Shell de linux: Una guía básica

Pedro Pablo Fábrega Martínez

http://dns.bdat.net/shell/book1.html

http://dns.bdat.net/shell/

Linux (Unix) para usuarios

Pedro Pablo fabrega(EN)jazzfree.com

Esta guía pretende que el lector se familiarice con el trabajo en la shell de un sistema Unix, Linux en particular. Todo el software descrito en la presente guía es software libre, por lo que el lector no debería tener problema en conseguirlo por los cauces habituales, es decir distribuciones de Linux e Internet. Por otro lado esta guía se limita a la descripción de tareas a nivel de usuario, o sea, que todo lo que aquí se indica (o casi todo) se puede hacer sin tener acceso a la cuenta de root. Las tareas de administración serán otro capítulo.

Tabla de contenidos

- **Orden less: Mostrar un fichero de texto**
  - **Comentarios:**
- **Orden mkdir: Crear un directorio**
- **Orden rm: Borrar un fichero o directorio**
  - **Comentarios:**
- **Orden cd: Cambia el directorio activo**
  - **Usos especiales de cd**
- **Orden mv: Mueve o renombra ficheros**
- **Orden cp: Copia ficheros y directorios**
- **lpr Imprime un fichero**
- **Orden pwd: Imprimir el directorio actual**
- **Orden ln: Enlaza ficheros o directorios**
- **Ejercicios con solución**
  - **Mostrar el contenido de todos los ficheros del directorio personal.**
  - **Copiar el fichero .profile en otro llamado perso.**
  - **Crear un directorio llamado prueba en nuestro directorio personal.**
  - **Expresar las rutas absoluta y relativa del directorio prueba que acabamos de crear. (Relativa al directorio actual).**
  - **Copiar el fichero /home/usuario/perso en el directorio prueba.**
  - **Cambiar al directorio /home/usuario/prueba**
  - **Copiar el fichero perso del directorio /home/usuario/prueba con el nombre perso.nuevo**
  - **Estando en el directorio prueba, copiar el fichero .profile en él con el nombre prof.nuevo**
  - **Crear un enlace simbólico llamado pro1, al fichero prof.nuevo.**
  - **Verificar el enlace simbólico pro1 que acabamos de crear**
  - **Crear un enlace duro llamado prof.d1, al fichero prof.nuevo.**
  - **Crear otro enlace enlace duro llamado prof.d2, al fichero prof.d1.**
  - **Borrar el fichero prof.d1 y verificar los demás**
  - **Crear un directorio dentro de prueba llamado src.**
- **Comprimir, descomprimir y agrupar ficheros**
  - **Comprimir ficheros y uso de ficheros compromidos**
    - **gzip**
    - **gunzip**
    - **zcat**
    - **zless**
    - **bzip2**

        - **bunzip2**
        - **zip**
        - **unzip**
    - **Agrupar y desagrupar ficheros: tar**
        - **Agrupar ficheros**
        - **Agrupar ficheros y comprimir**
        - **Desagrupar ficheros**
        - **Desagrupar ficheros y descomprimir**
    - **Ejercicios resueltos**
        - **Hacer una copia de seguridad del directorio /home**
- **Otras órdenes de usuario**
    - **Orden id: Información sobre el usuario**
    - **Orden passwd: Modifica la clave**
    - **Orden man: obtener información**
    - **Orden who: información sobre usuarios conectados**
    - **orden whoami: información sobre el usuario**
    - **Orden write: envía un mensaje a un usuario**
    - **orden mesg: activa/desactiva la recepción de mensajes**
    - **orden mail: envía un mensaje de correo electrónico**
    - **Orden date: muestra las hora y fecha actuales**
    - **Orden echo: muestra en pantalla el resto de la línea**
    - **orden sort: ordenar el contenido de un fichero**
    - **Orden more: muestra un fichero en pantalla**
    - **Orden cal: muestra un calendario**
    - **Orden expr: evalúa una expresión entera**
    - **Orden diff: muestra diferencias entre ficheros**
    - **find Localiza ficheros**
    - **Orden ps: muestra lista de procesos**
    - **Orden sleep: genera un proceso durante cierto tiempo**
    - **Orden stty: parámetros del terminal**
    - **Orden head: muestra las primeras línea de un fichero**
    - **tail: Muestra las últimas línea de un fichero**
- **Orden touch: actualiza las fechas de un fichero**
    - **Orden tty: muestra el terminal**
    - **Orden wc: cuenta información sobre ficheros**
    - **Ejemplos**
- **Gestión de procesos**

- **Guion de shell que genere un fichero llamado listaetc que contenga los ficheros con permiso de lectura que haya en el directorio /etc:**
- **Hacer un guion de shell que, partiendo del fichero generado en el ejercicio anterior, muestre todos los ficheros del directorio /etc que contengan la palagra ?procmail?:**

Introducción al Unix

Vamos a describir las generalidades de Unix de una forma ligera. Realizaremos una descripción de los elementos que son de utilidad en el posterior desarrollo de los contenidos y algunas curiosidades

Historia de Unix

El sistema operativo Unix tiene su origen en los laboratorios Bell de AT&T en los años 60. Estos laboratorios trabajaban en un proyecto muy ambicioso de sistema operativo nuevo llamado MULTICS (Multiplexed Information and Computing System. Este proyecto fue un fracaso debido a la complejidad, pero los componentes del equipo adquirieron una gran experiencia durante su desarrollo.

Uno de los componentes del equipo, Kem Thompson, escribió un juego llamado "Space Travel" y escribió un sistema operativo para poder jugar con él. Con este sistema operativo consiguió que dos personas pudieran jugar simultáneamente a "Space Travel". Con bastante ironía, usando un juego de palabras en comparación con MULTICS, llamó al sistema operativo UNICS, que más tarde derivaría en Unix.

Inicialmente, este sistema UNIX estaba escrito en lenguaje ensamblador, lo que dificultaba que se pudiera usar en máquinas con distintos procesadores. Viendo el problema, Ken Thomson y Denis Ritchie crearon un lenguaje de programación de alto nivel, el lenguaje C, en el cual reescrbieron todo el código del sistema operativo lo que permitió que se pudiera usar en prácticamente cualquier tipo de ordenador de la época. Sólo las partes críticas seguían en ensamblador. Unix fue el primer sistema operativo escrito en un lenguaje de alto nivel.

Lo que inicialmente comenzó como un juego, distribuyénsose como proyecto de investigación en algunas universidades, se convirtió en un éxito comercial por lo que los laboratorios Bell comezaron su distribución.

Más tarde un decisión judicial obligó a AT&T a dejar de vender su sistema operativo. Esta compañía dejó las fuentes del sistema operativo a diversas universidades, las cuales, junto con otras empresas, continuaron el desarrollo del sistema operativo Unix e hizo que tuviera una enorme difusión.

Cronología

1975 La Universidad de California en Berkeley continúa con el desarrollo de UNIX incorporando sus propias características y modificaciones y uno de los desarrolladores iniciales de Unix, Ken Thompson, edita su propia versión de UNIX, conocida con el nombre de BSD. Desde entonces BSD pasó a ser la gran competidora de los laboratorios Bell.

1980 A principios de los años 80 surge Unix Sistema III, la primera versión comercial del sistema operativo UNIX. En 1983 AT&T introdujo el UNIX Sistema V versión 1.

1983 Aparece Unix BSD versión 4.2 Entre sus características principales se encuentran una gran mejora en la gestión de ficheros el trabajo en red basadas en los protocolos TCP/IP. Esta versión de UNIX la adoptaron varios fabricantes, entre ellos Sun Microsystems, lo que dió lugar al conocido sitema SunOS.

Versiones de Unix

Unix tiene dos variantes fundamentales, los Unix Sistema V y los Unix BSD (Berkeley Software Development).

