportunidades como las siguientes:

[Opciones]:

- Podemos comprimir en un formato diferente, por ejemplo *lzma* que ofrece mejor compresión:

```
$ xz -F lzma archivo
```

- Para descomprimir tenemos otras opciones:

```
$ unxz archivo.xz
 $ unlzma archivo.lzma
$ xzcat archivo.xz > archivo_destino
$ lzcat archivo.lzma > archivo_destino
```

- Podemos pasar nombres de archivos que queremos comprimir a través de un archivo. El nombre de cada archivo debe de ir en una línea independiente y además finalizar con salto de línea.

```
$ xz --file=archivo-de-nombres-de-archivos
```

- Cambiar el nivel de compresión (por defecto es -6)

```
$ xz -8 archivo
```

Comentar que como es lógico a mayor nivel de compresión más RAM será necesaria. Con *gzip* y *bzip* es normal el uso de -9 pero se desaconseja en *xz*.

- Podemos fijar un límite de memoria RAM utilizada en el proceso de compresión:

```
$ xz -8 -M 32MiB archivo
```

Tabla de nivel de compresión y memoria usada:

-0 : 256KiB
-1 : 1MiB
-2 : 2MiB
-3 y -4 : 4MiB
-5 y -6 : 8MiB
-7 : 16MiB
-8 : 32MiB
-9 : 64MiB

Nota: Podemos usar el sufijo '**-e**' tras el nivel de compresión que ralentizará el proceso de compresión pero podría alcanzar mejores resultados sobre todo si lo usamos con los niveles -3 y -5

- Hacer que *xz* comprima usando un archivo de entrada que es la salida de otro comando:

```
$ tar -cf - foo | xz -3e > foo.tar.xz
```

- Concatenar varios archivos en uno solo:

```
$ xz -dcf a.txt b.txt.xz cl.txt dado.lzma > abcd.txt
```

Se añadirá *a.txt* a *abcd.txt* posteriormente se concatena el contenido de *b.txt.xz*, el cual es descomprimido en tiempo real y 'pegado' en *abcd.txt*, se añade *cl.txt* y de igual manera se descomprime *dado.lzma* y se añade a *abcd.txt*, dando como resultado un solo archivo con el contenido de los 4, aun estando 2 de ellos en formato comprimido.

Comandos para agrupar archivos y realizar copias de seguridad

tar : Agrupar varios archivos en uno solo (empaquetar).

tar [opciones] archivo.tar <archivos_a_empaquetar>

Opciones:

- crear un archivo tar: **-c**
- El siguiente parámetro será el nombre del archivo tar: **-f**
- Extraer archivos de un paquete tar: **-x** (–extract, –get)
- Añadir nuevos archivos a un paquete tar: **-r** (–append)
- Añadir archivos que ya se encuentran en el paquete, solo que estos a añadir son versiones posteriores: **-u** (–update)
- Añadir un archivo tar al final de otro archivo tar: **-A**
- Desplegar el contenido de un archivo tar: **-t**
- Comparar un archivo con los archivos del disco: **-d** (–diff, –compare)
- Pasar a un directorio X antes de desomprimir el paquete: **-C**
- Crear una copia incremental: **-g**
- Conservar la información de protección: **-p**
- Enumerar los archivos léidos o extraídos: **-v**
- Excluir un archivo del paquete tar: **–exclude** <archivo>
- Comprime un paquete .tar con gzip: **-z** (–gzip o –ungzip)
- Comprime un paquete .tar con bzip2: **-j** (algunas versiones utilizan I o y)
- Comprime un paquete .tar con xz: **-J** (–xz)
- Crear archivo tar de varios volúmenes o partes (junto a -c): **M**
- Hacer que tar se comporte de forma interactiva: **-w**
- Verificar los archivos después de haber sido añadidos al archivo tar: **-W**

Algunos ejemplos:

- Descomprimir un paquete *tar* comprimido con *gzip*, conservando sus permisos, en un directorio diferente de donde se encuentra el paquete:

```
$tar -xpvzf paquete.tar.gz -C /otro/directorio/
```

- Empaquetar un paquete comprimiéndolo con *xz* y excluyendo archivos mediante un patrón:

```
$tar -cpvJf todos-menos2.tar.xz ./* --exclude=*.txt
```

cpio : Similar a *tar*. Ambos comandos fueron creados para realizar copias de seguridad, aunque su funcionamiento difiere en algunos aspectos. Vamos a repasar algunos y opciones de *cpio*.

Opciones:

- Crear un nuevo archivo: **-o** (*output*)
- Extraer desde un archivo: **-i** (*input*)
- Copiar una estructura de directorios: **-p** (*copypass*)
- Reiniciar el tiempo de acceso de los archivos una vez copiados: **-a**
- Preservar tiempos de modificación (usado con -i): **-m**
- Crear directorios en caso de que hiciesen falta (usado con -o y -i): **-d**
- Especificar un archivo que contiene un patrón (usado con -i): **-e**
- Listar el contenido de un archivo (usado con -i): **-t**
- Solo escribe archivos existentes: **-u**

Copy-Out:

- Crear copia de seguridad de todo el contenido del Escritorio, sin comprimir o comprimido (respectivamente):

```
$ find /home/user/desktop/ | cpio -o > /tmp/miescritorio.cpio
$ find /home/user/desktop/ | cpio -o | gzip /tmp/miescritorio.cpio.gz
```

Copy-In:

- Restablecer el contenido de la copia de seguridad anteriormente creada:

```
$ cpio -i < /tmp/miescritorio.cpio
```

- ó solo un archivo:

```
$ cpio -i solo-este-archivo < /tmp/miescritorio.cpio
```

Copy-Pass:

- Copiar el árbol de directorio completo (comenzando desde el directorio en el que nos encontramos) en el directorio /tmp de nuestro equipo:

```
$ find . -depth -print | cpio -pvd /mnt/destino
```

- Crear un backup de /home, /usr y /opt en una cinta:

```
$find /home /opt /usr -xdev | cpio -oF /dev/st0
```

dd : Copia archivos. Es usado para copiar particiones completas o incluso clonar discos. Vamos a estudiar este comando mediante ejemplos:

Sintaxis: dd if=origen of=destino

- Clonar disco duro *sda* en *sdb*:

```
$ sudo dd if=/dev/sda of=/dev/sdb bs=1M
```

- Podemos clonar solo los sectores legibles:

```
$ sudo dd if=/dev/sda of=/dev/sdb bs=1M conv=noerror,sync
```

- Copiar un disco completo a la primera partición del disco *sdb*

```
$ sudo dd if=/dev/sda of=/dev/sdb1 bs=1M
```

- Crear una imágen de un disco duro o CD

```
$ sudo dd if=/dev/sda of=/home/syu/sda.[bin|iso]
$ sudo dd if=/dev/cdrom of=/home/syu/myCDRROM.iso
```

- Rellenar un disco o partición con 0, es decir elimina su contenido:

```
$ sudo dd if=/dev/zero of=/dev/sda[X]
```

- Para borrar solo el primer sector:

```
$ sudo dd if=/dev/zero of=/dev/sda count=1 bs=512
```

- Para eliminar el cargador de arranque, pero no la tabla de particiones:

```
$ sudo dd if=/dev/zero of=/dev/hda count=1 bs=446
```

- Respaldar el sector de arranque (MBR):

```
$ sudo dd if=/dev/sda of=/home/syu/Desktop/mbr.backup count=1 bs=512
$ sudo dd if=/home/syu/Desktop/mbr.backup of=/dev/sda
```

- Crear un archivo de *swap* sin espacios vacíos:

```
$ dd if=/dev/zero of=/extra-swap bs=1024 count=1024
```

El tamaño conviene que sea un múltiplo de 4, debido a que el kernel escribe páginas de memoria (memory pages), de 4 kb de tamaño. Si el tamaño no es múltiplo de 4 los últimos kilobytes del
archivo pueden ser desperdiciados. Podríamos haber indicado una velocidad mayor como de bs=4k.

Expliquemos las opciones aquí mencionadas y algunas otras:

- **ibs**: Lee N bytes del archivo de origen
- **obs**: Escribe N bytes en el archivo destino

- **bs**: Se usa para indicar un mismo tamaño para ibs y obs. Es la 'velocidad' de lectura/escritura. Si bs=4k se leerá/escribirá en bloques de 4Kb. A más velocidad menos seguro será el tratamiento de la información.
- **count**: Es el tamaño a copiar. Para el ejemplo de la memoria swap. Se creará una partición de 1GB
- **skip**: Se salta N bloques del archivo origen antes realizar la operación de copiado. El tamaño es dado por bs o ibs
- **seek**: Se salta N bloques del archivo destino antes realizar la operación de copiado. El tamaño del bloque es indicado por bs o obs.
- **conv**=modo,[modo2,…]: Realiza conversiones o acciones. Se pueden pasar varios parámetros separados por ',' . Por ejemplo conv=noerror,sync

Modos:

- **noerror**: No se detiene el proceso ante errores de lectura en el origen
- **sync**: Rellena cada bloque leído con ceros
- **ascii**: Convierte los caracteres a *ascii*
- **lcase**: Convierte caracteres a minúsculas
- **ucase**: Convierte caracteres a mayúsculas
- **notrunc**: No se trunca el archivo utilizado como destino

Comandos de directorios

ln : Crea enlaces con archivos/directorios

ln [opciones] fichero-de-origen <nombre-enlace>

Opciones:

- Intentar que el súper usuario cree enlaces duros a directorios (seguramente falle debido al sistema de archivos): **-d**
- Eliminar posible archivo de destino: **-f**
- Ejecutar en modo interactivo: **-i**
- Crear enlace simbólico: **-s** (–symbolic)

mkdir : Crear directorios

mkdir [opciones] nuevo_directorio

Opciones:

- Crear el directorio con un modo determinado: **-m** <modo>
- Crear directorios padres si no existen: **-p**

Imaginemos que el directorio */home/syu/noexist* , no existe y sobre este (que no existe queremos crear un directorio “nuevodir”):

```
$ mkdir -p /home/syu/noexist/nuevodir
```

Creará primero *noexist* y luego *nuevodir*

rmdir : Elimina directorios vacíos

rmdir [opciones] directorio

Comandos para la gestión de ACLs

setfacl : Crea una lista de acceso para un determinado directorio o archivo.

setfacl [opciones] archivo

Vamos a estudiar las *Listas de Control de Acceso* con ejemplos prácticos, como venimos haciendo en otros comandos 'robustos' o de gran interés.

IMPORTANTE: Antes de crear una lista de acceso, por ejemplo del directorio '*/home/usuarios/sistemás*' es conveniente realizar un respaldo de los permisos existentes, para que en caso de que queramos anular la ACL en un futuro, podamos usar este backup.

- Crear un respaldo de los permisos para /home/usuarios/sistemás

```
$ getfacl -R /home/usuarios/sistemás >
/home/usuarios/Desktop/persistemás.bak
```

- Restaurar los permisos de un directorio en caso de que apliquemos una ACL errónea

```
$ setfacl --restore=permsistemás.bak
```

De momento vamos a imaginar que tenemos un directorio '*/home/usuarios/sistemás*' donde diferentes usuarios pueden acceder pero en concreto queremos que sea directorio para el grupo de sistemás. Lo primero que vamos a llevar a cabo antes de implementar ACL sobre este directorio, será dejarlo 'limpio', es decir, anularemos otras posibles listas de acceso que este pudiese tener o incluso si tuviese una ACL por defecto. Esto lo haremos con el siguiente comando:

```
$ sudo setfacl -b -k -R /home/usuarios/sistemás
```

Opciones:

- Eliminamos la posible ACL que ya pudiese tener el directorio: **-b**
- Eliminamos la posible ACL por defecto que pudiese tener el directorio: **-k**
- Aplicamos los cambios de forma recursiva: **-R**

Ahora con el directorio limpio vamos a crear la ACL, acción que podremos realizar mediante el uso de dos parámetros: '**s**' o '**m**'. Si quisiéramos crear una ACL eliminando una ya existente usaríamos '**s**', si por el contrario queremos modificar una ACL o crearla si no existe usaremos '**m**'. Nosotros usaremos '**m**' ya que antes hemos eliminado todo lo relacionado con ACL en el directorio.

```
$ sudo setfacl -R -m g:sistemás:rw
```

Aquí igualmente hemos aplicado de forma recursiva (**-R**) el comando. Vamos a desglosar la cadena '**g:sistemás:rw**' para ver que significado tiene cada uno de sus campos. Como nota añadir que el 3º campo es opcional a la hora de eliminar una ACL.

- Indicar si se trata de una ACL de usuario (**u**) o de grupo (**g**): En nuestro caso, de grupo
- El segundo campo es el nombre del grupo (podemos pasar el GID igualmente): sistemás
- El tercer y ultimo campo son los permisos de la ACL, podemos pasar en valor octal como con chmod: **rw**

Ahora vamos a darle al grupo de explotación permisos de lectura y acceso para el subdirectorio */home/usuarios/sistemás/prod* pero además algo que no hemos hecho en la anterior ACL es crearla como **default**, esto implica que cada subdirectorio o archivo que se cree bajo ./prod heredará los permisos, muy útil para no tener que andar modificando la ACL continuamente (cada vez que se cree contenido). Usaremos la opción '**-d**'

```
$ sudo setfacl -d -R -m g:explotacion:rx /home/usuarios/sistemás/prod
```

[Otras Opciones]:

- Restaurar los permisos anteriores a partir de un archivo: **–restore**=FILE
- Cuando usamos **-R** podemos incluir o no los enlaces simbólicos (respectivamente): **-L** o **-P**

getfacl : Muestra las entradas de *Listas de Control de Acceso* para un archivo o directorio

getfacl [opciones] archivo

Una vez terminada la tarea de setear la ACL es conveniente usar el comando *getfacl* para comprobar como ha quedado todo:

```
$ sudo getfacl /home/usuarios/sistemás
```

```
# file: home/usuarios/sistemás
# owner: nebul4ck
# group: nebul4ck
user::rwx
group::r-x
group:sistemás:rw-
másk::rwx
other::r-x
```

Nota: Si hemos usado la opción '**-d**' (default) *getfacl* mostrará además estás entradas:

default:user::rwx
default:group::r-x
default:group:sistemás:rw-
default:másk::rwx
default:other::r-x

Donde:

- **user**: permisos para nebul4ck
- **group**: permisos del grupo nebul4ck
- **group**:sistemás: permisos rw- para el grupo sistemás (habrá tantas entradas como ACL hayamos asignado)
- **másk**: Esta entrada se puede manipular con setfacl y permite especificar el máximo de permisos que se pueden asignar en dicho fichero con las ACLs de usuario y grupo.
- **Other**: Es la entrada de los permisos generales o globales del modelo UGO (**u**ser**g**roup**o**ther).

