Curso para Aprender Linux desde Cero

Impartido por ...

Créditos...

OpenWebinars.Net

Septiembre 2015

¿Que es GNU/Linux?

GNU/Linux es un sistema operativo universal. El sistema operativo se define como un conjunto de programas que permiten interactuar con un pc y pueden ejecutar otros programas o aplicaciones.

En GNU/Linux, este sistema operativo se forma por un conjunto de programas que permiten controlar los diferentes elementos físicos del pc para ofrecerlos al usuario a la hora de ejecutar programas: escribir y leer datos de discos duros, utilizar dispositivos como impresoras o escáneres, etc. El núcleo que une todos estos programas, en este caso, se llama Linux; el resto del sistema fueron proyectos programados por o para el proyecto GNU, de ahí que para denominar al sistema operativo se utilice el término “GNU/Linux”, aunque de manera informal se suele llamar solamente Linux.

Este sistema operativo está basado en la filosofía de Unix, es decir, está diseñado para ser un sistema operativo multitarea y multiusuario. Estas características lo hacen desmarcarse de otros sistemas operativos conocidos, aunque también radica en la filosofía que engloba ya que, a diferencia de otros sistemas operativos, nadie es dueño de Linux ya que gran parte de su desarrollo lo han realizado y lo realizan voluntarios de todo el mundo de forma altruista.

Historia de Linux y GNU

La historia de este sistema operativo comienza en 1984 gracias a una organización llamada Free Software Foundation. Esta fundación comenzó a desarrollar un sistema operativo libre, de tipo Unix, que lo llamaron GNU. Este proyecto comenzó a crecer con herramientas de software diseñadas para ser utilizadas inicialmente en sistemas operativos Unix, aunque más tarde serían también compatibles para Linux. Aunque han sido muchos los colaboradores, la Free Software Foundation ha sido quien más ha contribuido en el desarrollo de dicho software, siendo Richard Stallman, fundador de la Free Software Foundation y proyecto GNU, el encargado de su difusión mundial.

El núcleo Linux apareció por primera vez en el año 1991 gracias a un estudiante de informática finlandés llamado Linus Torvalds, quien liberó su núcleo en el año 1992. A partir de ese momento, la evolución del núcleo junto con el software del proyecto GNU desembocó en lo que se conoce como GNU/Linux, que ha tenido un amplio y profundo desarrollo hasta hoy día.

¿Qué es una distribución de GNU/Linux?

Debido a que GNU/Linux es un sistema operativo completamente abierto, muchas organizaciones comenzaron a crear variantes de dicho sistema con unas ciertas características concretas orientadas a un grupo de usuarios específicos. Estas variantes recibieron el nombre de distribuciones GNU/Linux. Existen muchos tipos de distribuciones, entre las que se pueden encontrar distribuciones que están soportadas comercialmente como Fedora (Red Hat), openSUSE(Novell) o Ubuntu (Canonical Ltd.), distribuciones mantenidas por la comunidad, como pueden ser Debian y Gentoo, o distribuciones que no están relacionadas con ninguna de las anteriores como puede ser Slackware.

Además de esta clasificación, las distribuciones pueden estar orientadas también a grupos de usuarios

concretos. Hoy día existen multitud de distribuciones Linux orientadas tanto a profesiones como a dispositivos concretos. Como ejemplos en el ámbito educativo están Guadalinex o Edubuntu, en el artístico Musix, Ubuntu Studio, para las ciencias Scientific Linux, etc; así como recientemente las distribuciones de Android para Smartphones o Raspbian para los mini pcs tipo Rasperry Pi.

Razones por las que usar Linux

Existen numerosas razones por las que usar GNU/Linux, tanto en el ámbito laboral como en el personal.

A día de hoy existen múltiples distribuciones diseñadas para hacer la vida fácil al usuario particular, incluso distribuciones con una interfaz visual semejante a otros sistemas para que la curva de aprendizaje sea más corta. Además de esto, hay que tener en cuenta que todas las aplicaciones son libres, por tanto no existen limitaciones a la hora de adquirir software.

Al ser multiusuario y multitarea, un solo equipo puede ser usado por múltiples usuarios a través de la exportación del display gráfico por red, que puede ser una buena solución empresarial para el ahorro de costes. Se añade la particularidad de que GNU/Linux, al estar desarrollado por la comunidad y ser completamente gratuito, no es objetivo de organizaciones criminales u otros entes creadores de virus y malware a gran escala, como le puede suceder a otros sistemas operativos privados, además de contar con un soporte de una comunidad de millones de personas que aportan constantemente soluciones y parches a los problemas de seguridad de las distribuciones. Esto hace que GNU/Linux sea uno de los sistemas operativos más seguros y con mejor soporte del mercado actual.

Acceso a los sistemas Linux

Existen diversos métodos para la obtención de distribuciones GNU/Linux. La más usual es la descarga de la imagen ISO de la distribución, aunque también algunas distribuciones permiten la compra de CDs o DVDs de las distribuciones desde su página oficial. Además se suelen regalar distribuciones Linux en CD o Pendrives en las diferentes convenciones de software libre repartidas por todo el mundo.

Para este curso vamos a ver dos distribuciones muy relevantes en el ámbito empresarial y el personal: Debian y Ubuntu.

Debian: https://www.debian.org/

La portada web de la distribución muestra un menú superior donde podemos acceder a las ISOS desde el apartado “Cómo obtener Debian”. Es este apartado vienen los diversos métodos de descarga, así como la compra de material para su instalación.

Ubuntu: http://www.ubuntu.com/

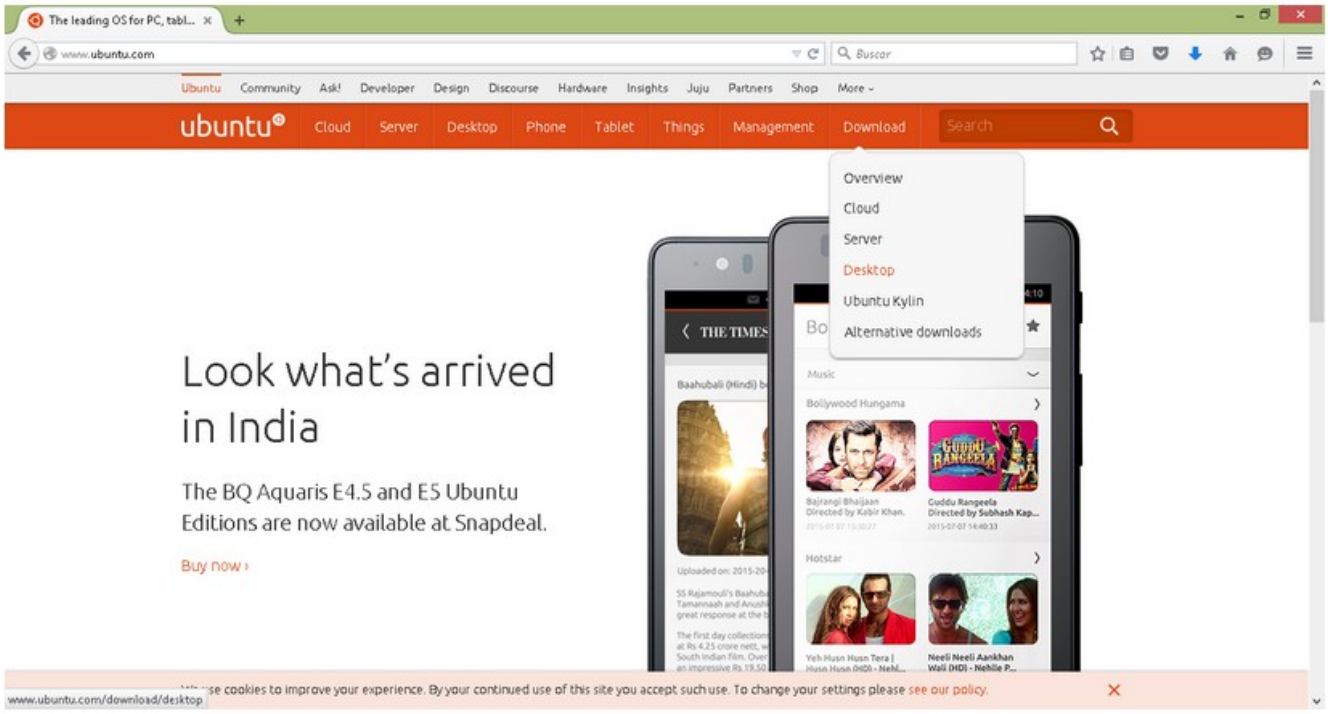
La portada de Ubuntu destaca por su aspecto visual, ya que tiene el soporte y mantenimiento económico de la empresa Canonical. En su menú superior vemos como se pueden apreciar diferentes productos, así como la zona de “Download”, que en su desplegable ya podemos elegir que tipo de producto descargarnos.

Instalación de SO Linux

Instalación máquina virtual

Una vez la aplicación está instalada vamos a crear una máquina virtual y procederemos a instalar Ubuntu Desktop.

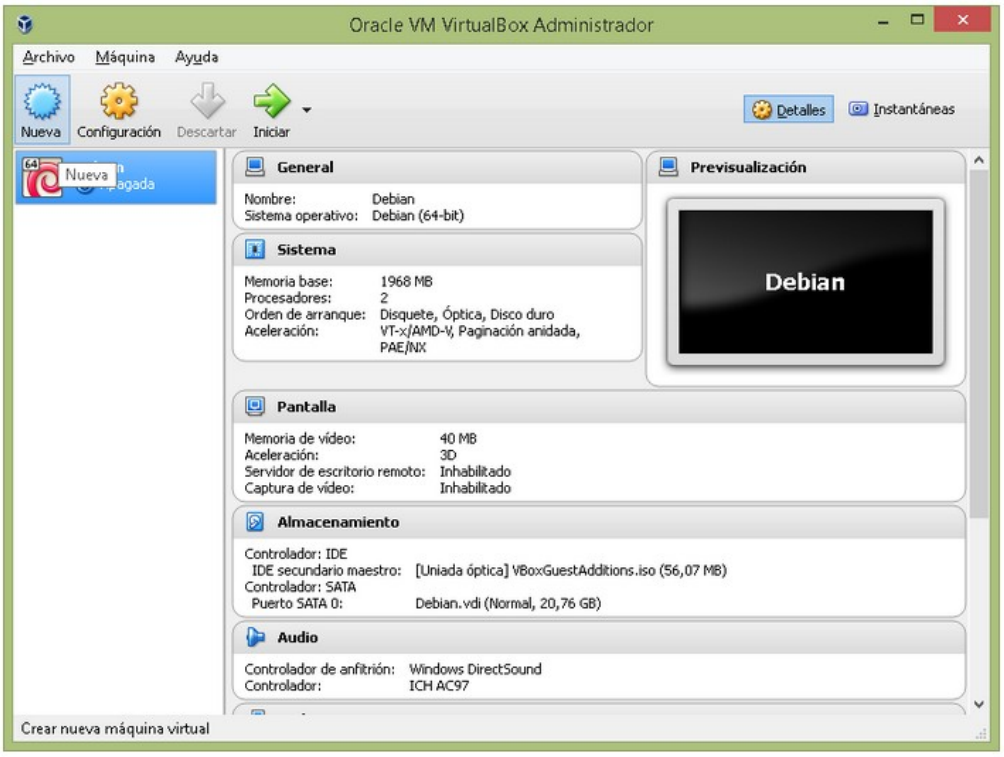
Lo primero que vamos a hacer es descargar la versión correspondiente de la web de Ubuntu:



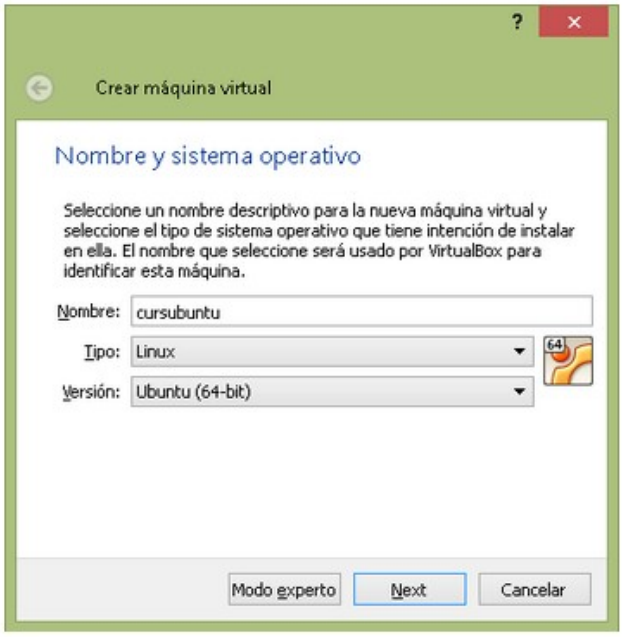
Es recomendable utilizar versiones de 64 bits en las máquinas virtuales si contamos con los suficientes recursos de ejecución y/o la distribución nos lo permite, puesto que algunas solo cuentan con la versión de 32 bits.

Una vez descargada la ISO, procedemos a crear la nueva máquina virtual en Virtual Box.

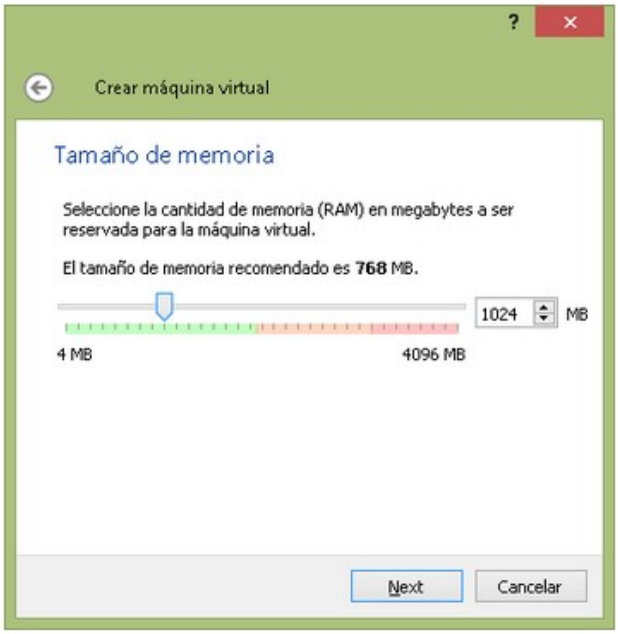
En la pantalla principal, pulsamos el botón “Nueva”:



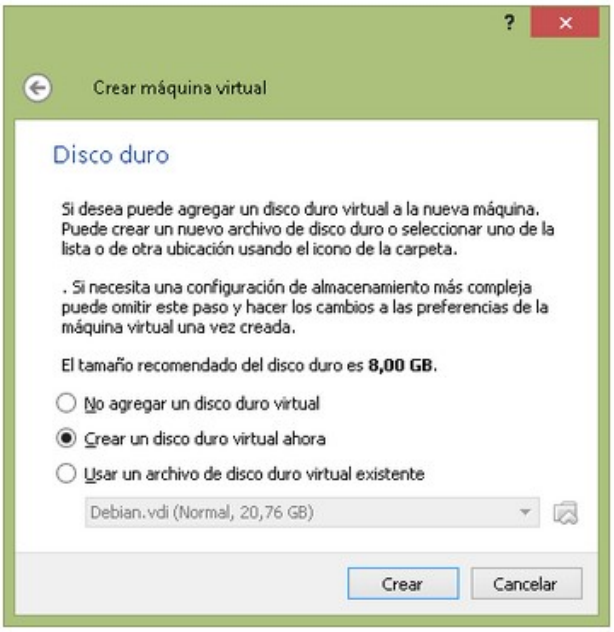
En la siguiente pantalla se establece el nombre de la máquina y que sistema operativo va a ser instalado. En el nombre estableceremos “cursubuntu” y en el Tipo “Linux” y versión “Ubuntu (64-bits)” (en el caso de bajar la ISO de 64 bits):



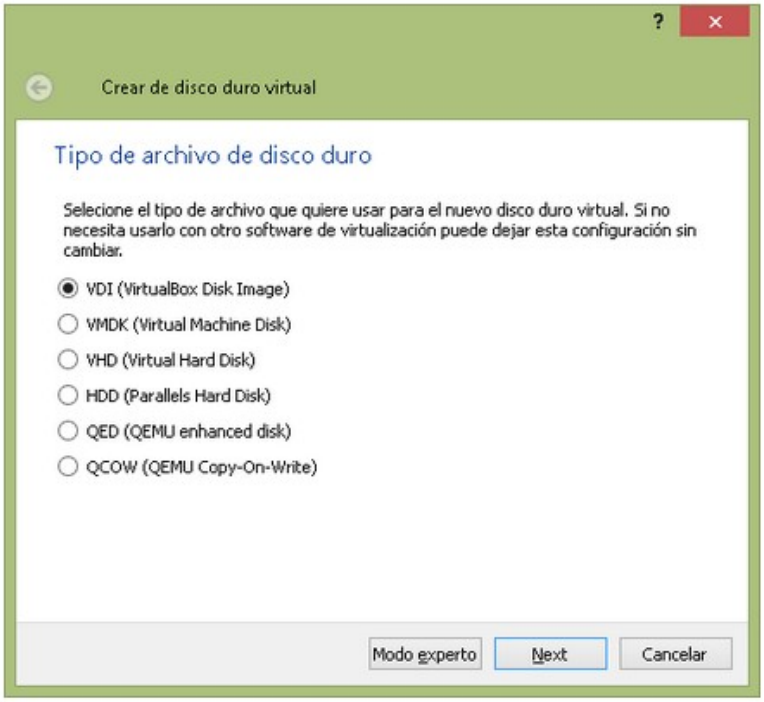
La siguiente pantalla nos permitirá configurar la memoria RAM de la máquina virtual. Lo aconsejable para que vaya fluido el sistema es establecer al menos 1Gb de RAM:



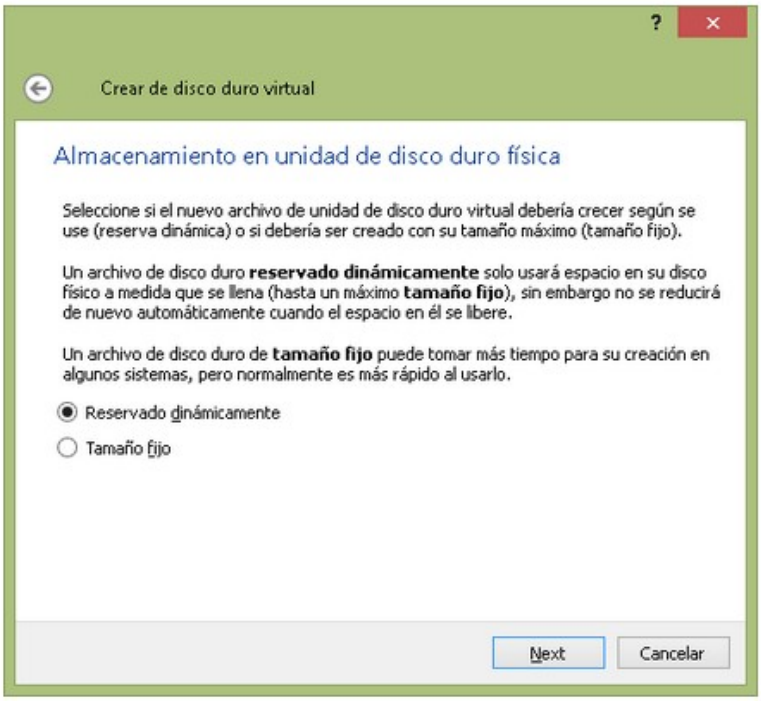
La siguiente pantalla nos permite establecer el disco duro que tendrá la máquina virtual. Le crearemos un disco nuevo que tenga una capacidad de 8 Gb:



