CURSO DE LINUX PARA NOVATOS, BRUTOS Y EXTREMADAMENTE TORPES

Autor: Antonio Castro Snurmacher.

Primera edición (Nov-2008)

(Previamente fue publicado en Ene-2000 en formato html en la web http://www.ciberdroide.com)

Ausencia de Garantía

Esta ausencia de garantía se hace extensa a cualquier tipo de uso de este material y muy especialmente a las prácticas, ejercicios, y de ejemplos que encuentre en estas páginas. Deberá trabajar siempre salvo indicación contraria con un SO Linux y con un usuario distinto de 'root' sin privilegios especiales. Como directorio de trabajo se procurará usar el directorio '/tmp' o algún otro que no contenga información valiosa. Tampoco se considera buena idea practicar en una máquina que contenga información valiosa.

Todo esto son recomendaciones de prudencia. En cualquier caso si algo sale mal toda la responsabilidad será únicamente suya. En ningún caso podrá reclamar a nadie por daños y perjuicios derivados del uso de este material. Absténgase de hacer prácticas si no está dispuesto a asumir toda la responsabilidad.

Notas del autor:

Este curso ha sido reeditado para cambiar el formato del texto. No se ha tocado apenas el texto que es de hace ocho años. Está pendiente de corrección ortográfica y de estilo. Pese a sus fallos, ha sido ampliamente descargado en Internet. La mayor parte del curso conserva su vigencia gracias a tratar los fundamentos UNIX del S.O Linux. El curso es una introducción general a los sistemas operativo tipo Unix en general y Linux en particular.

Perdone la crudeza de nuestro título pero cuando termine la primera parte de este curso sentirá que ha superado una desagradable fase de su existencia y podrá reírse de la falta de conocimientos de su vecino.

Cuando termine usted la segunda parte no sé como se sentirá usted, pero yo me sentiré feliz porque todavía no está terminada , y para cuando decida terminarla seguramente habrá que cambiar algunas cosas. La segunda parte de administración ha perdido más vigencia que la primera que continua siendo básica.

Para un correcto aprovechamiento del curso se recomienda una lectura secuencial de cada uno de los capítulos en orden establecido en este índice. (secuencial quiere decir primero el 1, luego el 2, luego el 3, etc...)

Efectivamente este último comentario entre paréntesis va especialmente dedicado a nuestro público predilecto al que nos referimos en el título, pero algunos listillos también deberían probar nuestro curso.

Las convenciones utilizadas en este curso y la metodología de trabajo también se explican de forma progresiva por lo cual saltarse un capítulo puede llevarle a interpretar erróneamente el material del curso. Posiblemente este usted tentado de empezar directamente por la segunda parte del curso pero de momento no se ha considerado facilitar esta labor.

Este curso, como ya hemos comentado, consta por el momento de dos partes bastante distintas.

* La primera parte Iniciación al SO Linux asume un nivel cero de conocimientos. Ni siquiera asumiremos una cierta experiencia con ordenadores y las cosas se explicarán en esta primera parte paso a paso facilitando lo máximo posible la labor del alumno. Romperemos algunos mitos que presentan a Linux como SO para gurús. En esta parte no entraremos nunca a explicar cosas que necesiten forzosamente una cuenta de superusuario. Ello

implicaría cierto riesgo ya que se trata de un usuario con permisos en el sistema para hacer cosas de todo tipo.

- La segunda parte Usuario avanzado de Linux asume que ya ha asimilado la primera parte del curso y probablemente todas sus inseguridades habrán desaparecido. Esta parte tocará algunos temas más relacionados con la administración del sistema y puntualmente necesitaremos acceso a una cuenta de superusuario, pero no será un curso de administración de Linux. El objetivo no es administrar un sistema con muchos usuarios sino lograr un razonable nivel de autosuficiencia con Linux considerando las necesidades de administración del sistema en ordenadores personales con Linux y debe contemplarse como un curso intermedio entre un curso de Iniciación y uno de Administración.

En cualquier caso finalizado este curso que empieza desde un nivel de conocimientos cero, y que avanza consolidando los conceptos paso a paso. Pese a ello es un curso muy ambicioso. Con él logrará con él una base de conocimientos que le darán una gran seguridad a la hora de abordar cualquier desafío en este S.O.

Índice de contenido

PARTE (I) INICIACICION AL S.O. LINUX

A QUIEN VA DIRIGIDO ESTE CURSO

Esta primera parte del curso es una introducción general a los sistemas operativos tipo Unix en general y Linux en particular.

Esto no pretende ser un HOWTO ni una FAQ, ni una guía de usuario, de programación, o de administración. Tampoco es una enciclopedia sobre el tema. Esto es un curso de iniciación de Unix usando Linux para ilustrar los ejemplos.

Tampoco es una guía para hacer las cosas arrastrando y soltando desde un escritorio precioso. Linux tiene varios escritorios de este tipo pero nosotros usaremos la consola. Un escritorio puede ser más agradable y más intuitivo pero no todas las cosas pueden hacerse apretando un solo botón. Además esto es un curso y para aprender a usar un escritorio no hace falta un curso.

Usted no necesita para esta primera parte del curso tener acceso a una cuenta de administrador (root), ni tendrá que usar ningún entorno de programación salvo la propia shell.

Lo que se busca es consolidar unas bases teóricas de los conceptos fundamentales comunes a los SO tipo Unix. En una palabra se busca comprender el funcionamiento de este SO. A pesar de ello su enfoque es práctico porque se ilustran los conceptos con ejercicios y porque se profundiza especialmente sobre los aspectos que pueden resultar de utilidad más inmediata para un novato.

No prejuzgue a Linux como un sistema no apto para usted.

Al finalizar la primera parte del curso usted tendrá conocimientos suficientes para entender como funciona Linux para un uso a nivel de

usuario normalito y podrá utilizar el potentísimo lenguaje shell-script. Este lenguaje y la base de conocimientos de este curso le abrirán una enorme variedad de posibilidades.

Por decirlo de alguna manera cuando termine la primera parte de este curso será plenamente consciente de la enorme potencia y flexibilidad de este SO y se sentirá capaz de hacer cosas que ahora quizás le parezcan impensables.

Esperamos que esto no le convierta en un repugnante listillo informático pero ya dijimos que no nos hacemos responsables absolutamente de nada.

Cuando termine la segunda parte se sentirá afortunado de ser un ... Mmmm ...superviviente que llegó al final del curso.

Usaremos un lenguaje acorde a la falta total de conocimientos previos, en cambio asumiremos que puede usar un ordenador con Linux instalado y listo para practicar. Es decir no explicaremos como se instala linux ni como crear una cuenta de usuario pero asumiremos que ya dispone de ambas cosas y que es capaz de abrir una sesión de trabajo (es decir hacer login con un usuario y password válidas)

Si es usted un virtuoso del azadón y la pala, en hora buena porque este curso es el indicado para usted pero sustituya esas herramientas por un ordenador. Los ejemplos y ejercicios están pensados para que los pruebe en su ordenador con su SO Linux, y la pala, azadón y otras artes similares no le serán de mucha utilidad para este curso. (Espero que mi amigo David se entere bien de esto último).

Bueno con este tipo de comentarios quizás piense que este curso es poco serio. El uso de un poquito de humor lo haremos durante las primeras lecciones para que se sienta mas relajado.

Linux es un SO tipo Unix y por lo tanto sus conceptos más básicos son comunes a los que incorpora cualquier sistema tipo Unix y resultan

bastante distintos de otros conceptos que forman parte de la cultura microinformática fundamentalmente de Microsoft.

La documentación de Linux es muy abundante, sin embargo muchas veces se asume una cultura general de Unix que realmente no siempre existe. Las generalidades más básicas de Unix muchas veces se tratan de un modo superficial y poco sistemático quizás porque son temas muy viejos.

El contenido de este curso es en más de un 90% serviría para cualquier SO tipo Unix y no solo para Linux.

En este curso se explicarán conceptos que inciden el conocimiento interno del sistema operativo. A estas alturas más de uno empezará a preguntarse si realmente es necesario conocer todos estos detalles para un simple uso de un SO. Lo cierto es que actualmente existen escritorios gráficos que permiten hacer bastantes cosas de forma intuitiva. En Linux tenemos por ejemplo KDE o GNOME que permiten usar los programas de forma mucho más amistosa e intuitiva. Sin embargo la amistosidad tiene un precio. Los entornos intuitivos no permiten hacer cualquier cosa y consumen muchos recursos de CPU y memoria. En Linux estos entornos son una opción no una obligación. Este curso esta orientado al uso de Linux desde la consola. Para sacarle el máximo provecho y para alcanzar cierto dominio del lenguaje shell-script no queda más remedio que tratar estos temas pero estamos seguros de que una vez alcanzado el final de este curso tendrá una visión de Unix que le permitirá atreverse con cosas impensables ahora.

También se propone el aprendizaje de los comandos más interesantes del sistema para poder hacer un gran número de cosas y alcanzar un alto nivel de autosuficiencia en su manejo.

Este curso es suficiente para que una persona que únicamente pretenda defenderse en este sistema operativo consiga su propósito, pero además permite que el usuario conozca aspectos que son imprescindibles para poder avanzar mucho más por su cuenta si ese es su deseo.

Se evitará en lo posible mencionar aspectos relativos a la administración del sistema ya que esto sería objeto de otro curso.

Existen documentos que tratan de la administración de Linux y de la instalación de Linux.

Si usted pretende usar su propio SO Linux en su ordenador personal convendría que continuara aprendiendo la administración de un SO Linux ya que usted sería el administrador de su sistema. Recuerde que Linux no fue concebido como sistema monousuario. Solo una recomendación muy básica. Tenga en su ordenador como mínimo dos usuarios. Uno sería el superusuario 'root' con capacidad ilimitada para lo bueno y para lo malo y otro su usuario de trabajo normal. La razón es que una equivocación cometida desde un usuario normal solo puede tener malas consecuencias para el área de trabajo de ese usuario. Por el contrario una equivocación desde 'root' puede destruir toda la información de su sistema.

Dado que vamos a usar Linux desde consola y algunos usuarios pueden tener experiencia previa con Msdos, me parece necesario hacer una nueva advertencia destinada a estos usuarios.

El comportamiento del interprete de comandos de Msdos y de Linux tienen alguna semejanza pero algunas cosas que estamos acostumbrados a usar en Msdos no son otra cosa que burdas imitaciones de las extraordinarias posibilidades del interprete de comandos de Linux. Como consecuencia de esto un intento de hacer en Linux cosas parecidas a las que hacemos en Msdos puede llevar a desagradables sorpresas ya que en Linux se requiere tener una idea de como funciona el interprete de comandos.

Dar una relación de comandos equivalentes entre (Unix o Linux y Msdos) sin explicar nada más puede llevar a confusiones peligrosas. En otras palabras, puedes cagarla si te pasas de listo. En este sentido los comandos básicos para el manejo de ficheros 'rm', 'mv', y 'cp' no se deberían usar sin comprender correctamente el funcionamiento básico del interprete de comandos de Linux y en particular como este expande las ordenes.

Un intento de usar Linux sin un pequeño es fuerzo de asimilación de algunos conceptos puede llevar a situaciones muy frustrantes. Existe incluso hoy en día una gran cantidad de profesionales que han tenido que trabajar durante meses o años con algún SO tipo Unix sin que nadie les explicara las cuatro cosillas básicas imprescindibles. Esto ocurre porque para otros SO resulta perfectamente posible aprender por uno mismo practicando con el sistema pero en nuestra opinión los SO de tipo Unix requieren la asimilación de unos conceptos bastante simples pero en absoluto intuitivos. Los usuarios que tienen alergia a los manuales y que no recibieron formación para Unix raramente llegan a entender este SO y es lógico que se encuentren a disgusto y que maldigan a este SO. Muchos no consideraron necesario que fuera importante asimilar unos conceptos básicos previos ya que la experiencia anterior con otros SO más intuitivos les indujo a pensar que con el simple uso del SO podrían alcanzar ese conocimiento por ellos mismos. Los que tuvieron la suerte de recibir un buen curso o para los que tuvieron la paciencia de investigar por su cuenta los fundamentos de este SO, es fácil que lleguen a enamorarse de el. Unix siempre ha despertado estas dos reacciones. Hay quien lo adora y hay quien lo odia. Los que lo adoran aseguran que es un SO sencillo, elegante, potente y flexible. Los que lo odian dicen que es difícil de usar. Intentamos con este curso que llegue a pensar como los primeros.

INTRODUCCION A LINUX

Un poco de historia

Linux es un kernel (un núcleo de un sistema operativo) creado por Linus Torwalds. Nació en Finlandia el 28 de Dic 1969 y estudió en la universidad de Helsinki. Desarrolló Linux porque deseaba disponer de un SO Unix en su PC. Actualmente trabaja en los EE.UU.

Linus T. quiso compartir su código para que cualquiera pudiera usarlo y contribuir a su desarrollo. Dado que en GNU ya habían desarrollado bastantes herramientas para Unix con la misma filosofía de software libre pronto se consiguió un SO Linux/GNU totalmente libre. Dos de las herramientas más importantes aportadas por GNU fueron el interprete de comandos, y el compilador de C.

En los comienzos Linux era un sistema principalmente adecuado para hackers y personas con muchos conocimientos técnicos. Actualmente ya no es así. El crecimiento en número de usuarios viene siendo exponencial desde sus comienzos en 1991. Actualmente ya empieza a ser visto como una alternativa a los SO de Microsoft. Pues esta historia de la historia de Linux es tan corta como su propia historia, por lo cual este apartado ya es también historia. No se desmoralice. Si no ha entendido este último juego de palabras, puede pasar al siguiente apartado sin preocuparse mucho por ello.

Tipos de licencias libres

La licencia más utilizada en Linux es la licencia GPL de GNU. Sin embargo hay otras licencias igualmente aceptables bajo el término de software libre.

Un buen lugar para informarse sobre los tipos de licencias libres es el 'Debian Policy Manual' en este documento se establecen unos criterios para establecer si Debian considera o no libre una licencia. Esto se describe a continuación de forma resumida.

- **Libertad de distribución**

Se refiere a la libertad de comercializar el software sin que sea necesario pagar derechos de ningún tipo.

- **Código libre**

Se considera que el código es libre cuando los fuentes son de dominio público.

- **Integridad de los fuentes**

Se recomienda no restringir los derechos de modificación del código fuente, aunque se aceptan algunas fórmulas que restringen la forma de efectuar ciertas modificaciones pero no entramos en estos detalles ahora.

- **No discriminación para grupos o personas**

La licencia no considera distinción alguna para ningún tipo de persona o grupo.

- **No discriminación para actividad o propósito**

Por ejemplo no se distingue entre uso comercial, doméstico, educativo, etc.

- **Distribución de la licencia**

La licencia afectará a las sucesivas distribuciones de los programas de forma automática sin necesidad de trámite alguno.

- **La licencia no debe de ser específica de Debian**

Es decir Debian no admitiría una licencia que impida el uso de un programa fuera de la distribución Debian.

- **La licencia no debe contaminar otros programas**

La licencia no debe imponer restricción alguna a otros programas. Por ejemplo no sería admisible obligar a que el programa solo se pueda redistribuir en un medio que no contenga software comercial.

- **Ejemplos de licencias libres**

GPL, BSD, y Artistic son ejemplos de licencias libres.

Cada licencia tiene sus peculiaridades. Por ejemplo si usted desarrolla aplicaciones haciendo uso de fuentes protegidas pajo la licencia GPL estará asumiendo para todo su desarrollo la condición de GPL. En cambio podría desarrollar software propietario derivado de fuentes bajo licencia BSD. La GPL se ha hecho muy popular porque protege el legado del software libre para que continúe como tal.

Que es Linux/GNU

Quizás ha leído que Linux significa 'L'inux 'I's 'N'not 'U'ni'X'. Pero en realidad Linux es un núcleo de SO tipo Unix. Su compatibilidad Posix es alta. El SO se complementa con una serie de aplicaciones desarrolladas por el grupo GNU. Tanto estas aplicaciones como el núcleo son software libre. Linux/GNU es un SO tipo Unix, SO Multiusuario, Multitarea, Multiprocesador, Multiplataforma, Multilingue, nacido en la red de redes Internet.

Unix se origino en los laboratorios Bel AT&T a comienzos de 1970 y el Msdos tomó muchas ideas de este SO pero sus planteamientos eran mucho más modestos y solo se intento implementar unas burdas imitaciones de unas cuantas buenas ideas de Unix. El sistema en árbol de directorios la redirección de entrada salida y la estructura de un comando por ejemplo. Msdos nació como un juguetito comparado con Unix y luego la necesidad de mantener la compatibilidad con versiones anteriores ha condicionado fuertemente el crecimiento de Msdos primero y de Windows después. Por el contrario Unix ha mantenido la compatibilidad con versiones anteriores sin ningún problema. Tradicionalmente los SO Unix se han caracterizado por ser poco intuitivos de cara al usuario. Esto esta cambiando rápidamente porque Linux está ofreciendo cada vez entornos más intuitivos para su utilización. Esto es resultado del acercamiento progresivo de Linux hacia el usuario doméstico y ofimático.

De todas formas existe una barrera de tipo cultural que conviene tener presente. Muchos usuarios hoy en día saben lo que significa 'format a:'. a: es el nombre de una unidad de disquete en Msdos o en Windows pero en Linux no existen unidades lógicas. En Linux deberíamos hablar de sistemas de ficheros en lugar de unidades lógicas que es un concepto muy distinto.

Distribuciones de Linux

Linux es un núcleo de un SO pero para tener un SO operativo completo hay que acompañarlo de un montón de utilidades, dotarlo de una estructura de directorios, así como dotarlo de ficheros de configuración, y scripts para muy distintas tareas. Un script es un fichero que contiene instrucciones para el intérprete de comandos. Todas estas cosas juntas y bien organizadas son las cosas que debe proporcionar una distribución. Algunas distribuciones incluyen software comercial de su propiedad. Otras en cambio solo incorporan software GPL o similar. (software libre) Distribuciones libres son Slackware y Debian por ejemplo. Distribuciones propietarias son RedHat, SuSE, Caldera, Mandrake etc.

Cada distribución tiene sus propias características que la hacen más o menos adecuada para ciertos usos.

Uno de los aspectos más importantes de las distribuciones es su sistema de actualización de paquetes que permite actualizar el SO a cada nueva versión teniendo en cuenta las dependencias entre unos paquetes y otros.

Las aplicaciones en formato binario pueden funcionar en una distribución y en cambio no funcionar en otra. Sin embargo partiendo de los fuentes de una aplicación casi siempre basta con recompilar la aplicación para obtener un binario que funcione en esa distribución. Esto se debe al uso de librerías dinámicas que pueden variar de unas distribuciones a otras. Para obtener un binario que funcione en cualquier distribución se puede compilar una aplicación estáticamente lo cual hace que el ejecutable sea mucho mayor. Estos problemas están en vías de solución ya que las diferentes distribuciones están haciendo esfuerzos de estandarización para que cualquier aplicación pueda funcionar en cualquier distribución de Linux. Entre todas ellas hay algunas especialmente famosas por alguna u otra razón.

- Slackware es una distribución totalmente libre y muy sencilla en el sentido de que está poco elaborada. Resulta adecuada para cacharrear con ella. Fue creada por Patric Volkerding. Fue una de las primeras y tuvo su época de gran auge pero actualmente ha cedido protagonismo. No dispone de un buen sistema de actualización.
- Debian es una distribución totalmente libre desarrollada por un grupo muy numeroso de colaboradores en el más puro espíritu de Linux. Su calidad es extraordinaria. Se trata de una distribución muy seria que trabaja por el placer de hacer las cosas bien hechas sin presiones comerciales de ningún tipo pero que resulta más adecuada para usuarios con conocimientos previos ya que el grado de amistosidad en especial para los novatos deja bastante que desear si se compara con algunas distribuciones comerciales. Los usuarios típicos de Debian son aquellos que tienen como mínimo algún conocimiento técnico y que tampoco tienen reparos a la hora

de investigar un poco las cosas. El idioma sobre el cual trabaja y se coordina el grupo Debian es el Ingles y por ejemplo los bugs deben ser reportados en Ingles. La seguridad, y la detección y rápida corrección de errores son sus puntos fuertes. Soporta un enorme número de paquetes. Es una distribución tremendamente flexible. Su sistema de mantenimiento de paquetes 'dpkg' también es de lo mejor.

- RedHat es actualmente la distribución más ampliamente difundida aunque eso no significa que sea la mejor. Ofrece un entorno amigable que facilita la instalación. Incorpora software propietario de gran calidad. El sistema de paquetes 'RPM' es muy bueno y utilizado por un gran número de distribuciones.
- SuSE Es una distribución comercial alemana que ha tenido un crecimiento espectacular. Ofrece un entorno muy amigable que facilita mucho la instalación. Seguramente es la más fácil de instalar y de mantener. Capacidad de autodetección de Hardware. Incorpora abundante software propietario de gran calidad. En general se puede decir que es muy completa y muy recomendable para cualquiera que no tenga muchos conocimientos de Linux.
- Caldera Es una distribución comercial. Ofrece un entorno amigable que facilita la instalación. Incorpora software propietario de gran calidad.

La elección de una distribución viene condicionada por muchos factores. Hay muchas distribuciones actualmente y cada poco aparecen nuevas.

ALGUNOS CONCEPTOS BÁSICOS

Visión panorámica:

En su momento trataremos los temas abordados en este capítulo de forma completa. En este momento dado que partimos de cero se hace necesario al menos esbozar una serie de conceptos que son tan elementales que difícilmente se puede explicar algo sin hacer referencia a ellos.

Por otra parte estos conceptos están interrelacionados de tal forma que tampoco se puede abordar ninguno de ellos en detalle en este momento. Por ello nuestro primer objetivo es ofrecer una visión panorámica del SO. Por borrosa que resulte siempre será mejor esto que empezar directamente abordando en detalle los temas.

Usaremos muchas veces de forma indistinta Unix y Linux. No son cosas equivalentes pero en este curso consideraremos ambas cosas como equivalentes salvo que concretemos detalles específicos de Linux. Para nosotros Linux es un SO tipo Unix y precisamente nos centraremos en los aspectos más generales de estos sistemas operativos. Entre sus características más destacables está la de ser un SO multitarea y multiusuario. Un sistema multitarea es aquel que puede ejecutar varios procesos simultáneamente. Para ello se puede usar uno o más procesadores físicos. En el caso de un solo procesador lo que ocurre es que el tiempo del procesador se va repartiendo para atender los distintos procesos creando la ilusión de que todo ocurre simultáneamente. Un sistema multiusuario es aquel que está pensado para ser utilizado por varios usuarios simultáneamente. En la práctica un sistema multiusuario requiere capacidad multitarea. En el caso concreto de Linux es además un SO multiplataforma ya que puede funcionar en diferentes arquitecturas.

Los conceptos que mencionemos en este curso procuraremos ilustrarlos con ejemplos.

El comando 'echo' es un comando que vamos a usar mucho para practicar. Este comando se limita a visualizar en pantalla todo aquello que se le pasa como argumentos. Por eso nos resultará muy útil.

Lo primero que vamos a practicar y que no debe olvidar es el hecho de que en Unix los ficheros, comandos etc. deben indicarse exactamente respetando la diferencia entre mayúsculas y minúsculas.

```
$ echo hola
$ ECHO hola
```

La segunda línea habrá producido un mensaje de error porque no existe ningún comando llamado ECHO con mayúsculas.

Variables de entorno:

Empezaremos hablando de las variables sin profundizar ahora en ello. Una variable solo es un elemento que tiene un nombre y que es capaz de guardar un valor. Para definir una variable basta poner su nombre un igual y su valor. (Ojo no dejar espacios).

```
$ VAR33=valor_de_la_variable_VAR33
```

En Unix las variables del sistema se llaman variables de entorno. Para consultar el valor de una variable se utiliza el nombre de la variable precedido por '$'.

```
$ echo $VAR33
```

Para ver todas las variables y sus valores se utiliza el comando set.

Para hacer la variable exportable se usa el comando 'export'. Se puede hacer que una variable sea de solo lectura con el comando 'readonly'. Este comando sin parámetros mostrará todas las variables que son de solo lectura. Pruebe estos comandos:

```
$ set
$ readonly
$ export
```

Existen algunas variables predefinidas y hay tres variables importantes que mencionaremos en seguida y que son $PATH, $PS1 y $PS2.

Directorios:

Aquí también tenemos que hacer un pequeño avance sobre la estructura de directorios de Unix porque si no puede que no se entienda lo que vamos a hablar sobre la variable $PATH.

En Unix la estructura de directorios es en forma de árbol similar a la de Msdos. Dado que la estructura de directorios se organiza como las ramas de un árbol para localizar un punto cualquiera hay que utilizar cada uno de los directorios que conducen a ese punto desde el directorio raíz. Nosotros lo llamaremos camino y en Unix se le suele llamar path. Se empieza en el directorio raíz representado por '/' y se avanza por las ramas de ese árbol separando cada identificador por un nuevo carácter '/'. De esta forma '/usr/local/bin' indica un lugar concreto en el árbol de directorios. Quizás se pregunte porqué Unix usa '/' en lugar de '\' como en Msdos. Recuerde que Msdos fue posterior a Unix y que a Bill Gates le gusta inventar cosas totalmente nuevas y revolucionarias. Aunque el astuto lector ya se habrá dado cuenta, advierto a los despistados que estoy ironizando.

Aprovechando que estamos hablando de directorios mencionaremos que hay directorios con significado especial. Está el directorio raíz '/' que ya hemos mencionado. Está el directorio 'home' que es el punto donde el

sistema nos sitúa para trabajar cuando entramos en él. Recordemos que en Unix normalmente no disponemos de todo el sistema para nosotros solos. Luego podemos cambiar de directorio de trabajo. El directorio de trabajo actual se representa como directorio '.', El directorio anterior o directorio padre de este directorio se representa por '..'. Los comandos 'cd', 'mkdir' y 'pwd' sirven para cambiar de directorio actual, crear un directorio y averiguar en que directorio nos encontramos actualmente.

Estamos dando ideas en forma intuitiva. En realidad cuando digo " ... en que directorio nos encontramos actualmente ..." es una forma de expresarse. Normalmente el usuario no estará en un directorio sino sentado en una silla sentado delante de una pantalla, pero la shell mantiene en todo momento algún directorio abierto como directorio de trabajo actual.

Comandos

Un comando es generalmente un fichero ejecutable localizado en alguna parte de nuestro sistema. Existe una variable llamada $PATH que contiene una lista de caminos de búsqueda para los comandos todos ellos separados por ':'. Compruebe el valor de su variable $PATH.

```
$ echo $PATH
```

Si un comando no está localizado en ninguno de esos caminos deberá ser referenciado indicando el camino completo ya que de otra forma no podrá ser referenciado. Si quiere saber donde se encuentra un ejecutable use el comando which. Este comando buscará en los caminos contenidos en la variable $PATH hasta que encuentre el comando y entonces mostrará el camino completo que conduce al comando. Puede ocurrir que un comando se encuentre duplicado y que los dos sitios donde se encuentre figuren en el $PATH. En ese caso se ejecutará el comando que se encuentre en el primer camino referenciado en el $PATH. Igualmente el comando which solo mostrará el primer camino referenciado en el

$PATH. Estamos usando un comando llamado echo para mostrar valores. Vamos a localizarlo.

```
$ which echo
```

Echo esta situado en uno de los caminos contenidos en $PATH. En resumidas cuentas $PATH es una variable que usa el interprete de comandos para localizar los comando. En la lección siguiente hablaremos de 'man'. Es un comando que sirve para consultar el manual en línea de Linux. Pues bien existe una variable llamada $MANPATH que contiene la lista de los caminos donde el comando 'man' debe buscar las páginas del manual. La variable $MANPATH será usada por el comando 'man' y quizás algún otro. La variable $PATH será usada por la shell y quizás por algún otro comando como por ejemplo 'which' del cual acabamos de hablar.

Somos conscientes de que no estamos explicando gran cosa ahora, pero son conceptos muy elementales que vamos a utilizar antes de explicar en profundidad todo el sistema de ficheros de Unix.

Más adelante también hablaremos más detenidamente sobre las variables de entorno. Estamos dando unos pequeños esbozos sobre algunas cuestiones porque hay mucha interrelación de unos conceptos con otros y resulta imposible abordar nada en profundidad al principio.

Usuarios

Linux es un sistema operativo multiusuario y eso implica una filosofía de uso muy distinta a la del tradicional ordenador personal. Cuando un usuario va a usar un SO Tipo Unix lo primero que necesita hacer es identificarse para ser autorizado a abrir una sesión de trabajo. También es multitarea y por ello en el mismo instante varios procesos pueden estar funcionando y cada uno puede pertenecer a usuarios distintos. La información que se guarda en el disco duro también puede pertenecer a distintos usuarios y para evitar que todo ello provoque conflictos existen

unos atributos de usuario que se asocian a los ficheros a los directorios, a los procesos, etc. En función de esto hay cosas que estarán permitidas a ciertos usuarios y a otros no.

Los usuarios pueden estar organizados en diferentes grupos a fin de poder manejar permisos a nivel de grupo. Esto se hace para simplificar la administración del sistema.

Los usuarios y los grupos dentro del sistema se manejan como un número llamado UID y GID respectivamente. Los números en un ordenador se manejan con mucha más facilidad que un literal. Por eso el nombre de usuario y el nombre de grupo se guardarán solo en un fichero junto al número UID y GID asociado, pero para el sistema un usuario, un grupo, un proceso, un fichero, y muchas otras cosas se identifican por una clave numérica. Para un ordenador resulta más sencillo consultar si el proceso 456 perteneciente al usuario 105 tiene permiso de escritura en el fichero 48964 que consultar si el usuario 'pepito' perteneciente al grupo 'alumnos' tiene permiso de escritura en el fichero '/home/pepito/leccion005.txt'. Cuando el ordenador tiene que mostrar información en formato inteligible buscará la descripción correspondiente a cada clave numérica. Cada clave recibe un nombre como por ejemplo los UID, GID que ya hemos comentado otra sería por ejemplo el PID, para procesos y existen otras muchas más que iremos aprendiendo y que forman parte de la jerga de este SO.

Para ver los datos de identificación relativos a su usuario pruebe lo siguiente:

```
$ id
```

El comando 'id -un' o el comando 'whoami' muestran su nombre de usuario.

```
$ whoami
```

Existe un usuario especial que goza absolutamente de todos los privilegios y que se llama root. Su número de usuario es decir su UID es 0.

El Núcleo del sistema (kernel)

El núcleo del sistema llamado también kernel es el encargado de realizar la mayoría de funciones básicas del sistema y gestiona entre otras cosas la memoria, los ficheros, los usuarios, las comunicaciones, los procesos, etc. La gestión de estas cosas se hacen por medio de un limitado número de funciones que se denominan llamadas al sistema y que pueden ser usadas por los programas. Los procesos que usan una llamada al sistema cambian su modo de ejecución. Mientras están ejecutando la llamada del núcleo se dice que están en modo núcleo y cuando están ejecutando código que no pertenece al núcleo se dice que están en modo usuario. Son dos niveles de ejecución distintos ya que el modo núcleo es un modo privilegiado. Esto garantiza a nivel de hardware que ningún programa de usuario pueda acceder a recursos generales del sistema ni interactuar con otros procesos a no ser que use las llamadas del núcleo las cuales establecerán si tiene o no permiso para hacer ciertas cosas. Esto proporciona gran robustez de funcionamiento. Un programa mal diseñado no perjudicará jamás al sistema ni a otros procesos. Cada proceso tiene su propia zona de memoria y no puede acceder fuera de ella ni intencionadamente ni accidentalmente. Para que un programa pudiera tener un efecto destructivo en el sistema tendría que pertenecer a 'root' o pertenecer al propio núcleo del sistema y solo el administrador 'root' puede alterar el dicho núcleo. Si el ordenador aparece un buen día destrozado a martillazos también buscaran la forma de culpar a 'root' para no perder la costumbre.

Procesos

Un proceso a diferencia de un programa es algo vivo es decir algo que está funcionando. En un sistema multitarea como este, un programa puede dar lugar a varios procesos. A cada proceso le corresponderá con un

número de identificación llamado PID que le identifica totalmente. Además de esto se guarda la información de identificación del usuario propietario. Cuando un usuario ejecuta un comando se arranca el proceso correspondiente del cual generalmente permanecerá como propietario. Es decir el sistema no acepta órdenes anónimas. Siempre figurará un usuario para hacer cada cosa. Esto se indica con un número UID para identificar el usuario correspondiente. No siempre este UID se corresponde con el usuario que arrancó el proceso. Por ello existe además de un UID un identificador de usuario efectivo (EUID) que es el que realmente es tenido en cuenta por el sistema a la hora de conceder permiso para hacer ciertas cosas. El EUID de 'root' es 0. Ahora no importa como pero algunos comandos de forma bien controlada podrían convertirnos virtualmente en superusuarios haciendo que su EUID valga 0. Esto serviría por ejemplo para permitir hacer cosas especiales y muy concretas nada más que en condiciones normales solo 'root' podría hacer. No hemos mencionado intencionadamente algunas cosas (como por ejemplo el grupo de usuario), porque estamos simplificando mucho intencionadamente para intentar que en este momento solo capte una primera idea general.

Si se siente un poco mareado y confuso relájese y acostúmbrese a esta sensación. Los primeros pasos suelen ser los más complicados. Especialmente en Unix porque no fue diseñado como un sistema intuitivo.

EL MANUAL DEL SISTEMA

man(1), apropos(1)

Generalidades del manual

Man es el manual en línea de todos los sistemas operativos tipo Unix. Esta no es la lección más atractiva de este curso pero si que es una de las más necesarias. Uno de los objetivos del curso es alcanzar cierto nivel de autosuficiencia en Linux. Man no es la única fuente de información pero frecuentemente suele ser el primer sitio donde se suele mirar. Si usted hace una consulta en una lista de usuarios de internet de Linux sobre un tema que podría haber averiguado por si mismo consultando el manual alguien le recordará (en tono más o menos amable dependiendo del día que tenga) que esa información estaba en el manual.

Se pueden explicar muchas cosas sobre man pero es imprescindible que practique usted. Para ello de un rápido primer vistazo a la página man relativa al comando man.

Mientras no indiquemos lo contrario conviene que pruebe todos los comandos que se indican a modo de ejemplo. Si tiene posibilidad de usar dos sesiones le resultará más comodo practicar sobre la marcha. Teclee ahora el comando siguiente:

```
$ man man
```

Habrá visto un documento que explica el uso de man y que tiene una estructura característica de todas las páginas man.

Las páginas del manual utilizan un formateador de documentos llamado troff. Permite especificar el estilo de un documento. (Manera en que se ven los títulos, encabezados, párrafos, donde aparecen los números de página, etc. Debido a su flexibilidad troff resulta bastante difícil de usar. nroff y groff sirven para lo mismo pero no explicaremos sus diferencias. Pensamos que solo necesitar conocer su existencia. Si no instala estos programas no podrá usar man en su ordenador.

Otro ejemplo. Para obtener toda la información relativa al comando 'ls' haga lo siguiente:

```
$ man ls
```

Cuando no se conoce exactamente la palabra que se desea buscar se puede buscar por palabra clave. Para ello usaremos la opción -k, y la opción -f. Esta opción no está disponible en muchos sistemas debido a que hay que generar referencias y esto consume mucho espacio en disco. El administrador puede generar las referencias usando el comando 'catman'

Cuando no conocemos el nombre de un comando y tampoco encontramos nada por palabra clave pero sabemos que es lo que hace podemos usar el comando 'apropos'. Compruébelo tecleando los siguientes comandos:

```
$ apropos man
$ apropos apropos
$ man apropos
```

Si tiene su sistema Linux correctamente configurado para imprimir documentos en formato PostScrip podrá imprimir una página del manual haciendo

```
$ man -t man | lpr
```

Secciones del manual

Para indicar en un documento una referencia a una página del manual se suele indicar con el comando seguido de la sección entre paréntesis. El título de esta lección 'man(1)' es un ejemplo. Para consultar un comando en una sección concreta habría que teclear man <número_sección> <comando_o_función>. Veamos un par de ejemplos con printf(1) y printf(3).

```
$ man 1 printf
$ man 3 printf
```

Como habrá podido observar se refieren a cosas distintas. En el primer caso printf es un comando y en el segundo una función estándar de C con el mismo nombre. Si no se indica la sección la búsqueda se lleva a cabo en todas las secciones de manual disponibles según un orden predeterminado, y sólo se presenta la primera página encontrada, incluso si esa página se encuentra en varias secciones. Por el contrario usando la opción -a presentará, secuencialmente, todas las páginas disponibles en el manual. Compruebe esto con los siguientes ejemplos:

```
$ man printf
$ man -a printf
```

Consulte las secciones que están disponibles en su sistema en **man(1)**. Acabamos de indicar lo que significa man(1) así que el astuto lector ya debería estar mirando en la sección 1 del manual.

Conclusiones

Para finalizar también mencionaremos otra fuente importante de consulta en Linux que son los HOWTOs y los FAQs. Estos son documentos que explican como hacer algo o las preguntas más frecuentes relativas a un determinado tema. Este tipo de documentos monotemáticos son

frecuentes en Linux. Proceden de contribuciones desinteresadas de usuarios bien informados y están disponibles en distintos formatos (HTML. texto plano, etc). Suelen ser más didácticos que las páginas de man, pero man es la referencia obligada para cada comando y cada función importante. Nos evita tener que recordar todas las opciones de cada comando ya que frecuentemente cada comando tiene un amplio juego de opciones. Hay algunos documentos en español especialmente adecuados para novatos.

Hemos visto unas pocas opciones del comando man. Intente utilizar algunas otras opciones. Es importante que se familiarice con él. Las páginas man no están pensadas para enseñar, pero cada vez que tenga una duda de como se utiliza algo tendrá que recurrir a ellas como primera fuente de información.

Con esto pretendemos que abandone usted algunas de sus viejas costumbres. Antes usted pensaba de la siguiente manera:

Si funciona aceptablemente para que tocarlo.

Si no funciona apagamos y volvemos a encender

Si nada de esto sirve llamamos a un amigo.

Ahora ya sabe que también puede optar por una lectura del manual. Hágalo pero consulte de momento solo la información relativa a los números de sección, partes de un manual, y las opciones -a, -k, -t. Asumimos que no es necesario la lectura en profundidad porque habrá cosas que aun no puede interpretar. Tenga en cuenta que partimos de un nivel cero. Aún no sabe usted lo que significan algunas cosas. Por eso una vez finalizado el curso convendrá que vuelva a mirar esta página man relativa al comando man para sacarle más provecho.

LA SHELL

sh(1) ksh(1) csh(1) bash(1)

Introducción a la shell de Unix

Existen varias shells para Unix, Korn-Shell (ksh), Bourne-Shell (sh), C-Shell (csh), y muchas más. Existen algunas para propósitos especiales. Por ejemplo la remote-Shell (rsh) se utiliza para ejecutar comandos en un ordenador remoto. La Secure Shell (Ssh) se utiliza para establecer una conexión segura con un ordenador remoto.

La más utilizada en Linux es la Bourne-Again SHell (bash).

Nosotros de momento vamos a tratar principalmente la Bourne Shell que es la más estándar.

La Korn-Shell y la Bash son distintos superconjuntos distintos de la Bourne-Shell y por ello todo lo que se diga para la Burne-Shell será válido también para la Korn-Shell y para la Bash.

En Linux se suele usar la Bourne-Again SHell (bash), como sustituta de la Bourne-Shell (sh). Puntualmente también explicaremos alguna peculiaridad de la bash.

Para saber que shell está usando usted haga lo siguiente:

```
$ ps | grep $$
```

Si aparece -bash o -sh puede continuar sin problemas ya que está usando una shell adecuada para practicar lo que viene a continuación. En caso

contrario tecle el comando 'sh' o el comando 'bash' antes de continuar. Vuelva a realizar la comprobación anterior y verá que ahora esta usando otra shell.

En cualquier caso cualquier Shell es un programa normal y corriente, pero incorpora muchos de los conceptos más prácticos de Unix. No tiene nada de particular que algunos sistemas incorporen algunas Shells distintas.

Una Shell no es solo un intérprete de comandos. Una Shell es sobre todo un intérprete de un potente lenguaje.

Estructura de la línea de orden

¿Sabe usted lo que es un introductor? (en ingles lo llaman prompt). Pues es aquello que el interprete de comandos muestra para indicar que está esperando a que se introduzca una orden. En Unix el introductor de la orden de comandos no es siempre el mismo. Por defecto suele venir configurado distintos introductores para distintos interpretes de comandos y también se usa un introductor distinto para el usuario root. Sin embargo el introductor puede ser variado ya que es almacenado en una variable del sistema. En realidad la shell utiliza dos introductores distintos. Para ver cuales está utilizando ahora teclee lo siguiente:

```
$ echo "Introductor 1=$PS1"
$ echo "Introductor 2=$PS2"
```

Cuando aparece el primer introductor del sistema $PS1 indica que la shell está esperando la introducción de una orden. Las ordenes se terminan mediante. Si después de pulsar la shell no considera que el comando este completo quedará esperando más entrada mostrando el segundo introductor $PS2.

Si alguna vez no es capaz de terminar la introducción de un comando pruebe a abortar usando <Ctrl-C>

Una orden constará de un número variable de elementos separados por blancos, o por <tab>.

En una orden se pueden distinguir comandos, opciones, argumentos, meta-caracteres, comentarios, comandos internos...etc. Los blancos se usan para separar las opciones y argumentos presentes en una linea de ordenes y usados de esta forma (como separadores) una secuencia de blancos tiene el mismo efecto que uno solo. (Ojo en Unix las mayúsculas y minúsculas son significativas.)

A continuación se menciona de forma no exhaustiva los elementos del lenguaje shell-script. No intentamos que se aprenda todos estos elementos ni vamos a comentar ahora todos ellos. Bastará que se haga una idea de que tipos de elementos pueden intervenir.

- **Comandos:**

  Son ficheros ejecutables. Para que la shell localice el comando deberá estar en un subdirectorio que forme parte de la variable PATH o de lo contrario debe especificarse el camino completo.

- **Opciones:**

  Generalmente las opciones de un comando son letras precedidas de un signo '-'. En algunos comandos se pueden poner varias opciones como varias letras seguidas precedidas del signo '-'. Algunas opciones pueden venir como '--<opcion>' y concretamente en Linux es muy frecuente poder usar las opciones --help y --version en casi todos los comandos. Precisamente con la opción --help obtendremos generalmente la lista de opciones disponibles para un comando. También es frecuente el uso de opciones precedidas por '+'. En algunos comandos el orden de las opciones es muy significativo. No merece la pena hablar más de esto ahora porque vamos a usar un montón de comandos con un montón de opciones en este curso y tendremos ocasión de practicar mucho con ellas.

- **Meta-caracteres:**

Tienen un significado especial para la shell y son uno de los siguientes caracteres:

; & ( ) | > > <espacio> <tab>

- **Operadores de control:**

|| & && ; ;; ( ) | <nueva-linea>

- **argumentos:**
  Son literales tomados como parámetros de entrada para algún comando.
- **comentarios:**
  Todo lo que sigue al carácter '#' hasta <nueva-línea> será un comentario.
- **Palabras reservadas:**

Son palabras reservadas para implementar el lenguaje shell-script. Son palabras reservadas:

case, do, done, elif, else, esac, fi, for, function, if, in, select, then, until, while, time.

- **Comandos internos:**

Comandos que están implementados dentro de la propia shell. No necesitan PATH. Ejemplos de comandos internos son:

cd, exec, arg, eval, exit,...

Para una referencia exacta y completa debería acudir a la página man bash(1) aunque eso resultaría excesivo en este momento.

Vamos a dar una serie de ejemplos para que los practique en su ordenador. No es necesario que introduzca los comentarios.

```
$ # Orden con un único comando sin opciones ni argumentos
$ ls

$ # Orden sin ociones pero con tres argumentos
$ ls . / ..

$ # Orden con un comando y una opción
$ ls -l

$ # Orden con un comando y tres opciones indicada de varias formas
$ # distintas pero equivalentes entre si.
$ ls -trl
$ ls -rtl
$ ls -ltr
$ ls -l -t -r
$ ls     -l    -t     -r
$ ls -lt -r
$ ls -l -tr

$ # Opciones que empiezan con '--'
$ ls --help
$ ls --version
$ ls --color=auto

$ # Ejemplo de opciones y argumentos sensibles al orden.
$ date -d now -R
$ date -d -R now

$ # ejemplo de opcion que empieza por '+'
$ date +'%a %b %e %H:%M:%S %Z %Y'
```

Expansión de la linea de orden:

Existe un detalle muy importante en el funcionamiento de la shell. Una cosa es lo que nosotros escribimos y otra lo que la shell ordena que se ejecute. Antes de que la shell ejecute comando alguno expande la línea de ordenes. Es decir esa línea se transforma en otra línea más larga. La orden resultante puede ser muy larga y tener muchos argumentos. Por ejemplo un '*' será sustituido por la lista de ficheros que se encuentren en el directorio actual. Quizás alguien encuentre que existe un parecido con el uso de '*.*' en Msdos por ejemplo pero el parecido es muy superficial y engañoso. Cuando en Unix hacemos 'ls *' el intérprete de comandos expande el asterisco y el comando ls recibe una lista de ficheros que tiene que listar. En Msdos cuando hacemos 'dir *.*' el interprete de comandos no expande nada. El comando dir recibe como argumento no la lista de ficheros que tiene que listar sino un '*.*' y será el propio comando dir quien tenga que expandir el argumento para obtener la lista de ficheros que tiene que listar. Por lo tanto la expansión en Unix está centralizada en el interprete de comandos el cual permite hacer expansiones mucho más potentes que en Msdos. Un comando no es más que un ejecutable y los ejecutables generalmente admiten argumentos. En Unix los comandos toman los argumentos que les pasa la shell después de la expansión.

El '*' se expande en base a los nombres de ficheros presentes en nuestro directorio actual, sustituyéndose por una cadena de caracteres cualquiera que no empiece por un punto.

Vamos a realizar una práctica completa. Algunos de los comandos que vamos a usar como 'cd', 'mkdir', 'touch' y otros son comandos que no explicamos en este momento pero que servirán para situarnos en un directorio de trabajo y crear algunos ficheros para la práctica.

```
$ comando ...

salida del
comando ...
```

Ahora introduzca los comandos que indicamos a continuación y compruebe la salida obtenida.

```
$ cd /tmp
$ mkdir pruebas
$ cd pruebas
$ # Ya hemos creado un directorio de pruebas y ya estamos
dentro de él.
$ # Para comprobarlo hacemos
$ pwd
/tmp/pruebas
$ # Ahora creamos unos ficheros para practicar
$ touch kk1 kk2 kkkk kk.txt kk.doc j2.txt .kk
$ echo *
kk1 kk2 kkkk kk.txt kk.doc j2.txt
$ echo k*
kk1 kk2 kkkk kk.txt kk.dox
$ echo *xt
kk.txt j2.txt
$ echo *.

$ echo .*
.kk
$ echo *.*
kk.txt kk.doc j2.txt
```

Fíjese que los resultados dependen de los ficheros existentes en el directorio actual. Los mismos comandos realizados desde otro directorio distinto darían otro resultado.

Mantenga la sesión de la práctica en este punto porque continuaremos haciendo algunas prácticas más desde este mismo punto.

El '?' se expande como un único carácter y tampoco expande un punto en el comienzo del nombre del fichero. Introduzca ahora los siguientes comandos y compruebe la salida obtenida.

```
$ echo ???
kk1 kk2
$ echo kk?
kk1 kk2
```

A continuación no teclee nada ya que el resultado es solo una hipótesis. Supongamos que obtenemos un error de este tipo.

```
$ ls *
ksh: /bin/ls: arg list too long
```

Esto significa que el '*' se expande en un número demasiado grande de ficheros y eso resulta un problema para el interprete de comandos. La forma de obtener la lista de ficheros sería haciendo:

```
$ ls
```

O también.

```
$ ls .
```

Si quisiéramos averiguar el número de ficheros podríamos contarlos con 'wc'. Aquí estamos usando un '|' que aun no hemos explicado pero que lo explicaremos en el próximo capítulo de redirección de entrada salida.

```
$ ls | wc -l
```

Peculiaridades de expansión en la bash de Linux

Solo vamos a mencionar unas cuantas y remitimos al lector a la página del manual de bash para más información.

- Expansión de corchetes.

  Este es un tipo de expansión que no tiene para nada en cuenta los ficheros existentes en el directorio actual.

```
$ echo a{d,c,b}e
ade ace abe
```

- Expansión de la tilde.

  Esta es una característica especial de la shell de Linux que resulta realmente útil. Expande la tilde como directorio home.

```
$ echo ~
$ echo ~root
$ echo ~root/*
```

- Arithmetic Expansion.

  Esto permite expandir expresiones $((expression))

```
$ echo $(((4+11)/3))
5
```

En bash(1) en el apartado de EXPANSION se mencionan más cosas.

Redirección de entrada salida:

Normalmente los proceso utilizan para entrada y salida de datos unos dispositivos estándar. Son entrada estándar, salida estándar y salida estándar de errores. Generalmente se utiliza como entrada estándar la entrada de teclado y como salida estándar y salida estándar de errores la pantalla. La salida estándar se usa como salida normal de datos y la salida estándar de errores se usa para sacar mensajes que indican algún error, aunque también se usa simplemente como flujo alternativo en caso de que no resulte deseable su mezcla con otros datos que deben salir por salida estándar.

Se puede alterar flujo de datos que va desde un dispositivo estándar a un programa o viceversa puede ser redirigido a otro dispositivo, o a otro programa, o fichero, etc.

Desde la shell se puede hacer esto de varias formas.

- >

  Redirige la salida estándar a un fichero o dispositivo.

- <

  Redirige la entrada estándar tomándola desde un fichero.

- |

  Comunica dos procesos por medio de entrada salida. Ojo no confundir con Msdos. En Unix los procesos comunican directamente pasándose los datos directamente sin crear ningún fichero temporal. El proceso que lee quedara en espera mientras el el proceso que escribe mantenga abierto el dispositivo de salida estándar incluso si momentáneamente no se produce salida. Cuando el proceso escritor termina cierra todos

sus ficheros y el proceso lector acusará la condición como un fin de datos de entrada.

- >>

Redirige la salida estándar a un fichero sin sobrescribirlo. En lugar de ello añade al final del mismo.

- <<FIN

Redirige entrada estándar desde la propia línea de órdenes. (En lugar de FIN se puede usar cualquier literal).

- 2>

Redirige la salida estándar de errores a un fichero o dispositivo.

- 2>&1

Redirige la salida estandard de errores donde esta redirigido la salida estándar. (0=entrada estándar, 1=salida estándar, 2=salida de errores estándar)

Ejemplos

La orden siguiente no produce ningún resultado visible porque la salida estándar se redirige al dispositivo /dev/null. Este dispositivo es como un pozo sin fondo. A diferencia de una papelera de Windows no se puede recuperar luego nada.

```
$ date > /dev/null
```

Ahora un ejemplo curioso. El comando 'time' sirve para medir consumos de tiempos de otros comandos y para evitar mezclas de datos en salida

estándar se decidió que la salida normal de time fuera la salida estándar de errores.

```
$ time whoami > /dev/null
```

Podemos ver el consumo de tiempo del comando 'whoami' porque este sale por la salida estándar de errores. Es un ejemplo de utilización de la salida estándar de errores para sacar información que no tiene nada que ver con ningún error. Sin embargo time es un comando interno y por ello lo siguiente no funcionaría como usted piensa.

```
$ time whoami 2> /dev/null
```

En este caso la redirección afecta solo al comando whoami.

Los comandos internos son parte de la shell y para redirigir su salida habría que redirigir la salida completa de la shell. Dejaremos esto para un capítulo posterior.

Antes de continuar vamos a asegurarnos que estamos en un sitio seguro para trabajar.

```
$ cd /tmp
$ mkdir pruebas > /dev/null
$ cd pruebas
$ # Para comprobar que estamos en '/tmp/pruebas' hacemos
$ pwd
/tmp/pruebas
```

El contenido de '/tmp' suele ser vaciado cuando re-arranca la máquina o quizás en algún otro momento. Contiene información temporal. Hemos usado /dev/null para ignorar la salida de errores del comando mkdir. Si ya existía '/tmp/pruebas' mkdir habría dado un error pero nosotros lo

ignoramos porque solo nos interesa que lo cree en caso de que no exista y en caso contrario da igual. El dispositivo '/dev/null' también resulta útil para simular una entrada nula de datos. Por ejemplo para crear un fichero vació. Si ya estamos situados en '/tmp/pruebas' pruebe lo siguiente:

```
$ cat < /dev/null > kk
$ ls -l kk
```

El mismo efecto podríamos conseguir usando.

```
$ > kk
$ ls -l kk
```

Esto se puede utilizar para vaciar ficheros respetando los permisos originales.

Vamos mirar el contenido del directorio raíz

```
$ ls -l
```

Ahora queremos repetir el comando pero guardando el resultado en el fichero kk

```
$ ls -l / > kk
```

No vemos nada porque toda la salida va a parar al fichero. Para visualizar el contenido de un fichero usaremos el comando 'cat'.

```
$ # Mostrar el contenido del fichero kk.
$ cat kk
```

```
$ # Equivale al anterior.
$ cat < kk

$ # Parecido a los dos anteriores. /dev/tty es un dispositivo que
$ # se identifica como nuestro terminal.
$ cat < kk > /dev/tty
```

También podemos visualizar la salida de un comando a la vez que guardamos la salida en un fichero. El comando tee actúa como un bifurcación.

```
$ ls -l / | tee kk1
$ cat kk1
```

Recuerde que lo que sigue a un '|' ha de ser siempre un ejecutable.

```
$ touch kk
$ date | kk
```

Esto habrá dado un error. Probemos ahora a redirigir la salida estándar de errores a un fichero y la salida estándar a otro.

```
S touch kk
$ date | kk 2> errores > salida
$ cat errores
$ cat salida
```

Existe un comando 'yes' que sirve para generar continuamente respuestas afirmativas en forma de caracteres 'y' y finales de línea. Este comando está pensado para ser usado con '|' y proporcionar entradas afirmativas a un comando. Usaremos el comando 'head' que sirve para sacar por salida

estándar solo una primera parte de los datos de la entrada. La cantidad datos puede especificarse en número de líneas en bytes, etc.. Nosotros vamos a utilizar bytes. La salida de 'yes' la entregaremos a head para que tome los 1000 primeros bytes que pasamos al programa 'wc' para que cuente líneas palabras y caracteres.

```
$ yes | head --bytes=1000 | wc
500 500 1000
Tubería rota
```

El comando 'yes' podría funcionar eternamente. Su salida redirigida por las buenas a un fichero llenaría el disco (cosa nada deseable por cierto) y acabaría dando un error. En lugar de esto lo hemos redirigido a un programa que aceptará solo los 1000 primeros caracteres y luego cerrará su entrada provocando un error de Tubería rota.

La shell también permite introducir datos en una forma especial que se llama documento-aquí. Para variar un poco usaremos ahora el comando 'sort' que sirve para ordenar. Observe que en este ejemplo un fin de línea no termina el comando. Por ello aparecerá el introductor secundario $PS2 que nosotros indicamos con un '>' en amarillo.

```
$ # Para ordenar unas lineas que introducimos en la
propia linea de
$ # ordenes usamos el operador '<<' seguido de una
clave de
$ # finalización, de la entrada.
$ sort <<FIN
> aaaaa
> cccccc
> zzzzz
> bbbb
> yyyy
> FIN

aaaaa
bbbb
cccccc
```

```
yyyy
zzzzz
```

Operador grave:

Cuando colocamos algo dentro de las comillas graves "`" la shell lo interpretara como una cadena cuyo valor es sustituido por el resultado de ese comando que produce ese comando en salida estándar.

```
$ echo date produce --- `date` ---

date produce --- lun ene 10 20:59:12 CET 2000 ---
```

El resultado es solo un ejemplo y no puede coincidir exactamente con el resultado obtenido por usted ya que debería poner la fecha correcta en su sistema.

Quizas el ejemplo que acabamos de usar no le de una idea exacta de la potencia del operador grave. Recuerde que esto ocurre durante la expansión de la línea de órdenes antes de ejecutarse el comando echo. Por ello podemos hacer cosas como las siguientes.

```
$ # Ejecutamos date y guardamos el resultado en un fichero cuyo
$ # nombre construimos en base a nuestro nombre de usuario y al
$ # año de la fecha actual.
$ date > fichero-`whoami`-`date +%Y`
$ ls -l fichero-*
$ cat fichero-*
```

Hemos usado un comando para construir el nombre del fichero de salida. No mostramos los resultados pero confiamos en que ya los ha comprobado.

Caracteres de escape:

Dado que la Shell interpreta caracteres. Blancos como separadores. Asteriscos e interrogaciones como comodines. Operador grave. Comillas dobles, Comillas normales, $, etc... Se plantea el problema de que hacer cuando queremos usar estos caracteres como tales sin que sean interpretados por la shell. Existen varias formas de escapar caracteres para que la shell no los expanda, o interprete.

La shell no el único programa con capacidad de expandir e interpretar caracteres especiales. Por ejemplo find, egrep, sed, y otros también interpretan ciertos caracteres que además pueden coincidir con algunos de los que interpreta la shell. Para usar caracteres especiales que pueden interpretarse por la shell habrá que escaparlos siempre que deseemos que lleguen al comando.

Vamos a ver tres formas distintas de escapar caracteres:

- Cuando usamos las dobles comillas, los comodines asterisco, interrogación y caracteres en blanco no se interpretan, aunque si se interpretan '$', comillas graves '`', y otros.
- Cuando usamos comillas simples se respeta todo el contenido de la cadena sin expandir nada.
- Cuando usamos el carácter '\' escapamos el carácter que va inmediatamente después. Cuando es un carácter que tiene un significado especial para la shell, dejará de tener ese significado especial.

Vamos a ver todo esto con unos ejemplos:

```
$ # La orden siguiente muestra el mismo literal que el entre comillas.
$ echo "* * ? *"

$ # En este caso la shell interpreta `pwd` y $PATH, en cambio '*' y
$ # '?' no se interpretan.
```

```
$ echo "* ? `pwd` $PATH"

$ # En este caso se conserva todo el literal sin interpretar.
$ echo '* ? `pwd` $PATH'

$ # A continuación la orden mostrará dos comillas dobles.
$ echo \"\"

$ # El caracter <nueva-linea> también puede ser escapado y en este
$ # caso puede servir para introducir comandos muy largos de forma
$ # más legible. Para hacer esta prueba deberá pulsar la tecla de
$ # <nueva-linea> justo a continuación del caratcter '\'
$ ls -l / | \
$ head -n 10 | \
$ tee /tmp/resultado | \
$ wc
```

Ejercicio:

Las siguientes órdenes producen resultados cuya explicación no resulta trivial. Esto es debido a que permitimos con la opción -e que el comando 'echo' interprete caracteres. Por lo tanto lo que ocurre es que primero el intérprete de comandos interpreta los caracteres de la línea de órdenes y luego el comando echo interpreta los caracteres recibidos. Consulte la página man de echo(1) y busque la opción -e. Ejecute los comandos que le proponemos a continuación e intente explicar los resultados. Puede que necesite papel y lápiz.

```
$ echo -e \n
n
$ echo -e \\n
```

```
$ echo -e \\\n


$ echo -e \\\\n
\n $ echo -e \\\\\n
\n $ echo -e \\\\\\n
\
$ echo -e \\\\\\\n
\
$ echo -e \\\\\\\\n
\\n
```

Resumen

Hemos visto muchas cosas que nos permiten hacernos idea de la potencia de la Bourne-Shell de Unix. La bash de Linux es un súper conjunto. Es decir es compatible pero más potente.

Queremos resaltar que para las personas que han conseguido terminar este capítulo ya se ha conseguido algo importante. Personas acostumbradas al usos de Msdos cometen barbaridades que luego ni siquiera son capaces de explicar. La clave muchas veces está en la expansión de la línea de órdenes.

En realidad este tema no está agotado ni mucho menos y lo completaremos en capítulos posteriores aunque intercalaremos material más ligero porque no deseamos hacerle sufrir demasiado. Solo lo justo y por ello creo que es el momento de confesar que existía un truco para realizar más fácilmente el ejercicio anterior aunque no lo mencionamos antes porque habría resultado demasiado fácil y cómodo. Se trata de activar la traza de la shell con 'set -x'. De esta forma podemos ver el resultado de la expansión de línea de órdenes precedida por el signo '+'.

```
$ set -x
```

```
+ set -x
$ echo -e \\\\\\\n
+ echo -e '\\\\n'
\\n
$ set +x
+ set +x
```

Puede utilizar este truco cuando no sepa que está ocurriendo con la expansión de una línea de órdenes que no funciona como usted pensaba.

INTRODUCCIÓN A LOS PROCESOS

Que es un proceso

A estas alturas más de uno empezará a preguntarse si realmente es necesario conocer todos estos detalles para un simple uso de un SO. Este curso esta orientado al uso de Linux desde la consola y resulta necesario conocer estas cosas incluso para un usuario normalito. No vamos a describir detalladamente como están implementados los procesos en Linux.

Quizás para un curso de programación avanzada si fuera necesario, pero lo que nosotros vamos a describir es únicamente los detalles más importantes desde un punto de vista didáctico y práctico para un usuario normal.

En un sistema multitarea todo funciona con procesos así que conocer unos cuantos principios básicos sobre procesos le resultará de la máxima utilidad. En un sistema monotarea se usa frecuentemente el término programa indistintamente para hablar de un programa en papel, en cdrom, en disco duro o en funcionamiento. En un sistema multitarea no se debe usar el término programa para hablar de la ejecución del mismo. En su lugar hablaremos de proceso indicando con ello que esta arrancado y funcionando. Un programa puede dar lugar a varios procesos. Por ejemplo en un mismo instante varios usuarios pueden estar usando un mismo editor. Un proceso puede estar detenido pero a diferencia de un programa existe una información de estado asociada al proceso. Un programa es algo totalmente muerto. Un proceso detenido es más bien como si estuviera dormido.

Los procesos tienen organizada la memoria de una forma especial muy eficiente. Por ejemplo la párte de código del proceso es una parte de solo

lectura y puede ser compartida por varios procesos a la vez. Imaginemos que hay varios usuarios en el sistema usando un mismo editor. En ese caso sería un desperdicio tener la misma información de código de ese programa repetida varias veces en memoria y ocupando un recurso tan valioso como es la memoria RAM.

En el caso de tener varios programas distintos en ejecución también se suele dar el caso de que contengan partes comunes de código pertenecientes a librerías que contienen gran cantidad de funciones de propósito general. Para hacer un uso eficiente de estas librerías existen librerías dinámicas de uso compartido. En Linux el uso de estas librerías no está organizado de una forma unificada para todas las distribuciones por lo cual los ejecutables binarios pueden ser incompatibles entre distintas distribuciones. Este problema se puede solucionar partiendo de los fuentes y recompilando la aplicación en nuestro sistema. Por esta razón un binario de RedHat, o Suse puede no funcionar en Slackware o en Debian.

PID y PPID

A cada proceso le corresponderá un número PID que le identifica totalmente. Es decir en un mismo momento es imposible que existan dos procesos con el mismo PID. Lo mismo que todos los procesos tienen un atributo PID que es el número de proceso que lo identifica en el sistema también existe un atributo llamado PPID. Este número se corresponde con el número PID del proceso padre. Todos los procesos deben de tener un proceso que figure como padre pero entonces que ocurre si un padre muere antes que alguno de sus hijos ? En estos casos el proceso 'init' del cual hablaremos en seguida adoptará a estos procesos para que no queden huérfanos.

El proceso init

Cuando arranca el sistema se desencadena una secuencia de procesos que a grandes rasgos es la siguiente. Primero se carga el núcleo de Linux

(Kernel) de una forma totalmente especial y distinta a otros procesos. Dependiendo de los sistemas puede existir un proceso con PID=0 planificador, o swapper. En Linux y en casi todos los sistemas tipo Unix seguirá un proceso llamado 'init'. El proceso init tiene PID = 1. Lee un fichero llamado inittab donde se relacionan una serie de procesos que deben arrancarse para permanecer activos todo el rato (demonios). Algunos de ellos están definidos para que en caso de morir sean arrancados de nuevo inmediatamente garantizando la existencia de ese servicio de forma permanente.

Es decir 'init' es un proceso que va a generar nuevos procesos pero esta no es una cualidad especial. Es muy frecuente que un proceso cualquiera genere nuevos procesos y cuando esto ocurre se dice que genera procesos hijos.

Este no es un curso de administración pero diremos que a init se le puede indicar que arranque el sistema de diferentes formas, por ejemplo en modo monousuario para mantenimiento. Este es un capítulo en el cual pueden surgir muchas preguntas retorcidas tales como, que pasa si matamos a init, o quien es el padre de init, pero no nos interesa responder a esto ya que init es un proceso muy especial y su padre aún más. En cada sistema de tipo Unix las respuestas a cosas como estas pueden variar mucho porque a ese nivel la implementaciones varían mucho. Ningún programa normal necesitará usar ese tipo de información. Quedan muchos detalles interesantes relativos a temas de administración. Los curiosos siempre tienen el recurso de mirar la página man de init(8) y de inittab(5) pero nosotros no insistiremos más en este tema. (Que alivio ¿verdad?)

UID y EUID

Los procesos tienen un EUID (Efectiv User Identif), y un UID normalmente ambos coinciden. El UID es el identificador de usuario real que coincide con el identificador del usuario que arrancó el proceso. El EUID es el identificador de usuario efectivo y se llama así porque es el

identificador que se tiene en cuenta a la hora de considerar los permisos que luego explicaremos.

El UID es uno de los atributos de un proceso que indica por decirlo de alguna manera quien es el propietario actual de ese proceso y en función de ello podrá hacer ciertas cosas. Por ejemplo si un usuario normal intentara eliminar un proceso del cual no es propietario el sistema no lo permitirá mostrando un mensaje de error en el que advierta que usted no es el propietario de ese proceso y por tanto no está autorizado a hacer esa operación. Por el contrario el usuario root puede hacer lo que quiera con cualquier proceso ya que el sistema no comprueba jamás si root tiene permisos o no para hacer algo. root siempre tiene permisos para todo. Esto es cierto a nivel de llamadas del sistema pero nada impide implementar un comando que haga comprobaciones de permisos incluso con root. Algunos comandos en Linux tienen opciones que permiten hacer estas cosas y solicitar confirmación en caso de detectar una operación peligrosa o infrecuente.

El UID también es un atributo presente en otros elementos del sistema. Por ejemplo los ficheros y directorios del sistema tienen este atributo. De esta forma cuando un proceso intenta efectuar una operación sobre un fichero. El kernel comprobará si el EUID del proceso coincide con el UID del fichero. Por ejemplo si se establece que determinado fichero solo puede ser leido por su propietario el kernel denegará todo intento de lectura a un proceso que no tenga un EUID igual al UID del fichero salvo que se trate del todo poderoso root.

Aunque estamos adelantando cosas sobre el sistema de ficheros vamos a continuar con un ejemplo. Comprobaremos cual es el UID de su directorio home.

```
$ # Cambiamos el directorio actual a home
$ cd
$ # Comprobamos a quien pertenece 'uid' y 'guid'
$ ls -ld .
$ # Ahora obtenemos el 'uid' y el 'gid' con sus
```

```
valores numéricos.
$ ls -lnd .
```

Si su directorio home está configurado de forma lógica deberá pertenecerle a usted. Si esto no es así reclame enérgicamente a su administrador, pero si el administrador resulta ser usted sea más indulgente y limítese a corregirlo y no confiese su error a nadie. En realidad casi todo lo que se encuentre dentro de su directorio home debería pertenecerle a usted. Usted debe ser el propietario de su directorio home porque de otra forma y dependiendo de los permisos asociados a este directorio los procesos que usted arranque se verían o bien incapaces de trabajar con él, o lo que es peor cualquiera podría hacer cualquier cosa con él. Como es lógico hemos mencionado de pasada el tema de permisos de directorios para ilustrar un poco la utilidad del uid pero esto se verá en detalle en el capítulo dedicado al sistema de ficheros de Linux.

Algunos ficheros ejecutables poseen un bit de permisos que hace que cambie el EUID del proceso que lo ejecute convirtiéndose en el UID del propietario del fichero ejecutado. Suena complicado pero no lo es. Es decir imaginemos que usted ejecuta un comando propiedad de root que tiene este bit. Pues bien en ese momento el EUID de su proceso pasaría a ser el de root. Gracias a esto un proceso puede tomar temporalmente la identidad de otro usuario. Por ejemplo puede tomar temporalmente la identidad de root para adquirir privilegios de superusuario y así acceder por ejemplo a ficheros del sistema propiedad de root.

El sistema recibe continuamente peticiones de los procesos y el EUID del proceso determinará que el kernel le conceda permiso para efectuar la operación deseada o no.

Muchas veces sorprende que en Linux apenas se conozcan unos pocos casos de virus, mientras que en otros sistemas parecen estar a la orden del día. Es perfectamente posible realizar un virus que infecte un sistema Linux pero de una forma o de otra el administrador tendría que darle los

privilegios que le convierten en peligroso. Por eso no es fácil que estos virus lleguen muy lejos.

Como se crea un nuevo proceso

El núcleo del sistema llamado también kernel es el encargado de realizar la mayoría de funciones básicas que gestiona entre otras cosas los procesos. La gestión de estas cosas se hacen por medio de un limitado número de funciones que se denominan llamadas al sistema. Estas llamadas al sistema están implementadas en lenguaje C y hablaremos ahora un poco sobre un par de ellas llamadas fork() y exec(). Si logramos que tenga una vaga idea de como funcionan estas dos importantísimas funciones facilitaremos la compresión de muchas otras cosas más adelante.

La llamada al sistema exec()

Cuando hablamos de proceso debe usted pensar solamente en algo que se está ejecutando y que está vivo. Un proceso puede evolucionar y cambiar totalmente desde que arranca hasta que muere. Lo único que no cambia en un proceso desde que nace hasta que se muere es su identificador de proceso PID. Una de las cosas que puede hacer un proceso es cambiar por completo su código. Por ejemplo un proceso encargado de procesar la entrada y salida de un terminal (getty) puede transformarse en un proceso de autentificación de usuario y password (login) y este a su vez puede transformarse en un interprete de comandos (bash). Si la llamada exec() falla retornará un -1. Esto no es curso de programación así que nos da igual el valor retornado pero lo que si nos interesa es saber que cuando esta llamada tiene éxito no se produce jamás un retorno. En realidad no tendría sentido retornar a ningún lado. Cuando un proceso termina simplemente deja de existir. En una palabra muere y ya está. La llamada exec() mantiene el mismo PID y PPID pero transforma totalmente el código de un proceso en otro que cargará desde un archivo ejecutable.

La llamada al sistema fork()

La forma en que un proceso arranca a otro es mediante una llamada al sistema con la función fork(). Lo normal es que el proceso hijo ejecute luego una llamada al sistema exec(). fork() duplica un proceso generando dos procesos casi idénticos. En realidad solo se diferenciaran en los valores PID y PPID. Un proceso puede pasar al proceso hijo una serie de variables pero un hijo no puede pasar nada a su padre a través de variables. Además fork() retorna un valor numérico que será -1 en caso de fallo, pero si tiene éxito se habrá producido la duplicación de procesos y retornará un valor distinto para el proceso hijo que para el proceso padre. Al proceso hijo le retornará el valor 0 y al proceso padre le retornará el PID del proceso hijo. Después de hacer fork() se pueden hacer varias cosas pero lo primero que se utiliza después del fork es una pregunta sobre el valor retornado por fork() para así saber si ese proceso es el padre o el hijo ya que cada uno de ellos normalmente deberá hacer cosas distintas. Es decir la pregunta sería del tipo si soy el padre hago esto y si soy el hijo hago esto otro. Con frecuencia aunque no siempre el hijo hace un exec() para trasformarse completamente y con frecuencia aunque no siempre el padre decide esperar a que termine el hijo.

También normalmente aunque no siempre esta parte de la lección es necesario releerla más de una vez.

Estamos dando pequeños detalles de programación porque en estas dos llamadas del sistema son muy significativas. Su funcionamiento resulta chocante y su comprensión permite explicar un montón de cosas.

MÁS SOBRE PROCESOS Y SEÑALES

Las formas de comunicación entre procesos

Los procesos no tienen una facilidad de acceso indiscriminada a otros procesos. El hecho de que un proceso pueda influir de alguna manera en otro es algo que tiene que estar perfectamente controlado por motivos de seguridad. Comentaremos solo muy por encima las diferentes formas de comunicación entre procesos.

1. A través de variables de entorno:

   Solo es posible de padres a hijos.

2. Mediante una señal:

   Solo indica que algo ha ocurrido y solo lleva como información de un número de señal.

3. Mediante entrada salida:

   Es la forma más corriente a nivel de shell. Ya hemos comentado el operador pipe '|' que conecta dos procesos.

4. Mediante técnicas IPC u otras:

   Semáforos, Memoria compartida, Colas de mensajes.

5. Mediante sockets:

   Este sistema tiene la peculiaridad de que permite comunicar procesos que estén funcionando en máquinas distintas.

No profundizamos sobre esto porque ahora no estamos interesados en la programación. Más adelante si comentaremos bastante sobre variables y entrada salida porque daremos nociones de programación en shell-script. También daremos a continuación unos pequeños detalles que tienen que ver con el arranque del sistema porque también nos resultará util para comprender como funcionan las cosas.

Secuencia de arranque en una sesión de consola

Para consultar la dependencia jerárquica entre unos procesos y otros existe en Linux el utilísimo comando pstree. No es esencial y quizás no lo tenga usted instalado pero resulta muy práctico. Si dispone de él pruebe los comandos 'pstree', y 'pstree -p'. Nosotros vamos a mostrar el resultado de ambos comandos en el sistema que estamos usando en este momento para que en cualquier caso pueda apreciar el resultado pero le recomendamos que lo instale si no dispone de él ya que resulta muy práctico. También puede usar como sustituto de 'pstree' el comando 'ps axf' pero no espere un resultado tan bonito.

```
$ pstree

init-+-apache---7*[apache]
     |-atd
     |-bash---pstree
     |-2*[bash]
     |-bash---vi
     |-bash---xinit-+-XF86_S3V
     |              `-mwm-+-.xinitrc---xterm---bash
     |                    `-.xinitrc---xclock
     |-cron
     |-getty
     |-gpm
     |-inetd
     |-kflushd
     |-klogd
     |-kpiod
     |-kswapd
     |-kupdate
     |-lpd
     |-portmap
```

```
      |-postmaster
      |-sendmail
      |-sshd
      |-syslogd
      `-xfs
```

Este formato nos da un árbol de procesos abreviado en el que procesos con el mismo nombre y que tengan un mismo padre aparecerán de forma abreviada. En el ejemplo anterior aparece '2*[bash]' indicando que hay dos procesos bash como hijos de init, y también hay algunos proceso apache arrancados. El que existan muchos procesos arrancados no indica necesariamente un alto consumo de CPU. Puede que estén todos ellos haciendo el vago. Bueno en el caso de apache quizás estén haciendo el indio. (Esto último es una broma que no he podido evitar).

```
$ pstree -p

init(1)-+-apache(204)-+-apache(216)
        |             |-apache(217)
        |             |-apache(218)
        |             |-apache(219)
        |             |-apache(220)
        |             |-apache(1680)
        |             `-apache(1682)
        |-atd(196)
        |-bash(210)---pstree(1779)
        |-bash(211)
        |-bash(212)---vi(1695)
        |-bash(215)---xinit(1639)-+-XF86_S3V(1644)
        |                         `-mwm(1647)-
+-.xinitrc(1652)---xterm(1660)---bash(1661)
        |
`-.xinitrc(1655)---xclock(1673)
        |-bash(214)
        |-cron(199)
        |-getty(213)
        |-gpm(143)
        |-inetd(138)
        |-kflushd(2)
        |-klogd(131)
        |-kpiod(4)
```

```
        |-kswapd(5)
        |-kupdate(3)
        |-lpd(153)
        |-portmap(136)
        |-postmaster(168)
        |-sendmail(179)
        |-sshd(183)
        |-syslogd(129)
        `-xfs(186)
```

En este otro formato. Aparece cada proceso con su PID. Podemos ver que el Proceso 'init' tiene pid = 1 y ha realizado varios forks() generando procesos con pid > 1. En algunos sistemas la generación de números de PID para procesos nuevos se realiza en secuencia. En otros resulta un número impredecible.

Entre los procesos generados por 'init' están los procesos 'getty'. Se arrancará un 'getty' por cada terminal. Este proceso configura la velocidad y otras cosas del terminal, manda un saludo y luego se transforma con exec el proceso 'login'. Todos estos procesos se ejecutan con EUID y UID = 0, es decir como procesos del superusuario root. Cuando el proceso 'login' conoce nuestra identidad después de validar usuario password se transformará con exec en la shell especificada e para nuestro usuario el fichero /etc/passwd

Para ver la línea que contiene sus datos pruebe a hacer lo siguiente:

```
$ grep `whoami` /etc/passwd
```

La línea tiene el siguiente formato. login:contraseña:UID:GID:nombre:dir:intérprete

Vamos a suponer que su shell por defecto sea la bash. Si esta shell arrancara con el EUID = 0 tendríamos todos los privilegios del super usuario pero esto no ocurre así. Esta shell ya tendrá nuestro UID y nuestro

EUID. Vamos a representar todo esto marcando los puntos en los que ocurre algún fork() con un signo '+'.

```
[init]-+fork()->[getty]
       |
       +fork()->[getty]-exec()->[login]-exec()->[bash]
+fork()-exec()->[comando]
       |
       +fork()->[getty]
       |
```

La shell puede arrancar un comando mediante un fork() y luego un exec() y esperar a que este muera. Recuerde que la función exec() no tiene retorno posible ya que finaliza con la muerte del proceso. En ese momento la shell detecta la muerte de su hijo y continua su ejecución solicitando la entrada de un nuevo comando. Cuando introducimos el comando 'exit' estamos indicando a la shell que finalice y su padre 'init' se encargará de lanzar nuevo proceso 'getty'. Lógicamente 'exit' es un comando interno de la shell. Quizás le llame la atención que la muerte de 'bash' termine provocando un nuevo 'getty' cuando 'getty' pasó a 'login' y este a 'bash' pero en esta secuencia getty-login-bash no hay ningún fork() por eso getty, login, y bash son en realidad el mismo proceso en distintos momentos con el mismo PID obtenido en el fork() realizado por 'init' solo que ha ido cambiando su personalidad manteniendo la misma identidad (mismo PID). Para 'init' siempre se trató del mismo hijo y la muerte de cualquiera de ellos (getty, login o bash) provoca que se arranque un nuevo 'getty' sobre ese mismo terminal con el fin de que ese terminal no quede sin servicio.

La presentación del mensaje de Login es mostrada por 'getty'. Una vez introducido el identificador de usuario será 'login' quien muestre la solicitud de introducción de la password, y una vez introducido el password será la shell quien muestre el introductor de comandos pero estamos hablando siempre del mismo proceso.

A modo de ejercicio compruebe estas cosas por usted mismo usando algunos de los comandos que ya conoce. Para hacer esta práctica no basta

con usar un terminal remoto sino que necesitará un PC completo para ir haciendo cosas desde distintas sesiones. Le proponemos hacerlo más o menos de la siguiente manera:

1. Entre en cada uno de los terminales disponibles de forma que todos los terminales estén ocupados por un intérprete de comandos. Bastará con hacer login en todos ellos.
2. Luego compruebe que no hay ningún proceso 'getty'.
3. Haga un exit desde uno de estos terminales.
4. Compruebe desde otro termina que ahora si existe un proceso 'getty' y anote su pid.
5. Introduzca el nombre de usuario en ese terminal que quedó libre.
6. Compruebe ahora desde otra sesión que existe un proceso login con el PID que usted anotó.
7. Termine de identificarse tecleando la password
8. Compruebe desde otra sesión que ahora existe una shell con el PID que anotamos.

   Si no tiene el comando 'pstree' tendrá que usar 'ps' pero con pstree puede ver más fácilmente lo que ocurrirá ahora.

9. Ahora teclee el comando 'sleep 222' desde la sesión que tiene el PID anotado por usted.
10. Compruebe desde otra sesión que el interprete de comandos ha realizado un fork() generando un comando que está ejecutando el comando indicado.

Si no ha podido realizar el ejercicio anterior tendrá que confiar en que las cosas son como decimos y ya está.

Comando ps

Muestra los procesos activos. Este comando es muy util para saber que comandos están funcionando en un determinado momento.

Siempre que se mencione un comando puede consultar la página man del mismo. En el caso de 'ps' se lo recomendamos ya que es un comando muy útil con una gran cantidad de opciones. Nosotros mencionaremos algunos ejemplos pero se aconseja probar 'ps' probando las distintas opciones que se mencionan en la página del manual.

Ejemplos:

```
$ # Para ver todos sus procesos que están
$ # asociados a algún terminal.
$ ps

PID TTY STAT TIME COMMAND
.......


$ # Para ver todos sus procesos y los de otros
$ # usuarios siempre asociados a algún terminal.
$ ps a

PID TTY STAT TIME COMMAND
.......


$ # Para ver todos sus procesos estén asociados o
$ # no a algún terminal.
$ ps x

PID TTY STAT TIME COMMAND
.......


$ # Para ver todos los proceso asociados al
$ # terminal 1
$ ps t1

PID TTY STAT TIME COMMAND
.......


$ # Para ver todos los procesos del sistema.
$ ps ax
```

```
PID TTY STAT TIME COMMAND
.......
```

Estos ejemplos que acabamos de ver obtienen un mismo formato de datos. Explicaremos el significado de estos atributos.

PID	Es el valor numérico que identifica al proceso.
TTY	Es el terminal asociado a ese proceso. Los demonios del sistema no tienen ningún terminal asociado y en este campo figurará un ?
STAT	Tiene tres campos que indican el estado del proceso (R,S,D,T,Z) (W) (N) La S indica que el proceso está suspendido esperando la liberación de un recurso (CPU, Entrada Salida, etc.) necesario para continuar. Explicaremos solo algunos de estos estados en su momento.
TIME	Indica el tiempo de CPU que lleva consumido ese proceso desde que fue arrancado.
COMMAND	Muestra el comando y los argumentos que le fueron comunicados.

Existen muchas opciones para el comando ps que ofrecen un formato distinto. Le recomendamos especialmente que pruebe 'ps u', 'ps l', y 'ps f'

En Unix los comandos suelen servir para una sola cosa, aunque suelen tener muchas opciones. La entrada de los comandos suele tener una estructura simple y la salida de los comandos también. Si un comando no encuentra nada que hacer existe la costumbre de que termine de modo silencioso. Todo esto permite que los comandos puedan combinarse enganchado la salida de uno con la entrada de otro. Algunos comandos

están especialmente diseñados para ser usados de esta forma y se les suele denominar filtros.