[Otras Opciones]:

- Listar recursivamente: **-R**
- Listar solo la ACL (no muestra default): **-a**
- Listar solo la ACL default: **-d**
- Omitir los comentarios '#': **-c**
- Mostrar en vez el nombre de usuario y grupo, los UID y GID: **-n**

Comandos para cambiar las propiedades de los archivos

chown: Cambia el propietario y el grupo de un archivo

chown [opciones] propietario:grupo archivo

Opciones:

- No imprimir mensajes de error: **-f**
- Modificar de forma recursiva: **-R**

chgrp: Cambia el grupo propietario de un archivo

chgrp [opciones] grupo <archivos>

Opciones: Acepta las mismás que *chown*, destacamos **-f** y **-R**

chmod: Modificar los modos de un archivo

chmod [opciones] modo <file>

Opciones: Igualmente destacamos **-f** y **-R**

Nota: Los modos se pasan en octal o en modo simbológico: 754 o u=rwx,g=rx,o=r

newgrp: Cambia de grupo propietario para nuevos archivos

newgrp nuevo-grupo

Nota: El usuario que lanza el comando, debe de pertenecer ya al grupo '*nuevo-grupo*'

umásk: Modifica la máscara por defecto para los archivos y directorios creados en el sistema. Este comando está detallado en el Tema-4 en su correspondiente apartado. Básicamente lo que hacemos es pasar un número en octal (por ejemplo 002) y será restado al máximo valor posible de permisos 777 (para directorios) y 666 (para archivos), quedando en este caso permisos 775 para nuevos directorios y 664 para nuevos archivos. Cuando se trata de archivos el bit de ejecución no se implementa por lo que aunque tengamos una umásk de 047, se quedará 620 pues el 4 no resta a 7 si no a 6 y el 7 resta todo (siendo el máximo 6, pero es igual, no afecta, no nos vamos a quedar en permisos -1!!)

chattr: Modifica los atributos de un archivo.

chattr [opciones] [+|-|=]Atributos <archivo>

Opciones:

- No imprimir mensajes de error: **-f**
- Modificar de forma recursiva: **-R**

Atributos:

- No se modifica la fecha del ultimo acceso en el archivo: **A**
- Cuando se trata de un directorio hacer que los datos se escriban de forma síncrona (**sync**, inmediatamente) en el disco: **D** (para kernel 2.5.19 o superiores)
- Trabajar de la misma forma que con el atributo D pero sobre archivos: **S**
- Utilizar extensiones para el mapeo extendido de los bloques en el disco: **e**
- No poder realizar cambios sobre un archivo, es decir el archivo se vuelve inmutable: **i**
- Establecer que un archivo sea escrito en journal en sistemás ext3 montados con las opciones '*data=ordered*' o '*data=writeback*' journal: **j** (útil si usamos ext3)
- Permitir que el contenido de un archivo sea guardado, permitiendo recuperar el archivo tras ser eliminado: **u**
- Borrado seguro. Los bloques son escritos con ceros en el disco: **s** (no valido para la familia de sistemás de archivos extendidos)

lsattr: Muestra los atributos de un archivo

lsattr <archivo>

Comandos para la gestión de cuotas de disco

quotaon: Activa el sistema de limites de cuotas para los sistemás de archivos marcados para usarlas.
quotaoff: Desactiva el uso de límites de cuotas para los sistemás de archivos que se encuentran activado y en uso.

edquota: Editar cuotas de usuario

edquota [opciones] usuario

- Opciones:

  Modificar cuotas de grupos: **-g**

quotacheck: Verifica y actualiza la información de las cuotas en los discos con cuotas activadas, además de asegurar que la contabilidad de las cuotas sea la adecuada.

quotacheck [opciones] filesystem

Opciones:

- Forzar a quotacheck para hacer copias de seguridad del archivo de cuotas antes de escribir los nuevos datos. **-b**
- Habilitad debug mode: **-d**
- Comprobar las cuotas de usuarios en los sistemás de archivos listados en mtab (predeterminado): **-u**
- Comprobar las cuotas de grupos para los sistemás de archivos marcados en /etc/mtab: **-g**
- Comprobar todos los sistemás de archivos menos NFS: **-a**
- Excluir el sistema de archivos ráiz: **-R**

repquota: Resume la información de las cuotas del sistema de archivos especificado

repquota [opciones] filesystem

Opciones:

- Resumir la información de todos los filesystem: **-a**
- Resumir la información de cuotas de usuario (predeterminado): **-u**
- Resumir la información de cuotas de grupos: **-g**
- Trata de mostrar el espacio utilizado y los inodos: **-s**

quota: Muestra el uso del disco y los límites. Posee un efecto similar. quota **-g** muestra las cuotas del grupo, con **-l** omite los montajes NFS, **-q** limita la salida a los sistemás de archivos cuyo uso sobrepasa el límite, **-i** ignora los puntos de automontaje, **-A** muestra cuotas de NFS y **-f** <filesystem> muestra cuotas del sistema de archivos especificado

Comandos para la búsqueda de archivos

find: Localiza los archivos buscando por todo el árbol de directorios especificado, revisando los nombres de los archivos, sus fechas de creación, etc. El comando *find* permite una gran cantidad de opciones y expresiones, es muy recomendable acudir a las páginas man, intentaremos cubrir gran parte pero aun así, no será suficiente.

find [OPCIONES-GENERALES] <PATH> [OPCION1 <argumento> [OPCION2 <argumento>] [OPCION3 <argumento>]…] [COMANDO] <argumentos comando>

La sintaxis es extensa y no es clara, pondré un ejemplo para que podamos verlo mejor y a continuación detallaremos algunas de sus opciones y comandos.

```
find [-H|-L|-P] /home/syu/* -name "*.txt" -type f -size 10M -perm -640
-exec rm -rf {} \;
```

- Opciones Generales:

-H | **-L** | **-P** : Destinadas a enlaces simbólicos, -P nunca busca enlaces simbólicos, -L siempre y -H los sigue con excepciones (man find). Estas opciones no suelen usarse habitualmente.

- Ruta de búsqueda (PATH):

*/home/syu/** Indica la ruta desde la cual se empezará la búsqueda. Por defecto find es recursivo, existen opciones para indicar hasta que nivel queremos bajar.

Opciones:

- **name**, **-type**, **-size**, **-perm** : Son opciones (nombre, tipo, tamaño y permisos, respectivamente) a las que le pasamos sus correspondientes argumentos.

Comandos a ejecutar:

- **-exec** es la acción que en este caso indica que una vez encontrado los resultados se ejecute el comando que se indica a continuación en este caso '**rm**' el cual toma como entrada el nombre de los archivos recogidos 1 a 1 mediante **{}** y por último cerramos la línea con **\;**

Otras opciones:

Nota: Cuando indiquemos una opción con un argumento '**n**' existirá la opción de añadir los prefijos + o –. Si ningún prefijo es añadido querrá decir que es un valor exacto.

- Descender hasta X niveles desde el directorio de búsqueda: **-maxdepth** <nivel>
- Se accedió al archivo por última vez hace n minutos: **-amin** n
- Se accedió al archivo por última vez hace n días: **-atime** n
- El estado del archivo fue modificado hace n minutos: **-cmin** n
- El estado del archivo fue modificado hace n días: **-ctime** n
- Los datos del archivo fueron modificados por última vez hace n minutos: **-mmin** n
- Los datos del archivo fueron modificados por última vez hace n días: **-mtime** n
- El archivo está vacío y además es o un directorio o un archivo regular : **-empty**
- Buscar por el número ID de grupo o usuario respectivamente: **-gid** <id> o **-uid** <id>
- Buscar por el nombre de grupo o usuario respectivamente: **-group** <name> o **-user** <name>
- Buscar por nombre de archivo pasando un patrón de búsqueda: **-name** <patrón>
- Buscar por nombre pero no distingue de mayúsculas y minúsculas: **iname** <patrón>
- Podemos buscar enlaces simbólicos por nombre: **-lname** <patrón> o **-ilname** <patrón>
- Buscar por tipo de archivo: **-t** <tipo>
  Nota: los tipos pueden ser algunos como *b* (bloque), *c* (caracter), *d* (directorio), *f* (fichero regular) o *l* (enlace simbólico)
- Buscar por tamaño: **-size** n[KMG]
- Buscar por permisos de archivos:

-perm mode: Se usa esta forma para expresar los permisos exactos que tiene que tener un archivo determinado. Por ejemplo:

-perm 0020 ó **-perm g=w** : Solo el grupo tiene permisos, y deben ser de escritura

-perm -mode y *-perm /mode*: No son tan restrictivos. Suelen ser las formás habituales. Aceptan permisos octal y en modo simbólico.

Archivos con permisos de lectura para todos los usuarios: **-perm -444** y **-perm -a+r**
Archivos con permisos de lectura y escritura para propietario, lectura para grupo: **-perm /640** ó **-perm /u=rw,g=r** ó **-perm /u+rw,g+w**

Algunos ejemplos *find*:

- Si usamos la opción **print** (imprimir todos los nombres de archivos que concuerdan con una expresión) debemos de asegurarnos que lo hacemos al final de la línea ya que *find* cuando ve la opción *-print* asume que no se proporciona ningún criterio de búsqueda. Supongamos que recordamos que el archivo que buscamos contenía la palabra departamento en su nombre:

```
$ sudo find -name "*departamento*" -print
```

- Podemos usar varios criterios y separarlos con un '**espacio**' o **-a** (cosa que *find* representará como un *AND*), **-or** y **-o** serán representado como un *OR* mientras que un caracter de admiración '**!**' niega el patrón (*NOT*) :

```
$ sudo find -name "marrón" -o -name "negro" -a -name "coche" ! -name
"motocicleta"
```

locate y **slocate**: Se utilizan para buscar sólo por el nombre del archivo (devuelve todos los archivos que contienen la cadena especificada). La utilidad *locate* trabaja con una base de datos. *slocate* incluye características de seguridad que impiden que los usuarios vean los nombres de los archivos de los directorios a los que no tienen acceso. En la mayoría de las distribuciones suelen ser un enlace simbólico.

locate [opciones] PATRÓN

Entre las opciones más destacadas, de la más importante es **–regex** que interpreta el patrón como una expresión regular.

updatedb: Se utiliza para actualizar la base de datos de *locate*. Si creamos directorios nuevos con nuevos archivos después de haber actualizado la base de datos y no la volvemos a actualizar, *locate* no encontrará estos archivos

whereis: Localiza archivos binarios, archivos de documentación o configuración.

whereis [opciones] comando

Opciones:

- Buscar unicamente binarios: **-b**
- Buscar solo páginas de manual: **-m**
- Localizar archivos fuentes: **-s**

which: Nos permite conocer la ruta de un determinado comando

type: Si queremos saber como interpreta Linux un determinado comando.

COMANDOS LPIC-1

Capítulo 5

Comandos para interactuar con los cargadores de arranque: GRUB y EFI

grub-install: Instala GRUB en un dispositivo

grub-install [opciones] dispositivo

Opciones:

- Instalar GRUB en una partición independiente. No se especifica cuando la partición por defecto es /boot

```
$ grub-install --boot-directory=/mnt/boot /dev/sdb
```

Nota: Por defecto se utiliza *–boot-directory=/boot*, cuando solo invocamos el comando *grub-install /dev/sda*

- Especificar la ESP de EFI: **–efi-directory=**DIR
- Revisar el mapa de dispositivos '*/boot/grub/device.map*'. Es recomendable hacerlo cuando se añadan o quiten discos: **–recheck**
- Especificar un ID para el cargador de arranque. Solo disponible para *UEFI* y *Macs*: **–bootloader-id=**ID

grub-mkconfig: Genera un archivo de configuración para GRUB

grub-mkconfig [opciones]

Opciones: La opción a usar es **-o** <file> (*–output=FILE*) que crea el archivo donde especifiquemos

$ grub-mkconfig -o /boot/grub/grub.cfg

Nota: *update-grub* ejecuta *grub-mkconfig -o /boot/grub/grub.cfg*. Normalmente es utilizado para crear un archivo de configuración grub2

grub-mkrescue: Crea una imagen de rescate para GRUB

$ grub-mkrescue [opciones]

Opciones: La opción más destacada es **-o** para indicar la ruta de creación

grub-mkimage: Crea una imagen booteable de GRUB

grub-mkimage [opciones]

Opciones:

- Utilizar un archivo de configuración ya existente: **-c** (*–config*=FILE)
- Utilizar una imagen y módulos concretos (por defecto */usr/lib/grub/<plataforma>*): **-d** <directorio>
- Utilizar un archivo específico de salida (por defecto stdout): **-o** FILE
- Utilizar un formato concreto: **-O** <formato> (*–format*=<formato>)

Formatos más comunes:

- i386-coreboot, i386-multiboot, i386-pc, i386-efi, i386-qemu y i386-xen
- x86_64-efi, x86_64-xen

efitbootmgr:Utilizado para interactuar con el administrador de arranque EFI. Durante la lección 4 vimos como añadir una entrada en EFI. Vamos a estudiar estas opciones y algunas otras.

```
# efibootmgr -c -d /dev/sda -p 1 -l /EFI/refind/refind_x64.efi -L
"Fedora"
```

Opciones:

- Crea una nueva variable '*bootnum*' y la añade al orden de arranque: **-c**
- Indicar el disco que contiene el cargador de arranque: **-d**
- Indicar la partición que contiene el cargador de arranque: **-p**
- Especificar el cargador: **-l**
- Dar un nombre a la entrada (etiqueta): **-L**

Otras opciones:

- Setear una entrada de variable como activa: **-a**
- Desactivar una entrada de variable: **-A**
- Modificar el '*bootnum*' de una variable : **-b**
- Ordenar las entradas ('bootnum1'.'bootnum2'…): **-o**
- Eliminar el orden de entradas: **-O**
- Especificar un *timeout* para el menú: **-t**
- Eliminar el tiempo: **-T**

Comandos para la administración de servicios y runlevels SysV

init (*telinit*): Nos permiten cambiar del nivel de ejecución desde la línea de comandos.