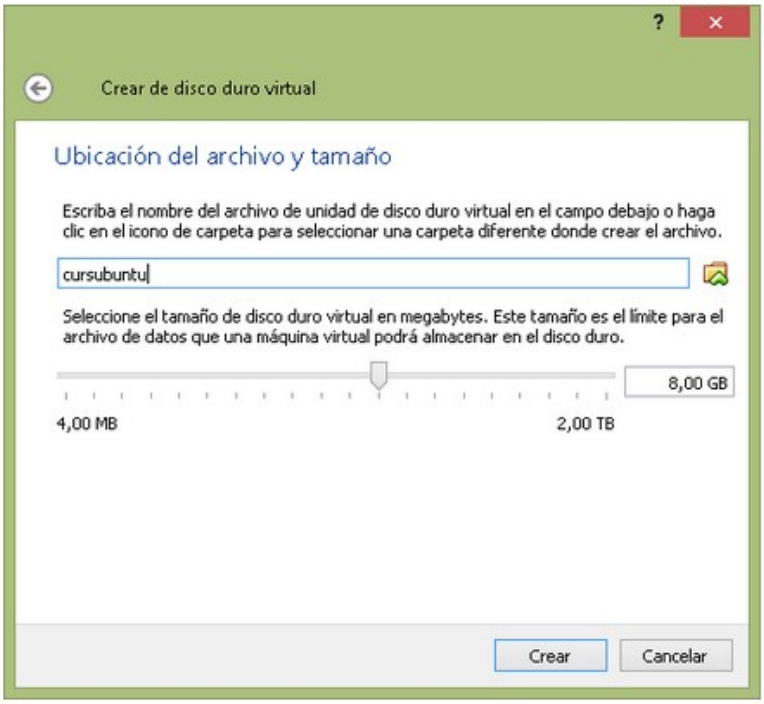
La siguiente pantalla nos permite seleccionar el tipo de fichero de imagen para la máquina virtual. Existen varios tipos, pero escogeremos el formato nativo de Virtual Box, que es VDI:



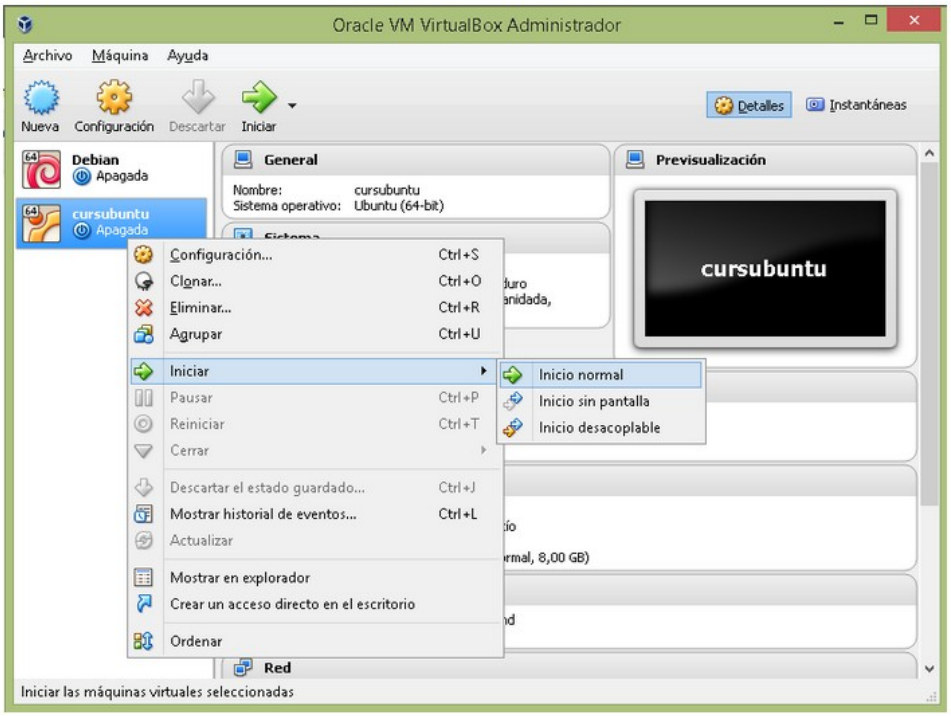
La siguiente pantalla nos permite gestionar el tamaño del fichero de imagen en relación con el uso de la máquina virtual, pudiendo escoger entre establecer desde un principio el tamaño fijo del fichero o ir incrementando el tamaño en función de los datos que se escriban en la máquina virtual. Escogeremos "Reservado dinámicamente":



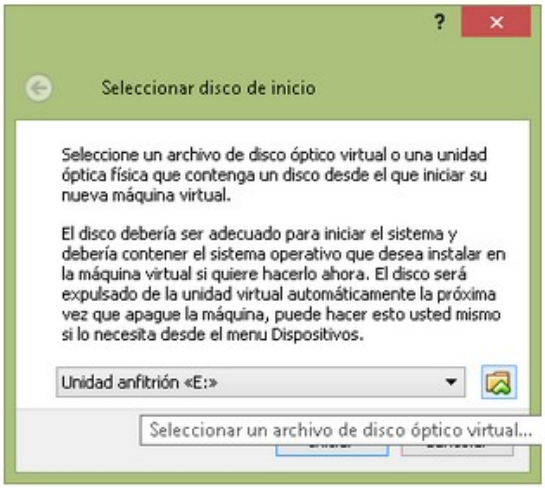
La siguiente pantalla nos permite establecer el tamaño del disco duro y la ubicación del fichero de imagen. Dejamos los valores por defecto:



Y con esto finaliza la fase de creación de la máquina virtual. Lo siguiente que veremos es en la pantalla principal la nueva máquina virtual ya creada en estado apagada. Para encenderla, podemos pulsar el botón “Iniciar” del menú superior o bien con click derecho del ratón sobre la máquina “Iniciar”:



Una vez iniciada por primera vez aparecerá una pantalla donde nos indicará que la unidad virtual de CD-ROM está vacía, y nos permite la opción de poder cargarle una ISO a la unidad para la instalación. Por tanto, en la ventana de disco de inicio pulsamos el icono de la carpeta con la flecha verde hacia arriba para poder seleccionar la ISO de Ubuntu que nos hemos descargado anteriormente:

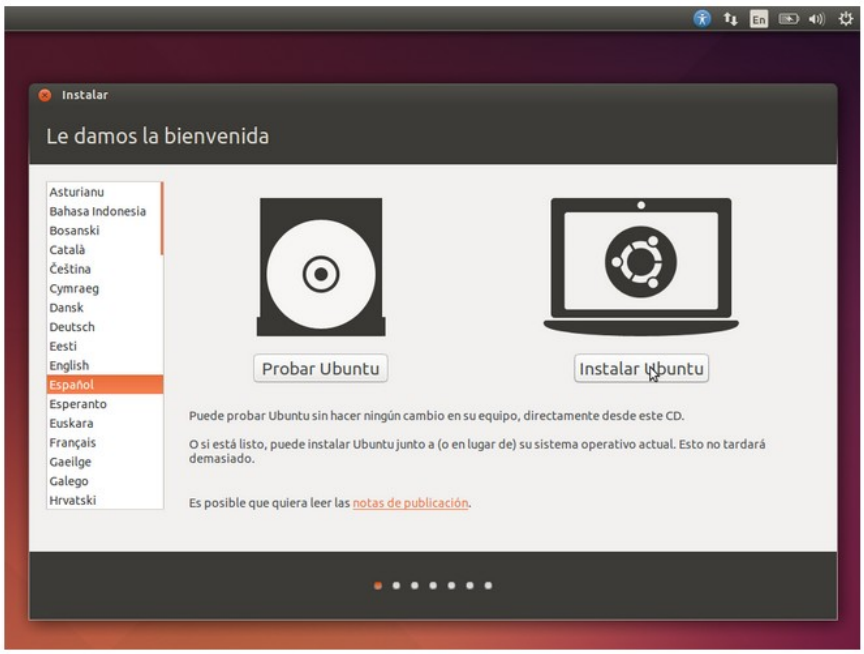


Nota: Para poner la pantalla de la máquina virtual a pantalla completa y viceversa, Host + F (la tecla host suele ser el Ctrl Derecha)

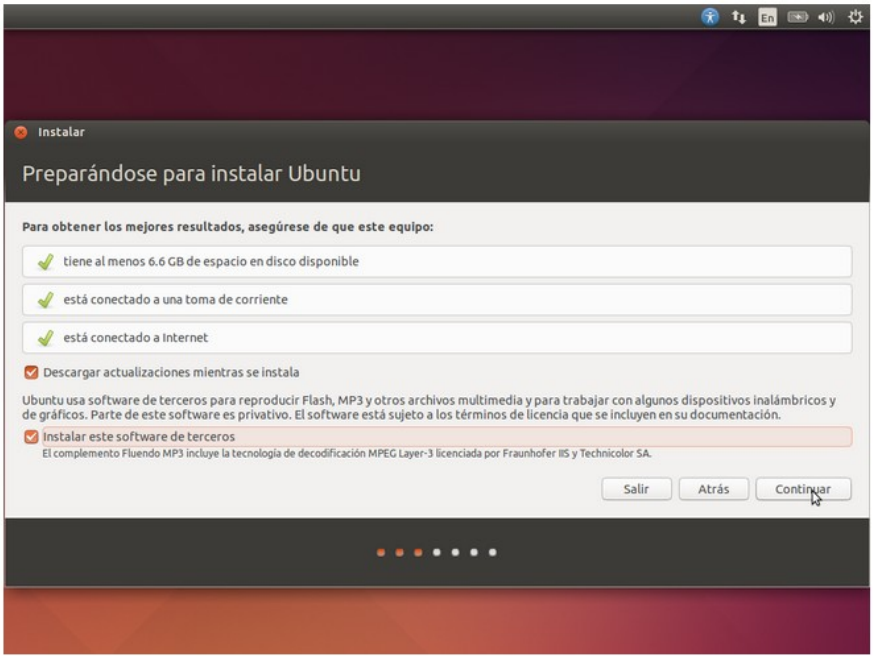
Instalando Ubuntu

La instalación del sistema operativo tiene muchas vertientes disponibles de configuración. Para este caso concreto vamos a realizar la instalación estándar.

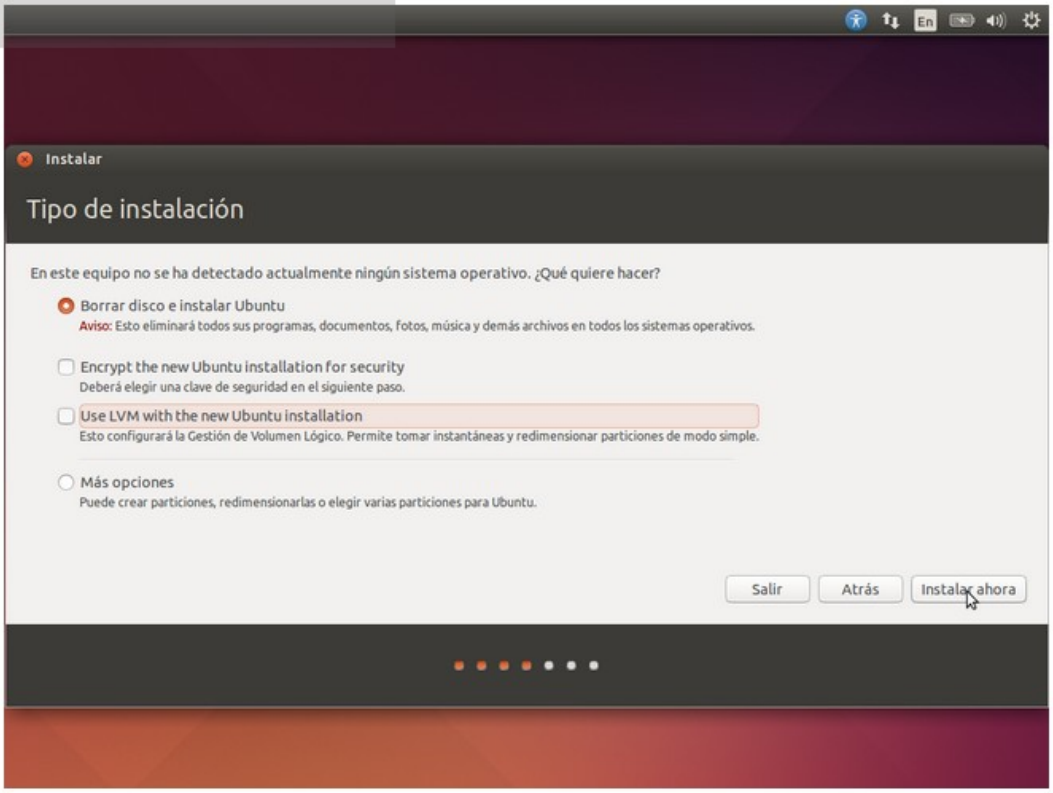
Una vez arrancada la máquina nos aparece una primera pantalla donde elegir el idioma y si queremos probar Ubuntu o instalarlo directamente. Seleccionamos la opción de instalación:



En la siguiente pantalla seleccionamos las dos casillas para actualizar el equipo mientras se instala e instalar el software de terceros:

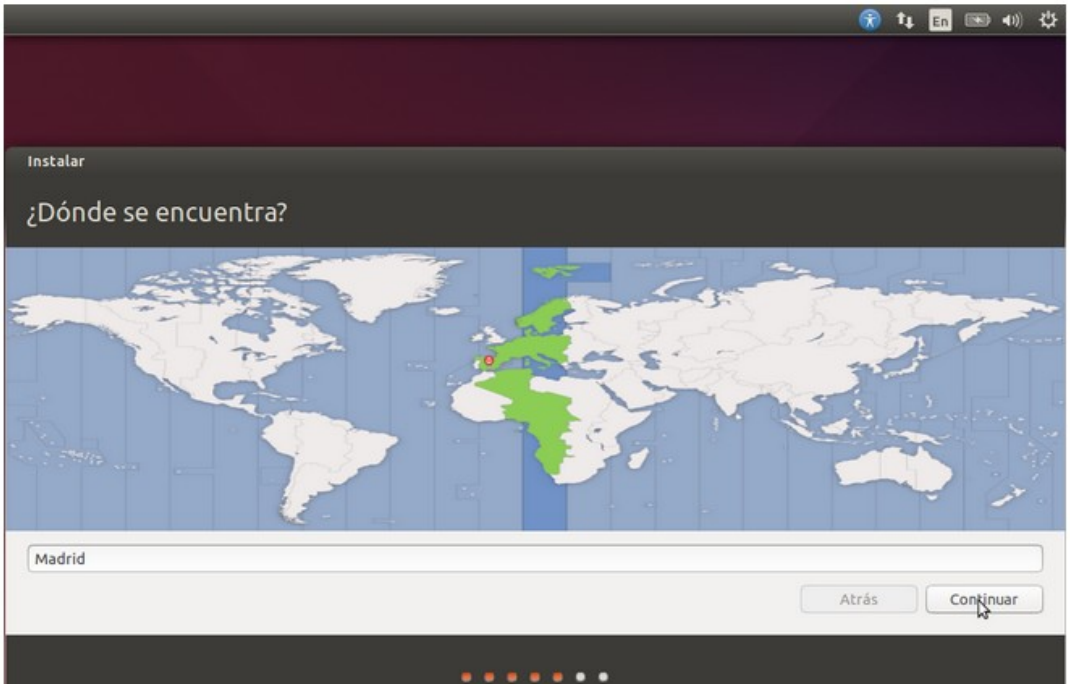


En el tipo de instalación, dejamos la opción de borrar el disco e instalar Ubuntu sin otras opciones:

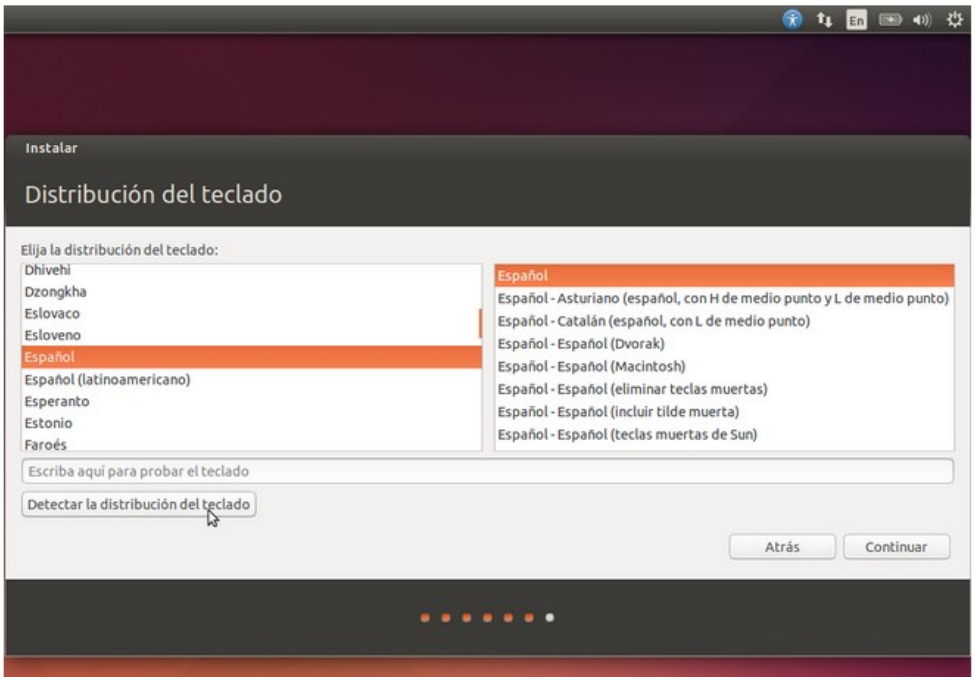


Tras darle a "Instalar ahora" nos aparecerá una ventana donde se nos indica el resumen de las particiones que va a realizar en el disco para el sistema. Aceptamos y ya procedemos a instalar.

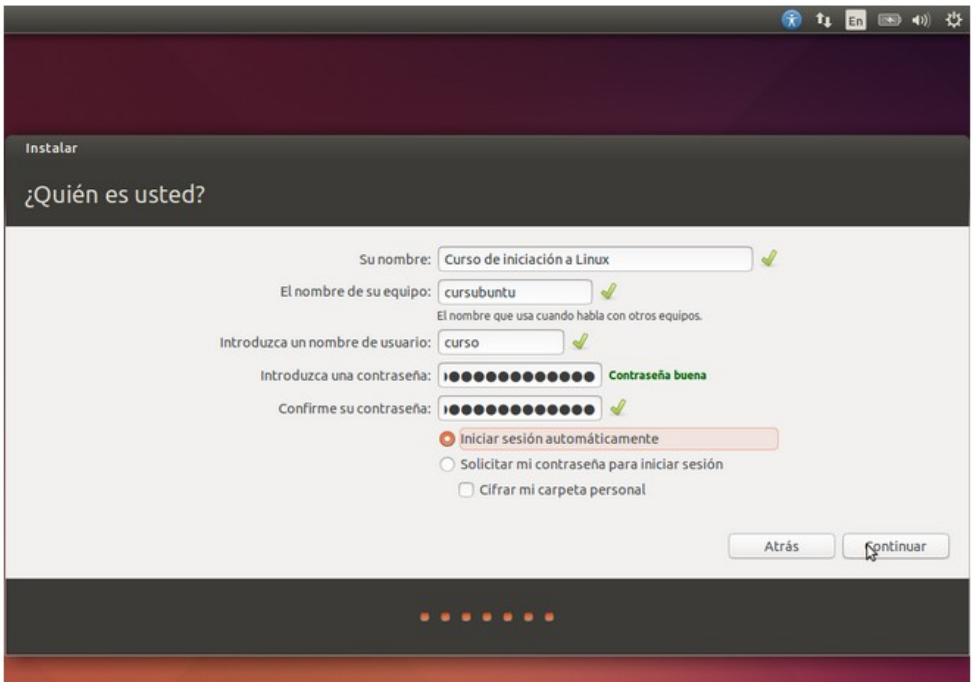
Durante la instalación terminaremos de configurar el sistema. La primera pantalla que nos aparece es la zona horaria, donde habrá que establecer el País donde nos encontremos.