La salida del comando 'ps' se puede filtrar con 'grep' para que muestre solo las líneas que nos interesan.

Configuración del terminal

Conviene que comprobemos si su terminal está correctamente configurado para poder interrumpir un proceso. Normalmente se usa <Ctrl-C> pero esto depende de la configuración de su terminal. Si en algún momento su terminal queda desconfigurado haciendo cosas raras como por ejemplo mostrar caracteres extraños intente recuperar la situación tecleando el comando 'reset'. Esto solo es válido para Linux. Para otros sistemas puede ser útil 'stty sane' que también funciona en Linux pero no es tan eficaz como el comando 'reset'. Para comprobar la configuración de su terminal puede hacer 'stty -a' aunque obtendrá demasiada información que no es capaz de interpretar, podemos indicarle que se fije en el valor de 'intr'. Debería venir como 'intr = ^C'. Si no lo localiza haga 'stty -a | grep intr'. De esta forma solo obtendrá una línea. Para configurar el terminal de forma que pueda interrumpir procesos con <Ctrl-C> puede intentar configurar su terminal haciendo 'stty ^V^C'. El carácter <Ctrl-V> no se mostrará en el terminal ya que actúa evitando que el siguiente carácter (<Ctrl-C> en nuestro caso) no sea interpretado como carácter de control.

No pretendemos ahora explicar los terminales de Linux pero si queremos que compruebe su capacidad para interrumpir procesos con <Ctrl-C> ya que usaremos esto en las prácticas que siguen. Una prueba inofensiva para comprobar la interrupcion de un proceso es el siguiente comando que provoca una espera de un minuto. Deberá introducir el comando e interrumpirlo antes de que el tiempo establecido (60 segundos se agote).

```
$ sleep 60
<Ctrl-C>
```

Si no ha conseguido interrumpir el proceso no siga adelante para evitar que alguna de las prácticas deje un proceso demasiado tiempo consumiendo recursos de su máquina. Si esta usted solo en la máquina eso tampoco tendría mucha importancia pero es mejor que averigue la forma de interrumpir el proceso del ejemplo anterior.

Comando time

Da los tiempos de ejecucion. Este comando nos da tres valores cuya interpretación es:

```
 real   Tiempo real gastado (Duración real)
user    Tiempo CPU de usuario.
sys.    Tiempo CPU consumido como proceso de kernel.
        (Es decir dentro de las llamadas al kernel)
```

La mayoría de los comandos están gran parte del tiempo sin consumir CPU porque necesitan esperar para hacer entrada salida sobre dispositivos lentos que además pueden estar en uso compartidos por otros procesos. Existe un comando capaz de esperar tiempo sin gastar tiempo de CPU. Se trata del comando 'sleep'. Para usarlo le pasaremos un argumento que indique el número de segundos de dicha espera.

Por ejemplo vamos a comprobar cuanta CPU consume una espera de 6 segundos usando sleep

```
$ time sleep 6

real    0m6.021s
user    0m0.020s
sys     0m0.000s
```

El resultado obtenido puede variar ligeramente en cada sistema pero básicamente obtendrá un tiempo 'real' de unos 6 segundos y un tiempo de CPU ('user' + 'sys') muy pequeño.

Vamos a medir tiempos en un comando que realice operaciones de entrada salida así como proceso de datos.

```
$ time ls /* > /dev/null

real    0m0.099s
user    0m0.080s
sys     0m0.010s
```

En este comando verá que el consumo total de CPU es superior al del comando sleep. En cualquier caso el tiempo real tardado en la ejecución del comando es siempre muy superior al consumo de CPU.

Vamos a probar un comando que apenas realice otra cosa que entrada salida. Vamos a enviar 10Mbytes al dispositivo /dev/null. Existe un comando 'yes' que provoca la salida continua de un carácter 'y' seguido de un carácter retorno de carro. Esta pensado para sustituir la entrada de un comando interactivo en el cual queremos contestar afirmativamente a todo lo que pregunte. Nosotros filtraremos la salida de 'yes' con el comando 'head' para obtener solo los 10Mbytes primeros producidos por 'yes' y los enviaremos al dispositivo nulo '/dev/null' que viene a ser un pozo sin fondo en el cual podemos introducir cualquier cosa sin que se llene, pero no podremos sacar absolutamente nada. En una palabra vamos a provocar proceso de entrada salida perfectamente inútil.

```
$ time yes | head --bytes=10000000 > /dev/null

Tubería rota

real    0m6.478s
user    0m5.440s
sys     0m0.900s
```

Podemos hacer un consumo fuerte de CPU si forzamos a cálculos masivos que no tengan apenas entrada salida. Por ejemplo podemos poner a calcular el número PI con 300 cifras decimales. 'bc' es un comando que consiste en una calculadora. Admite uso interactivo pero también acepta que le pasemos las operaciones desde otro proceso combinando entrada salida.

```
$ time ( echo "scale=300; 4*a(1)" | bc -l )

3.14159265358979323846264338327950288419716939937510582
09749445923078\
16406286208998628034825342117067982148086513282306647
09384460955058\
22317253594081284811174502841027019385211055596446229
48954930381964\
28810975665933446128475648233786783165271201909145648
56692346034861\
04543266482133936072602491412737
real    0m2.528s
user    0m2.520s
sys     0m0.010s
```

En un Pentium 200Mhz este comando tardó 3 segundos para 300 cifras y 20 segundos usando 600 cifras. Decimos esto para que vea que el tiempo que se tarda aumenta exponencialmente y que dependiendo de la potencia de su ordenador puede suponer bastante tiempo de proceso. Quizás tenga que variar el número de cifras significativas para poder medir tiempos con comodidad.

Este comando 'time' es un comando interno pero en Linux también hay un comando externo llamado de la misma forma. Para poder ejecutarlo con esta shell debería incluir el camino completo. No deseamos abusar de su escaso 'time' así que no lo comentaremos. Para buscar en el man el comando 'time' que hemos explicado o de cualquier otro comando interno tendría que mirar en la página del manual de bash.

Comando kill

Este comando se utiliza para matar procesos. En realidad envía señales a otros procesos, pero la acción por defecto asociada a la mayoría de las señales de unix es la de finalizar el proceso. La finalización de un proceso puede venir acompañada del volcado de la información del proceso en disco. Se genera un fichero 'core' en el directorio actual que solo sirve para que los programadores localicen el fallo que provocó esta brusca finalización del proceso. Por ejemplo si el proceso intenta acceder fuera del espacio de memoria concedido por el kernel, recibirá una señal que lo matará. Lo mismo ocurrirá si se produce una división por cero o algún otro tipo de error irrecuperable.

Un proceso unix puede capturar cualquier señal excepto la señal 9. Una vez capturada la señal se puede activar una rutina que puede programarse con toda libertad para realizar cualquier cosa.

'kill' por defecto es 'kill -15' envía un SIGTERM y generalmente provoca cierre ordenado de los recursos en uso. Esta señal puede ser ignorada, o puede ser utilizada como un aviso para terminar ordenadamente. Para matar un proceso resulta recomendable enviar primero un kill -15 y si no se consigue nada repetir con kill -9. Este último -9 envía SIGKILL que no puede ser ignorada, y termina inmediatamente. Solo fallara si no tenemos permisos para matar ese proceso, pero si es un proceso nuestro, kill -9 resulta una opción segura para finalizar un proceso.

Las señales actúan frecuentemente como avisos de que ha ocurrido algo. Existen muchos tipos de señales para poder distinguir entre distintas categorías de incidencias posibles.

Comando nice

El multiproceso esta implementado concediendo cíclicamente la CPU en rodajas de tiempo a cada proceso que esta en ejecución.

Existen dos números de prioridad. La prioridad NICE y la prioridad concedida por el Kernel mediante un algoritmo. Esta última no tiene porque coincidir con nice y puede valer mas de 39. En cambio el comando nice solo acepta valores comprendidos entre 0 y 39, siendo 20 el valor por defecto. Cuando nice sube el valor significa que el proceso tiene baja prioridad. El comando 'nice -10' incrementara el valor nice en 10 (es decir baja la prioridad). Para bajar el valor de nice (Es decir para subir la prioridad) hace falta permisos de superusuario.

En un sistema con poca carga de trabajo no se notará apenas diferencia al ejecutar un comando con baja prioridad o con alta prioridad. Pero en un sistema con la CPU sobrecargada los comandos ejecutados con prioridad más baja se verán retrasados. ya que el kernel concederá más tiempo de CPU a los procesos con prioridad más alta.

Hay otros comandos de interés. Por ejemplo 'top' muestra los procesos que mas CPU estén consumiendo. 'vmstat' saca información del consumo de memoria virtual.

Hay que tener en cuenta que el sistema gasta recursos en la gestión de los procesos. Por ejemplo si estamos compartiendo una máquina con otros usuarios y tenemos que realizar 15 compilaciones importantes terminaremos antes haciéndolas en secuencia una detrás de otra que lanzándolas todas a la vez. La avaricia de querer usar toda la CPU posible para nosotros puede conducir a una situación en la cual ni nosotros ni nadie sacará gran provecho de la CPU. El sistema realizará una cantidad enorme de trabajo improductivo destinado a mantener simultáneamente funcionando una gran cantidad de procesos que gastan mucha CPU y mucha memoria. La máquina comienza a usar el disco duro para suplir la falta de RAM y comienza a gastar casi todo el tiempo en el intercambio de la RAM con el disco duro. A esta situación se la denomina swaping.

Comando renice

Sirve para cambiar la prioridad de un proceso. Sigue la misma filosofía que el comando nice pero hay que identificar el o los procesos que deseamos cambiar su prioridad. Se puede cambiar la prioridad de un proceso concreto dado su PID o los procesos de un usuario dando su UID o todos los procesos pertenecientes a un determinado grupo dando su GID. Tanto el comando nice como el comando 'renice' tienen mayor interés para un administrador de sistemas que para un usuario normal. Consulte las páginas del manual para más detalles.

SISTEMA DE FICHEROS (Primera parte)

Introducción

Este es sin duda uno de los capítulos con mayor interés práctico.

A menudo la palabra fichero se puede usar de dos formas. Ficheros en sentido amplio como cualquier cosa que tiene nombre en un sistema de ficheros y otra como ficheros propiamente dichos que más tarde llamaremos ficheros regulares. Estos últimos son la clase de ficheros más normalitos y sirven para contener información legible o ejecutable.

Se trata de una coincidencia de nombres especialmente lamentable porque dificulta mi trabajo de explicar ahora las cosas.

Por ejemplo cuando hablamos de generalidades para los nombres de ficheros y cuando hablemos de tipos de ficheros estaremos hablando de ficheros en sentido amplio, en cambio cuando hablamos de dirigir la salida a un fichero o cuando decimos que leemos un fichero nos referimos a ficheros propiamente dichos es decir ficheros regulares que son los que se usan normalmente como contenedores de información.

Las palabras no siempre significan lo mismo usadas en diferentes contextos así que nosotros confiaremos en su capacidad de deducción y usaremos indistintamente la palabra fichero para ambas cosas aunque reconocemos que es una costumbre que no facilita las cosas, pero por desgracia verá este término utilizado en diferentes sitios usado para ambas cosas. No se preocupe y tenga paciencia al fin y al cabo si ya ha llegado hasta aquí, su sagacidad a quedado prácticamente demostrada.

Comprenderá todo esto cuando hablemos de tipos de ficheros.

Para poner a prueba una vez más su sagacidad, (santa paciencia la suya) usaremos el término ficheros en sus dos acepciones para describir una clase de ficheros llamada directorios.

Los directorios pueden considerarse como una clase de ficheros que pueden contener toda clase de 'ficheros' (en sentido general) incluidos nuevos directorios, lo cual hace que la estructura final tenga estructura similar a la de un árbol.

Las ramas de un árbol forman la estructura de todo el árbol y de forma muy similar los directorios forman el armazón de toda la estructura del sistema de ficheros.

Un sistema de ficheros en Unix tiene esta estructura de árbol y el punto de origen se llama root (raíz) y se caracteriza porque el padre de 'root' resulta ser el mismo 'root' y se simboliza por '/'.

Ya sabemos que esto es raro pero no pregunte como alguien puede ser padre de si mismo. Esto sencillamente es un truco para rematar la estructura de todo el árbol de alguna forma.

Ojo, al súper usuario también se le llama 'root' pero no tiene nada que ver con esto. Es solo otra molesta coincidencia.

Nombres de ficheros

Un nombre de fichero valido para el núcleo de Linux puede contener cualquier carácter salvo el carácter '/' y salvo el carácter '\0' (carácter nulo). Sin embargo no es prudente usar caracteres especiales interpretados por la shell ($,",',&,#,(,),*,[,],{,}, etc..) Tampoco debe empezar por el carácter '-' que puede ser confundido con una opción. Por ejemplo si llamamos a un fichero '-r' no solo no podremos borrarlo sino que se puede confundir con la peligrosísima opción '-r' de borrado recursivo.

Cuando un usuario es dado de alta se define la shell por defecto que usará y el directorio de entrada. Este directorio de entrada al sistema será el valor de la variable $HOME.

En cada directorio hay dos entradas como mínimo que son '.' y '..' que referencian respectivamente al directorio actual y al directorio padre. (Cuando no tenemos en la variable $PATH referenciado el subdirectorio actual tendremos que ejecutar los comandos de nuestro directorio actual mediante ./comando)

El carácter '/' se utiliza para separar los diferentes componentes de un nombre de fichero y por eso no se puede usar como pare de un nombre de fichero. El carácter nulo se usa como fin de cadena en lenguaje C que es el lenguaje en el que está escrito el SO. Por eso tampoco se puede usar ese carácter como nombre de fichero.

Cuando usamos un nombre de fichero que empieza por '/' se llama nombre de camino completo o nombre de camino absoluto. Si por el contrario empieza por '.', por '..', o por un nombre se denominará nombre de camino relativo, porque su utilización depende del subdirectorio actual donde nos encontremos. La última parte del nombre después del último carácter '/' se llama nombre base. basename(1).

```
$ # Ejemplo de uso del comando basename
$ basename /usr/include/sys/signal.h

signal.h
```

Los ficheros que empiezan por '.' no tienen realmente nada de especial. Ya explicamos que la shell no expande los nombres de estos ficheros y algunos comandos como 'ls' y otros los consideran de una forma especial. Por ejemplo al usar el comando 'ls' sin parámetros no muestra esos ficheros. Por todo ello aparentan ser invisibles cuando en realidad son algunos comandos como 'ls' y el propio 'bash' quienes los tratan de forma especial. Muchos ficheros que son de uso particular de algunos programas

se usan de esta forma para que el usuario pueda ignorarlos en condiciones normales.

En 'msdos' existe un concepto llamado extensión. Corresponde con un máximo de tres caracteres después del último punto en un nombre de fichero. En Unix los nombres pueden tener varios puntos y después del último punto puede haber más de tres caracteres alfanuméricos ya que no existe el concepto de extensión. De existir nombres que terminan de una cierta forma puede tener significado especial para algunos programas pero no para núcleo (kernel) y tampoco para el intérprete de comandos.

Lo que si tiene un tratamiento especial a nivel de sistema operativo son los caracteres '/' que forzosamente indican los elemento de un camino formado por la sucesión de caracteres. Es decir será imposible crear un fichero con nombre 'x/a' dentro de un directorio porque el sistema interpretaría que queremos crear un fichero 'a' dentro de un directorio 'x'.

Comandos para manejar el sistema de ficheros

Relación de comandos más importantes para manejar el sistema de ficheros.

```
ls <lista>      Muestra el contenido de directorio actual.
cd <dir>        Cambiar a un subdirectorio.
mkdir <lista-dir>       Crea uno o mas directorio.
rmdir <lista-dir>       Borra uno mas directorios.
cp <lista>      Copiar uno o mas ficheros al ultimo de la
lista.
                Si mas de dos el ultimo debe ser un
directorio.
mv <lista>      mover o renombrar ficheros o directorios al
ultimo
                nombre de la lista.
                Si mas de dos el ultimo debe ser un
directorio.
rm <lista>      borrar la lista de ficheros.
ln               Crear un enlace a un fichero
touch           Crea un fichero vacío o modifica la fecha de
un fichero
pwd             muestra el camino absoluto del directorio
actual.
chmod           Cambia los permisos de un fichero.
chown           Cambia el propietario de un fichero.
chgrp           Cambia el grupo de un fichero.
du <fichero>    Ocupación  de un fichero
tree            Listado recursivo
tree -d         Mostrar árbol de directorios
file <fichero>  Obtiene información sobre el tipo de
Ficheros
```

Recuerde que tiene a su disposición el comando 'man' para consultar las distintas opciones de cada uno de estos comandos.

Alias para el uso de los comandos más frecuentes

El comando 'cd' es un comando interno que sirve para cambiar de directorio actual. Existe otro comando interno llamado 'alias' que sirve

para algo que no tiene que ver con el sistema de ficheros y sin embargo lo mencionaremos aquí por razones prácticas.

En muchos sistemas parecerá que existe la disponibilidad de usar un comando 'll' que sirve para mostrar un directorio con abundante información. En realidad se trata de un alias. Usted puede verificar los alias definidos para su usuario tecleando 'alias' sin argumentos. Generalmente se usa para abreviar el uso de comandos muy utilizados. Esto puede resultarle de interés para personalizar su entorno.

Cuando el interprete de comandos arranca se ejecutan una serie de ficheros '/etc/profile', '~/.bash_profile', '~/.bash_login', y '~/.profile' en este orden. En '.bash_profile' suele existir una parte que pregunta por la existencia de un '~/.alias' y en caso afirmativo lo ejecuta. Este fichero '.alias' contiene una serie de comandos de alias que permanecerán disponibles durante su sesión de trabajo con esa shell.

Podría parecer que estamos obsesionados con la shell y que no sabemos hablar de otra cosa pero solo queremos proporcionarle ahora unos consejos prácticos que le faciliten el uso de su sistema. Para ello tendría que editar el fichero '.alias' presente en su $HOME y añadir las líneas siguientes:

```
alias ll='ls -lrt --color=auto'
alias ll.='ls -adlrt --color=auto .[a-zA-Z0-9]*'
alias ls='ls --color=auto'
alias ls.='ls -adrt --color=auto .[a-zA-Z0-9]*'
alias md='mkdir'
alias mv='mv -i'
alias rd='rmdir'
alias rm='rm -i'
```

Puede ejecutar el fichero '.alias' para definir todos estos alias

```
$ . .alias
```

O puede ejecutar cualquiera de estos comandos alias ahora mismo, por ejemplo

```
$ alias ll='ls -lrt --color=auto'
```

Luego teclee el comando 'alias'. Comprobará que queda definido. Teclee el nuevo comando 'll' y compruebe su efecto.

El bash solo expande los alias de la primera palabra de un comando y solo cuando no van entrecomillados. Por eso cuando definimos un alias que sustituye a un comando (Por ejemplo "ls='ls --color=auto'") tendremos que usar 'ls' entre comillas simples o dobles para si no deseamos que se produzca la expansión del alias.

Para eliminar una definición de alias utilice el comando 'unalias' En nuestro caso.

```
$ unalias ll
```

Para no tener que teclear manualmente cada vez cada uno de estos alias es para lo que sirve el fichero '.alias'.

Si no sabe editarlo tendrá que esperar a que expliquemos el manejo del editor 'vi' y volver a esta sección para modificar su fichero '.alias'. Los alias serán definidos la próxima vez que entre en una nueva sesión. Explicaremos las ventajas de usar algunos de los alias que acabamos de proponerle.

ll Los ficheros más interesantes son generalmente los más recientes y para evitar que desaparezcan durante un scroll largo los mostraremos en último lugar. La opción de color auto colorea la salida del comando únicamente si la salida ocurre en el terminal.

ll. Sirve para listar ficheros que empiezan por un punto y después tienen una letra o dígito como segundo carácter. Normalmente los ficheros de configuración de su usuario se encuentran en su $HOME y utilizan nombres de este tipo que no aparecen en la expansión de un '*'.

rm Este alias evita usar 'rm' tal cual está añadiéndole la opción -i para confirmar antes de borrar. Realmente útil. Es muy recomendable usar 'rm' por defecto de esta forma. Si no desea ser interrogado a cada fichero si realmente desea borrarlo o no, utilice este alias tal como está pero añadiendo la opción -f. rm -f <lista-de-ficheros> borrará silenciosamente todo lo que usted ordene que no tiene que coincidir con todo lo que usted supone que ha ordenado. En especial si no pone cuidado al usar la expansión de ordenes de la shell.

La respuesta a la pregunta de como recuperar un fichero borrado accidentalmente es... 'De ninguna forma'.

Cosas simpáticas de Unix y Linux.

Puede que sepa de alguien que en cierta ocasión recuperó una información valiosa usando trucos o herramientas especiales pero no cuente con ello. Hay programas de borrado que usan papeleras y quizás piense en poner un alias de rm para que use uno de estos programas. Esto introduce ineficiencia en el sistema pero puede hacerse a nivel de usuario. Lo más correcto es usar una buena política de copias de seguridad.

mv Se trata de mover información pero aplicamos el mismo principio que para 'rm'.

Ejemplos de uso de los comandos estudiados

Ejecutaremos paso a paso una serie de comandos y comprobaremos lo que ocurre.

```
$ # Crearemos un alias para ls con algunas opciones
```

```
$ alias ll='ls -lrt --color=auto'
$ # Creamos en /tmp una serie de directorios anidados
$ cd /tmp
$ mkdir kkkk
$ cd kkkk
$ # Consultamos en que directorio estamos situados
$ pwd

/tmp/kkkk

$ # Continuamos creando ficheros y directorios
$ echo xx > kk1
$ echo xx > kk2
$ echo xx > kk3
$ mkdir kkkk
$ # Usamos el alias por primera vez para ver el
contenido
$ # del directorio actual.
$ ll

total 4
-rw-r--r--   1 acastro  acastro         3 abr 16 20:01
kk1
-rw-r--r--   1 acastro  acastro         3 abr 16 20:01
kk2
-rw-r--r--   1 acastro  acastro         3 abr 16 20:01
kk3
drwxr-xr-x   2 acastro  acastro      1024 abr 16 20:01
kkkk

$ # Continuamos creando ficheros y directorios
$ cd kkkk
$ echo xx > kk1
$ echo xx > kk2
$ echo xx > kk3
$ mkdir kkkk
$ cd kkkk
$ echo xx > kk1
$ echo xx > kk2
$ echo xx > kk3
$ cd /tmp/kkkk
$ # El alias ll corresponde con el comando ls con una
serie de
$ # opciones pero podemos añadir nuevas opciones. Por
ejemplo
$ # podemos hacer un listado recursivo de toda la
```

```
estructura de
$ # ficheros y directorios que cuelga del directorio
actual.
$ # /tmp/kkk
$ ll -R .

total 4
-rw-r--r--   1 acastro  acastro         3 abr 16 20:01
kk1
-rw-r--r--   1 acastro  acastro         3 abr 16 20:01
kk2
-rw-r--r--   1 acastro  acastro         3 abr 16 20:01
kk3
drwxr-xr-x   3 acastro  acastro      1024 abr 16 20:03
kkkk

kkkk:
total 4
-rw-r--r--   1 acastro  acastro         3 abr 16 20:03
kk1
-rw-r--r--   1 acastro  acastro         3 abr 16 20:03
kk2
-rw-r--r--   1 acastro  acastro         3 abr 16 20:03
kk3
drwxr-xr-x   2 acastro  acastro      1024 abr 16 20:04
kkkk

kkkk/kkkk:
total 3
-rw-r--r--   1 acastro  acastro         3 abr 16 20:04
kk1
-rw-r--r--   1 acastro  acastro         3 abr 16 20:04
kk2
-rw-r--r--   1 acastro  acastro         3 abr 16 20:04
kk3

$ # Con el comando tree y la opción -d podemos ver la
estructura
$ # simplemente con los directorios y como están
colocados.
$ tree -d

.
`-- kkkk
    `-- kkkk
```

```
    2 directories


$ # Para ver la estructura completa usamos el comando
$ # tree sin opciones. Vemos ahora tambien los
ficheros.
$ tree

.
|-- kk1
|-- kk2
|-- kk3
`-- kkkk
    |-- kk1
    |-- kk2
    |-- kk3
    `-- kkkk
        |-- kk1
        |-- kk2
        `-- kk3

2 directories, 9 files

$ # Con find podemos buscar usando muchos criterios.
Por ejemplo
$ # por nombre.
$ find . -name 'kk*'

./kk1
./kk2
./kk3
./kkkk
./kkkk/kk1
./kkkk/kk2
./kkkk/kk3
./kkkk/kkkk
./kkkk/kkkk/kk1
./kkkk/kkkk/kk2
./kkkk/kkkk/kk3

$ # Podemos buscar por fecha de modificación comparanda
con la
$ # fecha de otro fichero. Por ejemplo que ficheros son
más
$ # recientes que './kkkk/kk2' ?
$ find . -newer ./kkkk/kk2
```

```
./kkkk
./kkkk/kk3
./kkkk/kkkk
./kkkk/kkkk/kk1
./kkkk/kkkk/kk2
./kkkk/kkkk/kk3

$ # Podemos comprobar la ocupación de los directorios
$ du .

4       ./kkkk/kkkk
8       ./kkkk
12      .
```

Para borrar todo esto se suele usar la opción de borrado recursivo. No debe usar esta opción como root sin estar muy seguro de lo que hace. Esta opción borra recursivamente y es muy peligrosa porque un error puede borrar recursivamente desde un lugar equivocado un montón de información.

No es necesario borrar nada porque en /tmp la información suele borrarse durante el arranque. Si quiere intentarlo con las precauciones mencionadas teclee 'rm -fr /tmp/kkkk'. Si lo prefiere puede dejarlo sin borar ya que no ocupa casi nada y se borrará la próxima vez que se vacíe el '/tmp' que será seguramente cuando vuelva a arrancar el ordenador.

Para que los comandos anteriores puedan funcionar correctamente deberemos tener los permisos adecuados sobre los ficheros y los subdirectorios.

Evidentemente hay muchos más y no podemos explicarlos todos ahora pero quizás sea bueno mostrar una pequeña lista de algunos de ellos que veremos más adelante en posteriores capítulos.

```
sum             Obtiene un valor de checksum
cksum           Obtiene un valor de checksum largo y por
tanto más seguro
```

```
gzip            Compresor de GNU
find            Buscar ficheros
tar             Empaqueta ficheros y directorios en un solo
fichero.
cpio            Empaqueta ficheros y directorios en un solo
fichero.
```

Sistema plano de ficheros

La estructura en forma de árbol tiene el inconveniente de que se requiere ir recorriendo el árbol para ir localizando cada uno de los elementos. Por ello se hace necesario un sistema mucho más directo.

A cada ficheros se le asocia un número para poder localizarlo directamente. Dicho número se llama inodo. Los números de inodos son el índice del sistema plano de ficheros, también llamado por ello tabla de inodos Cada fichero de cada sistema de ficheros tiene un inodo distinto salvo que se trate de un enlace rígido (hard link). Sin embargo hay inodos que no se corresponden con ningún fichero.

Un fichero en sentido estricto indica capacidad de contener una secuencia de bytes.

Un inodo indica básicamente la capacidad de tener un nombre asociado al sistema de ficheros.

Un fichero tiene asociadas funciones de posición y de lectura y escritura de información dentro del fichero. Un inodo es una referencia muy directa a un fichero y a grandes rasgos tiene asociadas funciones que permiten manejar y construir la estructura de árbol del sistema de ficheros.

Tipos de ficheros

- Regular. Son meros almacenes de información. Algunos contiene código ejecutable.
- Directorios Son una tabla con números de inodos y nombres de ficheros.

- Ficheros especiales. Pueden ser dispositivo tipo carácter o dispositivo de bloques. El manejo de estos ficheros depende del dispositivo en particular.
- Fifo son pipes con nombre. Son propios de System V y en BSD no existen pero en Linux si están disponibles.
- Enlaces simbólicos (symbolic links). Son ficheros que contiene un puntero a otro fichero que podría perfectamente estar en un sistema de ficheros distinto.
- El Soket es un fichero especial de 4.3 BSD y se utiliza para comunicar procesos que pueden estar en máquinas distintas.
- Enlaces rígidos (hard links). Realmente es un único fichero que puede ser visto con distintos nombres dentro de un mismo sistema de ficheros. Es decir se pueden observar como si fueran ficheros idénticos con el mismo inodo. La información reside en un mismo lugar y lo que ocurra aparentemente en un sitio ocurrirá instantáneamente en el otro lugar. Los enlaces de este tipo no pueden ocurrir entre sistemas de ficheros distintos. No todos los sistemas de ficheros soportan hard links ya que es un concepto muy ligado a los SO tipo Unix.

Generalmente los diferentes tipos de ficheros son capaces de aceptar operaciones de apertura, cierre, lectura y escritura de forma similar. Por ello se puede redirigir una entrada y una salida a diferentes tipos de ficheros. Parte de la enorme potencia y flexibilidad de los SO tipo Unix residen en este hecho. Gracias a esto la combinación de entradas salidas entre procesos, ficheros, dispositivos, etc.. resulta muy sencilla y flexible. Existe una salvedad importante. Los directorios en este sentido son totalmente especiales y un intento de escribir en un directorio dará siempre un error aunque sea realizado por 'root'. No tiene sentido permitirlo. Los directorios se crean, borran, y modifican con comandos específicos para ellos.

```
$ echo > .

bash: .: Es un directorio
```

Atributos de fecha en ficheros

En Unix y Linux los ficheros en sentido amplio tienen asociadas siempre tres fechas. En realidad estas fechas están almacenadas internamente como el número de segundos transcurridos desde el '1 de Enero de 1970'.

Una de ellas indica el tiempo del último acceso realizado sobre ese fichero. La otra indica la fecha de creación (en realidad es la fecha del último cambio de estado) de ese fichero y la última y quizás más importante (es la que vemos al consultar con 'ls -l') indica la fecha de la última modificación.

En realidad si cambiamos el estado de un fichero por ejemplo cambiando permisos o el propietario se modificarán las fechas de creación y del último acceso. Por eso la fecha que hemos denominado de creación puede ser posterior a la fecha de modificación del fichero.

Cuando se crea un fichero las tres fechas tendrán el mismo valor. Cuando se lee un fichero se modifica la fecha de acceso del mismo pero acceder al nombre de un fichero o consultar el estado de un fichero no modifica ninguna fecha. Su fecha de acceso tampoco ya que en realidad lo que se hace es leer la información del directorio que lo contiene el cual si que verá modificada su fecha de acceso.

Las fechas de modificación asociadas a los directorios cambian con las altas, o bajas de los elementos dentro del directorio. Esto es debido a que dichas operaciones se consideran como si fueran escrituras en el fichero especial de tipo directorio. Un directorio es al fin y al cabo una simple tabla. Esto lo recordaremos cuando hablemos de permisos asociados a directorios.

Para modificar las fechas de modificación y de acceso de un fichero se puede usar el comando touch(1)

SISTEMA DE FICHEROS (Segunda parte)

Permisos de ficheros

Usaremos el término fichero en su sentido más amplio. Es decir que el tema de permisos es aplicable a distintos tipos de ficheros con algunas matizaciones que explicaremos más adelante. Los ficheros tienen muchos atributos además de su nombre. Para ver los más significativos haremos:

```
$ ls -l /
```

Supongamos que tenemos el siguiente fichero llamado 'kkkkk'

```
-rwxrwxrwx  1 root root  14740 abr 15 12:05 kkkkk
^^^^^^^^^^  ^  ^    ^      ^    ^   ^  ^  ^  ^__nombre del fich.
|||||||||   |  |    |      |    |   |  |  |  |
|||||||||   |  |    |      |    |   |  |  |__minutos : Fecha y
|||||||||   |  |    |      |    |   |  |_____hora    : hora de la
|||||||||   |  |    |      |    |   |____día del més : última
|||||||||   |  |    |      |    |________més         : modificación
|||||||||   |  |    |      |
|||||||||   |  |    |      |___________ Tamaño en bytes
|||||||||   |  |    |
|||||||||   |  |    |_______nombre del grupo
|||||||||   |  |____________nombre del propietario del fichero
|||||||||   |
|||||||||   |_______________número de enlaces rígidos (hard links)
|||||||||
|||||||||___001_____________ permiso de ejecución para : Un usuario
||||||||____002_____________ permiso de escritura para : cualquiera
|||||||_____004_____________ permiso de lectura para   :
|||||||
||||||_______010_____________ permiso de ejecución para : Un usuario
|||||________020_____________ permiso de escritura para : pertene-
||||_________040_____________ permiso de lectura para   : ciente al
||||                                                     : grupo
||||
||||__________100_____________ permiso de ejecución para : El usuario
|||___________200_____________ permiso de escritura para : propieta-
||____________400_____________ permiso de lectura para   : rio
|
|_______________________________tipo de fichero
                                  -  Fichero regular (fichero normal)
```

```
d  Directorio
l  Enlace simbólico
p  Fifo con nombre
b  Dispositivo de bloques
c  Dispositivo de caracteres
```

En inglés se usan los términos owner , group, y others para designar respectivamente al propietario, al grupo y a cualquier usuario.

Notación numérica para permisos

Recuerde que también tenemos que pensar en la posibilidad de que mi amigo David acostumbrado al uso del azadón decida hacer este curso y teníamos un compromiso de explicar las cosas partiendo de cero. Los permisos de los ficheros son almacenados en formato binario y se puede referenciar numéricamente. Vimos que a cada permiso individual le asociábamos un número de tres dígitos formado por dos ceros y un tercer número que podía ser únicamente 1, 2, o 4. Por ejemplo el permiso de escritura para un usuario cualquiera era 002. Con estos números se puede codificar los permisos de la forma que indicamos en el siguiente ejemplo:

```
r w x - - w x - r - x     Esto equivaldría a un permiso
735
4 2 1 - 0 2 1 - 4 0 1     ( 4+2+1 ,  0+2+1 , 4+0+1  = 7,3,5
)
```

Los permisos también pueden representarse como una secuencia de bits. Un bit es un valor que solo puede valer 0 o 1. En el caso anterior podríamos representarlo de la forma 111 010 101 donde 1 indica que si hay permiso 0 que no lo hay y la posición de cada 0 y cada 1 representa a cada uno de los permisos.

Umask

Es un comando interno del bash. Se utiliza cuando se crean ficheros.

No podemos profundizar mucho en temas de matemática binaria porque nos saldríamos del tema pero la umask se aplica mediante una operación llamada AND NOT.

Consiste en tomar a umask como una máscara donde los bits a 1 especifican los bits de permisos que se pondrán a cero.

Por ejemplo si queremos abrir un fichero con permisos 664 y tenemos una umask de 022 obtendremos un fichero 644.

```
664   110 110 100
022   000 010 010
----------------
644   110 100 100
```

Esto tiene la utilidad de proteger el sistema frente a cierto tipo de descuidos durante la creación de ficheros.

Por ejemplo supongamos que un administrador de un sistema considera que por defecto todos los ficheros que el crea deberían carecer de permisos de ejecución y de escritura para todo el mundo y que para los usuarios de su mismo grupo deberían de carecer de permiso de ejecución. Para ello establecerá una 'umask = 023'. Es decir una umask que elimina los permisos - - - . - w - . - w x

Para un script que genere ficheros que solo deban ser accesibles para el usuario que los ha generado usaríamos. 'umask = 077'

Comando chmod

Este comando sirve para alterar una serie de atributos del fichero Existe dos formas de usarlo. Una indicando el tipo de acceso y a quien lo queremos conceder o eliminar. Existen una serie de atributos que se pueden modificar usando este comando.

```
04000   Set uid on execute.
02000   Set gid on execute.
01000   Save text on image after execution.
00400 r Permiso de lectura para el propietario   (owner)
00200 w Permiso de escritura para el propietario
00100 x Permiso de ejecución para el propietario
00040 r Permiso de lectura para el grupo         (group)
```

```
00020 w Permiso de escritura para el grupo
00010 x Permiso de ejecucion para el grupo
00004 r Permiso de lectura para cualquiera       (others)
00002 w Permiso de escritura para cualquiera
00001 x Permiso de ejecucion para cualquiera
```

Los atributos Set uid, Set gid, y Save text no los explicaremos de momento.

Consulte las páginas del manual relativas a este comando. Como de costumbre el manual resulta imprescindible a la hora de recordar la utilización de un comando pero no a la hora de comprender conceptos nuevos. No se preocupe con los ejercicios que le proponemos a continuación comprenderá perfectamente:

```
$ cd /tmp
$ echo > kk
$ ###################################
$ chmod 777 kk
$ ls -l kk

-rwxrwxrwx  .... .... ......

$ ###################################
$ chmod 707 kk
$ ls -l kk

-rwx---rwx  .... .... ......

$ ###################################
$ chmod 421 kk
$ ls -l kk

-r---w---x  .... .... ......

$ ###################################
$ chmod 124 kk
$ ls -l kk

---x-w-r--  .... .... ......

$ ###################################
$ # 'chmod 0 kk'  equivale a 'chmod 000 kk'
```

```
$ chmod 0 kk
$ ls -l kk

----------  .... .... ......

$ ##################################
$ chmod +r kk
$ ls -l kk

-r--r--r--  .... .... ......

$ ##################################
$ chmod +x kk
$ ls -l kk

-r-xr-xr-x  .... .... ......

$ ##################################
$ chmod -r kk
$ ls -l kk

---x--x--x  .... .... ......

$ ##################################
$ chmod u+r kk
$ ls -l kk

-r-x--x--x  .... .... ......

$ ##################################
$ chmod a-x kk
$ ls -l kk

-r--------  .... .... ......

$ ##################################
$ chmod g+x kk
$ ls -l kk

-r----x---  .... .... ......

$ ##################################
$ chmod o+x kk
$ ls -l kk

-r----x--x  .... .... ......
```

```
$ ##################################
$ chmod a+rwx kk
$ ls -l kk

-rwxrwxrwx  .... .... ......

rm kk
```

Comando chown

El comando 'chown' sirve para cambiar el UID y el GID de un fichero. Esto solo se puede hacer si tenemos los permisos para ello. Normalmente su uso está reservado a 'root' y por ello no diremos nada más. Es muy posible que si usted usa Linux en un ordenador personal necesite algunos de estos conocimientos pero se sale de los propósitos de este curso. Le basta con saber que existe y para que sirve.

Concesión de acceso por parte del kernel

Explicaremos el funcionamiento de los permisos de la forma más precisa que nos sea posible.

Para ello usaremos unas abreviaturas que ya conocemos.

- EUID es el Identificador de usuario efectivo de un proceso
- EGID es el Identificador de grupo efectivo de un proceso
- UID en un fichero es un atributo que identifica al propietario.
- GID en un fichero es un atributo que identifica al grupo del propietario.

En realidad hablamos de propietario como el usuario que creo el fichero. El Kernel realizará el siguiente test para conceder a un proceso el acceso de cierto tipo a un fichero.

1. Si el EUID del proceso es 0 se da acceso. (root puede hacer lo que sea)

2. Si el EUID del proceso es igual al UID del owner (propietario del fichero) se concede el acceso si los permisos de usuario rwx son los adecuados.
3. Si el EUID del proceso es distinto al UID del owner, y si el EGID del proceso es igual al GID del owner, se concede el acceso si los permisos de grupo rwx son los adecuados.
4. Si el EUID del proceso es distinto al UID del owner, y si el EGID del proceso es distinto al GID del owner, se concede el acceso si los permisos de others rwx son los adecuados.

NOTA el comando rm permite borrar cualquier fichero sea cual sean los permisos cuando el proceso tiene un EUID coincidente con el propietario del fichero. Únicamente ocurre que si el fichero esta protegido contra escritura suele pedir (dependiendo de la configuración) confirmación antes de borrarlo. El permiso para borrar un fichero no se guarda como atributo del fichero. Esto se hace a nivel de directorio y pasamos a explicarlo inmediatamente.

Significado de los permisos en directorios

Para entender como funcionan los permisos aplicados a un directorio hay que imaginar que un directorio es un fichero normal que solo contiene una tabla en la que se relacionan los ficheros presentes en ese directorio. En todos los directorios existen siempre un par de entradas obligadas que son '.' y '..' para el directorio actual y para el directorio padre respectivamente. Vamos a explicarlo sobre la marcha a la vez que hacemos el ejercicio. Recuerde usar un usuario normal (distinto de root) para realizar los ejercicios.

```
$ cd /tmp
$ mkdir kk
$ cd kk
$ echo > k1
$ echo > k2
$ cd /tmp
$ chmod 777 kk
```

```
$ ## Para ver como ha quedado
los permisos hacemos ...
$ ls -ld kk

drwxrwxrwx   2 ..... ......
kk

$ ## Para ver que ficheros
contiene el directorio kk
hacemos ...
$ ls  kk

k1 k2
```

Si un directorio no tiene permiso de lectura, resultara imposible para cualquier comando incluido ls averiguar cual es su contenido.

Sin abandonar la sesión anterior continuamos el ejercicio

```
$ ###(1)###
$ ## Eliminamos los premisos de lectura del
directorio kk
$ chmod -r kk
$ ls -ld kk

d-wx-wx-wx   2 ..... ...... kk

$ ls -l kk

ls: kk: Permiso denegado
$ ## En este intante podemos entrar dentro del
directorio
$ ## Incluso podemos crear un nuevo fichero pero ya
no podemos
$ ## Saber cual es el contenido del directorio.
$ cd kk
$ pwd

/tmp/kk

$ echo > k3
$ ls -l
```

```
ls: .: Permiso denegado

$ ## Hemos podido meternos dentro del directorio kk
porque aun
$ ## Tenemos permiso para ello pero seguimos sin
poder saber
$ ## cual es su contenido.
$ cd /tmp
```

Si no se tiene permiso de ejecución en un directorio no se podrá hacer cd a ese directorio, ni a ninguno de los directorios que cuelgan de el. Esto imposibilita todas las operaciones que utilicen ese directorio como parte del camino.

Sin abandonar la sesión anterior continuamos el ejercicio

```
$ ###(2)###
$ ## Ahora eliminamos permiso de ejecución
$ chmod -x kk
$ ls -ld kk

d-w--w--w-   2 ..... ...... kk

$ ## Intentamos entrar
$ cd kk

bash: cd: kk: Permiso denegado

$ ## No hemos podido entrar
$ rm kk/k2

bash: cd: kk: Permiso denegado

$ ## Tampoco nos deja borrar el fichero desde fuera
$ #
$ ## Recuperamos permiso de ejecución
$ chmod +x kk
$ ls -ld kk

d-wx-wx-wx   2 ..... ...... kk
```

```
$ rm kk/k2
$ ls kk

k1
```

Si no tiene permiso de escritura no se podra dar altas, baja o modificaciones en la tabla lo que se traduce en que no se pueden borrar sus ficheros ni crear otros nuevos, ni renombrar ficheros.

Sin abandonar la sesión anterior continuamos el ejercicio

```
$ ###(3)###
$ ## Ahora eliminaremos permisos de escritura
$ chmod 666 kk
$ ls -ld kk

dr-xr-xr-x   2 ..... ...... kk

$ ## Ahora no tenemos permiso de escritura
$ cd kk
$ ls

rm: ¿borrar `k1'? (s/n)  s
rm: k1: Permiso denegado

$ echo > k3

bash: k3: Permiso denegado

$ touch k3

touch: k3: Permiso denegado

$ ## Sin permiso de escritura en el directorio no se
pueden
$ ## Crear ficheros nuevos dentro de el ni borrar
ficheros
$ ## Pero los ficheros son perfectamente accesibles
$ echo "Mensaje de prueba" > k1
$ cat k1
```

```
Mensaje de prueba

$ ## Vamos a limpiar
$ chmod 777 /tmp/kk
$ rm /tmp/kk/*
$ rmdir /tmp/kk
```

Setuid y Setgid

Cada proceso tiene un (R)UID, (R)GID, EUID, y EGUID. Cada fichero tiene un UID y un GID owner.

El comando 'chmod 4000' pone el bit setuid . Esto hace que el proceso que ejecuta este comando temporalmente cambie su EUID tomando el valor UID del fichero. Es decir el proceso sufre un cambio de personalidad y se convierte en la persona que figura como propietaria de ese fichero. Para ello lógicamente ese fichero deberá ser un ejecutable. El comando passwd sirve para cambiar la clave. Se trata de un comando propiedad de root con setuid. Por lo tanto mientras estamos ejecutando este comando adoptamos la personalidad de 'root'. Gracias a ello se nos permite acceder a la clave para modificarla y gracias a que ese comando no permite hacer nada más que eso se evita que este proceso con personalidad de 'root' pueda hacer otras cosas.

Busque el comando 'passwd' en su sistema. Podría estar en '/usr/bin/passwd' pero si no se encuentra en ese sitio puede localizarlo con 'which passwd'

Haga un ll para comprobar sus permisos y verá que es propiedad de 'root' y que tiene una 's' en lugar de una 'x' en la parte de los permisos para el propietario.

```
-rwsr-xr-x   1 root     root        28896 jul 17  1998
/usr/bin/passwd
```

El comando 'chmod 2000' pone el bit setgid on execute. Hace algo parecido al anterior pero a nivel de grupo. Es decir el proceso que ejecuta

este comando temporalmente cambie su EGID tomando el valor GID del fichero. Por ejemplo el comado lpr no solo nos convierte en 'root' cuando lo ejecutamos sino que nos cambia el grupo por el grupo de impresión.

```
-rwsr-sr-x   1 root     lp          14844 feb  7  1999
/usr/bin/lpr
```

El comando 'chmod 1000' pone el bit sticky. Se usa poco y tenía más utilidad en sistemas Unix más antigüos. Sirve para que el proceso permanezca en memoria RAM todo el rato mientras se esté ejecutando. Normalmente cuando muchos procesos usan la memoria RAM se van turnando en su uso y se descargan a disco para ceder su espacio en memoria RAM a otros procesos. Con el uso de este bit se evita la descarga a disco. Es una forma de aumentar la velocidad de ese proceso.

Para ver el aspecto de estos permisos realice el siguiente ejercicio. Realice siempre los ejercicios desde un usuario normal distinto de root.

```
$ cd /tmp
$ echo  > kk
$ chmod 4000 kk
$ ll kk

---S------   1 ..... ..... kk

$ chmod 4100 kk

---s------   1 ..... ..... kk

$ chmod 2000 kk

------S---   1 ..... ..... kk

$ chmod 2010 kk

------s---   1 ..... ..... kk

$ chmod 1000 kk

---------T   1 ..... ..... kk

$ chmod 1001 kk
```

```
---------t   1 ..... ..... kk
```

SISTEMA DE FICHEROS (Tercera parte)

Tipos de enlaces (links) y su manejo

La palabra link es inglesa y para el concepto que vamos a tratar podríamos traducirla por enlace. Hay dos tipos de enlaces llamados 'hard link' y 'symbolic link'. Podriamos traducirlo por enlace rígido y por enlace simbólico. El término enlace simbólico se usa con frecuencia en español y parece muy adecuado pero realmente no existe una traducción para hard link tan aceptada. Nosotros emplearemos el término rígido para hard que literalmente significa duro.

Ya hemos mencionado algunas cosas sobre ellos pero sin profundizar demasiado.

El manejo de ambos tipos de enlaces se hace con el comando 'ln'. Como es habitual un repaso a la página man de este comando es muy recomendable. La creación de un enlace rígido y uno simbólico es muy similar. La diferencia es que para el enlace simbólico tendremos que usar la opción -s.

Como es habitual asumimos que las prácticas se realizarán desde un usuario distinto de root.

```
$ cd /tmp
$ mkdir /tmp2
$ cd tmp2
$ echo xxxx > ej0
$ ## Ahora creamos un par de enlaces rígidos con ej0
$ ln ej0 ej1
$ ln ej0 ej2
$ ## Creamos también un enlace simbólico ejs1 a ej0
$ ln -s ej0 ejs1
$ mkdir dir1
```

```
$ ## También creamos un enlace simbólico dir2 a dir1
$ ln -s dir1 dir2
$ ls -l

drwxr-xr-x   2 pepe user 1024 may 18 17:28 dir1
lrwxrwxrwx   1 pepe user    4 may 18 17:28 dir2 -> dir1
-rw-r--r--   3 pepe user    1 may 18 17:26 ej0
-rw-r--r--   3 pepe user    1 may 18 17:26 ej1
-rw-r--r--   3 pepe user    1 may 18 17:26 ej2
lrwxrwxrwx   1 pepe user    3 may 18 17:28 ejs1 -> ej0
```

Con esto acabamos de crear dentro de tmp un directorio tmp2 y dentro de el hemos creado ya algunos ficheros, algunos directorios y unos cuantos enlaces rígidos y simbólicos. Los enlaces rígidos no muestran nada particular listados con 'ls -l' pero los enlaces simbólicos vienen acompañados de una flecha que apunta a otro nombre de fichero.

El fichero 'ej0' lo hemos creado con un contenido 'xxxx' para ver que pasa con ese contenido más adelante. El nombre de usuario 'pepe' y el nombre de grupo 'users' son ficticios y en su sistema obtendrá otra cosa. Hay una columna de números a continuación de los permisos. Se trata de una columna que indica el número de enlaces rígidos que están asociados a un mismo fichero. En el caso de 'ej0', 'ej1', 'ej2' aparece un 3 y está claro porque son enlaces creados por nosotros pero el directorio 'dir1' tiene un 2. Esto significa que existe otro enlace rígido para ese directorio que nosotros no hemos creado. Se ha creado automáticamente al crear 'dir1'. Acuérdese que todos los directorios se crean con un par de entradas que son '.' y '..' El 2 por lo tanto en este caso se debe a la entrada '.' dentro del propio 'dir1' y si dentro de 'dir1' existieran directorios habría que contabilizar cada uno del los '..' de los directorios hijos como enlaces rígidos de 'dir1'. Un fichero se corresponde con un único inodo. Es decir tiene una única entrada en la tabla plana del sistema de ficheros, pero quizás aparezca varias veces en distintas partes del sistema de ficheros, o con distinto nombre. Si esto último no le ha quedado claro vuelva a leerlo después de finalizar este capítulo porque vamos a seguir explicando que

es un enlace rígido. Ahora retomamos la práctica en el punto donde la dejamos y continuamos.

```
$ cat ej0

xxxx

$ echo kkkkkkkkk > ej1
$ cat ej0

kkkkkkkkk

$ cat ejs1

kkkkkkkkk
```

Vemos que el contenido de los distintos enlaces con 'ej0' es idéntico, y si modificamos el contenido de cualquiera de ellos se afectará instantáneamente el contenido de los restantes. En realidad la información es accesible desde distintos nombres de ficheros pero no son copias sino que se trata de la misma unidad de información. Continuamos con el ejercicio.

```
$ rm ej0
$ cat ejs1

cat: ejs1: No existe el fichero o el directorio

$ cat ej1

kkkkkkkkk
```

Aquí ya vemos una diferencia. Pese a que 'ej1' se creó como un enlace de 'ej0', 'ej1' mantiene accesible la información incluso aunque desaparezca 'ej0'. Eso es porque en el caso de los enlaces rígidos da igual cual es el enlace o fichero original. Son totalmente equivalentes y la información solo desaparecerá del sistema cuando el último enlace rígido sea

eliminado. La diferencia con el enlace simbólico es que actúa simplemente accediendo al nombre del fichero que tiene almacenado en su interior. Por eso en el caso que acabamos de ver 'ejs1' queda apuntando a 'ej0' que es un fichero que ya no existe. Continuamos con el ejercicio y ahora usaremos una opción para 'ls' que no habíamos visto antes. Se trata de la opción -i que sirve para visualizar el número de inodo.

Un inodo es una clave numérica para el acceso al sistema plano de ficheros donde cada punto capaz de recibir o entregar información, tiene una única clave. Por eso los distintos enlaces rígidos contienen el mismo valor de inodo. Lo de recibir o entregar información se traduce en chorros de bytes producidos en operaciones de lectura escritura pero estas funciones pese a manejarse siempre igual realizan cosas muy distintas dependiendo del tipo de fichero. Un fichero regular es un almacén de información. Otros tipos de ficheros no son meros contenedores de bytes. Por ejemplo los dispositivos especiales de bloques o de caracteres pueden verse como emisores y receptores de bloques o caracteres respectivamente asociados a un dispositivo. Tampoco son meros contenedores de bytes los sockets, enlaces simbólicos, fifos con nombre, etc. Salvo los directorios, todos son capaces de recoger chorros de bytes o de entregar chorros de bytes o ambas cosas a la vez y lo más importante, todos ellos tienen asociado un inodo y al menos un nombre de fichero colgando de alguna parte de sistema de ficheros. Los directorios no se pueden manejar con funciones clásicas de lectura escritura. Tienen sus propias funciones específicas para manejo de directorios.

```
$ ls -li

  73449 drwxr-xr-x   2 pepe user 1024 may 18 17:28
dir1
  59173 lrwxrwxrwx   1 pepe user    4 may 18 17:28
dir2 -> dir1
  59171 -rw-r--r--   2 pepe user   10 may 18 17:30
ej1
  59171 -rw-r--r--   2 pepe user   10 may 18 17:30
ej2
  59172 lrwxrwxrwx   1 pepe user    3 may 18 17:28
```

```
ejs1 -> ej0
```

Como se puede ver 'ej1' y 'ej2' tienen el mismo valor de 59171 que en su sistema será otro valor cualquiera. En este momento después de borrar 'ej0' figuran con el valor 2 para el número de enlaces rígidos asociados. Vamos a mover el enlace simbólico 'ejs1' a 'dir1' pero vamos a usar 'mv ejs1 dir2' en lugar 'mv ejs1 dir1' pero debería dar lo mismo ya que 'dir2' es un enlace simbólico a 'dir1'.

```
$ mv ejs1 dir2
$ #
$ ## Comprobamos el resultado
$ ls -li

  73449 drwxr-xr-x   2 pepe user 1024 may 18 17:32
dir1
  59173 lrwxrwxrwx   1 pepe user    4 may 18 17:28
dir2 -> dir1
  59171 -rw-r--r--   2 pepe user   10 may 18 17:30 ej1
  59171 -rw-r--r--   2 pepe user   10 may 18 17:30 ej2

$ ls dir2

ejs1

$ ls -li dir1

  59172 lrwxrwxrwx   1 pepe user    3 may 18 17:28
ejs1 -> ej0
```

Hemos comprobado que un enlace simbólico se ha comportado igual que si fuera el propio directorio apuntado. En realidad podemos actuar a todos los efectos como si se tratara del verdadero fichero en lugar de un enlace simbólico salvo en el momento de su creación y en el momento de su destrucción. La operación 'rm' sobre un fichero simbólico no actúa sobre el fichero apuntado sino sobre el propio enlace simbólico destruyéndolo.

Ahora tenemos 'dir2/ejs1 -> ej0' pero 'ej0' ni siquiera existe. Vamos a cambiar el nombre de 'ej2' que era un enlace rígido de 'ej0' por 'ej0'.

```
$ mv ej2 ej0
$ cat dir2/ejs1

cat: dir2/ejs1: No existe el fichero o el directorio
```

Bueno este error es lógico porque el enlace 'dir2/ejs1' apunta a 'ej0' y no importa que estemos en el mismo directorio que 'ej0' sino que 'dir2/ejs1' y 'ej0' estén en el mismo directorio. Los enlaces son siempre relativos al sitio donde se encuentra el propio enlace. En caso contrario podría resultar graciosísimo pero poco práctico porque dependiendo de donde estuviéramos nosotros (más exactamente dependiendo del directorio actual del proceso que lo use) apuntaría a un directorio distinto cada vez.

De todas formas comprobémoslo trasladando 'ej0' a 'dir2', y observando como ha quedado todo.

```
$ mv ej0 dir2
$ cat dir2/ejs1

kkkkkkkkk

$ ls -li

  73449 drwxr-xr-x   2 pepe user 1024 may 18 17:34
dir1
  59173 lrwxrwxrwx   1 pepe user    4 may 18 17:28
dir2 -> dir1
  59171 -rw-r--r--   2 pepe user   10 may 18 17:30
ej1

$ ls -li dir2

  59173 lrwxrwxrwx   1 pepe user    4 may 18 17:28
dir2 -> dir1

$ rmdir dir2
```

```
rmdir: dir2: No es un directorio

$ rm dir2
$ ls -li

  73449 drwxr-xr-x   2 pepe user 1024 may 18 17:34
dir1
  59171 -rw-r--r--   2 pepe user   10 may 18 17:30
ej1

$ ls -li dir1

  59171 -rw-r--r--   2 pepe user   10 may 18 17:30
ej0
  59172 lrwxrwxrwx   1 pepe user    3 may 18 17:28
ejs1 -> ej0
```

Montage de dispositivos

Pueden existir varios sistemas de ficheros cada uno en un dispositivo o partición distinta, pero para hacerlo accesible ha de ser montado dentro del sistema principal de ficheros. Para ello se utilizan los comandos 'mount' y 'umount'.

Explicaremos esto un poco más. Supongamos que dispone de un disquete en el que desea guardar información accesible como parte del sistema actual. En MSDOS se accedería a la unidad que representa este dispositivo mediante el comando a: o b: por ejemplo. En Unix esto no ocurre porque ese patético invento llamado unidad lógica no existe.

Perdón por lo de patético pero es que en MSDOS, y en Windows cambias las cosas de sitio y terminas reinstalando todo.

Supongamos que tenemos en un dispositivo cdrom, cinta, disquete, o lo que sea la siguiente estructura de directorios.

```
.
|-- Guia-del-enROOTador-2.8
```

```
|-- curso
|-- glup_0.6-1.1-html-1.1
|-- lipp-1.1-html-1.1
|-- man_instal_debian_21
|-- novato-a-novato
`-- rhl-ig-6.0es
    |-- cpps
    `-- icons
```

Para poder acceder a esta información tendríamos que realizar una operación de montado sobre algún punto de nuestro sistema de ficheros actual. Por ejemplo si disponemos de un directorio '/misc/novato' podríamos montar en ese punto nuestro dispositivo y en ese caso la operación sería como enganchar un arbol en la rama de otro arbol. De esta forma para el sistema sería como si el árbol creciera.

```
.
`-- misc
    `-- novato
        |-- Guia-del-enROOTador-2.8
        |-- curso
        |-- glup_0.6-1.1-html-1.1
        |-- lipp-1.1-html-1.1
        |-- man_instal_debian_21
        |-- novato-a-novato
        `-- rhl-ig-6.0es
            |-- cpps
            `-- icons
```

Si el administrador considera que montar y un dispositivo no entraña riesgos para el sistema concederá permisos para que esto pueda ser realizado por los usuarios. Por ejemplo un administrador puede considerar que los usuarios solo puedan montar la unidad de cdrom.

En un entorno multiusuario y multiproceso no interesa hacer efectiva las actualizaciones sobre un dispositivo de forma instantánea sino que se intenta optimizar los accesos al dispositivo. Por eso si escribimos en un dispositivo de lectura escritura como por ejemplo un disquete y sacamos dicho disquete antes de desmontarlo lo más probable es que algunas operaciones de escrituras queden sin realizar porque estaban simplemente

guardadas en memoria a la espera de ser realizadas. Cuando se desmonta un sistema de ficheros se escriben las operaciones pendientes sobre ese dispositivo y entonces puede ser extraído del sistema sin peligro.

Directorio /proa

Este directorio es un directorio muy especial. Es un directorio en el cual está montado un sistema de ficheros virtual. En realidad muestra información que no reside en ningún dispositivo sino que es elaborada por el propio kernel. Por ello se trata de falsos archivos. Su ocupación en disco es 0. Sirve para comprobar la configuración y el funcionamiento del kernel. No entraremos ahora a comentarlo en detalle porque este tema es de interés para administradores del sistema. Lo que nos interesa comentar es que aquí la presentación en forma de sistema de ficheros es una simulación y se hace así para que pueda ser manejada exactamente de la misma forma que si realmente estuviera contenida en directorios y ficheros.

Algunos de estos ficheros son de solo lectura otros son de lectura y escritura y permiten cambiar la configuración del kernel sin detenerlo. Evidentemente todo ello está configurado con los permisos adecuados para mantener la seguridad del sistema.

Si miramos dentro del directorio '/proc' veremos que hay un montón de directorios cuyo nombre es un número. Estos directorios representan cada uno a un proceso por su pid. Ya hemos estudiado el proceso init en capítulos anteriores así que vamos a poner un ejemplo.

Pruebe a hacer lo siguiente:

```
$ cat /proc/1/status
```

Obtendrá la información de estado más importante del proceso init. Además de contribuir a la cultura general sobre nuestro sistema lo que

hemos mencionado sobre '/proc' nos sirve para introducir el siguiente tema.

Sistema de ficheros virtual

En Linux un sistema de ficheros puede corresponderse físicamente y lógicamente con cosas muy distintas. Acabamos de ver que el directorio '/proc' aparentemente está organizado como si fuera un sistema de ficheros idéntico a los que residen en disco duro y sin embargo se trata de algo totalmente distinto. Un sistema de ficheros puede residir, en memoria, en una partición de disco duro, en un dispositivo raid formado por varios discos duros funcionando en paralelo y con un sistema de redundancia, en disquetes, en cdroms, en un dispositivo remoto conectado a red, etc.. También se puede implementar un sistema de ficheros dentro de un fichero. Por ejemplo umsdos es un sistema de ficheros linux implementado dentro de un fichero msdos. Realmente es una solución muy poco eficiente pero pone de relieve la flexibilidad del sistema virtual de ficheros. Un sistema de ficheros puede tener un formato interno que no siempre es el mismo. Linux puede manejar sistemas de ficheros de otros sistemas operativos. En resumen lo que Linux ofrece con su sistema ficheros virtual (VFS) es un sistema unificado de acceso a toda una variedad de recursos muy distintos. Esto es posible porque se definen una serie de funciones para manejar un sistema de ficheros genérico. En Unix cada sistema de ficheros tiene asociado un sistema plano de ficheros. Las estructuras pueden ser diferentes para cada tipo de sistema de ficheros, pero las funciones que lo manejan quedan unificadas por el (VFS)

Estructura estandar del sistema de ficheros de Linux

Existe un documento 'fsstnd' donde se describe una recomendación para estandarizar la estructura del sistema de ficheros de Linux para las distintas distribuciones. Nosotros vamos a resumir brevemente la información más importante. Un detalle mayor sería necesario si este curso fuera un curso de administración del sistema.

Este último capítulo no explica conceptos nuevos pero servirá para que comprenda lo que tiene en su sistema. Puede investigar todo lo que quiera por su cuenta siempre que use un usuario normalito sin privilegios. Si alguna vez se siente perdido solo tiene que introducir el comando 'cd' para volver a casita.

Puede haber diferencias importantes en la estructura general del sistema de ficheros entre unas distribuciones y otras pero en líneas generales el aspecto de la disposición de los directorios más importantes sería más o menos el siguiente;

```
.
|-- bin
|-- sbin
|-- tmp
|-- boot
|-- dev
|-- etc
|-- home
|-- lib
|   |-- modules
|   `-- security
|-- home
|   |-- ftp
|   |-- httpd
|   |-- luis
|   |-- msql
|   |-- pili
|   `-- skeleton
|-- root
|-- usr
|   |-- bin
|   |-- sbin
|   |-- X11R6
|   |   |-- bin
|   |   |-- include
|   |   |-- lib
|   |   `-- man
|   |-- local
|   |   |-- bin
|   |   |-- doc
|   |   |-- man
|   |   |-- lib
```

```
|   |   |-- src
|   |   `-- tmp
|   |-- doc
|   |-- include
|   |-- info
|   `-- src
|       `-- linux
|-- proc
`-- var
    |-- cache
    |-- catman
    |-- lib
    |-- local
    |-- lock
    |-- log
    |-- run
    |-- spool
    |   |-- cron
    |   |-- lpd
    |   |-- mail
    |   `-- mqueue
    `-- tmp
```

Si observa diferencias con la estructura de ficheros de su distribución no debe preocuparse esto es solo un ejemplo. Comentaremos el cometido de los directorios más significativos.

El directorio raíz

Para arrancar el sistema, debe estar presente lo suficiente como para montar '/usr' y otras partes no-esenciales del sistema de archivos. Esto incluye herramientas, información de configuración y del cargador de arranque (boot loader) y alguna otra información esencial al arrancar.

Para habilitar la recuperación y/o la reparación del sistema, estará presente en el sistema de archivos raíz aquellas herramientas que un administrador experimentado necesitaría para diagnosticar y reconstruir un sistema dañado.

Los errores del disco, que corrompen la información en el sistema de archivos '/' son un problema mayor que los errores en cualquier otra

partición. Un sistema de archivos '/' (raiz) pequeño es menos propenso a corromperse como resultado de un fallo del sistema.

La principal preocupación que se usa para balancear las anteriores consideraciones, que favorecen el colocar muchas cosas en el sistema de archivos raíz, es la de mantener '/' (raiz) tan pequeno como sea razonablemente posible.

/ --- El Directorio Raíz

```
bin             Binarios de comandos esenciales
boot            Archivos estáticos de cargador de
arranque(boot-loader)
dev             Archivos de dispositivos
etc             Configuración del sistema local-máquina
home            Directorios home de los usuarios
lib             Librerías compartidas
mnt             Punto de montaje de particiones temporales
root            Directorio hogar del usuario root
sbin            Binarios del sistema esenciales
tmp             Archivos temporales
usr             Segunda jerarquía mayor
var             Información variable
```

La jerarquía /usr.

'/usr' es la segunda mayor sección del sistema de archivos. '/usr' es información compartible, de solo lectura, esto significa que '/usr', debe ser compartible entre varias maquinas que corren LINUX y no se debe escribir. Cualquier información que es local a una máquina o varía con el tiempo, se almacena en otro lugar.

Ningún paquete grande (como TeX o GNUEmacs) debe utilizar un subdirectorio directo bajo '/usr', en vez, debe haber un subdirectorio dentro de '/usr/lib' (o '/usr/local/lib' si fué instalado completamente local) para ese propósito, con el sistema X Window se hace una excepción

debido a un considerable precedente y a la práctica ampliamente aceptada.

/usr --- Segundo mayor punto de montaje (permanente)

```
X11R6           Sistema X Window Version 11 release 6
X386            Sistema X Windows Version 11 release 5 en
plataformas  X 86
bin             La mayoría de los comandos de usuario
dict            Listas de palabras
doc             Documentación miscelánea
etc             Configuración del Sistema (todo el site)
games           Juegos y binarios educacionales
include Archivos header incluidos por programas C
info            Directorio primario del sistema GNU Info
lib             Librerías
local           Jerarquía local (vacía justo después de la
instalación principal)
man             Manuales en línea
sbin            Binarios de Administración del Sistema No-
Vitales
share           Información independiente de la
arquitectura
src             Código fuente
```

/usr/local: Jerarquía local

La jerarquía '/usr/local' está para ser utilizada por el administrador del sistema cuando se instale el software localmente. Necesita estar a salvo de ser sobrescrito cuando el software del sistema se actualiza. Puede ser usado por programas y por información que son compartibles entre un grupo de máquinas , pero no se encuentran en '/usr'.

/usr/local   Jerarquía local.

```
bin             Binarios solo-locales
doc             Documentación local
etc             Binarios de configuración solo-local
games           Juegos instalados localmente
lib             Librerías para /usr/local
info            Páginas de info local
man             Jerarquías de páginas de manual para
/usr/local
```

```
sbin            Administración del sistema solo-local
scr             Código fuente local.
```

Este directorio debe estar vacío al terminar de instalar LINUX por primera vez. No debe haber excepciones a la regla, excepto quizá los subdirectorios vacíos listados.

La Jerarquía /var

El sistema necesita con frecuencia una zona de trabajo temporal. Si solo se requiere usar un espacio por un corto periodo de tiempo se puede usar '/tmp', pero muchas veces la información conviene manejarla y almacenarla en un lugar más permanente. El sistema puede sufrir una caída repentina y el contenido de '/tmp' puede ser borrado durante el arranque o depurado regularmente mediante alguna tarea periódica. Por el contrario '/var' contiene todo tipo de información alguna de ella puede ser importante. Por ejemplo información vital para el mantenimiento de la gestión de paquetes de la distribución o mensajes pendientes de ser enviados por correo, archivos y directorios en fila de ejecución, información de bitácora administrativa y archivos temporales y transitorios aunque se asume que su permanencia será mayor que '/tmp'

/var Información variable

```
        catman          Páginas del manual formateadas
localmente
        lib             Información del estado de
aplicaciones
        local           Información variable del software de
/usr/local
        lock            Archivos de bloqueo
        log             Archivos de bitácora
        named           Archivos DNS, solo red
        nis             Archivos base de datos NIS
        preserve        Archivos almacenados después de una
falla de ex o vi
        run             Archivos relevantes a procesos
ejecutándose
        spool           Directorios de trabajos en fila para
realizarse después
```

```
        tmp             Archivos temporales, utilizado para
mantener /tmp pequeño
```

ALGUNOS COMANDOS ÚTILES

Tenemos una excelente noticia para sus neuronas. En este capítulo no introduciremos conceptos nuevos.

Le proporcionaremos una lista alfabética de comandos que puedan resultarle de utilidad. Entre estos comandos no encontrará comandos internos de la shell, ni comandos relacionados con la programación o administración de su sistema. Solo incluiremos los comandos de usuario de uso más frecuente. En el tema de comunicaciones solo mencionaremos unos pocos comandos básicos.

Nos limitaremos a indicar brevemente para que se usan. En algunos de ellos pondremos algún ejemplo pero los detalles tales como forma de uso y opciones deben consultarse en el manual. El objetivo es simplemente que conozca la existencia de los comandos que pueden resultarle de mayor utilidad en este momento.

Recuerde que un número entre paréntesis a continuación de un comando indica el número de sección en el man.

Dado que no vamos a mencionar en este momento ningún comando interno del interprete de comandos bash lo que si podemos indicarle es la forma de saber cuales son y como obtener información sobre cualquiera de ellos. Bastará con teclear el comando 'help'. Si quiere obtener más información sobre cualquiera de ellos teclee 'help comando'.

Ejemplo:

```
$ help | less
$ help cd
```

Dijimos que no explicaríamos ningún comando interno pero en realidad 'help' es un comando interno y tenemos que añadir que admite el uso de meta caracteres.

```
$ help help
$ help *alias
```

No se va a estudiar en detalle ningún comando en este capítulo ya que se trata de tener una visión de conjunto. Por ello cuando lleguemos a un comando que merezca ser explicado con mayor profundidad nos limitaremos a decir para que sirve y le dejaremos con la miel en los labios con un oportuno "Se estudiará más adelante".

Selección de los comandos externos de usuario más útiles.

apropos at atq atrm *awk* banner batch bc cal cat chgrp chmod chown cksum clear cp *cpio* cut date df diff du echo *egrep* emacs env ex expr false *fgrep* file *find* finger free ftp fuser *gawk* *grep* gzip head hostname id info ispell kill killall less ln locate lpq lprm ls mail man mkdir more mv nice nohup passwd paste pr ps pstree pwd renice reset rm rmdir rsh script *sed* sleep *sort* split stty su tail talk tee telnet *test* tload top tr true *vi* w wc whatis whereis who whoami write xargs zcat zdiff zgrep zless zmore

Los comandos awk, cpio, egrep, fgrep, find, gawk, grep, sed, sort, test, y vi serán tratados en detalle en capítulos posteriores.

apropos(1)

Ya comentamos este comando en el capítulo dedicado al manual online de unix. Sirve para ayudar a localizar un comando que no sabemos exactamente como se escribe. Solo se usa para la búsqueda la descripción corta que figura al principio de la página del manual y hay que usar palabras completas y no secuencias de caracteres. Por ejemplo si nuestro manual está en ingles podríamos buscar compresores con:

```
$ apropos compress
```

at(1)

Se utiliza para programar un comando para que se ejecute en un determinado momento. Este comando no admite intervalos de tiempo menores a un minuto. La salida deberá estar redirigida ya que se ejecutará en segundo término sin un terminal asociado. Una vez ejecutado este comando devuelve un número de tarea y el momento en el cual se activará el comando. Ejemplos:

```
$ echo 'date > /tmp/salida' | at now + 1 minute
$ echo 'date > /tmp/salida' | at 8:15am Saturday
$ echo 'date > /tmp/salida' | at noon
$ echo 'echo feliz año nuevo > /dev/console' | at
11:59 Dec 31
$ echo 'banner a comer > /dev/console' | at 1:55pm
```

atq(1)

Permite ver la cola de tareas pendientes.

```
$ atq
```

atrm(1)

Permite cancelar una tarea programada y pendiente de ejecutarse más tarde.

```
$ atrm 188
```

awk(1)

Se trata de un programa que implementa un lenguaje para tratar ficheros de texto estructurados en campos y registros. Es muy potente y se estudiará más adelante.

banner(1)

Para generar cabeceras con letras grandes. Ejemplo:

```
$ banner hola

 #    #   ####   #         ##
 #    #  #    #  #        #  #
 ######  #    #  #       #    #
 #    #  #    #  #       ######
 #    #  #    #  #       #    #
 #    #   ####   ######  #    #
```

batch(1)

Es un comando muy similar al comando at pero no requiere la indicar el momento en ese caso se ejecutará en un momento con baja carga. Ejemplo:

```
$ echo 'date > /dev/console' | batch
```

bc(1)

Es una calculadora de precisión arbitraria. (Es decir con toda la precisión que uno quiera) Se puede usar interactivamente o admitir comandos por la entrada estándar. Admite expresiones bastante complejas y sentencias condicionales y de repetición que constituyen un potente lenguaje de programación para esta calculadora. Por ejemplo calcular el número PI con 300 cifras decimales.

```
$ echo "scale=300; 4*a(1)" | bc -l

3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651328230664709384460955058\
22317253594081284811174502841027019385211055596446229489549303819644\
28810975665933446128475648233786783165271201909145648566923460348610\
454326648213393607260249141272
```

cal(1)

Calendario Ejempos:

```
$ cal

$ cal 2000
```

cat(1)

Lee uno o más ficheros y los saca por la salida estándar.

```
$ cat fichero1 fichero2 > fichero1_mas_2
```

chgrp(1)

Permite cambiar el atributo de grupo de uno o más ficheros. Solo el propietario del fichero o un usuario privilegiado puede usarlo.

chmod(1)

Permite el cambio de los permisos asociados a un fichero. Solo el propietario del fichero o un usuario privilegiado puede usarlo.

```
$ chmod +r /tmp/fich
$ chmod u+r /tmp/fich
$ chmod 770 /tmp/fich
$ chmod a-wx,a+r /tmp/fich
```

chown(1)

Permite cambiar el propietario actual de un fichero. Solo el propietario del fichero o un usuario privilegiado puede usarlo. También permite cambiar el grupo.

```
$ chown usuario fichero
$ chown usuario:grupo fichero
```

cksum(1)

Obtiene un código (CRC) que está diseñado para detectar variaciones en la información por problemas en la grabación o transmisión de datos.

```
$ cksum fichero
```

clear(1)

Limpia el contenido de la consola.

cp(1)

Copia ficheros. Si se utiliza con un par de argumentos tomará el inicial como origen y el final como destino. Si el segundo argumento es un fichero entonces sobreescribirá su contenido. Si se usa con más de un argumento el último ha de ser obligatoriamente un directorio.

```
$ cp fich1 fichdest
$ cp fich1 dirdest
$ cp fich1 fich2 fich3 dirdest
```

cpio(1)

Admite una lista de nombres de ficheros para empaquetarlos en un solo fichero. Es un comando muy potente y versátil. Junto con tar se usa entre

otras cosas para hacer copias de seguridad. Suele usarse mucho en combinación con find. Lo estudiaremos más adelante.

cut(1)

Permite extraer columnas o campos desde uno o más ficheros.

```
$ cut -d: -f1 /etc/passwd
$ cut -d: -f6 /etc/passwd
```

date(1)

Obtiene la fecha. Las opciones permiten obtener distintos formatos de salida.

```
$ date

dom jun 11 18:17:14 CEST 2000

$ date +"%Y/%m/%d %T"

2000/06/11 18:16:49

$ date +%s

960740283
```

df(1)

Informa de la utilización de disco en las particiones que están montadas.

```
$ df -a
```

diff(1)

Sirve para localizar diferencias entre dos ficheros de texto.

```
$ diff fich1 fich2
```

du(1)

Permite averiguar la ocupación de espacio de todo aquello que cuelga de un determinado directorio. Este comando conviene usarlo de vez en cuando para localizar directorios demasiado cargados de información.

```
$ du -s .
$ du .
$ du -s * | sort -n
```

echo(1)

Permite sacar mensajes por salida estándar.

egrep(1)

Es una variedad del comando grep que permite el uso de expresiones regulares extendidas. Sirven para buscar cadenas de texto o secuencias de caracteres en ficheros. Se estudiarán más adelante.

emacs(1)

Es un editor multipropósito.

env(1)

Obtiene la lista de variables de entorno y sus valores.

ex(1)

Es un editor interactivo similar al vi. En Linux existe una versión mejorada llamada elvis.

expr(1)

Es un evaluador de expresiones.

```
$ expr \( 55 + 31 \) / 3
```

false(1)

Solo tiene sentido usarlo en programación y retorna un código de retorno que indica error. En otras palabras no solo no hace nada sino que además siempre lo hace mal. Parece absurdo pero tiene su utilidad. Si no fuera sí no existiría y no es el único comando de este tipo.

fgrep(1)

Es una variedad del comando grep. Sirven para buscar cadenas de texto o secuencias de caracteres en ficheros. Se estudiarán más adelante.

file(1)

Sirve para averiguar el tipo de fichero pasado como argumento. Distingue muchos tipos de ficheros.

```
$ file fich
$ file dir1 fich2
```

find(1)

Permite localizar ficheros dentro de la estructura de directorios. Es tremendamente versátil y se estudiará más adelante.

```
$ find /var -name '*.log'
```

finger(1)

Sirve para averiguar quien está conectado al sistema.

```
$ finger
$ finger -l
```

free(1)

Proporciona estadísticas de uso de la memoria del sistema.

ftp(1)

Comando para intercambiar ficheros entre distintos ordenadores.

fuser(1)

Indica que proceso está usando un fichero o un directorio.

```
$ fuser /
```

gawk(1)

Es la versión GNU de awk. Se trata de un programa que implementa un lenguaje para tratar ficheros de texto estructurados en campos y registros. Es muy potente y se estudiará más adelante.

grep(1)

Junto a egrep, fgrep y rgrep sirven para buscar cadenas de texto o secuencias de caracteres en ficheros. Se estudiarán más adelante.

gzip(1)

Compresor des-compresor de gnu. Se puede usar directamente sobre un fichero o por el contrario puede usarse como filtro. Cuando se usa directamente sobre un fichero se modificará el nombre del fichero añadiendo .gz para indicar que está comprimido o se eliminará cuando sea descomprimido.

```
$ ls -l / > directorio_raiz
$ gzip directorio_raiz
$ gzip -d directorio_raiz.gz
$ gzip < directorio_raiz > directorio_raiz.gz
$ gzip -d < directorio_raiz.gz > directorio_raiz
```

head(1)

Permite sacar las primeras líneas de un fichero.

hostname(1)

Obtiene el nombre de la máquina.

id(1)

Devuelve información de identidad de aquel que lo ejecuta.

info(1)

Es un lector de hipertexto de GNU. Muchos programas de GNU incluyen documentación en formato info. Los hipertextos son textos que incluyen enlaces sub-menús, y otros elementos que facilitan una lectura no secuencial. Por ejemplo el html es otro formato de hipertexto.

ispell(1)

Es un comprobador ortográfico que puede ser acoplado a diversos editores como por ejemplo al vi. Existen diccionarios ortográficos para distintos lenguajes.

kill(1)

Envía una señal a un proceso. Se llama kill porque generalmente se usa para "matar" procesos. Requiere que se conozca el identificador del proceso PID.

killall(1)

Como el anterior pero permite pasar el nombre del comando y "matará" a todos los procesos que tengan ese nombre.