init runlevel

runlevel: Nos permite ver el nivel de ejecución en el que estamos actualmente y desde el que vinimos. *Runlevel* devuelve dos números, por ejemplo '**1 3**', esto indica que estamos en el nivel **3** y el anterior (desde el que venimos) es el modo **monousuario**. Si el primero número es una '**N**' significa que no existe un modo de ejecución anterior.

update-rc.d: Instala o elimina scripts SysV mediante links a sus respectivos directorios de modo de ejecución. Algunos ejemplos:

update-rc.d [-n] foo defaults

Activa (genera los enlaces apropiados para el servicio) foo para los runlevels 2345 y los detiene para 016. La opción '**-n**' es opcional, si la pasamos se realizará una simulación pero no se efectuarán cambios. Si los enlaces ya existiesen no se crearan nuevos, esto es así para no machar posibles configuraciones personales del administrador en los scripts.

update-rc.d [-n] foo disable|enable [S|2|3|4|5]

Deshabilita o habilita el servicio foo para los modos de ejecución seleccionado. Opcionalmente podemos elegir entre [S|2|3|4|5]. Este comando realiza automáticamente el cambio de prefijo de los enlaces y además modifica el número. Si se usa **disable**, el número que adopta es 100 menos el número que poseía anteriormente y si se usa **enable** se realiza la opción contraria, es decir, se resta a 100 el número que adopta el servicio cuando está desactivado y la diferencia será el nuevo número de servicio. S20myservice (activado), lo desactivamos (100-20=80) K80myservice, lo volvemos a activar (100-80) S20myservice.

update-rc.d [-n] [-f] foo remove

Se eliminará cualquier links en los directorios */etc/rc?.d* para el script */etc/init.d/foo*. Para que los links sean eliminados correctamente deberemos de eliminar el script principal de init.d, de lo contrario dará error y, si usamos la opción **-f** o los eliminamos manualmente, cuando se actualice de nuevo el servicio, *update-rc.d* será ejecutado y los links creados. Para desactivar los servicios también podemos modificar su nombre en los directorios de cada modo de ejecución y cambiar el prefijo '**S**' por '**K**'

Vamos a ver un par de ejemplos donde veremos otra forma de activar un servicio de modo más específico:

update-rc.d foo start 30 2 3 4 5 . stop 70 0 1 6 .

update-rc.d foobar stop 80 2 3 4 5 .

Crear links interpretando que B depende de A

update-rc.d service_A defaults 80 20
update-rc.d service_B defaults 90 10

chkconfig: Consulta y actualiza los servicios para cada *runlevel*

chkconfig [opciones] servicio

Opciones:

- Listar todos los servicios y su modo de ejecución, o mostrar un solo servicio: **–list** [service]

```
$ sudo chkconfig --list [foobar]
```

- Habilitar/Deshabilitar un servicio para los runlevels 2345:

```
$ sudo chkconfig foobar on|off
```

- Habilitar/Deshabilitar un servicio en unos *runlevels* concretos:

```
$ sudo chkconfig foobar on|off --level 2
```

ntsysv: Herramienta 'gráfica' de línea de comandos, que nos permite habilitar/deshabilitar servicios en el runlevel predeterminado o seleccionar alguno(s) en particular.

Para su uso básico bastaría con escribir 'ntsysv' como root y se mostrará un panel para marcar o desmarcar aquellos servicios que queremos (o no) ejecutar durante el arranque.

Si por el contrario queremos especificar un runlevel concreto:

```
# ntsysv --level 35
```

service: Arranca, detiene, comprobar el estado de un servicio para scripts SysV

service <service> start|stop|restart|status|reload

Comando initctl: Administrando init en Upstart

initctl: Permite al administrador comunicarse e interactuar con el demonio *init* de *Upstart*. Nos permite iniciar, detener, comprobar el estado, etc… de los servicios.

initctl [opciones] comando [opciones comandos] servicio|job

Comandos:

- **start**, **stop**, **restart**, **status**: Inicia, detiene, reinicia y comprueba el estado de un servicio respectivamente.
- **reload**: Recarga el archivo de configuración (*servicio.conf*)
- **list**: Lista que muestra el nombre del servicio, su estado y su PID
- **reload-configuration**: Recarga el archivo de configuración/preferencias de un servicio (*smb.conf*). Ya no es necesario, gracias a *inotify*.
- **show-config**: Muestra el contenido del archivo de configuración del JOB. Podremos ver en que niveles arranca, etc…
- **check-config**: Comprueba la configuración del servicio. Es útil cuando algún servicio no se inicia o detiene con normalidad

Comandos para la gestión de systemd

systemctl: Nos permite controlar el demonio *systemd* y con ello llevar a cabo una eficaz administración de los servicios.

systemctl [opciones] comando [opciones del comando]

Opciones:

- Indicar el tipo de target: **-t** (*–type=*TARGET)
- Filtrar la salida en función del estado: **–state=**LOAD|SUB|ACTIVE
- Filtrar por las propiedades del unidad/servicio/trabajo: **-p** (*–property=*)
- Mostrar todas las unidades/servicios/job cargados: **-a** (*–all*)
- Si filtramos por socket, podemos indicar de que tipo de socket se trata: **–show-types**
- Si vamos a habilitar una unidad, podemos sobreescribir su enlace: **-f**

Comandos:

- Listar las unidades conocidas (opción por defecto): **list-units**
- Listar las unidades instaladas: **list-unit-files**
- Listar sockets: **list-sockets**
- Iniciar, Detener, Recargar (tipo smb.conf), Reiniciar y Estado de los servicios: **start**, **stop**, **reload** y **restar** (respectivamente)
- Inicia la unit indicada (y sus dependencias) y detiene el resto. Esto es parecido a cambiar de runlevel: **isolate**
- Entrar al modo por defecto (parecido a isolate pero sin especificar el target): **default**
- Entrar en modo de rescate: **rescue**
- Entrar en modo de emergencia: **emergency**
- Cerrar la sesión y apagar el sistema: **halt**
- Apagar el sistema (como si quitamos el cable de corriente): **poweroff**
- Reiniciar el sistema: **reboot**
- Entrar en modo suspensión: **suspend**
- Hibernar el equipo: **hibernate**
- Comprobar si una unit se encuentra en ejecución: **is-active**
- Comprobar si una unit está en estado “failed”: **is-failed**
- Mostrar las propiedades de una unit: **show**
- Mostrar las dependencias de una unit (si no especificamos una en concreto se toma default.target): **list-dependencies**
- Habilitar / Deshabilitar una unit: **enable** , **disable**
- Comprobar si una unidad está habilitada: **is-enabled**
- Resetear una unit: **preset**
- Mostrar el runlevel (target) que tenemos por defecto para iniciar el sistema: **get-default**
- Setear el target con el que queremos arrancar el sistema: **set-default**
- Listar los trabajos (Jobs) que están activos: **list-jobs**
- Cancelar un trabajo: **cancel**

- Recargar el archivo de una unidad (*httpd.service*): **daemon-reload**

shutdown, **poweroff**, **halt**, **reboot**: Estos comandos son equivalentes a init 0 e init 6. La mejor forma de apagar el sistema es con *shutdown*. Además este permite una gran variedad de opciones.

shutdown [opcion] argumento [mensaje]

Opciones de *shutdown*:

- Detener el sistema: **-H**
- Apagar el sistema: **-P**
- Apagar el sistema (equivalente a -P a menos que -H haya sido especificada): **-h**
- Reiniciar el sistema: **-r**
- Cancelar una tarea de shutdown pendiente: **-c**
- No ejecutar fsck al reinicio: **-f**
- Forzar fsck al reinicio: **-F**
- Enviar un mensaje automático de advertencia: **-k**
- Apagar sin llamar a init: **-n**
- Indicar el tiempo: **-t**
- No enviar mensajes después de ejecutar la orden shutdown: **–no-wall**

Ejemplos:

Apagar el sistema a las 01:30 AM y además envía un mensaje

```
# shutdown -h 1:30 “A las 1:30 de la madrugada el sistema va a ser apagado”
```

Cancelamos la acción anterior, pues nos hemos equivocado de hora:

```
# shutdown -c “Perdonar el sistema no va a ser apagado, quizás a las 13:30, pero avisaremos. Disculpen”
```

Opciones de *halt* y *reboot*:

- No ejecutar sync antes de invocar a halt o reboot: **-n**
- No escribir en el archivo */var/log/wtmp* (implica usar -n): **-d**
- Desconectar la interfaz de red antes de apagar el sistema: **-i**
- Ejecutar un *poweroff* después del *shutdown*: **-p**
- No llegar a apagar el sistema y además escribe en */var/log/wtmp*: **-w**

COMANDOS LPIC-1

Capítulo 6

Comandos para la configuración del servidor X

xfree86 -configure: Carga todos los drivers de vídeo, explora el hardware y crea el archivo de configuración base con el que podremos iniciar una configuración del servidor X.

xorg -configure: Igual que el anterior pero para sistemás que usan el servidor X.org.

system-config-display: Nos permite configurar las opciones del archivo xorg.conf utilizado para la configuración del sistema X de pantalla. Dispone de varias opciones como:

- Crear un archivo de configuración específico: **-o** <file> (*–output=*)
- Reconfigurar sin basarse en el archivo xorg.conf ya existente: **–reconfig**
- Cambiar el valor para determinadas características: **–set-<keys>=**valor
- Resolución: **–set-resolution=**800×600
- Profundidad de color: **–set-depth=**24
- Drivers de la targeta de video: **–set-driver=**<drivers>

```
 system-config-display --reconfig --set-driver=nvidia --set-
depth=24 --set-resolution=1024x768
```

yast/yast2: Es la herramienta de instalación y configuración de *openSUSE* y *SUSE Linux Enterprise*. Es popular por su facilidad de uso, por su interfaz gráfica atractiva y la capacidad de personalizar su sistema de forma rápida durante y después de la instalación. Se puede utilizar para configurar todo el sistema. La configuración de hardware, configurar la red, los servicios del sistema y ajustar la configuración de seguridad. Para iniciar el menú de configuración *yast* o *yast2* bastará con ejecutar desde la línea de comandos (o desde GUI) el ejecutable **/sbin/yast** o **/sbin/yast2**

xf86config/**xconfigurator**/**xf86setup**: Archivo de configuración y herramientas de configuración del servidor X y ajustes de su archivo de configuración (respectivamente) para el servidor XFree86 versiones 3.3.x o anteriores.

xdpyinfo: Nos permite obtener información sobre la configuración del servidor X. Sin opciones xdpyinfo enumera todas las extensiones disponibles.

Nota: Una extensión X es un módulo de software que proporciona capacidades ampliadas a X

Para obtener información más detallada sobre una ventana específica podremos usar el comando **xwininfo**. Su uso básico consiste en escribir *xwininfo* y mover el cursos del ratón sobre una ventana y hacer clic. Podremos pasar algunas opciones a *xwininfo* como **-id** <id> o **-name** <nombre> para especificar una ventana por su id o nombre respectivamente. Podremos identificar procesos hijos relacionados con las ventanas con las opciones **-children** o –**tree**. Mostrar información básica (predeterminada) con –**stats** o información adicional con: **-events** (eventos procesados por la ventana), **-size** (sugerencias de tamaños), **-wm** (datos sobre el administrador de ventanas), **-shape** (similar a *-stats* pero amplia su información con respecto a la forma de la ventana y su borde), **-frame** (que modifica la pantalla para incluir información en el marco del administrador de ventanas) y **-all** que nos mostrará toda la información disponible

lightdm: Administrador del display

lightdm [opciones]

Opciones:

- Usar un archivo de configuración alternativo: **-c** <file>
- Usar con un usuario sin privilegios (ignorará tareas que necesiten acceso de root): **–test-mode**
- Seleccionar el directorio desde donde cargar las Xsession: **–xsessions-dir=**<directorio>
- Seleccionar el directorio desde donde cargar las configuración de greeters: **–xgreeters-dir=**<directorio>
- Escribir el log de salida en un archivo adicional: **–log-dir=**<directorio>

dm-tool: Herramientas para controlar el administrador del display

dm-tool [opciones] comando [argumentos-comando]

Comandos:

- Cambiar de una sesión: **switch-to-user <usuario> [sesión]**
- Cambiar a sesión de invitados: **switch-to-guest [sesión]**
- Bloquear la sesión actual: **lock**
- Listar las sesiones activas: **list-seats**

Comandos para crear archivos índices de fuentes

mkfontscale: Crea un archivo índice para fuentes escalables. Leerá los archivos de fuentes de un cierto directorio y creará el archivo índice correspondiente a todas las fuentes encontradas.

mkfontscale [opciones] directorio

Opciones:

- Leer las fuentes de mapa de bits (por defecto ignorado): **-b**
- Ignora las fuentes escalables (por defecto son las que se leen): **-s**
- Escribe la salida en un archivo distinto al predeterminado (fonts.scale): **-o** <file>

mkfontdir: Crea un archivo índice de fuentes X de un determinado directorio.

mkfontdir [opciones] directorio

Comandos para la configuración local y zona horaria

tzselect/tzconfig/tzsetup: Ver las zonas horarias y poder generar el valor de la variable *TZ* y el nombre de dicha zona horaria para el archivo */etc/timezone*

locale: Muestra información sobre la configuración regional local o todas las que tenemos disponibles

date: Muestra o configura la hora y fecha del sistema

date [opciones] [formato]

Opciones:

- Imprimir la fecha de última modificación de un archivo: **-r** <file>
- Mostrar la hora en formato RFC 2822: **-R**

```
$ date
```

lun feb 2 19:12:56 CET 2015

```
$date -R
```

Mon, 02 Feb 2015 19:12:58 +0100

- Mostrar la hora UTC: **-u**
- Configurar la hora: **-s** <formato>

Algunos Formatos:

- Mostrar el día de forma abreviado o completo (Lun o Lunes), respectivamente: **%a** | **%A**
- Mostrar el mes abreviado o completo: **%b** | **%B**
- Día de la semana (1-7, 1 = Lunes): **%u**
- Día de la semana (0-6, 0 = Lunes): **%w**
- Día del mes: **%d**
- Día del año: **%j**
- Mes del año: **%m**
- Mostrar fecha en formato *%m/%d/%y* : **%D**
- Mostrar hora completa en formato 12h: **%r**
- Hora en formato 24h: **%H**
- Hora en formato 12h: **%I**
- Minutos: **%M**
- Acompañar la hora de AM o PM: **%p**