La siguiente pantalla nos consulta la distribución del teclado. Si no sabemos qué distribución de teclado tenemos y queremos estar seguros basta con pulsar en el botón "Detectar la distribución de teclado" donde, con una serie de preguntas, el sistema va a detectar y establecer de forma automática qué distribución tenemos.



La última pantalla de configuración nos permite establecer tanto el nombre de la máquina como el usuario que vamos a utilizar para acceder al sistema:



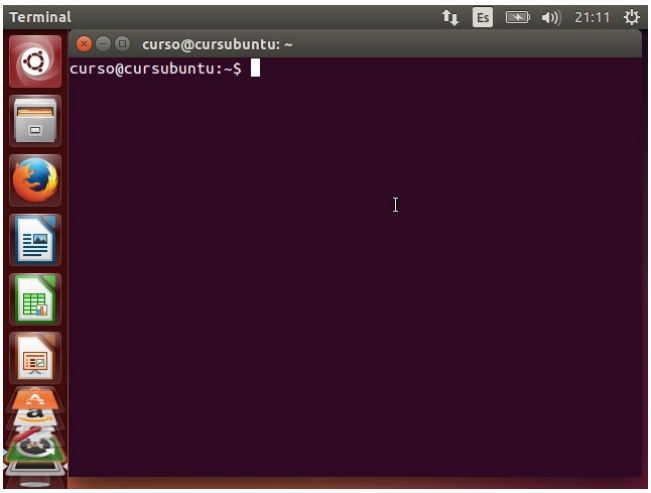
Tras esta pantalla comenzará la instalación que tarda un poco en finalizar. Una vez se haya instalado, pulsamos el botón "Reiniciar Ahora" y finaliza la instalación reiniciando el equipo.

Conectarse a Linux: Métodos de acceso al sistema operativo

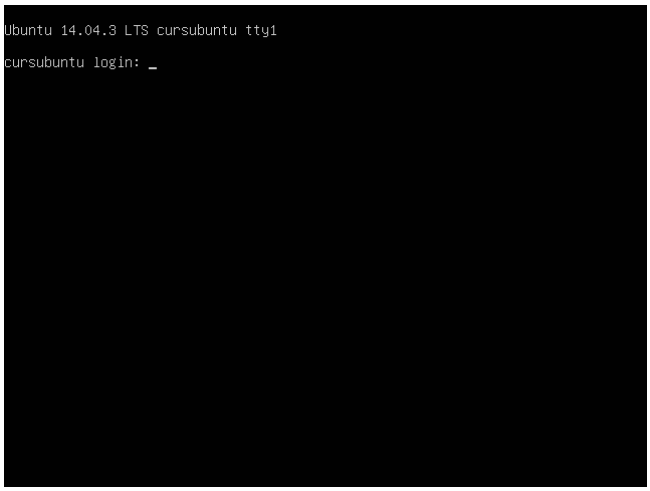
GNU/Linux permite muchas formas de poder acceder al sistema operativo, a través de interfaces gráficas, consolas virtuales e incluso a través de la red.

En este curso vamos a explicar, gracias a nuestra máquina virtual, el método de acceso local (interfaz gráfica y consola virtual) y el método de acceso en remoto a través del servicio ssh.

- Método local:
    - Interfaz gráfica: Esta interfaz es la que nos aparece por defecto en el arranque del sistema operativo Ubuntu. Nos permite ejecutar una terminal en modo gráfico
    - Terminal virtual: Este tipo de terminales se pueden visualizar en el sistema pulsando la combinación de teclas Alt + Ctrl + F1-F6. Linux crea por defecto 6 terminales virtuales para poder usar y dos displays gráficos, ubicados en Ctrl + Alt + F7-F8.
- Método remoto:
    - SSH: Ssh es el acrónimo de Secure SHell y permite el acceso a una terminal de nuestro sistema a través de la red con la característica de aportar seguridad gracias a la encriptación de sus comunicaciones. Es un servicio de estructura cliente-servidor, donde el servidor abre el puerto 22 para admitir conexiones y el cliente utiliza la aplicación para conectarse a dicho puerto.



Interfaz Gráfica



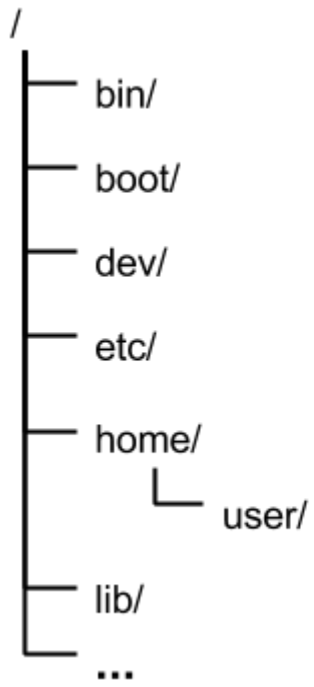
Terminal Virtual

Comprender y gestionar la estructura de directorios de Linux

Estructura de directorios Linux

Para comprender la estructura de directorios de Linux hay que analizar y conocer previamente el estándar FHS (Filesystem Hierarchy Standard). Esta norma determina como se definen los directorios principales y los contenidos del sistema operativo GNU/Linux, llegando a completarse en 1995.
Para entender cómo funciona Linux y su relación con los elementos físicos de almacenamiento, primero vamos a verlo desde el lado del software y a partir de ahí desarrollaremos la relación entre esto y los componentes físicos (particiones).

Esta es la composición del árbol de directorios de un sistema Linux:



Cada directorio que cuelga directamente del raíz (/) tiene un propósito concreto en el sistema. Según el estándar FHS, cada directorio tiene el siguiente cometido:

Directorios Importantes y sus Contenidos

FHS define algunos directorios con mucha precisión. Los más comunes definidos por FHS o utilizados por convención, son los siguientes:

/ : Los sistemas de ficheros Linux tienen su raíz en un mismo directorio conocido como sistema de ficheros raíz o directorio raíz. Todos los demás directorios se ramifican desde éste. Linux no utiliza letras de unidad sino que las particiones o discos extraíbles se montan en un punto dentro del sistema de ficheros raíz. Algunos directorios críticos deben residir siempre en la partición raíz pero otros pueden encontrarse en particiones independientes. No se debe confundir el directorio */root* con el directorio raíz.

/boot : Contiene ficheros estáticos y no compartibles relacionados con el arranque del ordenador. Algunos sistemas imponen limites particulares a */boot* , por ejemplo, en BIOS antiguas y versiones antiguas del LILO pueden requerir que */boot* se encuentre por debajo del cilindro 1024 del disco duro. También puede que se requiera que /boot sea una partición independiente.

/bin : Contiene algunos ficheros ejecutables que son accesibles para todos los usuarios y constituyen los comandos más importantes que pueden ejecutar los usuarios normales. Contiene ficheros estáticos. Sus ficheros son compartibles pero son tan importantes para el funcionamiento básico del ordenador que este directorio casi nunca se comparte. Cada cliente debe tener su directorio */bin* en local.

/sbin : Es similar a */bin* pero contiene programas que sólo ejecuta el administrador. Es estático y en teoría compartible. En la práctica sin embargo, no tiene sentido compartirlo.

/lib : Contiene bibliotecas de programa que son código compartido por muchos programas y que se almacenan en ficheros independientes, para ahorrar RAM y espacio en disco. */lib/modules* contiene módulos o drivers que se pueden cargar y descargar según necesitemos. Es estático y teóricamente compartible aunque en la práctica no se comparten.

/usr : Aloja el grueso de los programas de un ordenador Linux. Tiene un contenido compartible y estático lo que permite montarlo en modo sólo lectura. Se puede compartir con otros sistemas Linux; muchos administradores separan */usr* en una partición independiente aunque no es necesario. Contiene algunos subdirectorios similares a los del directorio raíz como */usr/bin y /usr/lib* , que contienen programas y bibliotecas que no son totalmente críticos para el funcionamiento del ordenador.

/usr/local : Contiene subdirectorios que reflejan la organización de */usr* . Aloja los ficheros que instala localmente el administrador y es un área a salvo de las actualizaciones automáticas de todo el SO. Después de la instalación de Linux debería estar vacío, excepto para determinados subdirectorios *stub*. Se suele separar en una partición para protegerlo de las reinstalaciones del SO.

/usr/X11R6 : Alberga los ficheros relacionados con el sistema X Window (entorno GUI). Contiene subdirectorios similares a los de */usr, como /usr/X11R6/bin y /usr/X11R6/lib.*

/opt : Es similar a */usr/local* pero está pensado para los paquetes que no vienen con el SO como los procesadores de texto o juegos comerciales, que se guardan es sus propios subdirectorios. El contenido de */opt* es estático y compartible. Se suele separar en su propia partición para convertirlo en un enlace simbólico a un subdirectorio de */usr/local.*

/home : contiene los datos de los usuarios y es compartible y variable. Se considera opcional en FHS, pero, en la práctica lo opcional es el nombre. El directorio */home* con mucha frecuencia reside en su propia partición.

/etc: Contiene archivos de configuración del sistema específicos del Host de todo el sistema.

/root : Es el directorio home del usuario root. Como la cuenta de root es tan crítica y específica del sistema, este directorio variable no es realmente compartible.

/var : Contiene ficheros efímeros de varios tipos, de registro del sistema, de cola de impresión, de

correo y news, etc. El contenido del directorio es variable pues algunos subdirectorios son compartibles y otros no. Se suele colocar */var* en su propia partición, sobre todo si el sistema registra una gran actividad en */var.*

/tmp : Es donde se crean los archivos temporales y variables que necesitan los programas. La mayoría de las distribuciones limpian este directorio periódicamente en el inicio. Este directorio raramente se comparte pero se suele poner en una partición independiente para que los procesos no controlados no provoquen problemas en el sistema de ficheros al ocupar demasiado.

/mnt : La finalidad de este directorio es albergar el montaje de los dispositivos. En la estructura de directorios, algunas distribuciones crean subdirectorios dentro de */mnt* para que hagan de puntos de montado; otras utilizan directamente */mnt* o incluso puntos de montado independientes de */mnt, como /floppy o /cdrom* . FHS sólo menciona */mnt* y no especifica cómo se ha de utilizar. Los medios montados en esta partición pueden ser estáticos o variables y, por norma general son compartibles.

/media : Es una parte opcional del FHS como */mnt* , pero que podría contener subdirectorios para tipos de medio específicos. Muchas distribuciones modernas utilizan subdirectorios */media* como punto de montado para los discos extraíbles.

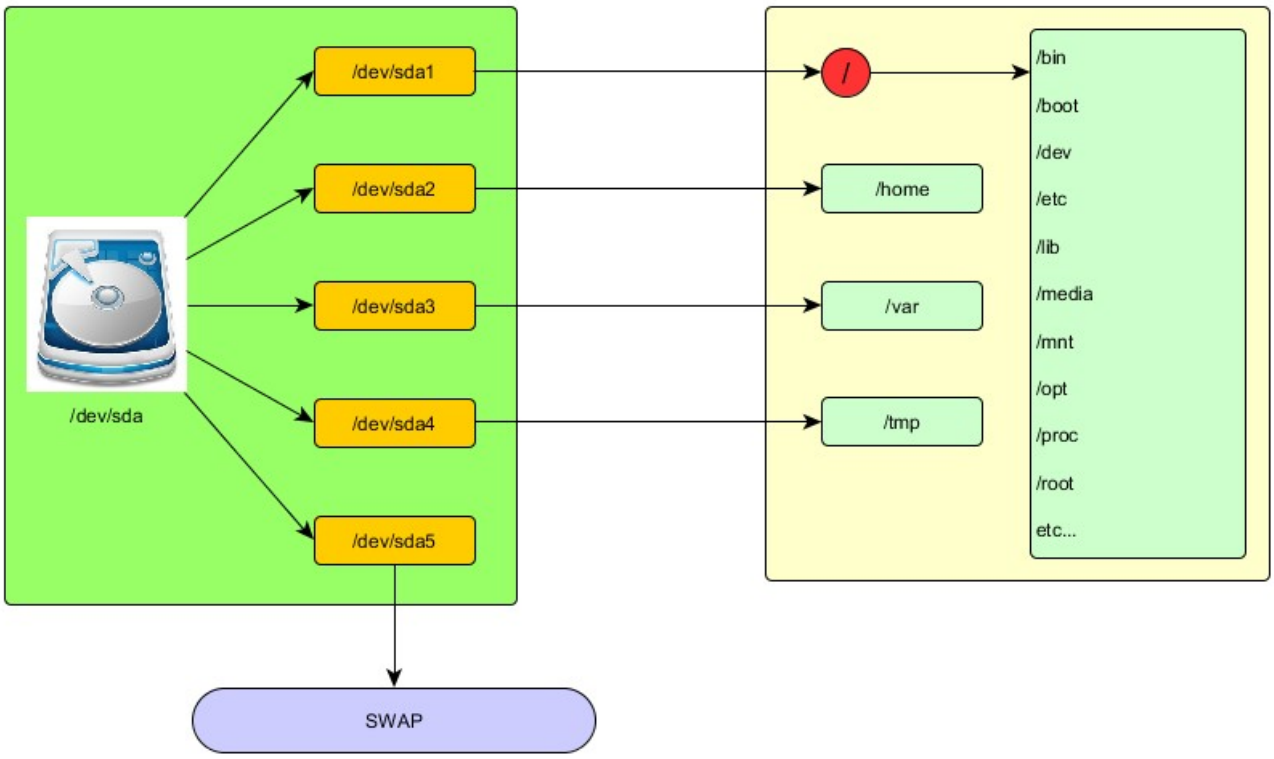
/dev: Linux trata la mayoría de los dispositivos de hardware como si fueran ficheros y el SO debe tener un lugar para estos en su sistema de ficheros. Ese lugar es el directorio */dev* que contiene un gran número de ficheros que hacen de interfaces de hardware. Con los permisos apropiados se accede al hardware del dispositivo leyendo y escribiendo en el fichero de dispositivo asociado. El kernel permite que */dev* sea un sistema de ficheros virtual creado automáticamente. El kernel y las herramientas de soporte crean sobre la marcha entradas en */dev* para adaptarse a las necesidades de los drivers específicos. La mayoría de las distribuciones emplean este recurso.

/proc : Es un directorio inusual ya que no corresponde a un directorio o partición normal sino que se trata de un sistema de ficheros virtual que proporciona acceso a ciertos tipos de información del hardware dinámicamente. Esta información no se encuentra accesible a través de */dev.*

En la administración de Linux conocer la finalidad de los directorios resulta tremendamente útil ya que si instalamos por ejemplo un programa en una ubicación equivocada, un binario colocado en */bin* c *u* ando debería estar en */usr/local/bin* puede que se sobrescriba o elimine al realizar una actualización del sistema.

Ref:
https://es.wikipedia.org/wiki/Filesystem_Hierarchy_Standard#Especificando_los_directorios_definidos_por_FHS

Tras este estudio del sistema de ficheros a nivel software, a continuación vamos a ilustrar con un ejemplo gráfico cómo se distribuyen los discos y particiones dentro del sistema de ficheros. El ejemplo consta de un disco SATA que tiene 5 particiones, y cada una de ellas albergará un directorio (o punto de montaje) de la estructura de directorios visto anteriormente:



Como se puede apreciar en la imagen, cada partición del disco duro está asociado a un directorio de la estructura de directorios única del sistema operativo, a diferencia de windows que cada partición del disco duro se le asigna una letra de unidad (C:, D:, etc) y en ella se contiene su propia estructura de directorios.

Esta forma de componer la estructura de directorios tiene una serie de ventajas en cuanto a la integridad y seguridad del sistema operativo que son:

- Soporte multi-operativo: Permite alojar varios sistemas operativos.
- Elección del sistema de ficheros: Cada sistema de ficheros ofrece diferentes características eligiéndolas según las necesidades que se tengan para esa partición.
- Control y administración del espacio en disco: Se puede controlar el acceso de los usuarios a las diferentes particiones.
- Protección de errores en el disco: Al estar dividida la partición un error físico del disco probablemente afectará a una parte del sistema operativo, dejando la posibilidad de que se pueda seguir trabajando con él e intentar recuperar el sector dañado.
- Seguridad: Puedes asegurar un sector de tu sistema de datos críticos montándolo en solo lectura. La ventaja de agregar esta característica a la partición y no a los ficheros es por cuestiones de redundancia.
- Backup: Las herramientas de copias de seguridad trabajan mejor en sistemas pequeños y aislados de tareas de escritura.

¿Qué es una shell?

El shell es el entorno que hace de intermediario entre el usuario y los recursos del ordenador, como si fuera un entorno de programación en tiempo real para ejecutar tareas. La shell aparece cuando nos logueamos en el sistema en modo consola, o bien cuando en el entorno gráfico abrimos una terminal.

La shell se compone de un prompt, que es un texto inicial que normalmente nos ofrece información útil como el usuario que está utilizando la shell, el hostname de la máquina o incluso el directorio sobre el que estamos posicionados en cada momento, y el cursor que recibe las ordenes de teclado.

Diferentes shells en Linux

Existen multitud de shells diferentes para poder interactuar con nuestro sistema operativo, aunque la más conocida y habitual en la mayoría de distribuciones es la shell bash. Cada shell cuenta con sus propias características de uso, contando con atajos de teclado, visualización en vivo de ficheros, y atajos durante la navegación entre directorios. A continuación vamos a citar las shells mas famosas y una pequeña descripción de su procedencia:

- bash (Bourne Again Shell): Se basa en los principios de shell Bourne de Unix pero se ha extendido en varios aspectos. Es la shell por defecto para la mayoria de las cuentas de usuario y es la que se tratará con más detalle en este curso.
- bsh: El shell Bourne es la shell sobre la que está basada bash. Se conoce con el nombre de BSH. Su uso no es frecuente en Linux aunque el comando bsh suele ser un enlace simbólico a bash.
- tcsh: Este shell tcsh se basa en el anterior shell C (csh). Es una shell bastante popular en algunos círculos pero no hay distribuciones de Linux que lo traigan por defecto. Aunque es similar a bash en muchos aspectos difiere en algunos aspectos de operación. Por ejemplo, no se asignan variables de entorno de la misma manera que en bash.
- csh: El original csh shell C no es muy utilizado en Linux pero si un usuario está familiarizado con csh, tcsh es un buen sustituto.
- ksh: El shell Korn (ksh) fue diseñado cogiendo las mejores opciones de la shell Bourne y el C shell. Tiene un pequeño pero dedicado numero de seguidores.
- zsh: El shell zsh Z (zsh) es la evolución de la shell Korn e incorpora características de esta última además de agregar otras.

Uso inicial de bash y variables de entorno

Para usar inicialmente la shell bash, en nuestra máquina virtual abrimos una terminal, buscando en el menú “Terminal”.