---

less(1)

Es un paginador que puede usarse en sustitución del paginador more.

```
$ less fichero
$ cat fichero | less
```

ln(1)

Crea enlaces rígidos y enlaces simbólicos que son cosas que ya hemos estudiado.

locate(1)

Permite la localización rápida de ficheros en su sistema de ficheros. Se utiliza una base de datos que debe de ser actualizada regularmente

mediante el comando updatedb. Este último lo realizará root cuando lo crea oportuno o mejor aun estará programado en el sistema para que se ejecute periódicamente. La idea es la siguiente. Para localizar un fichero se suele ejecutar el comando find capaz de explorar todo el árbol del sistema de ficheros pero esta operación es muy costosa porque requiere abrir y cerrar cada uno de los directorios en los que busca información. El comando updatedb hace esto mismo pero guardando el resultado de toda la exploración en una base de datos muy compacta donde se puede buscar con enorme rapidez. La desventaja es que la información de búsqueda tendrá cierta antigüedad y que no se guardan todos los atributos del fichero por lo que resulta imposible buscar usando criterios para esos atributos. Por ello usaremos locate para ciertas búsquedas aunque algunas veces tendremos que usar find.

lpq(1)

Informa sobre el estado de la cola de impresión. Muestra las tareas que están el la cola su identificador numérico, orden, tamaño y propietario.

lprm(1)

Permite cancelar tareas de impresión usando el identificador numérico obtenido con lpq.

ls(1)

Obtiene un listado de los ficheros de un directorio. Tiene una gran cantidad de opciones.

mail(1)

Se usa para enviar un correo electrónico. Su manejo comparado con otros programas de correo electrónico no resulta muy amigable pero no solo sirve para usarlo en modo interactivo. Se le puede pasar por la entrada estándar el contenido de un mensaje. Una aplicación de esto es que podemos coleccionar mensajes cortos por ejemplo para suscribirse o de-suscribirse de listas de correo.

```
$ echo unsubscribe l-linux | mail
majordomo@calvo.teleco.ulpgc.es

$ mail -s unsubscribe debian-user-spanish-
request@lists.debian.orga < null>
```

man(1)

Este fué el primer comando que se estudió en este curso y no nos cansaremos de recomendarle que se familiarice con el. La información de estas páginas del manual no suele resultar muy didáctica. En Linux no vienen apenas ejemplos. A pesar de ello suele traer información completa de opciones y formas de uso.

mkdir(1)

Crea uno o más directorios. Los permisos dependerán de el valor actual de 'umask'. Esto ya lo vimos en capítulos anteriores. Para eliminar un directorio se usará 'rmdir'.

```
$ mkdir dir1 dirA/dirB/dirC/dir2 dir3
```

more(1)

Es un paginador mucho más sencillo que el sofisticado 'less'.

```
$ ls | more
$ more fichero
```

mv(1)

Se utiliza para renombrar directorios y ficheros o para trasladarlos de un lugar a otro. Conviene usarlo con precaución porque se presta a obtener resultados diferentes en función de la existencia o no de un fichero o directorio de destino. Mover un fichero a otro que ya existe supondría sobrescribirlo así que un error puede provocar perdida de información. Cuando se usan más de dos argumentos en Linux se obliga a que el último sea un directorio de destino lo cual previene errores.

```
$ mv fich1 fich01
$ mv dir1 dir01
$ mv fich1 fich2 dir1 dir2 fich3 fich4  dir001
```

nice(1)

A un usuario normal le permitirá bajar la prioridad de los proceso lanzados con este comando. Solo root puede usarlo para aumentar la

prioridad. Una vez que un comando ha sido lanzado se puede modificar su prioridad con renice.

```
$ nice comando
```

nohup(1)

Los comandos lanzados con nohup no terminan al abandonar la sesión. Se puede usar combinado con la ejecución en background.

```
$ nohup GeneraInformeMensual > informe01-05-2000.out &

$ # logout
$ exit
```

passwd(1)

Permite cambiar nuestra clave de acceso al sistema. Conviene no olvidar la clave. Generalmente para evitar olvidos los usuarios eligen claves demasiado sencillas. Dependiendo del nivel de seguridad configurado en el sistema, este podría rechazar claves demasiado sencillas. No conviene usar claves cortas ni palabras que puedan ser localizadas en un diccionario ni fechas o nombres relacionadas con datos de alto significado personal. Palabras escritas al revés tampoco son seguras. Intercalar algún carácter de puntuación alguna mayúscula o algún dígito suele ser una buena práctica. foTo;21 pon.5.mar 7li-bRos Bueno si el ordenador es de uso personal y solo tienen acceso personas de confianza tampoco hay que ser demasiado paranoico.

paste(1)

Se usa para combinar columnas de distintos ficheros en uno solo. Viene a ser lo contrario del comando cut.

```
$ who | paste - -
```

pr(1)

Permite paginar un texto incorporando cabeceras.

```
$ ls /*/* | pr -h ejemplo-pr -o 5 -l 35 | less
```

ps(1)

Permite obtener información de los procesos en ejecución. Dependiendo lo que nos interese existen diversas opciones para una gran variedad de formatos y de selección de la información de salida.

pstree(1)

Muestra la jerarquía entre procesos mostrando las relaciones de parentesco entre ellos.

```
$ pstree
$ pstree -p
```

La opción -p es muy útil para averiguar el pid de un proceso.

pwd(1)

Este comando retorna el lugar donde nos encontramos.

```
$ pwd
```

Se puede usar por ejemplo para guardar el lugar donde estamos ahora con objeto de retornar ala mismo lugar más tarde.

```
$ AQUI=`pwd`
$ cd /tmp
$ ls
$ cd $AQUI
```

renice(1)

Es parecido a nice pero actúa sobre un proceso que ya fue arrancado con anterioridad. Por ejemplo hemos lanzado un proceso que consume muchos recursos y lleva mucho tiempo. En lugar de pararlo podemos bajarle la prioridad para que gaste muchos recursos. (Cuando decimos recursos nos referimos a uso de memoria, uso de CPU, etc) Un super usuario como root puede incrementar la prioridad usando un número negativo. Un usuario normal solo podrá decrementar la prioridad usando un número positivo. En cualquier caso se requiere conocer el pid del proceso que deseamos modificar y solo podremos hacerlo si es un proceso nuestro. Por ejemplo vamos a suponer que el PID de ese proceso que deseamos bajar de prioridad es el 778.

```
$ renice +15 778
```

reset(1)

Puede ocurrir que el terminal quede des-configurado por alguna razón. Esto se debe a que un terminal interpreta comandos en forma de secuencias de caracteres. Esto permite mostrar colores manejar el cursor y una serie de cosas más. Cuando accidentalmente enviamos al terminal un fichero binario que contiene caracteres de todo tipo en cualquier orden, es bastante normal que el terminal quede inutilizable. Muchas veces se puede recuperar introduciendo el comando reset.

rm(1)

Ya hemos explicado y usado este comando. Sirve para borrar ficheros.

rmdir(1)

Ya hemos explicado y usado este comando. Sirve para eliminar directorios que deben de estar vacíos.

rsh(1)

Permite siempre que tengamos permiso ejecutar un comando en un ordenador distinto. Se puede usar para transferir grandes cantidades de información a través de la red. Por ejemplo imaginemos que queremos sacar una copia de seguridad guardando la información en una unidad de cinta que se encuentra en un ordenador distinto. En ese caso se lanzan dos comandos simultáneos comunicándolos con un pipe. Un comando se

ejecutará en el ordenador remoto con rsh y el otro se ejecuta en local. Ambos procesos quedan comunicados por entrada salida pero eso implica que esa comunicación viajará a través de la red.

```
$ ### Para salvar
$ tar cf - . | rsh remotehost dd of=/dev/tape

$ ### Para recuperar
$ rsh remotehost dd if=/dev/tape | tar xf -
```

La forma de combinar comandos en unix conectando entrada salida permite hacer muchas veces cosas sorprendentes como en este caso. Existen un comando similar que hace lo mismo pero la información viaja encriptada a través de la red. Se trata de 'ssh'.

script(1)

Este es un comando realmente curioso. Su utilidad principal es grabar una sesión. Lanza una subshell que se ejecutara en un pseudo-terminal. Este palabro no deseamos explicarlo ahora pero mientras estamos en esa subshell todos los caracteres recibidos por el terminal son grabados en un fichero. Para terminar se teclea exit. Es útil para memorizar sesiones de trabajo.

sed(1)

Este es un editor de flujo. Dicho de otra forma es un editor que está pensado para usarlo como flujo. De esta forma se puede usar como una poderosa herramienta para transformar texto. Se estudiará más adelante.

sleep(1)

Sirve para esperar sin consumir recursos. El proceso queda dormido durante el número de segundos indicado.

```
$ echo hola; sleep 5 ; echo que tal
```

sort(1)

Este comando permite ordenar lineas de texto. Se puede usar como filtro. Se estudiará más adelante.

split(1)

Este comando sirve para partir en trozos más pequeños un fichero grande. Para volver a obtener el fichero original bastará con usar el comando 'cat'

```
$ split --bytes=52m ficherogrande parte
```

Este comando trocea un fichero grande en trozos de 52Mbytes que se guardan en ficheros que empiezan con el prefijo 'parte' seguido de una numeración secuencial.

stty(1)

Sirve para comprobar el estado actual del terminal y para cambiar la configuración del mismo. Se puede cambiar el significado de algunos caracteres de control, establecer el número de filas y columnas del terminal, la velocidad de transmisión, etc. Para comprobar el estado actual teclee lo siguiente.

```
$ stty -a
```

su(1)

Permite cambiar de usuario. Lo que se hace es lanzar una subshell. Hay dos formas de hacerlo. 'su nuevousuario' y 'su - nuevousuario'. Si no usamos la segunda forma solo cambiará el usuario efectivo pero el entorno del proceso se mantiene. Si por el contrario se usa la segunda forma se ejecutarán los scripts de inicio de sesión para el nuevo usuario y tendremos una sesión idéntica a la obtenida con la entrada con login para ese nuevo usuario. Para terminar esa sesión bastará hacer exit.

```
$ su nuevousuario
$ su - nuevousuario
```

Para un usuario normal se solicitará la password del nuevo usuario.

tail(1)

Así como head servía para obtener las primeras líneas de un fichero tail sirve para obtener las últimas líneas de un fichero

```
$ tail fichero
```

Tail tiene una opción muy útil para mostrar que es lo que está pasando en ficheros que se están generando o que crecen continuamente.

```
$ tail -f fichero
```

Este comando no termina nunca y muestra el final del fichero quedando a la espera de mostrar el resto de mismo a medida que se genere. Para terminar hay que matarlo con Ctrl-C.

talk(1)

Permite abrir una sesión de charla interactiva con otro usuario.

tee(1)

Este comando se utiliza para obtener una bifurcación en un flujo de entrada salida. Actúa como una derivación en forma de 'T'.

```
$ ll | tee todos-fich | tail > ultimos-fich
```

telnet(1)

Permite abrir una sesión de trabajo en otra máquina.

```
$ telnet localhost
```

Con esto podemos probar telnet conectándonos con nuestra propia máquina. A continuación saldrá la solicitud de login.

test(1)

Se usa para evaluar expresiones. Se usa mucho en la programación shell-script. Lo que se usa es su código de retorno. Tiene dos formas de uso 'test expresión' o '[expresión]'

```
$ test "2" = "3"
$ echo $?

$ test "2" = "2"
$ echo $?

$ A=335
$ B=335

$ [ "$A" = "$B" ]
$ echo $?
```

Se estudiará más adelante.

tload(1)

Convierte el terminal en un monitor semi-gráfico que indicará el nivel de carga del sistema. Se interrumpe con Ctrl-C. Si el sistema no está trabajando demasiado no verá nada demasiado interesante.

```
$ ls /*/* > /dev/null 2>&1  &
$ tload -d 2
```

top(1)

Muestra información de cabecera con estadísticas de uso de recursos. (número de procesos y en que estado están, consumo de CPU, de memoria, y de swap). Además muestra información muy completa sobre los procesos que están consumiendo más recursos. Si el ordenador va muy lento por exceso de trabajo podemos hacernos una idea muy buena de los motivos usando este comando. En realidad este comando es muy útil para el administrador del sistema pero los usuarios que comparten el uso de un sistema con otros usuarios también tienen que usar los recursos del sistema de forma inteligente bajando la prioridad de tareas poco urgentes y que consuman mucho. En especial no resulta buena idea lanzar muchas tareas pesadas de forma simultanea aunque sean urgentes porque se perjudica el rendimiento global del sistema que gastará excesivo tiempo en labores improductivas de gestión de procesos.

```
$ top
```

Para salir hay que pulsar 'q'.

tr(1)

Se utiliza para sustituir carácter.

```
$ # Pasar a mayúsculas
$ tr '[a-z]' '[A-Z]' < fichero > nuevofichero

$ # Eliminar el carácter ':'
$ tr -d : < fichero > nuevofichero
```

true(1)

Este comando es el contrario del comando 'false'. Solo tiene sentido usarlo en programación y retorna un código de retorno que indica éxito. En otras palabras no hace pero al menos lo hace correctamente.

vi(1)

Este es un editor muy potente y presente en todos los sistemas de tipo Unix. Se estudiará más adelante.

w(1)

Muestra información de usuarios actualmente conectados mostrando momento de inicio de sesión y el consumo de CPU que se ha realizado.

```
$ w
```

wc(1)

Este es un comando que se usa bastante. Sirve para contar caracteres, palabras y líneas en un fichero o en un flujo de entrada salida.

```
$ wc fichero
$ cat fichero | wc
```

whatis(1)

Sirve para mostrar la breve descripción de un comando que figura en las páginas del manual

```
$ whatis ls
$ whatis whatis
```

who(1)

Saca la información de quienes están conectados al sistema.

```
$ who
```

whoami(1)

Para averiguar quien es usted en ese momento. En un programa puede ser interesante comprobar quien lo está ejecutando.

```
$ whoami
```

whereis(1)

Sirve para localizar un comando en el sistema siempre que este esté localizado en $PATH

```
$ whereis ls
$ whereis whereis
```

write(1)

Siempre que se tenga permiso permite enviar mensajes a otro usuario.

```
$ write root
```

xargs(1)

Sirve para pasar argumentos a un comando mediante entrada salida.

```
$ echo '-l' | xargs ls
```

Cosas como estas añaden bastante potencia al lenguaje shell-script.

z...

Hay una serie de comandos. 'zcat, zmore, zgrep, zless, zdiff' que permiten trabajar directamente con ficheros comprimidos con gzip en la misma forma que lo haríamos directamente con 'cat, more, grep, less y diff' respectivamente.

```
$ zless documento.txt.gz
```

EXPRESIONES REGULARES

Introducción

Vamos a explicar las expresiones regulares porque se utilizan a menudo desde una gran variedad de aplicaciones en los SO tipo Unix como Linux. Permiten reconocer una serie de cadenas de caracteres que obedecen a cierto patrón que llamamos expresión regular. Por ejemplo si deseamos buscar lineas que contenga las palabras 'hola' o 'adiós' en los ficheros del directorio actual haremos:

```
$ egrep 'hola|adiós' *
```

No todos los comandos usan de forma idéntica las expresiones regulares. Algunos de los comandos que usan expresiones regulares son 'grep', 'egrep', 'sed', 'vi', y 'lex'. Este último en linux se llama 'flex' y es un analizador sintáctico muy potente pero no lo explicaremos porque para usarlo hay que saber lenguaje 'C'. Actualmente algunos lenguajes modernos como el 'perl' incluyen capacidad de manejar expresiones regulares lo cual les da una gran potencia y para lenguajes más antiguos como el 'C' existen librerías para poder usar expresiones regulares. En resumen las expresiones regulares están sinedo incorporadas en distintos sitios y ya no están limitadas a su uso en SO tipo Unix. Cada comando o aplicación implementa la expresiones regulares a su manera aunque en general son todas bastantes parecidas. Por ejemplo 'grep' permite usar expresiones regulares sencillas mientras que 'egrep' es capaz de usar expresiones regulares más complejas. Otros comandos adaptan el uso de expresiones a sus particulares necesidades y por ello si bien se puede hablar en general de ellas hay que tener en cuenta las peculiaridades de cada caso que deberán ser consultadas en las paginas del manual de cada comando. Las expresiones regulares vienen a ser una especie de lenguaje y cada comando usa su propio dialecto. En realidad las diferencias entre

los distintos dialectos suelen ser muy pocas. Por ejemplo si un comando usa los paréntesis y además admite el uso de expresiones regulares extendidas se establecerá una forma de distinguir si los paréntesis deben ser interpretados como patrón de la expresión regular o como otra cosa. Para ello se suele usar los paréntesis precedidos del carácter escape '\'. Vamos a tomar a 'egrep' y 'sed' como comandos para aprender expresiones regulares porque este curso tiene un enfoque práctico. Usaremos el comando 'egrep' con distintos patrones y veremos cuando cumple y cuando no cumple y de esa forma se entenderá perfectamente.

Operadores usados en expresiones regulares.

*****	El elemento precedente debe aparecer 0 o más veces.
+	El elemento precedente debe aparecer 1 o más veces.
.	Un carácter cualquiera excepto salto de linea.
?	Operador unario. El elemento precedente es opcional
|	O uno u otro.
^	Comienzo de linea
$	Fin de linea
[...]	Conjunto de caracteres admitidos.
[^...]	Conjunto de caracteres no admitidos.
-	Operador de rango
(...)	Agrupación.
****	Escape
\n	Representación del carácter fin de linea.
\t	Representación del carácter de tabulación.

Esta lista no es completa pero con esto es suficiente para hacer casi todo lo que normalmente se hace con expresiones regulares.

Ejemplos para cada operador con 'egrep'

Empezaremos usando un ejemplo lo más sencillo posible para ilustrar cada uno de estos operadores

Ejemplo para el operador '*' con el patrón 'ab*c'

Este patrón localizará las cadenas de caracteres que empiecen por 'a', que continúen con 0 o más 'b', y que sigan con una 'c'.

La 4 falla porque la 'a' y la 'c' no van seguidas ni existen caracteres 'b' entre ambas. La 5 falla por no tener una sola 'c'. y la 6 tiene los caracteres adecuados pero no en el orden correcto.

```
$ egrep 'ab*c' <<FIN
< 1 ac
< 2 aac
< 3 abbbc
< 4 axc
< 5 aaab
< 6 cba
< 7 aacaa
< FIN

1 ac
2 aac
3 abbbc
7 aacaa
```

Ejemplo para el operador '+' con el patrón 'ab+c'

Este patrón localizará las cadenas de caracteres que empiecen por 'a', que continúen con 1 o más 'b', y que sigan con una 'c'.

Solo la línea 3 cumple la expresión regular.

```
$ egrep 'ab*c' <<FIN
< 1 ac
< 2 aac
< 3 abbbc
< 4 axc
< 5 aaab
< 6 cba
< 7 aacaa
< FIN

3 abbbc
```

Ejemplo para el operador '.' con el patrón 'a..c'

Este patrón localizará las cadenas de caracteres que empiecen por 'a', que continúen con dos caracteres cualesquiera distintos de salto de línea y que sigan con una 'c'.

```
$ egrep 'a..c' <<FIN
< a00c
< axxcxx
< aacc
< abc
< ac
< axxxc
< FIN

a00c
axxcxx
aacc
```

Ejemplo para el operador '?' con el patrón 'ab?cd'

Este patrón localizará las cadenas de caracteres que empiecen por 'a', y que opcionalmente continúen con una 'b' y que continúe con los caracteres 'cd'.

```
$ egrep 'ab?cd' <<FIN
< abcd
< acd
< cd
< abbcd
< FIN

abcd
acd
```

Ejemplo para el operador '|' con el patrón 'ab|cd'

Este patrón localizará las cadenas de caracteres que contengan 'ab', 'cd', o '123'

```
$ egrep 'ab|cd|123' <<FI
< xxxabzzz
< xxxcdkkk
< badc
< dbca
< x123z
< FIN

xxxabzzz
xxxcdkkk
x123z
```

Ejemplo para el operador '^' con el patrón '^abc'

Este patrón localizará las lineas que empiecen por 'abc'.

```
$ egrep '^abc' <<FIN
< abcdefgh
< abc xx
< 00abc
< FIN

abcdefgh
abc xx
```

Ejemplo para el operador '$' con el patrón 'abc$'

Este patrón localizará las lineas que terminen por 'abc'.

```
$ egrep 'abc$' <<FIN
< abcd
< 000abc
< xxabcx
< abc
< FIN

000abc
abc
```

Ejemplo para el operador '[ ]' con el patrón '0[abc]0'

Este patrón localizará las cadenas que tengan un '0' seguido de un carácter que puede ser 'a', 'b', o 'c' y seguido de otro '0'.

```
$ egrep '0[abc]0' <<FIN
< 0abc0
< 0a0
< 0b0
< 0c0
< xax
< FIN

0a0
0b0
0c0
```

Ejemplo para el operador '^' (negación) dentro de '[ ]' con el patrón '0[^abc]0'

Este patrón localizará las cadenas que tengan un '0' seguido de un carácter que no podrá ser ni 'a', ni 'b', ni 'c' y seguido de otro '0'.

```
$ egrep '0[^abc]0' <<FIN
< 0abc0
< 0a0
< 000
< x0x0x
< 0c0
< FIN

000
x0x0x
```

Ejemplo para el operador '-' (rango) dentro de '[ ]' con el patrón '0[a-z]0'

Este patrón localizará las cadenas que tengan un '0' seguido de una letra minúscula, y seguido de otro '0'.

```
$ egrep '0[a-z]0' <<FIN
< 0a0
< 000
< 0Z0
< x0x0x
< FIN

0a0
x0x0x
```

Ejemplo para el operador '()' (agrupación) con el patrón '0(abc)?0'

Este patrón localizará las cadenas que tengan un '0' seguido opcionalmente de la secuencia abc, y seguido de otro '0'.

```
$ egrep  '0(abc)?0' <<FI
< hh00hh
< s0abc0s
< s0ab0s
< 0abc
< FIN

hh00hh
s0abc0s
```

Ejemplo para el operador '\' (escape) con el patrón '0\(abc\)?0'

Este patrón localizará las cadenas que tengan un '0' seguido opcionalmente de ')', y seguido de otro '0'.

```
$ egrep  '0\(abc\)?0' <<
< 0(abc)0xx
< 0(abc0xx
< hh00hh
< FIN

0(abc)0xx
0(abc0xx
```

Ampliando conocimientos sobre 'egrep'

Hemos aprendido cosas de egrep para ilustrar el uso de las expresiones regulares pero tanto grep como egrep permiten el uso de ciertas opciones que aún no hemos comentado y que no vamos a ilustrar con ejemplos porque no están relacionadas con las expresiones regulares. Solo vamos a señalar las opciones más útiles para que las conozca.

- -i Busca ignorando diferencias entre mayúsculas y minúsculas.
- -w Para forzar que la cadena reconocida sea una palabra completa.

- -l No muestra el contenido de la linea encontrada pero si que muestra el fichero que contiene la cadena buscada. Esto es útil cuando deseamos buscar entre muchos ficheros de los cuales algunos son binarios porque la muestra del contenido de un fichero binario puede desconfigurar el terminal.
- -n Muestra el número de línea dentro del fichero para ayudar a su localización.
- -v En lugar de sacar la lineas que cumplen la búsqueda sacará las que no cumplen.

Para un mayor detalle consultar las páginas del manual. Se trata de un comando de gran utilidad y le recomendamos que practique por su cuenta con el. Exísten varias modalidades de este comando.

- grep Comando básico.
- egrep Versión grep para uso de expresiones regulares extendidas.
- fgrep Versión grep interpreta los patrones no como expresiones regulares sino como cadenas de caracteres fijas.
- rgrep Permite buscar recursivamente en varios ficheros recorriendo un arbol de directorios.

Uso de expresiones regulares en 'sed'

Ahora veremos unos ejemplos con 'sed'. Este comando es capaz de editar un flujo o chorro de caracteres lo cual es de enorme utilidad dado que muchos comandos se comunican a través de entrada salida mediante chorros de caracteres. No podemos ver todas las posibilidades de 'sed' porque lo que nos interesa es su uso con expresiones regulares. Para ello usaremos un nuevo operador '&' que en la parte de substitución de 'sed' indica la parte que coincide con la expresión regular.

Para usar 'sed' se pone como primer argumento a continuación del comando la orden adecuada. Opcionalmente se puede poner un segundo parámetro que indicaría el nombre de un fichero. De no existir ese segundo parámetro esperará la entrada por la entrada estándar.

Se intentan reconocer las secuencias más largas posibles ^ y $ no consumen carácter '\n' si.

Empezamos con una sustitución sencillita.

```
$ echo "abc1234def" | sed "s/[0-9][0-9]*/NUMERO/"
abcNUMEROdef
```

Ahora usamos el operador '&'. Observe como se sustituye por el patrón reconocido.

```
$ echo "abc1234def" | sed "s/[0-9][0-9]*/<&>/"
abc<1234>def
```

Después eliminamos la secuencia numérica.

```
$ echo "abc1234def" | sed "s/[0-9][0-9]*//"
abcdef
```

Vamos a comprobar que en las expresiones regulares se intenta siempre reconocer la secuencia más larga posible.

```
$ echo "000x111x222x333" | sed "s/x.*x/<&>/"
000<x111x222x>333
```

Vamos ahora a trabajar sobre un fichero. Como siempre recordamos que trabaje con un usuario sin privilegios y dentro de /tmp. Ya sabe como hacerlo así que cambie ahora a /tmp antes de continuar. Para suprimir del fichero la palabra 'Hola' en las lineas de la 3 a la 4.

```
# Creamos un ficherito de prueba
$ cat <<FIN > prueba-sed.txt
< Hola este es un fichero con datos de prueba
< Hola otra vez.
< Hola otra vez.
< Hola otra vez.
< Hola otra vez y otra y otra y otra y otra y otra.
< Fin de los datos de prueba
< FIN
$ sed "3,4s/Hola//" prueba-sed.txt

Hola este es un fichero con datos de prueba
Hola otra vez.
 otra vez.
 otra vez.
Hola otra vez y otra y otra y otra y otra y otra.
Fin de los datos de prueba
```

El fichero no ha cambiado. El resultado sale por salida estándar. Ahora veamos que pasa si intentamos sustituir la palabra 'otra' por la palabra 'una' en todo el fichero.

```
$ sed "s/otra/una/" prueba-sed.txt

Hola este es un fichero con datos de prueba
Hola una vez.
Hola una vez.
Hola una vez.
Hola una vez y otra y otra y otra y otra y otra.
Fin de los datos de prueba
```

Vemos que solo se ha cambiado la primera ocurrencia de cada línea. Para obtener el resultado deseado tendríamos que usar la g al final.

```
$ sed "s/otra/una/g" prueba-sed.txt

Hola este es un fichero con datos de prueba
Hola una vez.
Hola una vez.
Hola una vez.
```

```
Hola una vez y una y una y una y una y una.
Fin de los datos de prueba
```

Ampliando conocimientos sobre 'sed'

Hemos aprendido cosas de sed para ilustrar el uso de las expresiones regulares pero sed tiene muchas más posibilidades que aún no hemos comentado. Comentaremos solo algo más pero sin extendernos demasiado. En sed se pueden especificar varias instrucciones separando con ';' cada una de ellas o usando la opción -e antes de cada instrucción. También podemos aprovechar el segundo introductor de la shell.

```
$ sed "s/otra/vez/g ; s/vez/pez/g" prueba-sed.txt

Hola este es un fichero con datos de prueba
Hola pez pez.
Hola pez pez.
Hola pez pez.
Hola pez pez y pez y pez y pez y pez y pez.
Fin de los datos de prueba

# Idéntico resultado podríamos haber conseguido usando
$ sed -e "s/otra/vez/g"  -e "s/vez/pez/g" prueba-sed.txt
# Una tercera forma para conseguir lo mismo
$ sed "
< s/otra/vez/g
< s/vez/pez/g
< " prueba-sed.txt
```

También podemos obtener el mismo resultado usando la opción -f y un fichero de instrucciones para 'sed'.

```
$ cat  <<FIN > prueba-sed.sed
< s/otra/vez/g
< s/vez/pez/g
FIN
$ sed -f prueba-sed.sed prueba-sed.txt
```

Existen posibilidades más avanzadas para usar 'sed' pero con lo que hemos mencionado se pueden hacer muchas cosas.

Podemos eliminar los blancos a principio y al final de linea así como sustituir más de un blanco seguido por un solo blanco.

```
$ cat  <<FIN > trim.sed
< s/^  *//g
< s/  *$//g
< s/  */ /g
< FIN
```

Si necesita hacer alguna transformación y no encuentra la forma fácil de hacer con sed piense que quizás necesite usar varias transformaciones en cadena.

```
#!/bin/bash
#########################################################
######
echo ; echo
# Filtraremos la salida del comando cal para que el
día de
# hoy resalte enmarcada entre corchetes.
#
# Primero obtenemos el día del més
DIAMES=`date '+%d'`
# Asi no vale. Hay que eliminar los posibles ceros a
la
# izquierda obtenidos # en la variable DIAMES.
# Usamos para ello printf(1)
DIAMES=`printf %d $DIAMES`
# Resaltar el día del més correspondiente al día de
hoy,
# Si es el último día del mes (patrón ' [23][0-9]$')
añadiremos
# un blanco al final y así usaremos siempre como
patrón de
# sustitución el día més ($DIAMES) entre dos
caracteres blancos
# sin que eso provoque errores en el último diá del
mes.
```

```
cal | head -n 1
cal | tail -n 7 | sed 's/  / +/g' | sed 's/^ /+/g' | \
sed 's/ /  /g' | sed 's/+/ /g' | sed 's/$/& /' | \
sed 's/^/ /' | sed "s/ $DIAMES /\[${DIAMES}\]/"
# Trazas
cal | tee traza.1 | \
tail -n 7 | tee traza.2 | \
sed 's/  / +/g' | tee traza.3 | \
sed 's/^ /+/g' | tee traza.4 | \
sed 's/ /  /g' | tee traza.5 | \
sed 's/+/ /g' | tee traza.6 | \
sed 's/$/& /' | tee traza.7 | \
sed 's/^/ /' | tee traza.8 | \
sed "s/ $DIAMES /\[${DIAMES}\]/" > traza.9
```

Las últimas líneas las hemos añadido simplemente para analizar paso a paso como se va transformando la salida original del comando cal del cual en primer lugar ya hemos separado la primera línea. Las distintas etapas quedan registradas en traza.1 a traza.8.

```
traza.1 contiene la salida del comando cal.

     April 2002
 S  M Tu  W Th  F  S
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30

En traza.2 se ha eliminado la primera línea.

 S  M Tu  W Th  F  S
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30

En traza.3 se ha sustituido toda secuencia de
dos blancos por ' +'

 S +M Tu +W Th +F +S
```

```
 + +1 +2 +3 +4 +5 +6
 7 +8 +9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30
```

En traza.4 sustituimos los blancos a comienzo de línea por '+'

```
+S +M Tu +W Th +F +S
++ +1 +2 +3 +4 +5 +6
+7 +8 +9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30
```

Gracias a los pasos anteriores ya tenemos una estructura de líneas formada por dos caracteres distintos de blanco separadas por un blanco y precisamente este blanco en traza.5 lo sustituimos por dos blanco para dar ensanchar toda la salida.

```
+S  +M  Tu  +W  Th  +F  +S
++  +1  +2  +3  +4  +5  +6
+7  +8  +9  10  11  12  13
14  15  16  17  18  19  20
21  22  23  24  25  26  27
28  29  30
```

Ahora ya no son útiles los '+' y los sustituimos por blancos. En traza.6 ya tenemos la salida ensanchada.

```
 S   M  Tu   W  Th   F   S
     1   2   3   4   5   6
 7   8   9  10  11  12  13
14  15  16  17  18  19  20
21  22  23  24  25  26  27
28  29  30
```

En traza.7 no se aprecia la diferencia pero si que existe. cal termina las líneas inmmediatamente despues del número, y no hay razón para que lo haga de otra manera pero a nosotros nos conviene que absolutamente todos los números tengan un blanco a la izquierda

```
y otro a la derecha para que a la hora de
enmarcar el número solo tengamos que sustituir
blancos y no se desalineen las columnas. Por
ello aunque no se aprecie en traza.7 hemos
añadido un blanco al final de las líneas.

 S   M  Tu   W  Th   F   S
     1   2   3   4   5   6
 7   8   9  10  11  12  13
14  15  16  17  18  19  20
21  22  23  24  25  26  27
28  29  30

Ahora en traza.8 se ve como hemos añadido un
blanco a la izquierda.

  S   M  Tu   W  Th   F   S
      1   2   3   4   5   6
  7   8   9  10  11  12  13
 14  15  16  17  18  19  20
 21  22  23  24  25  26  27
 28  29  30

Por último bastará sustituir el día del més
entre blancos por el día de més entre corchetes.

  S   M  Tu   W  Th   F   S
      1   2   3   4   5   6
  7   8   9  10  11  12  13
 14  15  16  17  18  19 [20]
 21  22  23  24  25  26  27
 28  29  30
```

Independientemente de que a usted se le ocurra otra solución mejor, se trataba de practicar con sed y con siete seds encadenados pensamos que ya habrá saciado su sed de sed(1).

De todas formas si lo quiere más bonito y cortito pruebe esto otro.

```
ESC=`echo -e "\033"`
cal | sed "s/$/& /" | sed "s/ $DIAMES / $ESC[1m$
{DIAMES}$ESC[0m /"
```

Ya nos salimos del tema pero si esto ha despertado su curiosidad mire console_codes(4).

EL EDITOR VI (Primera parte)

Introducción

En primer lugar vamos a dedicar bastante esfuerzo a explicar porque aprender 'vi'. Quizás pensó que este capítulo no era para usted porque ya sabe usar otro editor. También vamos explicar la necesidad de dedicar dos largos capítulos a este editor.

Cuando finalice este capítulo podrá editar sus ficheros con 'vi' y hacer bastante trabajo útil pero para ello quizas necesite algo más de esfuerzo que en capítulos anteriores y no porque sea dificil sino porque tendrá que practicar bastante por su propia cuenta.

Otra cosa será sacarle el máximo partido a 'vi'. En un segundo capítulo dedicado a 'vi' veremos opciones más avanzadas. Son opciones que quizás no necesite usar tan frecuentemente pero cuando tenga oportunidad de usarlas comprenderá la potencia el editor 'vi'.

Muchos editores actuales permiten usar distintos tipos de letras y generan documentos con un aspecto precioso que además coincide lo que se ve en pantalla cuando se edita, y con lo que saldrá por la impresora. Está claro que 'vi' no sirve para esto. 'vi' está diseñado para editar ficheros en formato de texto estándar. Pero aunque solo sirva para este formato de texto lo hace muy bien y siempre tendremos oportunidad de usarlo porque a un compilador, o a un intérprete le da igual la estética.

Son muchos los editores disponibles en Linux pero desde un principio dijimos que este curso podría servir en gran medida para aprender otros SO tipo Unix. El editor 'vi' está presente desde hace mucho tiempo en un montón de sistemas tipo Unix. Aquí explicaremos solo unos pocos aspectos de 'vi' con la idea de que pueda realizar con razonable soltura la

edición de un texto en 'vi'. Con muy pocos comandos de 'vi' se pueden hacer muchas cosas y 'vi' es un editor que para ciertas tareas resulta de una potencia increíble. El que escribe estas lineas lo está haciendo con 'vi' y lleva muchos años usando este editor prácticamente para casi todo y le saco un gran partido, pero cada vez que me encuentro con alguién que sabe bastante de 'vi' aprendo algo nuevo. 'vi' es un editor muy práctico sin tener que conocerlo a fondo pero en manos de un experto resulta de una eficacia espectacular.

'vi' es un editor de lineas interactivo y ha sido muy criticado. Una de las razones para ello es que tiene varios estados o formas de funcionamiento y si uno pasa de un estado a otro sin darse cuenta y continua tecleando cosas, puede ocurrir un pequeño destrozo en el fichero que está editando. Otro inconveniente de 'vi' es que requiere un poquito de aprendizaje. Hay muchísima gente que está acostumbrada a usar otros editores más intuitivos y porque no decirlo más cómodos, y rechazan a 'vi' como algo prehistórico. Otra cosa que suele criticarse es que se usan letras del teclado para cosas tan normales como el movimiento del cursor y la paginación del texto pero este es un claro ejemplo del condicionamiento de la cultura de Microsoft y de los PCs. 'vi' fue diseñado para que se pudiera usar en una gran variedad de sistemas y terminales. El uso de las teclas de flechas puede lograrse mediante la adecuada configuración de 'vi'. Uno de los puntos fuertes de 'vi' es su adaptabilidad.

Quizás una de las más poderosas razones para aprender 'vi' es que 'vi' forma parte de la cultura de los SO tipo Unix. Muchas personas acostumbradas a trabajar por ejemplo con Windows asumirán como estándar cosas que no lo son y cuando se encuentran algo diferente intentan aprovechar al máximo sus conocimientos anteriores.

Generalmente existe una exageradísima tendencia a continuar usando el primer editor que se aprendió y luego se cambia solo de editor obligado por circunstancias poderosas. Nosotros hemos decidido que en posteriores lecciones asumiremos que usted ya sabe usar 'vi'. Pronto tendrá que hacer sus primeros programas en shell-script. Entonces necesitará un editor y nada mejor que 'vi'.

En este curso se le da mucha importancia a la cultura Unix. Para una persona que aprende 'vi' surgen varias oportunidades para aplicar lo que sabe en comandos distintos de 'vi'. La forma de trabajar de 'vi' ha tenido mucho impacto en otros programas. Puede encontrar juegos, hojas de cálculo utilidades, etc que usan cosas parecidas a 'vi'. Ya vimos 'sed' que usa comandos similares a algunos comandos de 'vi' y ahora vamos a ver a continuación algunas nociones de otro editor ('ed') que es otro buen ejemplo de lo que estamos comentando.

'ed' versus 'vi'

Dado que 'vi' está pensado para su uso interactivo desde un terminal no resulta muy apropiado de usar en combinación con otros programas mediante redirección de entrada salida. En el capítulo anterior de expresiones regulares vimos como 'sed' se podía usar como filtro de entrada salida. Existe un editor llamado 'ed' que es capaz de editar texto en modo no interactivo. Para ello se suele recurrir a un script que proporcionará a 'ed' las instrucciones adecuadas. Somos de la opinión de que las lecciones que no dejan ver rápidamente cual es su utilidad y que no se practican están condenadas al olvido. Por eso queremos evitar que piense en términos de *"para que necesito yo conocer esto de ed"*. Por eso más tarde comentaremos algo de 'diff' combinado con 'ed'. Así comprenderá mejor la utilidad del editor 'ed'.

Ahora vamos a generar un fichero 'ejemplo.txt' con unas pocas lineas para hacer unas pocas prácticas.

```
$ cd /tmp
$ cat <<FIN > ejemplo.txt
> No olvide visitar nuestra web
> http://www.ciberdroide.com
> y esto es todo.
> FIN
```

Vamos a sacar una copia para comprobar luego los cambios.

```
$ cp ejemplo.txt ejemplo.txt.original
```

Ahora generamos un fichero con las instrucciones para editar el fichero anterior. Deseamos que primero localice la linea que contiene 'www.ciberdroide.com' y a continuación añada una linea (comando 'a') '(Tienda Virtual de Linux)'. Salimos al modo comandos (comando '.') Salvamos (comando 'w') y salimos (comando 'q').

```
$ cat <<FIN > ejemplo.ed
> /www.ciberdroide.com/
> a
> (Tienda Virtual de Linux)
> .
> w
> q
> FIN
$ ed ejemplo.txt < ejemplo.ed

73
http://www.ciberdroide.com
99
```

Veamos como ha quedado.

```
$ cat ejemplo.txt

No olvide visitar nuestra web
http://www.ciberdroide.com
(Tienda Virtual de Linux)
y esto es todo.
```

Podemos ver las diferencias con el original usando el comando 'diff'. El comando diff es un comando que sirve para señalar diferencias entre dos ficheros pero con la opción -e estas diferencias se expresan en una forma adecuada para ser interpretadas por 'ed'. Veamos como funciona.

```
$ diff -e ejemplo.txt.original ejemplo.txt
```

```
2a
(Tienda Virtual de Linux)
.
```

Vamos a partir del fichero original y le vamos a pasar por la entrada estándar únicamente las diferencias obtenidas con 'diff -e'

```
$ cp ejemplo.txt.original ejemplo2.txt
$ diff -e ejemplo.txt.original ejemplo.txt > ejemplo.diff
$ cat ejemplo.diff

2a
(Tienda Virtual de Linux)
.
```

Vemos que falta las instrucciones 'w' y 'q' para salvar y salir respectivamente así que las añadimos al fichero 'ejemplo.diff'.

```
$ cat <<FIN >> ejemplo.diff
> w
> q
> FIN
$ cat ejemplo.diff

2a
(Tienda Virtual de Linux)
.
w
q
```

Ya tenemos todas las instrucciones necesarias. Vamos a aplicar los cambios con 'ed'.

```
$ ed ejemplo2.txt < ejemplo.diff

73
y esto es todo.
```

```
99

$ cat ejemplo2.txt

No olvide visitar nuestra web
http://www.ciberdroide.com
(Tienda Virtual de Linux)
y esto es todo.
```

En lugar de guardar varias versiones de un gran fichero que sufre pequeñas diferencias de vez en cuando se podría guardar una sola versión y unos pequeños ficheros con esas diferencias ocurridas entre una versión y la siguiente. De esta forma podrían aplicarse secuencialmente para ir recuperando sucesivamente las versiones anteriores en caso de necesidad. Cuando se desarrolla una aplicación se van introduciendo sucesivamente cambios en los programas y para poder recuperar versiones antigüas habría que ir salvando y conservando cada nueva versión. Los programas que se utilizan para el mantenimiento de distintas versiones de una aplicación suelen basarse en esta idea para ahorrar espacio de almacenamiento.

'vim' versus 'vi'

En Linux existe un editor 'vi' muy mejorado que se llama 'vim'. Este editor es muy similar a 'vi' pero con importantes mejoras. La más importante sin lugar a dudas es la capacidad de deshacer retrocediendo uno a uno en los últimos cambios realizados. Esto permite recuperar aquello de los destrozos que dijimos antes. También permite resaltar la sintaxis de ciertos ficheros como. shell-script, html, lenguaje C, y otros muchos lenguajes. Este y otros detalles lo hacen muy adecuado para programar. El manejo del cursor en los distintos modos del editor resulta mucho más cómodo y un montón de cosas más.

Mas adelante veremos que se pueden usar distintas opciones para que el editor funcione en la forma que nosotros queramos. 'vim' puede ejecutarse en modo compatible 'vi' (:set compatible) pero es mejor usar el modo no

compatible que nos proporcionará las agradables ventajas sobre el 'vi' tradicional que antes comentamos. (:set nocompatible) Generalmente en Linux se instala 'vim' como sustituto de 'vi' de forma que aunque usted teclee 'vi' en realidad puede estar ejecutando 'vim' o 'elvis' u otros similares. Salgamos de dudas:

```
$ vi -? | less
```

Solo si le invita a usar la opción '-h' hágalo.

```
$ vi -h | less
```

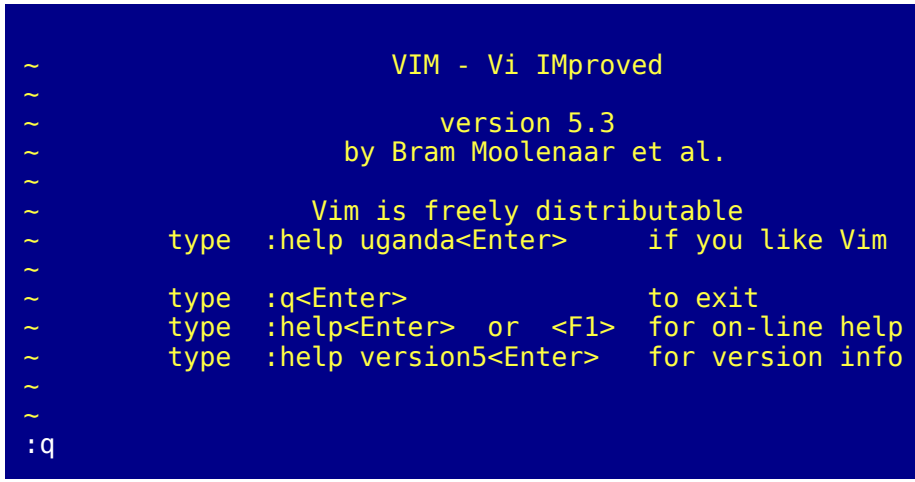
Compruebe en las primeras lineas si se trata de 'vim'. Si ya ha entrado por error en editor salga mediante <ESC> :q<ENTER> Continuemos averiguando cosas.

```
$ whereis vi
$ whereis vim
$ whereis elvis
$ man vi
$ man vim
$ man elvis
$ man ex
$ man view
$ man ed
```

Ahora si vamos a entrar y a salir en el editor.

```
$ vi
```

Ya está dentro del editor. Compruebe si sale información relativa al tipo de editor y salga tecleando ':q' Para significar que estamos en el editor 'vi' pondremos el fondo azul oscuro y las letras en amarillo son las que salen en pantalla y en blanco las que introducimos nosotros.



Cuadro resumen para el uso de 'vi'

A continuación se describen una serie de comandos de uso frecuente en 'vi'. Se destacan los que se consideran más importantes con un mayor numero de asteriscos. En una primera lectura le recomendamos que empiece a utilizar y a aprender aquellos comandos marcados con '***' que son los más importantes. Le aconsejamos que no pase de lección hasta que no se aprenda como mínimo los comandos calificados con '***'. Al final del capítulo como es costumbre pondremos un test pero solo haremos preguntas sobre comandos calificados con '***' que serán los que practicaremos de forma especial en este capítulo.

Hemos agrupado toda esta información en un único cuadro para que sea su chuleta, para usar y aprender 'vi'. Solo se incluyen los comandos que suponemos serán de la mayor utilidad para un usuario novato y dado que hay varias formas de hacer una misma cosa hemos elegido por usted y hemos clasificado de forma totalmente subjetiva una serie de comandos para que inicialmente tenga que aprender solo un reducido número de ellos. También se incluyen otros comandos para que pueda ser usado como un cuadro de consulta bastante completo.

```
***     --> Comando de uso imprescindible.
**      --> Comando de uso muy frecuente.
*       --> Comando de uso no tan frecuente.
```

LINEA DE COMANDOS		
** *	vi	Editar un nuevo fichero
** *	vi <fichero>	Editar un fichero. (Si el fichero es de solo lectura lo editara en modo solo lectura.
*	vi <fichero1> <fichero2> <fichero3> ...	Editar varios ficheros sucesivamente.
**	vi -R <fichero>	Editar en modo solo lectura.
**	vi -r <fichero>	Editar recuperando modificaciones no salvadas.
MODO COMANDOS Se pasa desde el modo entrada (inserción o reemplazo) mediante <ESC>.		
Movimientos del cursor		
**	h	Carácter anterior
**	<ENTER>	Linea siguiente
**	j	Linea siguiente
**	k	Linea anterior
**	<ESPACIO>	Carácter siguiente
**	l	Carácter siguiente
*	L	Ultima linea de la pantalla
*	H	Primera linea de la pantalla
*	M	Linea central de la pantalla
*	<CTRL+B>	Retroceso de página

*	<CTRL+F>	Avance página
*	+	Primer carácter de la linea siguiente
*	-	Primer carácter de la linea anterior
*	$	Desplazarse al final de linea.
*	0	Desplazarse al principio de linea.
Modo entrada de texto		
***	i	Pasar a modo entrada insertando.
***	R	Pasar a modo entrada reemplazando.
**	a	Avanzar el cursor y pasar a modo entrada insertando.
***	o	Insertar una linea después de la actual y pasar a insertar.
*	O	Insertar una linea antes de la actual y pasar a insertar.
Corregir		
***	u	Deshacer última modificación. (En vim se pueden ir deshaciendo los los últimos cambios en otros editores vi solo se deshace el último cambio).
***	<CTRL+R>	En 'vim' sirve para deshacer el último comando 'u'
**	U	Recuperar linea completa.
Buscar		
**	/	Busca una cadena de caracteres, hacia delante.
*	?	Busca una cadena de caracteres, hacia atrás.

*	n	Siguiente en la búsqueda.
*	N	Anterior en la búsqueda.
Copiar, Mover, y borrar lineas		
***	dd	Borrar una linea.
***	<n>dd	borrar <n> lineas.
***	yy	Meter una linea en el buffer.
***	<n>yy	Meter <n> lineas en el buffer.
***	p	Copiar la linea del buffer después de la línea actual.
*	P	Copiar la linea del buffer antes de la línea actual.
Varios		
***	<CTRL+L>	Refrescar pantalla. (Útil cuando se descompone la pantalla)
**	<CTRL+G>	Visualiza el nombre del fichero, numero de lineas totales y en que linea estamos.
***	J	Une la línea actual con la siguiente.
**	.	Repetir último comando.
***	:	Pasar al modo ex (Modo comandos)
***	ZZ	Terminar salvando si hubo cambios.

MODO EX

Se pasa desde el modo de comandos al modo ex con ':'

Se puede forzar a entrar directamente en este modo usando la opción '-e' o ejecutando el comando 'ex'.		
**	:<#linea>	Ir a una linea.
*	:e <fichero>	Pasa a editar de forma simultanea otro fichero.
*	:e #	Volver al fichero anterior.
** *	:w	Salvar. (No puede sobreescribir si se entró con vi -R)
** *	:w!	Salvar forzando (Cuando falla :w).
**	:x	Salir salvando si es necesario.
** *	:w <fichero>	Salvar con otro nombre. (No puede existir el fichero)
** *	:w! <fichero	Salvar con otro nombre forzando. (Permite sobreescribir)
** *	:r <fichero>	Lee un fichero y lo carga en el lugar indicado por la posición actual del cursor.
**	:r !<comando	Ejecuta en una subshell el comando y guarda el resultado del mismo en la posición indicada por el cursor.
** *	:q	Salir si no hay modificación.
** *	:q!	Salir sin salvar cambios.
**	:wq	Salir salvando cambios.
*	:sh	Ejecutar una subshell (se retorna con exit).
*	:g/<s1>/p	Visualizar lineas que tienen <s1>
*	:g/<s1>/s//<s2>/g	Sustituir globalmente <s1> por <s2>.

*	:g/<s1>/s//<s2>/gc	Ídem pero pidiendo confirmación.
*	:n	Siguiente documento.
*	:args	Mirar ficheros a editar.
*	:ab m n	Abreviaturas. Teclear 1 se sustituye por 2
*	:unab m	Desabreviar.
*	:map m n	Crear macro (m hace n) (Ver ejemplos con macros)
*	:map! m n	Crear macro en modo inserción (m hace n)
*	:unmap m	Destruir macro m
*	:unmap! m	Destruir macro de inserción m.
*	:cd	Cambiar de directorio actual.
**	:set nu	Activar visualización de números de línea
**	:set nonu	Activar visualización de números de línea
**	:set all	Ver todas las opciones con sus valores.
**	:set nocompatible	Opción de 'vim' para no obligar a ser compatible 'vi'
** *	:set showmode	Para visualizar siempre el modo en que nos encontramos.
** *	:!<comando>	Ejecutar un comando en una subshell. (Ver ejemplos con !)

Muchos de estos comandos pueden aplicarse dentro de un rango de líneas. Un número de línea puede venir como un número, un punto, un $.

Un punto	.	Línea actual
Un punto	.-N	N Líneas antes de la actual
Un punto	.+N	N Líneas después de la actual
Un número	N	Línea N
Un $	$	Ultima línea.
Un $	%	Desde la última a la primera línea (Abreviatura de 1,$).

***	:#linea1, #linea2 w <fichero>	Salvar desde 1 hasta la 2 en <fichero>
**	:#linea1, #linea2 co #linea3	Copiar desde 1 hasta 2 en 3.
**	:#linea1, #linea2 mo #linea3	Mover desde 1 hasta 2 en 3.
**	:#linea1, #linea2 de	Borrar desde 1 hasta 2.
**	:#linea1, #linea2 s/<s1>/<s2>	Sustituir la primera ocurrencia en cada linea desde 1 hasta la 2 la cadena <s1> por la cadena <s2>
**	:#linea1, #linea2 s/<s1>/<s2>/c	Sustituir la primera ocurrencia en cada linea desde 1 hasta la 2 la cadena <s1> por la cadena <s2> con confirmación.
**	:#linea1, #linea2 s/<s1>/<s2>/g	Sustituir todas las ocurrencias en cada linea desde 1 hasta la 2 la cadena <s1> por la cadena <s2>
**	:#linea1, #linea2 s/<s1>/<s2>/gc	Sustituir todas las ocurrencias en cada linea desde 1 hasta la 2 la cadena <s1> por la cadena <s2> con confirmación

Sería conveniente que sacara por la impresora esta información para usarla a modo de chuleta en sus primeras prácticas con 'vi'.

Existe otra forma de aplicar comandos dentro de un rango de lineas. Desde el modo de comandos pulsando 'v' se pasa al modo 'vi' o modo 'visual'. Entonces moviendo el cursor marcamos todo el rango de lineas que queramos tratar y pulsando ':' pasamos al modo 'ex' para aplicar el comando que queramos. Cualquier comando que se pueda introducir en 'vi' estando en el modo de 'ex' puede ser incluido en un fichero .exrc en el $HOME que servirá para que 'vi' ejecute esos comandos cada vez que arranque. Se usa mucho para definir macros y para activar determinadas opciones con set pero eso lo veremos en otro capítulo.

Practicando lo más esencial de 'vi'

Si no se usa ninguna opción inicial vi empezará en modo comandos. Para pasar al modo entrada hay varias formas pero recomendamos que se aprenda de momento solo. i, a, o, R. Ya hemos descrito estas funciones en 'chuleta-vi.txt' y practicaremos con ellas más adelante.

Del modo entrada (Insertar o reemplazar) se sale con <ESC>.

Normalmente trabajábamos en '/tmp' como directorio temporal pero esta vez vamos a crear un directorio nuestro de trabajo para poder conservar nuestro trabajo el tiempo que queramos. Para ello crearemos un directorio 'cursolinux' y nos situaremos en él para trabajar.

```
$ cd
$ mkdir cursolinux
$ cd cursolinux
$ vi ej-vi1.txt
```

Dado que no existía ningún fichero 'ej-vi1.txt' estaremos editando un fichero nuevo y dependiendo del editor usado (nosotros usaremos 'vim') aparecerá algo del tipo.

```
~
~
~
"ej-vi1.txt" [New File]
```

Está en modo comandos y va a teclear sus dos primeros comandos. Cuando vea '<ENTER>' en los ejemplos significa que debe pulsar esa tecla.

```
:set showmode<ENTER>
:set nu<ENTER>
```

Acabamos de activar la visualización del modo de funcionamiento y la visualización de los números de línea. Ninguna de estas cosas son necesarias para editar un fichero pero a nosotros nos van a ayudar a ver más claro lo que hacemos.

Ahora pulsaremos la 'i' y aparecerá en la linea inferior el indicador de que estamos en modo entrada para inserción. Desde este modo introduciremos las siguientes lineas de texto.

```
     1  Estamos practicando con el editor vi.<ENTER>
     2  No parece tan difícil de usar.<ENTER>
     3  Ya voy por la tercera linea y no ha ocurrido
nada grave.<ENTER>
     4  <ENTER>
     5  <ENTER>
     6  Como he saltado un par de líneas voy por la
linea seis.<ENTER>
     7  De momento no escribo nada más.<ESC>
    ZZ
```

Con la tecla <ESC> salimos al modo comandos y con 'ZZ' (mayúsculas) desde el modo comandos salimos salvando el fichero. Ahora vamos a volver a editarlo.

```
$ vi ej-vi1.txt
```

Volvemos a activar los modos 'showmode' y 'nu' y nos situamos en la línea 3 y la borramos con 'dd' luego nos situamos en la primera línea y hacemos 'p' que copiará la linea eliminada a continuación de la línea primera. 'dd' guarda siempre en un buffer la información eliminada para poder recuperarla con 'p'. Podríamos haber borrado cinco lineas de golpe usando '5dd' y las cinco líneas habrían sido guardadas en el buffer para su posible recuperación con 'p'.

```
:set showmode<ENTER>
:set nu<ENTER>
:3<ENTER>
dd
:1<ENTER>
p
```

Pero ahora la línea 2 dice que es la tercera así que intente usted mismo borrar una de las líneas vacías y situarla como segunda línea usando de nuevo 'dd' y 'p'. Esta vez mueva de linea usando 'j' desde el modo comandos para bajar a la linea siguiente y 'k' para subir a la línea anterior. Si el teclado está configurado en su editor para manejar correctamente las teclas de flechas podrá hacer esto mismo usando las teclas flecha abajo y flecha arriba en sustitución de la 'j' y la 'k' respectivamente. Si algo sale mal borre el fichero y vuelva a empezar procurando no volver a meter la pata. Bueno el caso es que el fichero debería quedar con este aspecto.

```
    1 Estamos practicando con el editor vi.
    2
    3 Ya voy por la tercera linea y no ha ocurrido
nada grave.
    4 No parece tan difícil de usar.
    5
    6 Como he saltado un par de líneas voy por la
linea seis.
    7 De momento no escribo nada más.
```

Vamos a sacar una copia de este fichero modificado con otro nombre, y luego vamos a duplicar varias veces el contenido de las líneas cuatro y cinco usando '2yy' (guarda en el buffer dos lineas) y 'p' para (copia el contenido del buffer en la línea actual). Asegurese que está en modo comandos. Pulse <ESC> si no está seguro y teclee los comandos siguientes. Advierta que algunos comandos empiezan por ':' y terminan por <ENTER> mientras que otros no.

```
:w ej-vi2.txt
:4<ENTER>
2yy
:3<ENTER>
p
:3<ENTER>
p
:3<ENTER>
p
```

Estamos repitiendo varias veces la línea 'No parece tan difícil de usar.' Debería quedar finalmente con este aspecto.

```
    1 Estamos practicando con el editor vi.
    2
    3 Ya voy por la tercera linea y no ha ocurrido
nada grave.
    4 No parece tan difícil de usar.
    5
    6 No parece tan difícil de usar.
    7
    8 No parece tan difícil de usar.
    9
   10 No parece tan difícil de usar.
   11
   12 Como he saltado un par de líneas voy por la
linea seis.
   13 De momento no escribo nada más.
```

Como hemos visto con '2yy' hemos metido en el buffer dos lineas en lugar de una. Para meter una sola linea habría bastado usar 'yy' o '1yy'. Ahora

vamos a salir pero sin guardar ninguno de los cambios realizados. Primero intentaríamos salir sin más con ':q' desde el modo comandos.

```
:q
```

Obtendremos un aviso del editor para no perder las últimas modificaciones.

```
No write since last change (use ! to override)
```

Pero realmente deseamos salir sin salvar así que intentamos de nuevo con ':q!' desde el modo comandos.

```
:q!
```

Vamos a comprobar que los últimos cambios no se han guardado, y vamos a ver que permisos tiene este fichero.

```
$ cat ej-vi1.txt

Estamos practicando con el editor vi.
No parece tan difícil de usar.
Ya voy por la tercera linea y no ha ocurrido nada grave.


Como he saltado un par de líneas voy por la linea seis.
De momento no escribo nada más.
$ ls -l ej-vi1.txt

-rw-r--r--   1  xxxx xxxx   216 ago 12 18:24 ej-vi1.txt
```

La información de permisos puede ser diferente. Recuerde que dijimos que los permisos de un fichero cuando se crea dependen de la 'umask' pero ahora solo nos interesa comprobar efectivamente tenemos permisos

para leer y escribir en este fichero. Vamos a eliminar los permisos de escritura del fichero y vamos a intentar editarlo nuevamente.

```
$ chmod 444 ej-vi1.txt
$ ls -l ej-vi1.txt
-r--r--r--   1  xxxx xxxx   216 ago 12 18:24 ej-vi1.txt
$ vi ej-vi1.txt
```

Probablemente aparecerá en la linea inferior de su editor una advertencia de que está en modo solo lectura.

```
"ej-vi1.txt" [readonly] 7 lines, 216 characters
```

Bueno a pesar de ello vamos a intentar hacer cambios y como estamos practicando continuaremos con una serie de comandos que aún no hemos visto y que teníamos clasificados con '***'.

```
:3
:.,$ w trozofinal
:1,. w trozoinicial
:1,$ de
:r trozoinicial
:$
:r !date
:$
:r trozofinal
:w tempfich
:w

'readonly' option is set (use ! to override)
```

Permitió escribir otros ficheros pero se niega a sobrescribir el fichero que estamos editando con los cambios porque está en modo solo lectura y sugiere que usemos '!' para forzar su escritura.

```
:w!
```

En este caso estamos forzando la situación y estamos escribiendo en un fichero en el que no tenemos permisos pero puesto que somos propietarios del mismo el editor puede manejar los permisos para dar permiso de escritura, sobrescribir el fichero y volver a cambiar los permisos dejando los permisos originales.

Podemos ejecutar comandos desde el editor así que porque no miramos los permisos con 'ls -l'.

```
:!ls -l

total 6
-r--r--r--   1 xxxx xxxx       304 ago 12 19:11 ej-
vi1.txt
-r--r--r--   1 xxxx xxxx       304 ago 12 19:11 ej-
vi1.txt~
-rw-r--r--   1 xxxx xxxx       216 ago 12 18:28 ej-
vi2.txt
-rw-r--r--   1 xxxx xxxx       304 ago 12 19:09
tempfich
-rw-r--r--   1 xxxx xxxx       147 ago 12 19:06
trozofinal
-rw-r--r--   1 xxxx xxxx       126 ago 12 19:06
trozoinicial

Press RETURN or enter command to continue
```

Debemos pulsar <ENTER> para volver al editor Puesto que podemos ejecutar cualquier comando también podemos ejecutar bash como una subshell. Aprovecharemos para ver los procesos activos.

```
:!bash
bash-2.01$ ps

  PID TTY STAT TIME COMMAND
  221   2 S    0:00 -bash
  893   2 S    0:00 vi ej-vi1.txt
  944   2 S    0:00 bash
  951   2 R    0:00 ps
bash-2.01$ exit
```

```
exit

1 returned

Press RETURN or enter command to continue
```

Por fin estamos de vuelta en el editor. Vamos a continuar practicando algunos comandos importantes que nos quedan.

```
:3<ENTER>
o
Hemos pasado a insertar abriendo una linea.<ESC>
J
```

Con la 'J' mayúscula hemos unido dos líneas en una. Ahora vamos a pasar al modo entrada para reemplazar.

```
:3<ENTER>
R
XXXXXXXXXXXXXXXXX<ESC>sertar abriendo una linea.  Ya
voy por la tercera linea y no ha ocurrido nada grave.

<ESC>
<CTRL+G>


"ej-vi1.txt" [Modified][readonly] line 4 of 10 --40%--
col 44
```

Desde el modo comandos podemos consultar el nombre del fichero, cuantas lineas tiene en total, en que línea y columna estamos, y otras cosas con solo pulsar <CTRL+G> .

Ahora utilice varias veces el comando 'u'. Si está en 'vim' verá como se deshacen retrocediendo secuencialmente en las últimas modificaciones que hemos realizado. Luego si está usando 'vim' use <CTRL+R> para

avanzar de nuevo en las últimas modificaciones hasta recuperarlas todas. Para terminar salga con ':q!'

Practique con la combinación de 'yy' o 'nyy' y 'p'. O con la combinación 'dd' o 'ndd' y 'p'. Se usan mucho para copiar o trasladar pequeños bloques de información.

Para mover una linea hacer 'dd', luego situarse en la posicion destino y hacer 'p'. Para mover varias lineas hacemos algo parecido. Borramos primero varias lineas y luego desde la posición destino hacemos 'p'. Para copiar en lugar de mover usaríamos 'yy' en lugar de 'dd'.

Bueno utilice su propia imaginación e investigue y practique por su cuenta. Esta lección no le obliga a digerir conceptos nuevos pero si requiere iniciativa para practicar con el editor.

Existe un comando que lo hemos marcado como esencial cuando en realidad se usa poquísimo pero es un comando que conviene conocer porque en el momento más inesperado puede ser necesario usarlo. Con este comando el editor redibuja totalmente el contenido de la pantalla. Esto puede ser necesario porque a su terminal podría llegarle información desde otro proceso. Por ejemplo un mensaje de error del sistema o un aviso de su administrador para que finalice antes de 10 minutos. Quizás su administrador es de los que repite el mensaje más de una vez y a cada mensaje recibido su pantalla queda totalmente destrozada. No pasa nada. Pulse <CTRL+L> y todo volverá a la normalidad. Esto puede ocurrirle cuando este usando otro programa distinto de 'vi' y quizás pueda usar gualmente <CTRL+L>.

Recuperar la información perdida por muerte prematura del editor

Otra cosa que debe saber es que cuando 'vi' recibe una señal para finalizar ordenadamente guardará el contenido de lo que está editando en un fichero de seguridad. Para recuperar esta información necesitará usar 'vi -r fichero'. Como ejercicio libre le proponemos que empiece a editar un fichero que llamaremos 'prueba-kill.txt'. Escriba unas cuantas cosas y sin

salvar nada, pase a otro terminal y envíe un kill con el valor por defecto (15 = SIGTERM). Para eso primero tendrá que averiguar el pid del proceso vi. Por ejemplo use 'ps | grep vi'. Supongamos que averigua que su pid es 445. Bastaría hacer 'kill 445'. Esto finalizará la edicion del fichero 'prueba-kill.txt'. No edite este fichero con 'vi prueba-kill.txt' porque perderá todos los cambios. En cambio si edita con 'vi -r prueba-kill.txt' si los recuperará. En este punto salve y continue editando cosas y antes de salvar los nuevos cambios vuelva a averiguar el pid desde otro terminal. Supongamos que el nuevo pid fuera 448. Envíe esta vez la señal SIGKILL no enmascarable mediante 'kill -9 448'. Comprobará que esta vez el editor no será capaz de recuperar los cambios. Por ejemplo si ocurre un apagado normal (shutdown) del sistema mientras está usted editando o mientras usted está ausente del terminal con la sesión de edición abierta, tampoco ocurrirá nada. La razón de esto, es que el sistema primero envía a todos los procesos activos una señal de SIGTERM para que finalicen ordenadamente. Después de esto y pasados unos segundos se supone que la mayoría de los procesos habrán terminado. (A los restantes procesos les enviará un SIGKILL para matarlos incondicionalmente). Cuando le ocurra esto tendrá que volver a entrar en sesión con login y seguramente recibirá una notificación en su correo advirtiendo de la existencia de un fichero que debe ser editado con la opción '-r' para recuperar información volcada durante la finalización prematura de 'vi'.

EL EDITOR VI (Segunda Parte)

Objetivos de esta lección

Esta lección menciona algunas posibilidades avanzadas del editor 'vi'. Se supone que el alumno ha estudiado las lecciones precedentes y en particular el capítulo anterior dedicado al editor 'vi'. Eso significa que ya sabe editar. Vamos a suponer que incluso ha practicado un poco por su cuenta. Por ello lo que vamos a explicar en este capítulo será de escasa utilidad si no ha asimilado la lección anterior.

Poco a poco vamos a ir asumiendo un mayor dominio del alumno. Ya no vamos a detallar tan a menudo paso por paso cosas que ya se han explicado y practicado más de una vez. Eso le obligará a poner un poco más de iniciativa por su parte.

Los ejemplos de las lecciones anteriores eran prácticas totalmente guiadas, comando a comando, pero a partir de ahora tendrá que probar cosas no necesariamente idénticas a los ejemplos que nosotros utilicemos.

No se preocupe. Seguro que lo hará muy bien. Si ha llegado hasta aquí es señal de haber conseguido un avance significativo en sus conocimientos, y como Linuxero se encuentra quizás a medio camino entre el novato bruto y extremadamente torpe y el repugnante listillo informático.

Confiamos plenamente en su capacidad pero recuerde que nosotros no somos responsables de lo que pueda pasarle a su ordenador. Continúe practicando con un usuario distinto de root y un directorio que no contenga información valiosa.

Seguramente en esta lección aprenda cosas que algunas personas que llevan usando 'vi' durante varios años aún no conocen, pero también

contiene cosas que pueden ser muy importantes para usted. No todos usamos las mismas cosas. La mayoría de los documentos que explican el uso de 'vi' lo hacen de forma muy incompleta. Nosotros no vamos a hacer un tratado completo de este editor porque necesitaríamos muchas lecciones y nos saldríamos de los objetivos del curso pero si procuraremos mostrarle un amplio abanico de posibilidades para que tenga donde elegir. En la primera lección sobre 'vi' le mostramos un juego reducido de comandos pero suficiente para defenderse. En esta lección le damos la posibilidad de convertirse en un virtuoso de este editor.

Búsquedas

Desde el modo comandos se puede usar '/' para buscar. Admite expresiones regulares. Por ejemplo si introduce '/debian.com' podría localizar una cadena 'debian-com' o 'debian:com' y no solo 'debian.com'. Recuerde que algunos caracteres como el punto el asterisco, etc tienen un significado especial y para usarlas en 'vi' hay que precederlas de '\' que actuará como carácter de escape. También hay que escapar los caracteres como por ejemplo '/' y '&' porque sin tener significado como operador de expresión regular si tiene un significado especial para 'vi' que veremos cuando expliquemos las sustituciones. No vamos a volver a explicar las expresiones regulares porque ya dedicamos una lección. Practique lo que va aprendiendo y recuerde que las expresiones regulares no funcionan igual en todas las aplicaciones. Vamos a poner varios ejemplos para su uso en búsquedas desde 'vi'. Para buscar '</TABLE>' habría que usar '/<\/ TABLE>'.

Es decir que algunos caracteres no los podemos usar tal cual para buscarlos sino que hay que es caparlos. En el caso anterior hemos escapado '/'. Tenga en cuenta esto cuando intente por ejemplo buscar un carácter '$' en un texto.

Con 'n' buscará la ocurrencia siguiente de la cadena en el fichero. Con 'N' buscará la ocurrencia anterior. Una vez llegado al final comienza de nuevo la búsqueda por el principio.

También se puede empezar la búsqueda desde el primer momento en sentido ascendente usando '?' en lugar de '/'. En este caso los comandos 'n' y 'N' también funcionaran en sentido contrario que con el comando '/'.

Si usamos '/' o '?' sin nada más a continuación, se repetirá la última búsqueda.

Las expresiones regulares ofrecen grandes posibilidades como ya hemos visto pero particularmente interesantes para 'vi' pueden ser las búsquedas a principio o a final de una línea. En el ejemplo buscaremo 'El' a principio de una línea y luego 'as' al fineal de línea.

```
'/^El'
'/as$'
```

Sustituciones

Ya dispone de un resumen de todos los comandos de 'vi' que proporcionamos en el capítulo anterior así que lo mejor es presentar una serie de ejemplos explicando para que sirven.

Vamos a poner tres formas distintas de sustituir 'encarnado' por 'colorado' en las tres primeras líneas de un texto, y solo deseamos que ocurra la sustitución en la primera ocurrencia de cada línea caso. Es decir si una línea contiene varias veces la palabra 'encarnado' solo deseamos sustituir la primera.

```
:1,3 s/encarnado/colorado/
:1,3 s:encarnado:colorado:
:1,3 sçencarnadoççoloradoç
```

Ahora de dos formas distintas vamos a sustituir en todo el texto 'encarnado' por 'colorado'. Es decir si una linea contiene varias veces la palabra 'encarnado' deseamos que la sutitución se realice en todas ellas. Para eso utilizaremos un comando similar al anterior pero finalizado en

'/g'. Los ejememplos solo varían en la forma de indicar con '1,$' o con '%' la búsqueda en todo el texto.

```
:1,$ s/encarnado/colorado/g
:% s/encarnado/colorado/g
```

Ahora si sustituimos en todo el texto 'encarnado' por 'colorado' pero confirmando cada cambio manualmente. Para eso utilizaremos un comando similar al anterior pero finalizado en '/gc'. La 'g' tiene el mismo significado que en el caso anterior y la 'c' significa 'confirmación'.

```
:1,$ s/encarnado/colorado/gc
```

Vamos a usar ahora las sustituciones con expresiones regulares.

1) Poner a doble línea todo un texto.

```
:%s/$/\
/g
```

2) Eliminar blancos y tabuladores a final de linea.

```
:%s/[space tab]*$//
```

3) Eliminar blancos y tabuladores al principio de línea.

```
:%s/^[space tab]*//
```

4) Sustituir cualquier secuencia de blancos y tabuladores por un único blanco.

```
:%s/[space tab][space tab]*/ /
```

5) Poner entre paréntesis todos los números enteros de un texto.

```
:%s/[0-9][0-9]*/\(&\)/g
```

En este último hemos usado el operador '&' en la misma forma que vimos con 'sed' unas lecciones atrás. Se pueden hacer en poco tiempo cosas bastante complejas. Por ejemplo vamos a hacer un shell-script que genere otro shell-script. Para ello primero lo que haremos es escribir el programa que deseamos generar.

```
echo 'kkkk'
echo 'kkkk'\''hhhh'
```

Para generarlo bastará tratar cada línea poniendo un "echo '" al comienzo de cada línea y una comilla simple al final "'" de cada línea, pero antes escaparemos cualquier posible comilla simple que pudiera existir. Para ello editamos el texto anterior y hacemos lo siguiente:

```
:% s/\'/\'\\\'\'/g
:% s/^/echo \'/
:% s/$/\'/
```

Deberá quedarnos de esta forma.

```
echo 'echo '\''kkkk'\'''
echo 'echo '\''kkkk'\''\'\'''\''hhhh'\'''
```

Bueno lo hemos realizado con 'vi' y a pesar de lo complicado del resultado solo hemos realizado tres operaciones sencillas. Ahora ejecute este fichero y verá como obtiene el primero.

Bueno si no conoce C no le entusiasmará este otro ejemplo pero eso de hacer programas que generan programas nos sirve para practicar las sustituciones usando 'vi'. Podemos escribir el programa en C que deseamos generar y luego tratarlo sustituyendo todas las '\' por '\\' y todas las '"' por '\"'. Es decir escapamos esos dos caracteres que darían

problemas más tarde. Luego añadimos a principio de cada línea 'printf (y finalizamos cada línea con "\n").

Las sustituciones que acabamos de mencionar serían de esta forma.

```
:% s/\\/\\\\/g
:% s/"/\\"/g
:% s/^/printf("/
:% s/$/\\n");/
```

Edición de varios ficheros secuencialmente

Al editor 'vi' se le puede pasar como argumento no solo un fichero sino varios para ser editados en secuencia. Por ejemplo:

```
$ vi fich1 fich2 fich2
```

Bueno esto puede ser más útil de lo que parece a simple vista. Vamos a suponer que tenemos que modificar una gran cantidad de ficheros para sustituir en muchos de ellos la cadena 'acastro' por la cadena 'A.Castro' pero tampoco lo podemos hacer de forma automática porque nos conviene ver caso a caso. Por eso pasamos a 'vi' el resultado de una búsqueda con 'grep'.

```
$ vi `grep -il acastro *.txt`
```

Una vez dentro del editor hacemos lo siguiente.

```
:args
```

El comando ':args' nos informará del nombre de los ficheros que vamos a editar secuencialmente. Intente probar esto creando sus propios ficheros de prueba. Ya sabe como hacerlo. Conviene que todos ellos tengan una cadena idéntica para ser localizada con grep. Una vez que el editor 'vi'

este preparado para editar secuencialmente varios ficheros busque la cadena con el operador '/' y aunque pasemos a editar el siguiente fichero se conservará almacenada la cadena de búsqueda. Esta forma de trabajar facilita mucho la modificación de varios ficheros.

Para editar ficheros secuencialmente le conviene tener en cuenta los siguientes comandos. Algunos no los habíamos comentado todavía.

```
:args   Obtiene la lista de los ficheros
:next   Pasa al fichero siguiente
:prev   Vuelve al fichero anterior
:rewind Vuelve al primero
:rewind! Vuelve al primero abandonando los cambios en el
fichero actual
:w      Salva los cambios del fichero anterior
:q      Sale del editor. (Solo si ya se han editados todos)
:q!     Sale del editor incondicionalmente.
```

Edición simultanea de ficheros

Podemos pasar de la edición de un fichero a la edición de otro usando ':e fichero' pero antes deberemos salvar el contenido de la edición actual. También podemos forzar el paso a editar otro fichero con ':e! fichero' pero en ese caso se perderán las modificaciones realizadas. Usando ':e#' volvemos a retomar la edición del fichero anterior situados en la misma línea donde lo dejamos. Intente practicar lo siguiente editando unos ficheritos de prueba.

```
:e fich1
:w
:e fich2
:e! fich3
:e#
```

Visualización de caracteres de control

En 'vi' los caracteres de control se visualizan como un acento circunflejo y una letra. Por ejemplo <CTRL+K> se visualizará como '^K' y la manera

de distinguir un carácter circunflejo real seguido de una 'K' es moviendo el cursor para ver si de desplaza uno o dos caracteres cuando pasamos por el carácter circunflejo. También se usa con similar significado el carácter tilde '~' pero para caracteres de 8 bits.

Introduciendo caracteres de control en el texto

Imaginemos que deseamos introducir un carácter de control para provocar un salto de página en la impresora. Para ello deberemos introducir un <CTRL+L> que será visualizado como '^L'. La forma de hacerlo es pulsar desde el modo inserción el carácter <CTRL+V> y a continuación el carácter <CTRL+L>. Es decir el carácter <CTRL+V> actúa como carácter de escape. Podemos pulsar <CTRL+V> y luego una tecla de función, u otras teclas especiales.

También se pueden buscar caracteres de control. Para ello tambíen hay que introducirlos usando <CTRL+V>.

Edite un fichero de prueba y proceda a poner varios saltos de página en distintos lugares. Luego busque con '/' y con '?' los saltos de página. Pruebe a insertar los códigos de las teclas <ENTER> y <ESC>, mediante <CTRL+V><ENTER> y <CTRL+V><ESC> respectivamente. Deberán aparecer como '^M' y como '^['.

El comando tilde

El comando '~' sustituye el carácter situado bajo el cursor de mayúscula a minúscula y viceversa avanzando luego el cursor.

Operaciones con bloques

Se pueden marcar hasta 26 posiciones ('a'..'z') para un conjunto de uno o mas ficheros editados simultáneamente. Permite trabajar marcando principio y fin de bloques para mover, copiar o borrar en varios ficheros. Se pueden usar las marcas en cualquier comando en sustitución de '.', '$', o

los números de línea como ya hemos visto. La ventaja es que si estamos insertando o borrando líneas tendríamos que volver a comprobar los números de líneas. Por el contrario si hemos colocado una marca en una línea continuará en esa línea mientras no la eliminemos. Para colocar una marca podemos situarnos con el cursor en la línea deseada y luego hacer ':k <letra-minúscula>' ':ma <letra-minúscula>' ':ma <letra-minúscula>' o directamente sin pasar al modo ex mediante 'm <letra-minúscula>'. Para practicar con ello edite un fichero de prueba y coloque varias marcas en distintas líneas. Luego realice distintas operaciones como por ejemplo el borrado de bloques. Compruebe como desaparece una marca cuando se borra la línea que la contiene. Las marcas no se ven pero si intenta usar una marca que no existe obtendrá un error. Compruebe como reaparece la marca cuando deshace la operación con 'u' (undo). Algunas formas de utilización de marcas son las siguientes.

```
:'a,'b de
:'b,'b+1 de
:'a,. de
:'a,'b co 'c
:'a,'b mo 'c
```

También podemos posicionarnos en el lugar de una marca pulsando comilla-simple y <letra-minúscula> desde el modo comandos. Un detalle interesante es que siempre podemos volver a la línea donde nos encontrábamos, la última vez que dimos un salto de posición en el texto. Para ello pulsamos '' (comilla-simple dos veces).

Repetición del último comando

Cuando hay que hacer varias veces la misma cosa resulta tremendamente útil poder repetir el último comando. Imaginemos que desde el modo de comandos hacemos en una línea 'RLínea <ESC>' para sustituir 6 caracteres por la cadena 'Línea ' Inmediatamente después de ello y antes de hacer ninguna otra cosa podríamos situarnos en otra línea y pulsar '.' (punto) para repetir ese mismo comando y luego cambiar a otra y otra línea y volver a pulsar '.' punto a cada vez para efectuar el mismo cambio

en varios sitios. Tambíen mencionamos la posibilidad de ejecutar un comando del sistema operativo lanzando una sub-shell. Pues bien este tipo de comandos ': <comando>' puede ser repetido usando ':!!'

Tabulación

El carácter con la tecla <TAB> o con el carácter <CTRL+I> insertará un carácter de tabulación. El ancho de la tabulación será por defecto igual a 8 espacios. Esto puede ser variado por ejemplo a 5 con ':set tabstop=5'. Si deseamos añadir un tabulador al principio de línea en las siguientes 7 lineas a la posición del cursor podemos hacer. '7>>'. Tambíen se podría haber hecho lo mismo con ':.,.+7 >>'. Para insertar un tabulador a principio de cada línea del texto sería ':% >>' Cuando se usa el editor para editar fuentes de programas puede resultar útil usar la indentación automática. 'set ai' para eliminar la indentación automática usaremos 'set noai'. Si en alguna ocasión necesita sustituir todos los tabuladores por caracteres en blanco use el comando 'expand'. No es un comando de 'vi' sino del SO. Consulte con el man su sintaxis y las distintas opciones. Para hacer una prueba edite un fichero insertando tabuladores y luego conviertalo de la siguiente forma.

```
expand fichero > ficheroexpandido.
```

Abreviaturas

En 'vi' hay muchos detalles pensdos a ahorrar pulsaciones de declas. Vamos a ver el comando ':ab' Si está acostumbrado al uso de abreviaturas le resultará muy práctico. Puede definir las abreviaturas que quiera y una vez tecleadas en modo entrada de dados (insercion o sobrescritura) serán sustituidas por la cadena que desee. Vamos a usar dos abreviaturas una 'ci' para 'Ciberdroide Informática' y otra 'INMHO' para 'en mi humilde opinión'. También vamos a listar las abreviaturas definidas.

```
:ab ci Ciberdroide Informática
```

```
:ab INMHO en mi humilde opinión
:ab
```

Use sus propias abreviaturas en un texto y observe como la abreviatura no será interpretada si la secuencia de caracteres forma parte de una palabra.

Macros

Es posible programar macros. Por ejemplo vamos a hacer una macro que mueva 4 líneas al final del fichero

```
:map xxx 4dd:$^Mp
:map
```

Con esta macro ocurrirá algo. En primer lugar si teclea las tres 'x' seguidas se ejecutarán los comando '4dd' (borrar cuatro lineas), ': $<ENTER>' (ir a la última línea) y 'p' (recuperar el contenido del buffer) Si pulsamos una sola 'x' notaremos que no pasa nada inmediatamente pero luego se eliminará un carácter tal y como ocurre normalmente. Es decir la macro tiene un tiempo para ser tecleada completamente en caso contrario no actúa. Para ver todas las macros definidas usaremos ':map'

Podemos eliminar la macro que acabamos de crear con 'unmap'

```
:unmap xxx
```

Muchas veces se usan las macros para definir comandos asociados a teclas especiales. Para ello hay que conocer la secuencia de caracteres que envían esas teclas. Si en modo inserción usamos <CTRL+V><F2> obtendremos '^[[[B'. Pero también podemos usar '<F2>' para definir macros. Vamos a poner algunos ejemplos prácticos.

F2 (Filtro PGP) Firmar.

```
:map [[B 1G!Gpgp -satf 2>/dev/tty
```

Sería lo mismo si usaramos '<F2>' en lugar de '^[[[B'.

```
:map <F2> 1G!Gpgp -satf 2>/dev/tty
```

F3 (Filtro PGP) Encriptar

```
:map [[C 1G!Gpgp -eatf 2>/dev/tty
```

F4 (Filtro PGP) Desencriptar y comprobar firma

```
:map [[D 1G!Gpgp  2>/dev/tty; sleep 7
```

F5 (Filtro ispell) Corrector ortográfico.

```
:map [[E :w
:!ispell %
:e!
```

Opciones de vi

Sería absurdo pretender explicar en una lección todo el editor 'vi'. Tampoco hay necesidad de ello. Hay montones de opciones y ya hemos visto algunas. Si el editor es 'vim' en lugar de 'vi' el número de posibilidades es aun mayor. Puede ver las opciones disponibles y sus valores actuales usando el comando ':set all' desde el modo de comandos. No se asuste. No ha que conocerlas todas. Hay opciones que seguramente la persona que las implementó ya ha olvidado para que sirven. Una opción se puede indicar muchas veces de dos formas. Una en forma completa y otra en forma abreviada. Por ejemplo ':set nu' en lugar de ':set number' y para desactivar la opción se suele usar la misma opción

precedida de 'no'. Por ejemplo ':set nonu' o ':set nonumber'. Las opciones que deben tomar un valor numérico se establecen con el nombre de la opción seguida de '=' y el valor. Por ejemplo ':set tabstop=5'

Configuración del editor

El fichero más estándar para configurar 'vi' en cualquier SO de tipo Unix es '~/.exrc'. Como alternativa se puede usar ficheros más específicos para el editor concreto que estemos usando. Por ejemplo en el caso de 'vim' tenemos '~/.vimrc'. Compruebe si existe un ejemplo '/etc/vimrc' en su Linux y mire su contenido. Se trata de un fichero de ejemplo para copiarlo en ' ~/.vimrc' Contiene algunas configuraciones que pueden resultar útiles. El carácter comilla doble '"' se usa para considerar el resto de la línea como un comentario. Si considera que el editor tiene un comportamiento que usted considera incómodo quizás se deba a la existencia de algunas opciones que no se adaptan al uso que usted da a su editor. Por ejemplo para editar programas C puede resultar cómodo la auto- indentación pero no le gustará que parta las líneas automáticamente cuando excedan un determinado ancho. También es buena idea incluir en su fichero de configuración las abreviaturas y las macros que desee usar. Ya hemos visto algunas y seguramente ya se ha preguntado como hacer para no tener que introducirlas cada vez. Es bueno perder un poco de tiempo en personalizar el editor.