- Igual que el anterior pero en minúsculas: **%P**
- Representar la fecha de forma local (01/27/2015): **%x**
- Representar la hora de forma local (22:31:43): **%X**
- Representar la fecha y hora local (Martes Feb 3 12:25:44 2014): **%c**

Ejemplos para setear la fecha y hora, o ver estás de un determinado modo:

- Configurar la fecha y hora en formato corto:

```
$ sudo date -s "2015-01-28 13:49"
```

- Configurar la fecha y hora en formato extendido:

```
$sudo date -s "Mon, 07 Aug 2014 12:34:56"
```

- Ver la fecha de forma estándar (lun ago 7 20:49:12 CEST 2006):

```
$ date
```

- Ver la fecha sin abreviaturas (Lunes Agosto 07 20:52:10 CEST 2015:

```
$ date "+%A %B %d %X %Z %Y"
```

- Ver la fecha de la siguiente manera: nombre-del-día, xx/xx/xx hh/mm/ss:

```
$ date "+%A, %x %X"
```

hwclock: Muestra o configura el reloj del hardware

Ejemplos:

- Mostrar el reloj del hardware:

```
$ sudo hwclock --show
```

- Configurar el reloj de hardware basándose en la hora del reloj de software:

```
$ sudo hwclock --systohc
```

- Configurar el reloj de software basándose en el reloj de hardware:

```
$sudo hwclock --hctosys
```

- Configurar la hora del reloj de hardware en formato corto:

```
$ sudo hwclock --set --date="2/28/2015 22:35:18"
```

- Configurar la hora del reloj de hardware en formato extenso:

```
$ sudo hwclock --set --date="`date '+%D, %Y %H:%M:%S'`"
```

Nota: Nos sirven de igual modo los formatos del comando **date**

- Indicar al reloj que será hora UTC (útil en relojes de hardware): **–utc**

timedatectl: Controlar la fecha y hora del sistema

timedatectl [opciones] comando

Opciones:

- Ejecutar el comando en un host remoto: **-H** <*username@host*>
- Pasar la siguiente opción junto con el comando "*set-local-rtc*" provocará la sincronización del reloj del sistema con *RTC*. De lo contrario *RTC* es sincronizado con el reloj del sistema: **–adjust-system-clock**

Comandos:

- Mostrar la configuración de tiempo del sistema actual (opción por defecto): **status**
- Configurar la hora del sistema (esto ajustará también *RTC*): **set-time** *"2015-01-18 22:34:52"*
- Listar todos los *timezones* disponibles: **list-timezones**
- Setear un *timezone* concreto: **set-timezone** <*timezone*>
- Ajustar *RTC* en función de *UT* – Universal Time (0) – o localtime (1). Se recomienda en *UTC*: **set-local-rtc** <0 | 1>
- Controlar la configuración de hora a través de la red (habilita (1) o deshabilita (0) *NTP* siempre y cuando este disponible el servicio): **set-ntp** <0 | 1>

ntpq: Realizar consultas a *ntp*

ntpq [opciones] [host]

Opciones:

- Forzar a *ntpq* a ejecutarse en modo interactivo: **-i**
- Pasar comandos de *ntpq* directamente desde la línea de comandos: **-c** <*comandos*>
- Listar los host en formato numérico en vez de sus nombres: **-n**
- Mostrar los '*peers*' conocidos, es decir los servidores con los que sincroniza *ntp*: **-p**

Comandos para la administración de impresoras

lp / **lpr** : Ambos comandos envían trabajos a la impresora. Comparten sintaxis y algunas opciones.

Opciones comunes:

- Especificar el usuario que hará uso del servidor de impresión: **-U** <usuario>
- Enviar un mail a la finalización del trabajo: **-m**
- Especificar un servidor de impresión alternativo: **-h** hostname[:port] / **-H** server[:port] (lp y lpr respectivamente)

Opciones **lp**:

- Imprime los archivos en la impresora especificada: **-d** <impresora>
- Especificar un existente trabajo a modificar: **-i** <job-id>
- Indicar el número de copias a imprimir (de 1 a 100): **-n** <copias>
- Indicar la prioridad de un trabajo (de 1 a 500, por defecto 50): **-q** <prioridad>
- Setear el nombre de un trabajo: **-t** <name>

Opciones **lpr**:

- Imprime los archivos en la impresora especificada: **-P** <impresora>
- Indicar el número de copias a imprimir (de 1 a 100): **-#** <copias>
- No imprimir la cabecera del archivo: **-h**
- Especificar que el archivo a imprimir ya está formateado, no necesita ser procesado por filtros: **-l**
- Eliminar los archivos una vez imprimidos: **-r**

lpc: Proporciona un control limitado sobre impresoras y colas de impresión suministradas por CUPS

lpq: Informa sobre el estado de la cola de impresión

cancel: Cancela trabajos de la cola de impresión. Podemos pasar la opción -a para eliminar todos los trabajos de una cola de impresión dada, o de todas las colas si no especificamos una en concreto.

lprm: Igual al anterior. Elimina trabajos de la cola de impresión.

lpmove: Mueve trabajos a un nuevo destino

```
lpmove job-id newprinter
```

lpoptions: Muestra o establece las opciones para una impresora.

lpstat: Muestra el estado de la impresora, la cola de impresión y trabajos.

lpstat [opciones]

Opciones:

- Mostrar el nombre del servidor de impresión y el puerto: **-H**
- Mostrar un trabajo determinado, por defecto mostrará si está o no completado. Hay que indicar esta opción antes de -o y/o del nombre de la impresora: **-W** <job>
- Mostrar la impresora de destino por defecto: **-d**
- Mostrar una larga lista de impresoras y trabajos: **-l**
- Mostrar la cola de impresión para un destino concreto: **-o** <destino>
- Mostrar si el servidor CUPS está en ejecución: **-r**

lpadmin: Nos permite definir la impresora por defecto y configurarla. Posee bastantes opciones. Algunas obvias podrían ser:

- Eliminar la impresora por defecto: **-R** <nombreimpresora>
- Imprimir información sobre la impresora: **-D**
- Habilitar la impresora destino y aceptar colas de impresión: **-E**
- Informar sobre la localización de la impresora destino: **-L**

system-config-printer: Configura un servidor de impresión CUPS

cupsaccept/**cupsreject**: Aceptar o rechazar trabajos enviados a una impresora.

cupsenable/**cupsdisable**: Iniciar o detener impresoras CUPS.

COMANDOS LPIC-1

Capítulo 7

Comandos para crear y eliminar usuarios y grupos

Herramientas nativas de bajo nivel para crear y eliminar usuarios y grupos

useradd: Crea un nuevo usuario, muestra las opciones en las que *useradd* se basa para crear un nuevo usuario o actualiza estas.

useradd [opciones] login

useradd -D

useradd -D [opciones]

Opciones:

- Para listar las opciones o valores de variables utilizadas para crear una cuenta usamos: **-D**
- Por defecto al crear un usuario se concatena el nombre del nuevo usuario al directorio base /home. Si queremos elegir otro directorio principal para el usuario lo haremos con: **-d** <*nuevodirectorio*>
- Cuando usamos la opción *-d*, necesitaremos acompañarla de la opción que realizará la creación del directorio: **-m**
- El directorio base para crear el home de un usuario es /home. Esto podemos cambiarlo con *useradd -D [opciones]*. Aún así se queremos especificar otro directorio base al que concatenar el nombre del usuario podremos hacerlo con: **-b** <*basedir*>
- Podemos añadir información extra a una cuenta, como números de teléfono, nombre completo, etc… con: **-c** <*comentarios*>
- Por defecto el *UID* y *GID* que se generan al crear una cuenta dependen del último *UID* y *GID* del archivo */etc/passwd*. Se sumará +1 a estos *ID*. Si queremos utilizar otros *ID* podremos especificarlos con: **-u**, **–uid** <*nuevo-id*> y **-g**, **–gid** <*nuevo-id*> (respectivamente)
- Podemos hacer que una cuenta sea deshabilitada un día concreto con: **-e**, **–expirate** <*yyyy-mm-dd*>
- Podemos hacer que una cuenta esté inactiva durante un “periodo de gracia” después de que una contraseña haya expirado. De esta manera el usuario contará con unos días para volver a cambiar la password antes de que la cuenta sea totalmente

inhabilitada. Un valor de 0 indica que la cuenta estará inactiva tan pronto como su contraseña expire. Un valor de -1 desactiva esta funcionalidad: **-f**, **–inactive** *<valor>*
- Si el usuario que estamos creando va a compartir *UID* con otros usuarios tendremos que pasar la opción: **-o**
- Podemos especificar un *shell* para el nuevo usuario: **-s**, **–shell** *<shell>*
- Crear un grupo con el mismo nombre que el usuario (opción por defecto): **-U**
- Cuando creamos un usuario se crea igualmente un grupo con el mismo nombre que además es el grupo principal. Si queremos añadir el usuario a grupos secundarios en el momento de ser creado: **-G**, **–groups** *<grupo1,grupo2…grupoN>*
- Crear un usuario del sistema, por lo general utiliza *ID* por debajo del 100: **-r**, **–system**
- Pasar la contraseña de la cuenta desde la línea de comandos: **-p** *<password>*
- Indicar el directorio esqueleto que contiene los archivos de configuración de la cuenta (por defecto */etc/skel*): **-k**, **–skel** *<directorio>*

userdel: Eliminar una cuenta de usuario y sus archivos relacionados

userdel [opciones] login

Opciones:

- Forzar la eliminación de una cuneta de usuario, directorio home y mail incluso estando logueado el usuario: **-f**
- Eliminar el home y cola de correo del usuario. El resto de archivos fuera de estos directorios deberán de ser eliminados manualmente: **-r**

groupadd: Crea un nuevo grupo

groupadd [opciones] grupo

Opciones: Comparte las opciones **-g**, **–gid** , **-o**, **–non-unique** , **-p** y **-r**, **–system** del comando *useradd*.

groupdel: Elimina un grupo

groupdel [opciones] grupo

Herramientas opcionales para algunas distribuciones de Linux

Nota: Son *front ends* de las anteriores. Por lo general su uso es más sencillo que las descrita anteriormente ya que vienen preconfiguradas con opciones adicionales que permiten por ejemplo crear un usuario, su directorio home y su shell por defecto con tan solo escribir *# adduser <usuario>*. Pero esto no funciona de la misma manera en todas

las distribuciones por lo que se aconseja utilizar las descritas anteriormente. El comportamiento de estas herramientas se establece a partir de los archivos */etc/adduser.conf* y */etc/deluser.conf*. Además puede existir el archivo */usr/local/sbin/adduser.local* cuyo contenido será ejecutado tras la creación del usuario estableciendo características locales adicionales.

adduser: Crea un usuario o añade un usuario a un grupo.

- Crear un usuario con *UID* y *GID* por defecto, directorio home (*/home/<usuario)*>), su grupo principal con el mismo nombre que el usuario, un *shell* predeterminado, directorio *mail* y archivos de configuración del directorio esqueleto:

```
$ sudo adduser <usuario>
```

- Crear un usuario de sistema (los *IDs* suelen oscilar entre 0-99), no son añadidos a su grupo principal y el *shell* por defecto es */bin/false*

```
$ sudo adduser --system <usuario>
```

Nota: Podemos cambiar las opciones por defecto al igual que con el comando *useradd*.

- Añadir un usuario a un grupo:

```
$ sudo adduser <usuario> <grupo>
```

addgroup: Crear un grupo. Su uso es como el de *adduser*, es decir bastará con pasar como argumento el nombre del grupo o el parámetro –system para crear un grupo de sistema.

deluser: Elimina un usuario. Su uso es básico, pasar como argumento el nombre del usuario a eliminar. Por defecto no se elimina su directorio home, ni su cola de mail ni sus archivos creados fuera de su directorio principal. Pero tenemos varias opciones interesantes.

Opciones:

- Si queremos eliminar además del usuario, su directorio principal (con sus archivos) y su cola de correo: **–remove-home**
- Si además de lo anterior también queremos eliminar los archivos creados fuera de su directorio home: **–remove-all-files**
- Podemos crear un *backup* de sus archivos antes de ser eliminado: **–backup** por defecto se almacena comprimido en su directorio principal pero podemos indicar otro con **–backup-to** *<nuevo directorio>*
- Eliminar un usuario de un grupo:

```
$ sudo deluser <usuario> <grupo>
```

- Para eliminar un grupo con *deluser* usamos la opción: **–group**

delgroup: Elimina un grupo. Podemos pasar la opción **–only-if-empty** para eliminar un grupo que no contiene usuarios, es decir de haber usuarios en el grupo nos avisará y no será eliminado.