En la actualidad las versiones comerciales más importantes de UNIX son:

- Solaris: El Unix de Sun Microsystems. Originalmente, Sun Microsystems editó SunOS de tipo BSD para posteriormente editar Solaris basado en Sistema . Exuisten versiones de Solaris para procesadores Power PC, Intel y Sparc.
- AIX: La versión del sistema operaivo UNIX de IBM se llama AIX y está basada en Sistema V versión 3 y BSD 4.3.
- A/UX: Desarrollo de UNIX de Apple

El sistema de ficheros

El sistema de ficheros es la organización lógica del disco que nos permite almacenar la información en forma de ficheros de un modo totalmente transparente. Esta palabra tan utilizada, transparencia, significa que no tenemos que preocuparnos de pistas, sectores, cilindros y otras menudencias; el sistema se encarga de eso por nosotros. Nosotros simplemente utilzamos un nombre de fichero, el sistema se encarga de el resto.

Cada partición del disco, o cada disquete debe tener un sistema de ficheros si queremos almacenar información en forma de fichero asignándole un nombre. Tenemos que resaltar que un sistemade ficheros forma parte de las propiedaes de cada partición de disco duro, de disquete, dispositivo de almacenamiento USB o cdrom. Una partición sin sistema de ficheros no permite almacenar información.

Tipos de sistemas de ficheros

Cada sistema operativo posee su propia organización lógica del disco para poder almacenar la información, y la usará normalmente, pero además puede tener la posibilidad de usar particiones propias de de otros sistemas. Entre los tipos de sistemas de ficheros podemos

citar:

- ext2: linux nativo. Extendido 2, es un sistema de ficheros propio de linux. Soporta características avanzadas: propietarios, permisos, enlaces, etc.
- ext3: linux nativo con journaling. Extendido 3 es similar a ext2 pero con transacciones para evitar que apagados accidentales puedan deteriorar el sistema de ficheros.
- msdos: es la organización clásica de este sistema. Es un sistema de archivos diseñado para un sistema monousuario. Utiliza nombres del tipo 8+3. En la actualidad sólo se utiliza en en ciertos dispositivos como cámaras digitales debido a su limitación en el nomvre de ficheros.
- vfat: es una ampliación del sistema de ficheros msdos, con soporte para nombres largos de ficheros. Existen los tipos FAT12, FAT16 y FAT32, y en todos los casos sólo tienen características monousuario: no admiten propietarios de ficheros y los permisos son muy limitados. Los valores 12, 16 y 32 indican el número de bits que se utiilzan para almacenar el número de una únidad de almacenamiento. (sectores o clusters). Con FAT 12 el número máximo de unidades de almacenamiento que se pueden direccionar son 2^12=4096, que indica el límite de almacenamiento. Con el resto podemos realizar los mismos cáculos.
- NTFS: sistema de ficheros de Windows NT/XP. Es un sistema de ficheros con características avanzadas y sí está preparado para utilizarse en entornos multiusuario. Es aconsejable utilizarlo en máquinas Win32 cuando exista la posibilidad.
- iso9660: es el sistema de ficheros de los CDs. Este estándar admite ciertas extensiones como «Joliet» o «Rock Ridge» que le añaden ciertas características.

El sistema de ficheros Unix

Los elementos del sistema de ficheros son el superbloque, i-nodos y bloques de datos. En el capítulo de administración del sistema de ficheros se ve todo esto con más detalle.

En primer lugar tenemos el superbloque, que contiene la descripción general del sistema de ficheros: Tamaño, bloques libres, tamaño de la lista de i-nodos, i-nodos libres, verificaciones, etc. El superbloque siempre es el primer bloque del sistema de ficheros.

En segundo lugar tenemos los i-nodos. Un i-nodo contiene toda la información sobre cada conjunto de datos en disco, que denominamos fichero:

- Donde se almacenan los datos, es decir lista de bloques de datos en disco. Esto son una serie de punteros o direcciones de bloques que indican bien donde están los datos en disco, o bien donde están los bloques que tienen más direcciones de bloques de datos (bloques indirectos).
- Quien es el propietario de los datos, un número que lo identifica (UID o User Identifier),

y a qué grupo pertenece el fichero GID Group Identifier).

- Tipo de fichero: regular, es decir un fichero que contiene información habitual, datos o programas; dispositivo, un elemento destinado a intercambiar datos con un periférico, enlace, un fichero que apunta a otro fichero; pipe, un fichero que se utiliza para intercambiar información entre procesos a nivel de núcleo. directorio, si el elemento no contiene datos sino referencias a otros ficheros y directorios.
- Permisos del fichero (quien puede leer(r), escribir(w) o ejecutar(x)). Estos permisos se asignan a se asignan de forma diferenciada a tres elementos: el propietario, el grupo (indicados con anterioridad) y al resto de los usuarios del sistema.
- Tamaño del fichero.
- Número de enlaces del fichero. Es decir cuantos nombres distintos tiene este fichero Hay que observar como el nombre de un fichero no forma parte del i-nodo. El nombre de fichero se asocia a un i-nodo dentro de un fichero especial denominado directorio. Esto le proporciona al sistema de ficheros la posibilidad de que un mismo i-nodo pueda tener varios nombres si aparece en varios directorios o con distintos nombres.

Entrando a un sistema Unix

Unix es un sistema multiusuario real, es decir, pueden haber varias personas trabajando a la vez en distintas terminales con un mismo host Unix. Esto implica que cada usuario tenga que identificarse de forma adecuada ante el sistema para que éste pueda determinar que privilegios le corresponden. La identificación consiste en suministrarle al sistema una pareja de nombre y contraseña correctas.

Iniciando una conexión

Existen diferentes métodos para poder conectar los terminales al sistema:

- En primer lugar podemos conectarnos a un sistema Unix a través de el puerto serie (RS232), con una terminal no inteligente o bien con otro equipo y un emulador de terminales. En ambos casos existe un programa que atiende las solicitudes de conexión a través del puerto serie. Cuando hay una solicitud de conexión, este programa la atiende solicitando al usuario que se identifique ante el sistema. Cuando termina la conexión, este programa se reactiva para seguir atendiendo nuevas solicitudes. En realidad citamos este sistema de conexión, más que nada, por motivos históricos, en la actualidad no se utiliza.
- Mediante tarjeta de red. En este caso, tenemos un programa que escucha las solicitudes de conexión a través de la tarjeta de red. Cuando llega una solicitud este

programa se desdobla de forma que una parte atiende la conexión y otra continúa atendiendo nuevas conexiones. Así podemos tener más de una conexión a través de la tarjeta de red. Para hablar con más propiedad, en lugar de decir tarjeta de red debería haber dicho interfaz de red, que puede corresponder a una tarjeta de red, a un modem, una conexión infrarojos, etc.

- La consola. Evidentemente, en un sistema Unix también podemos trabajar desde el teclado y monitor que están conectados directamente al sistema.

Iniciando una sesión

Una vez que hemos conseguido conectarnos a un sistema Unix tenemos que iniciar una sesión de trabajo. Como dijimos, Unix, y Linux como tal también, son sistemas multiusuario reales, y esto exige que el usuario se presente al sistema y que este lo acepte como usuario reconocido. Así cada vez que iniciamos una sesión Linux nos responde con

```
Login:
```

a lo que nosotros debemos responder con nuestro nombre de usuario. Acto seguido, Linux nos solicita una clave para poder comprobar que somos quien decimos que somos:

```
Password:
```

En este caso tecleamos la clave de acceso. Por motivos de seguridad esta clave no aparecerá en la pantalla. Si la pareja nombre de usuario/clave es correcta el sistema inicia un intérprete de órdenes con el que podemos trabajar. Habitualmente será el símbolo $, aunque puede ser también el símbolo % (si usamos una shell C). Cuando es el administrador (root) quien está trabajando en el sistema, el indicador que aparece es #.