Tag

Para editar fuentes C por ejemplo resulta muy practico. Supongamos que tenemos un fuente llamado pingpong.c. Crearemos en primer lugar un fichero tags ejecutando "ctags pingpong.c" (ver ctags(C)). Se crea un fichero tags que contendrá por ejemplo lo siguiente:

```
plot    pingpong.c      ?^plot(x, y, ch)$?
salir   pingpong.c      ?^salir(s)$?
```

Con ello se referencian las funciones. Para editar el fichero podemos hacer vi -tplot. Aparecera el cursor en la posición de esa función. Si

introducimos :ta salir desde el modo de comandos nos situaremos en esa función. Se pueden crear tags para varios ficheros simultáneamente mediante ctags */*.[cfg] (ver ctags(C)). De esta forma podríamos editar cómodamente varios fuentes.

PROGRAMACION SHELL-SCRIPT (Primera Parte)

Que es un shell-script

Ya vimos hace algunas lecciones una introducción al interprete de comandos de Linux (shell). Vamos a ampliar nuestros conocimientos sobre la shell y si bien antes consideramos a la shell como un interprete de comandos en esta lección veremos que es mucho más que eso. La shell de Linux que vamos a usar es la Bash que como ya dijimos es un superconjunto de la Bourne-Shell. Sin lugar a dudas esta lección le permitirá dar un salto enorme en el aprovechamiento de su SO Linux.

Con esta lección y las siguientes nos acercamos al momento en que usted podrá presumir ante algunos de sus amigos de haber realizado en su flamante SO cosas que ellos no saben hacer con Güindos, aunque ellos podrían presumir de su habilidad cerrando las ventanas del General Failiure y del Doctor Guasón.

El lenguaje shell-script es muy versátil aunque hay que advertir que es bastante ineficiente. Son mucho más eficientes los programas escritos en lenguaje C. Este es el lenguaje en el que se escribió el kernel de Linux y otros muchos SO. El Bash no es tan eficiente como el C. El valor del lenguaje shell-script es que permite hacer cosas complicadas con muy poco esfuerzo en perfecta combinación con una serie de comandos también muy potentes disponibles en Linux . Verdad que suena interesante ? Algunas partes del SO que no pertenecen al kernel están escritas en shell-script. Por ejemplo muchos comandos que se van ejecutando en secuencia mientras el sistema arranca son programas realizados en shell-script así que la ineficiencia no es ningún obstáculo para ciertas tareas. Por el contrario para un administrador tener ciertos programas del sistema en shell-script le permite retocarlos a su gusto con gran facilidad.

Después de hacer login en el sistema obtendremos una sesión con el intérprete de comandos asociado a nuestro usuario. (Esta asociación figura en el fichero '/etc/passwd' junto quizás con la clave encriptada y otras cosas). Decimos quizás porque la clave no siempre es guardada en este fichero. La razón es que pese a estar encriptada /etc/passwd es un fichero que cualquiera puede leer y resulta más seguro otro sistema llamado shadow. Que /etc/passwd sea legible nos permite consultar entre otras cosas cual es nuestra shell en el inicio de una sesión. Desde esta sesión podemos ir introduciendo ordenes desde el teclado pero nuestro interprete de comandos no deja de ser a su vez un programa normal como cualquier otro, así que si tuviéramos que teclear varias cosas podríamos guardar estas ordenes en un fichero y redirigir la entrada estándar de la shell para que tome la entrada desde el fichero en lugar de hacerlo desde el teclado.

Vamos a ver un ejemplo para mostrar fecha y hora en letras grandes. Teclee el comando siguiente:

```
$ banner `date '+%D %T'`
```

Ahora vamos a practicar repasando un poco las nociones que ya vimos en el primer capítulo dedicado al shell script porque son conceptos que conviene asentar. Para ello vamos a guardar este comando en un fichero para poder ejecutarlo cuando nos apetezca. Previamente nos situamos en el directorio '/tmp' como es costumbre y nos aseguramos de no estar usando el usuario root ni ningún otro usuario con privilegios especiales.

```
$ # Nos situamos en /tmp
$ cd /tmp
$ # Generamos el fichero con el comando
$ echo "banner `date '+%D %T'`" > fichero_ordenes
$ # Comprobamos el contenido del fichero
$ cat fichero_ordenes

banner 08/02/00 20:26:30
```

!! Vaya !! Esto no es lo que queríamos hacer. Este comando siempre mostraría la misma fecha y hora y seguramente no pretendíamos eso. Dado que hemos usado las dobles comillas se ha producido la expansión de la línea de órdenes concretamente el operador grave. Si en lugar de las dobles comillas usáramos las comillas simples no se habría producido ese error pero existen ya comillas simples dentro del propio comando y la shell tomaría el cierre de la comilla simple en un lugar que no es el que nosotros deseábamos. Todo esto lo comentamos a modo de repaso. ¿Entonces como tendríamos que haber generado este comando? Aquí es donde el listillo dice que usando un editor de texto. Bueno al fin de cuentas ya sabemos usar el 'vi' ¿no? Bueno pues vamos primero a hacerlo sin editor. Recuerda que mencionamos la posibilidad de escapar un solo carácter con el carácter '\' ? Intentemos de nuevo.

```
$ # Generamos el fichero con el comando
$ echo "banner \`date '+%D %T'\`" > fichero_ordenes
$ # Comprobamos el contenido del fichero
$ cat fichero_ordenes

banner `date '+%D %T'`
```

Ahora si. Vamos a usar la shell de la misma forma que haríamos con un programa normal redirigiendo la entrada estándar desde este fichero. Por lo tanto el fichero tiene tratamiento de datos y solo se requiere permiso de lectura para ejecutarlo de esta forma.

```
$ # Ejecutamos
$ bash < fichero_ordenes
$ # También podemos ejecutarlo de otra forma
$ cat fichero_ordenes | bash
```

Para ejecutarlo sin redirigir entrada salida podemos pasar como argumento el nombre del fichero. Nuevamente solo hará falta permiso de lectura.

```
$ # Ejecutar pasando el nombre del fichero como
argumento
$ sh fichero_ordenes
$FIN
```

Ejecución de un shell-script directamente como un comando

Para ejecutarlo directamente como un comando necesitamos darle permiso de ejecución.

```
$ # Ejecutar pasando el nombre del fichero como
argumento
$ chmod +x fichero_ordenes
$ ./fichero_ordenes
$FIN
```

Lo del './' antes del ejecutable resultará necesario si el directorio donde se encuentra el ejecutable no figura en la variable '$PATH' En realidad ocurre algo muy similar al caso anterior porque también se arranca una sub-shell que toma como entrada el fichero.

Esto ocurre cuando la shell detecta que el ejecutable está en formato de texto. Si se tratara de código binario lo ejecutaría sin arrancar una sub-shell. Para averiguar que tipo de fichero se trata, la shell mirará los primeros cráteres que contiene. Los ficheros ejecutables suelen tener un par de bytes iniciales que servirán para identificar el tipo de fichero. En caso contrario se asume que es un shell-srcip.

A estos dos caracteres iniciales juntos forman lo que se denomina número mágico. Es solo una cuestión de nomenclatura. El SO lo ve como un número entero de dos bytes y nosotros desde un editor lo vemos como dos caracteres.

Existe un programa llamado 'file' que utiliza esta información así como la presencia de ciertos patrones para identificar el tipo de un fichero.

Se usa el término script para ficheros que son legibles y ejecutables. Es decir el propio fuente del programa es ejecutable. Fuente de un programa es lo que escribe el programador. El lenguaje C no es un lenguaje tipo script ya que el fuente no es ejecutable. Hace falta un compilador que traduce el código fuente a código propio de la máquina. Los Script no se ejecutan directamente en la máquina sino dentro de un programa que va interpretando las instrucciones. Bueno en general esto es más o menos así.

Hay muchos tipos de scripts. La shell tiene su propio lenguaje. Existe un lenguaje distinto llamado 'perl' que también se ejecuta como un script. 'perl' es un lenguaje inspirado en el uso combinado del interprete de comandos y algunos comandos clásicos de los SO tipo Unix, y además incorpora cosas de otros lenguajes como el 'C' y otras cosas totalmente originales. 'perl' no es fácil de aprender pero es un lenguaje de script muy potente. Un script de 'perl' se puede ejecutar mediante el comando 'perl prueba.pl'. El fichero 'prueba.pl' deberá estar en lenguaje 'perl'. Si ejecutáramos directamente './prueba.pl' desde bash, el interprete de comandos intentará ejecutarlo como si fuera un script de bash dando errores de sintaxis. Para evitar esto se puede poner totalmente al principio del fichero los caracteres '#!' Estos caracteres serán vistos como un número mágico que avisa que a continuación viene el programa a usar con ese script. Para el script de perl debería haber empezado con '#!/usr/bin/perl' para poderlo ejecutar directamente sin problema. Para un script de bash no es necesario poner nada pero podríamos haber puesto '#!/bin/bash', o '#!/bin/sh'. De esta forma también se puede indicar cual es la shell que deseamos usar ya que podríamos desear usar otras shells como '#!/usr/bin/csh' para la C-shell o '#!/usr/bin/ksh' para la Korn shell. A nosotros con el bash nos basta y sobra.

Ejecución con la shell-actual

Hasta este momento siempre hemos arrancado una sub-shell que leía las ordenes del fichero, las ejecutaba y después terminaba y moría cediendo el control nuevamente a la shell original que arrancó la sub-shell.

Existe una forma de decirle a la shell actual que lea y ejecute una serie de órdenes por si misma sin arrancar una sub-shell. Para ello hay que anteponer un punto y un blanco al nombre del comando. Nuevamente solo hará falta permiso de lectura.

```
$ # Ejecutar ordenes de un fichero desde la shell
actual
$ . ./fichero_ordenes
```

En este caso no se ha ejecutado mediante una sub-shell pero no se aprecia ninguna diferencia. Esto no siempre es así. Veamos un ejemplo en el que se aprecie la diferencia. $$ tomará el valor del pid de la shell en ejecución y $PPID tomará el valor del pid del proceso padre.

```
$echo echo \$PPID \$$ > fichero_ordenes
$ bash fichero_ordenes

213 2396

$ . ./fichero_ordenes

1 213
```

Evidentemente cuando pruebe este ejemplo obtendrá un pid distinto de 213 pero lo interesante es ver como ambas ejecuciones establecen claramente que la primera fue efectuada por un proceso hijo de la shell que se ejecuta en segundo lugar usando el punto, seguido del espacio y del 'fichero_ordenes'.

Los procesos hijos heredan el entorno (variables y otras cosas) desde el proceso padre por lo cual un padre si puede comunicarse con su hijo de esa forma. Por el contrario un proceso hijo (sub-shell en nuestro caso) jamás puede comunicarse de esa forma con su padre. Resulta imposible que un proceso hijo comunique valores de variables del entorno a su padre. Si deseamos modificar variables del entorno ejecutando un shell-script no podremos hacerlo nunca si la ejecutamos desde dentro de una

sub-shell. Por el contrario deberemos usar algo del tipo '. ./fichero_ordenes' como acabamos de ver.

Existe un fichero en el directorio inicial de cada usuario que consiste en una serie de ordenes iniciales para shell. En el caso de la Bash este fichero es '~/.bash_profile'. Podemos ejecutarlo si queremos pero este tipo de ficheros incluyen ordenes para inicializar variables entre otras cosas, así que lo adecuado sería ejecutarlo desde la shell actual. Haga lo siguiente:

```
$ cd
$ . ./.bash_profile
```

Observe que hay tres puntos en el último comando con tres significados distintos. El primero es para ejecutar con la shell-actual, el segundo es para indicar el directorio actual, y el último forma parte del nombre y se usa para que no sea tan visible dado que un punto al principio del nombre de un fichero no será expandido por la shell cuando usemos '*'.

Vamos a comprobar la acción del número mágico. Pase a un directorio para practicar sus ejercicios, y edite un nuevo fichero que debe llamarse 'aviso'.

```
echo "Error este shell-script debió arrancarse con . /
comando"
```

Ahora edite otro nuevo fichero que llamaremos 'comando'.

```
#!/bin/bash /tmp/aviso
echo "COMANDO"
```

Vamos a ejecutar nuestro shell-script de dos formas distintas y observamos que el resultado no es el mismo. En el segundo caso se ejecuta el script 'aviso'. Este último ejemplo no es completo ya que

'/bin/bash /tmp/aviso' no actuará interpretando sino de una forma independiente del texto y mostrará siempre un mensaje de aviso.

```
$ # (1) Se ejecuta usando una sub-shell pero en este
$ # caso se pasa un argumento a la sub-shell y por ello
$ # no se ejecuta el código del fichero comando.
$ chmod +x aviso
$ ./comando

Error este shell-script debió arrancarse con . /comando

$ # (2) Ahora ejecutamos con la shell actual

$ . ./comando

COMANDO
```

La explicación es la siguiente. El número mágico indica con que shell se ejecutará el script. Nosotros hemos puesto en lugar una shell sin argumetos que ejecutaría ese texto una shell que recibe un argumento. Eso hace que se ejecute el comando pasado como argumento en lugar de tomar el texto del fichero que contiene el número mágico. Cuando ejecutamos con la shell actual la información siguiente al número mágico no será tenida en cuenta porque supondría arrancar una sub-shell.

Es un ejemplo rebuscado pero hay que intentar comprender lo que sucede en distintas situaciones porque el lenguaje shell-script tiene sus rarezas.

Ejecutar en una subshell usando paréntesis

Ya que estamos con el tema de la subshell vamos a comentar como funciona el uso de los paréntesis desde la linea de órdenes. No es necesario realizar un script para ejecutar algo en una subshell (lista) Ejecuta una lista de ordenes en una subshell.

```
$ var1=888
$ ( var1=111; echo $var1 )
```

```
111
$ echo $var1

888
```

Estamos avanzando muy deprisa, y por eso tenemos que practicar bastante para consolidar los conocimientos. No siempre resulta trivial averiguar como funcionan las cosas. Observe el siguiente ejemplo:

```
$ echo $$ ; ( echo $$ ; ( echo $$ ) )

218
218
218
```

Nuevamente el resultado puede ser distinto en su ordenador pero lo importante es que 'echo $$' parece obtener el mismo pid pese a ocurrir en distintas sub-shells. Cuando parece ocurrir algo distinto de lo que nosotros esperábamos resulta muy útil usar el modo traza con 'set -x'. Repetimos el ejemplo misterioso de antes.

```
$ set -x
$ echo $$ ; ( echo $$ ; ( echo $$ ) )

+ echo 218
218
+ echo 218
218
+ echo 218
218
$ set +x
```

Misterio aclarado. Realmente no ejecutamos ningún 'echo $$' sino que ya se ejecuta directamente 'echo 218' porque la shell lo primero que hizo antes de ejecutar las ordenes fué expandir cada '$$' presente en la línea de ordenes.

Ahora que ya sabemos que es un shell-script vamos a ir aprendiendo cosas pero no se pretende una descripción exhaustiva de la Bash. Puede consultar con man bash las dudas que tenga. Recuerde que el man no es una herramienta de estudio sino de consulta. Cuando consulte 'man bash' verá cantidad de información relativa a cosas que no hemos explicado y man no explica casi nada. Por ello muchas cosas de 'man bash' no las entederá aunque siempre queda el recurso de probar las cosas que no se comprenden para ver como funcionan.

Se irán proponiendo ejemplos que ilustren unos cuantos conceptos básicos. Una vez que haya practicado con estos ejemplos habremos conseguido dar los primeros pasos en la programación de shell-script que son los que más cuestan. Después un buen método para continuar aprendiendo a programar en shell-script es intentar comprender las shell-script que encontremos en el sistema con ayuda del manual online de unix.

El lenguaje shell-script es un poco especial con relación a otros lenguajes ya que los blancos y los saltos de línea tienen un significado que hay que respetar y la expansión de ordenes es algo que puede resultar desconcertante. Eso lo hace un poco antipático al principio. Es importante tener una buena base de conocimientos que en estos momentos ya tenemos, y continuar practicando.

Paso de parámetros a un shell-script

Imaginemos que usted tiene dificultad para pasar de Ptas a Euros, y quiere hacer un programita para pasar de una moneda a otra cómodamente. Edite el comando siguiente, y luego ejecutelo en la forma 'pta2eu 35':

```
#pta2eu
echo $1 '/ 166.386' | bc -l
```

En este caso $1 actúa como una variable que tomará el valor del primer parámetro. $0 toma el valor del propio comando ejecutado y $2, $3 etc el

valor de los parámetros segundo y tercero respectivamente. La variable $# tomará el valor del número de parámetros que usemos. La variable $* toma el valor de todos los parámetros concatenados. Vamos a poner un ejemplo que llamaremos 'pru_parametros'. Debe editarlo, darle permisos de ejecución con chmod +x por ejemplo y ejecutarlo pasando distintos parámetros. './pru_parametros', './pru_parametros hola', './pru_parametros pararm1 param2 param3'.

```
# pru_parametros
echo "Numero de parámetros = $#"
echo '$0=' $0
echo '$1=' $1
echo '$2=' $2
echo '$3=' $3
echo '$*=' $*
```

La ejecución de esta prueba quedaría como sigue.

```
$ ./pru_parametros

Numero de parámetros = 0
$0= ./pru_parametros
$1=
$2=
$3=
$*=
$ ./pru_parametros hola

Numero de parámetros = 1
$0= ./pru_parametros
$1= hola
$2=
$3=
$*= hola
$ ./pru_parametros  pararm1 param2 param3

Numero de parámetros = 3
$0= ./pru_parametros
$1= pararm1
$2= param2
$3= param3
```

```
$*= pararm1 param2 param3
$
```

No existe la posibilidad de usar $10 para acceder al parámetro décimo. En lugar de acceder de esta forma habría que usar un comando interno llamado 'shift'. El efecto de este comando es desplazar los argumentos una posición. De esta forma $1 se pierde y $2 pasará a ser $1, $3 pasará a ser $2, etc. Si existían más de 10 parámetros ahora el décimo que antes era inaccesible pasará a ser $9. $0 no varía. Se pueden desplazar varios parametros de un golpe pasando un número a la función 'shift' por ejemplo 'shift 5' desplazaría a 5 parámetros.

Vamos a editar un fichero de ejemplo que llamaremos 'pru_shift': Debe editarlo, darle permisos de ejecución con 'chmod +x pru_shift' y ejecutarlo pasando distintos parametros. './pru_shift pararm1 param2 param3'.Observe que es una prueba parecida a la anterior. Intente utilizar sus conocimientos de vi para no tener que teclear tanto. Recuerde que puede leer un fichero desde dentro del editor y que puede duplicar bloques de información de varias formas.

```
# pru_shift
echo "Numero de parámetros = $#"
echo '$0=' $0
echo '$1=' $1
echo '$2=' $2
echo '$3=' $3
echo '$*=' $*
shift
echo "Numero de parámetros = $#"
echo '$0=' $0
echo '$1=' $1
echo '$2=' $2
echo '$3=' $3
echo '$*=' $*
```

La ejecución de esta prueba quedaría como sigue.

```
$ ./pru_shift  pararm1 param2 param3

Numero de parámetros = 3
$0= ./pru_shift
$1= pararm1
$2= param2
$3= param3
$*= pararm1 param2 param3
Numero de parámetros = 2
$0= ./pru_shift
$1= param2
$2= param3
$3=
$*= param2 param3
```

PROGRAMACION SHELL-SCRIPT (Segunda Parte)

Funciones

Se pueden definir funciones que podrán ser llamadas por su nombre permitiendo el paso de parámetros de la misma forma que cuando se llama a una shell. También puede devolver códigos de retorno. Las variables alteradas dentro de una función tienen efecto visible fuera de la función ya que una función no se ejecuta invocando a una sub-shell. Una función se puede definir en un shell-script o en la propia linea de comandos. Si se hace esto último será como si añadiéramos un comando interno.

```
$ fff() { echo "--------------------------" ; }
$ fff

--------------------------
```

Si tecleamos 'set' después de esto veremos todas las variables definidas y también aparecerá esta función que acabamos de crear.

```
$ set
```

Ahora vamos a editar un shell-script que use una función. Edite un fichero que llamaremos 'pru_funcion1' que tenga lo siguiente.

```
# pru_funcion1
# Funcion que muestra dos parametros de entrada y
modifica una variable
funcion1()
{
        echo '<'$1'>'
```

```
        echo '('$2')'
        var=1234
}
## main ##
funcion1 111 222
echo $var
```

Ahora damos permisos de ejecución y ejecutamos simplemente mediante './pru_funcion1'.

```
$ ./pru_funcion1
<111>
(222)
33
1234
```

Edite y pruebe el siguiente código.

```
# Visualizar los parametros del 6 al 14
function()
{
        shift 5
        echo $1 $2 $3 $4 $5 $6 $7 $8 $9
}
## main ##
funcion 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
```

Para definir la sintaxis de una función en shell-script podríamos usar lo siguiente:

_nombre_funcion_ () { _lista_de_ordenes_ }

Una lista de ordenes sería una o más ordenes separadas por algún operador como por ejemplo ';', '||', '&&', '&' o . Recuerde que el shell-script tiene un significado similar a ';' y si desea suprimir este significado deberá preceder inmediatamente con '\' para escapar el carácter y que no sea interpretado por la shell. En una cadena delimitada por comillas

simples o dobles no tiene este significado pero puede usarse '\' para evitar que el carácter quede insertado en la cadena.

```
$ echo \
> kkk\
> xxx

kkkxxx
```

Códigos de retorno

La palabra reservada 'return' interrumpe la ejecución de una función asignando un valor al código de retorno de esa función. Su uso no es necesario al final de la función. La palabra reservada 'exit' termina la ejecución de un shell-script liberando recursos y devolviendo un código de retorno el intérprete de comandos. 'exit 0' terminará retornando cero. Si solo ponemos 'exit' el código retornado será como si hiciéramos 'exit $?'. Es decir retorna el valor del último comando.

Ahora edite un fichero que llamaremos 'pru_funcion2' que tenga lo siguiente.

```
# pru_funcion2
# Funcion que devuelve un código de retorno. Y shell
que devuelve otro
funcion2()
{
        var=1234
        return 33
        var=4567
}
## main ##
funcion2
echo $?
echo $var
exit 888
```

Ejecutaremos este ejemplo

```
$./pru_funcion2

33
1234
```

Comandos true y false

Los comandos que finalizan correctamente suelen retornar 0. Los que finalizan con algún error suelen retornar un valor distinto de 0 y de esa forma distintos códigos de retorno pueden indicar distintas causas de error o no éxito.

En una expresión lógica donde lo que interesa es simplemente distinguir entre algo que se cumple o que no se cumple se tomará 0 como valor lógico TRUE (o sea cierto) y distinto de 0 se tomará como FALSE (falso).

'true' y 'false' son comandos que no hacen nada. Simplemente retornan el valor lógico TRUE o FALSE respectivamente. El valor de $? siempre devuelve el último código de retorno obtenido por la shell después de ejecutar el último comando.

Ejemplos:

```
$ true ; echo $?

0
$ false ; echo $?

1
```

Esperamos que ejemplos tan complicados como estos dos últimos no le desanimen para continuar con el curso.

Comando test

Como ya dijimos el lenguaje shell-script utiliza muy frecuentemente comandos externos del sistema operativo. Queremos explicar el comando 'test' antes de explicar las sentencias condicionales de bash porque de esta forma podremos usar ejemplos más completos.

'test' es un comando externo que devolverá un código de retorno 0 o 1 dependiendo del resultado de la expresión que siga a continuación.

Se usa muchísimo y tiene dos formatos. 'test <expr>' o '[<expr>]'. ambas formas son equivalentes.

Las expresiones de este tipo que solo pueden valer TRUE o FALSE se denominan expresiones lógicas.

Comprobación sobre ficheros	
-r <fichero>	TRUE si el fichero existe y es legible.
-w <fichero>	TRUE si el fichero existe y se puede escribir.
-x <fichero>	TRUE si el fichero existe y es ejecutable.
-f <fichero>	TRUE si el fichero existe y es de tipo regular (normal).
-d <fichero>	TRUE si existe y es un directorio.
-c <fichero>	TRUE si existe y es un dispositivo especial de caracteres.
-b <fichero>	TRUE si existe y es un dispositivo especial de bloques.
-p <fichero>	TRUE si existe y es un pipe (fifo).

-u <fichero>	TRUE si existe y tiene el bit set-user-ID.
-s <fichero>	TRUE si existe y tiene tamaño mayor que 0.
Comparación de cadenas	
-z <cadena>	TRUE si es una cadena vacía.
-n <cadena>	TRUE si es una cadena no vacía.
<cadena1> = <cadena2>	TRUE si ambas cadenas son idénticas. OJO hay dejar espacios a un lado y otro del signo igual para no confundir la orden con una asignación.
<cadena1> != <cadena2>	TRUE si son distintas.
<cadena>	TRUE si no es cadena nula. (Una variable que no existe devuelve cadena nula).
Comparación de números	
<num1> -eq <num2>	TRUE si ambos números son iguales.
<num1> -ne <num2>	TRUE si ambos números son distintos.
<num1> -gt <num2>	TRUE si <num1> mayor que <num2>.
<num1> -ge <num2>	TRUE si <num1> mayor o igual que <num2>.
<num1> -lt	TRUE si <num1> menor que <num2>.

<num2> <num1> -le <num2>	TRUE si <num1> menor o igual que <num2>.
Operadores lógicos	
! <expresión_logica>	Operador lógico NOT retorna TRUE si <expresión_logica> es FALSE y retorna FALSE si <expresión_logica> es TRUE.
<ExprLogi1> -a <ExprLogi2>	Operador lógico AND retorna TRUE si <ExprLogi1> y <ExprLogi2> son ambas TRUE y FALSE en caso contrario.
<ExprLogi1> -o <ExprLogi2>	Operador lógico OR retorna TRUE si <ExprLogi1> o <ExprLogi2> son alguna de ellas TRUE y FALSE en caso contrario.
Agrupación de expresiones	
(<expr>)	Se pueden usar paréntesis para agrupar expresiones lógicas. Es decir que valgan TRUE o FALSE

Ejemplos:

```
$ test -r /etc/passwd ; echo $?

0
$ test -w /etc/passwd ; echo $?

1
$ test -x /etc/passwd ; echo $?

1
```

```
$ test -c /dev/null  ; echo $?

0
$ test -r /etc/passwd -a -c /dev/null ; echo $?

0
$ test -w /etc/passwd -a -c /dev/null ; echo $?

1
$ test -r /etc/passwd -a -f /dev/null ; echo $?

1
$ [ -s /dev/null ] ; echo $?

1
$ [ ! -s /dev/null ] ; echo $?

0
$ [ "$$" = "zzzzzzzzzzz" ] ; echo $?

1
$ [ 0 -lt $$  ] ; echo $?

0
$ [ 0 -lt $$  -a  true ] ; echo $?
```

Comando expr

Este comando admite como parámetros los elementos de una expresión aritmética pero hay que recordar que algunos operadores deben ser escapados con '\'. Por ejemplo.

```
$ expr 11 \* 2
```

A diferencia de 'bc' no podemos usar valores demasiado altos porque el resultado no sería correcto. Recuerde que 'bc' trabaja con precisión arbitraria.

El comando 'expr' es útil pero no deja de ser un comando externo y por lo tanto su uso resulta ineficiente. Afortunadamente la propia shell tiene capacidad de evaluar expresiones aritméticas.

Expresiones aritméticas dentro de Bash

La shell por defecto asume que todas las variables son de tipo cadena de caracteres. Para definir una variable de tipo numérico se usa 'typeset -i'. Esto añade un atributo a esa variable para que la shell realice una expansión distinta sobre esta variable. Para evaluar una expresión aritmética asignándola a una variable usaremos 'let'.

- +	Menos unário y Más unário
* / %	Multiplicación, División, y Resto
+ -	Suma, y Resta
<= >= < > == !=	Comparaciones Menor o igual, Mayor o igual, Menor, Mayor Igualdad, desigualdad
= *= /= %= += -=	Asignación simple, Asignación con operador (multiplicación, división, resto, suma, y resta)

Ejemplos:

```
$ typeset -i j=7
$ typeset -i k
$ typeset -i m
$ echo $j

7
$ let j=j+3
$ echo $j
```

```
10
$ let j+=3
$ echo $j
13
$ let k=j%3
$ let m=j/3
$ echo '(' $m '* 3 ) +' $k '=' $j
+AMA
( 4 * 3 ) + 1 = 13
```

Puede que el operador '%' no le resulte familiar. Es el operador resto también se llama módulo y es lo que sobra después de la división. Lo cierto es que es preferible continuar explicando otras cosas y más adelante tendremos oportunidad de poner algunos ejemplos con expresiones aritméticas.

Hay una serie de operadores que trabajan a nivel de bits pero no vamos a explicarlos de momento.

Operadores '&&' y '||'

Son operadores AND y OR a nivel de shell y poseen circuito de evaluación corto. No tienen que ver con los operadores -a y -o ya que estos eran interpretados por el comando 'test'. Estos operadores '&&' y '||' serán colocados separando comandos que lógicamente retornan siempre algún valor.

'<comando1> && <comando2> && <comando3>' significa que debe cumplirse <comando1> y <comando2> y <comando3> para que se cumpla la expresión y '<comando1> || <comando2> || <comando3>' significa que debe cumplirse <comando1> o <comando2> o <comando3> para que se cumpla la expresión.

Si el resultado de '<comando1> && <comando2> && <comando3>' fuera 0 (TRUE) significaría que los tres comandos han retornado 0 (TRUE). Si por el contrario el resultado hubiera sido distinto (FALSE) solo sabríamos que por lo menos uno de los comandos retornó FALSE.

Supongamos que el primer comando retornó TRUE. La shell deberá continuar ejecutando el segundo comando. Supongamos que el segundo comando retorna FALSE. En este momento la shell no continua ejecutando el tercer comando porque da igual lo que retorne el resultado será FALSE. La razón es que se necesitaban todos TRUE para un resultado TRUE. El hecho de no ejecutar las partes siguientes una vez se sabe el resultado es lo que se llama circuito de evaluación corto y su uso principal es el de ahorrar ejecuciones de ordenes innecesarias aunque se puede usar tambien para ejecutar cosas solo bajo ciertas condiciones.

Con el operador '||' (OR) pasa algo similar. Si el resultado de '<comando1> || <comando2> || <comando3>' fuera 1 (FASE) significaría que los tres comandos han retornado 1 (FALSE). Si por el contrario el resultado hubiera sido distinto 0 (TRUE) solo sabríamos que por lo menos uno de los comandos retornó TRUE. Supongamos que el primer comando retornó FALSE. La shell deberá continuar ejecutando el segundo comando. Supongamos que el segundo comando retorna TRUE. En este momento la shell no continua ejecutando el tercer comando porque da igual lo que retorne el resultado será TRUE. Se necesitaban todos FALSE para un resultado FALSE.

Un comando normal como por ejemplo 'grep' o 'echo' que termina bien devuelve TRUE. Si hubiéramos redirigido la salida de 'grep'o de 'echo' a un fichero sin permiso de escritura o a un fichero con un path inexistente el retorno de 'grep' o de 'echo' habría sido FALSE indicando la existencia de error. En otras ocasiones FALSE no indicará error sino que el comando no tuvo éxito. Esto último sería aplicable a 'grep' (patrón localizado o no localizado) pero no sería aplicable a un comando como 'echo'.

Veamos un ejemplo con 'grep'.

```
$ echo hola | grep hola ; echo $?

hola
0
```

```
$ echo hola | grep hola > /hh/hh/hh ; echo $?

bash: /hh/hh/hh: No existe el fichero o el directorio
1
$ echo xxx | grep hola ; echo $?

1
```

Cuando ejecutamos comandos con un pipe ('|') el código de retorno es el del último comando ejecutado. En nuestro caso el primer código 1 se debe a la imposibilidad de generar la salida. El segundo código 1 se debe a que la cadena no fue encontrada. En los tres casos $? recoge el código retornado por 'grep'.

Ahora vamos a probar ejemplos que ilustren el circuito de evaluación corto. Primero para OR

```
$ # Como un OR queda evaluado cuando se encuentra el
primer resultado TRUE
$ # solo se evaluaran los comandos hasta obtener el
primero que devuelva
$ # 0 (TRUE)
$ echo "hola" || echo "adiós"

hola
$ # El resultado sera 0
$ echo $?

0
```

Ahora para AND.

```
$ # Como un AND no queda hasta que se evalúa el primer
resultado FALSE
$ # se evaluaran los dos comandos y devolverá
igualmente TRUE.
$ echo "hola" && echo "adiós"

hola
```

```
adiós
$ # El resultado sera 0
$ echo $?

0
```

El circuito de evaluación corto ha de usarse correctamente. Si sabemos que un comando dará casi siempre un resultado que puede hacer innecesario la ejecución de otros comandos lo colocaremos en primer lugar. Los comandos lentos se colocan en último lugar porque de esa forma puede no ser necesario ejecutarlos. Algunas veces no nos interesa un resultado sino solo que se ejecute una orden dependiendo del resultado de otra orden. Es decir queremos ejecutar solo si se cumple una condición.

```
$ test ! -w fichero && echo "Cuidado fichero sin
permiso de escritura"
$ test  -w fichero || echo "Cuidado fichero sin
permiso de escritura"
```

PROGRAMACION SHELL-SCRIPT (Tercera Parte)

Sentencia condicional 'if'

Ya hemos visto una forma de hacerlo usando los operadore '||' o '&&' de la shell, pero existen formas que pueden resultar más versatiles y más legibles.

```
if  _condición_
   then _lista_de_ordenes_
   [ elif _condición_
       then _lista_de_ordenes_ ] ...
   [ else _condición_ ]
fi
```

En lugar de *_condición_* podríamos haber puesto *_lista_de_ordenes_* pero queremos resaltar que el código de retorno va a ser evaluado.

Las partes entre corchetes son opcionales y si van seguidas de tres puntos '...' indica que puede presentarse varias veces. Todas las sentencias condicionales 'if' empiezan con la palabra reservada 'if' y terminan con la palabra reservada 'fi'.

Mejor vamos a ilustrar con algunos ejemplos. Deberá editarlos y ejecutarlos.

```
# Esto siempre mostrara '123'
if true
  then echo '123'
fi
```

Acabamos de utilizar una condición que siempre se cumplirá. Vamos a ver algo un poco más util.

```
# Si la variable CMPRT01 está definida y contiene el
nombre de un fichero
# con permiso de lectura se mandará a la impresora con
'lp'
if test -r  $CMPRT01
   then lp $CMPRT01
fi
```

También podemos programar una acción para cuando se cumpla una condición y otra para cuando no se cumpla.

```
# Si La variable 'modo' contiene el valor 'lp'
imprimir el fichero $FICH
# en caso contrario sacarlo por pantalla.
if [ "$modo" = "lp" ]
   then lp  $FICH
   else cat $FICH
fi
```

El siguiente ejemplo editelo con el nombre 'tipofichero'. Nos servirá para ilustrar el uso de 'elif' y para repasar algunas de las opciones de 'test'.

```
# tipofichero
FILE=$1
if test -b  $FILE
   then echo $FILE 'Es un dispositivo de bloques'
   elif test -c $FILE
     then echo $FILE 'Es un dispositivo especial de
caracteres'
   elif test -d $FILE
     then echo $FILE 'Es un directorio'
   elif test -f $FILE
     then echo $FILE 'Es un fichero regular
(ordinario)'
   elif test -L $FILE
     then echo $FILE 'Es un Link simbólico'
   elif test -p $FILE
     then echo $FILE 'Es un pipe con nombre'
   elif test -S $FILE
     then echo $FILE 'Es un Socket (dispositivo de
comunicaciones)'
   elif test -e $FILE
     then echo $FILE 'Existe pero no sabemos que tipo
de fichero es'
   else echo $FILE 'No existe o no es accesible'
fi
```

Para usar este último ejemplo ejectute './tipofichero ..' o './tipofichero tipofichero'

Tenga en cuenta que si bien la 'bash' admite el uso de 'elif' quizas otras shell no lo admitan.

Setencia condicional 'case'

Ya hemos visto que con 'elif' podemos evaluar distintas condiciones programando una acción para cada caso.

Existe un caso especial. Imaginemos que deseamos evaluar una variable y en función de lo que contenga hacer una cosa u otra. Podríamos usar una sentencia condicional con abundantes 'elif' pero hay que tener en cuenta una cosa. Con 'elif' tenemos que ir evaluando a cada paso hasta que se

cumpla una vez y entonces se ejecuta la acción y ya no se ejecuta ninguna otra porque habremos finalizado. En el caso anterior si hubieramos introducido el nombre de un pipe con nombre se habrían ejecutado varios test antes de determinar que era un pipe con nombre, y eso resulta lento.

Si lo que deseamos hacer está en función de un valor utilizaremos la sentencia condicional 'case' que es mucho más eficiente porque no requiere ejecutar varias ordenes o una orden varias veces como en el caso anterior.

La sintaxis para la sentencia condicional 'case' sería de la siguiente forma:

```
case valor in
  [ _lista_de_patrones_ ) _lista_de_ordenes_ ;; ] ...
esac
```

Por valor queremos indicar el resultado de cualquier expansión producida por la shell. La _lista_de_patrones_ son uno o más patrones separados por el caracter '|' y cada patrón es del mismo tipo que los patrones utilizados para la expansión de nombres de ficheros. El orden es importante porque se van comprobando por orden y en el momento que uno de los patrones corresponda con el valor se ejecutará la *_lista_de_ordenes_* que le corresponda y ya no se ejecutará nada más. Es frecuente utilizar en último lugar un patrón '*)' para que en caso de no encontrar ninguna correspondencia con los patrones anteriores se ejecute siempre una acción por defecto. Edite el siguiente ejemplo y salvelo con nombre 'prueba_case'.

```
# prueba_case
case $1 in
1) echo Uno ;;
2) echo Dos ;;
[3-7]) echo "Entre tres y siete ambos incluidos" ;;
8|9|0) echo Ocho; echo Nueve; echo Cero ;;
[a-zA-Z]) echo Una Letra ;;
start|begin) echo Comienzo ;;
stop|end) echo Fin ;;
*) echo 'Fallo'
```

```
esac
```

Ahora le damos permiso de ejecución y probamos como funciona.

```
$ ./prueba_case 1

Uno
$ ./prueba_case 2

Dos
$ ./prueba_case 7

Entre tres y siete ambos incluidos
$ ./prueba_case h

Una Letra
$ ./prueba_case start

Comienzo
$ ./prueba_case begin

Comienzo
$ ./prueba_case aa

Fallo
$ ./prueba_case 12

Fallo
$ ./prueba_case 9

Ocho
Nueve
Cero
$ ./prueba_case stop

Fin
```

Recuerde que la ejecución de una orden externa en un shell-srcipt es una operación costosa. Hacer un script ineficiente puede no tener ninguna importancia dependiendo del uso que le demos y de lo sobrada de recursos que esté la máquina, pero en otros caso si será más importante

así que haciendo las cosas lo mejor posible estaremos preparados en un futuro para trabajos más exigentes.

Entrada de datos con read

La instrucción 'read' permite capturar datos desde la entrada estandar. Para ello queda a la espera de un fin de linea. Edite el fichero siguiente y llameló 'pru_read'.

```
echo -e "Introduzca su nombre : \c"
read NOMBRE
banner Hola $NOMBRE
```

Ejecute './pru_read'. Observe que los datos introducidos son guardados en la variable NOMBRE. Si pulsa directamente la variable tomará valor "".

Cuando se usa la instrucción 'read' sin una variable el contenido se guardará en la variable REPLY pero si lo que desea es guardar el contenido queda más claro guardarlo en una variable concreta. Más adelante veremos ejemplos que usan esta instrucción.

Bucles 'while' y 'until'

```
while _condición_
do
  _lista_de_ordenes_
done
```

En lugar de *_condición_* podríamos haber puesto *_lista_de_ordenes_* pero queremos resaltar que el código de retorno va a ser evaluado.

Mientras se cumpla la condición se ejecutará *_lista_de_ordenes_*. Resulta evidente que si dentro de la *_lista_de_ordenes_* no hay nada capaz de alterar la condición, resultará un blucle que se ejecutará de forma ininterrumpida. (Bucle infinito). Si esto ocurre habrá que matar el proceso enviando alguna señal.

Existe un bucle muy similar y que solo se diferencia en que no para nunca hasta que se cumpla la condición. Es decir justo al revés que antes.

```
until _condición_
do
   _lista_de_ordenes_
done
```

Edite como 'pru_while1' el siguiente ejemplo.

```
# pru_while1
# Hacer un bucle de captura por consola (terminara
cuando se pulse
# solo <INTRO> ) y mostrar el resultado de cada
captura entre parentesis.
# Inicializamos primero la variable LINEA con un valor
cualquiera
# pero distinto de ""
LINEA="x"
while test $LINEA
do
   read LINEA
   echo '('$LINEA')'
done
```

Cuando pulsemos <INTRO> directamente sin nada más LINEA valdrá "" y 'test $LINEA' devolverá FALSE y el bucle finalizará.

Vamos a ejecutar el ejemplo para comprobar su funcionamiento.

```
$ ./pru_while1
aaaaa

(aaaaa)

bbbbb

(bbbbb)


()
```

Vemos que en la última ejecución LINEA valía "".

Bucle 'for'

Se proporciona una lista de elementos y se ejecuta una lista de órdenes haciendo que una variable vaya tomando el valor de cada uno de estos elementos. Entre estas ordenes se puede utilizar un nuevo bucle 'for'.

El siguiente ejemplo funciona pero es una mala idea ya que supone Varias ejecuciones de un comando cuando se podía haber hecho con uno solo. Habría sido mejor hacer 'chmod +x *'.

```
for i in $*
do
   chmod +x  $i
done
```

El siguiente ejemplo no funcionará si no prepara un poco una serie de cosas que se necesitan. Concretamente deberá de existir un fichero 'lista' con varios elementos. Varios ficheros con nombre que empiecen por SCAN. Varios ficheros '*.doc' de los cuales algunos deberan contener la palabra 'CODIGO'. Estudie el contenido del ejemplo que sigue. Editelo y ejecutelo pasando varios argumentos. Su objetivo es hacerlo funcionar y entender como funciona.

```
# Para ver distintos modos de manejar listas vamos a
generar todos los
# nombres posibles formados por combinacion de las
partes siguientes:
# Como parte 1 los nombres de ficheros de directorio
actual que
#    empiezan por 'SCAN'.
# Como parte 2 los nombres contenidos en el fichero
'lista'.
# Como parte 3 los identificadores 'cxx0 cxx2 bxx5'
# Como parte 4 la lista de parametros $1 $2 $3 .. etc,
# Como parte 5 los ficheros '*.doc' que contienen la
palabra 'CODIGO'.
```

```
###
for i in `ls SCAN*`
do
  for j in `cat lista`
  do
    for k in cxx0 cxx2 bxx5
    do
      for l in $*
      do
        for m in `grep -l "CODIGO" *.doc`
        do
          echo $i.$j.$k.$l.$m
       done
      done
    done
  done
done
```

Cuando tengamos bucles unos dentro de otros decimos que son bucles anidados. El nivel de anidamiento de un bucle es el número de bucles que hay unos dentro de otros.

'break' y 'continue'

Existe una forma de controlar un blucle desde el interior del mismo. Para eso podemos usar 'break', o 'continue'. Se pueden usar en cualquiera de los bucles que acabamos de ver (while, until, y for).

La palabra reservada 'break' provoca la salida de un bucle por el final. Si viene seguido de un numero 'n' saldrá de 'n' niveles. No poner número equivale a poner el número 1.

La palabra reservada 'continue' provoca un salto al comienzo del bucle para continuar con la siguiente iteración. Si viene seguida de un numero 'n' saldrá de 'n' niveles.

```
# pru_break_continue
# Definir la variable j como una variable de tipo
entero e
```

```
# inicializarla a cero. Luego la incrementamos a cada
iteración
# del bucle y si j es menor que diez mostramos el
doble de su valor.
# y en caso contrario salimos del bucle
typeset -i j=0
while true
do
     let j=j+1
     if [ $j -et 3 ]
        then continue
     fi
     if [ $j -et 4 ]
        then continue
     fi
     if [ $j -lt 10 ]
       then expr $j \* 2
       else break
     fi
done
```

Probamos ahora a ejecutarlo y obtendremos

```
$ ./pru_break_continue

2
4
10
12
14
16
18
```

Ahora edite y pruebe el siguiente ejemplo que llamaremos 'pru_break_continue2'

```
# ./pru_break_continue2
for i in uno dos tres
  do
  for j in a b c
    do
```

```
    for k in 1 2 3 4 5
      do
        echo $i $j $k
        if [ "$j" = "b" ]
           then break
        fi
        if [ "$k" = "2" ]
           then continue 2
        fi
      done
    done
done
```

El resultado de la ejecución sería como sigue.

```
$ ./pru_break_continue2

uno a 1
uno a 2
uno b 1
uno c 1
uno c 2
dos a 1
dos a 2
dos b 1
dos c 1
dos c 2
tres a 1
tres a 2
tres b 1
tres c 1
tres c 2
```

Arreglos

Vamos a estudiar ahora un tipo de variables que se caracterizan porque permiten guardar valores asociados a una posición. Los llamamos 'arreglos' (en ingles array) también se pueden llamar tablas. En el siguiente ejemplo usaremos una varirable 'tab[]' que se comporta como si estuviera formada por varias variables. tab[1], tab[2], tab[3], etc... Vamos

a ilustrarlo con un ejemplo que tiene un poco de todo. Se trata de un programa que debe estudiar detenidamente.

```
# Desarrollar una función que admita un parametro de
entrada. Si el
# parametro de entrada contiene una cadena que ya esta
almacenada en
# la tabla 'tabnom' retornar sin mas, pero si no esta
añadir esa
# palabra a la tabla.
GuardaNombre(){
  # Si numero de parametros distindo de 1 salir con
error.
  if [ $# -ne 1 ]
     then
        echo "Numero de parametros invalido en
GuardaNombre()"
        return 2
  fi
  typeset -i j=1
  for var in ${tab[*]}
  do
     if [ $var = $1 ]
        then
           ind=$j
           return 1
     fi
     let j=j+1
  done
  ind=$j
  tab[$ind]=$1
  return 0
}
########################### main ##################
while true
do
  echo -e "Introduzca algo o puse <INTRO> directamente
para finalizar : \c"
  read DATO
  if [ ! "$DATO" ]
       then break
  fi
  GuardaNombre $DATO
done
echo "Ahora visualizamos los datos introducidos"
```

```
for l in ${tab[*]}
do
    echo $l
done
echo 2 ${tab[2]}
echo 1 ${tab[1]}
```

El resultado de ejecutar esto introduciendo una serie de datos sería como sigue:

```
Introduzca algo o puse <INTRO> directamente para
finalizar : hhhhh
Introduzca algo o puse <INTRO> directamente para
finalizar : jjjj
Introduzca algo o puse <INTRO> directamente para
finalizar : jjjj
Introduzca algo o puse <INTRO> directamente para
finalizar : jjjj
Introduzca algo o puse <INTRO> directamente para
finalizar : ooooooooo
Introduzca algo o puse <INTRO> directamente para
finalizar : kk
Introduzca algo o puse <INTRO> directamente para
finalizar :
Ahora visualizamos los datos introducidos
hhhhh
jjjj
ooooooooo
kk
2 jjjj
1 hhhhh
```

Con esto comprobamos que podemos acceder no solo a la lista completa de los datos introducidos sino a uno solo de ellos proporcionando un número con la posición donde el dato ha sido almacenado.

También comprobamos que el valor 'jjjj' se ha indroducido varias veces pero nuestro programa solo lo guarda una vez gracias a que antes de guardar cada valor se comprueba si dicho valor ya fúe guardado antes.

Un ejempo sencillo: Construcción de un menú

```
#########################################################
###################
muestraopcionesmenuprin() {
        clear
        echo '1) Fecha y hora'
        echo '2) Calendario del més actual'
        echo '3) Calendario del año actual'
        echo '4) Calculadora de precisión arbitraria'
        echo '5) Lista de usuarios conectados'
        echo '6) Memoria libre del sistema'
        echo '7) Carga del sistema'
        echo '8) Ocupacion de todo el sistema de
ficheros'
        echo '0) Terminar'
        echo
        echo -e "Introduzca la opción deseada : \c"
}

#########################################################
###################
pausa () {
        echo
        echo -e "Pulse  para continuar"
        read
}

#########################################################
###################
################################ MAIN
#################################
#########################################################
###################
while true
do
        muestraopcionesmenuprin
        read OPT
        clear
        case $OPT in
        3|7) echo "Para salir deberá pulsar 'q'" ;
pausa ;;
        4) echo "Para salir deberá introducir
'quit'" ; pausa ;;
```

```
        esac
        echo ; echo
        case $OPT in
        0) exit ;;
        1) date ; pausa ;;
        2) cal ; pausa ;;
        3) cal `date +%Y` | less ;;
        4) bc ;;
        5) who -iTH ; pausa ;;
        6) cat /proc/meminfo ; pausa ;; # Podría
usarse el comando free
        7) top -s ;;
        8) df ; pausa ;;
        *) echo -e "Opcion erronea.\a" ; pausa ;;
        esac
done
echo
echo
```

Vamos a comentar algunos aspectos de este programa. Comienza con 'while true' (Antes de esto lo que hay es la declaración de un par de funciones). Un programa así se ejecutaría en un bucle sin fin pero existe una instrucción 'exit' en medio del programa para finalizar cuando se elija la opción adecuada.

Primero se llama a una función 'muestraopciones' que nos muestra las opciones disponibles y nos invita a introducir una de ellas que gracias a la instrucción read será almacenada en la variable OPT. El contenido de esta variable se usa en un par de sentencias 'case'. La primera solo considera si es alguna de las opciones '3,7, o 4' y en función de ello muestra un mensaje adicional. El segundo 'case' sirve para desencadenar la funcionalidad elegida. Existe un '*)' que sirve para advertir que la opción introducida no es correcta. Quizás ya ha notado que en ocasiones usamos la opción -e con echo. Esto se hace cuando usamos un caracter especial que debe ser interpretado como en el caso de '\a' (emite un pitido), o como '\c' (evita el salto de carro). El comando 'clear' se usa para limpiar la pantalla.

El coste de los comandos externos

Estamos acostumbrados a usar comandos externos como forama de apoyo a la programación en shell-script. Esto es muy normal y el SO tiene una gran cantidad de comandos que estan diseñados para poderlos combinar unos con otros mediante entrada salida.

A pesar de esto el uso de comandos externos tiene un coste muy elevado en comparación con el de los comandos internos, implementados en la propia shell. Por esto conviene siempre que sea posible usar estos últimos. En caso contrario tampoco se notará mucha diferencia cuando los usemos en medio de un proceso interactivo. En estos la lentitud de respuesta del usuario no permite apreciar la ventaja de una programación eficiente. Donde si se apreciará esta diferencia es cuando usemos comandos dentro de bucles. Para ilustrar lo que decimos usaremos un programita de ejemplo que simplemente utilice un contador de bucle incrementado una variable.

```
# Bucle con let (comando interno)
typeset -i N=0
time while [ $N -le 20000 ]
do let N+=1
done

real    0m1.413s
user    0m1.070s
sys     0m0.030s


# Bucle con expr (comando externo)
typeset -i N=0
time while [ $N -le 100 ]
do N=`expr $N + 1`
done

real    0m1.311s
user    0m0.230s
sys     0m0.060s
```

Ambas pruebas tardan aproximadamente lo mismo pero el primer bucle ejecuta 20000 iteraciones y el segundo solo 100. Es decir que el uso del comando externo hace el comando unas 200 veces más lento. Quizás piense que es culpa de expr y en parte es cierto pero lo realmente importante es que hay que arrancar una subshell la cual arrancará expr para luego tomar la salida estandar y expandirlo todo correctamente. Esto comporta abrir los respectivos ficheros de esos ejecutables y abrir un fichero es algo que requiere un trabajo que no tiene lugar cuando ejecutamos comandos internos de la shell.

Recomendaciones finales

Acabamos de explicar un programa muy sencillo. Compliquelo un poco añadiendo una opción 9 que muestre un submenú que entre otras cosas contenga una opción que vuelva al menú principal. Use su imaginación y practique para intentar mejorarlo.

En estos momentos ya dispone de toda la base de conocimientos que le permitirá personalizar su entorno de trabajo o construir sencillos scripts que le ayuden a realizar una gran variedad de tareas, o también puede investigar algunos de los programas del sistema escritos en shell-script. Si se conforma con lo estudiado y no intenta ponerlo en práctica pronto lo olvidará todo. Practique la programación en shell-script ya mismo antes de que olvide la teoría. Empiece por las cosas que hemos explicado pero no se quede en eso. Intente llegar un poco más lejos.

En las lecciones que siguen continuaremos tratando temas que abrirán nuevas puertas a su conocimiento. En ningún caso podemos detenernos excesivamente en ninguna de ellas. A la programación en shell-script la hemos dedicado varias lecciones y anteriormente dedicamos algunos más a la shell como interprete de comandos, pero ha llegado el momento de decir que debe de poner mucho de su parte y que existe el 'man bash' para buscar las cosas que no pudimos explicar aquí.

NOCIONES DE PROGRAMACIÓN EN AWK

Que es awk y para que se usa

La palabra 'awk' se usa tanto para referirse a un lenguaje de manipulación de ficheros de datos como para referirse a su interprete.

Dado que los SO tipo Unix incluido Linux acostumbran con mucha frecuencia a usar ficheros de configuración del sistema en formatos de de texto perfectamente legibles y editables se diseño un lenguaje para poder procesar este tipo de ficheros de datos en formato de texto.

Cuando un programa necesita una velocidad muy alta para acceder a unos datos o para modificarlos se utilizan estructuras de datos más sofisticadas.

En muchos otros casos un fichero de configuración será accedido de forma muy ocasional, y resulta más interesante usar un formato de texto sencillo. Por ejemplo hay ficheros de configuración que solo se usan durante la carga de un programa. Algunos de estos programas suelen cargarse una sola vez mientras arranca el sistema y luego en condiciones normales permanecen arrancados todo el tiempo.

'awk' nació en 1978 como un lenguaje pequeño y sencillo pero desde entonces ha evolucionado mucho y en la actualidad se puede afirmar que es un lenguaje muy potente y versátil. Imposible tratarlo en profundidad en un curso como este.

'awk' es un complemento muy bueno para su uso con shell-script. Esta lección no va a condicionar la asimilación de lecciones posteriores pero recomendamos que como mínimo le de un breve repaso ya que 'awk' puede resultar extremadamente útil en determinadas circunstancias.

Nos vamos a conformar con explicar unas pocas cosas porque con ello conseguiremos dos objetivos. El primero que pueda usarlo para un limitado tipo de tareas bastante frecuentes, y el segundo que conozca su existencia y para que se usa. De esta forma puede ampliar conocimientos por su cuenta cuando lo necesite.

Nos vamos a centrar en el procesamiento de datos en los cuales cada línea estará estructurada en campos. Estos campos estarán delimitados entre si por algún carácter o por alguna secuencia especial de caracteres especialmente reservado para ello. Esta secuencia será el delimitador de campos y no debe aparecer en el interior de ningún campo. Cada línea equivale a un registro.

La mayoría de las bases de datos, y hojas de cálculo permiten volcar los datos en formato de texto para poder ser exportados entre distintas bases de datos. Estas salidas se pueden procesar fácilmente mediante 'awk'. También se puede usar 'awk' con la salida de diversos programas. Esto permite entre otras cosas usar 'awk' para acoplar una salida de un programa con la entrada de otro que necesite un formato muy distinto. En definitiva vamos a explicar solo una pequeña parte de este potente lenguaje pero comprobará su gran utilidad muy pronto.

Forma de uso

'awk' suele estar instalado en la mayoría de los sistemas ya que su uso suele ser necesario. Por eso en Linux suele encontrarse entre los paquetes básicos del sistema en todas las distribuciones.

Se puede usar de varias formas. Tenemos que pasar a 'awk' el texto del programa, y los datos. El primero se puede pasar bien como argumento o indicando -f nombre del fichero que contiene el texto del programa. La entrada se puede pasar dando el nombre del fichero de entrada como último argumento o en caso contrario lo tomará por la entrada estándar.

```
$ ## Generamos en /tmp un par de ficheros
$ echo -e "\n"  > /tmp/echo.out
$ echo '{  print "Hola mundo" }' > /tmp/ejemplo1.awk
$ ## Ejecutaremos el mismo programa de 4 formas
distintas
$ echo -e "\n" | awk '{  print "Hola mundo" }'
 Hola mundo
 Hola mundo
$ awk '{  print "Hola mundo" }' /tmp/echo.out
```

```
 Hola mundo
 Hola mundo
$ echo -e "\n" | awk -f /tmp/ejemplo1.awk
 Hola mundo
 Hola mundo
$ awk -f /tmp/ejemplo1.awk /tmp/echo.out
 Hola mundo
 Hola mundo
```

El programa que acabamos de utilizar imprimirá el literal "Hola mundo" a cada línea de datos que procese. En este caso usamos solo un par de líneas vacías como entrada de datos.

Vamos a localizar el binario de 'awk'

```
$ whereis awk
 /usr/bin/awk
```

Vamos a suponer que en su sistema se encuentre también en '/usr/bin'. Puesto que awk es un lenguaje interpretado perfectamente legible también podemos decir que los programas de awk son script. Para poder usarlos directamente podemos añadir una primera línea con número mágico y poner permiso de ejecución.

```
$ echo '#!/usr/bin/awk -f' > /tmp/ejemplo2.awk
$ echo '{  print "Hola mundo" }' >> /tmp/ejemplo2.awk
$ chmod +x /tmp/ejemplo2.awk
$ echo -e "\n" | /tmp/ejemplo2.awk
 Hola mundo
 Hola mundo
```

Estructura de un programa awk

Un programa 'awk' puede tener tres secciones distintas.

- Puede incluir una primera parte para que se ejecute antes de procesar ninguna de las líneas de entrada. Se usa para ello la

palabra reservada **BEGIN** seguida de una o mas instrucciones todas ellas englobadas dentro de un par de corchetes. '{' , '}'.

- Puede incluir una parte central que se procesará entera para cada linea de entrada de datos y que puede tener varios bloques '{' , '}'. Si uno de estos bloques contiene una expresión regular se procesará solo cuando la línea de entrada se ajuste al patrón de la expresión regular.
- Puede incluir una parte final que se procesará en último lugar una vez termine la lectura y procesado de todas las líneas de entrada. Se usa para ello la palabra reservada **END** seguida de una o más instrucciones todas ellas englobadas dentro de un par de corchetes. '{' , '}'.

El primer ejemplo que vimos anteriormente ("Hola mundo") solo tenía una de las tres partes. Concretamente era la parte central ya que no pusimos ninguna de las palabras reservadas **BEGIN** o **END**.

Vamos a poner ahora un ejemplo con las tres partes. Edite un fichero con nombre '/tmp/3partes.awk'

```
BEGIN { print "Erase una vez..." }
{ print "...y entonces bla, bla, bla ..." }
END { print "...y colorín colorado este cuento se ha
acabado." }
```

Ejecútelo con:

```
$ echo -e "\n\n\n" | awk -f /tmp/3partes.awk
çAma Erase una vez...
...y entonces bla, bla, bla ...
...y entonces bla, bla, bla ...
...y entonces bla, bla, bla ...
...y entonces bla, bla, bla ...
...y colorín colorado este cuento se ha acabado.
```

Es importante que comprenda que la parte central se ejecutará tantas veces como líneas de datos existan. En nuestro ejemplo son cuatro líneas generadas por 'echo -e "\n\n\n" '. En cambio las partes 'BEGIN { ... }' y 'END { ... }' se ejecutan una sola vez. La primera antes de procesar la primera línea y la última después de procesar la última línea.

Los comentarios en 'awk' comienzan con un '#' y terminan al final de la línea.

Expresiones regulares

Algunas veces los datos pueden venir con algunas lineas que no interesa procesar o que se deben procesar de forma distinta. Podemos usar una expresión regular delimitada por el carácter '/' para seleccionar una acción especial. Vamos a editar otro ejemplo que llamaremos '/tmp/expreg.awk':

```
BEGIN { print "Erase una vez..." }
/^$/ { print "Linea vacía" }
/[0-9]+/ { print "Tiene un número" }
/\.$/ { print "Termina con punto" }
# Esto es un comentario
{ print "--------------------------------------" }
END { print "...y colorín colorado este cuento se ha
acabado." }
```

Ahora editamos un segundo fichero '/tmp/expreg.dat':

```
Línea número 1.
Línea número 2

....
Fin de los datos
```

Ahora ejecute lo siguiente:

```
$ awk -f /tmp/expreg.awk /tmp/expreg.dat
```

```
Erase una vez...
Tiene un número
Termina con punto
--------------------------------------
Tiene un número
--------------------------------------
Linea vacía
--------------------------------------
Termina con punto
--------------------------------------
--------------------------------------
...y colorín colorado este cuento se ha acabado.
```

Vemos que cada línea de datos puede cumplir más de una regla y que cuando no ponemos una expresión regular siempre se ejecutará la acción. En este caso todas las líneas provocan la escritura de una línea de guiones '--------------------------------------'.

El uso de expresiones regulares puede ayudarnos a eliminar cabeceras, líneas vacías o incompletas o cosas así que no deseamos procesar.

Delimitadores de campos

No hemos tratado aun los campos de una línea. Una línea que tenga distintos campos debe usar alguna secuencia para delimitar los campos entre si.

Lo mismo para definir un delimitador que en cualquier otro caso donde se usen cadenas de caracteres podemos encontrarnos la necesidad de usar caracteres especiales que no pueden ser introducidos directamente. Para ello existen determinadas secuencias que empiezan por el carácter '\' y que tienen significado especial.

Caracteres de escape	
\a	Produce un pitido en el terminal
\b	Retroceso

\f	Salto de página
\n	Salto de línea
\r	Retorno de carro
\t	Tabulador horizontal
\v	Tabulador vertical
\ddd	Carácter representado en octal por 'ddd'
\xhex	Carácter representado en hexadecimal por 'hex'
\c	Carácter 'c'

El último caso se usa para eliminar el significado especial de un carácter en determinadas circunstancias. Por ejemplo para usar un '+' o un '-' en una expresión regular usaríamos '\+' o '\-'

Podemos elegir un solo carácter para separar campos. Hay ficheros de configuración como /etc/passwd, /etc/group, que usan un solo carácter para delimitar los campos. Por ejemplo los dos puntos ':' , el blanco '\ ', la coma ',' el tabulador '\t' etc...

'awk' permite usar como delimitador más de un carácter. Para ello se asignará a la variable 'FS' una cadena de caracteres que contenga una expresión regular . Por ejemplo para usar como delimitador el carácter ':' habría que incluir 'BEGIN { FS = ":" }'

Si no se especifica ningún delimitador se asumirá que los campos estarán delimitados por uno o más blancos o tabuladores consecutivos lo cual se expresa como "[\ \t]+". Las expresiones regulares ya fueron estudiadas en un capítulo especial. El carácter '\' debe usarse para escapar cualquier carácter con significado especial en una expresión regular y algunos caracteres normales precedidos de '\' se usan para representar caracteres especiales. '\t' es el tabulador.

En 'awk' se usa $1 para referenciar el campo 1, $2 para referenciar el campo 2, etc... y para referenciar el registro completo usaremos $0.

Edite el siguiente fichero '/tmp/delim1.awk'

```
{ print  "+", $1, "+", $2, "+", $3, "+", $4, "+" }
```

$1, $2, $3, y $4 representan a los campos 1, 2, 3, y 4 respectivamente. Edite el siguiente fichero de datos '/tmp/delim1.dat'

```
aaa bbb ccc           ddd            eee
111 222 333 444
```

En la primera línea debe introducir un blanco para separar los primeros blancos y una secuenciencia de ', , , ' para separar los dos últimos campos. Es importante que lo edite de esta forma porque el resultado de los ejemplos podría variar.

Ahora ejecute lo siguiente:

```
$ awk -f /tmp/delim1.awk /tmp/delim1.dat
+ aaa + bbb + ccc + ddd +
+ 111 + 222 + 333 + 444 +
```

Edite el siguiente fichero '/tmp/delim0.awk'

```
{ print  "+", $3, "+", $4, "+", $1, "+", $2, "+" }
```

Ahora ejecute lo siguiente:

```
$ awk -f /tmp/delim0.awk /tmp/delim1.dat
+ ccc + ddd + aaa + bbb +
+ 333 + 444 + 111 + 222 +
```

Con ello hemos conseguido variar el orden de aparición de los campos, pero todavía no hemos especificado ningún delimitador. Por ello hemos asumido el delimitador por defecto. (uno o más blancos y tabuladores).

Para especificar un delimitador distinto tenemos que asignar su valor a la variable FS y además tenemos que hacerlo antes de leer el primero registro por lo cual se incluirá la instrucción en la sección inicial precedida de BEGIN.

Edite el siguiente fichero '/tmp/delim2.awk'

```
BEGIN { FS = "\ " }
{ print  "+", $1, "+", $2, "+", $3, "+", $4, "+" }
```

Estamos definiendo un único carácter blanco como separador. Ahora ejecute lo siguiente:

```
$ awk -f /tmp/delim2.awk /tmp/delim1.dat
+ aaa + bbb + ccc +      +
+ 111 + 222 + 333 + 444 +
```

Vamos a cambiar de delimitador. Edite el siguiente fichero '/tmp/delim3.awk'

```
BEGIN { FS = "\t" }
{ print  "+", $1, "+", $2, "+", $3, "+", $4, "+" }
```

Estamos definiendo un único carácter tabulador como separador. Ahora ejecute lo siguiente:

```
$ awk -f /tmp/delim3.awk /tmp/delim1.dat
+ aaa bbb ccc  +   + ddd  +   +
+ 111 222 333 444 +  +  +  +
```

Selección de registros por campo

Vamos a editar un fichero que simulará la salida de datos obtenida desde una base de datos relacional. Usaremos estos datos en varios ejemplos.

Puede corresponder a una contabilidad de un alquiler de un piso. Lo llamaremos 'contabil.dat'.

```
     fecha|concepto|importe
----------+--------+-------
01-01-1999|-       |     96
16-12-1999|AGUA    | -14650
05-01-2000|LUZ     | -15797
24-01-2000|GAS     | -34175
27-01-2000|INGRESO | 141200
01-02-2000|MENS    | -96092
25-02-2000|LUZ     | -12475
01-03-2000|MENS    | -96092
06-03-2000|INGRESO | 101300
01-04-2000|MENS    | -96092
06-04-2000|AGUA    | -15859
07-04-2000|INGRESO | 134000
01-05-2000|MENS    | -98975
02-05-2000|LUZ     | -11449
09-05-2000|INGRESO |  95000
23-05-2000|GAS     | -21428
25-05-2000|GAS     | -16452
01-06-2000|MENS    | -98975
07-06-2000|INGRESO | 130000
01-07-2000|MENS    | -98975
04-07-2000|LUZ     | -12403
07-07-2000|AGUA    |  -5561
10-07-2000|INGRESO |  99000
24-07-2000|GAS     | -11948
01-08-2000|MENS    | -98975
10-08-2000|INGRESO | 122355
04-09-2000|LUZ     | -12168
10-09-2000|INGRESO | 129000
19-09-2000|AGUA    | -10529
28-09-2000|GAS     |  -2620
01-10-2000|MENS    | -98975
10-10-2000|INGRESO | 112000
(32 rows)
```

Lo primero que vemos es que tiene una cabecera de dos líneas inútiles y un final también inútil. Podemos asegurar que las líneas que deseamos procesar cumplirán un patrón de dos números guión dos números guión

cuatro números y línea vertical. Vamos a editar un programa que llamaremos 'contabil1.awk'

```
BEGIN { FS="\|" }
/[0-9][0-9]\-[0-9][0-9]\-[0-9][0-9][0-9][0-9]\|/  {
print NR, ", ", $1, ", ", $2, ", ", $3
}
```

Vamos a ejecutar este ejemplo y vamos a ver su salida

```
$ awk -f contabil1.awk < contabil.dat
3 ,  01-01-1999 ,  -         ,       96
4 ,  16-12-1999 ,  AGUA      ,   -14650
5 ,  05-01-2000 ,  LUZ       ,   -15797
6 ,  24-01-2000 ,  GAS       ,   -34175
7 ,  27-01-2000 ,  INGRESO   ,   141200
8 ,  01-02-2000 ,  MENS      ,   -96092
9 ,  25-02-2000 ,  LUZ       ,   -12475
10 ,  01-03-2000 ,  MENS      ,   -96092
11 ,  06-03-2000 ,  INGRESO   ,   101300
12 ,  01-04-2000 ,  MENS      ,   -96092
13 ,  06-04-2000 ,  AGUA      ,   -15859
14 ,  07-04-2000 ,  INGRESO   ,   134000
15 ,  01-05-2000 ,  MENS      ,   -98975
16 ,  02-05-2000 ,  LUZ       ,   -11449
17 ,  09-05-2000 ,  INGRESO   ,    95000
18 ,  23-05-2000 ,  GAS       ,   -21428
19 ,  25-05-2000 ,  GAS       ,   -16452
20 ,  01-06-2000 ,  MENS      ,   -98975
21 ,  07-06-2000 ,  INGRESO   ,   130000
22 ,  01-07-2000 ,  MENS      ,   -98975
23 ,  04-07-2000 ,  LUZ       ,   -12403
24 ,  07-07-2000 ,  AGUA      ,    -5561
25 ,  10-07-2000 ,  INGRESO   ,    99000
26 ,  24-07-2000 ,  GAS       ,   -11948
27 ,  01-08-2000 ,  MENS      ,   -98975
28 ,  10-08-2000 ,  INGRESO   ,   122355
29 ,  04-09-2000 ,  LUZ       ,   -12168
30 ,  10-09-2000 ,  INGRESO   ,   129000
31 ,  19-09-2000 ,  AGUA      ,   -10529
32 ,  28-09-2000 ,  GAS       ,    -2620
33 ,  01-10-2000 ,  MENS      ,   -98975
```

```
34 ,  10-10-2000 ,  INGRESO  ,   112000
```

Podemos apreciar varias cosas. NR es una variable del sistema que toma el valor del número de registro que se está procesando. Podemos ver que las dos primeras líneas y la última han sido descartadas. También vemos que las primeras líneas usan un solo dígito para el número de registro y luego usan dos dígitos. Esto que las columnas no queden alineadas.

Vamos a modificar el programa para que muestre los registros completos ($0) cuando no se cumpla la condición anterior. Para ello editaremos un fichero que llamaremos 'contabdescarte.awk'.

```
BEGIN { FS="\|" }
! /[0-9][0-9]\-[0-9][0-9]\-[0-9][0-9][0-9][0-9]\|/  {
print NR, $0
}
```

Vamos a ejecutar este ejemplo y vamos a ver su salida

```
$ awk -f contabdescarte.awk < contabil.dat

1       fecha|concepto|importe
2 ----------+--------+-------
35 (32 rows)
```

Formato de salida con printf

Para imprimir con formato usaremos 'printf' en lugar de 'print'. printf se usa en varios lenguajes. El primer argumento de esta función debe de ser una cadena de caracteres que contenga el formato de salida deseado para la salida. Los formatos de cada dato se expresan mediante unas directivas que empiezan con el carácter '%' y debe de existir en dicha cadena tantas directivas como datos separados por coma a continuación de la cadena de formato.

Hay que tener en cuenta que en 'awk' la concatenación de cadenas se usa poniendo una cadena a continuación de otra separada por blancos. Por ejemplo:

cad = "(unodos)" cad = "uno" "dos" ; cad = "(" cad ")"

Especificación de formato de datos para 'printf'	
%c	Carácter ASCII
%d	Entero representado en decimal
%e	Coma flotante (exponente = e[+-]dd)
%E	Coma flotante (exponente = E[+-]dd)
%f	Coma flotante sin exponente
%g	Equivale al más corto de los formatos 'e' o 'f'
%G	Equivale al más corto de los formatos 'E' o 'F'
%o	Entero representado en octal
%s	Cadena de caracteres
%x	Entero representado en hexadecimal con minúsculas
%X	Entero representado en hexadecimal con mayúsculas
%%	Carácter '%'

En estos formatos se puede intercalar inmediatamente a continuación del '%' primero un signo '+' o un '-' que son opcionales y significan respectivamente alineación a la derecha o a la izquierda. En segundo lugar y de forma igualmente opcional se puede intercalar un número para indicar un ancho mínimo. (Si el dato ocupa más que el dato especificado se muestra el dato completo haciendo caso omiso de la indicación de anchura). En el caso de coma flotante se puede indicar el ancho total del campo y la precisión (anchura) de la parte decimal.

Veamos unos pocos ejemplos.

```
$ echo  | awk '{  print "Hola mundo" }'

Hola mundo
$ echo  | awk '{  printf "Hola %s\n", "mundo" }'

Hola mundo
$ echo  | awk '{  printf "#%d#%s#\n", 77, "mundo" }'

#77#mundo#
$ echo  | awk '{  printf "#%10d#%10s#\n", 77,
"mundo" }'

#        77#     mundo#
$ echo  | awk '{  printf "#%-10d#%-10s#\n", 77,
"mundo" }'

#77        #mundo     #
$ echo  | awk '{  printf "#%+4d#%+4s#\n", 77,
"mundo" }'

# +77#mundo#
$ echo  | awk '{  printf "#%04d#%+4s#\n", 77,
"mundo" }'

#0077#mundo#
$ echo  | awk '{  printf "#%010.5f#%E#%g\n",
21.43527923, 21.43527923, 21.43527923  }'

#0021.43528#2.143528E+01#21.4353
$ echo  | awk '{  printf "#%10.5f#%E#%g\n", 2140000,
2140000, 2140000  }'

#2140000.00000#2.140000E+06#2.14e+06
```

Practique un poco investigando con más detalle el funcionamiento de estos formatos.

Uso de variables operadores y expresiones

En 'awk' podemos usar toda clase de expresiones presentes en cualquier lenguaje. Cualquier identificador que no corresponda con una palabra

reservada se asumirá que es una variable. Para asignar un valor se usa el operador '='

Vamos a editar un fichero que llamaremos 'ejemplexpr.awk' con algunas expresiones aritméticas.

```
{
contador = 0;  # Pone a cero la variable contador
contador ++;   # Incrementa en 1 la variable contador
contador +=10; # Incrementa en 10 la variable contador.
contador *=2   # Multiplica por 2 la variable contador
print contador
contador = ( 10 + 20 ) / 2 ;
print contador
contador = sqrt ( 25 ) ; # Raiz cuadrada de 25
print contador
}
```

Lo ejecutamos y observamos el resultado.

```
$ echo | awk -f ejemplexpr.awk

22
15
5
```

No podemos explicar en detalle todo el lenguaje 'awk'. Se trata de que comprenda su utilidad y de que sea capaz de utilizarlo para determinadas tareas en las cuales resulta extremadamente útil.

Algunas expresiones parecen inspiradas en el lenguaje C. Otras parece que han servido de inspiración para el lenguaje Perl. En realidad muchos lenguajes usan expresiones parecidas.

Por ello vamos a resumir en forma de tabla una serie de elementos que intervienen en las expresiones que 'awk' es capaz de manejar. Pero no vamos a explicar en detalle cada cosa. En lugar de eso daremos una

descripción resumida y procuraremos que los ejemplos posteriores tengan un poco de todo.

Operadores aritméticos	
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo (resto)
^	Potenciación
Operadores de asignación.	
var = expr	Asignación
var ++	Incrementa la variable en una unidad
var --	Decrementa la variable en una unidad
var += expr_aritm	Incrementa la variable en cierta cantidad
var -= expr_aritm	Decrementa la variable en cierta cantidad
var *= expr_aritm	Multiplica la variable por cierta cantidad
var /= expr_aritm	Divide la variable por cierta cantidad
var %= expr_aritm	Guarda en la variable el resto de su división por cierta cantidad
var ^= expr_aritm	Eleva el valor de la variable en cierta cantidad
Operadores lógicos y de relación.	
expr_aritm == expr_aritm	Comparación de igualdad
expr_aritm !=	Comparación de desigualdad

expr_aritm	
expr_aritm < expr_aritm	Comparación menor que
expr_aritm > expr_aritm	Comparación mayor que
expr_aritm <= expr_aritm	Comparación menor igual que
expr_aritm >= expr_aritm	Comparación mayor igual que
expr_cad ~ expr_regular	Se ajusta al patrón
expr_cad !~ expr_regular	No se ajusta al patrón
expr_logica || expr_logica	Operador lógico AND (Y)
expr_logica && expr_logica	Operador lógico OR (O)
! expr_logica	Operador lógico NOT (NO)
Funciones aritméticas.	
atan2(y, x)	Retorna el arco-tangente de y/x en radianes
cos(x)	Retorna el coseno de x en radianes
exp(x)	Retorna el exponencial de x (e^x)
int(x)	Retorna el valor entero de x truncado la parte decimal
log(x)	Retorna el logaritmo neperiano de x
rand()	Retorna un valor seudo aleatorio comprendido entre 0 y 1
sin(x)	Retorna el seno de x en radianes
sqrt(x)	Retorna la raiz cuadrada de x
srand(x)	Inicializa la semilla para generar números pseudoaleatorios

Funciones para usar con cadenas de caracteres	
gsub(r, s, t)	Sustituye 's' globalmente en todo 't' cada vez que se encuentre un patrón ajustado a la expresión regular 'r'. Si no se proporciona 't' se toma $0 por defecto. Devuelve el número de sustituciones realizado.
index(cadena, subcadena)	Retorna la posición de la 'subcadena' en 'cadena' (Primera posición = 1)
length(cadena)	Devuelve la longitud de la 'cadena'. Tomará $0 por defecto si no se proporciona 'cadena'
split(cadena, array, sep)	Parte 'cadena' en elementos de 'array' utilizando 'sep' como separador. Si no se proporciona 'sep' se usará FS. Devuelve el número de elementos del array
sub(r, s, t)	Sustituye 's' en 't' la primera vez que se encuentre un patrón ajustado a la expresión regular 'r'. Si no se proporciona 't' se toma $0 por defecto. Devuelve 1 si tiene éxito y 0 si falla
substr(cadena, beg, len)	Devuelve una subcadena de 'cadena' que empieza en 'beg' con una longitud 'len'. Si no se proporciona longitud devuelve hasta el final de la cadena desde 'beg'
tolower(cadena)	Pasa a minúsculas
toupper(cadena)	Pasa a mayúsculas
Algunas otras funciones	
match(cadena, expr_reg)	Indica si 'cadena' se ajusta o no a la expresión regular 'expr_reg'
system(comando)	
sprintf(formato [, expr-list])	Para obtener salida con formato.

Computo con registros

Vamos a modificar el programa 'contabil1.awk' para procesar solo los registros de consumo de luz, vamos a mejorar el formato de salida, vamos a incluir un contador de registros seleccionados, un contador de consumo de luz, y al final obtendremos el consumo total y el consumo promedio de luz. Lo llamaremos 'contabil2.awk'

```
BEGIN {
   FS="\|" ;
   cont_reg_luz=0;
   cont_importe_luz=0;
   }

$2 ~ /LUZ/  {
   cont_reg_luz = cont_reg_luz + 1 ;
   cont_importe_luz = cont_importe_luz + $3 ;
   printf ("%3d, %3d, %s, %s, %s, %10d\n", NR,
cont_reg_luz, $1, $2, $3, cont_importe_luz);
}
END {
   printf ("Consumo promedio = %d\n", cont_importe_luz
/ cont_reg_luz) ;
}
```

Vamos a ejecutar este ejemplo y vamos a ver su salida

```
$ awk -f contabil2.awk < contabil.dat
  5,   1, 05-01-2000, LUZ     ,  -15797,     -15797
  9,   2, 25-02-2000, LUZ     ,  -12475,     -28272
 16,   3, 02-05-2000, LUZ     ,  -11449,     -39721
 23,   4, 04-07-2000, LUZ     ,  -12403,     -52124
 29,   5, 04-09-2000, LUZ     ,  -12168,     -64292
Consumo promedio = -12858
```

Los datos que estamos usando para el ejemplo están ordenados por fechas. Vamos a obtener un informe con un campo más que será el saldo de la cuenta. Para ello editamos un fichero que llamaremos 'contabil3.awk'.

```
BEGIN {
   FS="\|" ;
   cont_importe=0;
   }

/[0-9][0-9]\-[0-9][0-9]\-[0-9][0-9][0-9][0-9]\|/  {
   cont_importe = cont_importe + $3 ;
   printf ("%3d, %s, %s, %s, %10d\n", NR,  $1, $2, $3,
cont_importe);
}
```

Vamos a ejecutar este ejemplo y vamos a ver su salida

```
$ awk -f contabil3.awk < contabil.dat
  3, 01-01-1999, -       ,       96,          96
  4, 16-12-1999, AGUA    ,   -14650,      -14554
  5, 05-01-2000, LUZ     ,   -15797,      -30351
  6, 24-01-2000, GAS     ,   -34175,      -64526
  7, 27-01-2000, INGRESO ,   141200,       76674
  8, 01-02-2000, MENS    ,   -96092,      -19418
  9, 25-02-2000, LUZ     ,   -12475,      -31893
 10, 01-03-2000, MENS    ,   -96092,     -127985
 11, 06-03-2000, INGRESO ,   101300,      -26685
 12, 01-04-2000, MENS    ,   -96092,     -122777
 13, 06-04-2000, AGUA    ,   -15859,     -138636
 14, 07-04-2000, INGRESO ,   134000,       -4636
 15, 01-05-2000, MENS    ,   -98975,     -103611
 16, 02-05-2000, LUZ     ,   -11449,     -115060
 17, 09-05-2000, INGRESO ,    95000,      -20060
 18, 23-05-2000, GAS     ,   -21428,      -41488
 19, 25-05-2000, GAS     ,   -16452,      -57940
 20, 01-06-2000, MENS    ,   -98975,     -156915
 21, 07-06-2000, INGRESO ,   130000,      -26915
 22, 01-07-2000, MENS    ,   -98975,     -125890
 23, 04-07-2000, LUZ     ,   -12403,     -138293
 24, 07-07-2000, AGUA    ,    -5561,     -143854
 25, 10-07-2000, INGRESO ,    99000,      -44854
 26, 24-07-2000, GAS     ,   -11948,      -56802
 27, 01-08-2000, MENS    ,   -98975,     -155777
 28, 10-08-2000, INGRESO ,   122355,      -33422
 29, 04-09-2000, LUZ     ,   -12168,      -45590
 30, 10-09-2000, INGRESO ,   129000,       83410
```

```
31, 19-09-2000, AGUA    ,  -10529,      72881
32, 28-09-2000, GAS     ,   -2620,      70261
33, 01-10-2000, MENS    ,  -98975,     -28714
34, 10-10-2000, INGRESO ,  112000,      83286
```

Sentencias condicionales y bucles

'awk' es un lenguaje muy completo y no podía faltar las sentencias de ejecución condicional y de ejecución en bucle.