Nota: Los dos comandos anteriores tienen definido su comportamiento en */etc/deluser.conf*

Comandos para modificar cuentas de usuarios y grupos

usermod: Modifica cuentas de usuario

usermod [opciones] login

Opciones: En realidad utiliza las mismás opciones que cuando un usuario es creado con *useradd*: **-c**, **-d**, **-u**, **-g**, **-o**, **-f**, **-e** y **-p**. Aunque existen otras o algunas que cambian como:

- Si acompañamos la opción *-d* con **-m** en vez de ordenar crear el directorio lo que hará será mover todo el contenido del antiguo directorio al nuevo directorio indicado con *-d*.
- Cambiar el nombre de la cuenta del usuario: **-l**, **–login** *<nuevo nombre>*
- Bloquear o desbloquear una cuenta (en realidad añade o elimina el caracter '**!**' del campo de contraseña): **-L** y **-U** respectivamente
- Podemos añadir usuarios a un grupo si acompañamos la opción *-G <grupo1,grupo2…>* con **-a**, **–append**. Para añadir un usuario a un grupo de esta manera, deberemos de nombrar además de los nuevos grupos todos aquellos a los que el usuario ya pertenecía, de lo contrario el usuario abandonará el grupo no indicado. Esto puede servir para eliminar al usuario de diferentes grupos.

groupmod: Modificar la configuración de un grupo.

groupmod [opciones] grupo

Opciones:

- Modificar el *GID* de un grupo: **-g**, **–gid** *<id>*
- Si el nuevo *GID* no es único usar: **-o**, **–non-unique**
- Darle un nuevo nombre al grupo: **-n**, **–new-name** *<nombre>*
- Crear una contraseña de grupo: **-p** *<password>*

chsh: Cambiar el *shell* de un usuario. Podemos emplear este comando en lugar de *usermod -s <nueva shell>*

chfn: Podemos modificar la información de usuario con *chfn* y sus opciones avanzadas, mejor que con *usermod -c <comentarios>*

chfn [opciones] login

Opciones:

- Cambiar el nombre completo del usuario: **-f**, **–full-name** *<Nombre Apellidos>*

- Crear un número de teléfono: **-h**, **–home-phone** *<número>* o **-w**, **–work-phone** *<número>*
- Añadir un número de habitación: **-r**, **–room** *<número>*
- Añadir información extra: **-o**, **–other** *<info>*

passwd: Cambia la contraseña de un usuario

passwd [opciones] login

Opciones:

- Bloquear o Desbloquear la contraseña de un usuario: **-l**, **–lock** o **-u**, **–unlock**
- Eliminar la contraseña para una cuenta: **-d**, **–delete**
- Podemos forzar a que un usuario cambie su contraseña, haciéndola que expire inmediatamente: **-e**, **–expire**
- Dejar una cuenta inactiva si el usuario no ha modificado una contraseña que ha expirado en un determinado plazo de días: **-i**, **–inactive** *<días>*
- Permitir el cambio de una contraseña solo si esta a caducado: **-k**
- Indicar el número mínimo de días entre cambios de contraseña. Un 0 indica que podemos cambiar la contraseña varias veces en un día: **-n** *<días>*
- Indica el número máximo de días entre cambios de contraseña. Una vez sobrepasado estos días la contraseña deberá ser cambiada: **-x** *<días>*
- Avisar con un determinado número de días sobre el cambio de la contraseña, antes de que esta esté bajo la obligación de ser modificada: **-w** *<días>*
- Muestra el estado de una contraseña: **-S**, *–status*
- Si acompañamos *-S* de **-a**, **–all** veremos el estado de las contraseñas de todas las cuentas.

chage: Con este comando podremos modificar los valores dados a una contraseña con el comando *passwd*, o crear unos nuevos

chage [opciones] login

Opciones:

- Mostrar información sobre la contraseña de un usuario: **-l**, **–list**
- Normalmente con el comando *chage* podremos indicar una fecha de dos formás diferentes, bien con el número de días transcurridos desde el *1 de Enero de 1970* hasta el día en el que queramos dejar constancia de algo (último día en el que se cambió la password, último día para usar una cuenta sin antes modificar su contraseña, etc… o bien con el formato '*YYYY-MM-DD*'. Entonces si queremos setear el último día en el que la contraseña fue modificada: **-d**, **–lastday** *<fecha>*
- Indicar el último día en el que una cuenta será accesible es decir, será bloqueada de forma definitiva y solo el administrador podrá restablecerla: **-E**, **–expiredate** *<fecha>*
- Indicar el número de días en los que una cuenta estará inactiva antes de ser bloqueada: **-I**, **–inactive** *<fecha>*
- Indicar el número mínimo de días entre cambios de contraseñas: **-m**, **–mindays** *<días>*

- Indicar el número máximo de días que pueden transcurrir sin cambiar una contraseña: **-M**, **–maxdays** <*días*>
- Avisar con unos días de antelación al usuario de que su contraseña debe de ser modificada: **-W**, **–warndays** <*días*>

Nota: En muchas de las opciones si utilizamos los valores -1 estaremos indicando la inhabilitación de esa función.

chpasswd: Cambiar las contraseñas de varias cuentas de una sola vez. Para invocar a *chpasswd* basta con nombrarlo y acompañarlo con una lista de nombre_usuario:contraseña (una cuenta por cada línea). Podremos crear un archivo con varias lineas y pasarselo mediante la redirección < archivo. Para encriptar las contraseñas *chpasswd* utiliza *PAM*. Este comando acepta varias opciones pero que su uso no está recomendado como:

- Indicar un método distinto de encriptación (por defecto SHA512): -c <método>
- Pasar contraseñas ya encriptadas: -e
- Usar el método md5 para encriptar las contraseñas: -m, –md5

gpasswd: Administrar los archivos */etc/group* y */etc/gshadow*. Crear administradores de grupo, añadir usuarios a grupos y crear una password para un grupo.

gpasswd [opciones] grupo

Opciones:

- Un grupo puede tener uno o varios usuarios administradores, para definir uno o una lista de ellos usamos: **-A** *user1,user2…userN*
- De igual forma podemos añadir usuarios a un grupo en forma de lista: **-M** *user1,user2…userN*
- Por defecto cualquier usuario perteneciente a un grupo puede hacer que este sea su grupo principal temporal mediante el comando *newgrp*, no así, usuarios que no pertenecen al grupo. Si añadimos una contraseña al grupo los usuarios que ya pertenecían al grupo pueden seguir usando *newgrp*, pero añadimos la posibilidad de que un usuario NO perteneciente al grupo, pueda utilizar *newgrp* para hacer de este su grupo principal temporal siempre y cuando conozca la contraseña del grupo. Para activar una contraseña en un grupo bastará con pasarle al comando *gpaswd* el nombre del grupo y se nos pedirá que introduzcamos la nueva contraseña. Deberemos de ser *root* o administradores del grupo.
- Si queremos hacer que los usuarios que NO pertenecen a un grupo no puedan asignarse este como grupo principal temporal aun conociendo la contraseña, es decir solo los miembros del grupo podrán usar *newgrp*: **-R**, **–restrict**
- Para eliminar la password de un grupo: **-r**, **–remove-password**
- Añadir un usuario a un grupo: **-a**, **–add** <*usuario*> <*grupo*>
- Eliminar un usuario de un grupo: **-d**, **–delete** <*usuario*> <*grupo*>

pwconv y **grpconv**: Implementar contraseñas *shadow* y *gshadow* respectivamente en un sistema. Esto es, las contraseñas de */etc/passwd* y */etc/group* pasan a los archivos */etc/shadow* y */etc/gshadow* de manera que solo *root* pueda leer las contraseñas encriptadas de los usuarios.

pwuncov y **grpunconv**: Realizan la acción inversa de los 2 comandos anteriores respectivamente..

Nota: El comportamiento de estos comandos es determinado a partir de los valores de las variables del archivo */etc/login.defs*

newgrp: Con este comando podemos hacer que un usuario cambie su grupo principal de forma temporal. Tiene limitaciones según la configuración del grupo. Por regla general el usuario que usa este comando tiene que pertenecer previamente al grupo en cuestión. Si el grupo tiene contraseña, un usuario que no pertenezca de antemano al grupo podrá hacer de este grupo su grupo principal temporal siempre y cuando conozca la contraseña. Si el grupo tiene password pero además tiene marcada la opción -R (*restrict*) el usuario deberá pertenecer si o si al grupo previamente, aun conociendo la password de este.

Para cambiar de grupo bastará con pasarle de argumento a *newgrp* el nombre del grupo. Podremos volver a nuestro grupo anterior invocando sin parámetro alguno a *newgrp* o, opcionalmente pasarle un guión '**–**' para simular un nuevo inicio de sesión para el usuario.

smbpasswd: Este comando es utilizado cuando tenemos un servidor *samba* el cual incluye a usuarios y contraseñas en su base de datos. Con este comando y el parámetro **-x** podremos eliminar la cuenta de un usuario de *samba* si no queremos hacerlo de forma manual a través de su archivo */etc/samba/smbpasswd*.

Nota: El servicio *samba* no entra dentro del contenido del examen *LPIC 1* pero estaría bien conocerlo.

Comandos para modificar el nombre de host o dominio

hostname: Sirve tanto para cambiar el nombre de host de forma temporal, es decir al reiniciar volveremos al nombre anterior o, mostrar el nombre actual del equipo.

- Mostrar el nombre del equipo:

```
$ hostname
```

- Modificar el nombre del equipo de forma temporal:

```
$ hostname nuevonombre
```

Nota: Para modificar el nombre de forma permanente podemos recurrir al Capítulo 8 donde se explica como hacerlo.

domainname/**nisdomainname**: Útil si buscamos modificar o mostrar el nombre del dominio del ordenador utilizado por **NIS**

dnsdomainname: Con este comando podremos definir el nombre del dominio utilizado por *DNS*

Comandos para el diagnóstico de la red

ping/**ping6**: Verificar la conectividad con un host remoto

ping [opciones] host

Opciones

- Limitar el número de envíos de paquetes ICMP: **-c** *<num>*
- Indicar un intervalo de tiempo entre el envío de cada paquete *ICMP* al host remoto: **-i** *<tiempo>*
- Hacer un envío másivo de ping. Envía los paquetes con la misma velocidad que los recibe o cien veces por segundo: **-f**
- Muestra solo salidas numéricas: **-n**
- Intenta enviar tantos paquetes tan rápido como le sea posible antes de volver a su comportamiento normal: **-l**
- Especifica el número de bytes que se van a enviar. La cantidad por defecto es *56 + 8 de la cabecera ICMP:* ***-s***

traceroute/**traceroute6**: Envía una serie de tres paquetes de prueba a cada ordenador entre nuestro sistema y el host remoto especificado, mostrando todos los *routers* que existen de por medio y los tiempos de respuesta.

traceroute [-6] [opciones] destino

Opciones:

- Mostrar solo salidas numéricas: **-n**
- Especificar por cual interfaz de red queremos enviar los paquetes: **-i** *<interfaz>*
- Especificar el número máximo de saltos: **-m** *<saltos>*
- Indicar una *gateway* alternativa: **-g** *<gateway>*
- Indicar el tiempo límite de espera para una respuesta: **-w**

tracepath/**tracepath6**: Una alternativa a *traceroute*. Genera una línea de salida por cada paquete de prueba, por lo que su salida es más extensa.

tracepath [opciones] destino

Opciones:

- Mostrar solo salidas numéricas: **-n**
- Mostrar tanto la dirección *IP* como el nombre de host: **-b**
- Especificar el número máximo de saltos: **-m** *<saltos>*
- Indicar el puerto inicial de destino. Esto también es posible hacerlo con *traceroute*. El puerto de destino se va incrementando en uno a medida que se envían paquetes de pruebas: **-p** *<puerto>*

netcat (**nc**): Es una de las herramientas de diagnostico y seguridad más populares dentro de la comunidad *GNU/Linux*. Trabaja tanto con conexiones *TCP* como *UDP* y permite hacer cosas como por ejemplo abrir puertos en host remoto quedándose a la escucha de posibles conexiones, rastrear puertos abiertos o realizar transferencias de archivos bit a bit entre dos equipos. Funciona tanto para *IPv4* como para *IPv6*.

- Para dejar a *nc* como servidor a la escucha (*-l, listen*) en un determinado puerto. *nc* aceptará la conexión de un único cliente y se cerrará:

```
# nc -l <unpuerto>
```

- Para abrir una conexión como cliente del servidor del ejemplo anterior:

```
# nc <equiposervidor> <unpuerto>
```

- Abrir una conexión en el servidor para recibir datos desde un cliente:

```
# nc -l <unpuerto> > <archivo-donde-se-vuelva-el-contenido>
```

- Enviar desde el cliente un archivo para que sea volcado en la conexión del ejemplo anterior:

```
# nc <equiposervidor> <unpuerto> < <archivo-que-será-volcado-en-el-
servidor>
```

- Revisar que puertos *TCP* (opción por defecto) están abiertos dentro de un ranto de puertos determinado (para nuestro ejemplo usaremos el rango 10-50) en un sistema:

```
# nc -vz <unequipo> 10-50
```

Netcat acepta algunas opciones entre las que destacamos (a parte de **-l**, *listen* y **-vz**) **-p** para especificar un puerto, **-k** para cambiar el comportamiento de escucha del primero de los ejemplos, es decir aceptar infinitas conexiones y evitar que la conexión se cierre, cambiar el protocolo de puerto por defecto *TCP* a *UDP* con **-u**, especificar un retraso (*delay*) en el envío o recepción de mensajes **-i**.

netstat: Puede considerarse la herramienta por excelencia para el diagnóstico de red. Dependiendo de los parámetros que le pasemos podrá utilizarse en lugar de muchas otras. Un uso de *netstat* sin opciones puede mostrarnos gran información, como por ejemplo los puertos que tenemos abiertos con conexiones establecidas o cerrados y además los hosts y servicios que se conectan a ellos.

netstat [opciones] host

Opciones:

- Recibir información sobre nuestras interfaces de red (similar a *ifconfig*): **-i**, *–interface*.

- Para obtener un listado de la tabla de enrutamiento (similar a route): **-r**, *–route*.
- Informar sobre las conexiones en las que intervenga las funciones de *NAT:* **-N**, *–másquerade*.
- Podremos usar la siguiente línea para indicar valores específicos:

```
# netstat -punta <host>
```

Donde:

-p muestra el *PID* y el servicio que hace uso del puerto
-u informa de los puertos *UDP* (además de los *TCP*, opción por defecto)
-n expresa los host y puertos mediante números (útil si queremos redireccionar la salida con una pipe para buscar un puerto específico con *grep*),
-t muestra el estado de la conexión (*LISTEN*, *ESTABLISHED*, *CLOSE*, *CLOSE_WAIT*, etc..)
-a muestra todas las conexiones y puertos de escucha.
Si queremos encontrar directamente servidores que escuchan conexiones podemos agregar **-l** a *-p*

lsof: Identifica que archivos están abiertos en el sistema: archivos regulares, directorios, tuberías, librerías o socket de red entre otros, quién accede a ellos, nombre e *ID* del proceso, la ruta absoluta del archivo, su descriptor, etc… útil por ejemplo si queremos desmontar un sistemás de archivos (incluso los de red) pero existen conexiones abiertas a determinados archivos y queremos tener información al respecto.

lsof [opciones] archivo

Opciones

- Listar los ficheros abiertos por un proceso: **-c** *<proceso>*
- Listar los procesos que están usando un directorio: **+D** *<directorio>*
- Mostrar todos los ficheros de red utilizados por procesos de red: **-i**

```
lsof -i :21
lsof -i :ftp
```

- Informar sobre todos los hilos TCP con su correspondientes programás ejecutados: **-i tcp**

Truco: Averiguar cuantos ficheros tiene abiertos cada proceso:

```
lsof | awk '{ print $1 }' | sort | uniq -c | sort -n
```

Con **awk** cortamos la primera columna (la de los servicios), la ordenamos con **sort** para tenerlos todos agrupados. Con **uniq -c** añadimos el prefijo a la línea con el número de ocurrencias y por último ordenamos por número de ocurrencias. Podemos terminar con |*grep <servicio o programa>* para concretar:

```
lsof | awk '{ print $1 }' | sort | uniq -c | sort -n |grep smbd
```

Nota: Este comando también se utiliza para la detección de intrusión.

nmap: El utilitario *nmap* es un escaner de puertos abiertos tanto del ordenador local como de un ordenador remoto. El comando *nmap* trabaja por defecto con paquetes **SYN**, paquetes que intentan abrir conexiones bajo *TCP*. Cuando escaneamos puertos con nmap podemos encontrarnos con tres estados diferentes del puerto: OPEN (*abierto*), CLOSE (*cerrado*) o FILTERED (*puerto no accesible, un cortafuego lo está filtrando*).

nmap [opciones] host

Opciones:

- Evitar conversiones *DNS:* **-n**
- Realizar un escáner que no deja registro en el sistema: **-sS**
- Realizar un barrido de puertos *TCP:* **-sT**
- Provocar un barrido de puertos *UDP* útil si necesitamos conocer puertos de nivel superior que pueden estar tras un firewall: **-sU**
- Usar paquetes *ACK* para lograr que el sistema responda (muchos firewall no filtran estos mensajes, por lo que podemos detectar puertos abiertos más sutilmente): **-sA**
- Intentar pasar un *firewall* mal configurado y permitirnos ver que servicios se están ejecutando en el sistema: **-sN** o **-sF**
- Intenta identificar los servicios en relación con los puertos abiertos (puede detectar vulnerabilidades): **-sV**

tcpdump: Vuelca el tráfico de la red

tcpdump puede interceptar paquetes de red para registrarlos o mostrarlos por pantalla o, incluso almacenar los resultados en un archivo para su posterior estudio, siempre que empleemos la opción **-w**.