Si entramos directamente en modo gráfico también deberemos tener de una forma parecida, que indicar un nombre de usuario y una contraseña.

La base de datos de los usuarios

Hemos visto que para iniciar una sesión de trabajo en un sistema Unix teníamos que suministrar al sistema una pareja de nombre de usuario/clave. Estos datos se almacenan en un fichero llamado /etc/passwd. Este fichero contiene una línea por cada usuario del sistema. Cada línea consta de una serie de campos separados por dos puntos (:). Estos campos son, en el orden que aparecen:

- Nombre de usuario. Es es nombre con el que no presentamos al sistema, con el que tenemos que responder a Login: y por el que nos identifica el sistema.
- Clave cifrada. El siguiente campo es la clave de acceso al sistema. Esta clave no se guarda como se introduce, sino que se almacena transformada mediante el algoritmo DES para que nadie pueda averiguarla.
- UID. Identificador de usuario. Es el número de usuario que tiene cada cuenta abierta en el sistema. El sistema trabaja de forma interna con el UID, mientras que nosotros trabajamos con el nombre de usuario. Ambos son equivalentes.
- GID. Identificador de grupo. Es el número de grupo principal al que pertenece el usuario.
- Nombre completo de usuario. Este es un campo meramente informativo, en el que se suele poner el nombre completo del usuario.
- Directorio personal. Este campo indica el directorio personal de un usuario, en el cual el usuario puede guardar su información.
- shell. El último campo indica un programa que se ejecutará cuando el usuario inicie una sesión de trabajo. Normalmente este campo es una shell que proporciona una línea de órdenes para que el usuario trabaje.

Ejemplo:

```
usuario:x%6YkH$Ss:505:705:Cuenta de usuario:/home/usuario:/bin/bash
  ^         ^    ^   ^        ^              ^         ^
  |         |    |   |        |              |         |
  |         |    |   |        |              |         shell
  |         |    |   |        |         directorio personal
  |         |    |   |    Nombre completo del usuario
  |         |    |  Número de grupo (GID)
  |         |   Número de usuario (UID)
  |      Clave cifrada
 Nombre de usuario
```

En la actualidad el almacenamiento y gestión de usuarios puede ser algo más complejo que lo indicado en las anteriores líneas. Por lo pronto nos basta saber qué estos son los datos imprescindibles que necesita conocer el sistema sobre un usuario.

La shell

Como dije en el punto anterior, la shell es el intérprete de órdenes de un sistema Unix. No hay que confundir la shell con el sistema operativo. El sistema operativo es el núcleo y la shell es un interfaz que nos proporciona utilidades de trabajo y permite establecer una

relación con el núcleo. Es decir, la shell reconoce una serie de órdenes, las interpreta y lanza los procesos necesarios para su realización.

Hay diversos tipos de shells, cada una con sus características. Podemos citar:

- Bourne shell (sh)
- Korn shell (ksh)
- Shell C (csh)
- Bourne again shell (bash)

Para desarrollar los contenidos que vienen a continuación voy a utilizar siempre las shell bash, y algún comentario ocasional con las diferencias con otras shell.

Metacaracteres de la shell

Existen ciertos caracteres que tienen un significado propio para la shell. La shell sabe interpretarlos para modificar algunos comportamientos. Estos caracteres son:

*	equivale a cualquier cadena de texto en un nombre de fichero	?	equivale a cualquier cadena de texto en un nombre de fichero
[ ]	evalúa una condición lógica	[!]	evalúa una condición lógica negativa
|	Tubería, utiliza la salida de la orden de la izquierda, como entrada de la orden a la derecha	\	Protege el siguiente metacarácter para que la shell no lo interprete
&	Ejecuta un proceso en segundo plano	$	Extrae el contenido de una variable. Para usar el contenido de una variable tenemos que ponerle delante un $
>	redirige la salida estándar, creando un fichero nuevo borrando el existente	>>	redirige la salida estándar, creando un fichero nuevo o añadiendo al existente
2>		2>>	redirige la salida de errores, creando un fichero nuevo o añadiendo al existente
<	Redirige la entrada estándar	( )	Agrupación de órdenes.

&&	Y lógico entre órdenes	||	O lógico entre órdenes

*

Si en alguna ocasión nos interesa usar alguno de estos caracteres como literal, es decir, que la shell no lo interprete como carácter especial es necesario que esté precedido (protegido) por el carácter de escape \ (barra invertida).

Entrada y salida estándares y de errores

Un sistema Unix dispone de tres vías o canales para comunicarse con el exterior de forma estándar.

Una de ellas, la entrada estándar, se utiliza para introducir datos en la shell; de forma predeterminada está asociada al teclado.

La salida estándar, se utiliza para mostrar información y de forma predeterminada está asociada al monitor (consola). La salida estándar es donde vuelca la información de salida la ejecución de una orden.

Por último existe un canal dedicado a mostrar la salida de errores, que de forma predeterminada está asociado a la salida estándar. Cuando la ejecución de una orden falla por cualquier motivo, el sistema lo notifica a través de esta canal. Normalmente, como el canal va asociado a la salida estándar, los errores aparecen en panaalla.

En múltiples ocasiones nos puede interesar redirigir alguna de estas salidas a otro canal; para realizar esto disponemos de los metacaracteres :

<	redirige la entrada estándar. Podemos utilizar este metacarácter cuando queramos sustituir una serie de entradas por teclado por el contenido de un fichero. Unix tatra exactamente igual el teclado qe si fuera un fichero, él ve una serie de caracteres terminados en un retorno de carro.
>	redirige la salida estándar. Si esta redirección es a un fichero, lo crea nuevo en caso de que no exista, y si existe elimina su contenido previo. Un descuido con la redirección no puede hacer perder un fichero fácilmente
>>	redirige la salida estándar. Si esta redirección es a un fichero, y el fichero existe la añade al final de éste, y si no existe lo crea. Utilizando esta redirección podremos añadir fácilmente una línea a un fichero.
2>	redirige la salida de errores. Si esta redirección es a un fichero, lo crea nuevo en caso

	de que no exista, y si existe elimina su contenido previo.
2>>	redirige la salida de errores. Si esta redirección es a un fichero, y el fichero existe la añade al final de éste, y si no existe lo crea.

Veremos algunos ejemplos en los próximos capítulos.

Organización del almacenamiento en disco

El almacenamiento en disco se organiza en dos elementos básicos, el fichero y el directorio. Intuitivamente la organización lógica de los ficheros y directorios en disco es similar a las hojas y ramas en un árbol. A cada hoja sólo se puede llegar siguiendo una serie de ramas y cada rama puede contener hojas y más ramas. En nuestro caso, las ramas serán los directorios y las hojas los ficheros.

Hablando con más propiedad, un fichero es un conjunto de datos en disco asociado a un i-nodo. El sistema operativo identifica a un fichero por su número de i-nodo. En el i-nodo se especifican entre otras cosas los permisos, el propietario y la distribución de los datos del fichero sobre el disco.

Un directorio es un fichero especial donde se asocia un nombre a un número de i-nodo. Esto produce la sensación de que el directorio contiene ficheros y otros directorios. Esta organización nos permite que podamos asignar varios nombres a un mismo conjunto de datos en disco.

En ocasiones puede suceder que cambiemos el nombre de un fichero y otro proceso que lo tuviera abierto y escribiendo en él, lo siga haciendo ya que el cambio de nombre no afecta al i-nodo en sí.

En el capítulo de administración del sistema de ficheros se trata todo esto con más profundidad.

Árbol de directorios

Los sistemas Unix se organizan en un único árbol de directorios. Existe un directorio principal, denominado raíz (/) dentro del cual aparecen el resto de directorios que tenga el sistema.