Algunos de los conceptos que vamos a comentar ya los hemos visto cuando hablamos de la programación en bash y no vamos a explicar con demasiado detalle cada tipo de sentencia. La sintaxis que usa awk no se parece a la sintaxis que ya hemos visto para bash. Se parece más a la sintaxis del lenguaje C. De todas formas los conceptos ya nos resultan familiares y usaremos algunos ejemplos para ilustrarlos.

Empezaremos describiendo la sintaxis de cada tipo de sentencia. Denominaremos acción a una sentencia simple o a una sentencia compuesta de la forma '{ sentencia1 ; sentencia2 ; ... }'

```
Sentencia condicional 'if'

if ( expresión_lógica )
        accion1
[ else
        accion2 ]

Sentencia condicional con los operadores '?' y ':'

expresion_lógica ? accion1 : accion2

Bucle 'while'

while ( expresión_lógica )

        accion
```

Bucle 'do' 'while'

```
do

        accion

while ( expresión_lógica )
```

Bucle 'for'

```
for ( inicializar_contador ; comprobar_contador ;
modificar_contador )

        accion
```

Dentro de los bucles podemos usar **break** para forzar la salida de un bucle o **continue** para saltar a la siguiente iteración.

Veremos de momento tan solo un ejemplo para la sentencia condicional 'if'.

Edite el siguiente fichero que llamaremos 'contabil4.awk'

```
BEGIN { FS="\|" ; }

/[0-9][0-9]\-[0-9][0-9]\-[0-9][0-9][0-9][0-9]\|/  {
   if ( $3 >= 0)  {
        printf ("%3d, %s, %s, %s\n", NR,  $1, $2, $3);
   }
}
```

Vamos a ejecutar este ejemplo y vamos a ver su salida

```
$ awk -f contabil4.awk < contabil.dat
  3, 01-01-1999, -        ,       96
  7, 27-01-2000, INGRESO ,   141200
 11, 06-03-2000, INGRESO ,   101300
 14, 07-04-2000, INGRESO ,   134000
 17, 09-05-2000, INGRESO ,    95000
```

```
 21, 07-06-2000, INGRESO ,  130000
 25, 10-07-2000, INGRESO ,   99000
 28, 10-08-2000, INGRESO ,  122355
 30, 10-09-2000, INGRESO ,  129000
 34, 10-10-2000, INGRESO ,  112000
```

Pasar valores al script awk

En ocasiones puede resultar interesante poder pasar algún valor al script awk. Vamos a modificar el programa anterior para que muestre los registros con un importe superior a un valor que pasaremos por parámetro.

Edite el siguiente fichero que llamaremos 'contabil5.awk'

```
BEGIN { FS="\|" ; }

/[0-9][0-9]\-[0-9][0-9]\-[0-9][0-9][0-9][0-9]\|/  {
   if ( $3 >= minimo && $3 <= maximo )  {
       printf ("%3d, %s, %s, %s\n", NR,  $1, $2, $3);
   }
}
```

Vamos a ejecutar este ejemplo pasando y vamos a ver su salida

```
$ awk -f contabil5.awk minimo=100000 maximo=120000 <
contabil.dat
 11, 06-03-2000, INGRESO ,  101300
 34, 10-10-2000, INGRESO ,  112000
```

Hay que advertir que el paso de parámetros equivale a definir una variable y a asignar un valor pero esto valor no será accesible hasta después de leer el primer registro. Si el valor pasado como parámetro tuviera que ser accesible en la sección BEGIN habría que usar la opción -v previo al paso del parámetro.

Repetiremos el ejemplo pasando el delimitador del registro que usaremos en la sección BEGIN.

Edite el siguiente fichero que llamaremos 'contabil6.awk'

```
BEGIN { FS = delimitador ; }

/[0-9][0-9]\-[0-9][0-9]\-[0-9][0-9][0-9][0-9]\|/  {
   if ( $3 >= minimo && $3 <= maximo )  {
        printf ("%3d, %s, %s, %s\n", NR,  $1, $2, $3);
   }
}
```

Vamos a ejecutar este ejemplo pasando valores y vamos a ver su salida

```
$ awk -f contabil6.awk  minimo=100000 maximo=120000
deliminador='|' < contabil.dat
```

Vemos que no hemos obtenido el resultado esperado.

Vamos a volver a ejecutar este ejemplo pasando el valor del delimitador con la opción -v y vamos a ver su nueva salida

```
$ awk -v delimitador='|' -f contabil6.awk minimo=100000
maximo=120000 < contabil.dat
 11, 06-03-2000, INGRESO ,  101300
 34, 10-10-2000, INGRESO ,  112000
```

Declaración de funciones

Como es lógico 'awk' permite la declaración de funciones. Normalmente se recurre a implementar una función cuando necesitamos una funcionalidad que el lenguaje no proporciona de forma predefinida o cuando queremos estructurar el código de un programa grande en fragmentos más pequeños y por tanto más manejables.

La sintaxis es muy sencilla.

```
function nombre_de_la_función ( lista_de_parámetros ) {
     sentencias
}
```

Por ejemplo para declarar una función que retorne un número aleatorio entre 1 y 6.

Edite un fichero con nombre 'dado.awk'.

```
function aleatorio ( minimo, maximo ){
    return ( ( ( maximo - minimo + 1 ) * rand () ) +
minimo ) ;
}

END{
        for (i=0; i<20; i++){
                printf ("%3d) Entre 1 y 6 = %3d
Entre 5 y 15 =%3d\n",
                        i, aleatorio (1, 6),
aleatorio(5, 15));
        }
}
```

Ahora lo ejecutamos

```
$ echo | awk -f dado.awk
  0) Entre 1 y 6 =   5    Entre 5 y 15 = 12
  1) Entre 1 y 6 =   6    Entre 5 y 15 =  7
  2) Entre 1 y 6 =   6    Entre 5 y 15 =  7
  3) Entre 1 y 6 =   5    Entre 5 y 15 = 13
  4) Entre 1 y 6 =   3    Entre 5 y 15 =  8
  5) Entre 1 y 6 =   3    Entre 5 y 15 =  6
  6) Entre 1 y 6 =   4    Entre 5 y 15 =  7
  7) Entre 1 y 6 =   6    Entre 5 y 15 =  7
  8) Entre 1 y 6 =   5    Entre 5 y 15 =  6
  9) Entre 1 y 6 =   3    Entre 5 y 15 = 10
 10) Entre 1 y 6 =   6    Entre 5 y 15 = 14
 11) Entre 1 y 6 =   5    Entre 5 y 15 = 10
 12) Entre 1 y 6 =   6    Entre 5 y 15 = 11
```

```
 13) Entre 1 y 6 =   4    Entre 5 y 15 = 12
 14) Entre 1 y 6 =   5    Entre 5 y 15 = 15
 15) Entre 1 y 6 =   2    Entre 5 y 15 =  9
 16) Entre 1 y 6 =   1    Entre 5 y 15 =  9
 17) Entre 1 y 6 =   3    Entre 5 y 15 = 14
 18) Entre 1 y 6 =   2    Entre 5 y 15 = 14
 19) Entre 1 y 6 =   2    Entre 5 y 15 =  7
```

Función system

Esta es una función fácil de usar que nos permite ejecutar un comando del sistema operativo. En caso de éxito retorna 0, y en caso de error retornará un valor distinto de cero.

```
$ awk ' BEGIN { if (system("ls") !=0) printf ("Error
de ejecución\n"); }'
```

Por ejemplo si quisiéramos verificar la existencia de un fichero almacenado en la variable 'nombre_fich' tendríamos que hacer

```
if (system("test -r " nombre_fich)) {
        fprintf ("%s no encontrado\n", nombre_fich);
}
```

La función getline y otras funciones avanzadas

Este es es un apartado en el que más que explicar cosas nos vamos a limitar a mencionar ciertas posibilidades. No podemos dedicar demasiado espacio a este tipo de cuestiones avanzadas pero si con lo que en este apartado contemos conseguimos ponerle los dientes largos nos daremos por satisfechos aunque no entienda una palabra.

En primer lugar hay que advertir que 'getline' que al igual que otras funciones devuelve un valor pero su sintaxis no es una típica sintaxis de función. No se usa como 'getline()' sino como una sentencia.

Esta función retorna 1 si lee una línea, 0 si alcanza el fin de la entrada de datos y -1 si se produce un error.

Usada simplemente como 'getline' sin nada más lee la siguiente linea de la entrada asignando $0 y desglosando los campos en $1, $2, $3, etc..

Se puede asignar el valor completo de la línea leída a una variable con 'getline variable' evitando de esta forma alterar el valor de $0.

Se puede leer de un fichero usando el operador redirección. 'getline < "fichero"'. Se puede simbolizar la entrada estándar como "-"

Se puede leer desde un pipe. '"whoami" | getline usuario'.

Edite un fichero llamado 'tipo_usuario.awk'.

```
BEGIN {
"whoami" | getline usuario
 if ( usuario ~ /root/ ) {
     printf ("Soy superusuario\n");
 }
 else{
    printf ("Soy usuario normal\n");
 }
}
```

Ejecute lo con

```
$ awk -f tipo_usuario.awk
```

No pretendemos con este sencillo ejemplo que sea capaz de usar estas funciones. El manejo de estas redirecciones es complicado y en ocasiones se hace necesario forzar el cierre de una entrada o de un pipe. Para eso existe la función 'close'. Se usa haciendo 'close ("fichero")' o 'close ("whoami")'.

Por el momento nos conformamos con lo explicado y veremos un poco más adelante el uso de getline tomando la entrada de un fichero cuando expliquemos los arrays asociativos.

Arrays

Los array permiten el almacenamiento de una serie de elementos que pueden ser accedidos mediante un índice. En realidad los arrays de awk son más potentes que los arrays que vimos cuando estudiamos la programación de la bourne-shell donde los índices de un array eran siempre números enteros. Vamos a usar en primer lugar los arrays de esta forma. Es decir nos vamos a limitar en los primeros ejemplos a usar números enteros como índices.

Vamos a usar awk para procesar la salida obtenida con 'ps'. Primero vamos a suponer que obtenemos un listado completo de los procesos del sistema en formato largo. Si intenta realizar este ejemplo obtendrá un resultado necesariamente diferente.

```
$ ps axl > ps-axl.out ; cat ps-axl.out

 FLAGS   UID   PID  PPID PRI  NI   SIZE   RSS WCHAN
STA TTY TIME COMMAND
   100     0     1     0   0   0    756     0 do_select   SW
?   0:03 (init)
    40     0     2     1   0   0      0     0 bdflush     SW
?   0:18 (kflushd)
    40     0     3     1   0   0      0     0 kupdate     SW
?   0:18 (kupdate)
   840     0     4     1   0   0      0     0 kpiod       SW
?   0:00 (kpiod)
   840     0     5     1   0   0      0     0 kswapd      SW
?   0:15 (kswapd)
   140     0   186     1   0   0    900   200 do_select   S
?   0:00 /sbin/sys
   140     0   188     1   0   0   1016     0 do_syslog   SW
?   0:00 (klogd)
   140     1   193     1   0   0    780     0 do_select   SW
?   0:00 (portmap)
   140     0   195     1   0   0    860     0 do_select   SW
?   0:00 (inetd)
```

```
   140     0   200     1   0   0    764   108 nanosleep   S
?   0:00 /usr/sbin
   140     0   210     1   0   0    908     0 do_select   SW
?   0:00 (lpd)
    40    31   226     1   0   0   3784     0 do_select   SW
?   0:00 (postmast
   140     0   237     1   5   0   1728   316 do_select   S
?   0:00 sendmail:
   140     0   241     1   0   0   1292   184 do_select   S
?   0:01 /usr/sbin
    40     0   244     1   0   0   1544    56 do_select   S
?   0:00 /usr/bin/
    40     1   254     1   0   0    840    96 nanosleep   S
?   0:00 /usr/sbin
    40     0   257     1   5   0    860   164 nanosleep   S
?   0:00 /usr/sbin
   140     0   262     1   0   0   1780    60 do_select   S
?   0:07 /usr/sbin
   100     0   268     1   0   0   1964   616 read_chan   S
1  0:00 -bash
   100     0   269     1   0   0    836     0 read_chan   SW
2  0:00 (getty)
   100  1001   270     1   0   0   2096   724 wait4       S
3  0:00 -bash
   100     0   271     1   0   0    836     0 read_chan   SW
4  0:00 (getty)
   100     0   272     1   0   0    836     0 read_chan   SW
5  0:00 (getty)
   100  1001   273     1   0   0   2088  1408 wait4       S
6  0:00 -bash
   140    33   274   262   0   0   1792     0 wait_for_co SW
?   0:00 (apache)
   140    33   275   262   0   0   1792     0 flock_lock_ SW
?   0:00 (apache)
   140    33   276   262   0   0   1792     0 flock_lock_ SW
?   0:00 (apache)
   140    33   277   262   0   0   1792     0 flock_lock_ SW
?   0:00 (apache)
   140    33   278   262   0   0   1792     0 flock_lock_ SW
?   0:00 (apache)
     0  1001   916   270   0   0   3536  1640 do_select   S
3  0:00 vi awk1.d
     0  1001  1029   273   0   0   1916   668 wait4       S
6  0:00 xinit /ho
   100     0  1034  1029  12   0   8824  3280 do_select   S
?   0:02 X :0 -bpp
     0  1001  1037  1029   0   0   4620  2748 do_select   S
6  0:01 mwm
    40  1001  1042  1037   0   0   1728   924 wait4       S
6  0:00 bash /hom
    40  1001  1045  1037   0   0   1728   924 wait4       S
```

```
6  0:00 bash /hom
     0     0  1050  1042   0   0   2976  1872 do_select   S
6  0:00 xterm -ls
     0  1001  1058  1045   0   0   2320  1220 do_select   S
6  0:00 xclock -d
   100  1001  1051  1050  14   0   2080  1400 wait4       S
p0  0:00 -bash
100000  1001  1074  1051  17   0   1068   528             R
p0  0:00 ps axl
```

Para dar una idea de la situación de parentescos entre los distintos procesos mostramos la salida obtenida con el comando 'pstree' ejecutado desde la misma sesión de xterm que en el caso anterior.

```
$ pstree -p > pstree-p.out ; cat pstree-p.out
init(1)-+-apache(262)-+-apache(274)
        |             |-apache(275)
        |             |-apache(276)
        |             |-apache(277)
        |             `-apache(278)
        |-atd(254)
        |-bash(268)
        |-bash(270)---vi(916)
        |-bash(273)---xinit(1029)-+-XF86_S3V(1034)
        |                         `-mwm(1037)-
+-.xinitrc(1042)---xterm(1050)---bash(1051)---
pstree(1068)
        |
`-.xinitrc(1045)---xclock(1058)
        |-cron(257)
        |-getty(269)
        |-getty(271)
        |-getty(272)
        |-gpm(200)
        |-inetd(195)
        |-kflushd(2)
        |-klogd(188)
        |-kpiod(4)
        |-kswapd(5)
        |-kupdate(3)
        |-lpd(210)
        |-portmap(193)
        |-postmaster(226)
        |-sendmail(237)
```

```
        |-sshd(241)
        |-syslogd(186)
        `-xfs(244)
```

Solo vamos a procesar los campos PID y PPID. $3 y $4 respectivamente. Los meteremos en un par de arrays llamados pid y ppid.

```
BEGIN { ind=0;  }

function padre(p){
   for (i=0; i <ind; i++){
       if (pid[i]== p)
          return ppid[i];
   }
}

! /FLAGS/ { pid[ind]=$3 ; ppid[ind]=$4 ; ind++ ; }

END {
   do {
      printf ("%d->", proc);
      proc= padre(proc);
  } while ( proc >= 1 )
  printf ("\n\n");
}
```

Ahora ejecutamos pasando el pid del proceso del cual deseamos averiguar su descendencia.

```
$ awk -f ancestros.awk proc=1051 < ps-axl.out

1051->1050->1042->1037->1029->273->1->
```

Con un número realmente reducido de líneas de código acabamos de procesar la salida de un comando que no estaba especialmente diseñado para ser procesado sino para entregar un resultado legible.

No se emocione todavía porque solo hemos utilizado los arrays con indices numéricos. Lo cierto es que los arrays de 'awk' a diferencia de los

arrays de otros lenguajes son arrays asociativos. Eso significa que podemos usar como índice una cadena de caracteres. Por ejemplo podemos hacer lo siguiente: nombre_cli["5143287H"]="Luis, García Toledano"

No es necesario dar un tamaño al array. Un array asociativo no establece un orden entre sus elementos. Hay que aclarar que el manejo de un array con índices numéricos corresponde a un mecanismo muy simple ya que se usan porciones consecutivas de memoria del ordenador y se accede directamente por posición. Por el contrario un array asociativo de las características de 'awk' se va creando dinámicamente. Internamente 'awk' gestiona el acceso mediante una técnica de hash que usa tablas auxiliares a modo de tablas de índices y funciones auxiliares que obtiene valores numéricos a partir de valores de una cadena. Todo ello permite un acceso muy rápido en este tipo de estructuras haciéndolas adecuadas para su uso en bases de datos.

```
FTLSUSE |CURSOS  |FTLinuxCourse para SuSE
| 11800
FTLREDH |CURSOS  |FTLinuxCourse para RedHat
| 11800
ASUSCOM |HARDWARE|Asuscom ISDNLink 128k Adapter (PCI)
|  6865
RAILROAD|JUEGOCOM|Railroad Tycoon (Gold Edition)
|  7700
CIVILIZ |JUEGOCOM|Civilization: Call to power
|  7700
MYTHII  |JUEGOCOM|Myth II
|  7700
LIAPPDEV|LIBROS  |Linux Application Development (537
Páginas)            | 11000
CONECT01|LIBROS  |Guía del Usuario de Linux  (413 Páginas)
|  5300
CONECT03|LIBROS  |Guía del Servidor (Conectiva Linux 437
Páginas)         |  5300
CONECT02|LIBROS  |Guía del Administrador de redes (465
Páginas)          |  5300
LIUSRESU|LIBROS  |Linux User's Resource (795 Páginas)
| 12000
RH70DLUX|LINUXCOM|RedHat Linux 7.0 Deluxe en español
|  9600
```

```
RH70PROF|LINUXCOM|RedHat Linux 7.0 Profesional en Español
| 20000
SUSE70  |LINUXCOM|Suse Linux 7.0 (6CDs)(Version española)
|  6850
RTIME22 |LINUXCOM|RealTime 2.2 (1CD)
| 13000
CONCT50E|LINUXCOM|Conectiva Linux 5.0 Versión Económica
Español (6CDs)   |  5200
CITIUS22|LINUXCOM|Linux Citius 2.2
|  7750
TRBLIW60|LINUXCOM|Turbolinux Workstation 6.0
|  6500
MOTIF   |LINUXCOM|Motif Complete
| 22000
CONCTSRV|LINUXCOM|Conectiva Linux  Ed.Servidor (Español
3CDs + 4 Manua   | 27500
RHORA8I |LINUXCOM|RedHat Linux Enterprise Edition optimized
for Oracle8i |270000
MANDRA72|LINUXCOM|Mandrake 7.2  (7CDs) PowerPack Deluxe
(versión española|  8300
PINGUINO|SUSEPROM|Pingüino de peluche
|  6000
```

```
BEGIN {
   FS="[\ \t]*\|[\ \t]*" ;
   while ( getline < "articulos.dat" > 0) {
     artic[$1]= "(" $4 " Ptas + Iva) " $3;
     printf ("%s ", $1);
   }
   for (;;){
      printf ("\n\nIntroduzca un código de artículo o
solo  para terminar: ");
      getline codigo ;
      if (codigo == "" )
         break;
      printf ("\n<%s>\n%s", codigo, artic[codigo]);
  }
}
```

```
$ awk -f articulos.awk

FTLSUSE FTLREDH ASUSCOM RAILROAD CIVILIZ MYTHII
LIAPPDEV CONECT01 CONECT03
CONECT02 LIUSRESU RH70DLUX RH70PROF SUSE70 RTIME22
```

```
CONCT50E CITIUS22 TRBLIW60
MOTIF CONCTSRV RHORA8I MANDRA72 PINGUINO

Introduzca un código de artículo o solo  para
terminar: RH70PROF


(20000 Ptas + Iva) RedHat Linux 7.0 Profesional en
Español

Introduzca un código de artículo o solo  para
terminar: CITIUS22


(7750 Ptas + Iva) Linux Citius 2.2

Introduzca un código de artículo o solo  para
terminar:
$
```

El programa que acabamos de realizar ilustra la potencia de 'awk' para el tratamiento de ficheros de datos. Si nuestro fichero de datos de ejemplo 'articulos.dat' tuviera un número de registros mucho mayor habríamos notado que inicialmente se tarda un cierto tiempo en leer todo el fichero de datos pero una vez almacenados los datos en el array su acceso a los mismos es rapidísimo. Esta rápidez se debe no solo a que los datos ya han sido leídos desde el disco duro y ya están en memoria sino porque toda la información está indexada de forma que la localizacion de cualquier elemento del array es muy rápida.

Faltan muchas cosas por contar sobre 'awk'. Si dispone de algún fichero de datos de interes personal saque una copia e intente realizar alguna utilidad en 'awk'.

EJERCICIOS RESUELTOS DE SHELL-SCRIPT

Bueno es hora de que pruebe sus conocimientos. No se preocupe si no consigue resolverlos todos. Para algo están las soluciones aunque eso si, tendrá que descargar el archivo ejerc1.tar.gz. Disculpe la molestia pero con esto evitamos que sus ojos se desvíen a la solución antes de tiempo.

---- ## donde_esta.txt ## ----

```
Hacer un shell-script que busque la presencia del comando
pasado como argumento en alguno de los directorios
referenciados
en la variable $PATH, señalando su localización y una breve
descripción del comando caso de existir su página man.
```

---- ## estadisticas_dir.txt ## ----

```
Programe un script llamado 'estadistica_dir.sh'
que realice un estudio de todo el arbol de directorios y
ficheros partiendo del directorio pasado como parámetro de
forma
que obtengamos la siguiente información:

Número de ficheros en los cuales disponemos de permiso de
lectura   =
Número de ficheros en los cuales disponemos de permiso de
escritura =
Número de ficheros en los cuales disponemos de permiso de
ejecución =
Número de ficheros en los cuales  carecemos de permiso de
lectura   =
Número de ficheros en los cuales  carecemos de permiso de
escritura =
Número de ficheros en los cuales  carecemos de permiso de
ejecución =

       Número de ficheros regulares encontrados =
              Número de directorios encontrados =
```

```
  Número de dispositivos de bloques encontrados =
Número de dispositivos de carateres encontrados =
                   Número de fifos encontrados =
```

---- ## gen_menu2.txt ## ----

```
Programar un shell-script que admita como argumentos
parejas formadas
por 'comando' y 'descripcion' y que construya con todo ello
un menu
de opciones donde cualquiera de los comandos pueda ser
ejecutado
seleccionando la descripcion correspondiente.

Es decir si dicha shell-script se llamara 'gen_menu2.sh' un
ejemplo
de uso sería:

./gen_menu2.sh \
        Listar ls \
        'Quien está conectado' who \
        'Fecha actual' date \
        Ocupacion du \
        'Quien soy' whoami \
        'Memoria libre' free \
        'Calendario' cal \
        'Nombre máquina' hostname

Con ello se obtendría un menu de la forma:

                0 Terminar
                1 Listar
                2 Quien está conectado
                3 Fecha actual
                4 Ocupacion
                5 Quien soy
                6 Memoria libre
                7 Calendario
                8 Nombre máquina

                Elija opción.
```

---- ## grafica.txt ## ----

Realizar un shell-script que reciba como argumentos numeros comprendidos
entre 1 y 75. Dara error en caso de que algun argumento no este
dentro del rango y terminará sin hacer nada. En caso contrario
generará una linea por cada argumento con tantos asteriscos como
indique su argumento.

---- ## incr.txt ## ----

Hacer un shell-scritp que utilice dos funciones 'incr1()' e 'incr2()'
las cuales ejecutaran un simple bucle iterando 1000 veces. La función
'incr1()' solo usara un contador IND1 que sera incrementado de uno en
uno usando un comando de tipo 'let' y la funcion 'incr2()' solo usara
un contador IND2 que se incrementará de uno en uno calculando su siguiente
valor mediante el uso del comando expr.

Hacer una tercera función 'PruebaDiezSegundos()' que aceptará como argumento
el nombre de la función de prueba (incr1 o incr2). Primero informara del
tiempo real, de cpu de usuario y de cpu del sistema que consume dicha
función y luego deberá ejecutar dicha función tantas veces sea posible
en el transcurso de 10 segundos (aproximadamente, basta usar date '+%s')
obteniendose el número de veces que se ejecuto en esos diez segundos.

Compare los resultados obtenidos con ambas funciones y explique los
resultados.

---- ## lista_fich.txt ## ----

Hacer un shell-script que acepte como argumentos nombres de ficheros
y muestre el contenido de cada uno de ellos precedido de una cabecera
con el nombre del fichero

---- ## media.txt ## ----

Hacer un shell-script que calcule la media aritmética de todos los argumentos pasados como parámetros con una precisión de 40 digitos decimales despues de la coma.

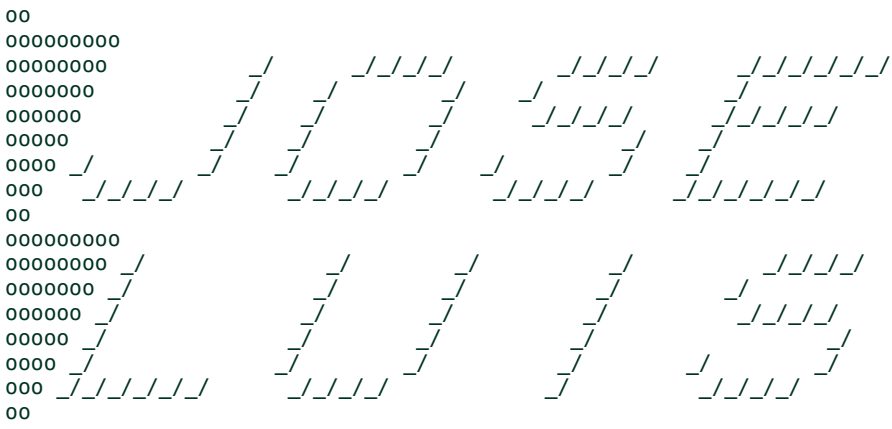
---- ## mi_banner.txt ## ----

Localice en su sistema la utilidad 'banner' para consola. Puede haber
mas de una utilidad para esto y algunas están pensadas para su uso
con impresora. Nos interesa la utilidad 'banner' para terminal de
forma que el comando 'banner hola' genere la siguiente salida:

```
 #    #   ####   #         ##
 #    #  #    #  #        #  #
 ######  #    #  #       #    #
 #    #  #    #  #       ######
 #    #  #    #  #       #    #
 #    #   ####   ######  #    #
```

Partiendo de esta utilidad realice un shell-script 'mi_banner.sh' que
admita hasta tres argumentos de tamaño máximo de cuatro caracteres
tal que el comando './mi_banner.sh hola jose luis' obtenga:

```
ooooooooo
oooooooo _/       _/      _/_/_/_/      _/              _/_/
ooooooo _/       _/    _/        _/    _/            _/    _/
oooooo _/_/_/_/_/_/    _/        _/    _/          _/        _/
ooooo _/       _/    _/        _/    _/          _/_/_/_/_/_/
oooo _/       _/    _/        _/    _/          _/        _/
ooo _/       _/     _/_/_/_/      _/_/_/_/_/_/   _/        _/
```



---- ## mi_cal1.txt ## ----

Realizar un shell-script 'mi_cal1.sh' que utitilzando la salida del
comando 'cal' filtre su salida para resaltar entre corchetes
el día actual. Ponga atención a los días a principio de línea,
los días a final de línea.

La salida para el día 5 de Mayo de 2002 debería ser la siguiente:

```
./mi_cal1.sh

      May 2002
  S    M   Tu    W   Th    F    S
                  1    2    3    4
 [5]   6    7    8    9   10   11
 12   13   14   15   16   17   18
 19   20   21   22   23   24   25
 26   27   28   29   30   31
```

No es un ejercicio trivial. Necesitará hacer un filtrado en varios pasos.
Utilice alguna combinación de filtros sed encadenados con pipe.

Para poder comprobar que el programa funciona correctamente para los
distintos días hacer que en caso de pasar un argumento este será
considerado el día del mes.

```
./mi_cal1.sh 1

      May 2002
  S    M   Tu    W   Th    F    S
               [1]   2    3    4
  5    6    7    8    9   10   11
 12   13   14   15   16   17   18
 19   20   21   22   23   24   25
 26   27   28   29   30   31
```

---- ## mi_cal2.txt ## ----

Basandose en el ejercicio anterior realizar una modificación
de 'mi_cal1.sh' que llamaremos 'mi_cal2.sh' que en lugar de encadenar varios comandos sed con pipes creará un fichero temporal
'/tmp/$0.$$.sed' para ser usar mediande la opción -f de sed.
Ambos scrits deberán producir idéntico resultado.

---- ## mi_logrotate.txt ## ----

Asumiremos que tenemos en un directorio una serie de ficheros de
log que van creciendo de forma ilimitada con el uso regular de
ciertos programas.

Realizar un shell-script que actue sobre los ficheros con nombre
tipo '*.log' del directorio actual de forma tal, que si alguno de
ellos supera en tamaño las 2000 lineas, dejará solo las últimas
1000 líneas del fichero y las restantes serán guardadas en un

```
directorio old_rot en formato comprimido.

En dicho directorio habrá que conservar en formato
comprimido
no solo la última porción eliminada del original, sino las
cuatro
últimas porciones eliminadas. Para ello será necesario ir
rotando
los nombres de estos ficheros comprimidos incorporando en
los mismos
un digito de secuencia.

(parte eliminada)  --> *.log.rot1.gz --> *.log.rot2.gz -->
*.log.rot3.gz --> *.log.rot4.gz --> eliminado

El programa durante su ejecución irá mostrando los ficheros
encontrados
y señalará aquellos que por su tamaño sea necesario rotar.
```

---- ## ocupa_tmp_log.txt ## ----

```
 Obtiene la ocupacion de los directorios presentes en el
sistema
 cuyo nombre sea de la forma '*tmp' o '*log', ordenados por
orden
 de ocupacion.

 Debera mostrar el numero de errores provocados por 'find'.
 Si se decide usar algzn fichero temporal debera usar el
 directorio '/tmp' para ello y usar un nombre que contenga
 el nombre del proceso que lo origino y su pid.
```

---- ## palabras.txt ## ----

```
Realice un shell-script que admita tres palabras como
argumentos
y que muestre un mensaje informando de las relaciones de
igualdad
y desigualdad entre esas palabras.

        "Las tres son iguales"
        "Son iguales primera y segunda"
        "Son iguales primera y tercera"
```

```
        "Son iguales segunda y tercera"
        "Son las tres distintas"
```

---- ## piramide.txt ## ----

Realice un shell-scrip 'piramide.sh' que admita como argumento un
número 'N' y que optenga a la salida una serie de 'N' filas de
forma triangular.

Para ./piramide.sh 12 la salida sería.

```
01
02 02
03 03 03
04 04 04 04
05 05 05 05 05
06 06 06 06 06 06
07 07 07 07 07 07 07
08 08 08 08 08 08 08 08
09 09 09 09 09 09 09 09 09
10 10 10 10 10 10 10 10 10 10
11 11 11 11 11 11 11 11 11 11 11
12 12 12 12 12 12 12 12 12 12 12 12
```

---- ## proceso.txt ## ----

Hacer un shell-script llamado 'proceso.sh' para ser lanzado en background
que como maximo permanecera vivo 30 segundos.

Podra ser lanzado varias veces en background y cada vez generara un
shell-script distinto 'stop_proceso.sh.' que al ser ejecutado matara el proceso lo origino y despues se borrara a si
mismo.

---- ## vigila.txt ## ----

Realizar un shell-script que escriba en el fichero '/tmp/usuarios'

```
una linea con la fecha y otra con el número de usuarios
distintos
que están conectados en ese momento cada dos minutos.
```

PARTE (II) USUARIO AVANZADO DE LINUX

INTRODUCCIÓN A LA SEGUNDA PARTE DEL CURSO

Llevamos ya algunos capítulos en los cuales estamos tocando muchos temas que podrían considerarse propios de un curso de administración o de un curso avanzado de Linux. Ya no hacemos tantas bromas y suponemos que ya ha perdido su inseguridad inicial.

Consideramos que ya a estas alturas del curso tampoco es necesario recordarle a cada momento las precauciones de trabajo. También asumimos que con la confianza que da el saber del porque de las cosas ya habrá practicado por su cuenta algunas otras cosas que específicamente le interesaban.

Para los que usen ordenador personal y que tengan acceso a una cuenta de root es el momento de dar algún consejo porque es probable que cada vez tengan mas ganas de probar cosas distintas para aprender por su cuenta. Para ellos les diremos que existe una máxima para todo administrador.

No modifiques nada que no puedas volver a dejar como estaba.

En otras palabras. Siempre, siempre, antes de hacer algo asegurate la posibilidad de poder retroceder fácilmente a la situación inicial.

Por ejemplo si vas a tocar un fichero de configuración del sistema saca antes una copia de ese fichero. O quizás la simple precaución de anotar en un papel un par de cosillas antes de cambiarlas pueda evitar un disgusto.

Muy importante. Haz regularmente copias de seguridad. No mantengas tus copias en el mismo disco que la información original. Como mínimo

usa tres copias de seguridad estableciendo un sistema de rotación entre ellas.

Usaremos el usuario root lo menos posible no para evitar errores sino para evitar que los errores se conviertan en Horrores.

Linux es un SO que no impone restricciones arbitrarias a los usuarios. Si una cosa no puede ser realizada por un usuario normal es porque no supone una amenaza a la seguridad del sistema y siempre que se pueda es mejor no usar root.

Admitimos que un usuario de Linux que lo instale en su propio sistema necesita realizar tareas de administración, pero root es un usuario que debe usarse lo menos posible. Errar es humano pero el desastre total es cosa de root. Por ello vamos a apurar las posibilidades de apreender desde una cuenta de usuario normalita

También hay que advertir que no todos los sistemas son igual de permisivos dependiendo de las necesidades reales de seguridad del sistema o del grado de paranoia del administrador. Por ejemplo una máquina que disponga de un elevado número de cuentas para estudiantes deberá ser administrada con mano firme porque siempre habrá más de un estudiante juguetón con ganas de reírse del administrador, y basta uno de estos para tener un problema serio si no se toman precauciones.

Si está realizando este curso en una máquina de esas características posiblemente no pueda hacer muchos de los ejercicios propuesto simplemente por limitaciones impuestas desde root. El aumento de la seguridad de un sistema suele resultar en un aumento de la incomodidad para los usuarios.

El precio de la seguridad es la incomodidad. El precio de la comodidad es la inseguridad.

En particular el acceso a distintos dispositivos hardware puede estar más o menos limitado. Nosotros para esta segunda parte vamos a suponer que

las restricciones del sistema donde usted va a practicar no son excesivas en este sentido. En cualquier caso ya habrá notado que en las prácticas mostramos y comentamos los resultados de las mismas. Por ello no se desanime si no puede realizar ahora algunas de las prácticas propuestas, pero también señalamos que Si esta usando un ordenador personal o dispone de una cuenta en un sistema donde la confianza en los usuarios por parte del administrador es elevada podrá seguramente sacarle mayor provecho a esta segunda parte del curso.

COPIAS DE SEGURIDAD

Porque dedicamos un capítulo completo a esto.

Este capítulo trata de las Copias de Seguridad, también llamados respaldos o en inglés Backups y vamos en primer lugar a justificar la razón de dedicar un amplio capítulo a este tema que podría parecer más apropiado para un curso de administración.

Evidentemente garantizar la integridad de la información de un sistema es un tema que es de responsabilidad del administrador del sistema pero, como ya hemos mencionado anteriormente, el objetivo de esta segunda parte del curso incluye las nociones más necesarias para administrar su propio ordenador personal con Linux. El objetivo es que sea autosuficiente.

La seguridad frente a ataques intencionados queda fuera de los propósitos de este curso pero no hay daño que no pueda ser reparado si tenemos una copia de seguridad de toda la información que nos interesa.

Un buen sistema de respaldo es la mejor forma de evitar perdidas de información. Estas deben ser prevenidas dentro de lo posible pero dependiendo de las causas no siempre resulta posible hacerlo.

Las posibles causas para una perdida de información son:

- Deterioro o borrado accidental por parte de un usuario autorizado.

- Ataque intencionado por parte de personas no autorizadas.
- Fallo del software.
- Fallo del hardware.
- Virus.
- Incendio, robos, y desastres naturales, etc.

La peor de todas es el borrado accidental por parte de un usuario autorizado, porque es totalmente imposible de prevenir.

Algo puede hacerse. Por ejemplo en Linux hay sustitutos para el comando 'rm' que dejarán los ficheros borrados en una papelera. También se pueden seguir ciertas recomendaciones de prudencia que minimicen los riesgos mediante determinadas prácticas pero siempre cabrá la posibilidad de un fallo humano. El tema es amplísimo y nos vamos a limitar a tratar solo de Respaldos o Copias de seguridad.

/ = = = = = = = = = = = = = = = = = /

Borrado accidental de ficheros

Debe saber que la operación normal de borrado no destruye toda la información de los ficheros. Si alguna vez borra algo muy importante en un sistema de ficheros y no tiene copia de seguridad. Deberá evitar volver a trabajar sobre ese sistema de ficheros en modo lectura-escritura. Por el contrario deberá montar ese sistema de ficheros en modo solo-lectura y confiar en que quede algo recuperable en algún lado. Hay programas que pueden ayudar en estas situaciones siempre que la información no esté machacada con posteriores operaciones de escritura sobre esa partición. Si no sabe que hacer no haga nada y pregunte antes de intentar cosas que podrían destruir definitivamente toda la información. Este tipo de recuperaciones son solo un último recurso y las posibilidades de éxito no son muy grandes.

Una cosa que se puede intentar en modo read-only (solo-lectura) es leer directamente del dispositivo intentando filtrar cadenas de ficheros

presentes en los ficheros borrados. Por ejemplo usando grep. Si las cadenas aparecen es señal de que no se ha destruido toda la información.

/ = = = = = = = = = = = = = = = /

Características de un buen sistema de respaldo

Un sistema de respaldo debe de estar perfectamente planificado. Su planificación de forma automática no siempre es posible debido a que requieren con frecuencia intervención humana. Especialmente la copias que no caben en una sola cinta (multivolumen). Lo ideal es organizar su propio sistema de respaldo en función de las necesidades de su sistema. En un sistema es conveniente clasificar las diferentes partes de la información atendiendo a su importancia. Lo ideal sería hacer respaldo de todo regularmente manteniendo distintas copias en rotación. Es decir si tenemos tres cintas para almacenar respaldos (Cinta A, Cinta B, y Cinta C) deberemos ir grabando siguiendo una secuencia circular A, B, C, A, B, C,... La frecuencia de los respaldos se determina en función de lo importante que sean los datos. Generalmente es bueno hacer una copia diaria al menos de la parte del sistema que suponga mayor perdida de horas de trabajo en caso de incidente. De todas formas hay que tener en cuenta que este sistema solo nos permite mantener copia de los últimos tres o cuatro días. Esto no es suficiente porque muchas veces nos interesa recuperar una información de hace una semana, un mes, o incluso más tiempo. Por ello algunas veces se usan rotaciones de siete cintas (una para cada día de la semana) y una vez al mes se saca otra copia en una cinta distinta que se conserva por un espacio de tiempo más largo según convenga.

Una práctica común es guardar regularmente copias en otro edifico distinto para protegerse de esa forma de incendios, inundaciones, terremotos, robos. Si después de un desastre conservamos los datos vitales de nuestro negocio quizás podamos resurgir de nuestras cenizas más tarde.

Algunas veces no es posible hacer demasiadas copias de todo el sistema con demasiada frecuencia. Por ello se pueden delimitar areas con distinto grado de necesidad de respaldo. El sistema operativo y las aplicaciones no suelen variar y pueden ser recuperadas desde CDs de instalación. Es importante siempre mantener copias de todo el sistema. En cualquier caso siempre ha de usarse rotación de cintas porque usar una sola copia es muy arriesgado.

Las cintas son razonablemente seguras pero se pueden romper con el uso. De todas formas es importante que la planificación de respaldo del sistema no resulte complicada. Trabajar con demasiados conjuntos de cintas distintos no es bueno.

En la copia general de todo el sistema seguramente se guardarán cosas que varían poco pero que quizás costaron mucho trabajo configurar. Por tanto es una copia que no necesita hacerse con frecuencia pero deberemos disponer siempre de algunas copias en buen estado aunque no sean muy recientes.

Todas las copias deben ser sometidas de forma automática a un proceso de verificación que en el caso más sencillo consiste en leer la lista de ficheros que contiene. No es muy fiable pero al menos detectaremos los fallos más importantes. En el mejor de los casos la comprobación se hace sobre el contenido completo comparándolo con el original. En este caso cuando la copia termina el original puede haber cambiado y por ello la detección de diferencias entre original y copia puede deberse a un cambio en el original en lugar de un error en la copia.

Una cosa que se usa mucho son las copias globales combinadas con posteriores copias incrementales. Es un sistema cuya principal ventaja es la comodidad.

Copia incremental significa que solo salvamos lo último que ha cambiado desde la última copia global. Por eso cada nueva copia incremental resulta generalmente de mayor tamaño que la anterior.

Si la copia global ocupa varias cintas. Podemos hacer copias incrementales hasta que debido a su progresivo aumento de ocupación no quepan en una sola cinta por ejemplo, momento en el que se puede hacer una nueva copia global. En general podemos hacer una nueva copia global cuando la incremental empiece a resultar incomoda debido a su progresivo aumento de tamaño.

Este sistema de copia hace recaer demasiada responsabilidad en una sola cinta. La última copia global suele ser muy voluminosa y puede contener información que no se encuentre en ninguna otra cinta.

Hay actividades que pueden generar enormes ficheros de datos y eso incrementará mucho la ocupación de nuestra siguiente copia incremental. Por ejemplo si estamos generando una imagen animada de gran tamaño o si estamos descargado de Internet imágenes de CDs enteras deberíamos guardar esos datos en un sistema de ficheros especialmente dedicado a ello. De esta forma cuando hagamos respaldo podemos desmontar antes esa partición para poder realizar respaldos independientes con la frecuencia que convenga en cada caso.

Siempre que se añadan paquetes, aplicaciones, se configure algo conviene volver a sacar una nueva copia global de todo el sistema.

/ = = = = = = = = = = = = = = = = = /

Los soportes de la información

El soporte más adecuado son las cintas. Es un sistema que supone un fuerte desembolso inicial pero a la larga resulta más rentable y más seguro que los demás. Además cuando cambiemos de ordenador podremos conservarlo.

En cualquier caso cada cinta debe de estar perfectamente etiquetada (Identificador clave de la cinta, Tipo de copia, máquina, contenido, formato, periodicidad, etc...)

Después en un papel habrá que registrar cada copia realizada con identificador de la cinta y fecha de la copia que contiene.

Es muy importante tener un sistema bien pensado para que ofrezca el máximo de seguridad con el máximo de comodidad. Hay que establecer un buen plan que deberá convertirse en una rutina de trabajo. Considere la incomodidad como un gravísimo peligro porque puede derivar en incumplimiento de esa importante obligación.

El día que no cumplamos nuestra obligación de hacer copia de respaldo será el día que el sistema elija para darle un disgusto (ley de Murphy). Para resultar cómodo debemos automatizar con scripts nuestro sistema de respaldo y las copias no deben tardar más de lo necesario.

Los discos duros extraibles son una buena alternativa para sistemas caseros. Lo que no tiene ningún sentido es hacer copias de seguridad en el mismo disco duro que contiene los datos aunque se utilice una partición diferente. Recuerde que los discos duros pueden perder toda su información de forma repentina.

Para guardar datos de interés histórico que deban conservarse durante años suele ser muy interesante usar CDs.

Los disquetes son un medio muy poco fiable y cabe poca cosa.

/ = = = = = = = = = = = = = = = = /

Respaldo por partes

Antes que nada debe plantearse si realmente necesita hacer respaldo por partes. Recuerde que no conviene complicar mucho el plan de respaldo y no hay nada más sencillo que hacer respaldo de todo de una vez siempre que el tamaño de los datos lo permitan.

Para determinar la necesidad de respaldo de las partes del sistema conviene que repase la lección de 'Sistema de Ficheros III' Concretamente el párrafo dedicado a la 'Estructura estándar del sistema de ficheros de Linux'. Esto le ayudará a comprender la naturaleza de los datos en las distintas partes del sistema.

Recuerde que pueden existir variaciones, en función de cada distribución.

Cada sistema dependiendo de la actividad a la que sea dedicado tendrá unas necesidades totalmente especiales de respaldo.

Las partes de mayor interés con vistas a respaldos frecuentes suelen ser /home/ , /root/, y /usr/local/ En algunos casos /var/log o un directorio que contenga una base de datos pueden ser muy importantes pero eso son cosas que cada cual debe determinar. Si lo importante ocupa más del 50% del sistema lo más sencillo sería por copias globales e incrementales de todo el sistema, pero en un ordenador de uso personal con gran cantidad de paquetes instalados la parte importante a la hora de hacer respaldo no suele ser tan grande.

La parte más variable del sistema está en /var/ y concretamente en /var/log es donde se registran los logs del sistema que son ficheros que informan de todo tipo de cosas que van sucediendo en el mismo. En algunos sistemas estos logs tienen una importancia mayor que en otros. Por ejemplo si existe alguna aplicación que toma estos logs para elaborar estadísticas. En otros sistemas no resultará demasiado crítico que la última copia de /var/ no sea muy reciente pero que exista al menos una copia si es fundamental.

Hacer las cosas en un solo paso no es siempre buena idea. Lo principal es dar pasos que puedan deshacerse si nos equivocamos en algo, por eso siempre que sea posible conviene recuperar en un lugar distinto al original. Luego si todo ha ido bien podemos trasladar las cosas a su destino definitivo. Los que se preguntan para que tantas precauciones son los que más tarde se preguntarán así mismo cosas bastante lamentables.

Cuando se pierde toda la información de un sistema tendremos que recuperarlo todo partiendo de los discos de instalación y de la copia.

/ = = = = = = = = = = = = = = = = /

Programas de utilidad para realizar respaldos

Para realizar copias de seguridad podemos usar aplicaciones completas pensadas para ello o fabricarnos algo a medida combinando una serie de utilidades de las cuales la mayoría ya han sido mencionadas alguna vez en capítulos anteriores. Esto último es lo que vamos a hacer nosotros porque así podrá diseñar algo personalizado y también porque resulta más didáctico.

Nosotros solo vamos describir todas estas utilidades de forma muy enfocada al uso que vamos a hacer de ellas. Puede usar el manual para consultar lo que no necesite.

Los programas más útiles son aquellos que sirven para empaquetar dentro de un solo fichero una gran cantidad de ficheros y directorios, como por ejemplo 'tar', 'cpio', y 'afio' pero de ellos hablaremos en el siguiente capítulo. Antes vamos a comentar otra serie de programas que pueden combinar con estos empaquetadores o que su uso puede ser interesante en tareas relacionadas con el respaldo.

find(1) Se usará para elaborar una lista de ficheros que deseamos salvar. Su versatilidad es enorme. La opción -newer resultará de especial interés en el caso de desear hacer copias incrementales. Observe que nosotros nos situaremos previamente en un directorio adecuado y luego usaremos 'find . ' para pasar a find nombres que empiezan con '.'. Es decir usaremos caminos relativos en lugar de caminos absolutos. Esto tiene su importancia porque de ese modo luego podremos recuperar la información situándola en otro lugar distinto al original si fuera necesario.

egrep(1) Se usa para intercalarlo entre la salida de 'find' y la entrada de 'cpio' de esa forma se aumenta la flexibilidad de uso. Se puede usar para incluir determinados ficheros que cumplan determinada expresión regular y/o para excluir determinados ficheros que cumplan determinada expresión regular. Para excluir se usa la opción -v.

gzip(1) Es el compresor de uso más frecuente en Linux. Esta muy probado y es muy fiable. Cuando se coloca un compresor como gzip o bzip2 a la salida de 'cpio' o de 'tar' se pierde la posibilidad de hacer copia multivolumen porque el compresor es el que está escribiendo en el dispositivo en lugar de hacerlo 'cpio' o 'tar'. Afortunadamente 'afio' si puede hacerlo.

bzip2(1) Es más eficiente en el sentido de que puede comprimir algo más pero a costa de tardar mucho más.

diff(1) Detecta diferencias de contenido entre dos ficheros. Especialmente adecuado para ficheros de texto.

md5sum(1) Permite asociar un código de comprobación al contenido de un fichero o dispositivo, etc..

dircomp(1) Permite comparar dos directorios analizando su contenido recursivamente.

dirdiff(1) Permite comparar dos directorios analizando su contenido recursivamente. Es más elaborado que el anterior.

mt(1) Es la utilidad que se usa para controlar las cintas magnéticas. Permite operaciones de rebobinado, retension, lectura de estado, etc.

dd(1) Sirve para copiar o leer dispositivos usando el tamaño de bloque que se desee. También permite realizar ciertas conversiones.

Por ejemplo para copiar un disquete:

```
$ dd if=/dev/fd0 of=fichero
$ dd of=/dev/fd0 if=fichero
```

touch(1) Sirve para actualizar la ficha de modificación de un fichero creándolo en caso de que no exista y se puede usar para guardar el instante en que se realiza una copia para luego tomar ese momento como referencia en copias incrementales.

Por ejemplo para modificar la fecha de modificación de un fichero ajustándola con la hora y fecha actual haríamos lo siguiente:

```
$ touch fichero
```

Para modificar la fecha de modificación de un fichero ajustándola con la fecha de ayer haríamos:

```
$ touch -t yesterday fichero
```

Para modificar la fecha de modificación de un fichero ajustándola con la misma fecha que la de un fichero de referencia haríamos:

```
$ touch -r fich_ref fichero
```

En todos los casos si el fichero no existe se creará con tamaño igual a cero.

tee(1) Permite bifurcar una salida en dos. De esa forma podemos guardar la lista de ficheros que estamos entregando a 'cpio' o a 'afio' con objeto de poder compararlas luego por ejemplo.

mount(1) Monta un sistema de ficheros.

umount(1) Desmonta un sistema de ficheros.

/ = = = = = = = = = = = = = = = = /

Programas para empaquetar ficheros 'tar', 'cpio', y 'afio'

Vamos a ver ahora una serie de programas capaces de empaquetar varios ficheros en uno solo.

tar(1) Es similar a cpio en el sentido de que sirve tanto para salvar como para recuperar ficheros. La salida puede ser a dispositivo, fichero o a un proceso igual que cpio, pero no acepta por su entrada estándar una lista de ficheros. Hay que pasarlos como argumentos.

'tar' admite nombres de ficheros y directorios como argumentos. Los directorios serán tratados recursivamente. salvando y recuperando todo su contenido. En Linux 'tar' tiene opciones para obtener una salida ya comprimida. Para especificar el fichero o el dispositivo donde se guardará la copia se usa -f archivo.

En lugar de archivo se puede especificar un dispositivo (/dev/dispositivo) o también se puede usar entrada salida mediante '-f -'.

Para crear la copia se usa -c, para leer la lista de fichero se usa -t, y para extraer los fichero se usa -x. Se puede usar compresión con la opción -z.

Veamos algunos ejemplos:

```
$ # Salvar el directorio actual completo en
'fichero.tar'
$ tar -cf fichero.tar .

$ # Lista abreviada (solo nombres de ficheros) del
contenido
$ # de la copia 'fichero.tar'
$ tar -tf fichero.tar

$ # Lista detallada del contenido de la copia
'fichero.tar'
```

```
$ tar -tvf fichero.tar

$ # Extracción completa del contenido  de la copia
'fichero.tar'
$ tar -xf fichero.tar
```

cpio(1) Se puede usar para salvar o recuperar una serie de ficheros. Su salida puede ser enviada a un dispositivo a un fichero o a un compresor. La lista de ficheros de un paquete cpio puede obtenerse con la opción -t. Esta lista de ficheros solo incluye los nombres de los mismos. Guardando esta lista en un fichero, editándolo después de forma que eliminemos todos menos aquellos que queramos recuperar podremos usarlo para recuperar con la opción -i. La opción -t combinada con -v obtiene no solo los nombres sino también permisos, fechas tamaños de ficheros, etc.. Es decir 'cpio -t' actúa como un 'ls' y 'cpio -tv' actúa como 'ls -l'. La opción -d combinada con '-i' permite la creación de directorios'. La opción '-m' conserva la fecha de modificación de los ficheros. La fecha de modificación de los directorios no siempre se conserva pero no suele considerarse importante. Seguramente el propio 'cpio' al copiar ficheros dentro del directorio creado se provoca la alteración de esta fecha y la única forma de respetar la fecha de modificación de los directorios sería hacer una segunda pasada una vez que todos los ficheros estuvieran creados. Puede que alguna utilidad o alguna versión de cpio si lo haga. Si tiene dudas compruebelo. La opción '-p' de cpio permite copiar estructuras enteras desde un lugar a otro. La recuperación de ficheros con cpio admite uso de patrones. Es interesante comprobar que aquí el operador '*' también expandirá caracteres '/' cosa que por ejemplo el bash y otros programas no hacen.

Algunos ejemplos sencillos:

```
$ #Salvar un fichero
$ echo fichero | cpio -o > fich.cpio

$ #Salvar varios ficheros
```

```
$ cpio -o > fich.cpio < cat lista_ficheros'

$ #Lista detallada de los ficheros contenidos en la
copia
$ cpio -itvc < fich.cpio

$ #Recuperacion de los ficheros creando directorios
cuando sea
$ #necesario (-d) y conservando la fecha de
modificación los ficheros
cpio -ivcdm < fich.cpio
```

Para usar un dispositivo en lugar de un fichero bastaría con sustituir en el ejemplo 'fich.cpio' por '/dev/dispositivo'.

Para evitar conflictos en sistemas de ficheros con número de inodos elevado se usa la opción (-H newc). Recomendamos que la use siempre.

La opcion -B usará bloques de 5120 bytes en lugar de bloques de 512 bytes.

Vamos a usar esta opción en los siguientes ejemplos y además usaremos entrada salida a cinta con compresión de datos.

```
$ # Salvar todo el rabol de directorios desde el
directorio actual
$ # en formato comprimido.
$ find . | cpio -ovBH newc | gzip > /dev/st0

$ # Listado detallado del contenido de la copia
$ gzip -d < /dev/st0 | cpio -itvBH newc

$ # Guardar listado sencillo (solo nombres de
ficheros) en
$ # el fichero 'contenido.txt'
$ gzip -d < /dev/st0 | cpio -itBH newc  >
contenido.txt

$ # Suponemos que editamos el fichero anterior
'contenido.txt'
```

```
$ # y dejamos solo aquellos ficheros que deseamos
recuperar en
$ # el fichero 'selección'.
$ # Recuperamos de la siguiente forma:
$ gzip -d < /dev/st0 | cpio -ivudmBH newc `cat
seleccion`
```

Con la opcion -p podemos copiar todo el contenido de un directorio a otro directamente sin pasar por una salida intermedia empaquetada en formato 'cpio'. Esto es muy util para trasladar directorios enteros.

```
$ cd $DIR_ORIGEN
$ find . | cpio -pdm $DIR_DESTINO
```

afio(1) Es una interesantísima variación de cpio. Tiene entre otras cosas la posibilidad de comprimir los ficheros de uno en uno. Esto le proporciona mayor seguridad, ya que una copia de cpio o de tar comprimida tiene el grave problema de que en caso de un pequeño error en la copia, toda la copia queda inutilizada. Con afio solo quedará dañado el fichero donde ocurra en error. Además con 'afio' es posible hacer copias multivolumen comprimidas. 'afio' reconoce las extensiones de aquellos ficheros que no interesa comprimir por que ya lo están: (.Z .z .gz .bz2 .tgz .arc .zip .rar .lzh .lha .uc2 .tpz .taz .tgz .rpm .zoo .deb .gif .jpeg .jpg .tif .tiff y .png). Una ventaja más de 'afio' es que permite verificar la copia con el original. (Opción -r)

Para hacer pruebas con 'afio' situese en el directorio /tmp. Genere algunos ficheros grandes para poder hacer luego pruebas de copias de seguridad usando disquetes. Por ejemplo para generar un fichero de 2Mbytes de perfecta basura puede usar lo siguiente:

```
$ cd /tmp
$ mkdir pruebafio
$ cd pruebafio
$ echo 1 > fich1
```

```
$ gzip --fast < /dev/zero | head --bytes=2m  > fich2
$ echo 3 > fich3
$ mkdir dir1
$ echo 4 > dir1/file4
$ cp fich2 fich5.gz
$ ls -lR

.:
total 4116
drwxr-xr-x    2 root     root         4096 abr  8
13:06 dir1
-rw-r--r--    1 root     root            2 abr  8
13:05 fich1
-rw-r--r--    1 root     root      2097152 abr  8
13:05 fich2
-rw-r--r--    1 root     root            2 abr  8
13:06 fich3
-rw-r--r--    1 root     root      2097152 abr  8
13:33 fich5.gz

./dir1:
total 4
-rw-r--r--    1 root     root            2 abr  8
13:06 file4
```

Bueno con esto tenemos una estructura de ficheros para hacer pruebas. Vamos a hacer pruebas de copias multivolumen comprimidas que son las más interesantes y usaremos disquetes de 1.44Mbytes. Ya hemos dicho que los disquetes no son fiables pero su pequeño tamaño lo hace ideal para hacer pruebas de copias multivolumen. Para usar disquetes conviene usar el parámetro -F.

Entre otras cosas advierta que tenemos dos ficheros idénticos 'fich2' y 'fich5.gz' que admiten una compresión mejor que la realizada con la opción --fast de gzip. Pero uno de ellos será reconocido como fichero comprimido y se guardará tal cual está y el otro será comprimido. (Por cierto de forma espectacular).

Formatee un par de disquetes de 1.44MBytes con 'mformat a:'

Compruebe que se encuentra en /tmp/pruebafio y ejecute los comandos siguientes:

```
$ # Para salvar a disquete multivolumen comprimido
$ find . | afio -o -v -s 1440k -F -Z /dev/fd0

. -- okay
fich1 -- okay
fich5.gz -- okay
fich3 -- okay
dir1 -- okay
dir1/file4 -- okay
fich2.z -- (00%)
afio: "/dev/fd0" [offset 2m+7k+0]: Next disk needed
afio: Ready for disk 2 on /dev/fd0
afio: "quit" to abort,"f" to format, anything else to
proceed. >
afio: "/dev/fd0" [offset 2m+7k+0]: Continuing


$ # Alteramos el original de fich3 en disco
$ echo jjjjj > fich3
$ #
$ # Volver a colocar el primer disquete de la copia
$ # Para comprobar con el original una copia
comprimida en varios disquetes
$ afio -r -v -s 1440k -F -Z /dev/fd0

drwxr-xr-x  1 root     root              Apr  8
13:33:23 2001 .
-rw-r--r--  1 root     root            2 Apr  8
13:05:24 2001 fich1
-rw-r--r--  1 root     root      2097152 Apr  8
13:33:11 2001 fich5.gz
afio: "/dev/fd0" [offset 1m+416k+0]: Input limit
reached
afio: Ready for disk 2 on /dev/fd0
afio: "quit" to abort,"f" to format, anything else to
proceed. >
afio: "/dev/fd0" [offset 1m+416k+0]: Continuing
-rw-r--r--  1 root     root            2 Apr  8
13:06:00 2001 fich3
afio: "fich3": Corrupt archive data
drwxr-xr-x  1 root     root              Apr  8
```

```
13:06:18 2001 dir1
-rw-r--r--  1 root     root            2 Apr  8
13:06:18 2001 dir1/file4
-rw-r--r--  1 root     root         4939 Apr  8
13:05:48 2001 fich2 -- compressed

$ # Volver a colocar el primer disquete de la copia
$ # Para obtener la lista de ficheros de una copia
comprimida en disquete
$ afio -t -v -s 1440k -F -Z /dev/fd0

drwxr-xr-x  1 root     root              Apr  8
13:33:23 2001 .
-rw-r--r--  1 root     root            2 Apr  8
13:05:24 2001 fich1
-rw-r--r--  1 root     root      2097152 Apr  8
13:33:11 2001 fich5.gz
afio: "/dev/fd0" [offset 1m+416k+0]: Input limit
reached
afio: Ready for disk 2 on /dev/fd0
afio: "quit" to abort,"f" to format, anything else to
proceed. >
afio: "/dev/fd0" [offset 1m+416k+0]: Continuing
-rw-r--r--  1 root     root            2 Apr  8
13:06:00 2001 fich3
drwxr-xr-x  1 root     root              Apr  8
13:06:18 2001 dir1
-rw-r--r--  1 root     root            2 Apr  8
13:06:18 2001 dir1/file4
-rw-r--r--  1 root     root         4939 Apr  8
13:05:48 2001 fich2 -- compressed


$ # Creamos un segundo directorio y nos situamos en
el
$ mkdir ../pruebafio2
$ cd ../pruebafio2
$ #
$ # Para recuperar una copia comprimida en disquete
$ # Volver a colocar el primer disquete de la copia
$ afio -i -v -s 1440k -F -Z /dev/fd0

. -- okay
fich1 -- okay
afio: "/dev/fd0" [offset 1m+416k+0]: Input limit
reached
```

```
afio: Ready for disk 2 on /dev/fd0
afio: "quit" to abort,"f" to format, anything else to
proceed. >
afio: "/dev/fd0" [offset 1m+416k+0]: Continuing
fich5.gz -- okay
fich3 -- okay
dir1 -- okay
dir1/file4 -- okay
fich2 -- uncompressed

$ # Vamos a comparar los dos directorios
$ cd ..
$ dircomp pruebafio pruebafio2

< 3241346245 6 ./fich3
> 4164007125 2 ./fich3

$ #
$ # Un análisis más detallado de las diferencias de
ambos directorios
$ # Se puede hacer con dirdiff
$ dirdiff pruebafio pruebafio2
2 d d - > =
2 d d - > =  dir1
= r r = = =  dir1/file4
= r r = = =  fich1
= r r = = =  fich2
1 r r X < =  fich3
= r r = = =  fich5.gz
^ ^ ^ ^ ^ ^
| | | | |  \__ POG
| | | |  \____ DAT
| | |  \______ CKS
| |  \________ FI2
|  \__________ FI1
 \____________ FI2


$ # Para comprobar dos ficheros podemos usar md5sum
$ md5sum pruebafio/fich1 pruebafio2/fich1
pruebafio/fich3 pruebafio2/fich3

md5sum pruebafio/fich1 pruebafio2/fich1
pruebafio/fich3 pruebafio2/fich3
b026324c6904b2a9cb4b88d6d61c81d1  pruebafio/fich1
b026324c6904b2a9cb4b88d6d61c81d1  pruebafio2/fich1
```

```
75894c2b3d5c3b78372af63694cdc659  pruebafio/fich3
6d7fce9fee471194aa8b5b6e47267f03  pruebafio2/fich3
```

Las páginas man de 'afio' contienen algunos ejemplos interesantes. Por ejemplo viene como hacer una copia sobre una unidad de cdrom.

find . | afio -o -b 2048 -s325000x -v '!cdrecord .. -'

Sin duda alguna afio es un excelente programa para sus respaldos y le recomendamos que lo ponga a prueba y se familiarice con él.

Resumiendo Conviene conocer tanto 'cpio' como 'tar' porque son dos clásicos de los sistemas operativos tipo Unix y ambos son muy utilizados. Cada uno tiene sus ventajas y sus inconvenientes. Por otra parte 'afio' es mucho más reciente y por lo tanto su uso no es tan amplio, pero eso podría cambiar en un futuro gracias a sus indudables mejoras técnicas respecto a los dos clásicos antes mencionados.

El único motivo que se me ocurre por el cual se podría desaconsejar el uso de 'afio' es por temas de portabilidad entre distintas máquinas, con distintos SO. En ese caso para garantizar que la copia tenga un formato lo más universal posible, lo recomendable sería usar 'tar' sin compresión, pero eso no justifica dejar de usar 'afio' para las copias de seguridad rutinarias renunciando a sus indudables ventajas.

Observe que la salida de información siempre que resulta posible se hace a salida estandar. En caso que se generen dos salida se usa la salida estandar como salida principal. La salida estandar de errores se usará como salida secundaria mezclada con posibles mensajes de error.

Concretamente la opción -t en 'tar' y 'cpio' entregan a salida estandar el resultado pero la opcion -o y la opción -v de 'cpio' juntas hacen que la salida de la copia se dirija a salida estandar y la lista de los ficheros se dirijirá entonces a la salida estandar de errores.

En cualquier caso las posibilidades de cualquiera de estos tres programas son muy variadas. Es imprescindible acudir va las páginas man de cada uno de ellos para tener una visión completa de sus posibilidades. Nosotros nos hemos limitado explicar con ayuda de ejemplos algunas de sus posibilidades.

/ = = = = = = = = = = = = = = = /

Un programa completo kbackup(1)

Es un programa muy completo para hacer backups. Es 100% libre (Licencia Artistic) y está escrito en lenguaje shell-script. La interfaz de usuario a base de un completo sistema de menús está programado haciendo uso de 'dialog'. Permite usar encriptación pgp. Compresión mediante doble buffering para para optimizar la escritura en cintas y como herramienta de empaquetado de ficheros usa 'afio' o 'tar' según convenga.

Es altamente configurable y versátil

Permite almacenar distintas configuraciones para distintos tipos de copias.

Si no quiere realizar un sistema a su medida, puede usar 'kbackup' para sus copias.

/ = = = = = = = = = = = = = = = /

Recuperación total desde cero

Muchas veces tener un disquete o CDROM de rescate con las herramientas necesarias simplifica mucho las cosas. Realmente todo podría resultar tan sencillo como arrancar con ese sistema de rescate y recuperar la cinta sobre el sistema de ficheros adecuadamente montado.

Si usa programas específicos como 'kbacup' deberá investigar la forma de recuperar una copia de 'kbacup' sin usar 'kbacup'. En su defecto debería tener un disquete o cdrom de rescate (autoarrancable) que contenga 'kbackup'.

Las copias de kbacup pueden ser manejadas por 'afio' o 'tar'. Por ejemplo en caso de perdida de todo el sistema quizás no tengamos forma de hacer funcionar 'kbackup' pero si 'tar' o 'cpio'. En este caso es importante saber que las copias de 'kbacup' escriben en primer lugar un fichero de cabecera que incluye información sobre el formato que se uso para esa copia.

En cualquier caso debe de tener claro cual será la recuperación de una cinta de seguridad de todo el sistema desde cero, y si tiene posibilidades de hacer pruebas sin comprometer datos importantes hagalo. Use por ejemplo un segundo ordenador sin información importante.

/ = = = = = = = = = = = = = = = = = /

Copia usando dispositivo de cinta SCSI

El uso de todos los dispositivos de cinta SCSI es bastante similar. Lo usaremos como para nuestro ejemplo. Estos dispositivos tienen varios controladores y nos fijaremos ahora en dos de ellos.

/dev/st0 Dispositivo de cinta SCSI Con autorebobinado.
/dev/nst0 Dispositivo de cinta SCSI Sin autorebobinado.

El autorebobinado significa que el cierre del dispositivo provoca el rebobinado automático al principio de la cinta. Nosotros usaremos para la copia de seguridad /dev/st0 (Con auto rebobinado) Solo usaremos /dev/nst0 para rebobinado y retension de la cinta. El retensionado de la cinta es una operación muy necesaria y en principio el programa de ejemplo provocará siempre un retensionado de la cinta. Las cintas que no son retensionadas de vez en cuando puede llegar a romperse o atascarse.

Los controles que vamos a usar sobre la cinta son:

mt -f /dev/nst0 rewind
mt -f /dev/nst0 retension
mt -f /dev/st0 status

Los dos primeros ya han sido mencionados y el último proporciona información sobre el estado de la cinta.

Vamos a proporcionar un script para hacer respaldos basado en el uso de afio. Deberá estudiarlo para adaptarlo a sus necesidades.

Está pensado para usar cintas scsi. Se puede adaptar fácilmente para otros dispositivos o para copia sobre fichero. Si esto no le gusta siempre puede usar un programa completo de backup como 'kbackup' y si le gusta pero lo encuentra incompleto puede encontrar solución a sus necesidades concretas inspeccionando el código de 'kbackup' que está escrito en shell-script.

Un programa de backup más que versátil ha de ser cómodo. Ya lo dijimos antes. El backup es una tarea rutinaria y muy importante. Interesa tener un sistema que permita meter la cinta, dar dos o tres teclazos y dejar que el resto lo haga el ordenador. De esa forma el día que estemos cansados no tendremos excusa para no hacer la copia.

Lo que viene a continuación es un ejemplo y los ejemplos han de ser sencillos, pero estamos seguros que le resultará muy fácil adaptarlo a sus necesidades.

En este ejemplo nosotros excluimos /mnt porque es un directorio donde suponemos que hay montados dispositivos que no deben ser respaldados.

Para hacer un respaldo sobre un fichero conviene usar un disco distinto montado en un lugar que quede excluido de la copia. En nuestro ejemplo podríamos situar el fichero de respaldo dentro de /mnt.

Analice el script a conciencia. Acuda al manual on line para cada uno de los comandos que representen alguna duda y si a pesar de eso, algo sale mal insistimos en recordarle que toda la responsabilidad en el uso de esto o de cualquier otro material del curso es solo suya.

```
#!/bin/bash
#########################################################
################
testTape(){
echo "## Inserte la cinta ${DATOS} ##"
echo ".... pulse <Intro> para continuar"
read
echo
mt -f /dev/nst0 rewind
mt -f /dev/st0 status
echo
afio -rvZ -b ${BLOCK} -s ${CAPCINTA} < /dev/st0  |
more
}

#########################################################
################
preparar(){
REF_INCR=""
if [ "$TIPO" = "I" ]
   then
      REF_INCR=T_${DATOS}
fi
if [ "$TIPO" = "i" ]
   then
      REF_INCR=I_${DATOS}
fi


T_DATOS=${TIPO}_${DATOS}
FECHA=`date +%y%b%d`
PREOUT="${BACKUP}/history/${FECHA}${T_DATOS}"
PRELAST="${BACKUP}/last/${T_DATOS}"
T_REF_INCR="${BACKUP}/last/${REF_INCR}.time"
FIFO3=/tmp/FIFO_salva_lst.$$
FIFO3OUT=${PREOUT}.lst

echo
```

```
echo '############ ' ${FECHA} ${T_DATOS} '
############'
echo
'-----------------------------------------------------
----------'
df
echo
'-----------------------------------------------------
----------'
if [ "$TIPO" = "i" ] || [ "$TIPO" = "I" ]
   then
      echo "Se recuperará información posterior
a ..."
      ls -ld $T_REF_INCR | cut -c 42-
fi
echo
echo "desde=$DESDE, incluir=$INCLUIR,
excluir=$EXCLUIR, datos=$DATOS"
echo "!! Compruebe que los dispositivos montados
coinciden para $DATOS !!"
echo ".... pulse <Intro> para continuar"
read
echo "## Inserte la cinta. Asegurese que está
desprotegida ##"
echo "....Espere a que esté lista y pulse <Intro>
para continuar"
read
echo
mt -f /dev/nst0 rewind
mt -f /dev/st0 status
echo "Espere. Realizando prueba de escritura ...."
if ! echo "____ Prueba de grabacion OK ____" |
blockout > /dev/st0
   then
     echo  " !!! Imposible escribir !!!"
     exit
   else
     blockout < /dev/st0
fi
mt -f /dev/nst0 rewind
echo
echo -e '\7\c' sleep 1
echo -e '\7\c' sleep 1
echo -e '\7\c' sleep 1
echo "## Compruebe estado de la cinta ##"
echo ".... pulse <Intro> para continuar"
```

```
read
}

#########################################################
################
salva(){

RETENS="S"
clear
echo "############ !! Backup en marcha !!
############## $T_DATOS"
COMIENZO=`date`
if [ "$RETENS" = "S" ]
   then
      echo "Retension de la cinta ............."
      mt -f /dev/nst0 retension
      sleep 5
fi
cd $DESDE
touch ${PRELAST}.time
chmod 400 ${PRELAST}.time
echo "Salidas en ${PREOUT}."

INI=`date "+%d %b %Y %r"`
echo "Grabando en cinta la copia de
seguridad ............"

mknod --mode=600 $FIFO3 p
cut -b 3-  < $FIFO3 | egrep -v '^$' | sort  >
$FIFO3OUT &
rm ${PRELAST}.lst 2> /dev/null
ln -s $FIFO3OUT ${PRELAST}.lst

### Comienza realmente la copia ##
if [ "$TIPO" = "T" ]
   then
      nice find .  -print | egrep -v $EXCLUIR | egrep
$INCLUIR | \
      tee $FIFO3 | afio -ovZ -b ${BLOCK} -s $
{CAPCINTA}  /dev/st0
   else
      nice find . -newer $T_REF_INCR -print  | egrep
-v $EXCLUIR | \
      egrep $INCLUIR  | tee $FIFO3 | afio -ovZ -b $
{BLOCK} \
      -s ${CAPCINTA} /dev/st0
```

```
fi
### Fin ###
FIN=`date "+%r"`
echo "$INI ======> $FIN"
mt -f /dev/st0 status
echo "Copia en cinta terminada ......."
echo "Iniciando comprobación ........."
afio -tZ  -b ${BLOCK} -s ${CAPCINTA} /dev/st0 | sort
> ${PREOUT}.tape
diff  $FIFO3OUT ${PREOUT}.tape  > ${PREOUT}.diff
echo
"**************************************************
**************"
echo "******************* F I N    B A C K U P
************************"
echo
> ${PREOUT}.log
echo "******* ${PREOUT}.log "
>> ${PREOUT}.log
echo "******* Comenzó  $COMIENZO"
>> ${PREOUT}.log
echo "******* Finalizó `date`"
>> ${PREOUT}.log
echo "******* Diferencias `cat ${PREOUT}.diff | wc
-l` "     >> ${PREOUT}.log
echo "******* DiferenciasHead `head -n 8 $
{PREOUT}.diff` "   >> ${PREOUT}.log
echo " ...."
>> ${PREOUT}.log
echo "******* DiferenciasTail `tail -n 8 $
{PREOUT}.diff` "   >> ${PREOUT}.log
echo "bzip2  $FIFO3OUT ${PREOUT}.tape ${PREOUT}.diff"
      bzip2  $FIFO3OUT ${PREOUT}.tape ${PREOUT}.diff
echo "rm $FIFO3"
      rm $FIFO3
echo
"**************************************************
**************"
cat ${PREOUT}.log
echo
"**************************************************
**************"
}

##(1)#######################################
salva_tmp(){
```

```
DESDE='/tmp'
INCLUIR=$INCLUIR_TODOS
EXCLUIR=$EXCLUIR_NINGUNO
DATOS="tmp"
preparar
salva
}

##(2)#######################################
salva_root(){
DESDE='/'
INCLUIR=$INCLUIR_TODOS
EXCLUIR="\.netscape/cache/|\/core$|^./tmp|^./proc|^./
mnt|^./mount"
DATOS="root"
preparar
salva
}

##(3)#######################################
salva_home(){
DESDE='/home'
INCLUIR=$INCLUIR_TODOS
EXCLUIR=$EXCLUIR_NINGUNO
DATOS="home"
preparar
salva
}

##########################################################
###################
################################## MAIN
###################################
##########################################################
###################
CAPCINTA=3700m
BLOCK=10k
INCLUIR_TODOS="^."
EXCLUIR_NINGUNO="^./tmp|^./proc"
BACKUP="/backup"
mkdir $BACKUP 2> /dev/null
mkdir ${BACKUP}/history 2> /dev/null
mkdir ${BACKUP}/last 2> /dev/null
cd ${BACKUP}/history
clear
echo "################ BACKUP `date` ###############"
```

```
echo
ls -rt *.log
echo
sleep 2
RETRY=TRUE
while [ "$RETRY" != "FALSE" ]
do
   echo
   echo '0) Comprobar contenido de una cinta'
   echo '1) salva_tmp (/tmp)  para pruebas)'
   echo '2) salva_root (menos  /mnt y /mount)'
   echo '3) salva_home  (/home)'
   echo
   echo 'Introduzca la opcion deseada.'
   read OPT1
   case "$OPT1" in
     [0123]) RETRY=FALSE
        ;;
     *) echo "Opcion invalida. Reintentar"
        ;;
   esac
done

if [ "$OPT1" != "0" ]
  then
   RETRY=TRUE
   while [ "$RETRY" != "FALSE" ]
   do
      echo 'Introduzca en primer lugar el tipo de
copia deseado'
      echo
      echo 'T) TOTAL'
      echo 'I) INCREMENTAL DESDE ULTIMO TOTAL'
      echo
      read TIPO
      case "$TIPO" in
        [TI]) RETRY=FALSE
         ;;
        *) echo "Opcion invalida. Reintentar"
         ;;
      esac
   done
fi

case "$OPT1" in
   0) testTape ;;
```

```
   1) salva_tmp ;;
   2) salva_root ;;
   3) salva_home ;;
esac
echo Copia finalizada.
```

Los menús de esta aplicación pueden hacerse más vistosos usando dialog(1), también puede intentar opciones para salvar y recuperar configuraciones. Ambas cosas están resueltas el 'kbackup' que está realizado igualmente en shell-script.

/ = = = = = = = = = = = = = = = = /

Ejemplos de recuperación de copias

En el ejemplo anterior no hemos incluido ninguna opción de recuperación porque cada caso es distinto dependiendo de lo que se quiera recuperar.

Un error en la recuperación puede suponer perdida de información. Siempre que sea posible conviene recuperar en un lugar distinto al original y solo se deben recuperar los ficheros necesarios.

Generalmente los programas de propósito general permiten la recuperación de todo el contenido o bien sacan una lista de todos los ficheros para señalar aquellos que deseamos recuperar.

Al contrario que el respaldo la recuperación no es una tarea rutinaria. Por esa razón y porque cada recuperación ha de hacerse bajo unos supuestos distintos los programas de propósito general no siempre proporcionan la flexibilidad necesaria.

Conociendo bien las herramientas sencillas tipo 'find', 'cpio', 'tar', 'afio', etc y combinándolas para recuperar solo lo que deseamos lograremos mucha flexibilidad.

Las distintas posibilidades son imposibles de comentar y ya vimos alguna cosa al explicar 'cpio' y en realidad basta con saber como sacar la lista de ficheros de una cinta y como poder usar esa lista para recuperar lo que queramos.

Ahora nos vamos a ver ejemplos concretos centrándonos exclusivamente en la recuperación de copias realizadas con 'afio' en formato comprimido, para no cambiar porque de alguna manera venimos mostrando nuestro favoritismo por 'afio' en toda esta lección.

Se puede confeccionar una lista de los ficheros contenidos en la copia haciendo lo siguiente.

```
$ afio -tZ /dev/st0 > listafich
```

Supongamos que en nuestra lista de ficheros aparecen los siguientes.

```
./var/lib/postgres/data/pg_variable
./var/log/apache/access.log
./var/log/apache/error.log
./var/log/auth.log
```

Si hace pruebas situese en /tmp. Para una recuperación de información real tampoco conviene recuperar directamente sobre el original así que /tmp puede ser igualmente un buen lugar si hay espacio para ello. Recuerde que algunos sistemas limpian /tmp al arrancar o regularmente cada cierto tiempo.

Continuando con nuestro ejemplo, para recuperar por ejemplo 'auth.log' haríamos lo siguiente.

```
$ afio -iZ -y '*auth.log' /dev/st0
```

Para recuperar los que ficheros que contengan 'log/apache' pero no 'access' haríamos lo siguiente.

```
$ afio -iZ -y '*log/apache*' -Y '*access*' /dev/st0
```

En este caso habríamos recuperado solo './var/log/apache/error.log'.

Para casos más complejos se usará las opciones -w o -W con ficheros que contendrán un nombre de fichero, o un patrón por cada línea, y donde cada una de ellas será considerada como si fuera una opción -y o -Y respectivamente para '-w fichero' o para '-W fichero'.

LA MEMORIA VIRTUAL EN LINUX

Introducción

Este curso no trata de la administración del sistema pero en este capítulo se van a explicar algunas cosas que son materia de administración del sistema pero que un usuario también debe conocer.

Muchas veces un administrador tiene dificultades para gestionar los escasos recursos de una máquina porque los usuarios no los utilizan del modo más racional.

En un sistema con muchos usuarios estos deben compartir los recursos de modo racional, pero generalmente ocurre lo contrario. El no comprender como funcionan las cosas y la avaricia de recursos puede conducir a conductas absurdas.

Por ejemplo un usuario que tiene que compilar 500 programas sabe que tendrá que esperar mucho. Si este usuario es suficientemente ignorante y egoísta puede pensar que lanzando las 500 compilaciones simultáneamente causará un retraso a los demás pero el terminará antes. Lo cierto es que los recursos del sistema se desperdician en trabajos internos totalmente inútiles para el usuario cuando el sistema está sobrecargado.

Tanto si se persigue compartir los recursos de forma razonable con los demás usuarios, como si lo que se pretende es usar los recursos de todo el sistema para sacar el mayor beneficio personal, se necesita comprender como funcionan algunas cosas.