Opciones:

- Si queremos mostrar el contenido de los paquetes en código *ASCII*: **-A**
- Para mostrar una lista de interfaces en las que puede escuchar *tcpdump*: **-D**
- Mostrar solo salidas numéricas: **-n**
- Mostrar información ampliada: **-vv** y **-vvv**

COMANDOS LPIC-1

Capítulo 9

Comandos para configurar y administrar el correo electrónico

postconf: Utilidad para configurar el servidor de correos *postfix*. Nos permite modificar y consultar los parámetros del archivo de configuración *main.cf* u otros archivos. Por defecto (sin opción alguna) usando como parámetro una directiva del archivo de configuración *main.cf* nos devolverá su valor. Existen numerosas opciones y posibilidades para este comando, no obstante veremos algunas de las más usuales.

- **SASL**: Para conocer los *plugins* disponibles en nuestra instalación *postfix* relacionados con la capa simple de seguridad tanto a nivel de servidor como para el cliente, usaremos las opciones **-a** y **-A** respectivamente.
- Para mostrar todos los valores por defecto de los parámetros de configuración usamos **-d** o **-df**.
- Podemos editar desde la línea de comandos los parámetros de *main.cf* de la siguiente manera:

```
$ sudo postconf -e parametro="valor"
```

- Para listar todos los parámetros que aparecen configurados en *main.cf* usamos la opción **-n**
- A partir de *postfix* 2.10 podemos sobreescribir el valor de un parámetro con la opción **-o** *parametro=valor*
- Para mostrar la configuración que tenemos en *postfix* (parámetros que vienen por defecto y los que hemos modificado en el archivo *main.cf*) podemos ejecutar *postconf* sin opción alguna o bien con **-p**
- Si lo que queremos es modificar entradas del archivo *máster.cf* podemos hacerlo con **-M** para reemplazar una o más entradas de servicios (*servicio/tipo=valor*) o bien listar el contenido de *máster.cf* si lo pasamos sin más parámetros, **-F** para modificar los valores de un cierto campo de un determinado tipo de servicio (servicio/tipo/campo=valor), también si lo pasamos sin parámetros nos mostrará todos los "*servicios/tipo/campo=valor*" disponibles para *máster.cf*, y **–P** para añadir o actualizar uno o más parámetros de configuración de servicios (*servicio/tipo/parametro=valor*).

Nota: Para modificar *máster.cf* se recomienda antes poseer buenos conocimientos sobre este archivo tan importante para la configuración del servicio.

newaliases : Con este comando podremos crear la base de datos */etc/newaliases.db* una vez que hemos modificado el archivo de alias de correo. Es imprescindible hacerlo para que los cambios en */etc/aliases* tengan efectos.

mail : El comando *mail* nos permite enviar un correo desde la línea de comandos o bien leer el correo de la bandeja de entrada. Lo podemos usar de forma interactiva ejecutando el comando sin opción alguna. Su uso es sencillo y la sintaxis la siguiente:

mail -s "asunto" <lista de correos>

Existen numerosas formás de enviar un correo. Por citar algunas:

- Enviar un correo cuyo cuerpo de mensaje será lo que tengamos en un archivo del equipo:

```
mail -s "Correo de pruebas 1" unadireccion@unservidor.com
otradireccion@otroservidor.es otramáspara@terminar.org <
/home/nebul4ck/Escritorio/mimsg.txt
```

- Enviar un correo escribiendo directamente el texto. Esta forma es útil para utilizarla dentro de *scripts*:

```
mail -s "Correo de pruebas 2" undestino@jojotamail.com << EOF
Lo que escriba aquí será lo que aparecerá en el cuerpo
del mensaje hasta que en el texto se encuentre la palabra "EOF"
como comienzo de línea. Acabo ya.
EOF
```

- Podemos usar las pipe para redireccionar un texto al comando *mail*. Por ejemplo, usando el texto del primer ejemplo o el comando echo, sería algo así:

```
$ cat /home/nebul4ck/Escritorio/mimsg.txt | mail -s "Esto funciona
igualmente" nebuldirc@midirec.org
$ echo Mandando este mensaje como cuerpo de email | mail -s "Ahora con
echo" nebuldirc@midirec.org
```

- Para terminar podemos usar el comando *mail* sin especificar texto ni archivo y de esta manera al pulsar *ENTER* tras escribir el asunto y la o las direcciones. Se nos pedirá que si queremos mandar con copia (**cc:**), si queremos introduciremos la dirección si no, basta con volver a pulsar *ENTER* y comenzar a escribir el *mail*. Finalizamos con *Control+D*.

Nota: Antes de pulsar *Control+D* hay que pulsar *ENTER* para dar un salto de línea.

mailq, postsuper y postqueue

El programa **mailq** es la principal herramienta para administrar la cola de correo. Originalmente este programa formaba parte del paquete *sendmail*, es por ello que usar **sendmail -bp** es equivalente a *mailq*. El comando básico (sin opciones) muestra el contenido de la cola de correo de todos los sistemás.

Podremos vaciar una cola de correos con **sendmail -q**, **postqueue** (*postfix*) o **runq** (*exim*). Además en *postfix* contamos con la utilidad **postsuper** con la que podremos eliminar toda la cola de correos con '**-d** ALL' o un solo correo con '**-d** <*nº de correo*>'

El comando source y shift

source

Utilizado para ejecutar un *script*. Podemos pasarle a *source* el nombre del *script* o utilizar el caracter '**.**' que tiene la misma función:

```
$ source myshell.sh
$ . myshell.sh
```

De este modo el *script* se ejecuta en la consola actual y no se inicia una instancia nueva de la consola, como sucede cuando se ejecuta un *shell script* introduciendo su nombre o utilizando el comando *exec* (que si que inicia una instancia nueva aunque esta controle a la terminal desde la que se ha ejecutado). Existen ciertas ventajas a la hora de ejecutar un *script* con *source*, por ejemplo, si queremos que el *script* haga uso de las variables de entornos definidas en la consola (aunque no se hayan exportado) desde la que se ha ejecutado, este deberá de ser llamado con *source*.

Nota: Por lo general, sólo las variables de entorno que se hayan exportado de forma explícita, estarán disponibles para los *script* que se ejecuten.

Esto pasa igualmente si un *script* que ha sido ejecutado de forma digamos '*convencional*' y en el se definen ciertas variables que se usarán a lo largo del programa. Si en un punto de este programa llamamos a otro *script* y queremos que este haga uso de esas variables, este nuevo *script* deberá de ser llamado con *source*.

Otra ventaja de ejecutar con *source* un *script* es que si este establece variables de entornos, estás estarán disponibles para la consola desde la que se ha invocado al *script*. Esto es lo que ocurre cuando se inicia una terminal y */etc/profile* o *~/.bash_profile* llaman a */etc/bashrc* o *~/.bashrc* respectivamente. Si nos fijamos en estos *script* de inicio de consola veremos que se ejecutan así:

```
if [ -f /etc/bash.bashrc ]; then
. /etc/bash.bashrc
fi
```

Esto produce que el entorno creado por *profile* pueda ser utilizado e incluso modificado por *bashrc*

También, ejecutar un *script* de esta manera evita la sobrecarga asociada a iniciar nuevas consolas, que aunque suele ser imperceptible, podría requerir de recursos si este *script* llamara de forma normal a múltiples *scripts* secundarios generando cada uno una nueva instancia de consola.

Otro dato y como último apunte, es interesante saber que al ejecutar un *script* con *source* se ejecuta en el idioma de la consola que realiza la invocación, mientras que al

ejecutar un *script* de la forma normal, se usa el lenguaje de consola especificado en la línea *hashbang*.

shift : El comando *shift* desplaza los parámetros, de modo que lo que es *$2* se convierte en *$1*, *$3* pasaría a ser *$2* y así sucesivamente, menos *$0* que no es alterado. Podríamos usar *shift* junto con un *bucle* para examinar los parámetros que son pasados a un *script*.

Como siempre mediante un ejemplo lo veremos más fácil. Supongamos que hemos ejecutado la siguiente línea:

```
$ ./miscript.sh uno dos tres cuatro y cinco
```

Y que el contenido de *myscript.sh* es:

```
#!/bin/bash
echo "El número de parámetros es: $#" \\ Esta línea desplegará un 5
echo "Los valores de los parámetros son: $*" \\ El valor que despliega
esta línea es uno dos tres cuatro cinco
shift 3  \\ Con el comando shift indicamos que ahora el parámetro
primero sea $4, pues hemos hecho que empiece inmediatamente después de
la tercera posición
echo -e "Ahora que hemos usado shitf 3 el parámetro 1 es: $1 y el
parámetro 2 es: $2 \\ Esto devuelve: cuatro cinco
echo "Ahora el número de parámetros es: $#" \\ Pues el número de
parámetros ahora es: dos (cuatro cinco)
```

COMANDOS LPIC-1

Capítulo 10

Limitar los recursos del sistema

ulimit : Limita el uso de los recursos del sistema

ulimit [opciones] limite

Opciones:

- Indicar que queremos establecer un límite blando (*soft*) el cual es posible sobrepasar: **-S**
- Establecer un límite duro (*hard*) no se puede sobrepasar: **-H**
- Conocer todos los límites actualmente establecidos: **-a**
- Limitar el tamaño de los archivos resultantes de los volcados del *kernel*: **-c**
- Limita el tamaño máximo del conjunto de datos de programás: **-d**
- Crear un límite a los archivos creados por el *shell* (opción por defecto): **-f**
- Tamaño máximo que puede ser limitado en memoria: **-l**
- Tamaño máximo de memoria del conjunto residente: **-m**
- El tamaño máximo del *buffer* de una tubería: **-p**
- Definir el tamaño máximo de la pila de memoria: **-s**
- Limitar el tiempo de *CPU* en segundos: **-t**
- Limitar el número de procesos que puede ejecutar un mismo usuario: **-u**
- Limitar el tamaño máximo de memoria virtual disponible para un proceso: **-v**

Administrar TCP Wrappers

tcpdchk : Comprueba la configuración de *tcpd* (*TCP Wrappers*)

tcpdchk [opciones]

Opciones:

- Informar sobre las reglas de control de acceso que permiten el acceso sin el uso de la palabra clave *ALLOW*: **-a**
- Examinar los archivos *hosts.allow* y *hosts.deny* del directorio actual: **-d**
- Mostrar el contenido de las reglas de acceso definidas: **-v**

tcpdmatch : Si le pasamos como argumento el nombre y host de un servicio seguido de un usuario determinado y el nombre del host, nos mostrará como se encuentra configurado el servicio para ese usuario.

```
tcpdmatch smb@localhost miuser@localhost
client: hostname localhost
client: address 127.0.0.1
client: username alberto
server: hostname localhost
server: address 127.0.0.1
server: process smbd
access: granted
```

Configurar el Netfilter IPtables

iptables : Herramienta que nos permitirá una administración eficiente del *Netfilter IPtables*. Podremos crear reglas en determinadas tablas y filtros, de manera que reenviemos información, filtremos paquetes, permitamos o deneguemos el acceso a conexiones, etc…

- Ver todas las reglas actualmente configuradas en IPtables:

```
# iptables -L
```

- Ver todas las reglas actualmente configuradas en la tabla FILTER de IPtables:

```
# iptables -t filter -L
```

- Ver todas las reglas actualmente configuradas en la tabla NAT y cadena OUTPUT de IPtables:

```
# iptables -t nat -L OUTPUT
```

- Eliminar todas las reglas definidas en una tabla concreta (-F) y además poner sus contadores a cero (-Z):

```
# iptables -t nat -F -Z
```

Nota: Podemos hacerlo para una cadena concreta indicando la cadena (por ejemplo *INPUT*) tas la -F y -Z

- Insertar (**-I**) en la línea **3** una regla "*compleja*" en la que la acción (**-j**) es de rechazar (**REJECT**) las conexiones entrantes (**INPUT**), desde cualquier origen (**-s 0.0.0.0**) hacia una máquina concreta (**-d 192.168.1.23**) por el puerto 80 (**–dport 80**) con el protocolo tcp (**-p tcp**). Además le añadimos la opción de emitir una respuesta tipo tcp-reset (**–reject-with tcp-reset**):

```
# iptables -I 3 INPUT -p tcp --dport 80 -s 0.0.0.0 -d 192.168.1.23 -j
REJECT --reject-with tcp-reset
```

Nota: hemos prescindido de la opción **-t <tabla>** pero se podría haber indicado. Para añadir una regla en la línea tres evidentemente antes tiene que existir la 1 y la 2. Las reglas más permisivas se colocan en las primeras líneas de lo contrario no tendrán efecto.