Esto quiere decir que los distintos discos no aparecen como tales discos independientes en es sistema, sino que aparecen como un directorio más y esto lo podemos organizar de acuerdo con nuestros intereses. Cuando hablábamos del sistema de ficheros decíamos que intuitivamente se podía ver cada sistema de ficheros como un árbo con ficheros y directoriasl,

pues ahora cada sistema de ficheros se integra en el árbol de directorio como una rama más. Los disspositivos extraibles se integran como ramas que se pueden poner o quitar.

En el siguiente ejemplo podemos observar una forma de organizar un sistema Linux

```
$ mount
/dev/hdb1 on / type ext3 (rw)
none on /proc type proc (rw)
/dev/hdb3 on /usr type ext3 (rw)
/dev/hdb4 on /opt type ext3 (rw)
/dev/hda5 on /tmp type ext3 (rw)
/dev/hda6 on /usr/src type ext3 (rw)
/dev/hda7 on /usr/local type ext3 (rw)
```

Esto quiere decir que la primera partición (1) del segundo disco duro (hdb) contiene el directorio raíz (/). La tercera partición (3) del segundo disco duro (hdb) aparece en en directorio /usr. La séptima partición (7) del primer disco duro (hda) aparece en el directorio /usr/local.

Rutas de acceso a ficheros y directorios

Nombres de ficheros y directorios

El nombre de un fichero o directorio está formado por letras, números y otros caracteres, salvo el carácter / que se utiliza como separador en una lista de directorios. Debemos evitar los metacaracteres de la shell en los nombres de ficheros. En caso necesario, si el nombre tiene del fichero tiene un metacarácter o espacio en blanco, será necesario proteger el nombre completo del fichero con comillas (") o el metacarácter concreto, incluido el espacio en blanco, con el carácter "\". El espacio es necesario protegerlo porque si no se hace la shell lo toma como final del nombre.

Por ejemplo, serían nombres válidos de ficheros:

```
fichero-datos.txt
"fichero de datos.txt"
fichero\ de\ datos.txt
fichero_copia\(1\).txt
```

No serían nombres de ficheros válidos:

```
fichero>datos.txt
fichero de datos.txt
fichero_copia(1).txt
```

Los sistemas unix no identifican ninguna extensión en los nombres de ficheros, el nombre de

un fichero en ningún caso determina su funcionalidad, es decir, un ejecutable no tiene por qué terminar en .exe, por ejemplo. Esto con respecto al sistema operativo, pero sí puede ocurrir que estemos utilzando un entorno gráfico de escritorio y este entorno sí que pueda identificar ciertos nombres de ficheros para aplicar unas características, pero esta labor es del entorno de escritorio, no del sistema operativo.

Plantillas para nombres de ficheros

Hay ocasiones en que nos interesa usar nombres de ficheros que hagan referencia, no a un fichero individual, sino a un conjunto de ficheros. Para estos casos tenemos las plantilla. Una plantilla se forma con caracteres normales y mediante los caracteres:

- * equivale a cualquier cadena de caracteres. Por ejemplo ab* equivale a todos los ficheros que empiecen por ab.
- ? equivale a un carácter individual. Por ejemplo ab? equivale a los nombres de ficheros (o directorios) con tres caracteres y los dos primeros son ab.

Ruta de un fichero

Para indicar la ubicación en disco de un fichero hay que indicar la lista de directorios que contienen al fichero. Es decir, un fichero puede estar dentro de un directorio que a su vez está dentro de otro y así varios. En esta lista, que denominaremos ruta de acceso, cada directorio está separado del siguiente directorio por el signo / , y sin dejar espacios en blanco. Por ejemplo:

```
/usr/bin/wc
```

hace referencia la fichero wc que está contenido en el directorio bin, que a su vez está dentro del directorio usr, que está en el directorio raiz.

Para facilitar las cosas existen ciertos directorios especiales:

- / Es el directorio raíz. El superior de todos.
- . Es el directorio activo. En el que nos encontramos en un momento dado. El directorio actual se puede cambiar con una simple orden que se verá con posterioridad.
- .. Es el directorio superior al que nos encontramos. El único directorio que no tiene directorio superior es el directorio raíz.

Una ruta tiene que especificar de forma única un elemento de sistema de ficheros. Ahora veremos como hay dos formas de expresar la ruta de un fichero, rutas absolutas y rutas relativas.

Ruta absoluta

Una ruta absoluta es aquella que parte del directorio raíz. Las rutas absolutas son válidas en cualquier caso. Si establecemos como ejemplo paralelo la dirección de la vivienda, la dirección abosluta incluye la calle y el número de la vivienda, de forma única e independiente.

Ejemplo:

```
/home/usuario/.profile
```

Ruta relativa

Es una ruta que parte del directorio actual como origen. Esta ruta sólo es válida desde un directorio actual concreto , es decir es relativa a un directorio. Si seguimos el ejemplo que pusimos para la dirección de una vivienda, una ruta relativa es como si te indicaran "dos casas más allá de cierta esquina". Observamos como la dirección "dos casas más allá una cierta esquina" es sólo válida para una calle, y esa misma dirección referida a otra calle sería otro inmueble distinto. Es decir la ruta relativa depende del punto de partida.

```
../../.profile
```

En este caso estamos haciendo referencia al fichero .profile que está dos directorios por encima del directorio activo.

Propietarios y Permisos

Propiedad

Como vimos en la descripción del sistema de ficheros, cada i-nodo guarda un espacio para indicar los números de usuario y grupo. Entonces cada i-nodo pertenece a un usuario y a un grupo. También el usuario que tenga el UID (número de identificación) descrito en la base de datos de usuario /etc/passwd que coincida con el del fichero será su propietario. Esto también es válido para directorios.

Los propietarios, se utilizan como mecanismo de seguridad para poder asignar ciertas propiedades en función del usuario, en particular los permisos.

Permisos

Cada i-nodo guarda un espacio para almacenar los permisos bajo los cuales se puede acceder a un fichero. Los permisos se aplican al propietario, al grupo y al resto de los usuarios. Unix dispone de tres permisos, lectura(r), escritura(w) y ejecución(x). En total

tendremos nueve bits que indican los distintos permiso en el siguiente orden: usuario, grupo, otros. Los permisos los podemos expresar en formato octal. Por ejemplo el valor 751 indicará:

```
usuario 7 = 111 = rwx
grupo   5 = 101 = r-x
```

otros 1 = 001 = --x

De esta forma los permisos quedan rwxr-x--x

Hay que tener en cuenta que los permisos tienen distinto significado si se aplican a un fichero o a un directorio.

- Permiso de lectura Permite o evita que alguien pueda leer el contenido de un fichero o de un directorio. En el directorio significa ver qué ficheros contiene.
- Permiso de escritura En el caso de un fichero, el permiso de escritura permite modificarlo o borrarlo. En el caso de un directorio este permiso da la posibilidad de crear o borrar ficheros de un directorio.
- Permiso de ejecución En el caso de un fichero, permite que sea ejecutado por quien tenga el permiso. En el caso de un directorio, el permiso de ejecuación permite entrar en él.

Órdenes

Son programas que se utilizan para gestionar el sistema y las labores de usuario y administración. Los mayoría de las órdenes siguen un formato estándar:

```
orden opciones argumentos
```

donde:

- orden: es el nombre de la orden que queremos ejecutar.
- opciones: son modificadores del comportamiento de la orden. Las opciones van precedidas de un guion (-) si son caracteres simples (letras), o por dos guiones (--) si la opción es una palabra completa. Cuando las opciones son caracteres simples se pueden agrupar con un solo guion. (Por este motivo, cuando las opciones son palabras completas van precedidas por dos guiones, para evitar confusiones). Las opciones pueden tener argumentos.
- argumentos: es la información que necesita una orden para poderse ejecutar.