Esta lección será totalmente teórica porque para un usuario normal no resulta prudente cierto tipo de experimentos que podrían provocar

incomodidades o incluso algunos problemas en el sistema. De todas formas si el sistema empieza a tener problemas, si que es bueno tener los conocimientos suficientes para averiguar si eso tiene que ver con algo que estemos haciendo nosotros. Quizás podamos incluso advertir a algún compañero que trabaje en el mismo equipo que el problema en el sistema tiene que ver con lo que el hace.

Limitar el uso de recursos

En un sistema se suele llamar recursos a ciertos elementos que no pueden ser utilizados de forma ilimitada. Por ejemplo. 'yes' es un programa que genera una salida infinita y no es buena idea redirigir su salida a un fichero porque llenaremos nuestro espacio de disco. En algunos sistemas el administrador establece un limite de espacio para cada usuario. Si no fuera el caso llenaríamos el disco. Si la salida de 'yes' la redirigimos a /dev/null no pasará nada. El número de ficheros que podemos tener abiertos simultáneamente también tiene un límite. Lo mismo puede decirse de otras muchas cosas.

La cantidad de memoria es limitada pero existen distintos limites que afectan al uso de la memoria. Para limitar el uso de ciertos recursos existe un comando interno de la shell llamado 'ulimit'.

Por ejemplo 'ulimit -c ' se usa para limitar el tamaño de los ficheros core producidos cuando un proceso muere bajo determinadas circunstancias. Un usuario normalito no utilizará para nada estos ficheros así que podría usarse 'ulimit 0' para evitar la aparición de ficheros core.

También se pueden limitar otras muchas cosas como la cantidad de CPU por segundo, tamaño máximo de un fichero, máximo número de procesos, máximo numero de ficheros abiertos, cantidad máxima de memoria virtual, etc...

Para ver las limitaciones con las que estamos trabajando haremos lo siguiente:

```
$ ulimit -a
core file size (blocks)     0
data seg size (kbytes)      unlimited
file size (blocks)          unlimited
max locked memory (kbytes)  unlimited
max memory size (kbytes)    unlimited
open files                  1024
pipe size (512 bytes)       8
stack size (kbytes)         8192
cpu time (seconds)          unlimited
max user processes          256
virtual memory (kbytes)     unlimited
```

Bueno en este ejemplo vemos que estamos en una sesión con muy pocas limitaciones. Eso es señal de que el administrador confía en nosotros y nos concede mucha libertad.

'ulimit' permite establecer dos clases de límites. Límites duros (solo se pueden poner una vez y generalmente esto lo hace root) y límites blandos que se pueden cambiar varias veces pero no se permite sobrepasar los valores establecidos en el límite duro. root establece estos límites en algún fichero de configuración y los usuarios pueden tener en su $HOME/.bashrc alterar los limites blandos. Para más información consulte la página man de 'ulimit'.

Multitarea

Ya hemos hablado algo de ella cuando estudiamos los procesos. Queda claro que en un sistema en el que se están ejecutando en apariencia varios procesos simultáneamente lo que ocurre realmente es que el tiempo de la CPU se va repartiendo entre los distintos procesos concediendo rápidamente sucesivas rodajas de tiempo a cada uno en función de su prioridad y de la política de reparto de ese tiempo. Esta política no nos interesa ahora. Se supone que las cosas están pensadas para que el funcionamiento global de todo el sistema sea el más adecuado.

Volvamos al caso del usuario ignorante y egoísta del capítulo anterior. El de las 500 compilaciones simultáneas. El sistema necesita parar cada uno de los procesos que usan la CPU una vez que han consumido su rodaja de tiempo para tomar el siguiente proceso en espera de CPU.

Cuando un proceso necesita usar algo y en ese instante no está disponible pasa a un estado de inactividad. Se dice que queda dormido.

Este cambio de un proceso a otro no es gratuito porque una vez parado el proceso hay que anotar la situación en la cual ese proceso ha sido detenido para luego poder volver a continuar exactamente en el mismo punto cuando le vuelva a tocar y quizás parte del código o de los datos del proceso ya no estén en memoria RAM (también la llamaremos indistintamente memoria física. Ver más adelante paginación). Todo esto consume muchos recursos y se puede alcanzar la situación en la cual el SO consume la mayor parte del tiempo en esta gestión de procesos intercambiando datos entre memoria y disco constantemente porque la memoria física no puede con todo por el contrario los procesos que son la parte realmente útil apenas dispondrán de recursos. En esta situación el sistema se vuelve muy lento y los discos del sistema tienen una actividad altísima. En realidad es una situación de auténtico agobio para el sistema y su comportamiento se vuelve torpe. Es como si el sistema actuara a la desesperada.

Por esa razón decimos que el usuario del capítulo anterior además de ser egoísta es ignorante porque está provocando un enorme retraso no solo a los demás usuarios (cosa que ya sabe) sino también a si mismo.

Si alguna vez notamos que el sistema va muy lento y sospechamos la causa tendremos que parar en primer lugar el proceso que está provocando la sobrecarga. Si no sabemos cual es o si se trata de un proceso que no es nuestro podemos investigar con el comando 'top'. Tenga en cuenta que de este comando se sale con 'q'.

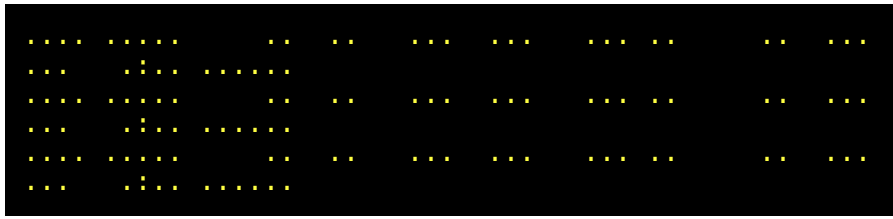
Este comando saca en cabecera un resumen de la situación general del sistema y luego saca solo la información de los procesos que mayor consumo de recursos están provocando en el sistema.

Tenga en cuenta que 'top' sirve para diagnosticar el uso de recursos y permite identificar procesos que consumen demasiado pero en si mismo 'top' también consume bastantes recursos. Por eso en situaciones de sobrecarga del sistema conviene no abusar de su uso. 'top' refresca cada cierto tiempo la información de la pantalla. Por defecto lo hace cada 5 segundos. Podemos aumentar a 15 segundos. Para ello usaremos 'top d 15'. Esto consumirá menos recursos y nos permitirá mirar con más tranquilidad la información de top.

Vamos a ver un ejemplo del uso de 'top' que corresponde a un sistema con muy poca actividad. Solo 5 usuarios. La CPU está ociosa 99.6% del tiempo, la cantidad de memoria disponible es alta y no se está usando la swap.

```
$ top
12:17pm  up  2:06,  5 users,  load average: 0.00,
0.00, 0.00
55 processes: 54 sleeping, 1 running, 0 zombie, 0
stopped
CPU states:  0.0% user,  0.3% system,  0.0% nice,
99.6% idle
Mem:  258100K av,  93320K used, 164780K free,  66800K
shrd,   7652K buff
Swap: 393552K av,      0K used, 393552K free
40744K cached

 PID USER     PRI  NI  SIZE  RSS SHARE STAT  LIB %CPU
%MEM   TIME COMMAND
1002 root      11   0  1312 1312   700 R       0  0.3
0.5   0:00 top
   1 root       0   0   464  464   404 S       0  0.0
0.1   0:05 init
   2 root       0   0     0    0     0 SW      0  0.0
0.0   0:00 kflushd
   3 root       0   0     0    0     0 SW      0  0.0
0.0   0:00 kupdate
```



USER es es usuario que arrancó el proceso, %CPU es el porcentaje de tiempo de CPU usado por el proceso. %MEM Porcentaje de memoria física usada por el proceso. Podría incluso superar el 100%. Un valor alto puede delatar a un proceso como el causante de los problemas de paginación (swaping) en el sistema.

Otro comando que nos da información sobre la carga del sistema es 'uptime'

```
$ uptime

  12:27pm  up  2:16,  5 users,  load average: 0.00,
0.00, 0.00
```

Coincide con la primera linea de 'top'. Los datos son hora actual tiempo transcurrido desde el arranque del sistema, número de usuarios y las tres últimas cifras se corresponden con promedios de carga en el último minuto, 5 últimos minutos y 15 últimos minutos respectivamente.

Memoria virtual

Vamos a explicar en primer lugar que significa eso de memoria virtual. En muchos sistemas operativos el SO ofrece a los programas un espacio de memoria superior al que realmente existe. La memoria física es la memoria RAM. La CPU no puede hacer nada con los datos si antes no pasan a la RAM. A pesar de esto un sistema con 64Mbytes de RAM puede ejecutar por ejemplo un programa use variables con el doble de tamaño que el de la memoria RAM disponible. Si el programa accediera mucho y de forma aleatoria dentro de esta enorme variable, se produciría

una fuerte sobrecarga en el sistema. Si alguien se pregunta si una barbaridad como esta, puede hacerla un usuario normalito sin privilegios especiales, la respuesta es, usuario normal si, pero autorizado.

Eso no significa que estemos burlando la seguridad del sistema ni que seamos más listos que el administrador.

Si el administrador configura el sistema de forma paranoica para evitar cualquier posible riesgo, los usuarios se verán sometidos a unas restricciones bastante incómodas para trabajar.

Por eso lo normal es que el administrador asuma que los usuarios autorizados utilizaran los recursos de forma razonable.

Para gestionar la memoria virtual lo que se hace es recurrir al espacio de disco para ampliar este espacio. La memoria RAM tiene una velocidad de acceso mucho más rápida que un disco duro y para conseguir que todo funcione de forma transparente y de forma eficiente se utilizan unas técnicas de segmentación y paginación que describiremos a continuación.

Segmentación de la memoria

La memoria de un ordenador es una especie de enorme casillero. Cada casilla contendrá cierta información pero un proceso no puede acceder a toda la información del sistema. Cada casilla tiene una dirección y cada proceso puede usar ciertas direcciones para almacenar información. De esta forma se evita que un proceso interfiera accidentalmente o intencionadamente a la información de otro proceso distinto.

Las casillas son las posiciones de memoria y a pesar de lo dicho existen posiciones de memoria compartidas. Por ejemplo el código de un programa que se está ejecutando en varios procesos será compartido en modo exclusivo de lectura por los procesos que estén ejecutando ese programa. Es decir que si por ejemplo hay varias personas ejecutando el editor 'vim' el código de ese programa no estará repetido en memoria. Sería un desperdicio porque el código es el mismo, no se puede alterar y

la CPU se limita a leer la información consistente en instrucciones para la CPU.

Todo este tipo de información es lo que se denomina segmento de texto. Los proceso manejan otros tipos de información distintas y para ello se organiza en segmentos de distinto tipo. Por ejemplo hay segmentos de datos de solo lectura, segmentos de datos de lectura escritura, segmentos de datos de pila, segmentos datos de memoria compartida, segmentos de memoria dinámica. No vamos a explicar estas cosas porque excede en mucho los propósitos de este capítulo pero si queremos dejar una idea. Los procesos tienen la memoria organizada en segmentos de distinto tipo. Gracias a esto se consigue utilizar la memoria RAM de forma muy eficiente.

Pese a que el código de un programa puede desaparecer de la RAM para dejar espacio a otros programas, no será necesario salvar la información a disco. Si fuera necesario volver a cargarlo en RAM bastará acudir al fichero que contiene su código ejecutable.

Esto hace que los ficheros que contienen programas que se están ejecutando no puedan ser borrados ni modificados por nadie, ni siquiera por root.

Copie el ejecutable 'top' en el directorio /tmp (Seguramente estará en 'bin/top' y sino deberiá ser capaz de encontrarlo). Ejecutelo como /tmp/top y desde otra sesión intente borrar el fichero. Sencillamente le resultará imposible pese a que es usted es dueño de esa copia.

Cuidado. Esto no es aplicable a los scripts de bash o a otros scripts porque en realidad estos programas son datos que se ejecutan dentro de un programa que los interpreta en lugar de hacerlo en la CPU.

Paginación

La información se organiza en bloques y los bloques que la CPU usa se mantienen el mayor tiempo posible en RAM. A estos bloques de información se les llama páginas.

Se intenta reducir el número de lecturas y escrituras a disco. Para ello se actúa en base a suposiciones de uso de la memoria. Las páginas recientemente utilizadas tienen mayor probabilidad de volver a ser usadas pronto que aquellas que ya hace tiempo que no han sido accedidas. Intentaremos explicar porque.

Los programas se comportan generalmente usando pequeñas partes de código durante la mayor parte del tiempo mientras que otras partes se ejecutarán de forma muy ocasional.

Además los programas usan partes comunes de código que pertenecen a librerías compartidas. El aprovechamiento de la RAM se consigue manteniendo una sola copia de ese código compartido en memoria RAM.

Los accesos a estas páginas de memoria que contienen código pueden serán para leer información. En el caso de datos de lectura escritura también podrán ser accedidas para su modificación. Cuando una página ha sido modifica se marca para escribir su contenido en disco en el caso de que se necesite el espacio que ocupa para otra página distinta.

El area de disco que se utiliza para ello se llama swap (también se denomina memoria de intercambio) y un usuario normal no puede modificarla aunque root si puede hacerlo para aumentarla o disminuirla de tamaño por ejemplo.

La memoria virtual es la suma de la memoria física (RAM) más el swap (area de intercambio) menos algunas páginas de reserva. Las paginas de reserva están limpias de información para poder cargar en ellas información nueva desde la swap cuando se necesita.

Al proceso de traer una página de la swap a memoria física se le llama (swap in). Esto necesita un tiempo y normalmente el proceso espera

dormido. Al proceso de volcar una página de memoria física a swap se le llama (swap out) y generalmente el proceso ya está dormido cuando esto ocurre.

Cuando se agota la memoria física (RAM) se empezará a usar la swap.

Cuando la memoria virtual se agota el kernel empezará a matar procesos. Dicho de otro modo un proceso que intenta usar memoria virtual cuando está agotada terminará recibiendo una señal del kernel que provocará su muerte.

Utilidades para monitorizar el uso de memoria virtual

Ya vimos el uso de 'top'. Nos sirvió para introducir el tema. Se trata de un comando realmente bueno para localizar los procesos responsables del mayor consumo de CPU y memoria.

La información del estado de núcleo del sistema se encuentra en /proc/ Ya comentamos en capítulos anteriores parte de su cometido y también resulta muy útil para obtener información sobre el uso de la memoria en el sistema.

```
$ cat /proc/meminfo
        total:    used:    free:  shared: buffers:
cached:
Mem:  264294400 103956480 160337920 58118144 10465280
53153792
Swap: 402997248        0 402997248
MemTotal:    258100 kB
MemFree:     156580 kB
MemShared:    56756 kB
Buffers:      10220 kB
Cached:       51908 kB
SwapTotal:   393552 kB
SwapFree:    393552 kB
```

También 'ps v' puede darnos información sobre el consumo de memoria virtual de procesos. Con top solo salía la información de los procesos que

más consumen pero quizás necesitemos información de determinados procesos. Por ejemplo para sacar la información de los procesos asociados al nuestro terminal bastará usar 'ps v'. Si necesita precisar otras condiciones consulte las opciones de ps en su página man.

```
$ ps v
  PID TTY      STAT   TIME  MAJFL   TRS   DRS  RSS
%MEM COMMAND
  301 tty3     S      0:00    600   420  1743 1300
0.5 -bash
 1640 tty3     S      0:00    326   518  1829 1568
0.6 vi recursyst.dat
 1665 tty3     S      0:00    194   420  1451  832
0.3 /bin/bash -c (ps v) >/tmp/vok2gkuv 2>&1
 1666 tty3     R      0:00    240    55  2844 1176
0.4 ps v
```

Si solo deseamos saber cuanta memoria queda libre podemos usar 'free'. Este comando también indica si se está usando la swap.

```
$ free
             total       used       free     shared
buffers     cached
Mem:        258100     101668     156432      56800
10248      51988
-/+ buffers/cache:      39432     218668
Swap:       393552          0     393552
```

Con 'vmstat' podemos tener información del uso de memoria virtual, cpu, y del consumo de entrada salida.

```
$ vmstat
   procs                  memory    swap
io     system         cpu
 r  b  w   swpd   free   buff  cache  si  so    bi
bo   in    cs  us  sy  id
 0  0  0      0 156404  10252  52000   0   0     2
1  141   132   1   0  98
```

Con 'memstat' podemos ver el uso de memoria virtual de los procesos y de las librerías compartidas.

```
$ memstat
     44k: PID     1 (/sbin/init)
     60k: PID    98 (/sbin/portmap)
     80k: PID   156 (/sbin/syslogd)
    468k: PID   158 (/sbin/klogd)
     40k: PID   165 (/sbin/kerneld)
     52k: PID   170 (/sbin/rpc.statd)
    140k: /lib/libreadline.so.4.1 228
     24k: /lib/libwrap.so.0.7.6 98 271
     72k: /lib/ld-2.1.3.so 1 98 156 158 165 170 178
179 180 181 182 188 193 ...
    864k: /lib/libc-2.1.3.so 1 98 156 158 165 170 178
179 180 181 182 188 19...
     20k: /lib/libcrypt-2.1.3.so 228 292 295 306 307
308 309 310 311 461 966...
     ...: ........................
     ...: ........................
     ...: ........................
--------
3768092k
```

Para mostrar solo la informacion del proceso 271 (en nuestro caso es sendmail) haríamos lo siguiente:

```
$ memstat | grep 271
    264k: PID   271 (/usr/sbin/sendmail)
     24k: /lib/libwrap.so.0.7.6 98 271
    232k: /lib/libdb-2.1.3.so 156 193 206 271 292 295
306 307 308 309 310 31...
     76k: /lib/libnsl-2.1.3.so 98 228 271 284 287 292
295 299 300 301 303 30...
     40k: /lib/libnss_compat-2.1.3.so 228 271 284 287
292 295 299 300 301 30...
     20k: /lib/libnss_db-2.1.3.so 156 193 206 271
     32k: /lib/libnss_files-2.1.3.so 156 170 193 206
271 292 295 306 307 308...
     48k: /lib/libresolv-2.1.3.so 271
    308k: /usr/sbin/sendmail 271
```

Con esto vemos lo bien aprovechado que está el sistema ya que muchas de las librerías aparecen compartidas con un montón de procesos que se están ejecutando.

Hemos conseguido ver las librerías compartidas que esta usando y cuanto ocupan en memoria pero la utilidad para saber usa o usará un ejecutable es 'ldd'. Este comando no necesita que el programa se esté ejecutando porque saca la información analizando el código del ejecutable.

```
$ ldd /usr/sbin/sendmail
        libdb.so.3 => /lib/libdb.so.3 (0x40019000)
        libwrap.so.0 => /lib/libwrap.so.0
(0x40054000)
        libnsl.so.1 => /lib/libnsl.so.1 (0x4005b000)
        libresolv.so.2 => /lib/libresolv.so.2
(0x40071000)
        libc.so.6 => /lib/libc.so.6 (0x40081000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2
(0x40000000)
```

Actuación con procesos conflictivos

Llegados a este punto conviene explicar que se puede hacer con un proceso conflictivo. Evidentemente la solución más drástica es matarlo. Para ello se usará primero un 'kill -15 PID'. La señal 15 es un SIGTERM que permite al proceso parar ordenadamente pero si el proceso ignora esta señal habrá que matarlo con un 'kill -9 PID'. La señal 9 es un SIGKILL que no puede ser ignorado.

De todas formas puede ocurrir que nos de pena matar a un proceso que se había comportado razonablemente durante bastante tiempo y que está llevando a cabo unas tareas que nos interesan.

Lo más suave que se puede hacer es bajarle la prioridad con 'renice' por ejemplo 'renice 20 PID'. Bajar la prioridad de un proceso no perjudica en nada al proceso cuando la CPU está ociosa por falta de trabajo. Si el proceso continua consumiendo demasiados recursos y no podemos bajar

más su prioridad podemos pararlo sin matarlo. Para ello se envía al proceso una señal SIGSTOP mediante 'kill -19' Para volver a arrancarlo podemos hacerlo mandando una señal SIGCONT 'kill -18'

LA PRIMERA INSTALACION DE LINUX

Introducción

No todos los usuarios que empiezan con Linux parten del mismo punto. Unos ya tienen conocimientos de Unix, otros no tienen conocimientos de Unix pero si de informática en general, otros disponen de algún amigo que les puede ayudar, y otros han leído cosas previamente. Nosotros no asumiremos nada de eso. Solo asumiremos que ya ha estudiado las lecciones anteriores.

Quizas se pregunte porque esta lección no se explicó antes. La razón es que durante la instalación de Linux se manejan una gran cantidad de conocimientos y en este momento ya disponemos de unos cuantos.

La instalación es un proceso que se ha simplificado bastante y las guías de instalación pueden ser de gran ayuda. A pesar de esto cuando se presenta una dificultad, muchas veces se requiere tener una base de conocimientos amplia para poder superar dichas dificultades.

He conocido a bastantes personas que deseosas de probar Linux me confesaban que no deseaban realmente perder nada de tiempo en aprender cosas sobre Linux. Se conformaban con aprender a instalar Linux y configurar las cuatro cosas que necesitan usar.

Siempre asumimos que ha leido las lecciones anteriores y por ello también asumiremos que este no es su caso.

Nosotros antes de llegar a este capítulo hemos realizado un amplio recorrido formativo. Conceptos como sistema de ficheros, núcleo del sistema, proceso, memoria virtual, etc.. ya han sido tratados. También sabe usar ya el editor vi, y ya tiene práctica con el interprete de comandos.

La principal utilidad de todo ello es que nunca se sentirá totalmente perdido.

De todas formas hay muchas cosas interesantes de cara a instalar Linux y que no han sido tratadas ni vamos a tratar en profundidad. No lo haremos porque desbordaría los propósitos de este curso. Por ejemplo unos conocimientos sobre hardware de PC's vendrían muy bien pero nos apartaríamos demasiado de los propósitos de este curso.

La facilidad o dificultad de la instalación de Linux puede variar desde ser algo trivial, o por el contrario puede resultar muy difícil dependiendo de las circunstancias. Debe tener esto muy presente y conformarse con objetivos sencillos la primera vez que intente instalar Linux.

Si el primer día consigue instalar un sistema capaz de arrancar y de abrir una sesión aunque sea en modo consola tendrá una buena base para ir añadiendo mejoras a su sistema poco a poco. No decimos que no se puedan hacer más cosas la primera vez pero no hay que obsesionarse con ello.

Cuando instale deberá tener a mano toda la documentación de su equipo y de sus periféricos. Quizás nunca leyó esos papeles, pero puede que ahora si los necesite.

No debe intentar avanzar demasiado deprisa. Debe de ir asimilando Linux poco a poco. Cuanto menores son los conocimientos técnicos más despacio hay que empezar porque cuesta más.

Si solo dispone de un ordenador lo mejor sería dejar una parte del disco duro para Windows y otra para Linux. De esa manera al arrancar podrá elegir con que SO quiere trabajar y no se verá forzado a hacerlo todo en un SO que inicialmente le resultará extraño y quizas incluso hostil.

Confiamos en que con el paso del tiempo Windows le resulte aún más extraño y muchísimo más hostil que Linux.

/ = = = = = = = = = = = = = = = /

Consideraciones hardware

Linux admite una amplísima variedad de hardware pero la autodetección de hardware está disponible principalmente en algunas distribuciones comerciales como SuSE, Mandrake y RedHat. La autodetección de hardware nunca es tan completa como en Windows. Ellos juegan con ventaja en esto.

Es importante comprobar que la tarjeta gráfica está soportada por la distribución que va a usar, en caso contrario es probable que solo consiga una resolución VGA. En las cajas de SuSE, RedHat o Mandrake viene una relación bastante amplia de tarjetas gráficas soportadas.

Para empezar resulta más sencillo usar un ordenador barato sin elementos de grandes prestaciones y a ser posible no demasiado nuevo.

Los procesadores compatibles con Intel 386 o Pentium están soportados aunque sean de otras marcas. Las placas principales y el tipo de memoria tampoco son críticos. En cuanto a los discos duros y controladoras que funcionen bajo bajo MSDOS suelen funcionar en su mayoría con Linux. Algunas controladoras SCSI no están soportadas y otras controladoras SCSI que si están soportadas pueden suponer una dificultad para una primera instalación debido a que pueden no estar incluidas en el núcleo (kernel) de arranque de la instalación.

/ = = = = = = = = = = = = = = = = /

Elección de su distribución

Puede que aun no tenga decidido que distribución usar. Si usted quiere la máxima facilidad de instalación quizás debería usar Mandrake, SuSE, o RedHat pero dado que esto forma parte de un curso de Linux nosotros le recomendamos Debian. Debian es la que tiene mayor interés educativo.

Quizás le cueste algo más empezar con ella pero también aprenderá mucho más.

Evidentemente hay otras consideraciones para le elección de su distribución de Linux. Compararlas en términos de mejor o peor no resulta adecuado. Nosotros no vamos a basar el curso en ninguna de ellas en particular. Pero vamos a hacer un poco de propaganda de Debian.

Al ser 100% libre es la más adecuada para aprender. Técnicamente es la distribución más robusta que existe. La actualización de los distintos componentes a traves de la red es mucho más cómoda y eficaz que en ninguna otra distribución. La comunidad de usuarios de Debian está muy bien organizada y dan muy buen soporte.

Debian no es para comodones. Hay que leer bastante y hay que molestarse en buscar las cosas. Gnome y Debian se llevan muy bien y Debian está evolucionando en el sentido de que cada vez resultará más intuitiva y facil de usar.

/ = = = = = = = = = = = = = = = = /

Utilidades MSDOS

En muchas distribuciones encontrará unos programitas en su CD de Linux que son herramientas para usar desde MSDOS. Son software libre y pueden venir muy bien para determinadas circunstancias.

fips20.zip Se usa para dividir una partición en dos sin perder los los datos. Se usa mucho para crear un hueco en un disco para poder instalar en el Linux. Esta versión puede trabajar con FAT32.

gzip124.exe Se trata de una versión del popular compresor 'gzip' de Linux para msdos

lodlin16.zip Permite arrancar Linux desde msdos. Se puede usar desde disquete o desde disco duro.

rawrite1.zip y rawrite2.zip. Se corresponden con rawrite versión 1.3 y con rawrite 2.0. Permiten crear un disquete desde msdos a partir de un fichero que contenga la imagen de ese disquete. Algunas distribuciones incluyen una variedad de imágenes de disquetes de arranques cada uno pensado para poder arrancar en distintos tipos de máquinas pero lógicamente para crear el disquete de arranque en una máquina donde no conseguíamos arrancar Linux necesitamos que este programa funcione bajo msdos. Se proporcionan a menudo estas dos versiones por si bien la versión 2.0 es más rápida puede bloquearse en algunos equipos.

unz512x3.exe Estamos poniendo ficheros con extensión .zip. Son ficheros comprimidos en un formato muy utilizado en msdos.

/ = = = = = = = = = = = = = = = = /

Programas de utilidad para instalar Linux

Las distribuciones suelen usar un sistema de menús y unos asistentes que le ayudan a instalar y configurar el sistema. Muchos de estos procesos podrían realizarse manualmente desde el interprete de comandos ya que se limitan a usar unos programas que están instalados en cualquier Linux.

Ahora nos limitaremos a señalar su existencia y comentar para que sirven. Conviene que consulte las páginas del manual de cada uno de ellos.

fdisk(1): Se usa para particionar. Advierta que en msdos existe un programa que también se llama fdisk y que sirve igualmente para particionar el disco.

mkswap(1): Para formatear una partición de swap

swapon(1): Para activar la partición de swap

mke2fs(1): Para formatear una partición Linux

mount(1): Para montar el sistema de ficheros El concepto de montaje de ficheros lo vimos muy de pasada en el tercer capítulo dedicado al sistema de ficheros y hemos practicado poco con él porque requería uso de la cuenta de root en un momento que no parecía prudente. Repase este capítulo si ya no se acuerda. Puede consultar en las páginas man los comandos mount(1), umount(1).

También es muy interesante que mire el contenido de su /etc/fstab. Es un fichero que indica al kernel que es lo que debe montar durante el arranque. Consulte la página man correspondiente fstab(5). Si edita este fichero puede especificar nuevos puntos de montaje pero un error podría hacer que el sistema no vuelva a arrancar.

Siempre que modifique algo del sistema, deje una copia del fichero sin modificar en algún lado.

Por ejemplo puede dejar copias originales ficheros terminadas en '.seg'. De esa forma si algo deja de funcionar luego puede recuperar exactamente la situación original. Lo hemos dicho muchas veces. Asegurese en todo momento la posibilidad de deshacer cualquier cambio.

Conociendo todos estos programas tendrá una idea bastante buena de las cosas que van sucediendo durante el proceso de instalación aunque los asistentes de instalación de muchas distribuciones los usarán de modo transparente (de modo no perceptible) mientras instalan.

/ = = = = = = = = = = = = = = = = = /

Disco de rescate

Se llaman así a los CDs o disquetes que pueden arrancar Linux directamente y dejan un sistema funcionando con una serie de utilidades para poder subsanar problemas en equipos que por ejemplo han perdido la

capacidad de arrancar o que no deseamos que arranquen sin subsanar antes algún problema.

Muchos de los primeros CDs o disquetes de instalación de muchas distribuciones pueden ser usados como discos de rescate en lugar de disco de instalación.

Los discos de rescate son en si mismos un SO Linux en miniatura y como es lógico arrancan un kernel predeterminado y montan un sistema de ficheros raíz.

Dado que se necesita permiso de escritura y no podemos usar ningún disco duro para ello porque podría no existir espacio adecuado para ello, lo que se hace es montar el sistema de ficheros raíz en un dispositivo RAM. De esta forma tenemos un sistema miniatura Linux con mínimos requisitos de hardware.

Lo que ha de hacerse muchas veces para arreglar problemas es montar las particiones del disco duro donde se encuentra el fichero que deseamos modificar y generalmente con un simple editor se corrige el problema. Por ejemplo si se olvida uno de la password de root se edita el fichero /etc/passwd eliminando el campo de la password, y de esa forma no pedirá password. Es solo una forma de hacerlo y lo comentamos solo como ejemplo de uso de un sistema de rescate.

También comentamos antes que un cambio desafortunado en /etc/fstab podía hacer que el sistema no volviera a arrancar. Sugeriamos entonces que dejara una copia del fichero original en algún lado. En este caso después de montar la partición raiz del sistema que no arranca deberá sobreescribir el fichero modificado con la copia original de seguridad. Después de cada cambio hay que acordarse siempre de desmontar el sistema de ficheros.

Si no dispone todavía de un disquete de rescate y tampoco dispone de un CD de rescate, debería crear uno cuanto antes. Nunca se sabe cuando va a ser necesario recurrir a el.

Para ello localize en los CDs de su distribución la imagen de uno de ellos que se llamará seguramente 'rescue.img', 'rescue.bin', 'rescue.raw', o algo parecido. Puede hacerlo desde MSDOS usando rawrite, o desde linux usando dd.

Antes deberá montar la unidad de cdrom. Generalmente se suele montar en '/cdrom/', en '/mnt/cdrom/' o en '/mount/cdrom/'. Viene determinado en /etc/fstab. Por ello para montarlo bastará hacer:

```
$ mount /dev/cdrom
$ df
```

Para Redhat sería:

```
$ dd if=/mnt/cdrom/images/rescue.img of=/dev/fd0
bs=1440k
```

Para Debian potato sería:

```
$ if=/mnt/cdrom/dists/potato/main/disks-
i386/2.2.16-2000-07-14/images-2.88/rescue.bin of=/dev/
fd0 bs=1440k
```

/ = = = = = = = = = = = = = = = = /

Lilo

Lilo es un cargador que puede ser instalado en disquete o en disco duro. En disco duro puede ser instalado en una partición o en en el MBR (Master Boot Record). El MBR es una zona especial del disco muy pequeña reservada para el arranque.

Se puede especificar arranques alternativos para distintos SO como Windows98, Linux, etc... Para Linux podemos especificar arranques con distintos núcleos, y arranques que monten el sistema de ficheros raíz en

distintas particiones siempre que efectivamente obtengan un sistema de ficheros raíz válido. Por ello se podría especificar arranques para distintas distribuciones de Linux instaladas en distintos sitios.

Durante el arranque se espera un tiempo para que el usuario pueda especificar el arranque deseado y pasado ese tiempo se arranca la opción de arranque que figure por defecto.

Lilo ofrece muchas posibilidades. Los asistentes de instalación lo que hacen es crear un fichero de configuración de Lilo /etc/lilo.conf y luego ejecutan Lilo.

Si decide intentar alguna modificación manual de este fichero de configuración deberá ejecutar después Lilo. Si olvida hacerlo la próxima vez el sistema no arrancará. Lo mismo pasará si toca cualquiera de los elementos que intervienen en el arranque.

Piense que Lilo se instala en un lugar muy pequeño en el cual se sitúan apuntadores a una serie de ficheros como el kernel, el /etc/lilo.conf , etc. Si alguno de estos ficheros se mueve de sitio o se modifica su contenido, los apuntadores de Lilo quedarán señalando un lugar equivocado donde ya no existe la información útil.

Ejecutando Lilo se vuelve a reconstruir todo y se actualizan esos apuntadores. Esos apuntadores o punteros son datos numéricos que expresan posiciones físicas en el disco en forma de cilindro, sector, cabeza, y dispositivo.

Precisamente estos apuntadores tienen limitaciones en muchas BIOS y si la partición de arranque de Linux no está contenida íntegramente dentro de los primeros 1023 cilindros de su disco duro el sistema puede no arrancar.

Linux se puede instalar en el MBR. Si desea volver a dejar el arranque original de msdos que estaba situado en el MBR puede arrancar msdos y hacer 'FDISK /MBR'.

Linux también se puede instalar en una partición del disco duro. Si por error instala Lilo en una partición donde tenían Windows posiblemente se pierda la información de los ficheros del sistema de Windows y deberá transferir el sistema nuevamente arrancando desde un disquete msdos y tecleando 'SYS C:' siendo C: la unidad que desea recuperar.

/ = = = = = = = = = = = = = = = /

Loadlin

Loadlin es un ejecutable MSDOS capaz de cargar Linux.

Conviene conocer esta otra forma de arrancar. Es una alternativa al uso de Lilo. Para ilustrar su uso pondremos como ejemplo la creación de un disquete de arranque con Loadlin.

1. Formatee un disquete desde msdos o Windows incluyendo los ficheros del sistema. (FORMAT /S).
2. Copie el ejecutable LOADLIN.EXE y un kernel en este disquete. Por ejemplo vamos a suponer que dicho kernel lo metemos en un fichero llamado 'kr3'.
3. Supongamos que nuestro sistema de ficheros raíz está situado en '/dev/hda9'.
4. Incluya un fichero AUTOEXEC.BAT que contenga lo siguiente: loadlin kr3 root=/dev/hda9 ro vga=3

Bueno simplemente con esto tiene un disquete de arranque que no usará Lilo sino Loadlin. Su sistema de ficheros podrá estar en cualquier parte del disco duro sin la limitación de los 1023 cilindros.

/ = = = = = = = = = = = = = = = /

La Swap

Una de las cosas que tendrá que determinar es la cantidad de swap. Ya hemos visto en la lección anterior el concepto de memoria virtual. La swap es la parte del disco destinada al intercambio de páginas de memoria de RAM.

Generalmente se suele dimensionar la swap con el doble de tamaño que la cantidad de RAM disponible. De esta forma la memoria virtual funcionará sin detrimento de velocidad. En el caso de que el total de memoria resultante RAM + SWAP no alcance para poder ejecutar alguna de las aplicaciones necesarias se podrá sobre dimensionar la SWAP a costa de reducir un poco su rendimiento.

En el caso de tener varios discos, conviene repartir a partes iguales la swap entre ellos para que todos los discos colaboren por igual al proceso de paginación. Esto no es imprescindible pero este proceso es uno de los que más repercute en la capacidad de trabajo de todo su sistema.

La partición de swap será la parte de su sistema que tenga mayor número de accesos y por ello tampoco conviene ponerla en un extremo del disco alejada de los sistemas de ficheros en uso. Esdto obligaría a hacer desplazamientos mayores a las cabezas del disco y se perdería velocidad.

En realidad todo esto son cosas que no tienen excesiva importancia en un ordenador que no va a tener demasiada carga. En su primera instalación no debe preocuparle demasiado estas cosas. Ya dijimos que no pretendemos hacer un curso de administración sino un curso de autosuficiencia para usuarios de estaciones de trabajo. Para equipos con muchos usuarios o grandes servidores si será conveniente afinar lo que se pueda a la hora de establecer la cantidad y la localización de la swap.

/ = = = = = = = = = = = = = = = /

Particionado

Es posible que su distribución le facilite la labor de particionado del disco duro pero no está de más un poco de información sobre esto.

Cada tipo de dispositivo usa un controlador que en el caso de los discos duros son '/dev/hda' para el primer disco IDE '/dev/hdb' para el segundo disco IDE, '/dev/sda' para el primer disco SCSI, etc..

Concretamente los discos IDE serán de la forma '/dev/hd[a-h]', los discos SCSI serán '/dev/sd[a-p] ', los ESDI serán '/dev/ed[a-d]', para los XT serán '/dev/xd[ab]'.

Las particiones se usan añadiendo el número de la partición al dispositivo. Por ejemplo '/dev/hda1/' será la primera partición del primer disco IDE.

Como máximo se podrán tener cuatro particiones primarias en un mismo disco. Si fueran necesarias más, habría que usar una partición de tipo extendida. Sobre esta se pueden establecer varias particiones lógicas.

La primera partición lógica será la número 5 porque los cuatro primeros números de partición se reservan para su posible uso en particiones primarias. Tampoco se puede usar más de una partición primaria de tipo extendido aunque puede ser cualquiera de la 1 a la 4.

Por ejemplo podríamos tener en una primera partición primaria un sistema operativo Windows y una segunda partición primaria de tipo extendido, particionada a su vez en 5 partes. De ellas una deberá ser para montar el sistema de ficheros raíz (root, '/') que conviene sea de tipo Linux y otra para el área de intercambio (swap).

El resto de las particiones se pueden usar para lo que guste. Por ejemplo para montar '/usr', '/var', '/boot', '/tmp' o '/home', etc. Estos cinco directorios son buenos candidatos para ser separados en particiones distintas, pero dar unas indicaciones generales para determinar el espacio necesario resulta complicado ya que depende del tipo de uso que tenga el sistema.

Por el contrario también se puede optar por montar todo en una sola partición que será la partición raíz. Por su sencillez debería considerar hacerlo así en su primera instalación. No es lo ideal pero tampoco hay que ser ambiciosos la primera vez.

Por otra parte con una sola partición se aprovecha mejor el espacio cuando tenemos poco disco. La principal desventaja es que la partición raíz quedará muy grande y eso aumenta el riesgo de problemas en una partición que conviene sea lo más segura posible.

Para minimizar este efecto es para lo que se usa el directorio /boot que se destina a contener únicamente algunos de los elementos más importantes que intervienen durante el arranque como por ejemplo el kernel. Idealmente sería una partición muy pequeña con unos 20MB que no crecerá nunca y que hará menos probable que que un error en el sistema de ficheros raíz afecte al arranque del sistema, lo cual supondría un mayor esfuerzo para recuperar el sistema.

El tema del particionado se puede dejar en manos de los distintos asistentes que usan algunas distribuciones los cuales ya sugieren las particiones y los tamaños más o menos adecuados en función del tipo de instalación, pero es mejor tener las ideas claras y partir ya con una situación adecuada antes de meter un disquete o CD de instalación de Linux. Nos referimos a dejar libre los espacios adecuados en su disco duro, y repartidos en las particiones que pensemos usar antes de empezar a instalar Linux.

Vamos a resumir los casos más frecuentes simplemente para que tenga una idea general.

- **Instalar Linux en un PC con un solo disco duro entero para Linux.** Es lo más sencillo. Puede empezar con todo el disco libre y sin particionar.

- **Instalar Linux en un segundo disco duro entero para Linux dejando el primer disco para Windows.** Es tan sencillo como el caso anterior aunque deberá identificar los dispositivos.

  La nomencatura de los discos es las siguiente:

  - hda primer canal maestro.
  - hdb primer canal esclavo.
  - hdc segundo canal maestro.
  - hdd segundo canal esclavo.

  Si no sabe como está configurado su hardware. Puede arrancar un disco de rescate Linux y desde una consola teclear 'fdisk -l' Con ello obtendrá un listado completo de los discos y particiones presentes en su sistema.

- **Instalar Linux en un disco duro que ya tiene instalado Windows.** Entre en Windows. Desactive cualquier tipo de antivirus porque suelen instalarse al final del disco y no se puede mover compactando. Compacte el sistema de ficheros con la opción de máxima compresión. Esto dejará todos los ficheros en la primera parte del disco duro y el resto queda libre. Obtenga el programa fips para Windows desde algún CD de Linux. Es una utilidad que divide una partición existente por donde indiquemos. Antes de usarlo conviene salvar el contenido del disco por si las moscas. Use una versión moderna de fips. Lea con atención las indicaciones. Cuando termine tendrá una partición con Windows del tamaño que usted indique y otra para Linux. Entre en el fdisk de Windows y borre la partición donde desee instalar Linux y ya puede proceder a instalar Linux.
- **Instalar Windows y Linux en un PC que no tiene nada instalado.** Haga una partición para Windows dejando espacio libre para Linux. Instale primero Windows y luego continúe instalando Linux.

/ = = = = = = = = = = = = = = = = /

Compatibilidad del hardware con Linux

Generalmente para instalar todas las distribuciones incluyen uno o más núcleos para arrancar generalmente bastante grandes y muy completos para poder soportar una variedad de hardware suficientemente amplia para soportar una gran variedad de dispositivos hardware y así no dar problemas en la instalación.

No obstante la extraordinaria variabilidad de componentes hardware para PC disponibles en el mercado hacen imposible contemplar todas las situaciones posibles. El soporte de estos drivers por el momento no suele ser facilitado por los fabricantes tal como ocurre en Windows sino que es producto del trabajo desinteresado de mucha gente que colabora desarrollando estos drivers para Linux muchas veces con la gran dificultad de no disponer información del fabricante.

Prácticamente se ven obligados a una labor de investigación muy costosa. Todo eso explica que los dispositivos más sofisticados y los más modernos sean los que tienen peor soporte en Linux. Esta situación está cambiando y ya hay algunos fabricantes que empiezan a dar la información necesaria para desarrollo de drivers no desarrollados por ellos generalmente para Windows.

Si algún elemento de su hardware no funciona con Linux puede deberse a que está mal configurado y cambiar de distribución podría no servir de gran cosa. Cuando algo así ocurra piense que esto le ofrece la oportunidad de aprender cosas nuevas.

Tendrá que investigar el tema y cuando consiga dar con la solución posiblemente habrá aprendido una serie de cosas de gran interés para una variedad de situaciones distintas. Los primeros pasos son siempre los más costosos.

Linux no está siendo usado exclusivamente por gurús y hackers . Linux está siendo usado por una cantidad enorme de personas que han compensado su carencia de conocimientos técnicos con tan solo un poquito de paciencia. A diferencia de Windows no hay secretos. Toda la información que usted pueda llegar a necesitar está a su alcance. Cada distribución viene con su manual de instalación.

En el caso de Debian y otras ese manual viene en formato electrónico pero conviene sacarlo a papel y dedicar algo de tiempo a su estudio. También suele venir en ese manual o en la misma caja de la distribución una relación del hardware soportado. Piense que el soporte para el hardware es casi siempre independiente de la distribución y solo depende de la versión del kernel que se esté usando. En el caso de Debian los primeros capítulos son de interés general con independencia de la distribución que use y también está disponible gratuitamente.

1. Que es Debian (y que es GNU/Linux)
2. Requerimientos del sistema
3. Antes de empezar
4. Particionado del disco duro

También puede consultar el manual de Linux Primeros pasos. Que puede obtener desde LuCAS, lugar que anteriormente recomendamos.

/ = = = = = = = = = = = = = = = = = /

El Kernel

Para que Linux reconozca todos los dispositivos del sistema deberá estar compilado con las opciones adecuadas. En el caso de Linux el kernel es un gran programa y lo que llamamos drivers solo son trozos de código de este gran programa. Este diseño se dice que es un kernel monolítico.

Linux permite que partes de este gran programa no estén permanentemente en memoria RAM. Estas partes son los módulos y se

corresponden con drivers que se cargan en el momento que el kernel necesita su uso. Esta carga y descarga es muy rápida y por ello dispositivos que no se usan continuamente como, impresoras, lectoras de CDs, disqueteras etc.. convienen estén configuradas como módulos.

Para configurar todo ha de compilarse el núcleo en un sistema Linux que ya esté funcionando y nosotros no vamos a tratarlo ahora. Si un dispositivo no funciona habrá que comprobar en primer lugar si ha sido correctamente reconocido por el kernel. Esto ocurre durante el arranque y el kernel va mostrando la información de todos los dispositivos presentes en el sistema que es capaz de reconocer. Para volver a consultar esta información después de arrancar se puede usar el comando 'dmesg | less'. Nos ocuparemos de la compilación del kernel en otro capítulo.

/ = = = = = = = = = = = = = = = = = /

Consejos finales

No resulta razonable empezar instalando una gran cantidad de cosas distintas. Empiece instalando los paquetes imprescindibles. Luego podrá ir añadiendo otras cosas, poco a poco y sacando copias de seguridad antes de añadir o quitar paquetes. Piense que unos paquetes pueden depender de otros o entrar en conflictos con otros y quizás no se de cuenta de que está modificando algo que necesita.

Una vez obtenida una primera instalación que funcione aunque falten cosas por configurar deberá avanzar despacio pero seguro. Es decir recurra a las copias de seguridad estableciendo puntos a partir de los cuales podrá recuperar la situación para no tener que empezar cada vez desde cero.

También es bueno ir apuntando las cosas que se van instalando y configurando la resolución de problemas, etc.. Todo ello para facilitar las cosas en caso de tener que volver a empezar, de lo contrario el esfuerzo en configurar su sistema se puede perder.

La primera vez que instale Linux no debe pensar en ello como un mero trámite para empezar a hacer cosas. Piense que es un proceso fraccionado en distintas etapas y que cada etapa tiene como premio el poder disfrutar de un sistema capaz de hacer cada vez más y más cosas.

/ = = = = = = = = = = = = = = = = = /

Documentación adicional

Este documento fué pensado para ayudar a los usuarios con menos conocimientos y menos recursos técnicos a tener éxito en su primera instalación de Linux. El tema es amplísimo y hay mucha documentación al respecto. Por ello nos hemos limitado a hacer unas consideraciones generales. No obstante hay una gran cantidad de documentación de gran utilidad para temas de instalación y configuración de Linux.

Para obtener dicha información le recomendamos que visite. Las páginas de LuCAS con gran cantidad de documentación en español o las páginas de LDP (Linux Documentatión Project) destinadas a centralizar toda la documentación de Linux en todo el mundo.

En el caso de que tenga una dificultad específica con algún dispositivo o con alguna configuración concreta de algún susbsitema puede encontrar gran ayuda en una serie de documentos cortos monográficos de enfoque práctico denominados "Howtos" o "Comos" que podrá encontrar en los lugares que acabamos de indicarle.

Para algunos casos conflictivos relativos al hardware de su equipo puede consultar la documentación de su kernel. '/usr/src/linux/Documentation/'

/ = = = = = = = = = = = = = = = = = /

Asistencia gratuita en listas de usuarios

Existen listas de correo en las cuales podrá obtener respuesta a sus preguntas sobre Linux. Realmente funciona en la práctica como un servicio de asistencia gratuito de gran calidad.

No debe tener reparo a preguntar siempre que comprenda que hay que respetar las normas de la lista y que el tiempo de los demás es valioso. Por ello no debería nunca preguntar cosas sin consultar antes las páginas del manual y otra documentación adicional que acabamos de mencionar.

Se asume que no va a preguntar por simple comodidad, sino porque ha buscado y no a encontrado la respuesta.

En España la lista de consultas técnicas sobre Linux en general es l-linux. Para suscribirse o desuscribirse se enviará un correo electrónico con un comando que será recibido por un servidor pero no será leído por ninguna persona sino interpretado por un programa que en este caso se llama Majordomo.

```
$ # Para suscribirse
$ echo subscribe l-linux       | mail
majordomo@calvo.teleco.ulpgc.es

$ #
$ # Para borrarse
$ echo unsubscribe l-linux     | mail
majordomo@calvo.teleco.ulpgc.es
çFIM




Recibirá una contestación automática y con ello
recibirá todo el

correo de la lista. Cada vez que alguien envía un
correo a la lista
```

todos los miembros recibirán una copia del mensaje.

La lista l-linux se modificó hace poco y desde entonces hay que estar

suscrito para poder enviar mensajes. Esto se hizo para combatir en lo

posible el uso de SPAM y el envío de mensajes anónimos molestos.

Desde entonces la calidad de la lista ha mejorado bastante.

Otras listas pueden tener un

funcionamiento más abierto admitiendo cualquier mensaje, y otras por

el contrario pueden tener un comportamiento más restrictivo.

Concretamente algunas

se llaman listas moderadas y solo se admite un mensaje si una persona

llamada moderador considera que es un mensaje oportuno.

Conviene leer atentamente las normas de uso de la lista y atenerse

a ello. Hay gente que no respetan las normas. Incluso hay algunos

que envían un mensaje admitiendo que está fuera de la temática de

la lista en tono de disculpa anticipada, pero estas cosas se suelen

considerar abusos y pueden provocar fuertes discusiones en la lista.

Concretamente l-linux es una lista que recibe y envía muchos mensajes.

Hay otras listas específicas para otras distribuciones o para temas

mucho más especializados.

TERMINALES

Que es un terminal

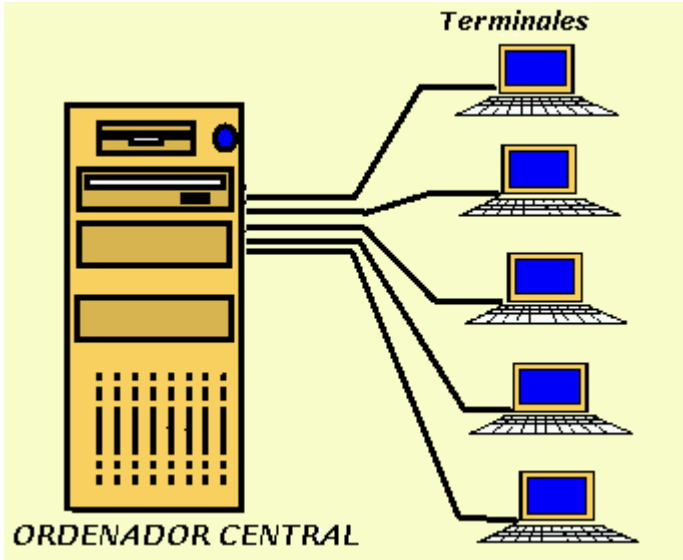
En Linux y otros SO similares se considera la posibilidad de que un ordenador tenga varios usuarios trabajando simultáneamente.

Cada dispositivo que permite a un usuario interactuar con una máquina se le llama terminal. Hay terminales de entrada como por ejemplo un lector de código de barras, los ya obsoletos lectores de tarjetas perforadas, terminales salida como por ejemplo una impresora, y terminales de entrada salida como los clásicos terminales de pantalla.

Hay otras muchas clases de terminales. Por ejemplo hay terminales para personas ciegas, pero el tipo más utilizado es el formado por pantalla y teclado. Dentro de esta categoría hay cientos de modelos distintos. En Linux y en otros SO tipo Unix se contempla la posibilidad de trabajar casi con cualquier tipo de terminal.

La palabra tty usada para designar los dispositivos que controlan los terminales viene de TeleTYpe (teletipo). Los primeros terminales no tenían monitor. Eran simples y primitivos teletipos.

En un PC la pantalla y el teclado son periféricos que se conectan de forma independiente al ordenador, pero en ordenadores grandes los terminales de pantalla y teclado son un todo que se comunica con el ordenador por un único cable. Por ejemplo a través de un puerto serie. Por estos cables lo que circulan bytes correspondientes a los caracteres intercambiados entre el terminal y el ordenador. Antes de que surgieran los ordenadores personales lo normal era que el ordenador estuviera en un cuarto cercano a la sala de terminales donde los usuarios se sentaban a trabajar.



Podemos ver en la figura un conjunto formado por oordenador central y una serie de ordenadores conectados a él.

Estos terminales que acabamos de mencionar aun se usan hoy en día y no son simples conjuntos de teclado y monitor. Tienen cierto grado de autonomía lo cual permite liberar al ordenador central de una parte del trabajo ya que en lugar de atender las entradas tecla a tecla, lo hará línea a línea. La pantalla de un terminal no es simplemente un monitor ya que su electrónica es más compleja. Entre sus circuitos electrónicos dispone de una memoria. Esta guarda la información hasta tener la línea completa que además puede ser editada sin que en eso intervenga el ordenador. La pantalla en cambio muestra los caracteres tecleados a medida que se van introduciendo. Por eso lo que vemos en el terminal quizás no ha sido transmitido aún al ordenador. En realidad la memoria del terminal contendrá no solo el buffer del teclado sino todo el contenido de la pantalla del terminal. Esto es necesario por varias razones. Una de ellas es la posibilidad de hacer scroll vertical y otra para recuperar la información de toda la pantalla en caso necesario.

Terminales virtuales

Si ha pasado lo anterior por alto pensando que no tiene que ver con su caso por trabajar con PCs, vuelva a leer los párrafos anteriores porque en un PC que use Linux una sesión de consola como las que estamos usando

en este curso, funcionará de forma muy similar por ser una simulación de este tipo de terminales.

En Linux existe una emulación de varios terminales sobre un único monitor. Se denominan frecuentemente terminales virtuales. Para cambiar de un terminal virtual a otro pulsaremos <Alt><F1>, <Alt><F2>, <Alt><F3>, etc. Con ello tendremos acceso a los terminales /dev/tty1, /dev/tty2, /dev/tty3, etc respectivamente. En cada uno de estos terminales puede mantenerse una sesión de trabajo distinta.

Es como si tuviéramos varios terminales físicos distintos formados por monitor, teclado, altavoz, memoria, y cable de conexión al ordenador, pero solo pudiéramos usar uno de ellos en un momento dado. La información de cada terminal virtual se encuentra almacenada en dispositivos /dev/vcs pero no es accesible para usuarios normalitos.

Dispositivos controladores de terminales

Cada terminal virtual está gestionado por un controlador de dispositivo. Si ejecuta el comando 'tty' obtendrá el nombre del controlador que está usando.

```
$ tty
 /dev/tty3
```

Bueno /dev/tty3 sería el nombre del terminal. Usted puede obtener otra cosa pero vamos a suponer que su terminal fuera este.

```
$ echo "hola" > /dev/tty3
```

Sacará el mensaje "hola" en la pantalla de su terminal. Si intenta hacer esto mismo sobre otro dispositivo, por ejemplo /dev/tty2 debería aparecer el mensaje en la pantalla de otro terminal. Normalmente esto no sucede a

no ser que /dev/tty2 también esté funcionando con su usuario. Con esto se evita que un usuario pueda molestar a otro.

Los terminales virtuales están numerados del 0 al 63. Para ver cual es el número del terminal actual también se puede usar 'fgconsole'

Hay otros tipos de terminales. Son dispositivos de tipo carácter y hay un montón de ellos disponibles.

```
$ ls /dev/*tty*
```

Existe un dispositivo especial que para cada usuario se comporta como si fuera su propio controlador.

```
$ echo "hola" > /dev/tty
```

Esto saca un mensaje en la pantalla de su propio terminal exactamente igual que el el caso anterior. Esto puede ser útil para usarlo por ejemplo en scripts que pueden tener ambas salidas (estándar y de errores) redirigidas a ficheros pero deseamos que determinado mensaje salga siempre por la pantalla.

Existe otro dispositivo especial llamado consola /dev/console. Este dispositivo suele encontrarse próximo a la máquina y por el salen los mensajes del sistema. Se asume que este puesto de trabajo es usado por el administrador del sistema.

La consola está asociada a /dev/tty0 que al igual que /dev/tty es un alias de terminal virtual actual pero pueden tener permisos distintos.

Vamos a hacer algunas pruebas y vamos a suponer que su sesión de trabajo con bash utiliza el tty3. Compruebe cual es su caso con el comando 'tty' y en caso de ser distinta téngalo en cuenta cada vez que hagamos referencia al tty3 o a <Alt><F3>. Usted deberá usar lo que

corresponda en su caso. También puede pasarse al tty3 para continuar la lección.

Primero vamos a comprobar la disponibilidad y los permisos de un tty que casi siempre queda libre. El tty12.

```
$ ls -l /dev/tty12
 crw-rw-rw-  1 root   tty      4, 12 ago 29  2000
/dev/tty12
```

Aquí lo importante es comprobar que podemos usar ese terminal y que tiene permisos de lectura escritura.

Si esto no es así habría que modificar los permisos o incluso crear el dispositivo para poder hacer la práctica pero eso requiere privilegios de root y nos saldríamos de los propósitos de este curso.

Vamos a probar si podemos escribir en el tty12.

Si usted hace 'echo xxxxx > /dev/tty12' y luego pasa al terminal tty12 con <Alt><F12> verá que no puede hacer nada en él pero las 'xxxxx' aparecen en la pantalla.

Vamos a probar si podemos leer del tty12.

Haga 'cat < /dev/tty12' desde su sesión de trabajo y luego pase al terminal tty12 con <Alt><F12> e introduzca una cadena sin usar la tecla <Intro>. Por ejemplo 'yyyyyyyyyyy'.

Si vuelve al tty3 con <Alt><F3> comprobará que el comando cat continua esperando recibir datos pero aun no ha recibido ningún carácter. La razón es que la cadena 'yyyyyyyyyyy' se encuentra todavía en la memoria del terminal tty12 y todavía no se ha enviado un solo carácter al ordenador.

El modo normal de funcionamiento de los terminales de texto es este. En lugar de enviar caracteres aislados envían chorros de caracteres.

Vuela al tty12 con <Alt><F12> y pulse <Intro>. Inmediatamente vuelva a tty3 con <Alt><F3> y comprobará que cat acaba de recibir la cadena de caracteres. Para finalizar cat interrúmpalo con <Ctrl+C>.

Si alguna de estas dos pruebas de lectura o escritura no ha funcionado posiblemente se deba a unos permisos demasiado restrictivos o a una diferente configuración en su equipo y eso supone que tendrá dificultades para practicar esta lección pero nosotros vamos a ir explicando todo lo que sucede en cada práctica.

Sesión de trabajo en un terminal (getty)

Repasemos un poco cosas que ya comentamos en lecciones anteriores. Hay un fichero de configuración '/etc/inittab' que durante el arranque activará un programa llamado 'getty'. Este mandará un mensaje a la consola en espera de que alguien introduzca su nombre de usuario. Cuando esto sucede 'getty' hace un exec al programa 'login' con el nombre de usuario obtenido y login solicitará la password. Si la password es correcta login hará un exec a la shell.

Esto ya lo vimos cuando hablamos de los procesos y si recuerda 'exec' es una llamada al sistema que cambia el código del programa que se está ejecutando sin cambiar su PID. Cuando la shell termine, el proceso 'init' detectará la muerte de su proceso hijo y arrancará un nuevo 'getty' porque en '/etc/inittab' estos procesos figuran en modo respawn. También figura el nombre del dispositivo y la velocidad de comunicación.

Los terminales que no tengan entrada en este fichero /etc/inittab no estarán disponibles para una sesión de trabajo en modo consola pero eso no quiere decir que sean inútiles. Por ejemplo si en /etc/inittab solo aparecen los terminales que van desde tty1 a tty6 solo podrá abrir estos seis terminales virtuales pero si se arrancan las Xwindows lo harán en el primer terminal disponible que en este caso será el tty7.

Este es un curso de usuario y a pesar de que a estas alturas del curso se explican cosas algo avanzadas no pretendemos enseñarle a administrar el

sistema. En particular le advertimos que no debe intentar modificar /etc/inittab aunque disponga de cuenta de root si no sabe exactamente lo que está haciendo, porque su sistema podría no volver a arrancar.

'getty' es un programa normal que no requiere privilegios especiales para ser ejecutado.

La práctica que ponemos ahora solo será posible si su puesto de trabajo es un PC con Linux. Si está trabajando en un terminal independiente del ordenador central no podrá hacerlo. Si la prueba de enviar 'xxxxx' al terminal tty12 no fue satisfactoria tampoco podrá hacer esta otra prueba.

Localice getty con el comando locate por ejemplo. Supongamos que lo encuentra en '/sbin/getty'.

Mire en /etc/inittab la parte donde aparece getty.

```
$ cat /etc/inittab

#  <id>:<runlevels>:<action>:<process>
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
3:23:respawn:/sbin/getty 38400 tty3
4:23:respawn:/sbin/getty 38400 tty4
5:23:respawn:/sbin/getty 38400 tty5
6:23:respawn:/sbin/getty 38400 tty6
```

Supongamos que figura lo que acabamos de poner. respawn es la acción realizada por init. Significa en este caso ejecutar y volver a ejecutar cada vez que termine. /sbin/getty es el programa que abrirá el terminal. El primer parámetro después de getty es la velocidad de transmisión (en este caso 38400), y el último parámetro es el terminal.

Nos basamos en esta información para lanzar getty sobre el tty12:

```
/sbin/getty 38400 tty12
```

Comprobará que aparentemente el programa no termina. Pase ahora al terminal tty12 con <Alt><F12>

Verá que 'getty' ha abierto una sesión en ese terminal. 'getty' esta funcionando sin ningún problema. Introduzca su nombre de usuario y verá que login no es tan permisivo. Dará un mensaje de error no reconociendo a su padre como un padre legítimo. Para evitar usos extraños de login se termina su ejecución. Volviendo a la sesión donde arranco getty verá que este también terminó.

Este intento fracasado no quiere decir que no se pueda abrir una sesión de trabajo en una consola que no esté controlada por 'getty'. Solo significa que 'login' es un vigilante celoso de la seguridad del sistema, y hemos intentado usarlo de una manera no convencional.

No solo se puede abrir una sesión de trabajo en una consola que no esté controlada por 'getty' sino que además es muy fácil y no requiere permisos especiales. Para abrir una sesión en el /dev/tty12 basta con ejecutar 'openvt -c 12 bash' y esto lo comentamos simplemente como curiosidad para que lo pruebe si quiere pero no merece más explicación en este momento.

Toda esta práctica un poco rara pretende que comprenda como funcionan las cosas. El tema de los terminales es bastante amplio.

La librería ncurses

No vamos a aprender a usar estas librerías porque este no es un curso de programación pero es una librería muy importante y encontrará montones de referencias en el manual de Linux a estas páginas. Resulta imprescindible comentar algo de ncurses para comprender el comportamiento de algunas aplicaciones.

Ya hemos dicho que hay muchas clases de terminales y que incluso un determinado tipo de terminal físico puede funcionar de formas distintas, emulando unos terminales a otros. Cada terminal tiene su propio lenguaje

de comandos que no son otra cosas de secuencias de caracteres asociadas a determinadas acciones. Por ello una aplicación debe conocer con que terminal está tratando.

Las aplicaciones hacen esto usando una librería intermedia que se encargará de traducir cada acción a la secuencia de caracteres adecuada. Esta librería se llama 'curses' o su versión moderna llamada 'ncurses'. La aplicación debe de ser informada de el tipo del terminal con el cual está trabajando y eso se hace a través de la variable TERM.

Si usted consulta (hágalo) el valor de esta variable seguramente obtendrá 'linux' que es el nombre del terminal que linux usa por defecto. Hay muchos tipos de terminales y cada uno tiene su nombre. Por ejemplo si usted ejecuta TERM='vt100'; export TERM; La aplicación que se ejecute a continuación puede pensar que tiene que dialogar con un terminal tipo 'vt100'.

Bueno algunas aplicaciones no hacen esto. Directamente asumen que funcionarán en un determinado tipo de terminal y por ello no usan ni la variable TERM ni la librería curses. Por ejemplo ls --color=auto asumirá siempre que el terminal es un terminal 'linux' y enviará directamente las secuencias de caracteres apropiadas para un terminal 'linux' aunque existe un fichero de configuración más que nada para poder poner el color que uno quiera a cada cosa. Puede probar a cambiar la variable TERM y comprobará que los colores salen perfectamente.

Por el contrario la utilidad 'less' si usa 'ncurses' úsela poniendo la variable TERM=att4424 y comprobará que la aplicación se comporta de un modo extraño. En cambio si usa la variable TERM=vt100. Se comportará de forma casi normal porque el terminal 'vt100' y el terminal 'linux' tienen algunas similitudes. En realidad vt100 (DEC) es un tipo de terminal muy conocido, y el terminal de linux deriva de él.

Hemos elegido 'ls' y 'less' porque no suponen un peligro en caso de uso incorrecto pero si intenta hacer esto con otro programa como por ejemplo

un editor podría destruir un fichero debido al desconcierto organizado en su pantalla.

La librería curses accede a una base de datos de terminales (terminfo) donde se establecen la capacidades de cada terminal.

Si usted está trabajando con linux en un terminal que no es de tipo 'linux' notará que algunas aplicaciones no van bien. La forma de arreglarlo es cambiar la variable TERM con el valor apropiado a su terminal y así por lo menos las aplicaciones que usen curses volverán a funcionar perfectamente.

El terminal de Linux

Las capacidades del terminal de linux son muy variadas. Manejo del cursor, borrado de partes o de la totalidad de la pantalla, colores y otros atributos, pitido del terminal, configuración de la pantalla en filas y columnas, guardar posición del cursor, etc.. Las páginas del manual anteriormente mencionadas tratan de todo esto y la cantidad de posibilidades es tan grande que nos limitaremos a poner algunos ejemplos.

Para conocer el juego de caracteres completo y caracteres de control que tiene un terminal linux consulte en la sección 7 del manual las páginas de ascii(7) e iso_8859_1(7). Para una relación de las secuencias de control consulte la página console_codes(4).

Antes que nada conviene saber que el terminal puede quedar en un estado que resulte muy difícil de usar. Si eso llega a ocurrir teclee '<Intro>reset<Intro>' y con un poco de suerte todo volverá a la normalidad. Pruebe ahora este comando para asegurarse de que dispone de el antes de hacer la prueba siguiente.

```
$ reset
```

```
Erase is delete.
Kill is control-U (^U).
Interrupt is control-C (^C).
```

Se muestra la configuración por defecto de algunos parámetros que luego explicaremos cuando hablemos de stty.

Vamos a hacer una prueba algo delicada. Vamos a enviar un carácter de control al terminal que lo dejará en un estado bastante lamentable para trabajar con él, para ver como se puede recuperar una situación de este tipo.

Vamos a enviar un <Ctrl+N> pero para ello hay que usar una secuencia de escape idéntica a la que se vio en el capítulo del editor 'vim'. Concretamente la secuencia para introducir el <Ctrl+N>, será <Ctrl+V><Ctrl+N>. Esto aparecerá como ^N. Antes que nada cámbiese al directorio '/tmp' donde no podremos hacer muchos destrozos. Después de enviar el '^N' al terminal no podremos ver lo que hacemos porque saldrán caracteres semi gráficos en lugar de letras.

```
$ cd /tmp
$ echo '^N''
$FIN
```

Ahora olvide se del monitor por un momento y haga <Intro>reset<Intro>. Se supone que habrá funcionado bien. En caso contrario no se aceptan reclamaciones. Dispone usted de otros terminales virtuales en otros ttys así que si un terminal no se recupera puede usar otros. Conviene que aprenda a recuperar estas situaciones.

Vamos a hacer un programita que usa las secuencias de escape del terminal de linux. La explicación la puede consultar en console_codes(4).

Edite un fichero que llamaremos 'pru_console_codes.bash'.

```
function muestra(){
        # echo -e "\033[${POSY};${POSX}H" # Manda el
cursor a $POSY $POSX
        let POSY=POSY+1 ; echo -e "\033[${POSY};$
{POSX}H\c"
        echo -e '\033[30mNegro\c'
        let POSY=POSY+1 ; echo -e "\033[${POSY};$
{POSX}H\c"
        echo -e '\033[31mRojo\c'
        let POSY=POSY+1 ; echo -e "\033[${POSY};$
{POSX}H\c"
        echo -e '\033[32mVerde\c'
        let POSY=POSY+1 ; echo -e "\033[${POSY};$
{POSX}H\c"
        echo -e '\033[33mMarron\c'
        let POSY=POSY+1 ; echo -e "\033[${POSY};$
{POSX}H\c"
        echo -e '\033[34mAzul\c'
        let POSY=POSY+1 ; echo -e "\033[${POSY};$
{POSX}H\c"
        echo -e '\033[35mRosa\c'
        let POSY=POSY+1 ; echo -e "\033[${POSY};$
{POSX}H\c"
        echo -e '\033[36mCeleste\c'
        let POSY=POSY+1 ; echo -e "\033[${POSY};$
{POSX}H\c"
        echo -e '\033[37mBlanco\c'
}

typeset -i POSY ; typeset -i POSX
echo -e '\033[2J\c' # Borrar pantalla

echo -e '\033[1m\c' # Atributo Brillo
POSY=1 ; POSX=16 ; echo -e "\033[${POSY};${POSX}H\c"
echo -e '*Brillo*\c'; muestra

echo -e '\033[2m\c' # Atributo Semi brillo
POSY=1 ; POSX=31 ; echo -e "\033[${POSY};${POSX}H\c"
echo -e '*Semi Brillo*\c'; muestra

echo -e '\033[0m\c' # Atributo normal
echo -e '\033[5m\c' # Atributo Intermitente
POSY=1 ; POSX=46 ; echo -e "\033[${POSY};${POSX}H\c"
echo -e '*Intermitente*\c'; muestra

echo -e '\033[0m\c' # Atributo normal
```

```
echo -e '\033[7m\c' # Atributo Inverso
POSY=1 ; POSX=61 ; echo -e "\033[${POSY};${POSX}H\c"
echo -e '*Inverso\c'; muestra

echo -e '\033[0m\c' # Atributo normal
POSY=1 ; POSX=1 ; echo -e "\033[${POSY};${POSX}H\c"
echo -e '*Normal*\c'; muestra

echo -e '\033[5m\c' # Atributo intermitente
echo -e '\033[7m\c' # Atributo Inverso
POSY=12 ; POSX=1 ; echo -e "\033[${POSY};${POSX}H\c"
echo -e '*Inverso+Intermitente\c'; muestra

echo -e '\033[0m\c' # Atributo normal
echo -e '\033[5m\c' # Atributo intermitente
echo -e '\033[1m\c' # Atributo brillo
POSY=12 ; POSX=26 ; echo -e "\033[${POSY};${POSX}H\c"
echo -e '*Brillo+Intermitente\c'; muestra

echo -e '\033[0m\c' # Atributo normal
echo -e "\033[25;1H\c"
```

El programa termina poniendo el atributo normal pero recuerde que si algo sale mal y el terminal queda en un estado poco operativo deberá recuperarlo con el comando reset. Una vez dicho esto ponga permiso de ejecución a este fichero y ejecutelo.

Podrá comprobar como sale la información con distintos colores y atributos. Para ello hemos usado secuencias de caracteres especificas de los terminales linux y que podrían no funcionar en otros terminales.

Control de un terminal (setterm)

Para cambiar los atributos de un terminal cualquiera se usa setterm(1) que utilizará la variable TERM para identificar el terminal y usar las secuencias de caracteres adecuadas.

setterm admite el nombre de un terminal para enviar los códigos de control a ese terminal en lugar de enviarlos al terminal actual.

Vamos a cambiar el sonido del pitido de nuestro terminal.

```
$ setterm -bfreq 100 -blength 500
```

Ahora tendremos un pitido más grave de lo normal de medio segundo. Antes de volverlo a su situación normal cambie a otro terminal y compruebe que el cambio de sonido no ha tenido efecto sobre ese terminal. No piense que su ordenador tiene un único altavoz porque es como si existiera un altavoz para cada terminal solo que no puede funcionar mas que uno de ellos en un momento dado. En eso consiste la emulación de los terminales virtuales. Es un solo terminal físico pero se comporta como varios.

Para dejar el sonido como al principio haremos lo siguiente:

```
$ setterm -reset
```

Ahora vamos hacer un programa que sacará distintos sonidos.

```
sound(){
        setterm -bfreq $1 -blength $2
        echo -e '\a\c' > /dev/console
}

sound 2000 900
sleep 1
sound 700 400
sleep 1
sound 150 300
sleep 1
sound 70 200
sleep 1
sound 40 1000
sleep 1
setterm -reset # Volver a la situación normal
echo -e '\a\c' > /dev/console
```

setterm -cursor off setterm -cursor on setterm -repeat on

Configuración de un terminal (stty)

El programa stty(1) sirve para comprobar o alterar el estado de un terminal.

```
$ stty

speed 38400 baud; line = 0;
-brkint ixoff -imaxbel
-iexten
```

Esto solo muestra los parámetros más importantes. Para mostrar la totalidad de los parámetros que controlan nuestro terminal haremos

```
$ stty -a

speed 38400 baud; rows 25; columns 80; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D;
eol = <no definido>;
eol2 = <no definido>; start = ^Q; stop = ^S; susp =
^Z; rprnt = ^R;
werase = ^W; lnext = ^V; flush = ^O; min = 1; time =
0;
-parenb -parodd cs8 hupcl -cstopb cread -clocal
-crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr
-igncr icrnl ixon ixoff
-iuclc -ixany -imaxbel
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel
nl0 cr0 tab0 bs0 vt0 ff0
isig icanon -iexten echo echoe echok -echonl -noflsh -
xcase -tostop -echoprt
echoctl echoke
```

Cuando ejecutamos reset nos muestra la configuración de los siguientes tres parámetros. (erase = ^? , kill = ^U , intr = ^C) que son respectivamente el caracter que el terminal interpretará como el caracter

para borrar el último caracter tecleado, el caracter para borrar la línea actual, y el caracter para mandar una señal de interrupción.

En la página man stty(1) viene la interpretación de todos los parámetros usados para configurar un terminal, Lo normal es que su valor permita un uso normal del terminal y por ello tocarlos podría afectar a la operatividad del terminal.

stty también permite guardar la configuración actual de un terminal en un fichero con la opción -g en un formato que permitirá al propio stty ajustar todos los parámetros de un terminal usando uno de estos ficheros generados con la opción -g.

En la prática que sigue debe recordar de para introducir un caracter de control deberá venir precedido de <Ctrl+V>

```
$ # Guardamos la configuración
$ stty -g > /tmp/stty.out
$ cat /tmp/stty.out
 1500:5:4bf:a3b:3:1c:7f:15:4:0:1:0:11:13:1a:
0:12:f:17:16:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0
$ stty werase ^A # Ojo usar <Ctrl+V> <Ctrl+A>
$ stty -a
$ # Comprobamos que ahora werase = ^A;
 speed 38400 baud; rows 25; columns 80; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D;
eol = ;
eol2 = ; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R;
werase = ^A; lnext = ^V; flush = ^O; min = 1; time =
0;
-parenb -parodd cs8 hupcl -cstopb cread -clocal
-crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr
-igncr icrnl ixon ixoff
-iuclc -ixany -imaxbel
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel
nl0 cr0 tab0 bs0 vt0 ff0
isig icanon -iexten echo echoe echok -echonl -noflsh -
xcase -tostop -echoprt
echoctl echoke
$ stty `cat /tmp/stty.out` # Recuperamos la
```

```
configuración anterior
```

Los resultados de esta práctica pueden variar de un sistema a otro. Lo importante es que compruebe como hemos salvado la configuración a un fichero antes de alterar la configuración del terminal y como podemos recuperar esa configuración desde ese fichero. Usamos solo werase para ello y no vamos a proponer más ejercicios sobre esto. Los terminales en Linux no emulan siempre a la perfección a los terminales tradicionales. Lo explicamos en el siguiente apartado (ver readline).

La librería readline

Nuevamente tenemos que comentar la existencia y propósito de una librería especial no para aprender a manejarla sino para comprender una situación bastante especial de los terminales en Linux. Los terminales virtuales en Linux no se comportan como los terminales otros sistemas tipo Unix. Por ejemplo podemos cambiar el caracter de control de 'erase' con stty y parecerá que no tiene ningún efecto, cosa que se contradice totalmente con lo que dijimos antes. Esto se debe a que muchos programas de linux usan la librería 'readline' que tiene sus propias funciones de edición de linea etc.. y actúan sobreescribiendo las funciones del terminal. Esto determina dos tipos de usos distintos del terminal y dos comportamientos distintos.

- El bash y muchos de los programas de GNU usarán la librería readline. Los programas que usen readline usaran $HOME/.inputrc para la configuración de ciertos caracteres de edición de línea. En caso de no existir este fichero se usa /etc/inputrc. Por ello el caracter de borrado de ultimo caracter y otros no responderán a lo mostrado por stty sino al fichero inputrc.
- Por el contrario un programa C que use gets() para leer directamente del teclado no usará la librería readline y por ello lo que funcionará será lo que determinen los parámetros de stty.

Es como si existieran dos clases de dispositivos. Uno sería tty y el otro lo podríamos llamar tty->readline.

No es cuestión de profundizar en estos temas que son más adecuados para un curso de administración o de programación. Con esto se pretende simplemente dar una explicación lógica a este tipo de comportamiento muy específico de los terminales Linux.

Terminales y pseudoterminales

Ya hemos visto como puede dirigirse entrada salida a un terminal pero un proceso puede tener previsto dos comportamientos distintos para el caso que su salida está conectada a un terminal o a un fichero. Por ejemplo la opción '--color=auto' para 'ls' resulta muy útil ya que producirá dos tipos de salidas distintos dependiendo de que esté conectada bien a un terminal bien a otra cosa tipo fichero pipe etc... En el caso de estar conectado a un terminal, sacará la información en color, pero si detecta otra cosa sacará la información sin color para evitar volcar un montón de caracteres de extraños. En otras palabras 'ls' y muchos otros programas pueden averiguar si sus descriptores de entrada salida estándar están conectado o no a un terminal. La salida a un terminal determina un uso interactivo con una persona y por ello los programas se adaptan a ello. Algunas veces queremos que el comportamiento de un programa sea como si estuviera conectado a un terminal pese a que no sea cierto y porv ello algunas veces hay que usar un terminal ficticio o pseudoterminal.

Un pseudoterminal es un componente software destinado a simular un terminal. Se implementan usando una pareja de dispositivos especiales y distintos denominados esclavo y maestro. Un proceso que use un pseudoterminal pensará que está conectado a un terminal auténtico y no notará diferencia alguna.

Por ejemplo una ventana de 'xterm' en windows funciona de esa manera.

Un ejemplo de pseudoterminal (script)

Scrip es un comando que funciona creando un pseudoterminal y arrancando una subshell que no notará que la entrada salida no es un autentico terminal. Script se usa para grabar una sesión completa de trabajo con bash en un fichero si se termina al finalizar la ejecución de la shell con exit.

Pruebe lo siguiente:

```
$ script /tmp/script.out
$ ls --color=auto /

$ ls --color=auto / > /tmp/ls.out
$ exit
```

El primer 'ls --color=auto /' mostrará el contenido del directorio raíz en colores. El segundo 'ls --color=auto / > /tmp/ls.out' guardará la salida en un fichero '/tmp/ls.out'. La sesión de script se finaliza abandonando la shell con exit. Cuando miremos el contenido de '/tmp/ls.out' con el editor 'vi' comprobaremos que no hay colores ni caracteres extraños. Si miramos la sesión completa grabada con script en el fichero '/tmp/script.out' con 'vi comprobaremos que está llena de caracteres extraños. Los retornos de carro aparecen como '^M' (representa <Ctrl+M> y los códigos de colores con secuencias del tipo '^[[0m...' o similares que responden a las secuencias de control para modificar atributos y colores en la pantalla según se describe en la página console_codes(4). Si envía el contenido de este fichero a pantalla por ejemplo usando 'cat /tmp/script.out' aparecerán los colores.

script captura absolutamente todo y por ello grabar una sesión con script en un fichero puede requerir un tratamiento de filtrar ciertas cosas antes de darle determinado uso a esa salida, pero resulta muy útil si solo deseamos registrar una sesión de trabajo para no tener que ir apuntando lo que vamos haciendo o lo que vamos obteniendo.

PROGRAMACION DE TAREAS EN EL TIEMPO

Introducción

En este punto del curso podemos asumir que ya sabe programar algunos scripts para hacer cosas que le interesan. Si le preguntáramos como hacer para que un programa se ejecute dentro de tres horas usaría el comando 'sleep 10800'. Y efectivamente eso sería una solución sencilla que funcionaría muy bien ya que sleep no consume nada de cpu y tres horas son 10800 segundos, pero probablemente se habrá preguntado como hacer para que una tarea se ejecute en un determinado momento o como hacer para que una tarea se ejecute periódicamente.

Para eso disponemos de los comandos 'at', 'batch' y 'cron'. A 'batch', y a 'at' hay que pasarles el comando por la entrada estándar, y con 'batch', 'at', y a 'cron' la salida del comando será dirigida a la cuenta de correo del usuario. Por eso conviene redirigir las salidas convenientemente y resulta evidente que de esta forma no se pueden ejecutar programas interactivos.

procmeter3 una herramienta para monitorizar el sistema

Este es un programa que nos resultará de gran utilidad para controlar la carga del sistema y en general para vigilar cualquier aspecto de la actividad en el sistema. La mayor parte de esta información es obtenida a partir de /proc y es altamente configurable.

Para esta práctica deberá activar los elementos para CPU y para LOAD en modo gráfico representado por una línea quebrada. Advierta que este tipo de gráfica poseen unas rayas horizontales y un numerito entre paréntesis en la parte inferior. Ese numerito indica la unidad correspondiente. Por ejemplo para CPU aparece (20%) y para LOAD aparece (1). Esto significa que en la primera gráfica cada ralla horizontal supone un 20% de

uso de CPU. En LOAD cada ralla significa una carga de 1. Estos valores son los establecidos por defecto y aunque esto y otras cosas pueden ser configuradas de forma distinta nosotros lo usaremos así.

Las utilidades at, atq y atr.

Permite programar una tarea para que se ejecute en un determinado momento. 'at' usa una jerarquía de colas y cada una de ellas se refiere a un nivel 'nice' que es el valor que otorga la cantidad de recurso de cpu que puede ocupar el proceso en detrimento de otros que se estén ejecutando simultáneamente. Los niveles se nombran con un único carácter que va de la 'a' a la 'z' o de la 'A' a la 'Z'. Las primeras letras del alfabeto están asociadas con alta capacidad para usar CPU y las últimas todo lo contrario. Por defecto el comando 'at' ejecutará con el nivel 'a'.