La pantalla inicial de la terminal ya veremos el prompt y el cursor para poder escribir. La shell bash cuenta con una serie de atajos de teclado que permiten la navegación más sencilla:

- Si al acceder a directorios pulsamos la tecla `TAB` se autocompleta en el caso de existir sólo una iteración o te muestra un desplegable de las diferentes iteraciones existentes.
- Las teclas del `cursor arriba y abajo` nos permiten navegar por el historial de comandos que hemos escrito anteriormente. La shell almacena los comandos ejecutados en un fichero de historial a la que se tiene acceso mediante esta combinación de teclas.
- Las teclas de `cursor derecha e izquierda` nos permiten navegar sobre lo que hemos escrito en la shell, de este modo nos permite hacer modificaciones en vivo en la línea de comandos antes de ejecutar por si en algún momento hemos escrito algo erróneo. También se puede utilizar la combinación `Ctrl + Derecha o izquierda` para saltar palabras o campos en vez de caracter a caracter.
- Existe un método para buscar en el historial de comandos mucho más afinado que con las flechas de cursor arriba y abajo. Si pulsamos la combinación de teclas `Ctrl + r` nos aparecerá un menú donde, si escribimos el comando que estamos buscando, nos aparecerá la primera iteración buscada. Si una vez encontrado, seguimos pulsando `Ctrl +r`, se seguirá realizando la búsqueda de forma inversa de esta misma iteración.

La shell, además de esto, se compone también de una serie de comandos internos y de variables de entorno que definirán parámetros en la configuración del sistema.

Variables de entorno: Las variables de entorno son muy utilizadas tanto por la shell como por otros programas para recuperar datos fundamentales del pc para su configuración. Algunas de las variables de entorno más importantes son las siguientes:

- SHELL: Define la shell que estamos usando. El valor que toma es el binario que ejecuta la shell de bash, que se ubica en /bin/bash
- USER: Define el usuario que está actualmente logueado en la shell.
- PATH: Esta variable es fundamental a la hora de ejecutar programas en Linux, ya que contiene un conjunto de rutas donde el sistema va a buscar los ejecutables. Para indicar varias rutas, cada una de ellas se separa de la otra con el carácter dos puntos “:”. De este modo, no es necesario indicarle al sistema en que ubicación se encuentra el programa para poder ejecutarlo.
- PWD: Esta variable muestra en qué directorio estamos posicionados en todo momento. Por ejemplo el uso más claro de esta variable es en el propio prompt del sistema, que se nutre de dicha variable para mostrar dónde estamos ubicados en cada momento. También el comando pwd consulta esta variable para mostrar nuestra ubicación en la estructura de directorios.

- LANG: Esta variable almacena el código de lenguaje y la codificación de caracteres que estamos utilizando.
- HOME: Esta variable almacena el home del usuario con el que estamos logueados en la shell. El home del usuario es el espacio en la estructura de directorios donde se almacenan todos los datos de un usuario. Su ubicación reside en el directorio /home, y dentro de este existen una serie de subdirectorios con los diferentes nombres de usuarios creados en el sistema. Por tanto, el contenido de esta variable siempre será del tipo "/home/usuario"
- LANGUAGE: Determina el lenguaje utilizado para el sistema.

Comandos internos: Los comandos internos de la shell tienen la particularidad que no generan un IDentificador de Proceso (PID) adicional, sino que utilizan el de la propia shell. Algunos de estos comandos internos son muy utilizados, a continuación veremos una lista de los más importantes y para que sirven:

- **alias:** Crea un nombre alternativo (alias) para un comando. Esto es útil para facilitar la entrada de comandos recurrentes que llevan varios argumentos. El ejemplo anterior muestra un alias para *rm* . Cada vez que se utilice el comando **rm** pedirá confirmación antes de borrar.

```
$ alias rm='rm -i'
```

Usar alias sin argumentos muestra que alias están configurados:

```
$ alias
alias ls='ls --color=auto'
alias rm='rm -i
```

En el ejemplo anterior hay configurado un alias para el comando **ls** con el modificador --color y el que se ha creado anteriormente para el comando **rm** .
Los alias pueden tener cualquier nombre siempre y cuando no tengan espacios o caracteres especiales, como signos de admiración o interrogación.

- **exec:** Este comando se utiliza para la ejecución de un programa especificado. Tiene la funcionalidad adicional de ejecutarse junto a la sesión de consola, no crea un nuevo proceso. Cuando se finalice la ejecución del programa se finalizará también la sesión en consola.
- **echo:** Repite el texto o el contenido de una variable introducido como argumento.

```
$ echo Prueba
Prueba
```

Devuelve la palabra Prueba

```
$ echo $PATH
```

Muestra el contenido de la variable de entorno PATH

- **env:** Sin argumentos muestra las variables de entorno y el contenido de las mismas. Se puede utilizar también para ejecutar un comando con una variable de entorno modificada temporalmente.

```
$ echo $DISPLAY
:0.0
$ env DISPLAY=:1.0 xterminal
```

En este ejemplo se ejecuta el programa xterminal con la variable DISPLAY modificada a :1.0 en vez de :0.0 que es el valor que tiene establecido. Una vez se finalice el comando la modificación de dicha variable se perderá.

- **export:** Define una variable de entorno para la sesión actual y para todas las sesiones iniciadas a partir de ésta.

```
$ export PATH=$PATH:/usr/local/bin
```

Incluye en la variable $PATH la ruta */usr/local/bin/*

- **pwd** . Muestra el directorio de trabajo actual.

```
 $ pwd
/home/usuario/
```

- **set** . Muestra las variables definidas.

```
$ set
```

- **unset** . Elimina una variable definida en la sesión.

```
$ unset NOMBRE
```

Comandos básicos

Antes de pasar a los comandos para la navegación debemos comprender un poco mejor como funciona el sistema de archivos y directorios.

Directorios y archivos

Una ruta señala la localización exacta de un archivo o directorio mediante una cadena de caracteres concreta. En líneas generales se compondrá de los nombres de los directorios que conforman el camino hasta nuestro archivo o directorio a lo largo del árbol de directorios, y finalmente estará el nombre del archivo o directorio al que se quiere llegar.

En Linux estos nombres estarán separados por el carácter delimitador "/".

El acceso a archivos y directorios se hace mediante rutas absolutas o rutas relativas.

- Las rutas absolutas señalan la ubicación de un archivo o directorio desde el directorio raíz del sistema de archivos. Por ejemplo es una ruta absoluta */home/usuario/prueba.txt* que señala la ubicación del archvo de texto *prueba.txt* desde la raíz del sistema de archivos.

```
root@maquina /home $ cat /home/usuario/prueba.txt
Esto es una prueba de fichero de texto
Linea 1
Linea 1
Linea 2
Linea 2
Linea 3
Linea 4
```

En este ejemplo dentro del directorio /home se hace uso de una ruta absoluta para visualizar el archivo de texto prueba.txt

Las rutas relativas señalan la ubicación de un archivo o directorio a partir de la posición actual en el sistema de archivos. Por ejemplo es una ruta relativa *usuario/prueba.txt* que señala al archivo *prueba.txt* dentro del directorio */home* en la ubicación actual.

```
root@maquina /home $ cat usuario/prueba.txt
Esto es una prueba de fichero de texto
Linea 1
Linea 1
Linea 2
Linea 2
Linea 3
Linea 4
```

En el ejemplo anterior se hace uso de una ruta relativa teniendo en cuenta que se está dentro del directorio /home.

A continuación veremos los comandos necesarios para la navegación y visualización de ficheros y directorios, así como la gestión de directorios y ficheros:

- **touch:** Se utiliza para modificar la fecha de un archivo, aunque también se puede utilizar para crear un fichero vacío. Si se usa sin argumentos, touch modifica la fecha y la hora de creación y modificación de un archivo a los valores actuales del sistema. Las opciones principales son:

```
$ touch hola
$ ls -l
-rw-r--r-- 1 blanca blanca    0 feb 23 16:22 hola
```

- -m: Se usa para modificar únicamente la fecha de modificación
- -a: Se usa para modificar únicamente la fecha de acceso.
- -t: Modifica otros valores de tiempo.

cd: Se utiliza para moverse por el árbol de directorios de linux. Cuando no se especifica ningún directorio como argumento va al directorio personal. Para acceder al directorio Descargas dentro del directorio personal:

```
$ cd /home/usuario/Descargas/
```

- **mkdir**. Se utiliza para crear directorios.
- -p: Crea un árbol de directorios recursivamente, sin necesidad de crearlos uno a uno:

```
$ mkdir -p documentos/curso/lpi/101
```

- -m: Permite especificar los permisos del directorio en el momento de creación.

```
$ mkdir -m 700 personal
$ ls -l
drwx------ 2 usuario usuario 4,0K feb 23 16:28 personal
```

- **rmdir.** Permite borrar directorios vacíos. Para borrar el directorio creado en el ejemplo anterior:

```
$ rmdir personal
```

El comando **ls** se utiliza para listar archivos y contenido de un directorio. Tiene varias opciones, entre las que destacaremos las siguientes:

- **-l** muestra detalles sobre los archivos.
- **-s** muestra el tamaño en Kb.
- **-d** muestra las propiedades de un directorio, no su contenido
- **-h** muestra el tamaño de los ficheros y directorios en formato de lectura humano (por ejemplo 1K, 234M, 2G)
- **-a** muestra todos los ficheros, incluidos los ocultos (en Linux los ficheros ocultos comienzan con un punto [.])
- **-R** lista los directorios de forma recursiva.

Un ejemplo del comando **ls -lh** :

```
$ ls -lh
total 1,3M
drwxr-xr-x 3 root root 4,0K 2011-12-15 08:26 acpi
-rw-r--r-- 1 root root 3,0K 2011-04-26 00:52 adduser.conf
drwxr-xr-x 2 root root 12K 2013-02-25 13:12 alternatives
-rw-r--r-- 1 root root 395 2010-06-20 10:11 anacrontab
drwxr-xr-x 6 root root 4,0K 2011-04-26 00:55 apm
drwxr-xr-x 3 root root 4,0K 2012-10-01 10:21 apparmor
drwxr-xr-x 8 root root 4,0K 2012-10-01 10:22 apparmor.d
drwxr-xr-x 5 root root 4,0K 2011-10-24 08:16 apport
drwxr-xr-x 6 root root 4,0K 2013-01-25 08:32 apt
-rw-r----- 1 root daemon 144 2010-06-27 21:38 at.deny
drwxr-xr-x 3 root root 4,0K 2011-04-26 01:03 avahi
-rw-r--r-- 1 root root 1,9K 2011-03-31 23:20 bash.bashrc
-rw-r--r-- 1 root root 58K 2011-04-04 22:37 bash_completion
drwxr-xr-x 3 root root 4,0K 2012-10-01 10:23 bash_completion.d
```

Esta información puede dividirse en columnas, de izquierda a derecha:

La primera columna muestra el tipo y los permisos del archivo.

La segunda columna muestra el número de enlaces de referencia (hardlinks) para el archivo.

La tercera y la cuarta muestran el propietario y el grupo a los cuales pertenece el archivo.

La quinta muestra el tamaño, por defecto en bytes.

La sexta y la séptima muestran la fecha y la hora de la última modificación en el archivo.

La octava columna muestra el nombre del archivo. Si el archivo fuera un enlace simbólico, se mostrará una flecha apuntando al archivo al que hace referencia dicho enlace.

Permisos y propietarios

Ya hemos visto que con el comando ls podemos ver tanto los permisos como el propietario y grupo al que pertenece un fichero o directorio. A continuación vamos a explicar la política de identidad de los ficheros contenidos en el sistema Linux:

Los ficheros y directorios tienen 3 tipos de identidad: La individual, la de grupo y la del resto del sistema. Por tanto, se puede indicar que un fichero o directorio en el sistema Linux tiene un perfil de propietario, un perfil de grupo y un perfil de sistema. Cada uno de estos “perfiles” se le pueden asignar unas “normas de uso” llamado permisos. Los permisos se componen de tres elementos: lectura, escritura y ejecución. De este modo, la combinación de perfiles con sus respectivos permisos nos ofrecen una administración eficiente de los diferentes ficheros y directorios existentes en nuestro sistema.

Existe una pequeña apreciación en el permiso de ejecución, ya que hay que discriminar su uso a ficheros y directorios, puesto que no realizan la misma función en ellos. Para los ficheros, el permiso de ejecución hace que el fichero sea ejecutable. Esto se suele hacer en shell scripts y otros ficheros especiales del sistema. Sin embargo para los directorios el permiso de ejecución permite que se pueda acceder al directorio (ya que un directorio, por lógica, no se puede ejecutar).

Teniendo esta regla en cuenta, lo habitual es que todos los directorios tengan el permiso de ejecución incluido en ellos para, al menos, el propietario del directorio.

Los permisos de lectura, escritura y ejecución se muestran con las siglas r, w y x respectivamente. Estos permisos se pueden observar en el comando ls al inicio:

```
# ls -l /etc/X11/
total 80
drwxr-xr-x 2 root root        4096 mar  6 09:41 app-defaults
drwxr-xr-x 2 root root        4096 mar  6 09:09 cursors
-rw-r--r-- 1 root root          18 mar  6 09:04 default-display-manager
drwxr-xr-x 4 root root        4096 mar  6 09:00 fonts
drwxr-xr-x 3 root root        4096 mar  6 09:33 ja_JP.eucJP
drwxr-xr-x 3 root root       4096 mar   6 09:33 ja_JP.UTF-8
-rw-r--r-- 1 root root       17394 dic  3  2009 rgb.txt
lrwxrwxrwx 1 root root          13 mar  6 08:44 x -> /usr/bin/Xorg
drwxr-xr-x 3 root root        4096 mar  6 09:00 xinit
drwxr-xr-x 3 root root        4096 jul  3  2012 xkb
-rwxr-xr-x 1 root root         709 abr  1  2010 Xreset
drwxr-xr-x 2 root root        4096 mar  6 08:43 Xreset.d
drwxr-xr-x 2 root root        4096 mar  6 08:43 Xresources
-rwxr-xr-x 1 root root        3730 ene  20 2012 Xsession
drwxr-xr-x 2 root root        4096 mar  19 10:15 Xsession.d
-rw-r--r-- 1 root root         265 jul   1  2008 Xsession.options
-rw-r--r-- 1 root root         601 mar   6 08:43 Xwrapper.config
```

Además de esto, la primera letra representa el tipo de archivo y determinará como Linux interpretará el archivo, como un dato, un directorio, etc.

El tipo puede ser:

d : Directorio: Contienen nombres de archivos que apuntan a inodos de disco.

l : Enlace simbólico. Este archivo contiene el nombre de otro archivo o directorio.

c : Dispositivo especial de caracteres. Un archivo que corresponde a un dispositivo de hardware el cuál transfiere los datos en unidades de un bit.

p : Se le denomina canal, ya que permite tener disponibles dos programas de linux y que se puedan comunicar uno con el otro. Uno abre el canal para leer y otro para escribir, permitiendo que los datos sean transferidos de uno a otro.

s : Socket. Es similar al llamado canal (p), pero éste permite redireccionar enlaces.
- : Archivo convencionalDebe ser texto, un archivo ejecutable, gráficos, etc.

Los restantes caracteres de la cadena se dividen en 3 grupos de 3 caracteres. El primer grupo, controla el acceso al fichero del propietario, el segundo grupo controla el acceso del grupo y el tercero controla los accesos de todos los demás usuarios. En cada uno de los tres caracteres la cadena de permisos determina la presencia o ausencia de cada uno de los tres tipos de acceso: lectura, escritura y ejecución.

Si se incluye el permiso de ejecución el fichero se podrá ejecutar como un programa y definir el bit de ejecución en un fichero que no es un programa no tiene ningún tipo de consecuencia. Si en la cadena de permisos aparece un guión (-) quiere decir que el permiso está ausente. La presencia del permiso se indica con una **r** para lectura, un **w** para la escritura y una **x** para la ejecución.

Por tanto, la cadena de permisos *rwxr-xr-x* significa que tanto el propietario del fichero como los miembros de su grupo y todos los demás usuarios pueden leer y ejecutar el fichero, pero sólo su propietario tiene permisos de escritura sobre él.

Linux codifica en formato binario la información de los permisos que se puede expresar como un único número de 9 bits. Este número se suele expresar en formato octal (base 8) debido a que un número en base 3 tiene 3 bits de longitud, lo que significa que la representación en base 8 de una cadena de permisos tiene 3 dígitos de longitud que corresponden al propietario, el grupo y los permisos globales. Los permisos de lectura, escritura y ejecución se corresponden con cada uno de los bits. El resultado es que se pueden determinar los permisos añadiendo número en base 8.

Permisos de ejemplo y sus usos:

Cadena de permisos	Códigooctal	Significado
rwxrwxrwx	777	Permisos de lectura, escritura y ejecución para todos los usuarios
rwxr-xr-x	755	Permisos de lectura y ejecución para todos los usuarios, el propietario también tiene permisos de escritura
rwxr-x---	750	Permisos de lectura y ejecución para el propietario y el grupo, permisos de escritura para el propietario y sin acceso al fichero para todos los demás
rwx------	700	Permisos de lectura, escritura y ejecución para el propietario, los demás no tendrán acceso.
rw-rw-rw-	666	Permisos de lectura y escritura para todos los usuarios, nadie tiene permiso de ejecución.
rw-rw-r--	664	Permisos de lectura y escritura para el propietario y el grupo y de solo lectura para el resto
rw-rw----	660	Permisos de lectura y escritura para el propietario y el grupo, y no hay permisos para el resto de usuarios
rw-r--r--	644	Permisos de lectura y escritura para el propietario, permisos de solo lectura para el resto
rw-r-----	640	Permisos de lectura y escritura para el propietario y de solo lectura para el grupo, no hay permisos para los demás
rw-------	600	Permisos de lectura y escritura para el propietario y nadie más tiene permisos
r--------	400	Permiso de lectura para el propietario y nadie mas tiene permisos

El permiso de ejecución tiene sentido para los ficheros normales pero no para el resto. Los directorios utilizan el bit de ejecución de otra manera: si está definido significa que se puede buscar en el contenido del directorio. Por eso es difícil ver un directorio en el que el bit de ejecución no se defina junto al de lectura. Si a un usuario se le da permiso para escribir en un directorio, este usuario podrá crear, borrar o renombrar los ficheros del directorio aunque no sea el propietario de los ficheros y no tenga permisos para escribir en ellos.

Los enlaces simbólicos siempre tienen permisos 777, sin embargo, este acceso se aplica sólo al propio fichero del enlace y no al fichero enlazado. En cambio, los permisos del enlace de referencia afectan al fichero enlazado.

La mayoría de las reglas de permisos no se aplican a root, por lo que puede leer o escribir en cualquier fichero del ordenador, incluso en aquellos que tienen permisos 000, ya que root puede cambiar los permisos de cualquier fichero. Hay algunos ficheros que pueden ser inaccesibles para root, pero sólo porque existe una restricción subyacente, como por ejemplo que un disco duro no esté instalado en el ordenador.

Modificar permisos y propietarios

Cambiar el propietario de un fichero

El superusuario (root) puede cambiar el propietario de un fichero mediante el comando chown cuya sintaxis es la siguiente:

```
# chown [opciones] [nuevo_propietario] [:nuevo_grupo] nombres_ficheros
```

Las opciones más usuales del comando chown son:

- **-R:** Cambia el permiso de archivos que están en subdirectorios del directorio en el que estés en ese momento.
- **-C** : Cambia el permiso para cada archivo.
- **-f:** cuando es incapaz de cambiar la titularidad del archivo, previene al comando chown de mostrar mensajes de error.