Ejemplo:

```
$ ls -la /etc
```

donde

- $ Es el indicador de la línea de órdenes del sistema (promtp). Este indicador no hay que escribirlo, se supone que lo pone el sistema para indicar que está esperando un orden.
- ls Es la orden para mostrar la lista de ficheros de un directorio.
- -la aplicamos las opciones l y a que indican como queremos obtener la lista. Podemos oobservar como hemos agrupado las opciones l y a tras un mismo guion poniendo "-la".
- /etc El directorio del cual queremos obtener la lista de ficheros. Este sería el argumento de la orden ls.

Ejecución de una orden

Para ejecutar una orden simplemente lo escribimos en la línea de orden con sus opciones y argumentos y pulsamos la tecla de retorno de carro. En algunos casos puede ser necesario indicar la ruta de búsqueda del fichero que contiene la orden para poderlo ejecutar. (Ver la variable de entorno PATH).

También en una misma línea podemos lanzar varias órdenes separándolas por ; (punto y coma). De esta forma se ejecutan en secuencia, según el orden de izquierda a derecha.

Si el argumento de una orden es un fichero, podemos añadir la ruta del fichero. Si no ponemos la ruta la orden utiliza el fichero que haya en el directorio activo con ese nombre.

Obtener información

Los sistemas Unix disponen de un sistema de ayuda en línea. Basta invocar la orden man con un nombre de orden como argumento y el sistema mostrará la ayuda que tiene disponible para esa orden.

Ejemplo:

```
$ man ls
```

Notación sobre órdenes

Para efectuar la descripción de las órdenes vamos a utilizar las siguientes convenciones:

- [ ] Unos corchetes indican que el contenido de los corchetes es opcional.
- a|b Una barra vertical nos indica que tenemos que escoger entre uno u otro, a ó b en este ejemplo.

- plantilla Indica un nombre de fichero o directorio formado por caracteres normales y posiblemente los comodines * y ? vistos con anterioridad.

Por ejemplo si pusiéramos:

```
ls [opciones][ruta[/fichero]|plantilla]
```

indicaría que podemos ejecutar la orden ls y añadirle opciones si nos interesa. Respecto al argumento, que es opcional, podemos elegir entre una rura de directorio o fichero y una plantilla.

Órdenes para ficheros y directorios

A continuación vamos a describir una serie de órdenes que se utilizan en la gestión y manipulación de ficheros y directorios por parte del usuario.

En este texto sólo se describen las opciones más frecuentes de cada orden. No pretendo hacer una guía de referenicia en la ejecución de órdenes, por lo que es conveniente acostumbrarse a leer las páginas del manual de cada orden; en muchos casos nos podremos sorprender de las funcionalidades adicionales que admite.

Orden cat: Mostrar el contenido de un fichero

La orden cat muestra el contenido de un fichero por la salida estándar.

Uso:

```
cat [ruta]fichero
```

Si no ponemos ruta de acceso, muestra los ficheros del directorio actual.

Ejemplos:

```
$ cat /etc/inittab
$ cat .profile
```

La orden cat se suele utilizar en muy diversos contextos. Su principal característica es que realiza un volcado de un fichero sin añadir ni quitar nada.

La orden cat nos puede servir para crear un fichero de texto sin tener que utilizar un editor de

texto:

```
$ cat >borrar
hola
como
estas
(C-c)
$ cat borrar
hola
como
estas
$
```

Donde ponemos C-c significa pulsar la tecla Ctrl y sin soltarla pulsar la tecla c para terminar de introducir líneas.

La orden cat puede servir para unir dos más ficheros en uno nuevo

```
cat fichero1 fichero2 fichero3 > fichero.nuevo
```

En los dos ejemplos anteriores también podemos observar como hemos redirigido la salida estándar hacia un fichero.

Orden ls: Mostrar el contenido de un directorio

Muestra el contenido de un directorio, la lista de ficheros y directorios contenidos en un directorio.

Uso:

ls [-abcdfgiklmnpqrstux] [--color][directorio...]

ls tiene más opciones, pero sólo vamos a citar las más importantes. Para obtener una información mas detallada consultar

```
$  man ls
```

Opciones:

- -a Se muestran todos los ficheros de los directorios, incluyendo los "invisibles"; es decir, aquéllos cuyos nombres empiezan por punto (`.').
- -l Se muestran el tipo, los permisos, el número de enlaces duros, el nombre del propietario, el del grupo, el tamaño en bytes, y una marca de tiempo.

Ejemplos:

```
$ ls /etc
$ ls -la /etc
$ ls /dev
```

```
$ ls -la /dev
```

Nota: En Linux la opción --color hace que cada elemento del directorio aparezca de un color distinto según sea su tipo de fichero o directorio.

Comentarios:

Esta es una de las órdenes básicas de un sistema Unix, una de las más utilizadas. Si se quieren obtener ordenacines de los ficheros en su salida, en la página del manual explica tods las opciones. Si queremos ordenaciones más complejas, podemos combinarla con sort , aunque también trae sus propias opciones de ordenación.

Ejemplos prácticos de ls:

Acabamos de guardar un ficheros en el directorio personal y no recordamos exactamente el nombre. ¿Cómo lo podemos resolver? Ejecutamos:

```
ls -latr
```

y obtenemos una lista ordenada por fecha de todos los ficheros del directoiro activo.

Queremos eliminar los ficheros grandes del directorio y los que están vacíos. Ejecutamos:

```
ls -laS
ls -laSr
```

y obtenemos dos listados ordenados por tamaño, creciente y decreciente.

Orden less: Mostrar un fichero de texto

La orden less muestra un fichero de texto línea a línea, a la vez que permite buscar palabras dentro de él.

Uso:

less [ruta]fichero

Ejemplo:

```
less /etc/hosts
```

Comentarios:

Esta orden no está presente en todos los Unix. Se utiliza con bastante frecuencia para ver el contenido de ficheros de texto que tienen un número de línea mayor que las que caben en la pantalla. Para salir de la visualización de un fichero mediante less utilice ls tecla 'q'. Véanse también las órdenes more, tail y head.

En muchas ocasiones es necesario hojear de forma cómoda algún fichero de configuración o de registro de incidencias y less nos permite hacerlo de una forma bastante cómoda.

Podemos utilizar less para ver como se van añadiendo líneas al final de un fichero en ? tiempo real?; basta pulsar May+F y las nuevas líneas que se agregan al fichero van apareciendo en pantalla. Esto es útil para ver los ficheros de log, registros de incidencias.

Orden mkdir: Crear un directorio

La orden mkdir crea un directorio

Uso:

mkdir [ruta]directorio

Si no ponemos ruta de acceso, mkdir, crea el directorio dentro de directorio actual. Si añadimos la opción -p creará todos los directorios necesarios hasta llegar al último en caso de que no existieran.

Ejemplo:

```
mkdir -p /home/usuario/documentos/textos/shell
```

y crearía un directorio llamado shell y los directorios documentos y textos en caso de que no existieran.

Orden rm: Borrar un fichero o directorio

Elimina ficheros o directorios

Uso:

```
rm [-ir] [ruta]plantilla
```

- -i Pregunta si debe borrarse cada fichero. Si la respuesta no comienza por 'y' o por 'Y' (o quizá el equivalente local, en español 's' o 'S') no se borra.
- -r Borra recursivamente los contenidos de directorios. Borra un directorio y todo su contenido.