Existe un comando llamado 'batch' que sirve para ejecutar comandos de forma no inmediata pero sin especificar el momento. Lo explicaremos más adelante y viene a ser lo equivalente a "ejecuta esto cuando no estés muy ocupado". Lo comentaremos luego pero resulta que el comando 'at' también puede funcionar de esta forma.

Si se usa 'at' especificando el nivel de ejecución con una mayúscula el programa no se ejecutará obligatoriamente en el momento indicado sino que esperará a que la carga del sistema sea inferior al valor establecido para los procesos batch.

echo 'echo Hola > /dev/console' | at now + 2 minutes

Podrá comprobar que el sistema devuelve información relativa a la tarea.

job at

Para comprobar las tareas planificadas ejecute 'atq' para eliminar una tarea planificada que aun no se ha ejecutado 'atrm '

Por ejemplo si su usuario fuera pepe y quiere enviarse asi mismo un mensaje de recordatorio para el medio día del Sábado puede hacer que el sistema a esa hora le envíe un correo electrónico.

echo "mail -s 'Recordatorio. Llamar a Jaime' pepe" | at NOON MON

Las especificaciones de tiempo utilizan una sintaxis especial pero quizás resulte más intuitivo y más práctico poner a modo de chuleta unas pocas reglas siguiendo una notación arbitraria que poner la gramática completa en formato yacc o en formato BNF. Esto nos llevaría a tener que explicar cosas que se salen del propósito de este curso y que serían más apropiadas para un curso de programación. Por ello y basandonos en un resumen arbitrario de esa gramática incluimos el siguiente cuadro resumen.

```
timespec        = time | time date | time increment
                | time date increment | time decrement
                | time date decrement | nowspec

nowspec         = NOW | NOW increment | NOW decrement

time            = hr24clock | NOON | MIDNIGHT |
TEATIME

date            = month_name day_number
                | month_name day_number ','
year_number
                | day_of_week | TODAY | TOMORROW
                | year_number '-' month_number '-'
day_number

increment       = '+' inc_number inc_period

decrement       = '-' inc_number inc_period

hr24clock       = INT 'h' INT

inc_period      = MINUTE | HOUR | DAY | WEEK | MONTH |
YEAR

day_of_week     = SUN | MON | TUE | WED | THU | FRI |
SAT
```

```
month_name      = JAN | FEB | MAR | APR | MAY | JUN |
JUL
                | AUG | SEP | OCT | NOV | DEC
```

Lo que sigue es una regla práctica para el uso de esta chuleta sin demasiado valor técnico. Esperamos que sea util:

Para saber si una especificación se ajusta o no, se parte siempre de 'timespec'. Las reglas las representamos como una serie de posibilidades separadas por '|'. Cuando una de estas posibilidades aparece en minúculas significa que a su vez esa posibilidad tiene unas reglas. Por ejemplo para comprobar que 'NOON MON' cumple la especificación nos fijamos en que 'timespec' entre otras cosas se corresponde con 'time' seguido de 'date'. NOON (medio día) es una forma válida de 'time' y MON (Lunes) es una forma válida de 'day_of_week' que a su vez es una forma válida de 'date'.

```
timespec    =  time date    | .....
time        =  NOON         | .....
date        =  day_of_week | ....
day_of_week =  MON          | .....
```

Ejemplos:

at 9am TUE
at now + 2 minutes
at 9am Feb 18
at 4pm + 3 days
at 10am Jul 31
at 1am tomorrow

En Debian la especificación de tiempo en formato yacc se encuentra en el fichero '/usr/doc/at/timespec' que no se corresponde con lo que hemos puesto porque hemos pretendido simplificar esta información.

'atq' solo da información del número de tarea y del momento en que se va a ejecutar pero la tarea en si misma no. En Debian las tarea se guardan en '/var/spool/cron/atjobs/' y resulta interesante como se guardan. Son simplemente un scrip que lanza el comando indicado pero antes de ello recupera todo el entorno desde el cual se lanzó y se situa en el directorio desde el cual se lanzó. En el fichero '/var/spool/cron/atjobs/.SEQ' se guarda un contador de secuencia para los trabajos pero en hexadecimal.

Si existe el fichero '/etc/at.allow' solo los usuarios contenidos estarán autorizados a lanzar trabajos con 'at'. Si no existiera '/etc/at.allow' se permitirá el acceso a todos los usuarios salvo que exista un fichero '/etc/at.deny' en cuyo caso los usuarios contenidos en el no podrán usar 'at'.

La utilidad batch

Ya hemos comentado que sirve para instruir al sistema que ejecute un comando cuando la carga del sistema no sea alta y además lo hará con una prioridad baja. Si la carga no desciende por debajo del valor establecido para las tareas batch, el programa no se ejecutará nunca.

A modo de orientación podemos decir que para sistemas monoprocesador suele usarse valores de 0.7 a 1.5 y para sistemas multiprocesador será superior y estará en función del número de CPUs.

Vamos a programar un shell-script que consuma mucha CPU y que deje una traza que nos permita saber cuando arranca y cuando para.

```
# piloop
if [ $# -ne 1 ]
        then echo "Uso piloop10 "
        else
        for i in 0 1 2 3 4 5 6 7 8 9
        do
           echo "$$ $i `date`" >> /tmp/piloop.out
           echo "scale=$1; 4*a(1)" | bc -l >>
/tmp/piloop.bc
```

```
        done
fi
```

El valor de la escala se pasa como parámetro y deberá probar primero con valores bajos hasta que consuma uno o dos minutos para poder hacer la práctica con comodidad. En nuestro equipo el cálculo del número pi con 1000 cifras decimales resulta un valor adecuado.

1. procmeter3
2. nice --19 piloop 1000 &
3. Esperar a que suba la carga
4. echo piloop 1000 | batch

Los procesos planificados con 'at' y 'batch' son gestionados por 'atd'. Muchos de los procesos de este tipo que están permanentemente funcionando terminan en 'd' de 'daemon'. En ingles daemon significa duende. Si por cualquier anomalía atd dejara de funcionar no se podría usar 'at' ni 'batch'. Curiosamente hay un proceso que se encarga de gestionar tareas periódicas y que igualmente está permanentente funcionando pero no se llama crond sino 'cron'.

Uso de cron y crontab

El comando 'cron' permiten ejecutar periódicamente una serie de tareas que estarán programadas en unos ficheros especiales.

La planificación de tareas para usuarios no se hace directamente sino pasando un fichero a 'crontab' o editándolo directamente con 'crontab -e'. Esto generará un fichero en /var/spool/cron/crontabs/ Para verificar el contenido del fichero crontab de un usuario basta con usar 'crontab -l' y para eliminarlo 'crontab -r'.

Poner siempre como mínimo 2 minutos más al actual para probar ya que en caso contrario puede no funcionar.

El formato consiste en una serie de lineas con el formato siguiente:

minuto hora día_del_mes mes día_de_la_semana

Campo	**Valores Permitidos**
minuto	0-59
hora	0-23
día_del_mes	1-31
mes	1-12 o nombres
día_de_la_semana	0-7 o nombres 0 o 7 es Domingo

Los nombres de meses y días de la semana dependen del idioma que se use (variable LC_TIME). Puede probar con el comando 'date' para ver cuales se están usando en su sistema.

Dos valores con un guión en medio indica un rango de valores que incluirían los valores extremos indicados. Un asterisco (*) equivale al rango de valores que van desde el primero al último. Y se pueden especificar una lista de elementos o de rangos separándolos por comas. Por ejemplo "20-23,0-3" equivale a "20,21,22,23,0,1,2,3"

Como veremos a continuación hay una forma de agrupar las acciones que se han de ejecutar en el mismo periodo.

A parte de las tareas periódicas que cada usuario pueda programarse usando 'crontab -e' para sus propias necesidades, el sistema tendrá una serie de tareas programadas y estas se especifican en /etc/crontab.

En el caso de Debian cron incluye no solo las tareas de /etc/crontab sino cada una de las tareas que se incluya en los ficheros contenidos en el directorio /etc/cron.d

La utilidad run-parts

En muchas distribuciones (y Debian es una de ellas) las tareas que se han de ejecutar en un mismo momento se agrupan en un único directorio para ser procesadas. En Debian estos directorios son /etc/cron.daily, cron.weekly, y cron.monthly.

De esta forma en crontab podemos indicar como acción run-parts directorio. La acción expresada de esa forma ejecutaría todos aquellos scripts contenidos en ese directorio. Todos estos scripts deberán comenzar forzosamente por #!/bin/interprete_de_comandos.

De esta forma añadir una nueva acción resulta muy sencillo y no requiere tocar el fichero de configuración de cron /etc/crontab que normalmente se limita a contener la programación de tareas genéricas para ciertos periodos. Se verá más claro con un ejemplo. El ejemplo que sigue está tomado de una Debian:

```
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the
`crontab'
# command to install the new version when you edit
this file.
# This file also has a username field, that none of
the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sb
in:/usr/bin
# (mon = 3,6 = Miercoles,Sabado)
# m h dom mon dow user command
25 6    * * *   root    test -e /usr/sbin/anacron ||
run-parts --report /etc/cron.daily
50 6    * * 7   root    test -e /usr/sbin/anacron ||
run-parts --report /etc/cron.weekly
```

```
55 6    1 * *   root    test -e /usr/sbin/anacron ||
run-parts --report /etc/cron.monthly
#
```

Cada acción consiste en una comprobación. Si no existe anacron que es un programa que explicaremos seguidamente se ejecutarán los scripts contenidos en los respectivos directorios. En caso contrario no se hace nada ya que se asume que anacron se encargará de ello.

Mejorar el comportamiento de cron con anacron.

El comando cron funciona muy bien en un sistema que funcione de modo ininterrumpido. Si un sistema permanece un mes apagado y lo encendemos, cron activará simultáneamente todas las tareas pendientes. Estas seguramente consistirán en tareas diarias, semanales, y mensuales entre otras. El comando 'anacron' permite corregir los problemas clásicos que presenta el uso de 'cron' en sistemas que no están funcionando de forma continua. Para ello se establece en el fichero /etc/anacrontab el periodo y el retraso para activar las tareas periódicas.

Por ejemplo en Debian tenemos por defecto lo siguiente en el fichero /etc/anacrontab:

```
# /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/sb
in:/usr/bin
# These replace cron's entries
1       5       cron.daily      nice run-parts
--report /etc/cron.daily
7       10      cron.weekly     nice run-parts
--report /etc/cron.weekly
30      15      cron.monthly nice run-parts --report /
etc/cron.monthly
```

¿Que sucederá usando todo esto en un sistema que arranque después de un largo periodo de estar apagado? Pues que a los 5 minutos arrancará las tareas diarias, a los 10 las semanales caso de estar pendientes, y a los 15 minutos las mensuales, caso de estar pendientes. De esa forma el sistema reparte en el tiempo la carga de trabajo de forma escalonada.

Un caso práctico. Tareas dos veces por semana

En el caso anterior podemos incluir una tarea para que se ejecute una vez cada siete días o una vez al día pero no disponemos de una programación intermedia. Vamos a suponer el siguiente contenido en los directorios de tareas periódicos.

```
# ll /etc/cron.daily/

-rwxr-xr-x    1 root     root          238 mar 15
1999 man-db
-rwxr-xr-x    1 root     root           51 sep 13
1999 logrotate
-rwxr-xr-x    1 root     root          592 feb 27
2000 suidmanager
-rwxr-xr-x    1 root     root         2259 mar 30
2000 standard
-rwxr-xr-x    1 root     root         3030 abr 29
2000 apache
-rwxr-xr-x    1 root     root          427 abr 30
2000 exim
-rwxr-xr-x    1 root     root          157 may 19
2000 tetex-bin
-rwxr-xr-x    1 root     root          311 may 25
2000 0anacron
-rwxr-xr-x    1 root     root         1065 jun  8
2000 sendmail
-rwxr-xr-x    1 root     root          450 jul 18
2000 calendar
-rwxr-xr-x    1 root     root          660 ene  9
2001 sysklogd
-rwxr-xr-x    1 root     root          485 ene  9
2001 netbase
-rwxr-xr-x    1 root     root           41 ene  9
2001 modutils
-rwxr-xr-x    1 root     root           57 feb 29
```

```
2000 htdig
-rwxr-xr-x    1 root     root          277 ene  9
2001 find

# ll /etc/cron.weekly/

-rwxr-xr-x    1 root     root          210 nov 28
1998 man-db
-rwxr-xr-x    1 root     root          540 nov  2
1999 isdnutils
-rwxr-xr-x    1 root     root          217 ene  9
2000 lpr
-rwxr-xr-x    1 root     root          116 feb 28
2000 man2html
-rwxr-xr-x    1 root     root         1302 mar 22
2000 cvs
-rwxr-xr-x    1 root     root          653 mar 23
2000 dhelp
-rwxr-xr-x    1 root     root          312 may 25
2000 0anacron
-rwxr-xr-x    1 root     root          781 ene  9
2001 sysklogd

# ll /etc/cron.monthly/

-rwxr-xr-x    1 root     root          129 mar 30
2000 standard
-rwxr-xr-x    1 root     root          313 may 25
2000 0anacron
```

Imaginemos que 'htdig' y 'find' resultan para nosotros tareas excesivamente pesadas para realizarlas diariamente pero tampoco deseamos dejar pasar toda una semana sin ejecutarlas.

La solución sería incluir una nueva categoría de tareas que se ejecutarían por ejemplo un par de veces por semana.

Para ello:

1. Crearemos un directorio que llamaremos por ejemplo /etc/cron.2_weekly/.

2. Moveremos los scripts 'htdig' y 'find' a ese directorio.
3. Incluiremos en /etc/crontab una nueva entrada para las tareas que han de ejecutarse dos veces por semana y puesto que las tareas semanales están programadas para activarse los Domingos usaremos los Miércoles y los Sábados a fin de que no coincidan y queden mejor repartidas en el tiempo.

```
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the
`crontab'
# command to install the new version when you edit
this file.
# This file also has a username field, that none of
the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sb
in:/usr/bin
# (mon = 3,6 = Miercoles,Sabado)
# m h dom mon dow user command
25 6    * * *   root    test -e /usr/sbin/anacron ||
run-parts --report /etc/cron.daily
40 6    * * 3,6 root    test -e /usr/sbin/anacron ||
run-parts --report /etc/cron.2_weekly
50 6    * * 7   root    test -e /usr/sbin/anacron ||
run-parts --report /etc/cron.weekly
55 6    1 * *   root    test -e /usr/sbin/anacron ||
run-parts --report /etc/cron.monthly
#
```

4. Incluiremos en /etc/anacrontab una nueva entrada

```
# /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/sb
in:/usr/bin

# These replace cron's entries
1       5       cron.daily      nice -19 run-parts
```

```
--report /etc/cron.daily
3       10      cron.2_weekly   nice -19 run-parts
--report /etc/cron.2_weekly
7       10      cron.weekly     nice -19 run-parts
--report /etc/cron.weekly
30      15      cron.monthly nice -19 run-parts
--report /etc/cron.monthly
```

Y con ello ya tendríamos modificada la planificación tal y como la hemos descrito. A modo de comprobación podemos incluir algún script en los directorios de tareas periódicas para ver cuando se ejecutan.

En /etc/cron.daily/traza pondremos:

```
#!/bin/sh
echo "daily `date`" >> /var/log/traza_cron.log
```

En /etc/cron.weekly/traza pondremos:

```
#!/bin/sh
echo "weekly `date`" >> /var/log/traza_cron.log
```

En /etc/cron.2_weekly/traza pondremos:

```
#!/bin/sh
echo "2_weekly `date`" >> /var/log/traza_cron.log
```

En /etc/cron.monthly/traza pondremos:

```
#!/bin/sh
echo "monthly `date`" >> /var/log/traza_cron.log
```

Pasados unos días comprobaremos /var/log/traza_cron.log que deberá mostrar la ejecución de cada parte en los momentos adecuados. Se

muestra la traza en un ordenador que suele encenderse unos minutos antes de las 8 de la mañana todos los días.

```
daily Wed Nov 28 08:00:24 CET 2001
2_weekly Wed Nov 28 08:19:09 CET 2001
daily Thu Nov 29 07:45:38 CET 2001
daily Fri Nov 30 07:48:29 CET 2001
daily Sat Dec  1 08:30:52 CET 2001
2_weekly Sat Dec  1 08:49:23 CET 2001
daily Sun Dec  2 07:37:13 CET 2001
daily Mon Dec  3 07:46:48 CET 2001
weekly Mon Dec  3 07:50:28 CET 2001
daily Tue Dec  4 07:53:11 CET 2001
2_weekly Tue Dec  4 08:12:19 CET 2001
daily Wed Dec  5 07:45:44 CET 2001
daily Thu Dec  6 08:21:58 CET 2001
```

INTRODUCCION A REDES

Introducción

Este documento pretende aclarar los conceptos básicos relativos a redes en general y a redes Linux en particular. De todos los protocolos existentes nos centraremos en el TCP/IP.

El TCP/IP es la familia de protocolos más ampliamente usada en todo el mundo. Ello es debido a que es el protocolo usado por Internet (la red de redes) y por que su uso esta muy extendido desde hace mucho tiempo entre los sistemas de tipo Unix. El éxito de Internet ha contribuido de forma decisiva en su amplio uso en todo tipo de sistemas y redes.

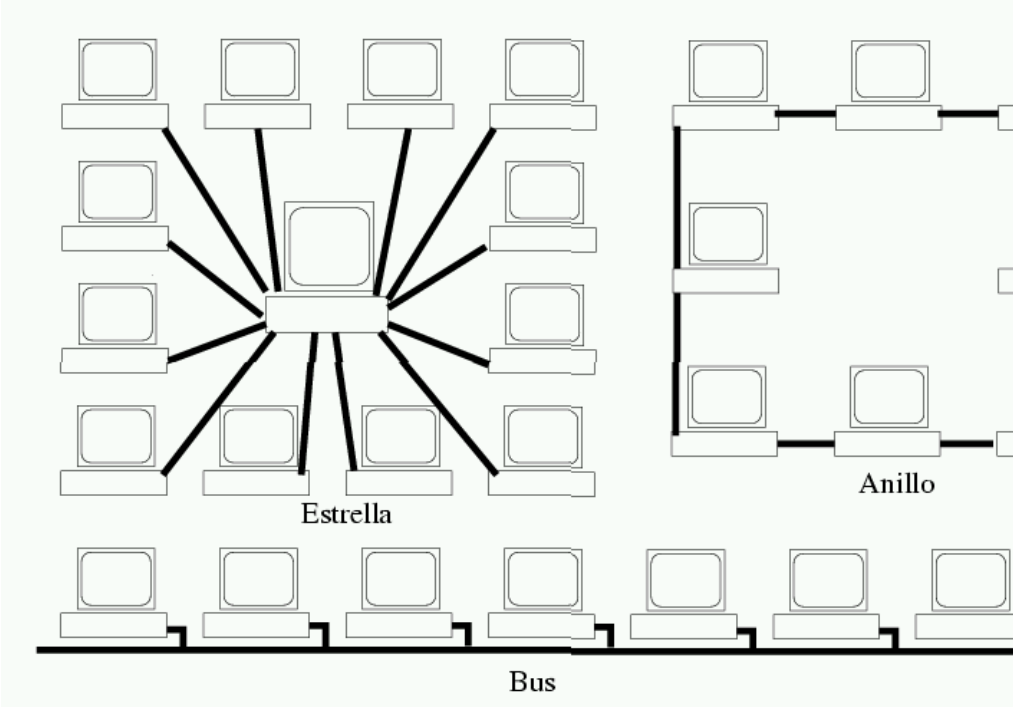
Este capítulo le ayudará a comprender los aspectos básicos de redes. Sin estos conocimientos instalar y configurar una red aunque sea pequeña resultaría complicado porque existe mucha terminología técgnica necesaria porque no siempre se pueden explicar todo con pocas y sencillas palabras.

Historia

En 1969, el departamento de Defensa de los Estados Unidos se vió obligado a dar solución a los problemas de comunicación electrónica interna que usaban sistemas informáticos variados. Por ello se encargó al ARPA (Avanced Research Projects Agency) que desarrollara, junto con las principales universidades y fabricantes de equipos informáticos, un protocolo estandar de comunicaciones. Esto dió lugar a dos redes. Una de uso exclusivamente militar MILNET y otra con fines experimentales de investigación ARPANET que en 1970 incorpora los protocolos TCP/IP. MILNet se escindió como tal en los años 80.

Topología

La topología de una red viene a ser el aspecto físico que presentan los nodos de una red y su interconexión entre ellos. Por ejemplo si los nodos se conectan a una larga línea que constituyen el medio de comunicación que todos los nodos comparten por igual diremos que es una **red en bus**. Si los nodos se organizan conectandose exclusivamente a través de un gran nodo central y con total independecia entre los restantes nodos diremos que es una **red en estrella**. Si los nodos se conectan organizando un círculo donde cada nodo solo tiene relación directa con un nodo predecesor y uno sucesor dentro del círculo, diremos que es una **red en anillo**. En la práctica cualquier red se puede conectar a otra originando modelos mixtos.



Protocolos de acceso a red

A bajo nivel son necesarias una serie de reglas preestablecidas para poder intercambiar paquetes de información. El uso de la palabra protocolo también se usará cuando hablemos de otros niveles de comunicación y ello no debe confundirnos. (El nivel que estamos tratando es el de acceso a red en terminología TCP/IP, y agrupa los niveles físico y de enlace de datos en terminología OSI.)

Cuando nos referimos a topología nos referimos a un nivel totalmente físico y cuando nos referimos a protocolos de transmisión de datos nos referinos a un nivel totalmente lógico que establece las reglas de comunicación.

En teoría la Topología de una red no determina su protocolo de transmisión de datos y efectivamente son posibles casi cualquier combinación de topología y de protocolo, pero en la práctica cada topologiá suele usarse preferentemente con unos determinados protocolos porque nacen de un diseño conjunto. Pondremos unos pocos ejemplos de protocolos de transmisión de datos y señalaremos en que topología suelen usarse con mayor frecuencia:

- Csma (Carrier Sense Multiple Access)

  Las estaciones comparten un mismo canal y quedan a la escucha respondiendo unicamente cuando un mensaje va dirijido a ellos. Esto se detecta gracias a una dirección única que consiste en números de 6 bytes asociados a cada tarjeta de red grabado generalmente por el fabricante. Es decir una colisión se produce cuando la identificación MAC del receptor que va en la trama de un mensaje, coincide con la dirección física del adaptador de red -unica para cada tarjeta-. En el caso de que dos puestos intenten usar simultaneamente el canal para transmitir los mensajes transmitidos en forma de impulsos electricos sufrirán distorsiones y la información de la señal se pierde. A este fenómeno se le llama colisión. En la practica será necesario retransmitir los paquetes colisionados.

  (Generalmente este método se asocia a topología en bus).

- Csma/CD (Carrier Sense Multiple Acces Colision Detect

Es una variación de la anterior que minimiza la probabilidad de colisión detectando antes de emitir si el bus está en uso, pero esto no impide totalmente que se produzcan colisiones. Hay otras variaciones basadas en Csma siempre intentando minimizar las colisiones.

(Generalmente se asocia a topología en bus).

- Polling (Llamada selectiva)
- Existe un nodo central que establece el turno de cada nodo. Para ello hace una llamada selectiva que solo será recibida por el nodo pertinente.

(Generalmente se asocia a topología en estrella).

- Token Passing (Paso de testigo)

Esta tecnología consiste en que cada puesto va pasando un mensaje que hace el mismo papel que un testigo en una carrera de relevos. Unicamente el nodo que tiene el testigo puede enviar el mensaje y generalmente los mensajes pasan al nodo siguiente siguiendo todos ellos una secuencia circular aunque hay variaciones de este modelo. Cuando el mensaje original vuelve al nodo que lo originó este lo elimina de la circulación del anillo. Se trata de un modelo muy estricto que ofrece un altísimo grado de control. No existe posibilidades aleatorias como en el caso de las colisiones del modelo Csma. El paso de testigo, permide conocer con total seguridad cual es el tiempo que se invertirá en hacer que un mensaje alcance su destino y permite que el uso intensivo de la red no afecte a su funcionamiento.

(Generalmente se asocia a topología en anillo).

Ethernet

Es una de las arquitecturas de red local más conocidas y populares. Cuando se habla de ethernet nos referimos a una tecnología de enlace de datos.

Ethernet fué desarrollada por Xerox en los laboratorios de Palo Alto en California. Utilizaron el protocolo CSMA/CD y los nodos de la red conectados mediande cable coaxial, usaban direcciones de 48 bits. Estas característica permitían reorganizar la red sin tener que efectuar cambios en el SO, lo cual tiene mucho que ver con su enórme éxito en la actualidad.

Considerando las desventajas de CSMA/CD se comprende que esta forma de funcionar ofrezca dificultades cuando en el canal de comunicación se produce un uso intensivo. Las inevitables colisiones se traducen en una perdida de rendimiento en situaciones de sobre carga de la red. En la práctica la mayoría de los nodos de una gran red intercambian información entre ellos de forma ocasional y las redes se dividen en subredes para minimizar colisiones de mensajes. El modelo de la arquitectura Ethernet es ante todo muy flexible y facilita la creación y gestión de redes de redes muy grandes sin grandes problemas.

En la actualidad hay una amplia variedad de formas de interconectar máquinas en una LAN. Incluso se pueden utilizar los puertos serie o paralelo para ello pero lo más frecuente es usar Ethernet. Una tarjeta a 10 Mb/s (10 Megabits por segundo) cuesta solo unas 5000 Ptas (30 Euros) y el cable cuesta unas 300 Ptas/metro (2 Euros/metro). Por lo tanto montar una pequeña red casera no supone mucho gasto. Se puede usar cable coaxial generalmente negro que se terminan en los extremos mediante terminadores especiales y admiten enlaces en forma de T para conectar tantos puestos como sea necesario sin tener que adquirir un *'hub'*.

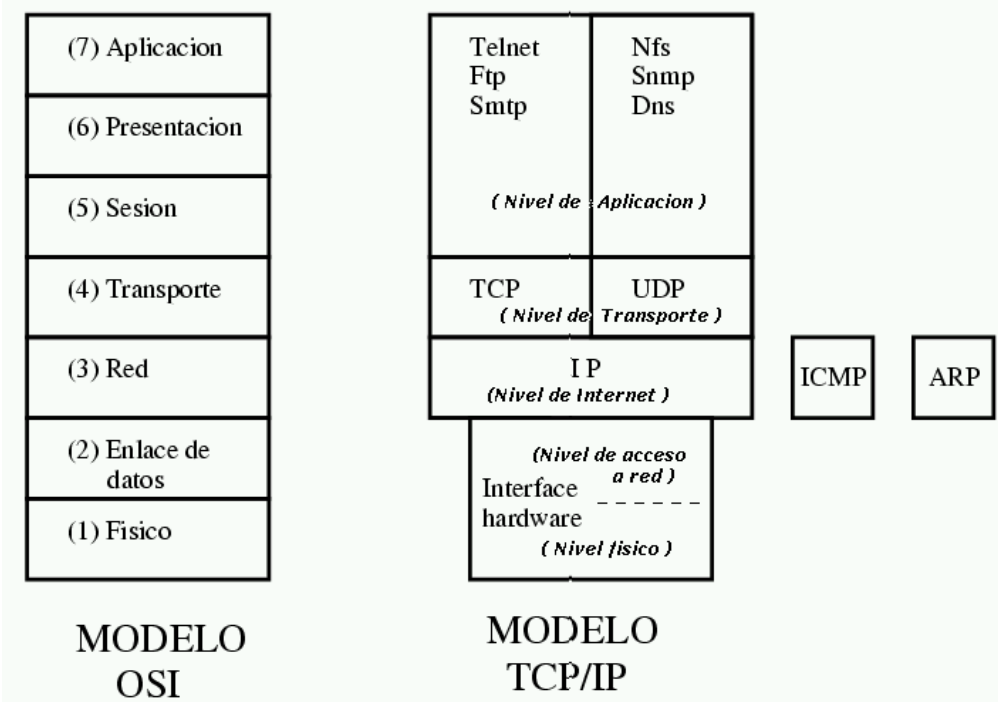
Un *'hub'* es un dispositivo concentrador que dispone de varias bocas de conexión. Resultará imprescindible su uso cuando se usa en una red de más de dos ordenadores con cable 10base2. Este cable tiene conectores de plástico RJ45 similares a los conectores de los teléfonos.

Con este cable se puede prescindir de *'hub'* solo en el caso de interconectar dos ordenadores, y para ello el cable ha de ser especial, ya que la posición de algunos de los 8 pines que tiene el conector RJ45 debarán ir cruzados. Concretamente el pin 1 (cable blanco naranja) se ha de cruzar con el pin 3 (cable blanco verde) y el pin 2 (cable naranja) se ha de cruzar con el pin 6 (cable verde).

Las tarjetas a 100 Mb/s tampoco son muy caras y son diez veces más rápidas. Cuestan unas 12.000 Ptas (72 Euros). Hay tarjetas que puede funcionar tanto a 100 como a 10 Mb/s pero en una red no pueden mezclarse tarjetas funcionando a distintas velocidades.

Modelo OSI y TCP/IP

Los protocolos TCP/IP se crearon y normalizaron en los 80, es decir mucho antes de que se definiera el modelo de referencia OSI de la ISO cuya arquitectura se impuso en los 90. Los gobiernos apoyaron desde un principio los estandares OSI que representa un modelo bien definido y estructurado. Por todo ello se pensó que este modelo triunfaría sobre el modelo TCP/IP que en realidad ni siquiera representa un modelo único sino una familia de protocolos que se han ido definiendo anarquicamente y que a posteriori se han estructurado en capas. Al final lo que realmente ha triunfado ha sido Internet que usa TCP/IP. En el modelo OSI se describen perfectamente siete distintos niveles. En el modelo TCP/IP realmente no se describe una estructura tan precisa de niveles como en el modelo OSI aunque se pueden asimilar y comparar ambos modelos de la forma siguiente.



En todos estos niveles podemos hablar de sus respectivos procolos de comunicación. El término protocolo indica un conjunto de reglas perfectamente establecidas que permiten y garantizan el intercambio de información. En cada nivel del modelo la comunicación tiene su propio protocolo y su propio cometido.

Los protocolos TCP están orientados a conexión con lo que, se podría hablar de sesión asimilandose al nivel 5 OSI pero UDP no está orientado a conexión y por ello no se puede hablar propiamente de sesión. Las corespondencias de niveles entre OSI y TCP/IP no son perfectas.

TCP está orientado a conexión. En una primera fase se establece conexión y con ello se mantiene una sesión de trabajo que finalizará con un cierre. UDP utiliza datagramas. Es decir que en lugar de abrir una conexión con el destino para establecer un dialogo, funciona simplemente enviando a la red fragmentos de información denominados datagramas. FTP, HTTP,

SMTP, y TELNET son servicios muy populares y son un buen ejemplo de protocolos orientados a conexión. SMNP, y NFS son ejemplos de servicios orientados a datagramas. Más adelante ampliaremos todo esto hablando del Modelo Cliente/Servidor

El nivel TCP y el nivel UDP delegan en el nivel IP la transmisión de la información pero no asumen que IP tenga éxito. Es decir supervisan la transmisión y se ocupan de reenviar los paquetes en caso necesario.

El nivel de IP gestiona la dirección del mensaje para que alcance su nodo destino. El protocolo IP se puede asimilar al nivel 3 OSI. En este nivel existen también los protocolos siguientes: ICMP (Internet Control Message Protocol) ARP (Address Resolution Protocol) RARP (Reverse Address Resolution Protocol).

Existen unas páginas del manual online que están traducidas que explican un poco todo esto. ip(7), udp(7), tcp(7), socket(7).

En el nivel 1 OSI llamado nivel físico se definen como son los cables, los conectores los niveles electricos de las señales, la topología, etc. Pero en el modelo TCP/IP los niveles 1 y 2 quedan asimilados a un único nivel de interfaz hardware que pasamos a describir. Tampoco aquí la correspondencia de niveles es perfecta ya que en en nivel de Interfaz no se definen cables ni conectores, etc.

A diferencia de otros SO tipo Unix, en Linux no se usan disposivos especiales tipo /dev/... sino que se crean interfaces dinámicamente mediante ifconfig.

En el núcleo de Linux se pueden incluir distintos tipos de interfaces o controladores. Algunos de ellos son:

- lo

Interfaz de bucle local (loopback). Se utiliza para pruebas aunque alguna aplicación puede necesitarla igualmente. No tiene sentido tener más de un interfaz de este tipo.

- eth

Es la interfaz genérica para tarjetas Ethernet de uso muy frecuente.

- sl

Interfaz SLIP para puerto serie. Montar una red mediante un puerto serie es poco eficiente pero la posibilidad existe.

- ppp

Interfaz PPP para modem.

- ippp

Interfaz IPPP para ISDN (RDSI).

- plip

Interfaz PLIP para puerto paralelo. Montar una red mediante un puerto paralelo es poco eficiente pero la posibilidad existe.

Para cada uno de estos controladores se requiere la correspondiente opción en en núcleo de Linux.

Modelo Cliente/Servidor

Cada servicio se instala asociado a un puerto y generalmente se usan los servicios más comunes en puertos previamente establecidos y universalmente conocidos. Cada servicio ha de tener un puerto distinto pero los servicios TCP y los UDP no entran en conflicto al usar el mismo puerto.

Los servicios en general (no solos los servicios de red) se pueden configurar en los SO tipo Unix de dos formas distintas.

- Modo Standalone: En este modo para que el servicio esté activo el servidor deberá estar permanentemente arrancado y escuchando su puerto para poder atender cualquier petición de servicio.
- Modo supeditado al servidor de servicios inetd:

  En este modo los servidores estarán parados y será inetd(8) quien estará permanentemente arrancado y a la escucha en todos los puertos. Para ello usará el fichero /etc/services que determina que puertos tiene que escuchar y que servicios están asociados a cada puerto. De esta forma en caso necesario arrancará un servidor de un servicio concreto cuando detecte una petición de servicio en su puerto asociado. Mire el contenido de /etc/services para saber como está configurado su inetd. Vease services(5).

En su sistema probalmente tendrá simultaneamente servicios activos en modo Standalone y otros supeditados mediante inetd. Esto depende del tipo de servicio y del tipo de uso que tenga. Está claro que un servicio utilizado de forma esporádica no es bueno tenerlo permanentemente arrancado consumiendo recursos mientras permanece inactivo la mayor parte del tiempo. Por el contrario un servicio que recibe un auténtico bombardeo de peticiones de servicios no es bueno que tenga que arrancar y parar a cada petición.

Si detiene el servidor de inetd, dejará a todos los servicios supeditados sin posibilidad de activarse. Los servicios se pueden asociar a puertos distintos de aquellos que por convenio se suelen usar pero si usa un servicio en un puerto distinto del preestablecido los clientes de ese servicio no podrán usarlo a no ser que antes averiguen el puerto.

Las redes en Linux utilizan por lo tanto son capaces de usar servicios siguiendo el modelo cliente/servidor mediante el cual uno de los procesos es capaz de aceptar determinado tipo de peticiones y generar determinado tipo de respuestas de acuerdo un protocolo preestablecido que establece

las reglas de comunicación entre ambos. Un servidor y un cliente son por tanto procesos independientes que generalmente aunque no necesariamente residirán en maquinas distintas.

Sustituir a un programa cliente que habla usando el protocolo SMTP con un servidor o a un programa cliente que habla POP3 con un servidor es muy fácil. SMTP usa el puerto 25 y POP3 usa el puerto 110. Si nos conectamos por telnet a un servidor usando alguno de estos puertos se iniciará una conversación de acuerto al protocolo y de esta forma mediante telnet podremos enviar correo mediante SMTP o bajar correo por POP3, o borrar un ménsaje en nuestro servidor de correo, etc. Basta para ello conocer el protocolo que en estos dos caso se limita a un reducido número de comandos sencillos.

Si su máquina tiene activo el servicio de SMTP puede probar lo siguiente.

telnet 127.0.0.1 25
HELP
HELO 127.0.0.1
QUIT

Lo que acaba de hacer es dialogar en SMTP con su servidor local de SMTP asociado al puerto 25. Esto es aplicable a muchos otros servicios.

El interfaz de llamada PPP

Para conectar a internet primero su modem telefoneará a su Proveedor de Servicios de Internet (ISP) y para establecer conexión se negocia la forma de enviar y recibir paquetes por la línea en función de las capacidades de ambos extremos. Es decir que tienen que ponerse de acuerdo en la velocidad a usar, forma de compresión de los datos y más cosas. Para ello a pppd, o en su caso a ipppd (el segundo es para RDSI) hay que pasarle algunos parámetros en forma de argumentos en la línea de comandos o bien a través de un fichero de configuración (/etc/ppp/options) y con estos datos la negociación se hace de forma automática y transparente. También se solicita la identificación del usuario que ha realizado la llamada. A no

ser que contrate la obtención de una dirección IP fija y reservada para usted obtendrá en cada conexión una dirección IP distinta. A la primera se la llama también dirección IP estática porque es siempre la mísma y a la segunda dirección IP dinámica. De todos estos detalles se ocupa el protocolo PPP (Point-to-Point Protocol)

El acceso a un ISP mediante modem merecería un capítulo entero pero suponemos que con lo explicado ya habrá comprendido que funciona como un interfaz de red ppp en su sistema en el cual pueden existir como ya hemos mencionado otros interfaces.

Una vez lograda la conexión dispondrá de un interfaz de red que le dará acceso a internet. Para comprobar si tiene conexión pruebe a hacer ping a un servidor conocido.

ping www.altavista.com

PROTOCOLOS TCP/IP

Introducción

Este capítulo es en realidad una segunda parte del anterior en el cual ya se comentaron algunas cosas. Se dijo que el protocolo TCP/IP es uno de los protocolos mas ampliamente usados en todo el mundo, y por esta razón merece la pena profundizar en este protocolo en particular.

OSI y TCP/IP son pilas de protocolos distintos y el intento bien intencionado que se hizo el el capítulo anterior solo era una t aproximación necesariamente imprecisa de ambos modelos dificilmente comparables. En el caso de TCP/IP es mejor utilizar OSI sólo como modelo de referencia teórico y estudiar en profundidad el verdadero modelo TCP/IP que es lo que haremos ahora.

Este capítulo contiene fragmentos de un anterior trabajo en el cual participó Luis Colorado Urcola y por ello aprovecho para agradecer su aportación. (En la actualidad desconozco su dirección de correo

electrónico) Conviene aclarar una serie de conceptos y empezaremos por explicar que es una dirección IP.

Direcciones ip

El principal beneficio de IP es que es capaz de convertir un conjunto de redes físicamente distintas en una sola red aparentemente homogénea. Una internet es una red IP aparentemente homogénea. La Internet (con mayúscula) es la red de redes. En realidad solo es una internet muy grande y que se exitende por todo el planeta. La característica de cualquier internet es que cada uno de los nodos tiene una dirección IP única y distinta a la de cualquier otro nodo. Las direcciones IP son cadenas de treinta y dos bits organizadas como una secuencia de cuatro bytes. Todas las tramas (paquetes) IP llevan una dirección de origen (donde se origino la trama) y una dirección destino (a donde va la misma). Básicamente esto es todo lo que hay que saber cuando se conecta uno a la red y la dirección IP te la asigna un administrador de red.

Estas direcciones tienen una representación como cuatro numeros enteros separados por puntos y en notacion decimal. Las direcciones representan el interface de conexión de un equipo con la red. Asi, un host que está conectado a varias redes como regla general, no tendrá una única dirección de red, sino varias (normalmente una por cada red a la que esta conectado). Pero internamente, Esto no es del todo cierto. Las direcciones IP se dividen en dos partes (cada una con un cierto numero de bits) cuyo significado tiene que ver con el sistema de enrutado de tramas. La primera parte (cuya longitud no es fija y depende de una serie de factores) representa la red, y debe ser igual para todos los hosts que esten conectados a una misma red física. La segunda parte representa el host, y debe ser diferente para todos los hosts que estan conectados a la misma red física.

El mecanismo de decisión de IP que hace que todas las tramas lleguen a su destino es el siguiente: Cuando la dirección origen y la dirección destino estan ambas en la misma red (esto se sabe por que su dirección de

red es igual en ambas, la dirección de red será la consecuencia de sustituir por ceros toda la parte de host en la dirección considerada) IP supone que existe un mecanismo de nivel inferior (en este caso Ethernet, Token Ring, etc.) que sabe como hacer llegar la trama hasta el host destino. Cuando la direcciónn de red origen y la de destino no coinciden, entonces hay que enrutar.

Para enrutar, se dispone de una tabla que contiene entradas para cada una de las redes a las que se quieren hacer llegar tramas, que no sean locales a este host (un host, en general, esta conectado a varias redes, de las que hace de gateway (pasarela), si la dirección destino de la trama tiene una dirección de red que coincide con alguna de las direcciones de red propias---las que resultan de sustituir por ceros la parte de host en cada uno de los interfaces---, entonces no hace falta enrutar). Esta tabla tiene más o menos entradas en función de la complejidad de una internet (o red de redes) y la dirección del siguiente host en el camino hasta la red de destino.

Por otro lado, la parte que corresponde a red y la parte que corresponde al host, se realiza usando este modelo (salvo para el subnetting, que añade algo de complejidad).

Clases de redes

Existen 5 tipos de direcciones IP. A cada tipo se le asigna una letra, asi, existen direcciones de clases A, B, C, D, E. Las direcciones pertenecen a estas clases en función de los cuatro bits más significativos del primer byte.

Clase A 7 Bits red + 24 bits host	0nnnnnnn	hhhhhhhh	hhhhhhhh	hhhhhhhh
Clase B	10nnnnnn	nnnnnnnn	hhhhhhhh	hhhhhhhh

14 bits red + 16 bits host				
Clase C 21 bits red + 8 bits host	110nnnnn	nnnnnnnn	nnnnnnnn	hhhhhhhh
Clase D (multicasting)	1110xxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
Clase E (reservadas)	1111xxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx

La **n** representa la parte de red de la dirección y **h** la parte de host. **x** tiene otro tratamiento.

Las clases D y E no participan en el sistema de enrutado IP y no las comentaremos (las direcciones de clase D se usan en multicasting y las direcciones de clase E estan reservadas, por ello no deben usarse para configurar hosts en internet)

(Como se puede ver hay muy pocas redes de clase A, que permiten tener muchos hosts por ejemplo ---InfoVía (Muchas redes de clase C para poquitos hosts)---Los centros proveedores de Internet; y un rango intermedio para redes que tengan ambos requisitos). Por otro lado, las redes de clase A tienen el primer byte como parte de red y los tres restantes como parte de host, las redes de clase B tienen los dos primeros bytes como parte de red y los dos últimos como parte de host y las redes de clase C tienen como parte de red los tres primeros bytes y como parte de host el último.

Así, puesto en notación punto, las redes de las distintas clases cubren los siguientes rangos de direcciones:

clase A de la 1.0.0.0 --> 127.255.255.255

clase B de la 128.0.0.0 --> 191.255.255.255
clase C de la 192.0.0.0 --> 223.255.255.255

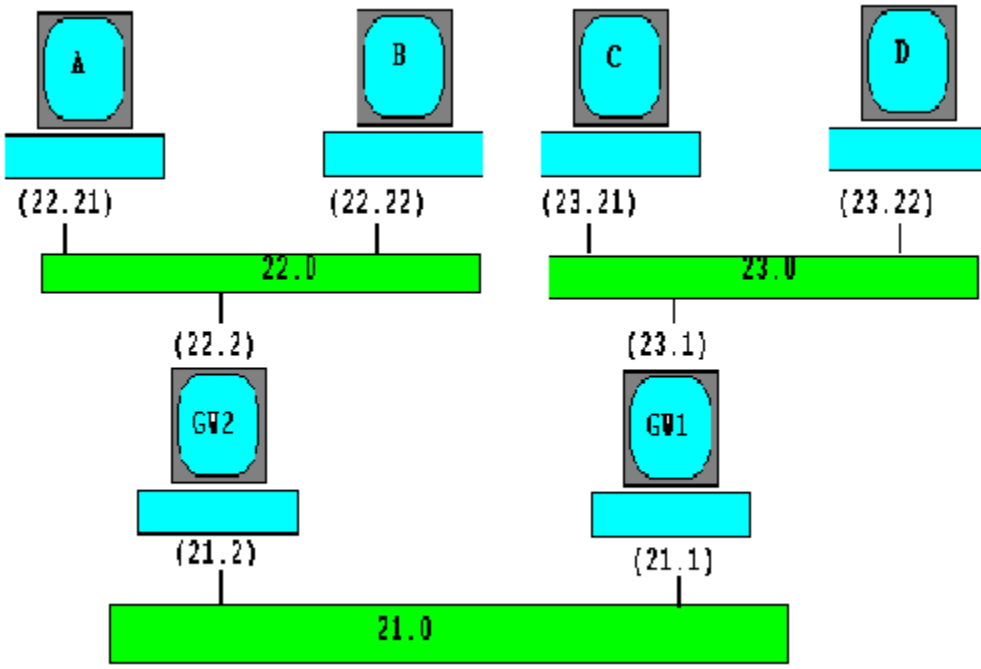
Ejemplo de una pequeña red

Vamos a explicar como construiriamos por ejemplo tres redes pequeñas interconectadas entre sí y que direcciones podriamos usar para ello. De esta forma podremos explicar como funciona el enrutado.

Usaremos un ejemplo sencillo para explicar esto. Existen una serie de direcciones especialmente reservadas para usos privados. Es decir no se usaran en una red pública de internet.

Direcciones reservadas para la clase A 10.0.0.0
Direcciones reservadas para la clase B 172.16.0.0 --> 172.31.0.0
Direcciones reservadas para la clase C 192.168.0.0 --> 192.168.255.0

Para nuestro ejemplo usaremos tres redes de clase B de tipo 172.22.n.n. En todas las redes (incluidas las que tienen subnetting) existen dos direcciones de host que están reservadas. La primera es la que tiene todos los bits correspondientes a la parte de host a cero. Esta dirección se utiliza para representar a la propia red y por tanto no se debe asignar a ningun host. La otra es la que tiene todos los bits puestos a uno, y representa a todos los hosts que estan conectados a una misma red. En redes donde se permite el broadcast (donde un host puede hacer que todos reciban el mensaje que ha enviado) esta dirección se utiliza para este fin. Estas direcciones se llaman direcciones de red y de broadcast respectivamente. También está reservada la dirección IP que tiene todos los bits a cero en la parte de red para indicar esta red, aunque esta solo está; permitido usarla cuando aun no conocemos nuestra dirección IP completamente (por ejemplo cuando hay que adquirirla por la propia red, en el arranque, al configurar, etc.)



La configuración de los puestos A, B, C, D sería la siguiente:

	A	B	C	D
IP.ADR	172.22.22.21	172.22.22.22	172.22.23.21	172.22.23.22
N.MASK	255.255.255.0	255.255.255.0	255.255.255.0	255.255.255.0
BROADK	0.0.0.255	0.0.0.255	0.0.0.255	0.0.0.255
NETWRK	172.22.22.0	172.22.22.0	172.22.23.0	172.22.23.0

Hay dos gateways, con dos direcciones cada uno, y su configuracíon sería:

	GW1	GW2
IP.Addr	172.22.22.2/172.22.21.2	172.22.23.1/172.22.21.1

N.Mask	255.255.255.0(ambas)	255.255.255.0(ambas)
Broadc	172.22.22.255/172.22.21.255	172.22.23.255/172.22.21.255
NetAddr	172.22.22.0/172.22.21.0	172.22.23.0/172.22.21.0

Según lo que hemos explicado, la red 172.22.0.0 es una red clase B y por tanto su máscara por defecto es la 255.255.0.0. En el ejemplo está subneteada con máscara 255.255.255.0 y dado que esta máscara no es la máscara por defecto la red ha de indicarse por su dirección IP seguida de su máscara. Estas tres redes por tanto son:

172.22.22.0 / 255.255.255.0
172.22.23.0 / 255.255.255.0
172.22.21.0 / 255.255.255.0

Máscaras de red

La red del ejemplo está formada por tres subredes de la red de clase B 172.22.0.0 con máscaras 255.255.255.0 (o sea, consideradas como o si la red 172.22.0.0 se hubiera subneteado)

La máscara de red no es más que una dirección IP donde se ha sustituido todos los bits de la parte de red de la dirección por unos y los bits correspondientes a la parte de host por ceros. Así, la máscara de red de una red de clase A será 255.0.0.0, la de una red de clase B será 255.255.0.0 y la de una clase C será 255.255.255.0.

¿Por que se usa entonces la máscara de red si está implícita en el tipo de direccion?

Se ha comprobado que a veces el sistema de clases no es apropiado. A un proveedor le dan una clase C completa pero el quiere dividirla en cuatro redes diferentes, y no va a usar cuatro clases C para esto 1000 direcciones IP, porque se malgastarían muchas. Pues bien, dentro de una misma red, se puede extender el mecanismo de routing, considerando que la parte de

host son los bits cero y la parte de red son los bits uno de la máscara y asociando a cada dirección IP una máscara en el momento de configurarla (por supuesto, los valores por defecto serán los de la clase de la red, aunque se podrán añadir, y solamente añadir, bits uno a la máscara, con el fin de subnetearla.

En el caso del ejemplo de este parrafo, el proveedor podría extender la máscara de subred dos bits más allá de la frontera impuesta por el tipo de la dirección adquirida y considerar que tiene cuatro redes (en este caso la parte de red serían los primeros tres bytes y dos bits del cuarto y la parte del host los restantes).

Tablas de rutas

Por ejemplo partiendo de la red 21.0 para ir a la 23.0 habría que usar el GW1, y para ir de la 23.0 a la 22.0 habría que usar GW1 y GW2.

¿Pero como se indica la ruta a tomar para cada destino?

Para ello se usan las tablas de rutas y para construirlas, formaremos para cada máquina, una tabla con la forma de dirigir las tramas hacia las redes que no vemos directamente, con la ventaja adicional de que como la dirección 0.0.0.0 representa obviamente la red local, podemos usar esta dirección especial para indicar la ruta por defecto (es decir, la dirección local a la que enviar las tramas que no tienen ruta y que necesariamente hay que enrutar). Así, las tablas de rutas quedaran:

```
Host A: 0.0.0.0(default) ---> 172.22.22.2
Host B: 0.0.0.0(default) ---> 172.22.22.2
Host C: 0.0.0.0(default) ---> 172.22.23.1
Host D: 0.0.0.0(default) ---> 172.22.23.1
GW1:    172.22.22.0      ---> 172.22.21.2
GW2:    172.22.23.0      ---> 172.22.21.1
```

Si nos fijamos, podemos ver que la ruta por defecto es el gateway por defecto que aparece en la configuración de muchas máquinas Linux, pero para los gateways no aparece ningun gateway por defecto. Esto es una característica de las redes que tienen backbones.

Backbon

Literalmente, backbone significa columna vertebral. Las tramas se canalizan hacia el backbone, pero este, al final debe decidir a donde van a parar las tramas. Como conclusión de esta última frase se desprende que en todo backbone (red de nivel jerárquico superior donde no hay ninguna red mas importante. p.ej. el backbone de Internet, o la red principal de una empresa que no tiene conexiones con el exterior) los gateways que forman el backbone deben tener rutas a todas las redes y no aparecer una ruta por defecto en sus tablas de rutas, mientras que para el resto de las redes, la ruta por defecto (y para los hosts) será siempre la que lleve hacia el backbon.

Configuración de un puesto

Una vez comprendidos los conceptos pasemos a calcular las direcciones. Con la dirección IP y la máscara de red se pueden obtener las demás. Tomando como ejemplo el nodo A observamos que se cumplen las siguientes relaciones lógicas usando los operadores lógicos (AND, NOT y OR) sobre los bits que los componen:

Máscara de Red	255.255.255.000
Dirección IP	172.022.022.021
	AND
Dirección de Red	172.022.022.000
NOT (Máscara de Red)	000.000.000.255
Dirección de Red	172.022.022.021
	OR
Dirección de Broadcast	172.022.022.000

Observe que con la dirección IP y la máscara de red se puede deducir las restantes direcciones. Por lo tanto si tiene que conectarse a una red local bastará con preguntar al administrador de la misma esos dos únicos datos.

Criterios para subnetear una red

Vamos a comentar los criterios que se pueden tener en cuenta a la hora de decidir subnetear una gran red en varias más pequeñas.

Subnetear una red no sirve unicamente a criterios topológicos relacionados con la instalación física de la red. Está claro que la redes

topológicamente distintas o que usen diferentes protocolos de transmisión, deberían tener direcciones distintas para no mezclar artificialmente cosas distintas pero también se puede dividir una gran red en varias por otros motivos. Por ejemplo para facilitar la administración de la red delegando a cada administrador la gestión de direcciones de una subred. Quizás las estructura de una empresa aconseje separar redes disintas para distintos departamentos por ejemplo. Es decir las subredes pueden facilitar la adaptación de la red a la estructura de una organización. También pueden servir para aislar redes con tráfico interno abundante y facilitar el diagnóstico de problemas en la red.

En tehoría se puede dividir una sola red física en varias lógicas pero si la separación física fuera posible conviene hacerla uniendolas con un gateway porque aumentaremos el ancho de banda total en la red. Hay que tener en cuenta que una red ethernet es un bus con aceso Csma/CD y por lo tanto en un instante de tiempo dado, varios puestos pueden escuchar simultaneamente pero nunca pueden hablar dos usando el mismo bus a la vez porque origina la colisión de los paquetes y estos deberán ser reenviados de forma no simultánea.

Los términos Pasarelas (Gateways), Routers (Routers) y Puentes (Bridges) se refieren a una serie de dispositivos para interconectar redes. Muchas veces estos dispositivos son un PC con un par de tarjetas de comunicaciones para conectarse simultaneamente a dos redes funcionando con un software apropiado. Un puente conecta dos redes parecidas (por ejemplo pueden ser distintas en su velocidad) sin cambiar la estructura de los mensajes que circulan por ellas. Un Router conecta dos redes que son iguales (o parecidas). No hay traducción de un protocolo a otro pero si traducción de direcciones. Una pasarela puede conectar redes distintas (por ejemplo una ethernet y una token ring) realiza la conversión de protocolos y la traducción de direcciones. Si recordamos los modelos OSI y TCP/IP explicados anteriormente vemos que en los distintos dispositivos mencionados actuarían a distinto nivel del modelo de comunicaciones.

Puesto que un router incluye la función de puente, y la pasarela engloba las funciones de router y de puente. ¿Porque entonces no se usa siempre una pasarela (gateway)?

Cuando todos los nodos de una red deben alcanzar todos los nodos de otra red, se provoca una sobre carga de la red debido a los protocolos de rutado. En este caso y cuando ambas redes utilizen el mismo protocolo de red, la mejor solucion es un puente.

Existe otro tipo de dispositivo que sirve para conectar dos redes y que se llama Firewall. Se trata de un dispositivo lógico que sirve para conectar de forma segura la parte privada de una red con la parte pública. Estos dispositivos pueden establecer una política de restricciones a la información que circula por ellos.

Para determinar una ruta como la mas idonea, se utiliza el siguiente criterio. El protocolo de información de rutado (RIP) es un protocolo que está basado unicamente en el coste.

A cada ruta se le asocia un coste en función del rendimiento de la red, tipo de la linea, etc. Esto permite determinar de entre varias rutas posibles cual es la de menor coste.
Cada router de una red envía y recibe información de rutado que permite actualizar las tablas de rutado cada 30 segundos. Cuando el coste de una ruta supera el valor 16 el sistema se considera fuera de alcance. La caida de un nodo de la red suele provocar la perdida de algunos mensajes hasta que las tablas de rutado de todos los routers se reajustan.

Servidores de Nombres

Las direcciones IP son muy adecuadas para las máquinas pero son dificiles de recordar. Por esta razón se usan nombres de dominios. Además usando solo una dirección IP sin nombre tendríamos que cambiar de dirección cada vez que nos alojemos en un servidor distinto. Si asimilamos una dirección a un nombre podemos mantener nuestra presencia en internet cambiando de servidor sin que apenas se note el

cambio. En la práctica el cambio no sucede instantaneamente porque tiene que propagarse la información del cambio de nombre en gran parte de la red.

Para una red pequeña nos bastaría usar un fichero /etc/hosts que sea común a todos los nodos de la red. Si la red es más grande cada cambio ha de ser propagado a los demás lo antes posible y controlar que la información se mantiene coerente o usar un sistema de base de datos compartido como NIS, etc.

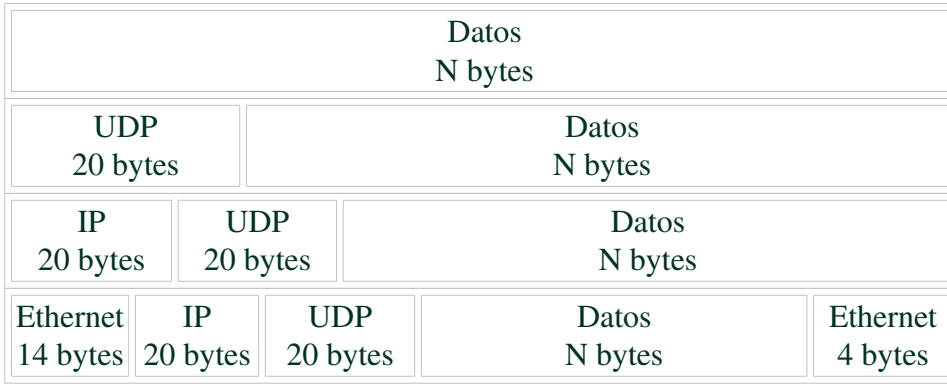
A pesar de ello todo esto resulta insuficiente cuando hablamos de redes muy grandes. El fichero /etc/hosts o equivalente sería enorme asi que lo que se ha implementado para que en internet exista un servicio de nombre es que es un protocolo llamado DNS que permite tener esta información distribuida adecuadamente mediante un sistema de dominios que se reparten la carga de gestionar estos nombres organizados de forma jerárquica generalmente hasta tres niveles de los cuales el primero suele indicar el tipo de actividad. Por ejemplo:

- com: Empresas comerciales
- org: Organizaciones no comerciales.
- edu: Universidades y centros de investigación.
- net: Pasarelas y nodos administrativos de la red.

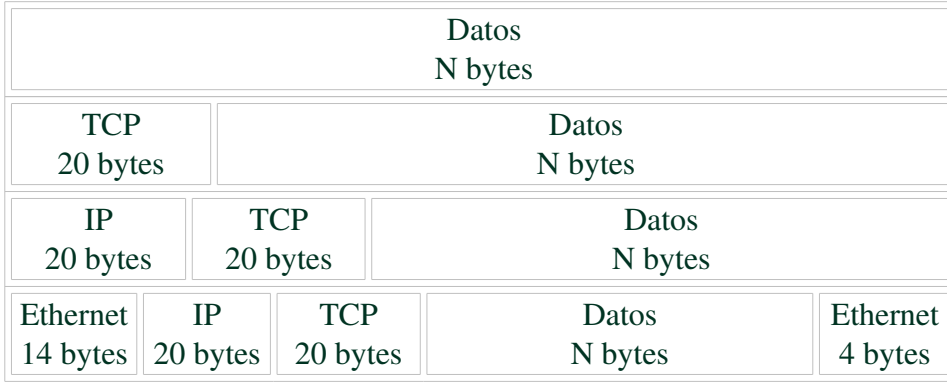
Con el fin de aumentar la eficiencia y la tolerancia a fallos cada zona tiene un servidor de nombres autorizado o maestro y cada uno de estos tiene varios servidores secundarios. Las zonas se definen como superconjuntos de redes IP redondeadas a nivel de un octeto.

Estructura de paquetes

Los paquetes o tramas son secuencias de bits que contienen no solo los datos con información de usuario sino datos de cabecera y de cola que se estructuran de forma que los niveles inferiores de transmisión de datos van encapsulando a los niveles superiores. Para formar un paquete UDP habría que envolverlo de la forma siguiente.



La parte que representamos a la izquierda de los datos son datos de cabecera, y los de la derecha son datos de finalización del paquete. Únicamente el nivel de Interfaz Ethernet tiene parte de finalización, y todos los niveles tienen datos de cabecera. Para el caso de TCP la estructura será similar.



La parte de datos puede tener distinto tamaño según los casos pero generalmente no será mucho mayor de 1KByte.

Estudiar en detalle la estructura de estos paquetes excede las pretensiones de este curso a pesar de ello conviene ilustrar con un par de ejemplos lo que serían un par de cabeceras y para ello usaremos la estructura de una cabecera IP y de una cabecera TCP. Ambas son estructuras de 20 Bytes.

Cabecera IP (20 bytes)																																			
Versión				IHL				Tipo de Servic.								Tamaño Total																					
Identificación																Flags				Desplazamiento del Fragmento															
Tiempo de Vida								Protocolo								Checksum de Cabecera																			
Dirección de Origen																																			
Dirección de Destino																																			

Simplemente nos conformaremos con resal tar que la cabecera IP tiene datos de identificación y direccionamiento y la cabecera TCP contiene información de los puertos así como indicadores de control, numero de secuencia de los paquetes, etc. para detectar la perdida de paquetes o su deterioro a fin de gestionar el reenvió de paquetes en caso necesario.

Cabecera TCP (20 bytes)																																			
Puerto de Origen																Puerto destino																				
Número de secuencia																																			
Número de Confirmación																																			
Desplz. de los datos				Reservado								URG	ACK	PSH	RSI	SYN	FIN	Ventana																		
Checksum																																			
Puntero de Urgencia																																			

En Internet un destino y un origen no implica una ruta fija. Los mensajes buscan su ruta pasando a traves de varios nodos y dicha ruta puede variar incluso dentro de una misma sesión o de una misma transferencia de ficheros. Por ejemplo la caida en una ruta supondría la perdida de algunos paquetes pero no afectaría a la transmision ya que esta continuaría rápidamente por otra ruta sin que apenas podamos percibirlo.

Recordemos que Internet proviene en sus origenes de diseños con objetivos militares.

Si hace un ping a una máquina remota recibirá una información util para diagnosticar la calidad de su conexion con ese servidor. Se trata de un servicio muy corriente orientado a datagramas.

Primero le irá mostrando una serie de líneas donde podrá comprobar el tiempo de respuesta expresado en milisegundos. Si interrumpe su ejecución mostrará unas estadísticas donde podrá ver entre otras cosas el porcentage de paquetes perdidos.

En este momento podemos entender mejor cual es la diferencia entre un repetidor (repeater), un puente (bridge), un enrutador (router), una pasarela (gateway), y un cortafuegos (FireWall). Salvo los repetidores y en general, estos dispositivos pueden ser un PC con un par de tarjetas de red conectada cada una de ellas a una red distinta.

Representaremos en azul claro la parte de las tramas que pueden ser afectadas o solo tenidas en cuenta. **REPETIDOR *(REPEATERS)***

Un repetidor solo amplifica y retoca la señal a nivel puramente eléctrico. Por lo tanto ignora cualquier cosa sobre la información transmitida.

Ethernet 14 bytes	IP 20 bytes	UDP 20 bytes	Datos N bytes	Ethernet 4 bytes
Ethernet 14 bytes	IP 20 bytes	TCP 20 bytes	Datos N bytes	Ethernet 4 bytes

PUENTE *(BRIDGE)*

Un puente actúa traduciendo o controlando la circulación de paquetes a nivel de Interfaz de red. En nuestro caso es la capa Ethernet. Ambas redes deberán ser muy parecidas ya que cualquier diferencia en los protocolos superiores haría imposible la conexion de ambas redes. En la práctica pueden permitir conectar redes con diferentes velocidades, pero ambas redes físicas se comportarían como una misma red lógica.

Ethernet 14 bytes	IP 20 bytes	UDP 20 bytes	Datos N bytes	Ethernet 4 bytes
Ethernet 14 bytes	IP 20 bytes	TCP 20 bytes	Datos N bytes	Ethernet 4 bytes

ENRUTADOR *(ROUTER)*

Un Router actua traduciendo o controlando la circulación de paquetes a nivel de red. Podemos ver en el dibujo siguiente como un router puede actuar a nivel de Interfaz de red (En nuestro caso Ethernet) y a nivel de red (en nuestro caso capa IP).

Ethernet 14 bytes	IP 20 bytes	UDP 20 bytes	Datos N bytes	Ethernet 4 bytes
Ethernet 14 bytes	IP 20 bytes	TCP 20 bytes	Datos N bytes	Ethernet 4 bytes

PASARELA *(GATEWAY)*

Una pasarela (Gateway) actuará a nivel de transporte que en nuestro caso puede ser TCP o UDP. También engloba los niveles inferiores.

Ethernet 14 bytes	IP 20 bytes	UDP 20 bytes	Datos N bytes	Ethernet 4 bytes
Ethernet 14 bytes	IP 20 bytes	TCP 20 bytes	Datos N bytes	Ethernet 4 bytes

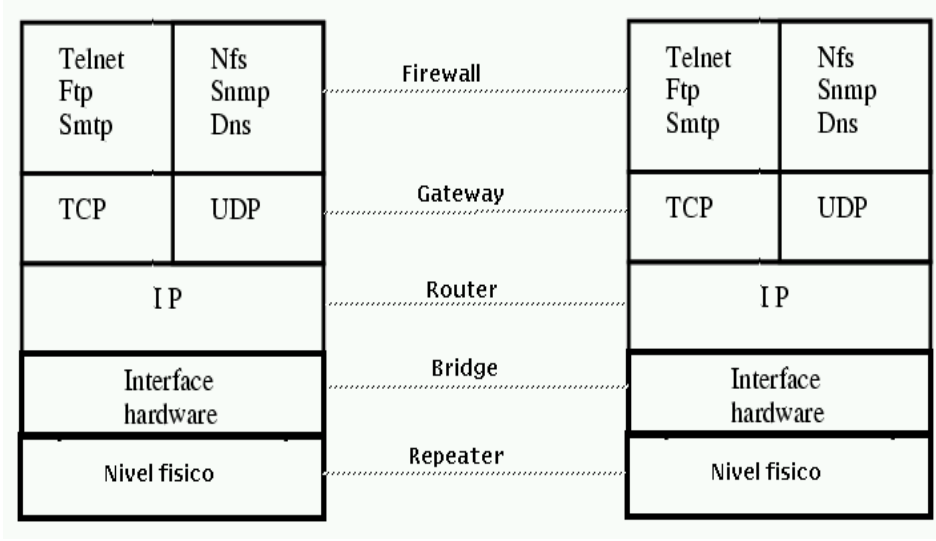
CORTAFUEGOS *(FIREWALL)*

Un CortaFuegos (FireWall) puede actuar permitiendo o denegando el paso de paquetes considerandolos en su totalidad permitiendo el más amplio control posible. Uno de los usos más comunes es el de proteger una intranet de accesos indeseados en ambos sentidos con el exterior

especificando una políticas restrictivas a nivel de puertos, servicios, direcciones IP, o cualquier otra cosa.

Ethernet 14 bytes	IP 20 bytes	UDP 20 bytes	Datos N bytes	Ethernet 4 bytes
Ethernet 14 bytes	IP 20 bytes	TCP 20 bytes	Datos N bytes	Ethernet 4 bytes

En la siguiente ilustración se muestra a que nivel se comunican dos equipos intercomunicados por uno de estos dispositivos



EL ARRANQUE EN LINUX Y COMO SOLUCIONAR SUS PROBLEMAS (Primera Parte)

Introducción

En este capítulo intentaremos describir las posibles causas por las cuales un sistema Linux puede tener problemas para arrancar o para funcionar correctamente y la forma de solucionarlo.

Estos problemas pueden aparecer durante la instalación con relativa frecuencia pero también pueden presentarse por distintas causas en un sistema que lleva tiempo funcionando. En estos últimos casos suele tratarse de un descuido cometido desde root o bien por un fallo del hardware. Algunas veces se convierten en auténticas pesadillas para el usuario que las padece.

Para defenderse de todo esto tenemos que aprender bastantes cosas sobre el sistema y su funcionamiento. Trataremos primero de describir el proceso de arranque.

Este tema ha sido dividido en dos partes. La primera describe principalmente los diferentes cargadores de arranque, y la segunda el proceso de inicialización.

Cargadores de arranque en general

Un cargador de arranque es un software encargado de cargar en memoria el sistema operativo. Se trata de algo curioso ya que bien mirado es un software encargado de cargar en memoria otro software. Pero entonces el astuto lector se preguntará quien carga al cargador ? La respuesta es muy sencilla. La BIOS. (Basic Input Output Sistem). Es un software que reside en una memoria especial no volátil del ordenador. No volátil quiere decir

que se conserva incluso no el ordenador apagado. Esta memoria es de solo lectura. Denominada por ello ROM (Read Only Memory) por contraposición a la RAM que es memoria totalmente volátil.

No se pretende explicar el hardware del PC pero para no dejar esto incompleto mencionaremos que existe otro componente hardware que a diferencia de la ROM y de la RAM admite la alteración de ciertos valores de configuración que no se perderán cuando el ordenador se apague, aunque para ello se recurre a una pequeña pila. Estamos hablando de la CMOS.

La BIOS espera encontrar un cargador en algún dispositivo. Conviene tener presente que la CMOS debe configurarse de modo que intente arrancar explorando los distintos tipos de dispositivos, (disquete, cdrom, o disco duro, etc en la secuencia que a nosotros nos interese).

Existen distintas alternativas para arrancar en Linux. Nosotros comentaremos Lilo, Loadlin, y Grub. No se puede afirmar tajantemente que uno sea superior a otro. Son muy distintos en su filosofía y ello presenta ventajas e inconvenientes para cada uno dependiendo de las circunstancias. Todos ellos como paso final pasan el control al sistema operativo. Algunos cargadores ofrecen la posibilidad de elegir arrancar con distintos sistemas operativos.

La BIOS deberá cargar una sección especial del disco o disquete que no se localiza en ninguna partición pues se encuentra en una pequeña localización previa a la particiones. Esta sección es el registro maestro de arranque abreviadamente MBR (Master Boot Record). Una vez cargado el MBR ya se puede localizar la partición donde reside el kernel o núcleo del sistema operativo y cargarlo. También podrá localizar la partición donde reside el sistema de ficheros raíz que no tiene porque ser necesariamente la misma.

Un disquete no posee particiones. Únicamente dispone de un sector de arranque de 512 bytes y una zona de datos.

Un disco duro tiene un MBR y hasta un máximo de cuatro particiones primarias. Si se necesitan más particiones se toma una de las particiones primarias y se subdivide en particiones lógicas pero entonces la partición primaria tratada de esta forma pasa a llamarse partición extendida.

Resumiendo todo el proceso de arranque sería algo así:

1. La señal eléctrica de reset o la de establecimiento de la alimentación eléctrica instruyen a la CPU para ejecutar las instrucciones de la BIOS desde su punto inicial.
2. La BIOS basándose en la configuración de la CMOS localiza el dispositivo de arranque y carga el MBR.
3. El MBR pasa el control al sector de arranque en la la partición adecuada.
4. El sector de arranque carga el SO y le cede el control.

Discos y particiones

Recordamos la nomenclatura de los discos y las particiones en Linux.

(/dev/hd[a-h] para discos IDE, /dev/sd[a-p] para discos SCSI, /dev/ed[a-d] para discos ESDI, /dev/xd[ab] para discos XT, /dev/fd[0-1] para disquetes).

Estos nombres de dispositivo se refieren siempre al disco entero. El dispositivo puede ser un disco duro no removible, un disco duro removible, una unidad lectora o lectora/grabadora de cdrom, o de cdrom/dvd, unidades zip, u otras cosas.

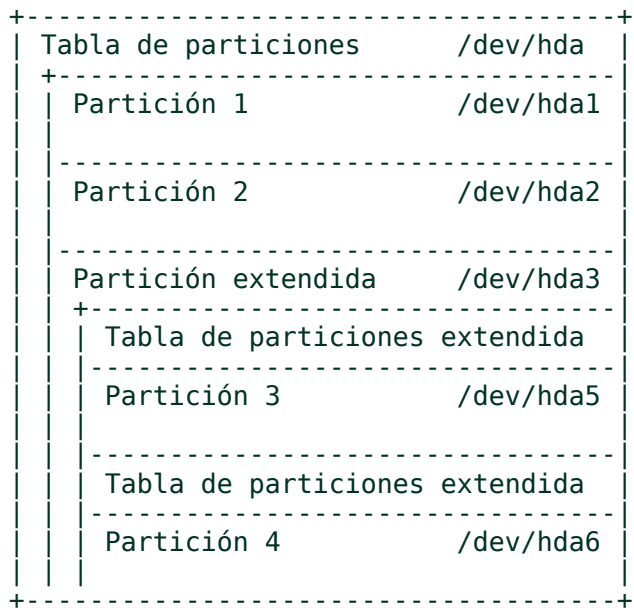
Imaginemos un PC con tres discos duros IDE, dos discos SCSI, una disquetera y un lector de CDROM IDE conectado al controlador IDE primario esclavo

```
/dev/hda    controlador IDE  primario  maestro
/dev/hdb    controlador IDE  primario  esclavo
```

```
/dev/hdc    controlador IDE secundario maestro
/dev/hdd    controlador IDE secundario esclavo
/dev/sda    primer dispositivo SCSI.
/dev/sdb    segundo dispositivo SCSI
/dev/fd0    disquetera
/dev/cdrom  -> /dev/hdb (link simbólico para unidad
cdrom conectado al
 controlador IDE primario  esclavo
```

Como máximo cada disco podrá tener cuatro particiones primarias Para un primer disco IDE (/dev/hda) las particiones primarias serían /dev/hda1 ... /dev/hda4. Las particiones lógicas se nombran siempre a partir de la /dev/hda5 en adelante.

Por ejemplo vamos a suponer que deseamos disponer de cuatro particiones distintas. Vamos a suponer que usamos 2 particiones primarias y una tercera extendida sobre la cual se han definido 2 particiones lógicas el resultado sería el siguiente.



Es decir tenemos /dev/hda1, /dev/hda2, /dev/hda5, y /dev/hda6. La partición /dev/hda3 extendida no se usa directamente. Es un mero contenedor de particiones lógicas. (No se debe formatear).

Seguramente le resulte extraño porque en los SO operativos de Microsoft no se suele usar más de una partición primaria. Realmente se podría haber usado una partición primaria y una extendida con tres unidades lógicas. Es decir podríamos haber usado /dev/hda1, /dev/hda5, /dev/hda6, y /dev/hda7 dejando /dev/hda2 como partición extendida.

También podríamos haber usado las cuatro particiones como particiones primarias /dev/hda1, /dev/hda2, /dev/hda3, /dev/hda4.

Para ver cual es la situación actual en su PC. Haga lo siguiente:

```
$
fdisk -l
```

Conviene sacar esta información por impresora y guardarla. Si accidentalmente se pierde la información de la tabla de particiones, por ejemplo por escritura accidental en el primer sector de un disco podría volver a recuperarla usando ese listado. También se puede guardar la información del primer sector de un disco duro salvándolo a disquete mediante:

```
$  dd if=/dev/hda of=/dev/fd0 count=1 bs=512
```

Para recuperar este sector basta hacer:

```
$  dd of=/dev/hda if=/dev/fd0 count=1 bs=512
```

Puede valer si solo usa particiones primarias, pero esto no salva la información de las particiones lógicas. Lo mejor es conservar la salida de

'fdisk -l'. No cuesta nada volver a particionar todo igual que estaba nuevamente usando fdisk.

Compartir el arranque con otro SO

Instalar Linux en un disco que ya tiene otro SO tipo Windows. Daremos una receta muy simple suponiendo que parte de un sistema con un solo disco duro y con una sola partición que ya tiene otro SO tipo Windows.

1. Desinstale antivirus y otros programas que pueda residir en posiciones fijas al final del disco duro.
2. Desfragmentar el disco duro con la opción de máxima descompresión.
3. Asegúrese de que toda la información queda agrupada y tome nota de cuanto espacio queda libre.
4. Dividir la partición con fips cortando por una parte donde no exista información.
5. Eliminar la segunda partición. Puede hacerlo con el propio fdisk de Linux.
6. Añadir una partición pequeña para swap. y el resto la para la futura partición raíz de Linux. Para la swap puede usar un tamaño de unos 150Mbytes.
7. Por último instale Linux.

Lilo

Lilo es probablemente el cargador más utilizado en Linux pero se puede usar para cargar una gran variedad de sistemas operativos.

El cargador Lilo puede ubicarse en distintos lugares.

- Sector de arranque de un disquete Linux. boot=(/dev/fd0, ...)
- MBR de la primera unidad de disco duro. boot=(/dev/hda, /dev/sda, ...)

- Sector de arranque del primer sistema de ficheros Linux en el primer disco duro. boot=(/dev/hda1, ...)

Lilo también permite arrancar desde una partición extendida aunque no es frecuente tener que recurrir a esto.

No debe confundir /dev/hda y /dev/hda1. Como acabamos de explicar son cosas bien distintas.

Si usted tiene un SO distinto de Linux en /dev/hda1 y usa boot=/dev/hda1 deteriorará el sistema de arranque de ese otro SO. En MSDOS se puede regenerar esa pérdida mediante la transferencia de los ficheros del sistema de arranque desde un disquete del sistema mediante el comando sys.

Si utilizo boot=/dev/hda Lilo se grabará en el MBR. Si desea recuperar el MBR de MSDOS o Windows use desde un disquete de arranque que contenga el comando fdisk de MSDOS el comando fdisk /MBR. Se trata de un comando no documentado de MSDOS.

Por lo tanto cada vez que usted ejecuta el comando Lilo de Linux puede estar afectando el proceso de arranque de otros sistemas operativos instalados en su máquina. A Microsoft no le gusta compartir demasiado su sistema de arranque con otros sistemas operativos. Durante la instalación de Windows no se respeta el MBR y por ello si debe instalar Windows y Linux conviene instalar Linux en segundo lugar. Para ello deje una parte del disco libre con suficiente espacio o reserve un disco completo para su Linux.

Lilo se configura mediante el fichero /etc/lilo.conf. Se muestra un ejemplo de configuración básica. Consta de tres partes:

- Sección general o global.
- Sección de entradas Linux (image = <kernel>)
- Sección de entradas no Linux. (other = <particion de arranque>)

Existe una parte general previa y luego para cada opción de arranque etiquetada con un identificador label = <etiqueta>. Para cada opción de arranque se puede definir cosas nuevas o redefinir ciertas cosas previamente definidas en la parte general.

```
boot = /dev/hda
delay = 40
vga = normal
root = /dev/hda1
read-only
image = /zImage-2.5.99
        label = kern2.5.99
image = /zImage-1.0.9
        label = kern1.0.9
image = /boot/vmlinuz
     label = linux
     vga = ask
     root = /dev/hdb2
other = /dev/hda3
     label = dos
     table = /dev/hda
```

Salvo que se índice cual es la entrada por defecto con la opción default = , la primera entrada se considera la entrada predeterminada por defecto y en caso de no intervención el sistema arrancará usando esta entrada. Antes de eso esperará un tiempo (delay = <décimas de segundo>) para dar tiempo a intervenir y elegir el arranque deseado. Si no se recuerda cuales son las opciones para poder arrancar basta usar la tecla de cambio a mayúsculas <Shift> y se mostrarán la lista de las distintas opciones disponibles. También se puede usar para lo mismo la combinación de las teclas <Alt>+<Ctrl>. Si se quiere forzar la aparición de esta lista sin tener que usar las teclas usaremos la opción prompt.

El kernel a usar se establece mediante image = <fichero del kernel> La variable root permite especificar cual será la partición raíz. De no existir esta variable lilo tomará el valor determinado en el propio kernel, lo cual puede alterarse con la utilidad rdev.

En nuestro ejemplo hemos definido una partición raíz para los sistemas linux mediante el uso de root = /dev/hda1 en la sección global pero esto se puede alterar y la entrada label = linux usará root = /dev/hdb2 y vga = ask. Esto último permite elegir durante el arranque el modo de pantalla a usar. La opción read-only hace que el sistema de ficheros raíz se arranque inicialmente en modo de solo lectura. Esto es lo normal pero posteriormente durante el arranque cambiara a modo lectura-escritura.

Se puede incluir un mensaje durante el arranque mediante message = <fichero>. La opción compact permite una carga más rápida y es recomendable por ello su uso en disquetes (boot = /dev/fd0) pero puede ser incompatible con la opción linear. Bueno hay muchas opciones más y conviene mirar la documentación de lilo.

Una vez editado el fichero 'lilo.conf' hay que ejecutar el comando 'lilo' para instalar el cargador de arranque, que se activará la próxima vez que se arranque el sistema. El más mínimo cambio que afecte a alguno de los ficheros que intervienen en el arranque hará que este falle. Por ejemplo si saca una copia del kernel, borra el original y renombra la copia con el mismo nombre que el original todo parecerá que está igual que antes pero las posiciones en disco ya no son las mismas y el arranque fallará. Para evitar esto después de la recompilación de un kernel o de cualquier otro cambio en cualquiera de los elementos involucrados durante el arranque deberá reconstruir el cargador volviendo a ejecutar el comando lilo.

LILO puede informar de la limitación de la BIOS para acceder por encima del cilindro 1024. Si la partición de arranque supera este límite cabe la posibilidad de que el kernel u otro elemento necesario durante el arranque resida por encima de ese límite y resulte inaccesible para Lilo impidiendo el arranque del sistema. La opción linear puede evitar este problema.

Como norma de precaución deje siempre una entrada con el último kernel de funcionamiento comprobado para poder arrancar desde él en caso de que el nuevo kernel generado falle.

Si usted no puede arrancar porque olvidó ejecutar lilo después de algún cambio, deberá arrancar desde un disquete y recuperar el arranque. Vamos a suponer que su sistema usaba la partición raíz montada en /dev/hda1. (Es una suposición nada más). En ese caso y si dispone de un disquete de arranque que ya esté preparado para arrancar usando la partición raíz /dev/hda1 el problema será fácil de solucionar. Bastará ejecutar 'lilo' una vez que arranque desde disquete. Un disquete así puede obtenerse de varias formas. Una de ellas es si usted compiló el kernel usando 'make zimage'. (Veremos en posteriores capítulos la compilación del kernel). También podemos usar un disquete o CD de rescate que arranque con una partición raíz cualquiera siempre que el kernel utilizado sea compatible con nuestro sistema. En cualquier caso se puede siempre intentar. Una vez arrancado con disquete de rescate creamos un directorio para montar en él nuestra partición raíz.

```
mkdir /hda1
cd /hda1
mount -t ext2 /dev/hda1 /hda1
chroot /hda1
lilo
shutdown -r now
```

La clave de este procedimiento está en 'chroot'. Se trata de un comando que obtiene una sesión bash o lanza un comando cambiando el directorio raíz actual por uno diferente. Evidentemente el nuevo directorio debe de tener la estructura adecuada y los ficheros adecuados a cualquier directorio raíz que se precie de serlo. El directorio raíz es la caja donde ocurre cualquier cosa en el sistema. Hacer 'chroot' equivale a cambiar de caja. Fuera de la nueva caja es como si nada existiera para nuestro SO. Nosotros lo usamos para que lilo funcione desde la partición raíz que nos interesa.