- En vez de insertar, vamos a añadir (**-A**) una regla más básica también en la tabla *FILTER* pero esta vez, indicándolo. En ella se aceptará todo el tráfico entrante hacia un segmento de red determinado:

```
# iptables -t filter -A INPUT -s 192.168.1.0/24 -j ACCEPT
```

- Ahora vamos a eliminar esta regla concreta, que sabemos que se encuentra en la línea 2 de */etc/sysconfig/iptables*:

```
# iptables -D INPUT 3
```

- Si no sabemos su posición exacta:

```
# iptables -D INPUT -s 192.168.1.0/24 -j ACCEPT
```

- Creamos una regla en la tabla NAT en la que haremos PREROUTING de manera que todo el tráfico entrante desde Internet por la interfaz eht0 (**-i eth0**) sea analizado y aquello que vaya dirigido al puerto 80 lo desvíe (**DNAT**, cambia la dirección *IP* destino) a un determinado host de nuestra red, por ejemplo un servidor web apache:

```
# iptables -t nat -A PREROUTING -p tcp --dport 80 -i eth0 -j DNAT --to
192.168.1.52
```

- Crear la regla de oro, es decir, conectar todos los ordenadores de una red privada con Internet a través de un dispositivo que posee dos interfaces de red, una conectada hacia Internet y la otra hacia el *switch* de la red privada. Esto recibe el nombre de *Source NAT* o **SNAT** (cambia la dirección *IP* de origen, en este caso para convertirla en la *IP* pública de nuestro *ISP*):

  ```
  # iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -o eth0 -j
  SNAT --to 88.23.144.210
  ```

  Añadimos esta regla a la cadena *POSTROUTING* indicando el origen (**-s**) nuestra red privada. Solo se aplicará a los paquetes que salgan (**-o**, *output*) por la interfaz de red '*eth0*', la acción (**-j**) será *SNAT* y la nueva *IP* es *88.23.144.210* (la ofrecida por nuestro *ISP*). Existen diferentes formás de conocer nuestra *IP* pública, por ejemplo a través del *router* o de Internet, accediendo a direcciones como http://www.cualesmiip.com

  Aviso: Este tipo de regla es configurada de esta manera cuando la *IP* que nos ofrece nuestro *ISP* es estática. Suele ser un servicio extra ofrecido por la compañía mediante pago.

- Ahora vamos a crear la misma regla pero para una *IP* dinámica, es decir, lo común en cuanto a las *IP* ofrecidas por los *ISP*.

  ```
  # iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -o eth0 -j
  MÁSQUERADE
  ```

  Como vemos solo cambia la acción, que en este caso usamos *MÁSQUERADE* que lo que hace es usar la *IP* que tiene en esos momentos configurada nuestra interfaz *eth0*

ufw : Algunas distribuciones como *Debian* y sus derivados, traen preinstalados el paquete *ufw* (*Uncomplicates Firewall*) que no es más que un *front-end* para *IPtables*, de manera que podamos crear reglas de una forma más sencilla. Para los fanáticos del GUI existe **GUFW** que igualmente es un *front-end* (esta vez gráfico) para crear reglas *IPtables*.

Un uso sencillo de *ufw* podría ser el siguiente:

- Denegar todo el tráfico entrante y permitir todo el saliente:

```
$ sudo ufw default deny incoming
$ sudo ufw default allow outgoing
```

- Abrir un puerto para un determinado servicio (bien por número o por nombre de servicio):

```
$ sudo ufw allow 139
$ sudo ufw allow samba
```

Nota: Para eliminar una regla añadimos **delete** entre *ufw* y *allow*

- Comprobar los nombres de servicio que maneja *ufw*:

```
$ sudo ufw app list
```

- Abrir un puerto y especificar el tipo de protocolo:

```
$ sudo ufw allow 443/tcp
```

Nota: Para abrir un rango usamos *xxxx:yyyyy/protocolo*

- Crear una regla algo más compleja (permitir todas las conexiones ssh a mi máquina cuyo origen se encuentra dentro de mi red)

```
$ sudo ufw allow from 192.168.1.0/24 to 127.0.0.1 app ssh
```

- Ver la configuración actual:

```
$ sudo ufw status verbose
```

- Resetear todas las reglas:

```
$ sudo ufw reset
```

- Habilitar o deshabilitar el servicio al inicio:

```
$ sudo ufw enable/disable
```

Comandos SSH

ssh : Nos permite realizar *logins* remotos de forma segura. Es el sustituto de *rlogin* y *rsh*. Contiene bastantes opciones pero su uso básico se basa en escribir un usuario y un host al que acceder seguidos del comando *ssh*.

ssh [opciones] usuario@ip-destino

Opciones:

- Usar la versión 1 del protocolo: **-1**
- Usar la versión 2 del protocolo: **-2**
- Forzar el uso de direcciones IPv4: **-4**
- Forzar el uso de direcciones IPv6: **-6**
- Utilizar un archivo de configuración para el usuario alternativo a *~/.ssh/config*: **-F** *<archivo>*
- Indicar el puerto a utilizar: **-p** *<puerto>*
- Habilitar el uso de X11 forwarding: **-X**
- Deshabilitar X11: **-x**
- Imprimir el número de versión y salir: **-V**

Nota: Estas opciones pueden ser especificadas directamente desde el archivo de configuración del usuario *~/.ssh/config*

scp : Secure copy. Copia archivos a equipos remotos de forma segura.

scp <archivo-origen> usuario@ip-destino:/dir/de/destino/

Ejemplo:

```
$ scp /home/nebul4ck/Desktop/peliculas.txt
films@videoteca.freeser.es:/home/films/Cine/
```

Copiará el archivo *peliculas.txt* dentro del directorio *Cine* del usuario *films* en el host *videoteca.freeser.es*

ssh-keygen : Genera par de claves por ejemplo para tener acceso a máquinas a las que se le haya transferido la parte pública de la clave una vez generada. Podremos generar un par de claves de la siguiente forma:

```
# ssh-keygen -q -t rsa1 -f /etc/ssh/ssh_host_key -C '' -N ''
```

ssh-copy-id : Nos permite copiar la clave pública en el archivo de host conocidos de un

equipo remoto. Se nos solicitará la clave la primera vez que estemos copiando la clave pública. El resto de conexiones se podrán hacer sin contraseñas. Combinando este comando y el anterior (*ssh-keygen*) podremos hacer conexiones seguras sin requerir de la contraseña del usuario remoto de la siguiente forma:

1 . Creamos nuestro par de claves:

```
ssh-keygen -b 4096 -t rsa
```

donde:

-b genera una clave de 4096 bits, por defecto (si no usamos -*b*) se generará de 2048, que suele ser suficiente.

-t indica el tipo de clave, en nuestro caso *rsa* para protocolo 2

Nota: Deberemos de pulsar tres veces *[ENTER]*

2 . Copiamos de forma automática nuestra clave pública en un equipo remoto para un usuario específico, tal que así:

```
ssh-copy-id usuario@ip_del_equipo
```

En este paso se nos pedirá el password de "*usuario*". Con esto habremos acabado.

ssh-agent : Agente de autenticación. Utilizado para conectar a un equipo sin requerir la contraseña del usuario. Antes debe de ser configurado. Es similar a la opción anterior en la que utilizábamos ssh-keygen y ssh-copy-id.

Para utilizar *ssh-agent* deberemos de seguir unos pasos sencillos pero necesarios:

1 . Desde el sistema cliente (el que utilizaremos para conectar contra un servidor) generaremos una clave *SSH* versión 2 (*rsa y dsa*, para versión 1 usar *rsa1*):

```
$ ssh-keygen -q -t rsa -f ~/.ssh/id_rsa -C ''
```

2 . Se nos pedirá una frase de contraseña la cual será la que utilicemos para futuras conexiones

3 . Ahora para que todo empiece a funcionar deberemos de acceder al equipo remoto (*ssh* ***usuario_remoto****@servidor*) escribiendo la contraseña del **usuario_remoto** y copiaremos el contenido de nuestro archivo *id_rsa.pub* al final del archivo *~/.ssh/authorized_keys* o *~/.ssh/authorized_keys2* del directorio */home* de **usuario_remoto**. Si no existiese ninguno de estos dos deberíamos de crear uno de ellos.

Nota: Puede que en algunos sistemás tengamos que cambiar los permisos del archivo *~/.ssh/authorized_keys* y directorios. Normalmente es suficiente con eliminar los permisos de escritura (0600) y asegurándonos de que el propietario es **usuario_remoto**.

4 . Ahora ya podremos usar *ssh-agent* desde el sistema cliente. Para ello ejecutaremos

```
$ ssh-agent /bin/bash
```

Desde esta sesión conectaremos escribiendo primeramente **ssh-add ~/.ssh/id_rsa** para añadir nuestra clave *RSA* al conjunto gestionado por *ssh-agent*. Se nos pedirá que introduzcamos la frase/clave que introducimos al crear el par de claves en el paso 1.

Aviso: Cada vez que queramos volver a realizar accesos al sistema remoto al que hemos facilitado nuestra clave pública no deberemos de dar la contraseña del usuario remoto, pero si que tendremos que repetir el paso 4.

Si utilizamos este sistema con frecuencia desde nuestro usuario local, podemos hacer que cada vez que se inicia *bash* lo haga ejecutando *ssh-agent*. Esto podemos lograrlo modificando */bin/bash* por **ssh-agent /bin/bash** en la línea de nuestro usuario del archivo */etc/passwd*.

Comandos para el uso de GPG

gpg : Alternativa libre del sistema de encriptación y firmás *pgp*

- **Generar** claves:

```
$ gpg --gen-key
```

Tras responder a una serie de preguntas relacionadas con la clave, como por ejemplo el nombre completo de la persona, dirección de correos, contraseña (muy importante recordarla y no perderla), etc… se crearán las claves en el directorio *~/.gnupg*.

- **Comprobar** las claves que tenemos en nuestro anillo de claves:

```
$ gpg -k
```

o

```
$ gpg --list-keys
```

- **Exportar** claves a usuarios en formato binario o en *ASCII* para poder enviar por correos:

```
$ gpg --export ID_Clave > nebul4ck-gpg.pub
```

o

```
$ gpg --output nebul4ck-gpg.pub --export ID_Clave
```

Con este comando estaremos generando una clave pública (*nebul4ck-gpg.pub*) la cual se creará en el directorio en el que estemos trabajando y que posteriormente distribuiremos para que puedan verificar nuestros correos/documentos o encriptar sus mensajes para que solo nosotros podamos descifrar.

Si teníamos pensado enviar nuestra clave pública por correo, podríamos haber añadido la opción **–armor** al comando *gpg –export* de manera que se generara una salida *ASCII*.

- **Exportar a depósitos de claves** (servidores de red que contienen multitud de claves públicas):

```
$ gpg --keyserver pgp.rediris.es --send-keys ID_denuestraclave
```

Este comando enviará nuestra clave al servidor de claves *pgp.rediris.es* (servidor de claves públicas) para que posteriormente pueda ser descargada por algún interesado.

Aviso: Para crear claves públicas anteriormente utilizamos el **uid** (nombre o correo electrónico) o bien el código de la línea **pub** de la salida del comando *gpg -k*. Para subir una clave pública a un depósito de clave solo servirá el ID de la linea **pub**.

- **Importar** claves públicas de terceros:

```
$ gpg --import <nombre-archivo>
```

O bien, importarla desde un servidor de claves:

```
$ gpg --key-server <nombre_host> --recv-keys <id_clave>
```

- **Revocar** clave pública:

Nota: Es muy importante generar un certificado de revocación de clave si hemos decidido enviar nuestra clave a un servidor, ya que está será distribuida en pocos minutos al resto del mundo.

Para **crear** el certificado de revocación haremos lo siguiente:

```
$ gpg --gen-revoke <ID_clave>
```

Podemos añadir la opción *–output <nombre_archivo.gpg>* si en vez de mostrar el certificado por la *stdout* se añada a un archivo.

Una vez tengamos generado el certificado deberemos de copiar y pegar el texto mostrado por la *stdout* en un archivo (en caso de que no hayamos agregado la opción *–output* directamente). Este será nuestro certificado de revocación. Si llega el momento de revocar la clave deberemos de importar este certificado en nuestra clave, bastará con:

```
$ gpg --import <nombre_archivo.gpg>
```

Deberemos de distribuir esta revocación a los usuarios que estaban haciendo uso de nuestra clave. Si la teníamos subida a un depósito de claves público, deberemos de volver a usar el comando *gpg* con las opciones *–keyserver y –send-keys* para importar nuestra clave revocada.

- Para **cifrar** un documento con la clave pública de un usuario (por ejemplo *jjgomez*) desde el usuario *nebul4ck* podremos hacer lo siguiente:

  1 . Se supone que ta tenemos importada a nuestro anillo de claves la clave pública de *jjgomez* por lo que vamos a comprobar su ID:

```
$ gpg -k
/home/nebul4ck/.gnupg/pubring.gpg
--------------------------------
pub 2048R/BCK2ADAK 2015-03-28 [expires: 2015-03-30]
uid nebul4ck <nebul4ck@freeser.com>
```

```
sub 2048R/2AD3B0D9 2015-03-28 [expires: 2015-03-30]
pub 2048R/5GJ945FD 2015-03-30 [expires: 2015-04-01]
uid Jose Julio Gómez (Mi mail) <jjmgomez@gmail.com>
sub 2048R/CB851754 2015-03-30 [expires: 2015-04-01]
```

2 . Ciframos de la siguiente manera (usaremos el *nombre*, *correo* o *ID* de *jjgomez*)

```
$ sudo gpg --output contabilidad_2015.dot.gpg --recipient
5GJ945FD --encrypt contabilidad_2015.gpg
```

De esta manera obtendremos un archivo binario, por lo que no podremos pasarlo directamente por *email* a menos que sea como adjunto. Podremos añadir la opción **–armor** para codificarlo en *ASCII* y poder copiar y pegar el texto en un correo.

```
$ sudo gpg --output contabilidad_2015.dot.gpg --recipient
5GJ945FD --armor --encrypt contabilidad_2015.dot
```

Ahora vamos a **descifrar** el documento desde el usuario *jjgomez*:

```
$ sudo --output contabilidad_2015.dot --decrypt
contabilidad_2015.dot.gpg
```

Nos pedirá la contraseña (la que introdujo en este caso *jjgomez* cuando generó su par de claves). Y listo! ya podremos leer el documento.

- Para **firmar** un documento podremos usar las opciones **–sign** o **–clearsign** de *gpg*. La diferencia entre ambas opciones es que *–sign* además de firmar el documento, lo encriptará con la clave privada, de manera que el destinatario (*jjgomez*) podrá verificar que se trata de un documento procedente de nosotros mediante la opción **–verify <archivo>** pero no podrá leerlo a menos que lo descifre antes con *–decrypt*. Si utilizamos *–clearsign* el documento irá sin cifrar, por lo podrá leerse sin problemás y comprobar igualmente mediante *–verify* que el documento fue emitido por *nebul4ck*.