Comentarios:

Esta orden es muy peligrosa usada como root y sobre todo con la opción '-r'; un fichero borrado no se puede recuperar. Antes de usar rm es conveniente simular la orden de borrado con otra orden no destructiva como 'ls' para no llevarnos sorpresas desagradables. Por ejemplo si queremos borrar todos los ficheros que terminan en '.bq' podemos hacer:

ls *.bq

y una vez que comprobamos que los ficheros que se van a borrar son los que realmente nos interesan, podremos

```
rm *.bq
```

Es necesario usar esta orden para mantener el sistema aseado, limpio de ficheros viejos que ocupan espacio en el sistema y nos llevan a confusiones, pero es una orden para usarla con precaución, sobre todo si estamos trabajando como root.

Orden cd: Cambia el directorio activo

La orden cd cambia el directorio actual.

Uso:

```
 cd [ruta]directorio
```

Usos especiales de cd

Si no ponemos argumentos,

cd

sin ningún argumento, cambia al directorio personal del usuario.

Si ejecutamos

```
cd ..
```

entonces subimos un nivel de directorios, subimos al directorio superior.

Si ejecutamos

```
Cd -
```

volvemos al directorio activo previo.

Orden mv: Mueve o renombra ficheros

La orden mv mueve un fichero o directorio de un directorio a otro o le cambia el nombre.

Uso:

```
 mv [ruta]origen [ruta]|[destino]
```

Si en origen ponemos una plantilla o varios ficheros como argumento, mv, necesita que destino sea un directorio para mover a él la lista especificada.

Si las rutas origen y destino son iguales lo que hacees cambiar el nombre del fichero.

Si la ruta destino es un directorio mueve el fichero dentro de él con el mismo nombre.

Si especificamos un nombre destino mueve el contenido y cambia el nombre ebn la misma operación.

Todo lo dicho antes para ficheros es igualmente válido para directorios.

Orden cp: Copia ficheros y directorios

La orden cp copia un fichero o directorio en otro fichero y/o directorio distintos.

Uso:

cp [opciones] [ruta]origen [ruta]destino

-i Pregunta si debe sobreescribir cada fichero destino que exista. Si la respuesta no comienza por 'y' o por 'Y' (o quizá el equivalente local, en español 's' o 'S') no se borra.

-r Copia recursivamente los contenidos de directorios.

-a Preserva los atributos del fichero copiado en la medida de lo posible.

cp tiene bastantes más opciones, por lo que se debería consultar la página correspondiente del manual.

Ejemplos:

Queremos copiar el contenido del directorio /home/httpd/html, incluyendo subdirectorios, en /var/www/html

cp -r /home/httpd/html/* /var/www/html

Queremos copiar el contenido del directorio /home/httpd/html, incluyendo subdirectorios, en /var/www/html pero conservando las propiedades de los ficheros, lo permisos:

cp -dPr /home/httpd/html/* /var/www/html

Queremos copiar el contenido del directorio /home/httpd/html, incluyendo subdirectorios, en /var/www/html pero conservando las propiedades de los ficheros, lo permisos y que además pida confirmación para sobreescribir:

cp -ari /home/httpd/html/* /var/www/html

Queremos copiar el contenido del directorio /home/httpd/html, incluyendo subdirectorios, en /var/www/html pero conservando las propiedades de los ficheros, lo permisos y que además no pida confirmación para sobreescribir:

```
cp -arf  /home/httpd/html/* /var/www/html
```

lpr Imprime un fichero

La orden lpr envía un fichero a la impresora.

Uso:

lpr [opciones][ruta]fichero

En el capítulo de impresión de administración sel sistema se ampliará esta orden.

Orden pwd: Imprimir el directorio actual

La orden pwd imprime el directorio actual.

Uso:

```
pwd
```

Esta orden es útil cuando tenemos varias consolas abiertas y cada una de ellas tiene un directorio activo distinto y queremos saber rápidamente cual corresponde a una determinada consola.

Orden ln: Enlaza ficheros o directorios

La orden ln enlaza un fichero o directorio con otro fichero o directorio. Un enlace es una forma de asignar varios nombres a unos datos en disco. Los enlaces pueden ser "duros" que consisten en asignar otro nombre a un mismo *i-nodo*. Los enlaces duros no son válidos para directorios y lógicamente tampoco se pueden establecer enlaces duros para ficheros de distinto sistemas de ficheros ya que tienen que compartir un i-nodo. Los enlaces simbólicos son otro mecanismo más flexible que los enlaces duros. Un enlacee simbólico es un fichero especial que apunta a otro fichero o directorio. Se permiten enlaces simbólicos entre ficheros y también entre directorios.

Uso:

ln [-sfi] [ruta]origen [ruta]enlace

- -i Pregunta si debe sobreescribir cada fichero destino que exista. Si la respuesta no comienza por 'y' o por 'Y' (o quizá el equivalente local, en español 's' o 'S') no se borra.
- -s Crea un enlace simbólico.
- -f Elimina el fichero enlace si existía con anterioridad.

Si no indicamos otra cosa, la orden ln crea un enlace duro que, como hemos indicado anteriormente, consiste en crear una nueva entrada en un directorio que apunta a un *i-nodo*.

Los enlaces duros no se permiten entre ficheros de sistemas de ficheros distintos, ya que todos los enlaces duros apuntan al mismo i-nodo, cosa imposible en dos sistemas de ficheros distintos (dos particiones distintas); el mismo número de i-nodo en dos sistemas de ficheros distintos hace referencia a ficheros distintos.

Si ponemos las opción -s entonces lo que creamos es un enlace simbólico, que consiste en un nuevo fichero que apunta al original, sea fichero o directorio. A diferencia del enlace duro, podemos enlazar ficheros entre diferentes sistemas de ficheros. El enlace simbólico crea un nuevo i-nodo mientras que el enlace duro no.

Ejercicios con solución

Mostrar el contenido de todos los ficheros del directorio personal.

Para mostrar el contenido de un fichero usamos la orden cat.

$ cd cambiamos al directorio personal del usuario. Cada usuario tiene su directorio personal que suele ser /home/usuario, donde usuario se corresponde al nombre que usamos para conectarnos.

$ pwd Nos aseguramos que estamos en él.

$ ls -la Mostramos la lista de ficheros del directorio.

$ cat fichero Mostramos el contenido de cada uno de los ficheros.

Repetimos las líneas anteriores tantas veces como sea necesario. Si por alguna circunstancia mostramos un fichero binario y los caracteres de la pantalla son ilegibles al terminar, debemos teclear reset. Podemos usar las flechas de cursor par buscar líneas lineas anteriores.

Copiar el fichero .profile en otro llamado perso.

$ cp .profile personal

Crear un directorio llamado prueba en nuestro directorio personal.

```
$ cd
```

$ pwd

$ mkdir prueba (Creamos el directorio)

$ ls -la (Verificamos la creación)

Expresar las rutas absoluta y relativa del directorio prueba que acabamos de crear. (Relativa al directorio actual).

/home/usuario/prueba Absoluta

./prueba Relativa

Copiar el fichero /home/usuario/perso en el directorio prueba.

Usando rutas absolutas

```
$ cp /home/usuario/perso /home/usuario/prueba/
```

Usando rutas relativas

```
$ cp /home/usuario/perso ./prueba/
```

o

```
$ cp ./perso /home/usuario/prueba
```

o

```
$ cp ./perso prueba
```

Cambiar al directorio /home/usuario/prueba

Usando rutas absolutas

```
$ cd /home/usuario/prueba
```

Si estamos en /home/usuario

```
$ cd prueba
```

Si no estamos en /home/usuario

```
$ cd
```

$ cd ./prueba

Copiar el fichero perso del directorio /home/usuario/prueba con el nombre perso.nuevo

Para copiar ficheros u directorios usamos la orde cp

(Suponemos que estamos en el directorio prueba, sin ver el ejercicio anterior).

```
$ cp perso perso.nuevo
```

$ ls -la

Estando en el directorio prueba, copiar el fichero .profile en él con el nombre prof.nuevo

```
$ cp ../.profile prof.nuevo
```

$ ls -la

Crear un enlace simbólico llamado pro1, al fichero prof.nuevo.