En la sintaxis de este comando, los argumentos "nuevo propietario" y" nuevo grupo" se refieren al nuevo propietario y el nuevo grupo del fichero. Se pueden proporcionar ambos pero sólo se puede omitir uno de ellos. El comando chown admite muchas opciones, pero la más utilizada es -R o –recursive , que permite cambiar de propiedad un árbol de directorios completo. Sólo root puede utilizar el comando chown ; si trata de utilizarlo un usuario normal devolverá un mensaje de error.

Cambiar el grupo de un fichero

El comando **chgrp** cambia el grupo de un fichero de un archivo o directorio. Este comando lo pueden ejecutar tanto el usuario root como los usuarios normales. Los usuarios sólo pueden cambiar el grupo de un fichero por un grupo al que pertenezcan. Su sintaxis es la siguiente:

```
# chgrp [opciones] nuevo_grupo nombres_ficheros
```

Este comando acepta muchas opciones, entre las que se incluyen **-R o –recursive.** Esta herramienta es un subconjunto de funcionalidades de **chown** , pero la pueden utilizar los usuarios normales.

Las opciones más usuales del comando **chgrp** son las siguientes:

-R: Cambia el permiso de archivos que están en subdirectorios del directorio en el que estés en ese momento.

-C : Cambia el permiso para cada archivo.

-f : Esta opción lo que hace es forzar el cambio. No informa de mensajes de errores.

Cambiar Permisos de un Archivo

Con el comando **chmod** se pueden modificar los permisos de un fichero. Este comando se puede ejecutar de varios modos para obtener el mismo efecto. Su sintaxis es:

```
chmod [opciones] [modo[,modo...]] nombre_fichero…
```

La opción **–recursive o -R** modifica los permisos de un árbol de directorios.

Se puede especificar el modo de dos formas básicas: con un número en octal o con un modo simbólico, que es un conjunto de códigos relacionados con la representación en cadena de los permisos.

```
chmod 644 report.txt
```

Además de modificar los 3 dígitos de permisos del archivo, si al comando anterior se le añade un cuarto bit al principio se pueden definir permisos especiales: si se añade un 4, se definirá el bit de la ID de usuario (SUID); si se añade un 2, se definirá la ID de grupo (SGID); y si se añade un 1 se definirá el sticky bit.

Si se omite el primer dígito como en el ejemplo anterior Linux tomará los 3 bits como bits para permisos y no como bits para permisos especiales.

El modo simbólico consta de tres componentes: un código que indica el conjunto de permisos que se quieren modificar, un símbolo que indica si se desea añadir, borrar o definir el modo del valor indicado y un código que especifica cuál debería ser el permiso. Estos códigos distinguen entre mayúsculas y minúsculas:

Código conjunto permisos	del deSignificado	Código tipo cambio	de deSignificado	Permisos modificar código	para elSignificado
u	propietario	+	añadir	r	lectura
g	grupo	-	eliminar	w	escritura
o	global	=	definir igual que	x	ejecución
a	todos			X	ejecutar solo si el fichero es un directorio o tiene permisos de ejecución
				s	SUID o SGID
				t	sticky bit
				u	permisos del propietario existentes
				g	permisos del grupo existente
				o	permisos globales existentes

Para utilizar los parámetros de los permisos simbólicos, se combinarán uno o más códigos de la primera columna con un símbolo de la tercera columna y uno o más códigos de la quinta columna. Se pueden combinar varios parámetros, separándolos por comas:

Ejemplos de permisos simbólicos con **chmod:**

Comando	Permisos Iniciales	Permisos Finales
chmod a+x program	rw-r–r–	rwxr-xr-x
chmod ug=rw report.txt	r——–	rw-rw—-
chmod o-rwx bigprogram	rwxrwxr-x	rwxrwx—
chmod g=u report.txt	rw-r–r–	rw-rw-r–
chmod g-w,o-rw report.txt	rw-rw-rw-	rw-r—–

Encontrar ficheros y directorios

Herramientas para Localizar Ficheros

Lo más habitual para localizar ficheros es hacerlo por su nombre, pero a veces se pueden localizar por otros criterios, como por ejemplo la fecha de modificación. Los comandos de localización de ficheros de Linux pueden buscar en un árbol de directorios un fichero que coincida con el criterio especificado e incluso podrían rastrear el sistema completo.

El Comando find

El comando **find** implementa un sistema de fuerza bruta para localizar ficheros ya que realiza una búsqueda por todo el árbol de directorios especificado para localizar los que satisfagan el criterio especificado. **find** tiende a ser lento pero es flexible y tiene una mayor probabilidad de éxito.

```
find [ruta...] [expresión...]
```

Se le pueden especificar una o más rutas para restringir sus operaciones. La expresión es un modo de especificar lo que queremos buscar; las expresiones más comunes permiten buscar por varios criterios:

- Buscar por nombre de fichero: Utilizando la expresión **-nombre patrón** se buscarán los ficheros que coincidan con el patrón especificado. Se pueden emplear comodines delimitando el patrón entre comillas.
- Buscar por el modo de permiso: La expresión **-perm modo** permite localizar ficheros que tengan ciertos permisos. El modo se puede expresar simbólicamente o en modo octal si va precedido de un mas (+). El comando **find** localizará los ficheros en los que estén definidos todos los bits de permisos especificados.
- Buscar por el tamaño del fichero: Se puede realizar una búsqueda de un fichero de un tamaño determinado con la expresión *-size n* , donde “n” especifica en bloques de 512 bytes (se puede modificar con una letra que sirva de código para el valor, por ejemplo, k para kilobytes).
- Buscar por grupo: La expresión **-gid GID** , busca los ficheros cuya ID de grupo es GID. La opción **-group** nombre localiza ficheros cuyo grupo tiene el nombre indicado. Esta última opción suele ser más fácil de utilizar pero la primera es útil en caso de que la GID se haya quedado huérfana y no tenga nombre.
- Buscar por ID de usuario: **-uid UID** busca los ficheros cuyo propietario tiene UID como ID de usuario. **-user nombre** busca los ficheros cuyo propietario es nombre. Esta última opción suele ser más fácil de utilizar pero la primera es útil en caso de que la UID se haya quedado huérfana y no tenga nombre.
- Restringir la profundidad de la búsqueda: Para limitar el número de subdirectorios en los que buscar se emplea la expresión **-maxdepth niveles** .

find tiene muchas variantes y opciones adicionales, pero por ejemplo si se desean localizar todos los

ficheros fuente en código #C con nombres que normalmente acaban en *.c* y se encuentran el directorio *home* , se ejecutará:

```
find /home -name "*.c"
```

Esta expresión devolverá todos los ficheros que coincidan con el criterio de búsqueda.

Los usuarios normales pueden utilizar **find,** pero si carecen de permisos para listar el contenido de un directorio, **find** devolverá un error junto con el nombre del directorio.

El Comando locate

Esta utilidad funciona de forma similar a **find** pero difiere en dos aspectos importantes:

- Es mucho menos sofisticado en sus opciones de búsqueda ya que se utiliza para buscar por el nombre de fichero con lo que el programa devuelve todos los ficheros que contienen la cadena especificada.
- Trabaja con una base de datos que se encarga de mantener. Normalmente, se incluye una tarea cron que llama a **locate** para que actualice su base de datos periódicamente. El comando **updatedb** sirve para actualizar la base de datos manualmente. Es importante que se mantenga actualizada, ya que en caso contrario, al realizar las búsquedas puede que devuelva nombres de ficheros que ya no existan o que no localice ficheros recientes.

Al trabajar con una base de datos es más rápido que **find** , sobre todo en las búsquedas por todo el sistema. También es probable que devuelva muchas falsas alarmas, especialmente si se quiere localizar un fichero con un nombre corto. Algunas distribuciones utilizan **slocate** en vez de **locate,** que incluye características de seguridad que impiden que los usuarios vean los nombres de los ficheros o de los directorios a los que no tienen acceso. En la mayoría de los sistemas con **slocate, locate** es un enlace a **slocate.**

El Comando whereis

Busca los ficheros en un conjunto restringido de localizaciones. Esta herramienta no busca en directorios de usuario pero es un modo rápido de localizar ejecutables de programas y ficheros relacionados como ficheros de documentación o configuración. El programa **whereis** devuelve los nombres de fichero que comienzan por aquello que se escribe como criterio de búsqueda, incluso los ficheros que contienen extensiones, con frecuencia suele hallar los ficheros de configuración de */etc* , paginas MAN y ficheros similares. Para localizar por ejemplo el programa **ls,** se escribirá:

```
whereis ls
ls: /bin/ls /usr/share/man/man1/ls.1.bz2
```

El Comando which

Está considerado comando de búsqueda pero es poco potente. Busca la ruta del comando escrito y la lista completa, desde la primera coincidencia que encuentre. La opción **-a** permite buscar todas las coincidencias. Puede resultar útil, por ejemplo, para saber la ruta completa de un programa que se desee llamar desde un script.Realiza la búsqueda solamente en los directorios definidos en la variable del entorno PATH.

```
$ which xterm
/usr/bin/xterm
```

Visualización de ficheros

Existen múltiples formas de visualizar ficheros con Linux, con diferentes tipos de criterios a la hora de visualizarlos.

A continuación vamos a ver algunos de estos comandos para la visualización de ficheros:

- **cat:** Concatena ficheros o la entrada estándar, y lo muestra a la salida estándar; aunque el uso mas habitual de este comando es para mostrar ficheros.

```
$ cat prueba.txt
Esto es una prueba de fichero de texto
Linea 1
Linea 2
Linea 3
```

El comando cuenta con varias opciones, entre las que se destacan las mas importantes:

-b añade el número de línea en la salida. Si hay líneas vacias, no las cuenta.

-E muestra el signo dolar ($) al final de cada línea. Util para scripts.

-n similar a la opción **-b** con la diferencia que si cuenta las líneas vacias.

-T muestra el simbolo **^I** para las tabulaciones en el fichero.

- **tac:** Tiene la misma función del comando **cat** pero muestra el contenido de forma inversa.

```
tac prueba.txt
Linea 3
Linea 2
Linea 1
Esto es una prueba de fichero de texto
```

head: Este comando muestra por defecto las diez primeras líneas de un fichero dado. Si se le pasan mas de un fichero al comando, este mostrará las diez primeras líneas de cada fichero indicando el nombre del mismo con una cabecera.

La opción **-n** *X* indica que muestre *X* líneas de cada fichero, en vez de diez por defecto.

La opción **-c** *K* muestra los primeros *K* bytes de cada fichero.

```
head -n2 prueba.txt
Esto es una prueba de fichero de texto
Linea 1
```

tail: Este comando muestra por defecto las diez últimas líneas de un fichero dado. Al igual que con el comando **head** , si se le pasa mas de un fichero mostrará las diez primeras líneas de cada fichero. Cuenta con las siguientes opciones:

La opción **-n** *X* indica que muestre *X* líneas de cada fichero, en vez de diez por defecto.

La opción **-c** *K* muestra los primeros *K* bytes de cada fichero.

La opción **-f** (follow) hace que el comando no finalice, haciendo que se muestre por la salida estándar los datos que se van introduciendo en el fichero. Esta opción tremendamente útil para visualizar en tiempo real cualquier log del sistema, haciendo que sea una opción muy utilizada por los administradores.

```
tail -n2 prueba.txt
Linea 2
Linea 3
```

nl: Numera las líneas en los ficheros. Algunas de sus opciones son las siguientes:

Con las opciones **-ba** se consigue que se numeren todas las líneas.

Con las opciones **-bt** se consigue que se numeren únicamente las líneas que no estén en blanco.

```
$ nl prueba.txt
     1    Esto es una prueba de fichero de texto
     2    Linea 1
     3    Linea 2
     4    Linea 3
```

uniq: Este comando muestra por defecto el contenido de archivos eliminando las líneas consecutivas que esten repetidas. Algunas de sus opciones son:

-u muestra únicamente las líneas que no se repiten.

-d muestra únicamente la línea repetida.

```
$ cat prueba.txt
Esto es una prueba de fichero de texto
Linea 1
Linea 1
Linea 2
Linea 2
Linea 3
Linea 4
```

```
$ uniq prueba.txt
Esto es una prueba de fichero de texto
Linea 1
Linea 2
Linea 3
Linea 4
```

sort: Ordena alfabéticamente. Con la opción **-n** ordena numéricamente y la opción **-r** invierte el resultado.

```
$ sort prueba.txt
Esto es una prueba de fichero de texto
Linea 1
Linea 1
Linea 2
Linea 2
Linea 3
Linea 4
```

```
$ sort -r prueba.txt
Linea 4
Linea 3
Linea 2
Linea 2
Linea 1
Linea 1
Esto es una prueba de fichero de texto
```

fmt: Formatea un texto para determinado número de caracteres por línea. Por defecto es 75.

```
$ cat parrafo.txt
No nos podemos meter en la piel de nadie, por mucho que nos parezca haberlo logrado mediant
e un espejismo momentáneo de fusión. Cada ser es radicalmente distinto a otro cualquiera, a
unque a veces estallemos al mismo tiempo, como las olas que se persiguen y coinciden un ins
tante en su cumbre de espuma.
```

```
$ fmt parrafo.txt
No nos podemos meter en la piel de nadie, por mucho que nos parezca
haberlo logrado mediante un espejismo momentáneo de fusión. Cada ser
es radicalmente distinto a otro cualquiera, aunque a veces estallemos
al mismo tiempo, como las olas que se persiguen y coinciden un instante
en su cumbre de espuma.
```

Las opciones más utilizadas son:

- **w** : Indica el número de caracteres por línea.

- **s** : Divide líneas grandes, pero no las rellena.

- **u** : Un espacio entre palabras y dos espacios entre sentencias.

```
$ fmt -w 45 -u -s parrafo.txt
No nos podemos meter en la piel de nadie,
por mucho que nos parezca haberlo logrado
mediante un espejismo momentáneo de
fusión. Cada ser es radicalmente distinto a
otro cualquiera, aunque a veces estallemos
al mismo tiempo, como las olas que se
persiguen y coinciden un instante en su
cumbre de espuma.
```

pr: Prepara y formatea un archivo para su impresion desde la línea de comandos. Por defecto es de 66 líneas por 72 caracteres de ancho, modificados por **-l** y **-w** , respectivamente.

```
$ pr prueba.txt

2013-03-21 20:58                prueba.txt                Página 1


Esto es una prueba de fichero de texto
Linea 1
Linea 2
Linea 3
```

Formatea el archivo *prueba.txt* para su impresión desde línea de comandos.

Edición de ficheros

En el apartado de edición de ficheros, Linux cuenta con varias herramientas para esta función. Las más famosas con vim y nano, aunque existen otras como mcedit, emacs o joe. Para este curso vamos a ver nano, ya que es un editor bastante sencillo de utilizar, y vim para el uso más profesional

Nano: Es un sencillo editor de textos que viene instalado en la mayoría de distribuciones Linux de uso profesional y doméstico. Debido a su sencillez, cualquier usuario poco experimentado con la terminal puede aprender a utilizar este editor de textos gracias a las líneas de ayuda inferiores que se muestran durante la ejecución de la aplicación. El uso del comando es bastante sencillo, basta con escribir nano nombre_archivo y pasaremos a la pantalla de edición de la aplicación. Durante la ejecución de la aplicación podremos modificar todo el contenido del texto, aunque los cambios no se harán efectivos hasta que salvemos. Nano tiene una serie de atajos que vienen explicados en la ayuda inferior. El carácter ^ representa la tecla < **Ctrl** >. Asi, la combinación de teclas < **Ctrl** + **o** > nos permiten guardar los cambios que hemos realizado en el fichero. Un uso interesante de los atajos son la combinación de teclas < **Ctrl** + **k** > y < **Ctrl** + **u** >, que nos permiten cortar y pegar texto respectivamente.

Vim: Este potente editor de texto nos permite la edición de forma profesional de un fichero. Su uso es bastante mas complicado que Nano, ya que tiene diferentes “modos” de uso, pero esto otorga a la aplicación unas mejoras en cuanto a la gestión del sistema durante el uso de la aplicación. Hay que decir que vim es una evolución de la herramienta vi, y hereda de esta la funcionalidad básica de la herramienta. A continuación procedemos a explicar un poco su uso:

El uso de la herramienta es similar a nano, tecleando vim nombre_archivo pasaremos a la pantalla de edición de la aplicación. Una vez ahí, vamos a comprender los diferentes modos de la herramienta:

Modo de navegación

Es el modo inicial de Vim . En éste, las teclas del teclado actúan básicamente para la navegación y edición de bloques de texto. Generalmente los comandos son letras únicas. Si viene precedido por un número, el comando se repetirá de acuerdo con el valor de este número. Algunas teclas comunes utilizadas en el modo de navegación son:

```
0, $            - Inicio y final de línea
1G, G           - Inicio y final de documento.
(  )            - Inicio y final de sentencia.
{  }            - Inicio y final de párrafo.
w, W            - Saltar palabra y saltar palabra contando con la puntuación.
h, j, k ,l      - Izquierda, abajo, arriba, derecha.
/, ?            - Busca hacia adelante y hacia atrás
i               - Entra en el modo de inserción en la posición actual del cursor
a, A            - Entra en el modo de inserción después del cursor o al final de la línea.
o, O            - Agrega línea y entra en el modo de inserción después o antes del cursor.
s, S            - Borra ítem o línea y entra en el modo de inserción.
c               - Modifica un ítem por medio de inserción de texto.
r               - Sustituye un único carácter.
x               - Borra un único carácter.
y, yy           - Copia un ítem o toda la línea.
p, P            - Pega el contenido copiado después o antes del cursor
u               - Deshacer
ZZ              - Cierra y guarda si fuera necesario.
ZQ              - Cierra y no guarda.
```

Modo de inserción

La manera más común de entrar en el modo de inserción es por medio de la tecla [i] o [a] , ya sea en mayúscula o minúscula. Es el modo más intuitivo usado para escribir texto en un documento. La tecla [ESC] sale del modo de inserción y vuelve al modo de navegación.

Modo de comando

Accesible al pulsar la tecla [:] en el modo de navegación. Se utiliza para realizar búsquedas, guardar, salir, ejecutar comandos en el shell, modificar configuraciones de VI , etc.

Para volver al modo de navegación se utiliza la instrucción visual o simplemente se pulsa [Enter] con la línea vacía. A continuación se muestran los comandos de VI :

```
:!              Permite ejecutar un comando del shell.
:quit o q:      Cierra.
:quit! o :q!    Cierra sin guardar.
:wq             Guarda y cierra.
:exit o :x o :e Cierra y guarda si fuera necesario.
:visual         Vuelve al modo comando.
```

Gestión de ficheros

Las herramientas de gestión de ficheros en Linux nos permiten realizar las tareas cotidianas de eliminación, copia, movimiento y renombramiento de ficheros y directorios. Vamos a explicar a continuación cuales son estas herramientas y sus opciones de uso, tanto en el ámbito local como en el ámbito de red:

Ambito local: Las herramientas que se utilizan son cp, mv y rm:

- **cp:** Se utiliza para copiar ficheros y directorios. Por ejemplo, para copiar el archivo de texto *prueba.txt* al directorio Documentos haciendo uso de rutas absolutas:

```
$ cp /home/usuario/prueba.txt /home/usuario/Documentos/
```

Sus opciones principales son:

-i: Modo interactivo, pregunta antes de sobrescribir un archivo.

-p: Copia también los atributos del archivo original.

-r: Copia recursivamente el contenido del directorio de origen.

Es importante saber que al copiar un directorio recursivamente el uso de la barra [/] al final del directorio de origen determinará que únicamente se copie el contenido de dicho directorio al destino. En caso que no se use la barra se copiará el propio directorio de origen y su contenido al destino.