  Utilizar *–sign* provocará que se cree un archivo encriptado con el mismo nombre que el original pero con extensión **.gpg**. Si utilizamos *–clearsign* el archivo resultando tendrá la terminación **.asc**

  ```
  $ gpg --clearsign contabilidad_2015.dot
  $ gpg --verify contabilidad_2015.dot.asc
  ```

  Si queremos profundizar algo más en el uso de *gpg* podremos acudir a la siguiente dirección https://www.gnupg.org/gph/es/manual.html, y además leer la página man de *gpg*.

Glosario GNU/Linux

- Acceso aleatorio (**RAM**): Memoria con escasa o nula repercusión en la velocidad de E/L.
- Administrador de arranque: Los equipos basados en BIOS combinan el *Administrador de arranque* (quien selecciona el OS) y el *Cargador de arranque* (quien arranca el OS).
- **ALSA** (Advance Linux Sound Architecture): Controladores de sonido para Linux.
- Archivo estático: Las particiones que solo contienen archivos estáticos se recomienda montarlas en solo lectura.
- **ARP** (Address Resolution Protocol): Protocolo encargado de averiguar una dirección de hw mediante su IP.
- **ASCII** (American Standard Code for Information Interchange): Código Americano Estándar para el Intercambio de Información. Al carecer de simbología para la escritura en otros idiomás, se está reemplazando por UTF-8.
- Back port: Práctica por la cual se incorporan funciones más avanzadas en versiones "anteriores" de un paquete. Un ejemplo sería incluir soporte para hw de *kernels* en desarrollo, en *kernels* más estables.
- Baudio: Unidad de medida de la velocidad de transmisión de datos sobre líneas en serie (nº elementos de señal/segundos). En ocasiones un baudio no corresponde con bits/s, ya que algunos módems codifican más de un bit por elemento de señal, por lo que no siempre el baudio es sinónimo del bits/s.
- Bibliotecas dinámicas y estáticas: La dinámica es compartida normalmente por más de un programa (cargándola en RAM y **HDD**), mientras que la estática es introducida en la compilación de un programa, algo que solo se debe hacer si es estrictamente necesario.
- **BIOS** (Basic Input/Output System): Se instala en **ROM** y su código es leído al encender el PC. Es la encargada de escalar el arranque del OS al Administrador de arranque.
- Búfer circular del *kernel*: Registro de los mensajes recientes del *kernel.* Una forma de acceder a estos es a través del comando *dmesg*.
- Carga media: Forma de medir la demanda de tiempo en CPU de un programa en ejecución. Un 0 indica que ningún programa está haciendo uso de la CPU. Un 1 indica que existe un programa en ejecución realizando una demanda constante de CPU. Un 2 o más indica que varios programás están compitiendo por el uso de la CPU.
- **CIDR** (Classes Inter Domain Routing): Método para dividir direcciones IP en subredes.
- Código fuente: Código de texto plano (ASCII) que al compilarlo se convierte en un binario capaz de ser ejecutado por el PC.
- Dirección de circuito cerrado: Normalmente 127.0.0.1 y hace referencia a la interfaz de red del propio ordenador (localhost).
- **DMA** (Direct Memory Addressing): La localización directa de memoria se emplea para transferir datos entre dispositivos (tarjetas de sonido, adaptadores SCSI…)
- **EEPROM** (Electronically Erasable Programmable Read-Only Memory): PROM programable y borrable electrónicamente.
- **EFI** (Extensible Firmware Interface): Al igual que BIOS, EFI controla el inicio del sistema de nivel inferior.
- **EHCI** (Enhanced Host Controller Interface): Interfaz de host mejorada. Es un tipo de controlador para puertos USB 2.0.

- **ESP** (EFI System Partition): Partición FAT del sistema EFI en la que este busca cargadores de arranque, controladores o archivos de arranque de nivel inferior.
- **FHS** (Filesystem Hierarchy Standard): Es el estándar de jerarquía de un sistema de archivos. En Linux define el nombre de los directorios esenciales así como su contenido.
- Firmware: Software instalado en Hardware, que controla las funciones de este.
- Formateo a alto nivel: "Creación de un sistema de archivo".
- Formateo a bajo nivel: Creación de la estructura de datos en un disco que define la ubicación de los sectores y pistas individuales. En Linux puede hacerse con el comando *fdformat*.
- Frame: Paquete de red asociado al hw de red (ethernet).
- **FSF** (Free Software Foundation): Fundación que promueve el software libre. Cambridge (Mássachusetts).
- Geometría **CHS** (Cylinder/Head/Sector): Método de creación de direcciones para los HDD con el que identificar un sector específico.
- Globbing: El uso de comodines. glossary.txt => glos*.txt
- **GMT** (Greenwich Mean Time): La hora de Inglaterra sin aplicar el horario de verano. Linux convierte esta hora (usada internamente) a la hora local, basándose en la zona horaria especificada.
- **GNU**: Proyecto patrocinado por la FSF, para crear una implementación libre de UNIX.
- **GPG**: Software para encriptar datos. Hace uso de una clave pública y otra privada.
- **GPT** (Guid Partition Table): Tipo de tabla de particiones que pretende sustituir a MBR. Es parte de EFI, aunque también BIOS permite su uso.
- **GUID** (Globally Unique Identifier): Identificador Único Global. Número de 128 bits con el que se identifica la ejecución de un programa. La posibilidad de que 2 programás o la ejecución doble de un mismo programa compartan GUID es remota.
- Hash: Método de encriptación en el que un archivo o cadena de texto se codifica de una manera que no se puede revertir.
- **HFS** (Hierarchical File System): El sistema de archivos jerárquico usado por Mac OS.
- **ICMP** (Internet Control Message Protocol): Es un tipo de paquete de red que se puede utilizar para señalar estados de error, como cuando existen paquetes dañados.
- **IPsec** (Internet Protocol Security): Son una serie de herramientas que mejoran la seguridad de Internet mediante la codificación y autenticación de paquetes de red concretos. Se originó a partir de IPv6, pero tiene compatibilidad inversa con IPv4.
- **IRQ** (Interrupt Request): Petición de interrupción. Método por el que los periféricos indican que necesitan que la CPU les preste atención.
- **IPv4**: Primera versión implementada a gran escala del Protocolo de Internet. Usa direcciones de 32bits y se encuentra limitado 4.294.967.296 de IPs únicas.
- **IPv6**: Es el Protocolo de Internet que sustituirá a IPv6 por la escasez de nuevas direcciones IP. Usa direcciones representadas en hexadecimal y admite un total de 670 mil billones de direcciones por cada milímetro cuadrado de la superficie de la tierra.
- **JFS** (Journaled File System): Uno de los sistemás de archivos con respaldo de transacciones para Linux, desarrollado por IBM para SO AIX y posteriormente se creó otra implementación para OS/2 (es de esta de la que deriva la de Linux).
- Joliet: Extensión de ISO-9660 para CDROM la cual admite nombres largos al estilo de MS Windows.

- **LBA** (Logical Block Addressing): Método contrapuesto a CHS por el que se accede a los datos de un disco a través del número de sector (de forma lineal).
- **MBR** (Máster Boot Record): Hace referencia al primer sector de un disco y contiene la tabla de particiones, así como el código que lee la BIOS para escalar el arranque del OS. Puede albergar hasta un máximo de 4 particiones primarias y una de ellas extendida.
- Módulo: Controlador del *kernel* u otro componente del *kernel* almacenado en un archivo independiente. Linux puede cargar módulos bajo pedido ahorrando espacio en RAM cuando los módulos no son utilizados y reduciendo el tamaño del *kernel*.
- Multidifusión: Mientras la difusión de mensajes suele enviar datos a todos los ordenadores de una red, la multidifusión envía datos de red a varios "sitios" remotos específicos.
- **NDP** (Neighbor Discovery Protocol): Protocolo de detección de vecinos. Como ARP pero para IPv6.
- **NFS** (Network Fyle Sistem): Sistemás de archivos de red. Protocolo para compartir archivos entre ordenadores Linux/Unix
- Nodo de indice: Corresponde a una estructura de datos del sistema de archivos, que contiene información esencial del archivo, como su tamaño y ubicación en el disco.
- Open Relay: Conexión abierta. **SMTP** por ejemplo usa este termino para enviar correos desde cualquier origen a cualquier destino. Los spammers hacen un uso abusivo de este termino.
- **OSS** (Open Sound System): Es un controlador como ALSA, normalmente usado en versiones de *kernel* anteriores al 2.6.x.
- Partición activa: Partición que contiene una entrada en el MBR en la que se indica que es arrancable.
- Partición primaria: Una de las 4 posibles particiones primarias posibles en la tabla de particiones MBR.
- Partición extendida: Son particiones marcadoras de posición de otras particiones (lógicas). Es posible tener 3 particiones primarias (una con cada OS por ejemplo) y una partición extendida que nos permita crear más particiones 8ahora lógicas) en las que crear otros sistemás de archivos.
- Partición lógica: Partición que no posee entradas en la tabla de particiones MBR, ya que están referenciadas por la partición extendida.
- **PCL** (Printer Control Language): Lenguaje creado por Hewlett Packard para controlar impresoras de gama media. Existen diferentes versiones de PCL, que van desde PCL3 a PCL6.
- Protocol stack o Pila de protocolos: Colección de controladores, procedimientos del *kernel* y otro software que implementan un medio estándar para la comunicación a través de la red.
- **PnP** (Plug and Play): Algo así como "Conectar y usar", conjunto de estándares de hw que permite una configuración automatizada del hw o bien mediante la instalación de software.
- **POST** (Power-On Self-Test): Auto diagnostico de encendido. Chequeo que realiza la BIOS antes de escalar el arranque del OS al administrador de arranque.
- PostScript: Lenguaje de programación utilizado para el control de muchas impresoras de gama alta. Linux usa *Ghostscript* para traducir el formato Postscript a mapa de bits para que sea entendido por muchas impresoras de gama media/baja.

- **PPD** (Postscript Printer Difusion): Archivo de configuración que proporciona características y configuraciones de una impresora.
- **PPP** (Point to Point Protocol): Método para iniciar una conexión TCP/IP entre dos ordenadores sobre una linea serie RS-232 o modems.
- **PPPoE** (PPP over Ethernet): Protocolo punto a punto sobre Ethernet, variante PPP optimizado para conexiones en redes ethernet.
- Respaldo de transacciones: Función avanzada de un sistema de archivos que registra la información de las operaciones de disco, con el fin de que si se produce un corte en el suministro eléctrico, a la hora de arrancar la máquina se haga de una forma, segura, eficaz y rápida.
- Rock Ridge: Conjunto de extensiones del sistema de archivo ISO-9660 que permite además de almacenar nombres largos (Joliet) propiedades, permisos y otras características.
- RootKit: Conjunto de scripts y demás software que permiten a los *scripts kiddies* entrar en ordenadores ajenos.
- **SCSI** (Small Computer System Interface): Interfaz estándar para discos duros, CD-ROM, dispositivos de cinta, escáneres y otros dispositivos.
- **SAS** (Serial attached SCSI): SCSI de conexión en serie.
- Script Kiddies: Persona con pocos conocimientos o habilidades que acceden a ordenadores ajenos empleando scripts creados por otros. Suelen dejar rastros evidentes.
- Sector de arranque: Sector de disco desde donde la BIOS lee códigos de cargadores de arranque. Por lo general suele ser el primer sector del disco. EFI no depende de sectores de arranque.
- Sistema de archivos: Estructura de bajo nivel grabada en un disco con el fin de saber dirigir los datos de archivos, determinar características como el tamaño máximo de la partición, las reglas de nomenclatura de archivos, información adicional que se puede asociar a un archivo (marcas temporales, propietario…), además de estructurar el árbol de directorios. Algunos sistemás de archivos con respaldo de transacciones son: ext3, ext4, reiserfs, JFS y XFS).
- **SLAAC** (Stateless Address Auto-Configuration): SImilar a DHCP pero para IPv6.
- Smart Relay (Transmisión Inteligente): Configuración de un servidor de correos electrónico que conlleva a redirigir todo el tráfico saliente a un servidor de correos superior, en lugar de hacerlo directamente al destino. Configuración típica en redes pequeñas donde el ISP bloquea las conexiones SMTP a cualquier servidor que no sea el propio como medida de control de spam o si por ejemplo se desea que un servidor de correos saliente sea el que envíe toda la información, para por ejemplo tener un control de seguimiento.
- Socket: Estructura informática que permite la conexión hacia puertos finales. Conjunto de IP y puerto.
- Squashing: En un servidor NFS hace referencia al cambio del UID o GID (aquel que envía peticiones de acceso) como medida para limitar la vulnerabilidad. La cuenta *root* del cliente suele enmáscararse como cuenta *nobody* en el servidor.
- Sticky bit: Bit de permiso especial utilizado mayormente en directorios. Cuando está definido, solo el propietario del archivo puede realizar modificaciones sobre este, aunque el directorio tenga permisos de escritura/lectura/ejecución para todos.

- **SUID** o **SGID**: Permiso especial que hace que un determinado programa se ejecute con los privilegios de su propietario.
- Suma de control: Una sencilla comprobación de la integridad de un archivo en la que se suma los valores de los bits o bytes individuales y se comparar con un valor almacenado de una versión de referencia del archivo.
- Súper Servidor: Servidor que escucha las conexiones de red destinadas a otros servidores (los cuales son controlados por el súper servidor) y los inicia. Como ejemplo en Linux: inetd y xinetd.
- Súper bloque: Estructura de datos de un sistema de archivos que contiene información sobre este, como su tamaño y número de revisión.
- TrueType: Formato de fuentes desarrollada por Apple y que domina los mundos MacOS y MS Windows. La compatibilidad con TrueType en Linux la proporcionan determinados servidores de fuentes: XFree86 4.x, Xorg-11 y Xfs.
- **UEFI** (Unified Extensible Firmware Interface): La versión 2.x de EFI.
- **Umásk** (Máscara de usuario): Patrón de bits que representa los bits de permisos que se van a eliminar de los nuevos archivos según los creen los procesos ejecutados.
- **XDMCP** (X Display Manager Control Protocol): Protocolo para administrar conexiones X en una red, también utilizado para mostrar pantallas GUI de acceso local en estaciones de trabajo Linux. Los servidores X habituales en Linux son: **XD**M, **GDM** y **KDM**.