Para crear un enlace usamos la orden ln

$ ln -s prof.nuevo pro1

Verificar el enlace simbólico pro1 que acabamos de crear

$ ls -la Comprobamos el tipo de fichero

$ cat pro1 Comprobamos el contenido del fichero

Observamos que aparece un nuevo fichero con un tamaño muy pequeño. Sin embargo al mostrar el contenido aparecen los datos que corresponden al fichero al que apunta.

Crear un enlace duro llamado prof.d1, al fichero prof.nuevo.

Para crear un enlace usamos la orden ln

```
$ ln prof.nuevo prof.d1
```

Podemos ahora ejecutar

```
$   ls -lai
```

para que muestre el número de i-nodo de cada fichero y comprobar que es el mismo.

Crear otro enlace enlace duro llamado prof.d2, al fichero prof.d1.

```
$ ln prof.nuevo prof.d2
$ ls -lai
```

Con esta operación estamos creando una nueva entrada al directorio que corresponde a unos datos existentes en disco previamente. Al añadir la opción -i a la orden ls también nos informa del número de i-nodo y podemos observar que son el mismo.

Borrar el fichero prof.d1 y verificar los demás

Para borrar un fichero usamos la orden rm

```
$ rm prof.d1
```

$ ls -lai

Podemos observar como tras borrar prof.d1 seguimos teniendo el fichero prof.d2 y con el mismo i-nodo.

Crear un directorio dentro de prueba llamado src.

Si suponemos que el directorio activo es prueba simplemente ponemos:

```
$ mkdir src
```

$ ls -la

Con esto último verificamos la creación.

Si el directorio activo hubiera sido el directorio personal, es decir /home/usuario entonces deberíamos haber puesto:

```
$ mkdir prueba/src
```

$ ls -la

Realizar las siguientes operaciones

Crear un enlace simbólico llamado fuente, al directorio src.

$ ln -s src fuente

Copiar el fichero perso.nuevo, al directorio src.

```
$ cp perso.nuevo src
```

Copiar el fichero pro1, al directorio fuentes.

```
$ cp pro1 fuentes
```

Verificar los contenidos de los directorios src y fuentes.

```
$ ls -la fuentes
$ ls -la src
```

Cambiar al directorio superior.

```
$ cd ..
```

Cambiar directamente al directorio fuentes.

```
$ cd pruebas/fuentes
```

Borrar el directorio src.

```
$ cd ..
$ rm -r src
```

Verificar el estado del enlace simbólico fuentes. Borrar el enlace.

```
$ ls -la fuentes
```

Observamos como la orden ls -la tiene ahora un significado distinto a cuando existía el directorio src. Observa el error que se produce cuando hacemos cat fuentes.

```
$ rm fuentes
```

para borrarlo

Copiar el fichero /home/usuario/perso en el fichero nuevo.

```
$ cd
$ cp perso nuevo
```

Cambiar el nombre del fichero nuevo a viejo.

```
$ mv nuevo viejo
$ ls -la
```

Crear un directorio llamado practica en el directorio personal del usuario.

```
$ cd
```

$ mkdir practica

Mover el fichero viejo al directorio practica

```
$ mv viejo practica
```

Mostrar la lista de usuarios del sistema

```
$ cat /etc/passwd
```

Mostrar la lista de grupos del sistema

```
$ cat /etc/group
```

Copiar el fichero /etc/hosts en el directorio personal

```
$ cp /etc/hosts .
```

Mostrar el fichero hosts del directorio personal

$ cat hosts .

Mostrar el contenido del directorio /tmp. Observar permisos y propietarios.

```
$ ls -la /tmp
```

Copiar el fichero /etc/inittab en el directorio personal

```
$ cp /etc/inittab .
```

Cambiar de nombre al fichero inittab del directorio personal y llamarlo tabla.inicio

```
$ mv inittab tabla.inicio
```

Crear un directorio llamado programas y dentro de él, otros cuatro directorios llamados src, lib, bin y include

```
$ mkdir programas
$ mkdir programas/src
$ mkdir programas/lib
```

```
$ mkdir programas/bin
$ mkdir programas/include
```

Cambiar al directorio src

```
$ cd programas/src
```

Cambiar al directorio superior

```
$ cd ..
```

Crear un directorio llamado config en el directorio personal

```
$ cd
```

$ mkdir config

Copiar en directorio config todos los ficheros de /etc que empiecen por s.

```
$ cp /etc/s* ./config
```

Mostrar el contenido del directorio config

```
$ ls -la ./config
```

Mostrar el contenido del directorio config

```
$ ls -la ./config
```

Borrar el contenido del directorio config

```
$ rm ./config/*
```

Borrar el directorio config

```
$ rm -r ./config
```

Crear un enlace simbólico con el directorio /tmp llamado temp.

```
$ ln -s /tmp temp
```

Ejecutar cd ?/programas y verificar el directorio actual.

```
$ cd ?/programas
```

```
$ pwd
```

Ejecutar cd ?/programas/bin y verificar el directorio actual.

```
$ cd ?/programas/bin
$ pwd
```

Podemos observar como el símbolo ? hace referencia al directorio personal del usuario.

Comprimir, descomprimir y agrupar ficheros

Comprimir ficheros y uso de ficheros compromidos

La forma habitual de transmitir y almacenar información es hacerlo con ficheros comprimidos. Se ahorra ancho de banda en las transmisiones y en el almacenamiento. Los mecanismos de compresión son distintos, y en muchos casos incompatibles entre sí, pero Linux dispone de utilidades para gestionar todos ellos.

gzip

La orden gzip comprime un fichero y es un estándar en sistemas Linux. Su uso es muy simple:

```
$ gzip fichero
```

y se crea un fichero llamado fichero.gz comprimido.

Esta orden normalmente se instala en todos los sistemas Linux

gunzip

Se utiliza para descomprimir un fichero comprimido con gzip. Su uso es el lógico:

```
$ gunzip fichero.gz
```

Sí hay que tener en cuenta que al descomprimir el fichero borra el original comprimido.

zcat

La orden zcat es idéntica a cat, salvo que trabaja directamente con ficheros comprimidos con gzip.

Por ejemplo, podemos comprimir un fichero

```
gzip fichero.txt
```

y obtenemos fichero.txt.gz

Ahora podemos hacer:

```
zcat fichero.txt.gz
```

y lo mostraría en pantalla normalmente.

zless

Esta orden es idéntica a less salvo que trabaja directamente con ficheros comprimidos con gzip.

Siguiendo el ejemplo de zcat también podríamos poner:

```
zless fichero.txt.gz
```

bzip2

Esta orden comprime ficheros. Su uso es idéntico a gzip, pero es algo más efectiva que gzip, genera ficheros comprimidos algo más pequeños. Esta orden genera fichreos con extensión .bz2.

Este compresor puede que no se instale con Linux si no se indica explícitamente.

bunzip2

Esta es la orden que nos permite descomprimir ficheros comprimidos con bzip2. Los ficheros comprimidos com bzip2 tienen la extensión .bz2.

zip

Esta es la orden se utiliza para comprimir ficheros en formato zip. El uso es ligaramente

distinto a las órdenes anteriores:

```
$ zip ficheros-comprimido lista_ficheros
```

Es decir, primero ponemos el nombre del fichero comprimido y luego la lista de ficheros que queremos comprimir. En la lista de ficheros podemos incluir directorios.

Los ficheros comprimidos con zip tienen extensión .zip.

Este compresor puede que no se instale con Linux si no se indica explícitamente.

unzip

Esta orden descomprime ficheros previamente comprimidos con zip.