- **mv:** Se utiliza para mover archivos de una ubicación a otra. Se puede utilizar con la opción **-i** que pedirá confirmación antes de sobrescribir un archivo que ya esté en el destino. Si se utiliza el mismo archivo que en el anterior ejemplo y se quiere mover desde el directorio */home/usuario/Documentos/* a donde se encontraba inicialmente:

```
$ mv -i /home/usuario/Documentos/prueba.txt /home/usuario/
mv: ¿sobreescribir «/home/usuario/prueba.txt»? (s/n)
```

- **rm.** Borra un archivo especificado.

-p: Borra un árbol completo de directorios vacíos.

-r. Borra directorios con contenido.

-f: Fuerza la eliminación del directorio especificado.

```
$ rm -rf /home/usuario/Documentos
```

Fuerza el borrado del directorio Documentos con todo su contenido.

Ambito remoto: Las herramientas de copia de ficheros en remoto mas utilizadas son scp y rsync. En este curso solo veremos scp, que se basa en el comando cp a través de ssh. La sintaxis es la siguiente:

```
scp [opciones] origen destino
```

Tanto el origen como el destino puede ser una dirección remota. Para acceder a la ubicación remota, el formato es usuario@host:ruta. Las opciones mas utilizadas del comando son las siguientes:

- -i identity_file: Utiliza una llave privada de certificado concreta para realizar la comunicación con el host remoto.
- -P puerto: Especifica el puerto de conexión con el host remoto.
- -p: Preserva los tiempos de modificación, de acceso y permisos de los ficheros que se copian.
- -r: Copia de forma recursiva (obligatorio para copiar directorios)

Entrada, salidas, redirecciones y tuberías

Introducción

Los flujos, la redirección y las tuberias son algunas de las herramientas más potentes de la línea de comandos. Linux trata la entrada y salida de comandos como un flujo (datos que podemos manipular). Normalmente la entrada es el teclado y la salida la pantalla, pero se pueden redirigir estos flujos de entrada y salida hacia otros comandos o archivos. La tuberia [|] se utiliza para redirigir la salida de un comando hacia otro. Esto supone una magnífica herramienta de conexión entre comandos, lo cual permite realizar tareas complejas combinando varias herramientas sencillas.

Para empezar a entender el redireccionamiento y tuberías hay que comprender primero los diferentes tipos de flujos de entrada y salida. Todo comando en Linux posee tres canales básicos por donde fluye la información y estos canales son:

- **Entrada Estándar** , en inglés standard input (**stdin**) es el mecanismo por el cual un usuario le indica a los programas la información que estos deben procesar. Esta información suele introducirse por el teclado. Ejemplos:

```
$ cat
```

```
$ wc
```

Al ejecutar estos ejemplos sin argumentos el comando espera a que el usuario inserte información para posteriormente procesarla, dicha información de entrada debe ser proporcionada por la entrada estándar, en este caso por el teclado.

- **Salida Estándar** , en inglés standard output (**stdout**) es el método por el cual el programa puede comunicarse con el usuario. Por defecto la salida estándar es la pantalla donde se ejecutan dichos comandos. Ejemplos:

```
$ ls -l /etc/hosts /etc/passwd
-rw-r--r-- 1 root root  251 feb 23 15:52 /etc/hosts
-rw-r--r-- 1 root root 1635 feb 27 19:05 /etc/passwd
```

```
$ head -10 /etc/shells
# /etc/shells: valid login shells
/bin/csh
/bin/sh
/usr/bin/es
/usr/bin/ksh
/bin/ksh
/usr/bin/rc
/usr/bin/tcsh
/bin/tcsh
/usr/bin/esh
```

Al terminar de procesar la información dada como argumentos el comando mostrará un mensaje en la pantalla donde se ha ejecutado. Dicha pantalla o ventana será la salida estándar del comando ejecutado.

- **Error Estándar** , en inglés standard error output (**stderr**) es utilizado para mostrar mensajes de error que surjan durante el transcurso de su ejecución. Al igual que **stdout, stderr** será la pantalla donde se procesaron las instrucciones. Ejemplos:

```
$ rm esteArchivoNoExiste
rm: no se puede borrar «esteArchivoNoExiste»: No existe el fichero o el directori
```

```
$ rmdir /carpetaInexistente
rmdir: fallo al borrar «/carpetaInexistente»: No existe el fichero o el directorio
```

De igual forma **stderr** devolverá mensajes que indicarán los errores de los comandos ejecutados.

El flujo de datos para redireccionamiento y tuberías en línea de comandos se inicia de izquierda a derecha.

Redireccionamiento

Para redireccionar la salida estándar de un comando a un archivo se utiliza el símbolo > después del cual debe indicarse el nombre del archivo que se creará:

```
$ cat /etc/passwd > copia_passwd
```

Si el archivo existe previamente, se sobrescribirá. Para agregar los valores sin borrar el contenido existente, se utiliza >> .

Para redireccionar el contenido de un archivo a la entrada estándar de un comando se utiliza < . En este caso, el flujo de datos va de derecha a izquierda. Este tipo de redirección se suele utilizar con comandos como **tr** que no lee archivos directamente.

Normalmente se suele redireccionar la salida estandar pero si se quiere redireccionar la salida de errores podemos utilizar **2>** . Para redireccionar ambos simultáneamente, se usa **&>** .

```
$ ls x* z*
ls: z*: No such file or directory
xaa xab
```

Muestra un error porque no existe ningún archivo que empiece por z y muestra los archivos que empiezan por x

```
$ ls x* z* >stdout.txt 2>stderr.txt
```

Se ejecuta el mismo comando anterior pero redirigiendo la salida de errores al fichero stderr.txt y la salida estándar al fichero *stdout.txt*

```
$ ls w* y*
ls: w*: No such file or directory
yaa yab
```

Se ejecuta **ls** de nuevo en el directorio y muestra un error porque no existe ningún archivo que empiece por w y muestra los archivos que empiezan por y.

```
$ ls w* y* >>stdout.txt 2>>stderr.txt
```

Se ejecuta el mismo comando anterior pero redirigiendo la salida de errores al fichero *stderr.txt* y la salida estándar al fichero *stdout.txt*. Como estos ficheros contenían información se ha utilizado el redireccionador >> para agregar el contenido y no sobrescribirlo.

```
$ cat stdout.txt
xaa
xab
yaa
yab
```

Se muestra el contenido del fichero al que hemos redireccionado la salida estándar. Como se observa contiene la salida estándar de las dos ejecuciones del comando **ls**.

```
$ cat stderr.txt
ls: z*: No such file or directory
ls: w*: No such file or directory
```

Se muestra el contenido del fichero al que ha redireccionado la salida de errores. Como se observa contiene la salida de errores de las dos ejecuciones del comando **ls**.

> Crea un fichero nuevo con el contenido de la salida estándar. Si el archivo existe se sobrescribirá.

>> Añade la salida estándar al contenido de un archivo existente. Si no se especifica el archivo este se crea.

2> Crea un fichero nuevo con el contenido de la salida de errores. Si el archivo existe se sobrescribirá.

2>> Añade la salida de errores al contenido de un archivo existente. Si no se especifica el archivo este se crea.

&> Crea un fichero nuevo con el contenido de la salida estándar y la salida de errores. Si el archivo existe se sobrescribirá.

< Envía el contenido del fichero especificado para su uso en la entrada estándar.

<< Añade más texto a la entrada estandar.

<> Usa un fichero tanto para la entrada estándar como para la salida estándar.

Tubería (pipe)

Es posible enviar la salida de un comando a la entrada de otro comando, utilizando el carácter de tubería [|]. Por ejemplo:

```
$ cat /home/usuario/prueba.txt | grep -i linea
```

Filtra las líneas que contengan la palabra línea en el archivo *prueba.txt*.

También es posible redireccionar la salida simultáneamente tanto para un archivo como para *stdout*, por medio del comando **tee** . Para ello se redirige la salida del comando al comando **tee** suministrando a éste un nombre de archivo para almacenar la salida:

```
$ cat /home/usuario/prueba.txt | tee salida
```

El contenido de */home/usuario/prueba.txt* se mostrará en la pantalla y se copiará en el archivo salida.

Sustitución de Comandos

También es posible utilizar la salida de un comando como argumento para otro, utilizando comillas ejecutivas [`] (en el teclado se ubica en la tecla de apertura de corchetes):

```
$ ls -ld `echo $PATH|tr ':' ' '`
drwxr-xr-x 2 root root   4096 feb 24 13:50 /bin
drwxr-xr-x 2 root root  69632 mar  1 21:39 /usr/bin
drwxr-xr-x 2 root root   4096 feb 24 13:46 /usr/games
drwxrwsr-x 2 root staff  4096 mar  1 21:04 /usr/local/bin
drwxrwsr-x 2 root staff  4096 ago  4  2010 /usr/local/games
```

En este ejemplo se formatea el contenido de la variable PATH con el comando **tr** , para listar con **ls** los directorios contenidos en dicha variable.

El resultado será el mismo con:

```
$ ls -ld $(echo $PATH|tr ':' ' ')
drwxr-xr-x 2 root root   4096 feb 24 13:50 /bin
drwxr-xr-x 2 root root  69632 mar  1 21:39 /usr/bin
drwxr-xr-x 2 root root   4096 feb 24 13:46 /usr/games
drwxrwsr-x 2 root staff  4096 mar  1 21:04 /usr/local/bin
drwxrwsr-x 2 root staff  4096 ago  4  2010 /usr/local/game
```

Parecido a la sustitución de comandos el comando **xargs** desempeña la función de intermediario, pasando los datos que recibe por **stdin** como argumento para un segundo comando.

```
$ ls /home/usuario/Descargas/ | xargs rm -rf
```

Ejecuta "rm -rf" para cada archivo del directorio */home/usuario/Descargas*.

Visualización de ficheros II: Uso de expresiones regulares

Como ya se ha visto, la potencia de las redirecciones y tuberias nos permiten personalizar muchisimo la busqueda de contenido en fichero y su posterior tratamiento. Un aspecto fundamental en el uso de estos filtros son las expresiones regulares, que nos permiten establecer carácteres comodin para realizar filtros en las búsquedas de texto.

Las herramientas que suelen utilizarse para el uso de expresiones regulares son grep y sus derivadas.

Comando grep

Muchos programas soportan el uso de estos elementos y el comando **grep** es el más común para realizar búsquedas en textos mediante patrones. Algunos caracteres tienen un significado especial en expresiones regulares como se muestra a continuación:

^ Inicio de línea.

$ Final de línea.

. (punto) Cualquier carácter

***** Cualquier secuencia de cero o más caracteres.

[ ] Cualquier carácter que esté presente en los corchetes.

Un uso común del comando **grep** es mostrar el contenido de archivos de configuración ignorando únicamente las líneas que corresponden a los comentarios, es decir, las líneas que se inician con el carácter almohadilla [#]. Para hacer esto, es necesario añadir el parámetro **-v** que invierte el patrón de busqueda, seleccionando las líneas que no coincidan con el patrón.

Por ejemplo, para hacerlo con el archivo */etc/default/grub* :

```
$ grep -v '^#' /etc/default/grub
```

En el siguiente ejemplo, el uso de los corchetes muestra las líneas de */etc/default/grub* que contienen los términos *hda o hdb*.

```
$ grep 'hd[ab]' /etc/default/grub
```

-c: Cuenta las líneas en las cuales esta el patrón.

-i: Ignora la diferencia entre mayúsculas o minúsculas.

-f: Usa la expresión regular incluida en el archivo indicado por esta opción.

-n: Busca solamente en el número de línea indicada por esta opción.

-v: Realiza la acción inversa, es decir, muestra todas las líneas excepto la que corresponde al patrón.

Comandos egrep y fgrep

Los comandos **egrep y fgrep** complementan las funciones del comando **grep.**

El comando **egrep** es equivalente al comando **grep -E** que incorpora otras funcionalidades además de las expresiones regulares estándar. Por ejemplo, con **egrep** se puede utilizar el operador pipe "|" (tubería), que actúa como el operador O lógico:

```
$ egrep 'invención|invenciones'
```

Se devolverán todas las ocurrencias del término invención o invenciones.

El comando **fgrep** actúa de la misma forma que **grep -F** ya que deja de interpretar expresiones regulares. Es especialmente útil en los casos más sencillos donde lo que se desea es solamente encontrar la ocurrencia de algún término simple:

```
$ fgrep 'andalucia'
```

De este modo se ignora toda la operación de expresión regular lo que determinará que el proceso de búsqueda sea mucho más rápido. Incluso si se utilizan caracteres especiales, como dolar [$] o punto [.] se interpretarán literalmente y no por lo que representan en una expresión regular.

Expresiones regulares

Bueno en esta guía os quería explicar de forma mas clara y con ejemplos explicados que es esto de las expresiones regulares. Para comenzar lo primero que hay que aclarar es la diferencia entre **expresiones regulares** y **patrones de ficheros**, ya que se pueden confundir y a nivel funcional son diferentes. Las **expresiones regulares** son las utilizadas para buscar un texto dentro de un fichero, mientras que los **patrones de ficheros** son normalmente los argumentos pasados a los comandos como **rm, cp, mv,** etc. para hacer referencia a varios ficheros en el disco duro. Por tanto, podemos concluir en que tenemos un grupo de comandos que serán los que utilicen expresiones regulares y que están orientados al tratamiento de texto: **grep, egrep, ed, sed, awk, vi,** etc...

Existen dos tipos de expresiones regulares: las **básicas** y las **extendidas**. Como se puede deducir de su nombre las extendidas otorgan muchas mas opciones a la hora de tratar con los patrones a buscar. Antes de comenzar con la guía y ver ejemplos vamos a ver todas las expresiones regulares así como las reglas de uso de estas mismas:

Reglas

- Cuando hablamos de *expresiones*, estas pueden ser un carácter, un conjunto de caracteres o un metacáracter.
- Para utilizar las expresiones regulares es necesario ponerlas entre comillas simples ' '
- Para usar las expresiones regulares extendidas, es necesario otorgar al comando la funcionalidad. Por ejemplo, para el comando **grep** es necesario utilizar la opción **-E** o utilizar directamente el comando **egrep**, y para el comando **sed** es necesario añadirle la opción **-r**.
- Los caracteres son tratados de forma literal, es decir, que concuerdan consigo mismo. Por ejemplo x concuerda con x, abc concuerda con abc, etc...
- La excepción a la regla anterior esta en los metacaracteres:

```
. [ ] ^ $ * ( ) \
```

Para utilizar estos metacaracteres como literales, es necesario *escaparlos*, esto se consigue antecediendole el metacaracter de barra invertida ****

A continuación seguiremos viendo algunas reglas más en conjunto de las diferentes expresiones regulares. Estas expresiones se pueden subdividir en varias categorías, que son las siguientes:

Expresiones regulares básicas de un solo carácter

Dentro de esta categoría tenemos las siguientes expresiones:

Expresión regular	Concuerda con....
.	Cualquier carácter
[ ]	Cualquiera de los caracteres que estén dentro de los corchetes
[^]	Cualquiera de los caracteres que NO estén dentro de los corchetes
^	El principio de línea
$	El final de la línea
*	Cero o mas ocurrencias de la expresión anterior (izquierda)
\(\)	Permite agrupar varias expresiones regulares
\	*Escapa* un metacarácter

Estas expresiones tienen las siguientes reglas especiales:

Dentro de los corchetes **[ ]** los metacaracteres pierden su función especial y se tratan como literales.

- Hay varias excepciones a la regla anterior. Por ejemplo el carácter del “sombrerito” si se pone al inicio del corchete gana una funcionalidad diferente, como hemos visto en la tabla anterior.

Expresiones regulares básicas de repetición

Podemos repetir una expresión regular tantas veces como queramos con la secuencia **\{ \}.** Esta repetición se puede realizar de las siguientes formas:

Constructor	Proposito
\{n\}	Concuerda exactamente con *n* ocurrencias de la expresión anterior
\{n,\}	Concuerda con la menos *n* ocurrencias de la expresión anterior
\{n, m\}	Concuerda con entre *n* y *m* ocurrencias de la expresión anterior

Pongamos un ejemplo rápido de este tipo de expresiones. Por ejemplo para buscar números de tres cifras podemos utilizar **[0-9]\{3\}.** Esto significa que repetirá 3 veces la búsqueda de números del cero al nueve, podría igualarse a **[0-9] [0-9] [0-9]**

Expresiones regulares extendidas

Como hemos visto antes, para poder utilizar estas expresiones es necesario darle al comando que se ejecuta el soporte para poder usarlas. Antes de comenzar a verlas, es necesario ver reglas de conversión:

- El uso de las barras invertidas en los corchetes y los paréntesis no sirve en las expresiones regulares extendidas. Esto quiere decir que lo que hemos visto antes de **\(\)** y **\{ \}** ahora se debe poner de este modo: **( )** y **{ }**. Si queremos que sean caracteres literales es necesario *escaparlos*.

Expresiones regulares extendidas de un solo carácter

En este grupo se agregan las siguientes expresiones:

Expresión regular	Concuerda con...
+	Una o mas ocurrencias de la expresión anterior (izquierda)
?	Cero o una ocurrencia de la expresión anterior (izquierda)

Expresiones regulares extendidas de alternancia

Con el metacarácter tubería **|** podemos alternar entre dos expresiones regulares. Por ejemplo si queremos buscar una palabra que sea abc o bbc podemos indicarlo del siguiente modo **(a|b)bc**

Expresiones regulares extendidas de etiquetado

Un etiquetado consiste en referenciar una expresión regular en otro lugar del patrón de búsqueda. Para etiquetar una expresión regular debemos hacer uso de los paréntesis **( )** y luego para referenciar esa expresión debemos indicarlo con ***n*** siendo *n* el número de la etiqueta en orden ascendente comenzando por uno (pueden existir varias etiquetas dentro del patrón).