También puede obtener un disquete de arranque que use la partición raíz / dev/hda1. Introduzca un disquete virgen formateado en la disquetera. Tendrá que saber donde se encuentra su kernel. Puede mirar el fichero

/etc/lilo.conf para recordarlo. Este es el fichero de configuración de lilo. Vamos a suponer que el kernel fuera /hda1/boot/kernel.

```
mkdir /hda1
cd /hda1
mount -t ext2 /dev/hda1 /hda1
dd if=/hda1/boot/kernel of=/dev/fd0
rdev /dev/fd0 /dev/hda1
shutdown -r now
```

Con ello creará un disquete de arranque para su partición raíz. Después de arrancar desde el nuevo disquete, ejecute lilo para poder arrancar desde el disco duro en ocasiones posteriores.

Son muchas las cosas que se pueden decir sobre lilo y algunos detalles técnicos son complicados de explicar. Generalmente es fácil configurar una arranque básico que funcione y en caso de fallo daremos algunas indicaciones sobre las posibles causas.

Mensajes de progreso del cargador de arranque lilo.

A medida que Lilo va progresando va mostrando las letras 'L','I','L','O'. y si se detiene en algún punto será porque ha surgido algún problema que le impide progresar.

1. Si no aparece ninguna letra durante en arranque significa que el cargador Lilo no pudo encontrarse. Quizás olvidó ejecutar el comando lilo.
2. Si solo aparece **L** probablemente exista algún problema hardware.
3. Si solo aparece **LI** el kernel no fue localizado. Añada la opción linear para ver si se soluciona el problema. Esta opción genera direcciones lineales de sectores en lugar de direcciones utilizando cilindros, cabezas y sectores. Esto permite direccionar sin necesidad de usar la geometría del disco. Por ello también puede servir para solucionar el límite de los 1024 cilindros impuesto por algunas BIOS. Esta opción no debe usarse con la opción lba32 son

mutuamente excluyentes. La opción linear también puede entrar en conflicto con la opción compact.

4. Si solo aparece **LIL** significa que no se pueden leer los datos del fichero de mapa y puede tratarse de algún problema hardware.
5. Si aparece **LILO** significa que lilo finalizó correctamente.

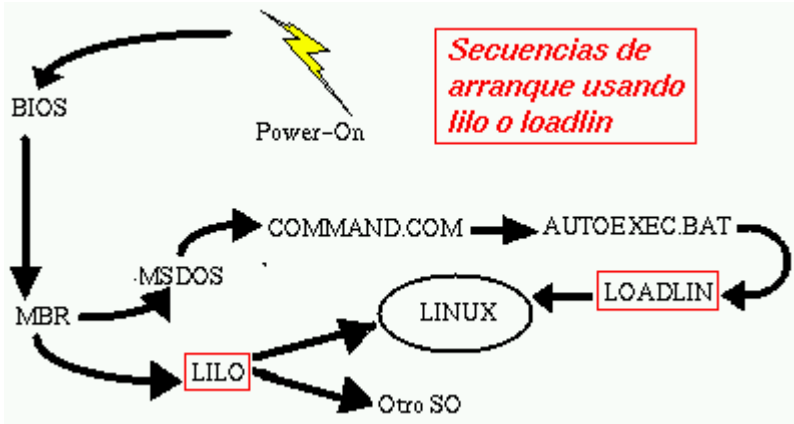
Loadlin

Se trata de un cargador de linux que actúa desde un sistema DOS. Loadlin ha de ejecutarse desde un sistema DOS que podrá ser un disquete con autoarranque o un disco duro.

Este cargador puede venir bien en algunas circunstancias. Hay que tener en cuenta que los PCs suelen estar pensados para arrancar sistemas DOS y alguna vez pueden incluir un componente hardware que dependa de un controlador DOS para configurar unos pocos registros en el hardware antes de arrancar el SO.

El proceso de arranque con loadlin se divide en dos etapas. Una es el arranque del propio sistema DOS y posteriormente se ejecuta loadlin.exe como un programa DOS normal y corriente pasando como argumentos el kernel y algunas cosas más.

Compararemos la secuencia de arranque con lilo o con loadlin en el siguiente gráfico:



Vamos a comentar una receta para construir un disquete de arranque con loadlin.

Formatee un disquete desde DOS incluyendo los ficheros del sistema de arranque. Es decir 'format a: /s'

Desde Linux copie a ese disquete el kernel. Por ejemplo

```
$  mcopy /boot/kern24 a:
```

Edite un fichero autoexec.bat para que contenga lo siguiente:

```
$  loadlin kern24  root=/dev/hda3 ro vga=3
```

Copielo al disquete. 'mcopy autoexec.bat a:' También copie loadlin.exe, 'mcopy loadlin.exe a:' Es un fichero que puede venir en algún paquete de su distribución o en el CD de instalación. Antes de retirar el disquete asegure que se grabaron todos esos ficheros usando 'sync ; sync'

Grub

GRUB Es un cargador que ofrece algunas ventajas sobre LILO pero también algunas desventajas. GRUB no es sensible a cosas como la sobreescritura de un kernel antiguo por otro nuevo con el mismo nombre ni necesita volver a ejecutar el cargador si no hay cambios a nivel de ficheros en los elementos que participaran en el arranque. Estos suelen encontrarse localizados generalmente en /boot/grub/. La linea de comandos del cargador es más compleja en GRUB. El fichero de configuración de GRUB es /etc/grub.conf

Un ejemplo de grub.conf para una configuración básica sería:

```
        root (hd0,2)
        install /boot/grub/stage1 (hd0,2) /boot/grub/stage2
0x8000 (hd0,2)
        /boot/grub/menu.1st
        quit
        ############
        (fd0) /dev/fd0
        (hd0) /dev/hda
        ############
        timeout=7
        splashscreen = (hd0,1)/boot/messages.col24
        default=0

        title=Linux
        root=(hd0,2)
        kernel=/boot/vmlinuz.2.4 noapic nosmp
          root=/dev/hda3

        title=Windows
        chainloader=(hd0,0)+1
```

El timeout viene expresado en segundos (en Lilo eran decimas de segundos). El default señala con un número la opción de arranque por defecto. En este caso por valer 0 tomará la primera que es title=Linux donde title hace las veces de label en Lilo. root=(hd0,2) indica que la partición raíz se encuentra en el primer disco duro, tercera partición. La linea del kernel en GRUP mezcla cosas que irían como append= y cosas que en Lilo aparecían en otras líneas de lilo.conf

Montar y desmontar particiones (mount)

La utilidad para montar y desmontar particiones es mount(8).

Gracias a mount los distintos dispositivos o particiones se pueden colgar en algún punto del gran arbol del sistema de ficheros raíz.

El fichero de configuración que utiliza mount para saber como donde, y cuando debe montar las particiones es fstab(5) que se localiza en /etc/fstab.

Cada linea de fstab representa un registro donde se describe un determinado sistema de ficheros del sistema.

Los campos en cada línea están separados por tabs o espacios. El orden de los registros en fstab es importante porque fsck(8), mount(8), y umount(8) recorren fstab secuencialmente a medida que trabajan. Los distintos campos de fstab son:

1. **fs_spec)** Describe el dispositivo a ser montado.
2. **fs_file)** Describe el punto de montaje para el sistema de ficheros. Para particiones de intercambio (swap), este campo debe decir ``none"
3. **fs_vfstype)** Describe el tipo del sistema de ficheros. (minix, ext, ext2, msdos,

   vfat, autofs, iso9660, nfs, swap, ... etc)

4. **fs_mntops)** Opciones de montaje. Es una lista de opciones separadas por comas. Estas pueden ser:
   - **defaults** Resume todas las opciones (rw, suid, dev, exec, auto, nouser, atime y async).
   - **rw, ro** Respectivamente monta en escritura lectura o en solo lectura

- **suid, nosuid** Respectivamente permite o impide el funcionamiento del delicado bit de permiso setuid.
- **dev, nodev** Respectivamente interpreta o no caracteres o bloques de un dispositivo especial de bloques.
- **exec, noexec** Respectivamente permite o impide la ejecución de binarios. (La ejecución de scripts no puede ser impedida y tampoco confíe en esto para impedir la ejecución de un binario. Cualquier programa que pueda ser leído puede ser ejecutado, aunque sea indirectamente).
- **auto, noauto** Respectivamente se puede montar con la opción -a o se requiere hacerlo explícitamente. Es decir uno por uno.
- **user, nouser** Respectivamente permite o impide que cualquiera pueda montar y desmontar dicho sistema de fichero. Lo normal es no permitir que cualquier usuario pueda hacerlo.
- **sync, async** Toda entrada salida sobre este sistema de fichero se realizará de modo síncrono o asíncrono. El modo asíncrono no asegura el momento en que las cosas realmente ocurren pero gracias a esto, el SO será capaz de optimizar las operaciones de entrada salida.
- **atime, noatime** No actualizar la última fecha de acceso a un inodo. Con ello se aumenta la velocidad de acceso.

5. **fs_freq)** Este quinto campo se usa para determinar si el comando dump(8) necesita volcar o no el contenido de este sistema de ficheros.
6. **fs_passno)** Lo usará fsck(8) para saber en que orden deber comprobar los sistemas de ficheros. Si no tiene valor se asume que el sistema de ficheros no debe ser chequeado. El sistema de ficheros raíz debería llevar fs_passno igual a 1, y otros sistemas de ficheros deberían llevar fs_passno igual a 2 o mas. Sistemas de ficheros en un mismo disco serán chequeados secuencialmente y conviene establecer el orden mas conveniente entre ellos, pero sistemas de ficheros en diferentes discos, con mismo valor fs_passno serán chequeados al mismo tiempo para ganar tiempo.

Para que mount sea capaz de montar un determinado sistema de ficheros resulta imprescindible que exista el soporte adecuado en el kernel bien sea estáticamente o como módulo. En el caso de la partición raíz no sirve usar un módulo para dicho sistema de ficheros ni para su controlador de disco ni nada ya que para acceder a los módulos es requisito previo imprescindible tener montado el sistema de ficheros raiz.

Ejecutando mount a secas podemos ver los sistemas que están montados y con que opciones. Con el comando df(1) vemos los sistemas montados y el numero de bloque totales, libres, usados y el porcentaje de uso.

Una vez montadas la particiones podemos considerar que el arranque ha tenido lugar dado que existe un kernel funcionando y los controladores básicos de memoria, de discos y de sistema de ficheros funcionan.

Será ahora cuando se arranque el primer proceso encargado de inicializar todo el sistema. Esto lo veremos en el siguiente capítulo.

EL ARRANQUE EN LINUX Y COMO SOLUCIONAR SUS PROBLEMAS (Segunda Parte)

Introducción

En la primera parte de este tema ya comentamos los diferentes cargadores de arranque, y ahora comentaremos el proceso de inicialización.

También repasaremos los problemas que pueden aparecer en nuestros discos o en nuestros sistemas de ficheros. Asumiremos en este capítulo que la particiones Linux son de tipo ext2. Entre otras cosas trataremos de explicar como reparar sistemas de ficheros ext2. Para otro tipo de sistemas de ficheros como ext3, reiserfs, JFS, XFS se usarán utilidades diferentes y cambiarán algunas cosas respect a ext2. En reiserfs y en ext3 no resulta necesario chequear el sistema de ficheros después de un apagón pero eso no lo hace más seguro frente a cortes de fluido eléctrico. Puede haber diferencias de rendimiento entre los diferentes sistemas de ficheros dependiendo de los tamaños de ficheros y de otras circunstancias pero tampoco deben esperarse diferencias espectaculares de rendimiento entre ellos.

No siempre lo nuevo es lo mejor. El sistema ext2 es muy estable porque lleva mucho tiempo funcionando. Los sistemas no son perfectos y sobre el sistema de ficheros recae mucha responsabilidad. Un fallo en un sistema de ficheros puede provocar la pérdida de toda la información del sistema. Hay algunas distribuciones que por defecto instalan sistemas de ficheros distintos de ext2. En parte es una cuestión de gustos. Nosotros nos limitamos a ext2 pero aunque usted use otro sistema de ficheros encontrará mucha información util en este capítulo.

En este capítulo se mencionará la localización de muchos e importantes scripts del sistema. Usted ya tiene nivel suficiente como para entenderlos.

Muchos de ellos son bastante críticos y es mejor no modificarlos otros no son tan críticos y pueden adaptarse sin riesgo a las necesidades de su sistema. En cualquier caso sea curioso, repase los diferentes scripts y aprenderá muchas cosas interesantes.

Muchos son los aspectos del kernel que inciden en la problemática del arranque pero esa parte la dejaremos para un posterior capítulo.

Inicialización del sistema (init)

Una vez el cargador completa la carga del kernel, este termina cediendo el control de ejecución al sistema operativo el cual efectuará una serie de tareas que conviene conocer.

1. Existe una limitación inicial que viene del diseño original de los PCs. IBM eligió como límite 1 MegaByte porque en su momento no consideró la posibilidad de que un ordenador personal necesitara jamás acceder a más de 1 Mega de memoria RAM. Por esta razón el kernel de Linux se carga en forma comprimida para no rebasar ese primer Megabyte de la memoria RAM. A continuación se ejecuta una primera cabecera de esa información grabada en RAM que pasa la CPU a modo protegido y de esta forma elimina la restricción de acceso a RAM pudiendo entonces acceder más allá del primer Megabyte de memoria, y pasando seguidamente a descomprimir todo el kernel para su posterior ejecución.
2. El kernel monta el sistema de ficheros raíz raíz en modo solo lectura, y lo hace sin poder usar ninguno de sus módulos ya que estos solo son accesibles una vez el sistema raíz está perfectamente montado. El kernel también necesita ser capaz de manejar el sistema de memoria virtual sin usar ningún módulo antes de que se monte el sistema raíz.
3. El kernel arranca el proceso init quizás precedido de algún proceso planificador dependiendo de los sistemas. El proceso init tendrá

como PID el valor 1. Será el proceso que genere a todos los demás procesos del sistema.

4. 'init' lee el fichero '/etc/inittab'. Este fichero contiene la lista de los procesos que deben ser arrancados. Los procesos se reparten en distintos niveles de ejecución para garantizar un orden de arranque o parada entre ellos dependiendo del nivel de ejecución al que están asociados. Los procesos serán arrancados mediante scripts rc situados en /etc/rcX.d/ donde X es el nivel de ejecución. Estos scripts usan una nomenclatura 'Snn...' o 'Knn...' donde el prefijo S significa arranque de un servicio y el prefijo K parada de un servicio. Viene de (S)tart, y (K)ill). Los dos dígitos siguientes determinan el orden de ejecución de cada script dentro de un mismo nivel.
5. 'init' arranca los procesos que están asociados al nivel 1 de ejecución, que se corresponde con el modo mono usuario. Antes de entrar en este nivel 1 se ejecutaran los scripts asociados al nivel 'S' que no es en realidad un nivel de ejecución que se pueda usar directamente, sino que contiene scripts específicos que se ejecutan durante el arranque, y como acabamos de explicar se ejecutan antes incluso del nivel 1. Se determina si el sistema debe chequear el sistema de ficheros con 'fsck(8)'. Esto será necesario si se terminó bruscamente sin desmontar los sistemas de ficheros o bien porque el sistema de ficheros alcance un cierto número de veces de montaje y desmontaje sin que se chequee el sistema de ficheros. Por lo tanto si resulta necesario se chequeará el sistema de ficheros raíz y si el chequeo termina sin novedad se continua con el montaje de los restantes sistemas de ficheros, pero si se detecta algún tipo de problema serio el sistema solicitará la contraseña de root quedando el sistema en el nivel 1 monousuario para se pueda ejecutar 'fsck' manualmente. Trataremos este tema más adelante.

   Los sistemas de ficheros tienen incorporado un contador de forma que alcanzado un cierto valor máximo se considera necesario chequear el sistema de ficheros. Dicho valor máximo puede ser

alterado usando 'tune2fs(8)', pero generalmente no es buena idea hacerlo. Un chequeo más frecuente resultará incómodo, uno menos frecuente supone aumentar el riesgo de un deterioro del sistema de ficheros.

6. Finalizado el chequeo de la partición raíz esta se vuelve a montar pero esta vez en modo lectura escritura. Chequear el sistema de ficheros Se montan el resto de los sistemas de ficheros especificados en '/etc/fstab' salvo aquellas entradas (lineas) que figuren como noauto. Vease 'mount(8)' y 'fstab(5)'
7. El sistema entrará en el runlevel predeterminado que por defecto que en el caso de Debian es el 2. Este al igual que muchas cosas viene especificado en '/etc/inittab'
8.
```
# The default runlevel.
```
9.
```
id:2:initdefault:
```

   En este nivel en Debian se arranca el entorno gráfico de ventanas mientras que en otros sistemas esto ocurre en el nivel 5. Los niveles de ejecución (runlevel) pueden variar de un sistema a otro pero en términos generales están organizados de la forma siguiente:

   - **0** Detiene el sistema.
   - **1** Entra en modo modo usuario.
   - **2 - 5** Entra en una variedad de modos multiusuario.
   - **6** Provoca el rearranque del sistema.
10. Después de ejecutar los scripts del runlevel predeterminado, 'init' procede a arrancar mediante 'getty(8)' una serie de terminales o seudoterminales conectados al sistema.

    'getty' presenta sobre el terminal un introductor 'login:' que tomará el nombre de usuario. Una vez introducido el nombre de usuario este proceso se transforma en el proceso login por medio de una llamada exec. (Esto ya se explicó en la parte del curso dedicada a los procesos). El programa 'login(8)' solicitará una password y en caso de que esta sea válida se transformará mediante un nuevo

exec en la shell destinada a ese usuario mediante su correspondiente entrada en '/etc/passwd'.

Cuando la shell muere, init lo detecta como si el proceso getty que arrancó hubiera muerto. (En realidad el pid no varía al hacer los sucesivos exec). Dado que los getty se arrancan en modo respawn, 'init' los rearranca cada vez que mueran.

Explicación de inittab

Primero consute la página man inittab(5). Luego examine su fichero '/etc/inittab'. Tiene que tener claro que la modificación de este fichero conduce con enorme facilidad a situaciones donde el sistema no será capaz de volver a arrancar.

Por el momento es mejor que se limite a leer inittab sin modificarlo. Las líneas que empiezan por '#' son simples comentarios. Las demás líneas constarán de cuatro campos separados por ':'

1. **Identificador** Generalmente el identificador admite un máximo de dos caracteres que deberá ser único para cada línea.
2. **Niveles de ejecución** Contiene los niveles de ejecución en los cuales la acción resulta adecuada. Si no aparece ninguno se entiende adecuada para cualquier nivel.
3. **Tipo de acción** Hay unas cuantos tipos de acciones (once, sysinit, boot, bootwait, wait, off, ondemand, initdefault, powerwait, powerfail, powerokwait, ctrlaldel, kbrequest) Puede consultar la página man de inittab(5) donde vienen explicados.
4. **Proceso** Programa a ejecutar.

El comando runlevel(8) devuelve los dos últimos niveles de ejecución.

En caso de no existir un nivel de ejecución previo al actual se representará con la letra N.

Pruebe a pasar modo mono usuario con el comando init o telinit.

```
# init 1
```

O con

```
# telinit 1
```

Recuerde que en este modo nadie más que es super usuario podrá usar el sistema y no podrá usar ningún servicio del sistema. Esto se hace para reforzar el control de root sobre todo el sistema.

En situaciones delicadas le permite asegurar de que en sistema no esta ocurriendo absolutamente nada salvo las ordenes que 'root' va ejecutando. Ofrece un absoluto control de operación para el administración frente a todo tipo de problemas incluidos los ataques intencionados.

La mejor manera de salir del modo monousuario es rearrancar con

```
# shutdown -r now
```

O con

```
# init 6
```

O con

```
# telinit 6
```

Los scripts rc

La información sobre los scripts rc o sobre inittab permite comprender el proceso completo del arranque de Linux. Gracias a esta información podrá con un poco de suerte diagnosticar una enorme variedad de problemas y solucionarlos, pero experimentar por puro juego con estas cosas puede ocasionar problemas con una gran facilidad.

Los scripts rc o el fichero inittab son fáciles de interpretar pero su funcionamiento es bastante crítico para el buen funcionamiento del resto del sistema.

Nunca debería tocar nada sin asegurarse que puede llegado el caso dar marcha atrás y dejarlo todo como estaba. Por ejemplo si edita alguno de estos ficheros saque previamente una copia del original y guardela por un tiempo.

Existe una opción para arrancar el proceso init sin que ejecute ninguno de los ficheros de script.

```
# init -b
```

Esto está especialmente indicado para ciertos problemas de arranque. Como veremos luego ficheros de configuración del tipo que sean que impidan un arranque pueden ser recuperados montando la partición en modo lectura escritura.

Los scripts rc son ficheros presentes en los directorios /etc/rc.d#.d donde # es el runlevel . Cada script rc suele implementarse como un link simbólico que empieza por S o K (start) o (kill) seguido de dos dígitos y que apunta a un fichero con el mismo nombre pero sin S ni K ni los dos dígitos.

Haga lo siguiente:

```
# cd /etc
# ls -l rc*.d/*cron* init.d/*cron*
```

Obtendrá algo parecido a lo siguiente:

```
init.d/anacron
init.d/cron
rc0.d/K11cron -> ../init.d/cron
rc1.d/K11cron -> ../init.d/cron
rc2.d/S20anacron -> ../init.d/anacron
rc2.d/S89cron -> ../init.d/cron
rc3.d/S20anacron -> ../init.d/anacron
rc3.d/S89cron -> ../init.d/cron
rc4.d/S20anacron -> ../init.d/anacron
rc4.d/S89cron -> ../init.d/cron
rc5.d/S20anacron -> ../init.d/anacron
rc5.d/S89cron -> ../init.d/cron
rc6.d/K11cron -> ../init.d/cron
```

Los servicios cron y anacron son servicios que ya hemos tratado en un capítulo anterior. Observe que estos servicios de forma similar a la mayoría de los servicios se arrancan y paran desde 'init.d/cron' y 'init.d/anacron' respectivamente. Observe que los enlaces simbólicos de arranque apuntan al mismo script que los enlaces simbólicos de parada.

Cualquier servicio puede arrancarse o detenerse de forma individual haciendo: '/etc/init.d/servicio start' o '/etc/init.d/servicio stop' respectivamente.

El proceso 'init' procesa los contenidos de los directorios 'rc#.d/' para ir arrancando y parando los servicios adecuados.

Como puede verse en '/etc/inittab' el encargado de procesar los scripts 'rc' es '/etc/init.d/rc' que se usa pasándole el nivel de ejecución como argumento. El script 'rc' es el que usa la letra inicial del enlace simbólico

para lanzar el script añadiendo un argumento 'start' o 'stop' según resulte la letra inicial.

El orden de ejecución de los diferentes scripts del sistema en Debian es es siguiente:

1. Como puede verse en inittab (si::sysinit:/etc/init.d/rcS) desde 'init' se arranca '/etc/init.d/rcS'.
2. Desde '/etc/init.d/rcS' son arrancados los scripts que se ajustan al patrón '/etc/rcS.d/S[0-9][0-9]*'
3. Desde '/etc/init.d/rcS' se arranca los scripts en '/etc/rc.boot/*' usando 'run-parts /etc/rc.boot'.
4. Desde '/etc/init.d/rcS' se arranca '/sbin/setup.h' (Esto en Debian Potato, en Slink era '/root/setup.sh')
5. Como puede verse en inittab(5) desde 'init' se arranca los scripts del nivel de ejecución pasado como argumento a init o en caso de ausencia los scripts correspondientes al nivel de ejecución por defecto. (En Debian es el 2). Para ello se usa '/etc/init.d/rc' pasándole el nivel de ejecución como argumento.

Como hemos visto hay una serie de scripts destinados a procesar un conjunto de scripts contenidos en un directorio. Esta técnica se usa en varios casos:

- En el caso de '/etc/init.d/rc' se procesa los scripts del directorio correspondiente a ese nivel de ejecución añadiendo un argumento start o stop dependiendo de que el nombre empiece por S o por K.
- En el caso de '/etc/init.d/rcS' solo se procesan los scripts encontrados que empiecen por S añadiendo la opción start.
- En el caso de 'runparts(8)' solo se procesa los scripts del directorio pasado como argumento sin añadir argumentos al script. Ya habiamos visto que 'runparts', es utilizado con frecuencia para procesar las tareas periódicas con 'cron(8)' organizando en directorios las distintas tareas dependiendo de que sean diarias, semanales, mensuales, anuales, etc.

Personalizando el proceso de arranque

No decimos que estos scripts de arranque no se puedan tocar pero hacerlo es arriesgado. Por ejemplo un script que no termine su ejecución colocado en '/etc/rc.boot' detendrá el proceso de arranque.

Si intenta añadir un servicio que deba estar permanentemente activo lo mejor es incluirlo en el propio inittab asociado a la opción respawn para que en caso de morir el proceso init lo rearranque. Sin embargo si el proceso muere nada más arrancar init lo rearrancaraá continuamente creando un problema en el sistema. Afortunadamente init detectará eso. Si init detecta que un proceso rearranca más de 10 veces en 2 minutos suspenderá durante cinco minutos el nuevo intento de arranque. Tiempo que tendrá que aprovechar solucionar el problema de alguna forma.

Por ejemplo. Puede intentar editar directamente '/etc/inittab'. Si no se pone nervioso dispone de 5 minutos antes de que se produzcan otros 2 minutos de locura del sistema.

Si edita '/etc/inittab' desde el propio sistema el proceso init no tomará en cuenta los cambios mientras no se arranque o mientras no se le informe de ello mediante:

```
# init q
```

Si esto no le convence o no se le ocurre nada mejor puede rearrancar desde un sistema de disquete para eliminar la entrada en inittab mientras el problema no esté solucionado. Más adelante se explica como solucionar un problema de olvido de password de root y básicamente en este caso habría que hacer lo mismo pero en lugar de editar '/etc/passwd' habrá que editar '/etc/inittab'.

Recuerde que la modificación de '/etc/inittab' es una operación crítica que puede afectar a la capacidad de arranque de su sistema.

En Debian se puede añadir un script asociado a un nivel de ejecución deseado. Para ello se coloca el script en /etc/init.d/ y luego se ejecuta 'update-rc.d(8)' con los argumentos adecuados que creará los enlaces simbólicos correspondientes.

Por ejemplo:

```
# update-rc.d mi-script defaults 88
```

Crearía los enlaces simbólicos S88mi-script y K88mi-script para todos los runlevels.

```
# update-rc.d mi-script stop 83 1 2 3 5 6 start 85 2 3 5
```

Crearía K83mi-script para los niveles 1 2 3 5 y 6, y S85mi-script para los niveles 2 3 y 5.

Para eliminar servicios innecesarios en Debian puede hacer lo siguiente:

1. Cree un directorio por ejempo '/etc/services_off/'
2. Guarde la información actua siguiente: 'ls -l -d rc*/* > /etc/services_off/ls-l.out', y después mueva los scripts correspondientes a los servicios innecesarios desde '/etc/init.d/' a '/etc/services_off/'. (Se pueden en lugar de eso eliminar los scripts pero de esta forma podemos dar marcha atrás con más facilidad si fuera necesario).
3. Para cada unos de los servicios eliminados hacer 'update-rc.d nombre_servicio remove'. Con ello se eliminarán selectivamente todos los links simbólicos que apunten a '/etc/init.d/nombre_servicio' desde los directorios.

Continuamos con la descripción del proceso de rranque. Cuando arranca el sistema aparecen unos mensajes que pueden ser alterados editando '/etc/

motd', '/etc/issue', y '/etc/issue.net' Quizás le sorprenda ver unos caracteres extraños pero son secuencias de escape destinadas a la consola. Si le interesa comprenderlas consulte la página man de console_codes (4).

Recordemos que existen otros scripts que se ejecutan previamente al establecimiento de una sesión de intérprete de comandos.

El primero que se ejecuta en el caso de interpretes de compatibles con sh (sh(1), bash(1), ksh(1), ash(1), ..) será '/etc/profile'

En el caso de abrir una sesión con bash desde login se ejecutará '~/.bash_profile' que a su vez ejecutará '~/.bashrc'. Si la sesión no se arranca desde login, (Por ejemplo si arrancamos una subshell desde un editor.), solo se ejecutará '~/.bashrc'.

Hay otros scripts que se ejecutan para arrancar sesiones de X Window mediante xdm(1) o startx(1). Para configurar el arranque de xwindows mediante el arraque de una serie de procesos en background que personalizan el arranque en modo gráfico se usa '~/.xinitrc' en su defecto se usará '/etc/X11/xinit/xinitrc'.

Para establecer el servidor se usará '~/.xserverrc' en su defecto se usará '/etc/X11/xinit/xserverrc'. **Arranque directo con sh sin usar init**

Existe una forma de arrancar usada en emergencias que puede salvar en caso de apuro pero debe usarse con la máxima precaución.

Consiste en arrancar usando en lugar del proceso init, hacerlo arrancando directamente una shell. Eso se puede hacer pasandole al kernel la opción 'init=/bin/sh'. Esto se puede hacer desde lilo por ejemplo. De esta forma se podría montar la partición raíz en modo lectura/escritura y modificar nuevamente el fichero dañado, cambiar una password olvidada, o cualquier otra cosa.

Si lo hacemos así hay que tener en cuenta que no estamos usando un modo de funcionamiento normal, y por ello antes de rearrancar conviene

asegurarse que el sistema de ficheros queda perfectamente escrito sin operaciones de entrada salida pendientes en memoria. Para ello haga lo siguiente.

```
# sync
# sync
# shutdown -r now
```

Por el contrario si apaga de cualquier manera, el daño podría ser considerable e incluso podría verse obligado a reinstalarlo todo.

Algunas veces se arranca de esta forma cuando se sospecha que el fichero correspondiente al ejecutable 'init' pudiera estar dañado impidiendo totalmente el arranque pero podría ocurrir que arrancanco con 'sh' tampoco logremos nada. Tanto 'init' como 'sh' usan librerías compartidas comunes y un daño en alguna de estas librería impedirá el arranque usando cualquiera de las versiones normales tanto de 'init' como de 'sh'.

Una librería dinámica compartida consiste en un conjunto de funciones que pueden cargarse en memoria una sola vez y ser aprovechadas y compartidas por varios procesos funcionando simultaneamente.

Por ejemplo 'libc.so.6' , '/lib/ld-linux.so.2' son librerías comunes a muchos programas en Debian 2.2. Puede comprobarlo consultando las librerias compartidas usadas por 'init' y 'sh'. Para ello haga lo siguiente:

```
# ldd /bin/sh /sbin/init
```

Obtendrá lo siguiente:

La libc contiene todas funciones de la conocidísima librería estandar de C y algunas funcionalidades añadidas por GNU. este curso no está enfocado a la programación. De dotas formas si necesita información sobre esta librería puede consultarlo usando:

```
# info libc
```

Hay quien mantiene una versión compilada estáticamente de init o sh. En realidad no es necesario. En caso de que el sistema tenga deteriorada librerías dinámicas lo apropiado es partir de un disco de rescate y recuperar desde copias de seguridad antiguas todo el sistema.

Que es un CD de rescate y para que sirve

Un sistema puede sufrir algún accidente o ataque que le impida volver a arrancar. La pérdida de información de un sistema se puede subsanar mediante copias de seguridad como las que ya se comentaron en lecciones anteriores. Si se ha perdido absolutamente todo resulta evidente que habrá que recurrir a la última copia de seguridad global pero una copia global resulta muy costosa y quizás no se haga con demasiada frecuencia. Por ello generalmente se hace ocasionalmente y se complementan con copias incrementales posteriores.

A pesar de ello si se recupera un sistema a partir de la última copia puede haber información no respaldada por ser demasiado reciente.

Afortunadamente la pérdida de la capacidad de arrancar no significa que se trate de un desastre que obligue a recuperar todo el sistema desde las copias de seguridad o volviendo a instalar, etc.

Generalmente se puede recuperar el arranque interviniendo exclusivamente donde reside el problema.

De esta forma la recuperación resultará mas completa, más rápida y puede que nos sirva para prevenir un caso similar en el futuro.

Para poder arrancar de forma independiente y poder iniciar la reparación del sistema, es para lo que se usan los disquetes o CD's de rescate.

Evidentemente en caso del CD es importante que nuestro sistema sea capaz de arrancar desde CD. Para ello las opciones de arranque correspondientes deben de estar habilitadas en la CMOS.

Si nuestro sistema no puede arrancar en modo alguno desde CDROM habrá que conformarse con el uso de un disquete de rescate pero en un disquete de rescate no caben demasiadas utilidades de ayuda.

Puede darse el caso que un sistema que arranca necesite ser reparado desde un sistema de rescate. Por ejemplo si el sistema de ficheros está deteriorado y no podemos usar de modo fiable las utilidades del sistema para autorrepararse usaremos el CD de rescate.

Muchas distribuciones permiten que alguno de los CDs de instalación pueda ser usado para arrancar en modo rescate. Una vez que un sistema de rescate ha arrancado tenemos un pequeño sistema Linux arrancado. Una de las formas en las que un disco de rescate puede arrancar es de forma totalmente independiente sin necesidad de usar ningún disco duro para funcionar.

Generalmente se recurre a usar ramdisk para montar la partición raíz aunque generalmente menos adecuado también podría usarse para eso un disquete.

En Debian también podemos introducir el CD de instalación y progresar hasta seleccionar el teclado y el idioma para poder trabajar cómodamente. Después de esto se puede abandonar la instalación y continuar abriendo una sesión nueva en un seudoterminal. Por ejemplo y continuar como si estuviéramos en modo rescate o también se puede arrancar desde este CD usando distintas modalidades. Por ejemplo indicando la partición raíz. Desde un sistema Debian también se puede montar el cdrom de instalación y crear con ello un par de disquetes de rescate. Para ello montamos el cdrom en /cdrom y ponemos un disquete formateado en la disquetera y luego hacemos:

```
 # dd if=/cdrom/install/rescue.bin of=/dev/fd0 bs=1k
count=1440
```

Esto crea el rescue disk, con el que se arranca el equipo.

```
 # dd if=/cdrom/install/root.bin of=/dev/fd0 bs=1k
count=1440
```

Con esto otro se crea el root disk que pedirá mientras arranque.

Si usamos la partición raíz en disco duro solo servirá para casos en los que el problema solo afecte a elementos muy concretos del arranque tales como kernel, cargador de arranque, etc. Pero no para casos donde la propia partición raíz esté comprometida por algún problema.

En cualquier caso un disco de rescate deberá incluir un conjunto de herramientas adecuadas para solucionar todo tipo de emergencias.

En Debian como ya hemos dicho el primer CD es multipropósito. Por defecto empezaría el proceso de instalación pero leyendo atentamente en las primeras pantallas de arranque con lilo descubrirá las distintas posibilidades para arrancar.

Si se va a usar una partición raíz independiente por ejemplo en ramdisk se podrá usar un CD de rescate de una distribución cualquiera para rescatar un sistema de otra distribución distinta. Un administrador de sistemas con varios sistemas Linux a su cargo solo necesita un CD de rescate para solucionar una gran cantidad de emergencias en cualquier sistema Linux. Algunos de estos CDs especialmente diseñados para rescate se ofrecen en su versión miniatura del tamaño de una tarjeta de crédito.

Una vez arrancado el sistema conviene comprobar que los discos están perfectamente reconocidos por el kernel del sistema de rescate. Para ello haga:

```
# fdisk -l
```

Obtendrá la información de los discos presentes en su sistema y sus diferentes particiones. Esta es una información que usted debería conservar en papel por si acaso algún día se borra por accidente la tabla de particiones. La perdida de la tabla de particiones no es un fallo frecuente pero puede ser muy grave ya que provoca la pérdida de absolutamente toda la información salvo que se vuelva a particionar exactamente todo igual que estaba. El particionado se hace con fdisk(8) que es una de las utilidades que no debe faltar en un disco de rescate. La lista de las herramientas más importantes que debe contener un CD de rescate son:

- Un kernel capaz de reconocer una gran variedad de sistemas de ficheros y de dispositivos de discos.
- Todo tipo de herramientas de diagnóstico, exploración, manipulación y reparación de sistemas de ficheros, y de discos. La lista sería muy larga. Casi todas son muy importantes.
- Un editor sencillo de texto.
- Si queda espacio, otras muchas utilidades de uso frecuente pueden venir muy bien.

IMPRESCINDIBLE. Compruebe que su sistema de rescate contiene las utilidades adecuadas para recuperar las copias de seguridad del sistema. Empaquetado y desempaquetado de ficheros (afio, cpio, tar) compresor y descompresor (gzip, unzip, bzip2, etc), otros como (find, etc) drivers para los dispositivos de disco y de unidad de respaldo.

Asegurese haciendo copias pequeñas de prueba y recuperandolas arrancando desde el CD o disquete de rescate. Tenga mucho cuidado. La recuperación de un backup es una operación que exige la máxima prudencia. Un error puede ser fatal. Repase el capítulo dedicado a las copias de seguridad. Dedique tiempo a estudiar y dominar sus herramientas de copia y respaldo.

El tiempo gastado en dominar estos temas es tiempo ganado. Lo normal en personas que administran ellos mismos sus equipos y que no dominan estos temas es que tarde o temprano pierdan un montón de días o semanas de valioso trabajo.

Lo mejor es ilustrar el uso de los disquetes o CDs de rescate mediante mediante un ejemplo que es todo un clásico. Olvidar la password de root.

Olvidar la password de root

No se preocupe.

Este es un problema relativamente frecuente. Afortunadamente no es grave. Algo que muchos preguntan es: ¿ Si un olvido de la password de root puede ser solucionado fácilmente entonces que utilidad tiene la password de root ?

La password no impedirá que alguien se acerque a su ordenador y lo destroce a martillazos, o lo robe. Tampoco evita que otras personas con acceso físico a su ordenador puedan cambiar la password. Se asume que usted no va a permitir acceder físicamente al lugar donde se encuentra su equipo a cualquier persona.

```
boot:  alias_arranque init=/bin/bash
```

Esto hace que se arranque una sesión shell sin necesidad de poner password. Es una operación bastante delicada repase las instrucciones relativas a Arrancar con init=/bin/bash. Una vez arrancado basta cambiar la password. usaremos el comando passwd(1).

```
# passwd root
```

Otra forma sería arrancar desde un disco de rescate que no use nuestra partición raíz para arrancar.

Deberá crear un directorio temporal que nosotros llamaremos mnttmp y montar en él nuestra partición raíz en modo lectura escritura. Repase el comando 'mount(8)'.

Después de eso entramos en '/mnttmp/etc/passwd' y eliminamos el campo de la clave. En este fichero 'passwd(5)'

```
cuenta:contraseña:UID:GID:GECOS:directorio_home:intérprete
```

Basta usar un editor en ese fichero para cambiar por ejemplo:

```
     root:x:0:0:root:/root:/bin/bash por
root::0:0:root:/root:/bin/bash
```

Por

```
     root::0:0:root:/root:/bin/bash por
root::0:0:root:/root:/bin/bash
```

Si existe un fichero '/mnttmp/etc/shadow' deberá eliminar igualmente el segundo campo en este otro fichero. ***Muy importante***. Antes de rearrancar desmonte el sistema de ficheros haciendo lo siguiente.

```
# sync
# sync
# umount /mnttmp
# shutdown -r now
```

Con ello la próxima vez que se haga login: root no le pedirá ninguna password. No debe olvidar dar de alta una password usando el comando passwd.

Recuperar un sistema de ficheros dañado

Puede ocurrir que un sistema de ficheros dañado no pueda ser reparado de forma automática durante el arranque. En el caso de que el sistema detect la necesidad de chequeo en alguna de las particiones 'fsck' las chequeará y en caso necesario las reparará. El orden de los registros en '/etc/fstab' (vease 'fstab(5)' es importante porque 'fsck(8)', 'mount(8)', y 'umount(8)' recorren fstab secuencialmente a medida que trabajan. El sexto campo, (fs_passno), lo usa el programa fsck(8) para determinar el orden en el cual se van a chequear los sistemas de ficheros cuando el sistema arranca. El sistema de ficheros raíz debería llevar fs_passno igual a 1, y otros sistemas de ficheros deberían llevar fs_passno igual o superior a 2. Sistemas de ficheros en un mismo disco serán chequeados secuencialmente y si tienen el mismo valor de (fs_passno) el orden será el orden de los registros en '/etc/fstab', pero sistemas de ficheros en diferentes discos con el mismo valor para fs_passno, serán chequeados al mismo tiempo para utilizar el paralelismo disponible en el equipo. Si fs_passno (sexto campo) no está presente o tiene un valor de 0, fsck asumirá que ese sistema de ficheros no necesita ser chequeado.

En caso de que la partición con el sistema de ficheros raíz no termine satisfactoriamente no se prosigue con el montaje de otros sitemas de ficheros sino que el sistema solicitará la contraseña de root para ejecutar fsck manualmente.

Si esto le ocurre alguna vez y no es usted un experto limitese a chequear la partición con la opción -y que en lugar de preguntar cosas quizás incompresibles para usted asumirá contestación afirmativa por defecto, ya que es la contestación menos comprometida en estos casos. Una vez finalizado el chequeo deberá hacer 'shutdown -r now' para rearrancar. Mientras tanto deberá cruzar los dedos. (Generalmente tiene éxito).

No es posible chequear un sistema de ficheros si este está montado con opciones de lectura/escritura. Si se llegara a forzar tal posibilidad de alguna manera, la destrucción de todo el sistema de ficheros sería inevitable pero afortunadamente fsck se da cuenta de ello y no lo intentará chequear si se da tal circunstancia.

fsck normalmente no hace nada si la partición parece estar correcta pero si usted quiere forzar un chequeo a pesar de todo deberá usar la opción -f. En el supuesto que no sea un experto continúe usando la opción –y, Es decir

```
# fsck -fy /dev/....
```

fsck(8) actualmente está implementado como un intermediario que a su vez usará algún fsck específico para el tipo de sistema de ficheros detectado sobre esa partición. La nomenclatura para estos comandos específicos es fsck.tipofs. Por ejemplo fsck.ext2(8) que puede usarse directamente en lugar de fsck. Debería consultar las páginas man de ambos comandos si está usando sistema de ficheros linux ext2. El comando mount(8) usado sin argumentos le puede informar sobre los tipos de ficheros que actualmente están montados en su sistema.

La información clave de un sistema de ficheros se sitúa en el superbloque. Esta estructura está duplicada por motivos de seguridad. Si alguna vez obtiene un error indicando que no se puede montar el sistema de ficheros debido a que el superbloque está dañado intente hacer lo siguiente sin olvidar la opción -S. También debe ir seguida de un cheque de todo el sistema de ficheros.

```
# mke2fs -S /dev/...
# e2fsck -fy /dev...
```

Esta operación no garantiza la recuperación de sus sistema de ficheros

La información del superbloque puede ser obtenida mediante dumpe2fs(8) pero si usted no conoce en profundidad la estructura de su sistema de ficheros no le resultará demasiado útil. A pesar de ello encontrará alguna cosa que si le puede interesar. Por ejemplo le permitirá saber cual es el máximo número de veces que se puede montar su sistema de ficheros antes de estimar necesario un chequeo y cuantas veces ha sido montado desde el último chequeo y en que fecha tuvo lugar. Lo que queremos decir es que tampoco es necesario entenderlo todo para sacarle algún provecho a este comando.

Recuperar un disco duro dañado

Puede ocurrir que un sistema se vuelva inestable y solicite con frecuencia el chequeo de alguna partición. Puede que a pesar de que aparentemente el fsck solucione el problema este reaparezca nuevamente con mucha frecuencia, o incluso a cada nuevo arranque.

Esto se debe generalmente a un daño en su disco duro. Los motivos pueden ser varios. Puede ser una avería grave o puede ser que algún sector como consecuencia de un corte de luz durante una escritura física a disco tenga deficiencias de formato a bajo nivel en los sectores grabados, es como si quedaran grabados de forma incompleta debido al corte de suministro eléctrico en el preciso instante de esa operación.

Este tipo de daños puede afectar a cualquier partición y la situación no es equivalente. Se puede ver afectada la partición raíz, alguna otra partición de datos o programas, o una partición de intercambio swap.

Una partición de swap dañada provocará cuelgues del sistema de forma más o menos aleatoria.

Si algún programa o librería dinámica compartida esta dañada puede que el sistema no pueda arrancar normalmente y deba arrancar desde un sistema de rescate. En cualquier caso en este tipo de situaciones puede que los elementos destinados a reparar su sistema estén igualmente dañados y por ello lo mejor es arrancar desde un disco de rescate y hacer un chequeo a todas las particiones incluidas las particiones de swap.

De esta forma se cerciora uno de la amplitud del daño.

Puede intentar usar badblocks(8) pero si no funciona adecuadamente puede que algún bloque defectuoso esté afectando al propio badblocks o a alguna de las librerías dinámicas.

Use badblocks desde algún disco de rescate. Los primeros CDs de las distribuciones suelen ofrecer la opción de arrancar en modo rescue.

badblocks es independiente del tipo de sistema de ficheros que use la partición. Puede ser ext2, ext3, reiserfs, linuxswap, umsdos, vfat, etc.

No es necesario usar un CD de rescate de la misma distribución. Vale cualquiera. Lo único que se necesita es pasar badblocks y luego se pueden hacer varias cosas. La opción -o de badblocks produce una salida que puede ser usada por e2fsck(8) o por mke2fs(8) ambos con la opción -l. Estos programas también admiten la opción -c para hacer que e2fsck o mke2fs usen a su vez badblocks. Realmente es muy sencillo y conveniente sin embargo le recomendamos lo siguiente.

1. Pase badblocks a todos los sistemas de ficheros presentes en el disco afectado uno a uno y anote o guarde lo resultados. (Por ejemplo redirigiendo salida a un disco sano).
2. Pase e2fsck con la opción -c en las particiones dañadas. De esta forma los sectores defectuosos ya no se usarán
3. Si el problema persiste saque una o dos copias de seguridad. Quizás tenga que olvidarse de sus utilidades y necesite recurrir a las utilidades de su disco de rescate para eso.
4. Verifique muy bien sus copias de seguridad.

5. formatee primero a bajo nivel usando alguna utilidad de la BIOS u otra proporcionada por el fabricante del disco duro.
6. Formatee usando mke2fs -c.
7. Recupere sus copias de seguridad.
8. Antes de usar el sistema vuelva a pasar badblocks. Si el problema continua o vuelve a aparecer al poco tiempo y tiene un mínimo de aprecio a sus datos, considere seriamente tirar el disco duro a la basura.

Con esto terminamos este denso capítulo y lleno de sugerencias del tipo mire usted tal cosa o consulte usted tal otra. Para la asimilación de este capítulo se exige más de una lectura y una cantidad de trabajo personal curioseando a fondo un montón de cosas. Quizás a usted le baste conocer superficialmente estas cosas para en caso de apuro poder localizar el remedio adecuado pero en ese caso guarde este material en más de un ordenador, o saque una copia por impresora.

Quizás cuando más lo necesite no pueda acceder a él precisamente por tener un serio problema en su equipo.

Si, si... ya lo sé, pero lo siento. Este curso solo se edita en formato HTML y no queda muy estético al imprimir. (Actualmente está en *.doc, *.odt y *.pdf).

No obstante existe la libertad para que aquel que lo desee adapte el formato según sus necesidades. En tal caso no estaría de más compartir el fruto de su esfuerzo de la misma forma en que yo lo estoy haciendo, al publicar este curso.

CONSEJOS GENERALES PARA COMPILAR KERNELS

Introducción

Como de costumbre no vamos a profundizar en aspectos teóricos sino que vamos a realizar un enfoque práctico de este tema. Vamos a hablar del núcleo (kernel) porque con mucha frecuencia resulta necesario recompilar el núcleo para adaptarlo a las necesidades de nuestro hardware.

Para una persona que únicamente pretende usar Linux en un o unos pocos ordenadores simplemente con propósitos personales puede que todo esto le suene a algo muy complicado o no apto para simples mortales.

Las distribuciones por regla general proporcionan un núcleo con amplias capacidades para abarcar una gran cantidad de hardware disponible. Esto conduce a núcleos poco optimizados por estar sobre-dimensionados. Para agravar el problema algunas distribuciones comerciales desaconsejan recompilar el kernel si no se tiene suficiente experiencia y advierten que en caso de recompilar el núcleo, se pierde el derecho a la asistencia gratuita en periodo de instalación.

Hay mucha gente que cree que la recompilación del kernel es una tarea para expertos. Una cosa está claro. Cuando se recompila un nuevo núcleo resulta una temeridad no contemplar la posibilidad de arrancar con el núcleo antiguo por si algo falla en el nuevo. En realidad es muy fácil olvidar algo y generar un núcleo que no sea capaz siquiera de arrancar pero eso no es un problema si mantenemos el antiguo.

Lo cierto es que la compilación de un kernel en Linux es mucho más simple que en otros SO. Con frecuencia se considera la recompilación de un kernel a medida como una tarea obligada ya que con ello se puede

obtener un núcleo más reducido y personalizado a nuestras necesidades. Esto resulta especialmente interesante en los casos de ordenadores con poca RAM.

Las primeras versiones del kernel de Linux residían íntegramente y permanentemente en la RAM física del ordenador.

En la actualidad los núcleos de Linux son modulares. Es decir, hay partes del kernel que pueden residir en disco y solo se cargan en el momento que resulte necesario. Se llaman módulos cargables y generalmente hay un programa llamado 'kerneld' encargado de su carga y descarga.

Las versiones del kernel

Hay dos líneas de fuentes del núcleo. Las de producción que son estables y las de desarrollo que no lo son. Las de desarrollo son adecuadas para los programadores que desean experimentar con funcionalidades nuevas del kernel, o para los usuarios que compraron hardware que no está soportado de una manera totalmente segura y probada en el kernel.

La nomenclatura de las versiones del núcleo utiliza tres números separados por puntos. Por ejemplo si hacemos 'cat /proc/version' podemos obtener la versión del kernel que estamos usando y que podría ser algo del tipo 'Linux version 2.2.17'.

- El primer número en nuestro caso es un '2' y representa el máximo nivel de cambio posible. De un kernel '1.x.x' a uno '2.x.x' hay un enorme salto.
- El segundo número debe considerarse de forma totalmente independiente si es par o si es impar. La serie par corresponde a la serie estable y la serie impar a la de desarrollo. Por ello a la 2.2.x la seguiría la 2.4.x y a la 2.3.x la seguiría la 2.5.x. El nivel de este curso hace suponer que usted no debería siquiera intentar usar la serie impar reservada para desarrolladores con ganas de probar cosas nuevas, para lo cual se suele usar un ordenador especialmente reservado para eso y sin información valiosa. Los

cambios en este segundo número 2.0.x, 2.2.x, 2.4.x realmente representan un importante cambio en el SO y por ello no se asegura la compatibilidad de todos los programas con la versión anterior. Es decir los programas desarrollados para 2.4.x pueden no funcionar para la versión 2.2.x y lo que es peor los programas desarrollados para 2.2.x pueden dejar de funcionar para las versiones 2.4.x. Lo cual significa que debe acompañarse de numerosas actualizaciones del software. Las distribuciones suelen facilitar estos cambios cuando se cambia a una versión superior de esa distribución.

- El tercer número representa generalmente una versión mejorada de la versión anterior manteniendo total compatibilidad. Suele corregir fallos detectados y añadir algo de soporte para nuevo hardware manteniendo la compatibilidad.

Quizás algo de lo que acabamos de decir sea totalmente exacto pero si que le permite entender bastante bien todo lo que significa esta nomenclatura.

¿Cuando conviene subir la versión de un kernel?

En un servidor dando servicio en Internet puede resultar necesario actualizarlo regularmente para corregir los problemas de seguridad que aparezcan, pero nosotros estamos tratando temas básicos de administración para sistemas de uso personal ya sea profesional o doméstico. Por lo tanto si su sistema operativo está funcionando correctamente y su hardware está perfectamente soportado no hay motivo para actualizarlo. La idea es que si algo funciona no hay que arreglarlo.

Otra cosa es recompilar la misma versión del kernel simplemente para cambiar algunas cosas a efecto de optimizarlo. De todas formas si tiene usted noticia de que cierto problema o carencia se soluciona usando una versión superior del kernel recuerde que mantener los dos primeros números de la versión del kernel es mucho más sencillo que pasar a una

serie superior en la que el segundo número es superior y recuerde usar siempre versiones pares estables.

Cuando conviene usar módulos

A la hora de configurar el kernel tendrá que ir leyendo detenidamente el significado de cada opción y verá que para algunas opciones resultará posible únicamente incluirlas o no incluirlas de forma estática. Es decir solo podrán estar en el kernel de forma permanente o no estar. Por el contrario otras opciones admiten una tercera posibilidad que es su inclusión como módulo y aquí viene la pregunta interesante. Cual es la mejor forma de incluirlo ? La respuesta es que todo lo que pueda ser puesto como módulo estará mejor como módulo a no ser que incluirlo de forma permanente resulte imprescindible para arrancar o que su uso vaya a ser casi continuo. Tenga en cuenta que si su kernel es por ejemplo un 2.2.17 los módulos serán cargados desde '/lib/modules/2.2.17/' pero para eso la partición raíz debe de estar montada y si usted tiene su partición raíz en un disco ide necesitará que las opciones correspondientes a BLK_DEV_IDE, BLK_DEV_IDEDISK deben de estar en modo estático (permanente) por que si están como módulos cargables el sistema jamás podría arrancar desde un disco ide. Hay otras opciones que también pueden ser necesarias en modo estático EXT2_FS, BINFMT_ELF, VT_CONSOLE, PROC_FS, etc…

Por el contrario esta claro que la unidad de cdrom, la impresora, sistemas de ficheros que se usen de forma ocasional, etc. pueden y deben ser configurados como módulos. Si tiene dudas deberá leer para que sirven intentando deducir si son necesarios para arrancar o no. EXT2_FS es para sistema de ficheros Linux y lógicamente es imprescindible. PROC_FS es un sistema de ficheros que se monta en /proc y que es usado por un montón de programas de forma casi continua. BINFMT_ELF permite ejecutar la inmensa mayoría de los binarios de Linux y por ello es imprescindible, etc. Si no es capaz de llegar a una conclusión sobre si es imprescindible, conviene que lo ponga como estático la primera vez que lo intente, y si todo va bien puede en un futuro probar a ponerlo como

módulo pero siendo consciente de que quizás no arranque. Todos estos cambios deben hacerse teniendo muy claro como volver a la situación anterior usando el kernel antiguo.

Primera prueba

Suponiendo que usted está usando arranque desde disco duro con el cargador LILO le proponemos practicar lo siguiente:

1. Repase las lecciones anteriores que tratan sobre el arranque.
2. Localice su kernel actual mirando /etc/lilo.conf.
3. Saque una copia de su kernel con otro nombre distinto.
4. Edite su /etc/lilo.conf añadiendo una entrada para usar la nueva copia del kernel.
5. !! Importante !!. Ejecute el comando 'lilo'.
6. Rearranque usando alternativamente una copia del kernel u otra.

En realidad estará usando el mismo kernel para arrancar pero de esta forma ya puede comprobar que puede arrancar a su elección usando dos ficheros de kernel que no tendrían porque haber sido iguales. Cuando obtenga un nuevo kernel debe de asegurarse que en caso de que este no arranque podrá usar el antiguo.

En cualquier caso no le recomendamos compilar el núcleo sin haber leído detenidamente el capítulo completo.

Instalación a partir de las fuentes

1. Lo primero que tendrá que hacer es descargarlo. Para bajar los fuentes del kernel existen muchos sitios en Internet para poder hacerlo. Por ejemplo ftp://ftp.us.kernel.org. Puede consultar en http://mirrors.kernel.org para localizar otros lugares adecuados. Vamos a suponer que deseamos instalar un kernel 2.4.10. Una vez descargado tendremos un fichero empaquetado y comprimido.

Probablemente algo similar a linux-2.4.10.tar.gz o linux-2.4.10.tar.bz2.

2. Cree un directorio 2.4.10 (mkdir 2.4.10)
3. Conviene engancharlo a un link simbólico /usr/src/linux. Compruebe si en /usr/src existe un directorio o un link simbólico 'linux'.
    1. Si no existe un directorio o un link simbólico 'linux'.
        1. Cree un link simbólico 'linux' mediante: 'ln -s 2.4.10 linux'
    2. Si existía un directorio 'linux'.
        1. Renombrarlo con el nombre de la versión correspondiente. Por ejemplo 2.4.2 mediante: 'mv linux 2.4.2'
        2. Cree un link simbólico 'linux' mediante: 'ln -s 2.4.10 linux'
    3. Si existía un link simbólico 'linux'.
        1. Borre el link simbólico mediante: 'rm linux'.
        2. Cree un link simbólico 'linux' mediante: 'ln -s 2.4.10 linux'
4. Despúes hay que descompactar.
    1. Para linux-2.4.10.tar.gz usar 'tar xzvf linux-2.4.10.tar.gz'
    2. Para linux-2.4.10.tar.bz2 usar 'bzcat linux-2.4.10.tar.bz2 | tar xv'

    Esto expande el fichero comprimido volcando su contenido en un directorio linux pero como nosotros lo hemos preparado lo meterá en 2.4.10 gracias al link simbólico, sin afectar al kernel anterior.

5. Situese ahora dentro del directorio linux -> 2.4.10. Comprobará que existe un Makefile. Este fichero es el que determina la forma y secuencia en que se compilará el kernel así como otras acciones de preparación y configuración.
6. La primera vez que se compila un nuevo kernel conviene hacer: 'make mrproper' y no será necesario volverlo a hacer cada vez que se recompile aunque sea con una configuración distinta. Tenga en

cuenta que 'make mrproper' eliminará el fichero que guarda la configuración de la última compilación del kernel. '/usr/src/linux/.config' y generará un '.config' con valores por defecto los cuales pretenden resultar adecuados para la mayoría de los casos.

7. Para configurar el kernel desde consola el método más cómodo es usar 'make menuconfig'. Para X Window se puede usar 'make xconfig' más amistoso pero requiere tener bien configurado el entorno gráfico. La configuración del kernel es una labor delicada que trataremos en una sección posterior.
8. Para compilar el kernel habrá que ejecutar una serie de pasos secuencialmente. 'make dep ; make clean ; make bzimage' Esto dejará un kernel en el fichero '/usr/src/linux/arch/i386/boot/bzimage' Si tiene dudas porque le ha parecido ver algún error, compruebe la fecha de creacion de este fichero para asegurarse de que es el que acaba de crear. Para usar este kernel habrá que añadirlo al fichero de configuración de LILO o del cargador que esté usando. La simple sustitución del kernel antiguo no le permitiría arrancar con el kernel antiguo si algo falla. También se puede generar directamente el kernel de arranque sobre un disquete desde el cual el sistema será capaz de arrancar. Para eso en lugar de 'make bzimage' use 'make bzdisk'. Otra alternativa más que no vamos a explicar es usar bzlilo. Es más fácil equivocarse y no poder arrancar o sobreescribir el kernel antiguo no pudiendo volver a usarlo. En Debian se puede construir un disquete de arranque mediante el comando 'mkboot' que necesita como parámetro la ruta al kernel.
9. Si la compilación falló leer el README para saber si el compilador es el adecuado para compilar ese núcleo. Error Signal 11 La compilación del kernel es un proceso muy exigente y resulta una verdadera prueba de fuego para la gestión de la memoria. Algunas veces una memoria aparentemente sana, falla y provoca un error 'signal 11' durante la compilación del núcleo. Esto puede ocurrir por ejemplo si los chips usados para la memoria RAM no

tienen exactamente la misma velocidad de acceso o por algunos otros problemas en su hardware.

10. Salvo que el kernel fuera compilado en modo estático, que genera un kernel enorme poco eficiente y solo se usa para casos especiales, lo más habitual es que tenga que generar los módulos y que tenga que instalarlos. Si está recompilando la misma versión de su kernel pero con opciones distintas estará modificando cosas que anteriormente se usaban con el kernel antiguo (el que está actualmente funcionando) y por lo tanto dar marcha atrás para dejarlo todo como estaba puede ser más complicado. Si por el contrario está compilando una versión distinta de la actual los módulos se instalarán en un lugar diferente. Para la compilación e instalación de los módulos haga lo siguiente: (make modules && make modules_install). Por lo tanto le aconsejamos usar un kernel diferente al que ya usa para mantener una versión antigua sin problemas.
11. Hay varias formas de arrancar con un determinado kernel. Si arranca por ejemplo desde un disquete MSDOS con loadlin.exe bastará duplicar el disquete y sustituir el kernel. Si usó bzlilo bastará generar un disquete nuevo. En ambos casos conserve el antiguo disquete de arranque durante algún tiempo. Etiquete correctamente los disquetes. Recuerde que con 'rdev(8)' se puede retocar la imagen de un kernel ya sea en disquete o en disco duro por ejemplo para establecer el tamaño de la ram o la partición raiz, o el modo video de consola.

    Para arrancar desde disco duro también hay varias formas. Dependiendo del cargador como ya hemos explicado tendrá que copiar el nuevo kernel al lugar adecuado para que el cargador lo use. Comentaremos únicamente el caso de usar el cargador LILO que es el más utilizado.

    1. Copie el kernel '/usr/src/linux/arch/i386/boot/bzimage' a '/boot/bzimage-2.4.10'.

2. Copie el nuevo fichero '/usr/src/linux/System.map' creado durante la compilación a '/boot/System.map-2.4.10'
3. Es importante que sepa utilizar LILO que se explicó en un capítulo anterior. La configuración de LILO se guarda en /etc/lilo.conf. Partiremos de un ejemplo:

```
boot = /dev/hda
delay = 40
vga = normal
root = /dev/hda1
read-only
image = /boot/bzimage.2.4.2
     label = linux.2.4.2
other = /dev/hda3
     label = dos
     table = /dev/hda
```

4. Tenemos el kernel antiguo en '/boot/bzimage.2.4.2', para añadir una nueva entrada para '/boot/bzimage-2.4.10' haríamos los siguiente:

```
boot = /dev/hda
delay = 40
vga = normal
root = /dev/hda1
read-only
image = /boot/bzimage.2.4.2
     label = linux.2.4.2
image = /boot/bzimage.2.4.10
     label = linux.2.4.10
other = /dev/hda3
     label = dos
     table = /dev/hda
```

5. Lógicamente los nombres (label) pueden ser diferentes. Solo deben ser distintos entre ellos y que sirvan para recordar que kernel es. Es muy importante una vez editado

y salvado '/etc/lilo.conf' no olvidarse de ejecutar el comando 'lilo' ya que de otra forma el sistema no arrancará la próxima vez.

12. La primera vez que arranque con un nuevo kernel conviene que ejecute el comando 'depmod -a'. Esto chequea las dependencias de los módulos usando el fichero 'etc/modules'. Este esquema resulta sencillo y es el que se utiliza en Debian pero otras distribuciones usan un esquema diferente para poder arrancar diferentes kernels con diferentes conjuntos de módulos. Por ejemplo en RedHat existe un archivo '/etc/rc.d/modules' que sirve para determinar la ubicación de los diferentes conjuntos de módulos. Vease depmod(8) y modules(5). Para la carga y descarga manual de los diferentes módulos consulte modprobe(8), insmod(8), rmmod(8), y lsmod(8). Ninguno de ellos efectua cambios permanentes en su sistema. Para determinar cuales son los módulos que deben cargarse de forma automática existen distintas herramientas para las distintas distribuciones. Concretamente para Debian existe 'modconf' que es una herramienta diseñada a base de menús y es muy fácil y agradable de usar. Permite determinar los módulos que serán cargados de forma automática.

Documentación

La documentación específica de su núcleo se encuentra en el directorio '/usr/src/linux/Documentation/' Si tiene dudas sobre el soporte de determinado Hardware o su configuración en el kernel es en este lugar donde tiene que buscar. Use grep para buscar.

/usr/src/linux/Documentation/Changes le permitirá saber los requisitos del nuevo kernel. Contiene un listado con las versiones mínimas necesarias de determinados programas compatibles con ese kernel. Si la versión del kernel solo difiere en el último número de versión la compatibilidad con kernels anteriores de esa misma serie está garantizada.

Configuración

Es recomendable salvar el '.config' con un nombre distinto para evitar perderlo al hacer 'make mrproper' o al hacer 'make menuconfig' etc. De esa forma si generamos un nuevo kernel con muchas opciones distintas y no conseguimos que funcione ni sabemos que opciones son las que están mal elegidas podremos volver a la configuración inicial y a partir de ella ir generando nuevos kernel variando cada vez pocas cosas para detectar cual es la opción que da problemas. Conviene usar un nombre largo y descriptivo como por ejemplo '.config.original.distribucion' o '.config.sinsoporte.sonido', etc. y guardarlo por un tiempo.

Si no lo ha mirado nunca mire su '.config' mediante 'less .config'. Podrá ver algo del tipo:

```
....
CONFIG_SYSVIPC=y
# CONFIG_BSD_PROCESS_ACCT is not set
# CONFIG_SYSCTL is not set
CONFIG_BINFMT_AOUT=m
CONFIG_BINFMT_ELF=y
# CONFIG_BINFMT_MISC is not set
....
```

Es decir que el resultado de una configuración es la creación de un fichero con una lista de opciones y sus valores. Las opciones que no se incuiran aparecen comentadas con un '#' a principio de línea.

Evidentemente las opciones para configurar un kernel son las mismas independientemente del procedimiento make lo_que_sea_config empleado.

Vamos a suponer que configuramos desde consola de texto. Las opciones pueden aparecer en distinto estado:

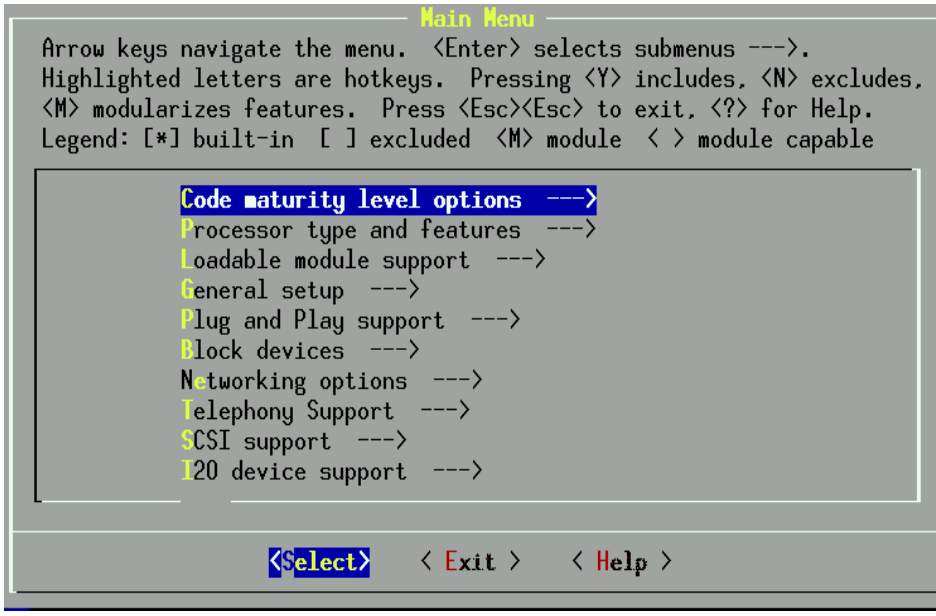
'[*]'	Opción estática seleccionada.
'[ ]'	Opción estática no seleccionada.
'<*>'	Opción cargable seleccionada en modo estático.

'<M>'	Opción cargable seleccionada como módulo cargable.
'< >'	Opción cargable no seleccionada.

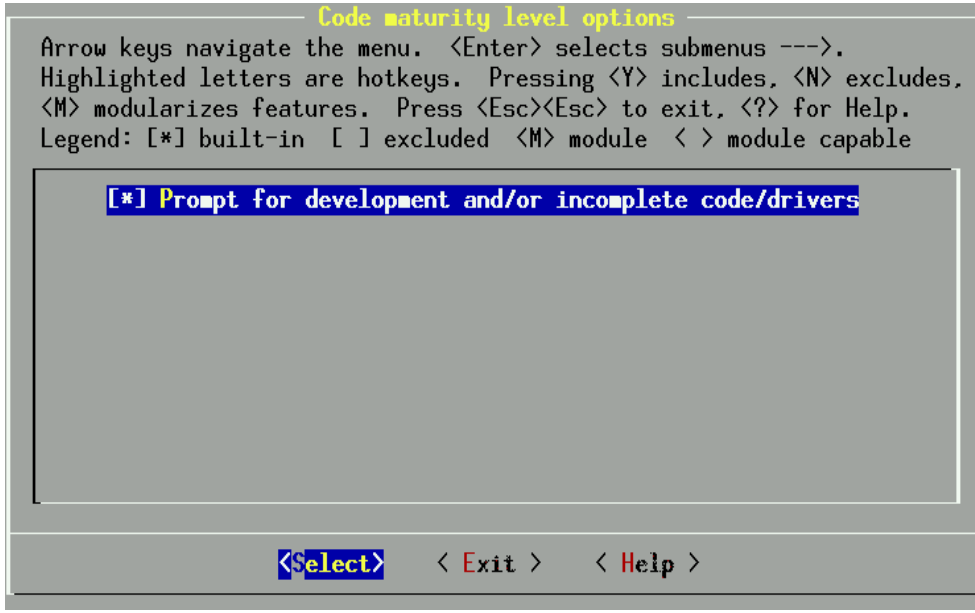
Para alterar el estado de las diferentes opciones se usan la teclas 'N', 'M', 'Y', y la barra de espacio. Esta última permite ir variando las opciones por las diferentes posibilidades. Las otras determinan las siguientes transiciones de estados.

'[*]'	--(N)-->	'[ ]'
'[ ]'	--(Y)-->	'[*]'
'<*>'	--(N)-->	'< >'
'< >'	--(Y)-->	'<*>'
'<*>'	--(M)-->	'<M>'
'< >'	--(M)-->	'<M>'
'<M>'	--(N)-->	'< >'
'<M>'	--(Y)-->	'<*>'

Usando 'make menuconfig' desde /usr/src/linux. Aparecerá el siguiente menú:



Puede usar las teclas de flechas verticales para recorrer las opciones del menú. Las pociones que disponen de la indicación '--->' son aquellas que acceden a un submenú. Seleccione la primera y aparecerá el siguiente submenú:



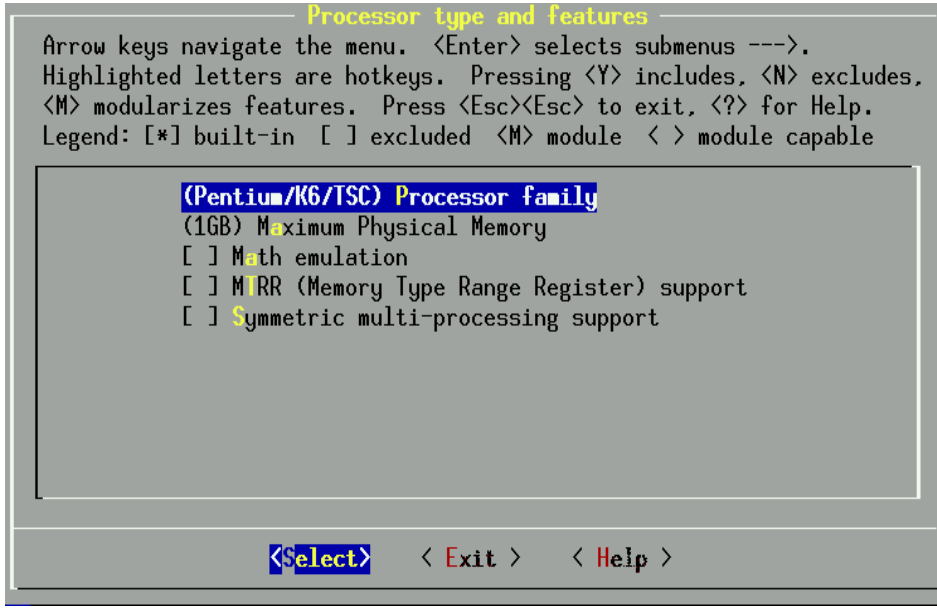
Puede ver que aparece una sola opción. Si está seleccionada '[*]' conviene generalmente deseleccionarla, '[ ]'. Para ello pulse la tecla 'N'. Con ello el presente menú ocultará todas las opciones consideradas experimentales. Dependiendo del hardware puede verse obligado a usar esta opción. Lo lógico es que la documentación del fabricante mencione tal necesidad pero algunos fabricantes de hardware no ofrecen mucha información sobre el uso de sus productos en Linux. Generalmente esta información se obtiene buscando en la web.

Si necesita incluir la opción deberá usar la tecla 'Y'. En este caso esta opción no está disponible para ser incluida como módulo. Si fuera así en lugar de '[*]' o '[ ]' figuraría '<*>', '' o '< >' para indicar respectivamente opción incluida estáticamente en el kernel, (se selecciona con 'Y') opción seleccionada como módulo cargable, (se selecciona con 'M') u opción no seleccionada. (se deseleciona con 'N') en este caso no es aplicable el concepto de módulo a esta opción ya que solo estamos eligiendo una forma de funcionamiento de menuconfig para que muestre más o menos opciones como disponibles. Las primeras veces que se usa menuconfig todo es muy nuevo y estamos obligados a leer gran cantidad de

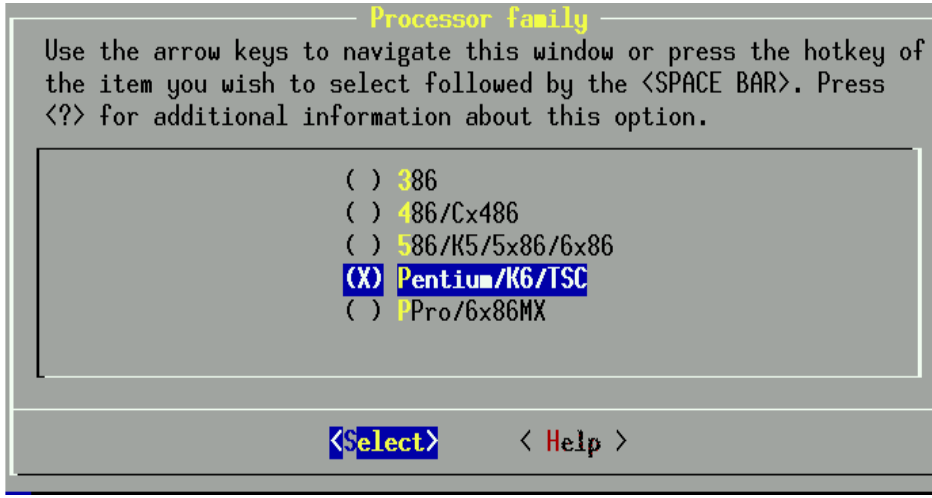
información en inglés generalmente. Para acceder a la información relativa a una opción basta situarse en esa opción y pulsar la tecla '?', o hacer click en el botón 'Help'.

Si no entendemos nada de lo que pone y no tenemos ni idea de lo que nos conviene hacer lo mejor es volver a leer la última parte de esa explicación donde se explica cual es la opción más recomendable en caso de no tenerlo claro.

Vuelva al menú principal usando la tecla 'ESC' o pulsando el botón 'Exit'. Seleccione a continuación la segunda opción:

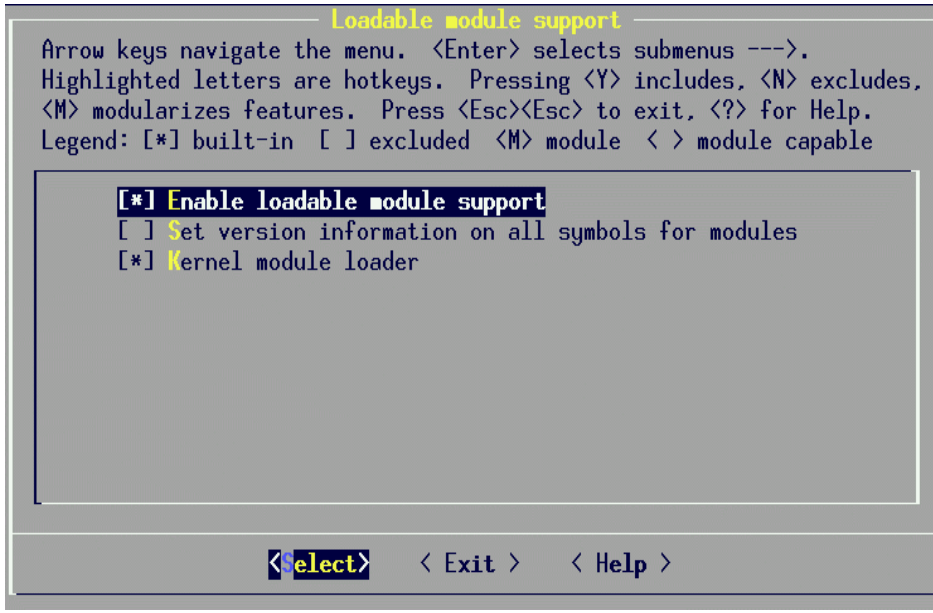


Este menú muestra varias opciones para distintas familias de procesadores. Si usted selecciona una opción para un procesador inferior al que realmente tiene seguramente el kernel funcionará aunque puede que no lo haga aprovechado todas las capacidades de su procesador. En cambio si selecciona una opción correspondiente a un procesador superior al suyo el kernel no funcionará. Suponiendo que eligiera la opción relativa al procesador Pentium aparecerá un nuevo menú:



Seleccione la opción adecuada y pulse <Enter>. Una vez seleccionada la opción retroceda hasta el menú principal.

Una vez en el menú principal, seleccione a continuación la tercera opción:



Recomendamos que elija las opciones que mostramos en la imagen como seleccionadas '[*]'. En cualquier caso ya sabe que puede acceder a la ayuda de las diferentes opciones. Una vez realizada la selección retroceda al menú principal mediante el botón 'exit'.

Si pulsa la tecla <ESC> desde el menú principal aparecerá una pantalla que preguntará si quiere guardar la configuración actual. Si contesta afirmativamente la configuración actual se guardará en '.config' perdiéndose la configuración anterior.

El número de opciones a configurar son muchas y no podemos detallar todo el proceso pero ya tiene una idea. Recuerde que muchas opciones del núcleo dependiendo de su sistema serán obligatorias para que el kernel pueda funcionar.

A modo de guía se resumirán unas pocas indicaciones críticas respecto a la capacidad de arranque de un kernel que permita trabajar con el:

- **Procesor type and features (Procesor Family)**

  Como ya hemos dicho su ordenador no funcionará si elige una opción correspondiente a un procesador superior o incompatible.

- **General configuration**
  - **PCI suport**

    Obligatoria en sistemas con PCI

  - **Sysctl suport**

    Activela ya que solo interesa desactivarla en sistemas especiales llamados incrustados.

  - **Kernel core**

Debe ser ELF

- **kernel suport for ELF binaries**

Incluyala como parte estática usando 'Y'. No debe estar como módulo.

- **Block devices**

Cualquier dispositivo que resulte necesario para arrancar y montar la partición raíz debe ser compilado estáticamente. Si no se compila o so compila como módulo no arrancará.

- **Character devices**

Serán necesarios como mínimo las siguientes opciones:

- **Virtual terminal**
- **Suport for console on virtual terminal**
- **UNIX98 PTY suport**

- **Filesystems**

Como mínimo deberá incluir soporte para el sistema de ficheros del sistema de ficheros raíz que generalmente es ext2, y sea cual sea deberá estar incluido estáticamente y no como módulo.

- **Console drivers**
  - **VGA text console**

Instalar un nuevo kernel personalizado en Debian

Todo lo dicho sirve también para Debian pero en Debian existe una forma alternativa muy interesante.

Antes de la aparición del kernel modular en Linux un kernel completo era un simple fichero. Esta sencillez permitía por ejemplo migrar un kernel de una máquina a otra simplemente copiando este fichero. Es lo que sucede ahora si compilamos un kernel en modo totalmente estático. Es decir obtenemos un gran y pesado kernel, totalmente completo y funcional pero sin soporte para cargar módulos.

Para poder migrar un kernel completo en el caso general de que tenga soporte para módulos habría que trasladar como mínimo el fichero compilado estáticamente y como mínimo todos sus módulos.

Por ello existe en Debian un make-kpkg capaz de generar un paquete Debian 'mi_kernel.deb' con el binario del kernel y sus módulos. Se puede por ejemplo copiar ese 'mi_kernel.deb' a otra máquina y bastará hacer "dpkg -i 'mi_kernel.deb'" para instalará el kernel en esa máquina. Para usar este sistema deberá instalar 'kernel-package' y algunos otros paquetes que dependen de él y que no estén todavía instalados.

Las primeras veces que compile un kernel no necesita usar este sistema pero este procedimiento resulta especialmente indicado en la instalación de kernels personalizados en más de una máquina o de diversos kernels personalizados en una misma máquina.

Para terminar

Solo usted puede saber que otras cosas son necesarias o convenientes en función del ordenador utilizado. Las primeras veces le costará trabajo porque tendrá que leer la ayuda de cada una de las opciones pero muchas de ellas recordará su significado la próxima vez. Lo mejor es partir de un kernel que funcione y eliminar todas aquellas opciones que sabemos con certeza no necesitaremos usar nunca en nuestro equipo. Recuerde que las opciones que están como módulos apenas suponen un ahorro prescindir totalmente de ellas. Por el contrario una opción que figure como estática innecesariamente está sobrecargando la parte estática del kernel y eso

reduce su eficiencia. Un kernel pequeño es sensiblemente más rápido que uno grande.

Si tiene dificultades para arrancar con el nuevo kernel intente hacerlo con el antiguo y si esto tampoco resulta repase el capítulo dedicado a las dificultades con el arranque en Linux.

No olvide que lo primero que hace cualquier kernel al arrancar es informar de todos aquellos dispositivos que es capaz de soportar. Esto lo puede visualizar nuevamente con 'dmesg(8)'. Si algún componente de su hardware no aparece y tampoco está disponible como módulo significará que ese dispositivo no esta soportado. También puede acceder a '/proc' para comprobar una amplísima variedad de capacidades de su kernel actual.