Por ejemplo la expresión **(.)e\1e** concuerda con *pepe* y con *nene*

Otros metacaracteres utilizados en las expresiones regulares extendidas

Existen otros metacaracteres para las expresiones regulares extendidas, entre los cuales solamente vamos a ver uno muy utilizado que hace referencia a la delimitación entre el principio y el fin de una palabra. Para referenciar el principio se debe usar **\<** y para referenciar el final **\>**

Casos prácticos

Bueno, una vez visto todo esto, vamos al caso práctico! Vamos a utilizar este texto para nuestros ejemplos:

```
Primeros pasos
Si desea empezar a usar Debian, puede obtener fácilmente una copia y seguir la Guía de Instalación para
 instalarla.
Si está actualizando a la última versión estable desde una versión anterior, por favor, lea las Notas d
e Publicación antes de hacerlo.
Para obtener ayuda sobre el uso o instalación de Debian, consulte nuestras páginas de documentación y s
oporte.
Los usuarios que hablen en idiomas que no sean el inglés deberían echar un vistazo a nuestra sección in
ternacional.
Los usuarios que usen sistemas distintos de Intel x86 deberían revisar la sección de adaptaciones a otr
as arquitecturas.
RSS
Últimas noticias.
[15 de feb de 2014] Updated Debian 6.0: 6.0.9 released
[8 de feb de 2014] Updated Debian 7: 7.4 released
[14 de dic de 2013] Updated Debian 7: 7.3 released
[20 de oct de 2013] Updated Debian 6.0: 6.0.8 released
[12 de oct de 2013] Updated Debian 7: 7.2 released
[28 de sep de 2013] Debian Edu / Skolelinux Wheezy - una solución completa basada en Linux para la escu
ela
URLs
http://ftp.debian.org/debian/dists/wheezy/ChangeLog
http://ftp.debian.org/debian/dists/stable/
http://ftp.debian.org/debian/dists/proposed-updates
https://www.debian.org/releases/stable/
https://security.debian.org/
```

Lo primero que vamos a hacer para esta guía es configurar el comando **grep** con la opción de color. Esto nos va a ayudar mucho a saber realmente lo que estamos filtrando con el patrón que hemos puesto. Para activar esta opción se puede hacer de dos formas: o bien con la variable de entorno**GREP_OPTIONS** contenga el valor **--color** o bien creando un alias al comando **alias grep='grep –color=always'**.

```
$ export GREP_OPTIONS='--color'
```

Comenzamos con el ejemplo mas básico que es buscar una palabra concreta. En este caso vamos a buscar la palabra “Debian”:

```
$ grep 'Debian' texto
Si desea empezar a usar Debian, puede obtener fácilmente una copia y seguir la Guía de Instalación para
 instalarla.
Para obtener ayuda sobre el uso o instalación de Debian, consulte nuestras páginas de documentación y s
oporte.
[15 de feb de 2014] Updated Debian 6.0: 6.0.9 released
[8 de feb de 2014] Updated Debian 7: 7.4 released
[14 de dic de 2013] Updated Debian 7: 7.3 released
[20 de oct de 2013] Updated Debian 6.0: 6.0.8 released
[12 de oct de 2013] Updated Debian 7: 7.2 released
[28 de sep de 2013] Debian Edu / Skolelinux Wheezy - una solución completa basada en Linux para la escu
ela
```

Como podemos apreciar, no nos ha cogido las palabras “debian” que existen en el texto. Esto se debe a que, como ya sabemos, Linux discrimina entre mayúsculas y minúsculas. Para ignorar esta discriminación, debemos ponerle al comando **grep** la opción **-i**

Esto mismo podriamos abordarlo del siguiente modo, aunque no es el mas correcto. Usando el punto podemos sustituir cualquier carácter dentro del patrón de búsqueda, por tanto:

```
$ grep '.ebian' texto
Si desea empezar a usar Debian, puede obtener fácilmente una copia y seguir la Guía de Instalación para
 instalarla.
Para obtener ayuda sobre el uso o instalación de Debian, consulte nuestras páginas de documentación y s
oporte.
[15 de feb de 2014] Updated Debian 6.0: 6.0.9 released
[8 de feb de 2014] Updated Debian 7: 7.4 released
[14 de dic de 2013] Updated Debian 7: 7.3 released
[20 de oct de 2013] Updated Debian 6.0: 6.0.8 released
[12 de oct de 2013] Updated Debian 7: 7.2 released
[28 de sep de 2013] Debian Edu / Skolelinux Wheezy - una solución completa basada en Linux para la escu
ela
http://ftp.debian.org/debian/dists/wheezy/ChangeLog
http://ftp.debian.org/debian/dists/stable/
http://ftp.debian.org/debian/dists/proposed-updates
https://www.debian.org/releases/stable/
https://security.debian.org/
```

De este modo buscaría cualquier palabra que contenga cualquier carácter seguido de “ebian”.

Ahora vamos a complicarlo mas. Vamos a realizar una búsqueda de todas las líneas que contengan la fecha de release pertenecientes a los dias 10 hasta el 19 de cualquier mes. Analizando el texto debemos notar que la expresión debe coincidir los números al inicio de línea. Viendo la estructura del fichero lo podemos acometer del siguiente modo:

```
$ grep '^\[1[0-9]' texto
[15 de feb de 2014] Updated Debian 6.0: 6.0.9 released
[14 de dic de 2013] Updated Debian 7: 7.3 released
[12 de oct de 2013] Updated Debian 7: 7.2 released
```

El acento circunflejo ^ lo utilizamos para indicar que la expresión se debe buscar al inicio de linea, despues *escapamos* el corchete para hacerlo literal, a continuación ponemos el uno y luego un rango que comprenda entre el 0 y el 9.

Lo siguiente que vamos a ver es el uso de una expresión regular extendida, que va a ser la interrogación. Por ejemplo, vamos a seleccionar las líneas que contengan tanto *http* como *https*. Para ello construimos el patrón del siguiente modo:

```
$ grep -E 'https?' texto
http://ftp.debian.org/debian/dists/wheezy/ChangeLog
http://ftp.debian.org/debian/dists/stable/
http://ftp.debian.org/debian/dists/proposed-updates
https://www.debian.org/releases/stable/
https://security.debian.org/
```

La interrogación permite que el carácter que esta a su izquierda sea opcional, por tanto nos va a buscar tanto cadenas http como https.

Para comprender como funciona el metacarácter **+** vamos a ver una sucesión de comandos para poder comprenderlo:

```
$ cat texto | cut -d " " -f 1-5 |grep -E '[0-1]+'
[15 de feb de 2014]
[8 de feb de 2014]
[14 de dic de 2013]
[20 de oct de 2013]
[12 de oct de 2013]
[28 de sep de 2013]
```

```
$ cat texto | cut -d " " -f 1-5 |grep -E '[0-2]+'
[15 de feb de 2014]
[8 de feb de 2014]
[14 de dic de 2013]
[20 de oct de 2013]
[12 de oct de 2013]
[28 de sep de 2013]
```

```
$ cat texto | cut -d " " -f 1-5 |grep -E '[0-3]+'
[15 de feb de 2014]
[8 de feb de 2014]
[14 de dic de 2013]
[20 de oct de 2013]
[12 de oct de 2013]
[28 de sep de 2013]
```

```
$ cat texto | cut -d " " -f 1-5 |grep -E '[0-4]+'
[15 de feb de 2014]
[8 de feb de 2014]
[14 de dic de 2013]
[20 de oct de 2013]
[12 de oct de 2013]
[28 de sep de 2013]
```

El siguiente ejemplo vamos a utilizar el concepto de repeticiones. Vamos a complicarlo un poco buscando palabras que solo contengan 4 letras. Hay que tener en cuenta que debemos delimitar el patrón con el inicio y el fin de una palabra, ya que si no buscaría dentro de las palabras secuencias de 4 letras, y eso no es lo que queremos. Por tanto la construcción se haría del siguiente modo:

```
$ grep -E '\<[a-zA-Z]{4}\>' texto
Si desea empezar a usar Debian, puede obtener fácilmente una copia y seguir la Guía de Instalación para
 instalarla.
Si está actualizando a la última versión estable desde una versión anterior, por favor, lea las Notas d
e Publicación antes de hacerlo.
Para obtener ayuda sobre el uso o instalación de Debian, consulte nuestras páginas de documentación y s
oporte.
Los usuarios que hablen en idiomas que no sean el inglés deberían echar un vistazo a nuestra sección in
ternacional.
Los usuarios que usen sistemas distintos de Intel x86 deberían revisar la sección de adaptaciones a otr
as arquitecturas.
[28 de sep de 2013] Debian Edu / Skolelinux Wheezy — una solución completa basada en Linux para la escu
ela
URLs
http://ftp.debian.org/debian/dists/wheezy/ChangeLog
http://ftp.debian.org/debian/dists/stable/
http://ftp.debian.org/debian/dists/proposed-updates
```

Como podemos apreciar, con los delimitadores **\<** y **\>** hacemos que solo encuentren un carácter comprendido entre la a y z tanto minúscula como mayúscula repetida la secuencia 4 veces.

Ahora algo mas complicado, vamos a seleccionar las líneas que tengan frases de dos palabras. Para ello lo primero a la hora de montar la expresión regular es saber que una palabra es una serie de letras consecutivas. Para afrontarlo como patrón de búsqueda podriamos indicar **[a-zA-z]+**

Lo siguiente que debemos entender que entre palabras hay un espacio, por tanto lo siguiente que deberiamos buscar sería cualquier cosa que no sea una letra, también hay que tener en cuenta que el final de la frase entendemos que se realiza con un punto, por tanto **[^a-zA-Z]**

Bien, ya tenemos una palabra y un espacio/punto. Ahora para saber una frase de dos palabras lo único que tendriamos que indicarle es que se repitiera 2 veces, pero claro, si lo ponemos de forma literal no va a comprender lo que queremos realizar. Por tanto es necesario agrupar las dos expresiones anteriores para realizar la repetición y delimitándolas para que entienda que es una frase, quedando una cosa asi:

```
$ grep -E '^([a-zA-Z]+[^a-zA-Z]){2}$' texto
Últimas noticias.
```

Si analizamos la expresión regular, el inicio de la frase lo expresamos con ^ y el final con $. Lo siguiente es entre paréntesis la unión de una letra con un espacio o un punto, y ese conjunto se repite dos veces.

Procesos y trabajos

En el sistema Linux, cuando una aplicación se ejecuta genera lo que llamamos un proceso. Este proceso tiene un identificador de proceso (PID) y, si la propia aplicacion ejecuta otra aplicación, generará otro proceso hijo de la aplicación inicial, que también tendrá su propio (PID).

Esta regla tiene la excepción de los programas internos de la shell, que como ya vimos en el apartado de la shell, utilizan el propio PID de la shell para su ejecución.

A continuación veremos las diferentes aplicaciones que se utilizan en Linux para monitorizar y gestionar procesos.

Monitorizar procesos

Pueden utilizarse diversos comandos para inspeccionar procesos y son especialmente útiles para localizar y finalizar procesos que no son necesarios o son sospechosos.

Una de las herramientas más importantes de administración de procesos es el comando **ps** , que muestra el estado de los procesos. Incluye algunas opciones muy útiles y ayuda a monitorizar lo que está ocurriendo en el sistema. Su sintaxis es bastante simple:

```
ps [options]
```

Algunas de las características más comunes son:

- Mostrar todos los procesos: Por defecto, **ps** muestra solo los procesos arrancados desde la propia terminal (por ejemplo xterm, login en modo texto o login remoto). Las opciones **-A** y **-e** muestran todos los procesos en el sistema, y **x** muestra todos los procesos de los que es propietario el usuario que ejecutó el comando. La opción **x** también incrementa la cantidad de información mostrada sobre cada proceso.
- Mostrar procesos de un usuario: Se pueden mostrar los procesos de un usuario concreto con la opción **-u** *user*, **U** *user*, y **--User** *user*. La variable *user* debe ser un nombre de usuario o el ID de un usuario.
- Mostrar información extra: Las opciones **-f** , **-l** , **j** , **l** **u** y **v** expanden la información sacada en pantalla. La mayoría de los formatos de salida incluyen una línea por proceso, pero **ps** puede mostrar tal cantidad de información que sea imposible mostrar en una sola línea la información de un proceso concreto.
- Mostrar jerarquía de procesos: Las opciones **-f** , **-H** y **--forest** agrupan procesos y muestran la jerarquía y relaciones entre ellos. Estas opciones pueden ser útiles si se desea saber cual es el padre de un proceso, o viceversa.

Ejemplo del comando **ps** :

```
$ ps -u usuario --forest
PID TTY TIME CMD
1461 ? 00:00:00 ck-launch-sessi
1512 ? 00:00:00 \_ ssh-agent
1587 ? 00:00:02 \_ x-session-manag
1600 ? 00:00:04 \_ xfwm4
1602 ? 00:00:04 \_ xfdesktop
1604 ? 00:00:21 \_ xfce4-panel
1610 ? 00:00:00 \_ xfce4-menu-plug
2189 ? 00:00:06 | \_ pidgin
1615 ? 00:00:00 \_ xfce4-mixer-plu
4631 ? 00:00:14 epdfview
2795 ? 00:00:09 xfce4-terminal
2796 ? 00:00:00 \_ gnome-pty-helpe
2797 pts/0 00:00:00 \_ bash
5989 pts/0 00:00:00 \_ ps
2354 ? 00:01:24 exe
2383 ? 00:03:34 \_ chrome
2388 ? 00:00:00 \_ chrome
2389 ? 00:00:00 \_ chrome-sandbox
2390 ? 00:00:00 \_ chrome
2393 ? 00:00:00 \_ nacl_helper_boo
2394 ? 00:00:00 \_ chrome
2422 ? 00:00:02 \_ chrome
2427 ? 00:03:43 \_ chrome
2950 ? 00:01:13 \_ chrome
2331 ? 00:00:00 gvfs-afc-volume
2329 ? 00:00:00 gvfs-gphoto2-vo
2325 ? 00:00:00 gvfs-gdu-volume
1643 ? 00:00:00 gnome-keyring-d
1635 ? 00:00:00 gconfd-2
1631 ? 00:00:00 polkit-gnome-au
1629 ? 00:00:00 update-notifier
1627 ? 00:00:01 nm-applet
1622 ? 00:00:02 Thunar
1612 ? 00:00:00 gvfsd
1609 ? 00:00:01 xfce4-settings-
1608 ? 00:00:25 gam_server
1606 ? 00:00:00 xfce4-power-man
1601 ? 00:00:00 xfsettingsd
1594 ? 00:00:00 xfconfd
1591 ? 00:00:00 dbus-daemon
1590 ? 00:00:00 dbus-launch
```

Aquí se muestran los procesos iniciados por el usuario "usuario" y la jerarquía y relación entre ellos.

Una variante del comando **ps** es **pstree** que muestra procesos activos en formato de árbol genealógico (procesos hijos vinculados a sus respectivos procesos padre).

Para monitorizar continuamente los procesos, mostrando información como utilización de memoria y CPU de cada uno de ellos, se usa el comando **top** . La tecla *[H]* proporciona ayuda sobre la utilización del programa. Puede utilizarse para modificar la prioridad de un proceso. Esta herramienta es un programa en modo texto que se puede arrancar desde una terminal, aunque también existen algunas variantes en modo gráfico. Como cualquier comando Linux, acepta diferentes opciones,

algunas de las más útiles son las siguientes:

-d delay *:* Especifica el intervalo entre actualizaciones que por defecto es 5 segundos. Esto indica el tiempo que tarda el programa en actualizar los datos que se están presentando por pantalla.

-p PID: Para monitorizar procesos específicos se puede usar esta opción.

-n iter: Se puede especificar que muestre cierto número de actualizaciones, y después finalice el programa.

k: Para matar un proceso, el comando preguntará por el PID y matará dicho proceso siempre que esté capacitado de hacerlo.

q: Salir del programa.

r: Se puede cambiar la prioridad de un proceso con esta opción. Habrá que insertar a continuación el PID del proceso seguido de un nuevo valor de prioridad. Un valor positivo decrementará su prioridad y un valor negativo la incrementará, asumiendo que la prioridad por defecto con que inicia un proceso es 0.

P: Ordena la salida por pantalla según el uso de la CPU que hace cada proceso.

M: Ordena la salida por pantalla según el uso de la memoria que hace cada proceso.

Una variante del programa top es **htop** , que a diferencia de top muestra una lista completa de procesos en ejecución en lugar de los procesos que están consumiendo recursos. Además usa colores y da un aspecto visual más amigable ofreciendo información sobre la CPU, la swap y el estado de la memoria.

Se puede obtener el PID de un programa concreto con el comando **pidof** , siempre que dicho programa esté en ejecución. Su sintaxis es como sigue:

```
pidof [options] programa
```

Como programa se debe especificar el nombre del programa cuyo PID deseemos consultar. El PID puede ser uno solo o varios, ya que un programa puede tener distintos procesos en ejecución relacionados entre sí (procesos padre e hijos).

Ejemplo de **pidof:**

```
$ pidof chrome
3061 3016 2645 1888 1855 1851 1849 1844
```

Otra aplicación que permite visualizar los procesos filtrando por cierto criterio es el programa **pgrep** . Funciona de un modo similar al comando **grep** pero esta variante permite filtrar procesos según un criterio dado. La sintaxis básica es la siguiente:

```
pgrep [-flvx] [-d delimitador] [.n|-o] [-P ppid] [-g pgrp,...] [-s sid,...] [-u euid,...] [-U uid,...] [
```

```
n|-o] [-P ppid] [-g pgrp,...] [-s sid,...] [-u euid,...] [-U uid,...] [-G gid,...] [-t term,...] [patrón]
```

Como se puede ver, existen muchos criterios de filtrado para **pgrep** , permitiendo un control total en la selección de criterio a la hora de mostrar procesos. Este comando es muy utilizado en scripts, donde el filtrado y selección de procesos para su gestión se vuelve fundamental para no cometer errores en la automatización de tareas. Un ejemplo sencillo de utilización de este comando es, por ejemplo, el de filtrar los procesos sshd del usuario root. Esto se realiza del siguiente modo:

```
# pgrep -u root sshd
4532
19485
```

Finalizar procesos

Un programa puede llegar a quedarse totalmente sin respuesta o bien puede ocurrir que se desee finalizar un programa que no debería estar arrancado. En estos casos el comando **kill** es la herramienta a usar.

Este comando envía una señal (es un método que Linux usa para comunicarse con los procesos) a un proceso. La señal estándar cuando no se informa ninguna señal es SIGTERM, con el valor numérico 15 que solicita al programa en cuestión su finalización. El proceso no necesariamente obedece a la señal a menos que la señal sea SIGKILL. En algunos casos la señal SIGHUP puede interpretarse como orden para que el proceso lea nuevamente su(s) archivo(s) de configuración.

Por ejemplo, para enviar la señal SIGTERM al proceso número 4902:

```
# kill -SIGTERM 4902
```

Algunas de las señales más utilizadas se listan a continuación:

- **SIGHUP:** Termina o reinicia el proceso. Valor numérico 1.
- **SIGINT:** Interrumpe el proceso, igual a [Ctrl]+[C]. Valor numérico 2.
- **SIGQUIT:** Cierra el proceso. Valor numérico 3.
- **SIGKILL:** Fuerza la finalización del proceso. Valor numérico 9.
- **SIGTERM:** Solicita al proceso para finalizar. Valor numérico 15.

Existen otros comandos muy relacionados con **kill** para la eliminación de procesos. El comando **pkill** funciona del mismo modo que el comando **pgrep** visto anteriormente, salvo que la única diferencia es que el primero le envía la señal por defecto **SIGTERM** (se puede modificar con la opción *-singal*) y el segundo solo muestra el PID del proceso por pantalla.

Otro de los comandos utilizados para eliminar procesos es **killall** . Este comando envía una señal a todos los procesos en ejecución que se relacionen con el patrón dado. Por defecto, al igual que **pkill** , envía la señal **SIGTERM** , siendo modificable por sus parámetros. La diferencia con **pkill** reside en su posibilidad de seleccionar el criterio de eliminación de procesos mediante expresiones regulares, eliminación de procesos de forma interactiva y verificación de que todos los procesos se han eliminado con éxito. Su sintaxis básica es la siguiente:

```
killall [-Z patron] [-e] [-g] [-i] [-q] [-r] [-s señal] [-u usuario] [-v] [-w] [--ignore-case] [-V] pat
ron
```

Las opciones mas destacadas de este comando son las siguientes:

- **--ignore-case:** Ignora la sensibilidad de mayúsculas a la hora de buscar el *patrón* dado.
- **-i, --interactive:** Pregunta de forma interactiva antes de eliminar los procesos.
- **-r, --regexp:** Permite el uso de expresiones regulares en el *patron* dado.
- **-u, --user:** Elimina solo los procesos del *usuario* dado.
- **-w, --wait:** Espera a que todos los procesos se eliminen antes de finalizar el comando. Esto hace que el comando verifique cada segundo y si algún proceso todavía existe espera hasta que no quede ninguno.

Tareas en Primer y Segundo Plano (jobs), tareas desatendidas

Una de las tareas más básicas de la administración de procesos es controlar si un proceso se ejecuta en primer o segundo plano. Cuando se inicia un programa, éste normalmente toma el control de la terminal impidiendo realizar cualquier otra tarea. Para liberar la terminal basta con pulsar *control + Z* que pausa el programa.

El inconveniente de este procedimiento es que el programa no sigue ejecutándose. Si se pausan procesos se pueden recuperar con **fg** y si hay más de uno debe añadirse el número de tarea que se quieren recuperar. El número de tarea asociado con un terminal se obtiene con el comando **jobs** .

```
$ jobs
[1]-  Stopped                 top
[2]+  Stopped                 htop
```

Para recuperar el proceso *htop* a primer plano, se ejecuta lo siguiente:

```
$ fg 2
```

Una variante de **fg** es **bg** que, a diferencia de **fg** que devuelve la tarea al primer plano, devuelve una tarea a un segundo plano.

El comando **fg** es útil para programas que no necesiten interacción con el usuario y puedan realizar su tarea manteniendo libre la shell. Si se inicia un programa GUI (gráfico) desde una shell, ésta quedará bloqueada por el programa. Para retomar el control de la shell, hay que pulsar *control + Z* y recuperar el trabajo con **bg** , lo que permite al programa ejecutarse en segundo plano y dejar la terminal libre.

Sintaxis **bg**:

```
bg [job_spec]
```

La alternativa a no tener que pausar y volver a ejecutar los programas con **bg** es añadir un caracter ampersand [&] al final de la línea del comando a ejecutar que permitirá iniciar un programa sin que tome el control de la terminal.

```
$ gedit &
[2] 23055
```

Los números que se muestran al ejecutar el comando en segundo plano corresponden al número de la tarea y al PID, respectivamente. El número de una tarea puede utilizarse como argumento del comando **kill** , siempre y cuando esté precedido del símbolo de tanto por ciento [%]:

```
$ kill %2
```

Programación de tareas con cron

¿Qué es cron?

Cron es el nombre del programa que permite a usuarios Linux/Unix ejecutar automáticamente comandos o scripts (grupos de comandos) a una hora o fecha específica. Es usado normalmente para comandos de tareas administrativas, como respaldos, pero puede ser usado para ejecutar cualquier cosa.

Cron es el demonio del sistema con el que se programan acciones en base al tiempo. Las realiza cada hora, día, semana, mes y con cierto intervalo de días, desde segundo plano, siempre y cuando el sistema se encuentre en modo multiusuario. Para esto, debe leer el archivo llamado crontab, que se localiza normalmente en /etc o algún otro directorio, dependiendo de la versión de Unix que se maneje.

Entendiendo el papel de cron

El programa cron es un demonio, por lo que funciona de forma continua, en busca de eventos que hacen que entre en acción. A diferencia de la mayoría de los demonios, que son los servidores de red, cron responde a eventos temporales. En concreto, se "despierta" una vez por minuto, examina los archivos de configuración en los directorios /var/spool/cron y /etc/cron.d y el archivo /etc/crontab y ejecuta los comandos especificados por estos archivos de configuración si el tiempo coincide con la hora indicada en estos archivos.

Se pueden agregar comandos o scripts como tareas de cron para automatizar algunos procesos. Esto es útil por ejemplo para automatizar la actualización de un sistema o establecer un robusto sistema de copias de seguridad.

Existen dos tipos de tareas de cron: tareas cron del sistema y tareas cron de usuario. Las tareas de cron del sistema se ejecutan como root y se convierten en tareas de mantenimiento del sistema. Por defecto, la mayoría de las distribuciones incluyen tareas cron del sistema que limpian los archivos antiguos del /tmp, realizan rotaciones de logs, y demás tareas de mantenimiento.

Normalmente los usuarios pueden crear tareas cron de usuario, las cuales ejecutarán algunos programas de usuario habituales. Se pueden crear tareas cron de usuario como root, las cuales pueden ser útiles si se necesita realizar alguna tarea en algún momento no cubierto por las tareas cron del sistema, que están programadas de forma mas rígida.

Una de las cuestiones críticas de las tareas del cron es que se ejecutan sin supervisión. Por lo tanto, no se debe incluir cualquier programa en una tarea programada si el programa requiere intervención del usuario. Por ejemplo, no se podría ejecutar un editor de texto en una tarea cron, pero es posible ejecutar un script que manipule automáticamente los archivos de texto.

Creando tareas cron del sistema

Crontab es un simple archivo de texto que guarda una lista de comandos a ejecutar en un tiempo especificado por el usuario. Crontab verificará la fecha y hora en que se debe ejecutar el script o el comando, los permisos de ejecución y lo realizará en segundo plano (background). Cada usuario puede tener su propio archivo crontab, de hecho el **/etc/crontab** se asume que es el archivo crontab del usuario root, cuando los usuarios normales (e incluso root) desean generar su propio archivo de crontab, entonces se utiliza el comando crontab.

Crontab es la manera mas sencilla de administrar tareas de cron en sistemas multiusuario, ya sea como simple usuario de sistema o usuario root.

El archivo /etc/crontrab controla las tareas cron del sistema. Este archivo normalmente comienza con varias líneas que establecen las variables de entorno como por ejemplo $PATH y $MAILTO.

Se ejecuta la edición del crontab con crontab -e, en algunas distribuciones (como ubuntu) existe la opción de elegir el editor de textos que se desea, aunque en la mayoría de las distribuciones se usa por defecto vi. El archivo **/etc/crontab** tiene la siguiente sintaxis:

```
m h dom mon dow user command
```

donde:

- m corresponde al minuto en que se va a ejecutar el script, el valor va de 0 a 59.
- h la hora exacta, se maneja el formato de 24 horas, los valores van de 0 a 23, siendo 0 las 12:00 de la medianoche.
- dom hace referencia al día del mes, por ejemplo se puede especificar 15 si se quiere ejecutar cada dia 15.
- mon hace referencia al mes del año en el que se desea ejecutar, puede ser un valor numérico (1-12) o los tres primeros caracteres en inglés (Jan-Dec).
- dow significa el día de la semana, puede ser numérico (0 a 7, donde 0 y 7 son domingo) o las 3 primeras letras del día en inglés: mon, tue, wed, thu, fri, sat, sun.
- user define el usuario que va a ejecutar el comando, puede ser root, u otro usuario diferente siempre y cuando tenga permisos de ejecución del script.
- command se refiere al comando o a la ruta absoluta del script a ejecutar, ejemplo:/home/usuario/scripts/actualizar.sh, si acaso llama a un script este debe ser ejecutable.

Ejemplos:

```
15 10 * * * usuario /home/usuario/scripts/actualizar.sh
```

Ejecutará el script actualizar.sh a las 10:15 a.m. todos los días.

```
15 22 * * * usuario /home/usuario/scripts/actualizar.sh
```

Ejecutará el script actualizar.sh a las 22:15 p.m. todos los días.

```
00 10 * * 0 root apt-get -y upgrade
```

Ejecutará una actualización todos los domingos a las 10:00 a.m.

```
45 10 * * sun root apt-get -y upgrade
```

Usuario root ejecutará una actualización todos los domingos (sunday) a las 10:45 de la mañana.

```
30 7 15 12 * usuario /home/usuario/scripts/actualizar.sh
```

El día 15 de diciembre a las 7:30 el usuario ejecutará el script.

También se puede establecer un recordatorio cada minuto de cada hora y cada día, pero esto no se recomienda por ser demasiado tedioso.

Igualmente se pueden establecer rangos especiales:

```
30 16 * * 1,2,3,4,5
```

A las 16:30 de la tarde todos los días de lunes a viernes.

```
15 12 1,15,28 * *
```

A las 12:15 del día todos los días 1, 15 y 28 de cada mes.

Las entradas del archivo /etc/crontab generalmente utilizan run-parts, cronloop, o una utilidad similar que ejecuta todos los scripts ejecutables dentro de un directorio. La mayoría de las distribuciones incluyen tareas cron del sistema mensuales, diarias, semanales, y por cada hora, cada una de ellas correspondiente a los scripts en un directorio llamado /etc/cron.interval, en el que interval es la palabra asociada a la frecuencia de funcionamiento. Otras distribuciones colocan estos scripts en directorios /etc/cron.d/interval.

Creando tareas cron de usuario

En un sistema Linux, cada usuario tendrá su propio crontab, incluido root. Por supuesto, en el crontab de cada usuario sólo se permitirá ejecutar tareas para las que ese usuario tenga permisos (por ejemplo, un usuario que no sea root no podrá apagar el sistema). Para crear tareas cron de usuario se utilizan las crontab, pero no se debe confundir con la configuración del archivo /etc/crontab.

La sintaxis del comando crontab es la siguiente:

```
crontab [-u usuario] [-l | -e | -r] [archivo]
```

Si se utiliza sin el parámetro -u, crontab modifica la tarea cron asociada con el usuario actual. La utilidad crontab puede llegar a ser confusa por el uso del comando su para cambiar la identidad del usuario, por lo que es recomendable usarla con el parámetro -u.

Si se desea trabajar directamente con una tarea cron, se debe usar la opción -l, -r o -e.

- La opción -l causa que crontab muestre la tarea de cron actual.
- La opción -r elimina la tarea actual de cron.
- La opción -e abre un editor (el editor vi por defecto) para que se pueda editar la tarea cron actual.

Como alternativa, se puede crear un archivo de configuración de la tarea cron y pasar el nombre del archivo a crontab utilizando el parámetro file.

Si se crea la tarea programada y se envía a través del parámetro file o editándolo con -e, el formato del archivo cron es similar a lo descrito anteriormente. Se pueden establecer variables de entorno usando la forma VARIABLE=valor o se puede especificar un comando precedido de cinco números o comodines que indican cuando se va a ejecutar la tarea. En una tarea cron de usuario no se especifica el nombre de usuario utilizado para ejecutar el trabajo como en las tareas cron del sistema. Esta información está derivada del propietario de la tarea cron.

Los archivos que almacenan las tareas de usuario se pueden localizar en los directorios /var/spool/cron, /var/spool/cron/tabs o /var/spool/cron/crontabs. Cada archivo en este directorio es el nombre del usuario a cuyo nombre se ejecuta la tarea. Por ejemplo, el archivo de tareas programadas del usuario carlos será nombrado como /var/spool/cron/tabs/carlos. No se puede editar directamente este archivo en este directorio pero sin embargo, el uso de crontab permitirá realizar cambios.

Permisos ejecutables

El acceso a la utilidad cron puede ser restringido de varias maneras.

Los permisos de los programas de cron y crontab pueden ser restringidos usando mecanismos estándar en Linux. No todas las distribuciones se configuran de esta forma, pero en aquellas que lo hacen, los usuarios deben ser capaces de programar los trabajos concron y añadirse al grupo apropiado.

Este grupo normalmente se le llama cron, pero se debe verificar el propietario del grupo y los permisos en el directorio /usr/sbin/cron y /usr/bin/crontab para estar seguro. El archivo /etc/cron.allow contiene una lista de usuarios a los que se le permite el acceso a cron.

Si este archivo está presente, solo los usuarios cuyos nombres aparecen en la lista pueden usar cron y al resto se les deniega el acceso. Si el archivo no está presente cualquiera podrá usar cron asumiendo que el acceso no está restringido por permisos ejecutables o lista de usuarios no permitidos.

La lista a los que no se les permite el acceso está en el archivo /etc/cron.deny. Si este fichero está presente a cualquier usuario cuyo nombre aparezca en la lista tendrá el acceso denegado a cron, pero el resto podrá utilizarlo, asumiendo que los permisos ejecutables y los usuarios de la lista permitida no tienen el acceso restringido.

Gestión de usuarios

Los usuarios del sistema es uno de los aspectos fundamentales de GNU/Linux. Debido a que GNU/Linux es multiusuario, la gestión de estos en el sistema se torna sumamente importante en el aspecto tanto de la propia gestión como de la seguridad del sistema. Podemos identificar 3 perfiles de usuarios en un sistema Linux:

- Usuario root: También llamado superusuario o administrador. Es la única cuenta de usuario que tiene privilegios sobre todo el sistema, contando con el acceso a todos los ficheros y directorios con independencia del propietario o permisos. Además, puede controlar la administración de cuentas de usuario, ejecutar tareas de mantenimiento, detener el sistema, instalar software y tareas más avanzadas como reconfigurar el Kernel, compilar e instalar controladores, etc.
- Usuarios especiales: También llamadas cuentas del sistema. Estas cuentas tienen privilegios para el servicio o programa al que estén asociados, siendo esto así para aumentar la seguridad del sistema. La mayoría de estas cuentas no tienen contraseña debido a que no están pensadas para iniciar sesión con ellas, y su creación o gestión dependen del servicio. Así, estas cuentas se crean o destruyen dependiendo si el paquete está o no instalado en el sistema. Un ejemplo de estas cuentas pueden ser bin, daemon, adm, lp, sync, shutdown, mail, operator, squid, apache, etc.
- Usuarios normales: Estas son las cuentas de los usuarios personales. Cada usuario dispone de un directorio de trabajo que se ubica generalmente en /home. Estas cuentas de usuario tienen todos los privilegios solo en su espacio de trabajo, por tanto pueden personalizar su entorno modificando ficheros relacionados con el inicio de sesión.

Linux tiene un sistema de identificación de usuarios, llamado UID, así también para los grupos del sistema llamado GID. El usuario root siempre tiene el UID cero, la gestión de cuentas del sistema suele tener una reserva de identificaciones que van del 1 al 100, y finalmente los usuarios normales que suelen comenzar a partir del 500 en adelante. Todos estos valores se pueden modificar en el fichero **/etc/login.defs**

Todos los usuarios almacenan la definición de sus cuentas en el fichero **/etc/passwd**. Este fichero tiene un formato concreto, dividido en 7 campos separados por el carácter dos puntos “:”. Estos campos determinan el nombre del usuario, la contraseña, el UID del usuario, el GID del usuario, comentarios, el directorio de trabajo y la shell que utiliza.

La contraseña del usuario, ya de forma predeterminada, se encuentra, de forma encriptada, en el fichero **/etc/shadow**. En este fichero, además de esto, se encuentra otro tipo de información relacionada con la administración avanzada de usuarios: caducidad de la cuenta, cambios de contraseña, etc.

Para cambiar de usuario la forma general es con el comando su. Esta aplicación también nos permite la ejecución de comandos con un usuario diferente al nuestro. Si escribimos solamente el comando su, cambiaremos de forma automática al usuario root.

```
# su - usuario2
Enter password:
```

El uso del guión intermedio en el comando anterior es importante como elemento de seguridad. La utilización de “su -” permite al nuevo usuario o comando ejecutado tomar sus variables de entorno, debido a que la opción de guión crea una nueva shell para el usuario. Para la ejecución de comandos como otros usuarios existe otra herramienta que se ha convertido en un estándar de administración llamada sudo.

La mayoría de distribuciones de escritorio (como el caso de Ubuntu) se basa en un estándar de administración que indica que el sistema debe tener usuarios normales logueados en su uso habitual, y para tareas puntuales de administración usar al usuario root. El uso del comando su tiene una desventaja o problema de seguridad debido a que, para ejecutar un comando de administración, se debe escribir la clave de root en la terminal, y aplicaciones como los keyloggers pueden obtenerla. El comando sudo fue creado precisamente para evitar esta situación, ya que la aplicación permite elevar los permisos del usuario a root estableciendo la clave del propio usuario para tareas concretas.

Sudo tiene un fichero de administración y unos comandos para su gestión. El fichero **/etc/sudoers** tiene información de los usuarios permitidos por la aplicación, además de que comando concreto pueden ejecutar.

Un ejemplo del fichero es el siguiente:

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

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root ALL=(ALL:ALL) ALL

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

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

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

#includedir /etc/sudoers.d
```

Este fichero sólo puede ser editado como root con el comando “visudo”. Para utilizar el comando sudo, la sintaxis es similar a su:

```
# sudo usuario2
```

Instalación de Software

La gestión de software de Linux es una de las bazas más importante de las que usar software libre. Como su propio indica, al ser libre los sistemas Linux utilizan los llamados repositorios de paquetería para nutrirse de aplicaciones. Estos repositorios son servidores web esparcidos por internet donde cuentan con el almacenamiento de todos los paquetes de las distribuciones. De este modo, cualquier usuario de Linux se podrá descargar de una forma cómoda y sencilla cualquier aplicación, sin tener que buscar por páginas web dicho software.

El sistema cuenta con varias formas de poder obtener este software. A nivel gráfico en Ubuntu contamos con la herramienta centro de software, y en otros sistemas está la famosa aplicación Synaptics para la gestión. Nosotros para este curso vamos a ver la herramienta utilizada en sistemas Debian para la gestión de paquetería llamada apt.

El comando **apt** posee las siguientes características:

- Facilita la actualización del sistema de una forma sencilla.
- Realiza la gestión inteligente de las dependencias de la paquetería (instalación de un paquete y sus dependencias, eliminación, etc...)
- Puede realizar consultas a diferentes repositorios, siendo muy sencillo buscar una aplicación o servicio.
- Ofrece servicios de eliminación de paquetes obsoletos o no usados.

Los ficheros de configuración se encuentran principalmente en el directorio */etc/apt* **.**

A continuación se indica para qué sirve cada uno:

/etc/apt/apt.conf: Fichero de configuración global

/etc/apt/sources.list: Fichero de repositorios

/var/cache/apt/: Directorio de caché

El primer comando se utiliza principalmente para realizar búsquedas de paquetería y mostrar estadísticas de la caché, este comando es **apt-cache:**

```
apt-cache [option] [action] [package]
```

Este comando cuenta con las siguientes acciones disponibles:

showpkg *paquete* **:** Muestra la información del paquete.

stats: Muestras estadísticas sobre la caché.

unmet: Muestra un resumen de todas las dependencias no satisfechas en la caché.

show *paquete* **:** Muestra todas las versiones del paquete

search *regularexp* **:** Hace una búsqueda en el repositorio de un paquete. Puede hacer búsquedas de expresiones regulares.

El siguiente comando es el responsable de la gestión de la paquetería en el sistema, pues se encarga de la instalación, desinstalación, actualización, etc. Se trata del comando **apt-get:**

```
apt-get [option] [action] [package]
```

Este comando cuenta con las siguientes acciones:

update: Actualiza las cabeceras del listado de paquetes disponibles obteniéndolos del repositorio ubicado en **/etc/apt/sources.list**

upgrade: Realiza una actualización parcial del sistema.

dselect-upgrade: Realiza cualquier cambio en *package status.*

dist-upgrade: Realiza una actualización total del sistema, resolviendo conflictos que podrían romper dependencias.

install: Instala un paquete.

remove: Elimina un paquete.

--**purge:** Elimina sus ficheros de configuración.

source: Se descarga las fuentes el paquete deb. Se requiere el repositorio *deb-src* y el paquete *dpkg-dev.*

check: Actualiza la caché de paquetes y revisa la existencia de dependencias rotas.

clean: Borra totalmente el repositorio local que contiene los ficheros de paquetes descargados.

autoclean: Similar a **clean** , con la diferencia que solo borra los paquetes que no se pueden descargar o son inservibles.

autoremove: Sirve para eliminar paquetes que han sido instalados automáticamente como dependencias y ya no están en uso (por desinstalación del paquete).

Sus posibles opciones son:

-d: Solamente se descarga el paquete.

-f: Intenta arreglar un sistema con dependencias rotas.

-m: No tiene en cuenta los paquetes que no se hayan podido descargar o después de la descarga estén dañados.

-q: Produce una salida silenciosa, puede ser un nivel mas silenciosa si se añade una segunda **q.**

-s: Realiza una simulación de la acción a hacer.

-y: Supone una respuesta afirmativa a todas las preguntas.

-b: Descarga los paquetes fuente y luego los compila.

--no-upgrade: No actualiza los paquetes.

Curso para Aprender Linux desde Cero

Impartido por ...

Créditos...

OpenWebinars.Net

Septiembre 2015

PDF Realizado con LibreOffice 5.0