Agrupar y desagrupar ficheros: tar

En las órdenes anteriores hemos visto como comprimir ficheros individuales, pero en muchas ocasiones nos interesa agrupar en un archivo diferentes ficheros, por ejemplo todos los contenidos en un directorio.

También es habitual en linux distribuir los programas con todos sus ficheros agrupados y comprimido, por lo que será necesario saber como descomprimir y desagrupar los ficheros.

Estas operaciones las realiza la orden tar (tape archive); las extensiones que utiliza esta orden para indicar un archivo de ficheros es .tar.

Agrupar ficheros

Para agrupar una serie de ficheros en un archivo pondremos:

```
$ tar cf archivo.tar lista de ficheros
```

Si en lugar de poner ?lista de ficheros? ponemos un directorio, agrupará todos los ficheros contenidos en el directorio y conservando la estructura del subárbol de directorios, los propietarios de los ficheros y sus permisos.

La orden ejecutada de esta forma simplemente comprime los ficheros, pero sin realizar ningún tipo de compresión.

Por ejemplo, para hacer una copia de todo el fichero /etc llamda etc.tar podríamos:

```
tar cf etc.tar /etc
```

Agrupar ficheros y comprimir

Una posibildad mucho más interesane es agrupar una serie de ficheros en un archivo y comprimirlo con gzip simultáneamente. Podemos combinar la agrupación de tar con un sistema de compresión. Para hacer esto pondremos:

```
$ tar cfz archivo.tar.gz lista de ficheros
```

Por ejemplo, para hacer una copia de seguridad comprimida del directorio /etc podríamos:

```
$ tar cfz etc.tar.gz /etc
```

Para agrupar una serie de ficheros en un archivo y comprimirlo con bzip2 simultáneamente pondremos:

```
$ tar cfj archivo.tar.bz2 lista de ficheros.
```

Como veíamos antes, si en lugar de poner ?lista de ficheros? ponemos un directorio agrupará todos los ficheros contenidos en el directorio y conservando la estructura del subárbol de directorios, permisos y propietarios en el fichero comprimido.

Desagrupar ficheros

Para desagrupar una serie de ficheros en un archivo pondremos:

```
$ tar xvf archivo.tar
```

Si el archivo contiene directorios, al desagruparlo estos directorios se crean en el directorio activo. Por ejemplo, agrupamos el directorio /etc como veíamos antes:

```
tar cf etc.tar /etc
```

Ahora si nuestro directorio activo es /home/usuario y ejecutamos:

```
tar xvf etc.tar
```

creará el directorio /home/usuario/etc y alí guardará el resto del contenido del archivo. Resumiendo , al agrupar se elimina el directorio raíz de la ruta de los ficheros contendos por lo que siempre resultan rutas relativas al directorio activo.

Desagrupar ficheros y descomprimir

El siguiente paso lógico es manipular los ficheros agupados y comprimidos (los llamare tar.gz). Igual que creábamosun fichero tar.gz en una sola operación, ahora también vamos a extraer su contenido en una sola operación.

Para desagrupar una serie de ficheros en un archivo comprimido con gzip pondremos:

$ tar xvfz archivo.tar.gz

Si el archivo estuviera comprimido con bzip2 pondríamos:

$ tar xvfl archivo.tar.bz2

En ambos casos podríamos descomprimir el fichero con gunzip o con bunzip2, según el caso y luego desagrupar el archivo no comprimido como se indicaba más arriba.

Otras opciones de tar

tar tiene además otras opciones que resultan interesantes, por ejemplo:

-l Un solo sistema de ficheros. Si tenemos varias particiones y queremos copiar un directorio per no las particiones que tuvier montadas en su interior tendríamos que usar la opción -l. Por ejemplo, para hacer una copia de seguridad de la partición raíz excluyendo el resto de particiones y dispositivos pondríamos:

```
tar cflz raiz.tar /
```

-k No borra los ficheros que existan previos a la extracción. Puede resultar útil para añadir ciertos ficheros que puedieran faltar en un árbol de directorios pero sin borrar ninguno de los que ya existe.

Por ejemplo, estamos trabajando con unos ficheros de texto en un directorio del que tenemos una copia de seguridad comprimida. Si hemos borrado un accidentalmente podremos reponerlo con tar xvfkz sin que afecte al resto de los ficheros.

Ejercicios resueltos

Hacer una copia de seguridad del directorio /home

Para mayor comodidad, cambiamos al directorio donde queramos guardar la copia y posteriormente ejecutamos la orden:

```
cd copias
tar cfz home.tar.gz /home
```

Como práctica copia tres tipos de ficheros, de texto, una imagen y un programa y comprímelos usando los tres programas y determina cual tiene mayor tasa de compresión para cada tipo.

Otras órdenes de usuario

Orden id: Información sobre el usuario

La orden id imprime los identificadores de grupo y usuario reales del usuario. Esta orden dispone de diversos argumentos que se pueden consultar en la correspondiente página del manual.

Uso:

```
id [-gGnru] [usuario]
```

-g muestra sólo la información del grupo efectivo

-G muestra todos lo grupos

-n muestr ael nombre ne luta del Id donde proceda

-r muestra el ID read en lugar del efectivo

-u muestra sólo el ID de usuario efectivo

Si indicamos un nombre de usuario entonces muestra su información, si no muestra la del usuario conectado.

Ejemplo:

```
$ id
```

uid=500(fabrega) gid=500(fabrega)
groups=500(fabrega),4(adm),100(users),505(usuwin),511(fax)

Orden passwd: Modifica la clave

Permite a un usuario modificar su clave simplemente ejecutándola. El root puede ejecutar

esta orden añadiendo como argumento el nombre de un usuario para cambiar su clave. Los cambios de clave implican modificaciones del fichero /etc/passwd (o /etc/shadow en su caso).

Uso:

```
passwd [usuario]
```

Un usuario sólo puede modifcar su contraseña, sólo root puede modificar contraseñas de otros usuarios.

Orden man: obtener información

La orden man proporciona información sobre sobre una orden, programa o función del lenguaje C. La información se guarda en el directorio /usr/man. Hay una serie de variables de entorno que permiten modificar el comportamiento de la orden man: La variable LANG define el idioma en que suministra la información. En nuestro caso nos interesará poner LANG=es. Si una página no existe en el idioma, se suministra en inglés. Para que aparezcan acentos y eñes tendremos que usar la variable LESSCHARSET asignándole, por ejemplo, el valor latin1.

Uso:

```
man [sección] orden
```

Ejemplos:

```
$ man ls
 $ man 2 mount
 $ man mount
```

Orden who: información sobre usuarios conectados

La orden who proporciona información sobre los usuarios conectados al sistema en un momento dado. Esta orden dispone de diversos argumentos que se pueden consultar en la correspondiente página del manual.

Uso:

who

Por ejemplo, tendremos:

```
$ who
fabrega  tty1     Ene 19 14:42
```

root tty2 Ene 30 20:17

orden whoami: información sobre el usuario

Esta orden es equivalente a id -un.

Esta orden es útil cuando estamos trabajando con varios usuarios simultáneamente y queremos saber en un momento dado cual es el que estamos utilizando.

```
$ whoami
fabrega
```

Orden write: envía un mensaje a un usuario

La orden write envía un mensaje a un usuario conectado a la misma máquina en otro terminal distinto.

Uso:

```
 write usuario  [tty]
```

donde tty indica el terminal donde está conectado el usuario. Los nombres de los terminales los podemos obtener utilizando la orden who.

orden mesg: activa/desactiva la recepción de mensajes

Con esta orden podemos impedir que otro usuario escriba en el terminal que estamos

utilizando. En particular desactiva los mensajes enviados con write. Observe como se modifican los permisos del terminal correspondiente en /dev/.

Uso:

```
mesg y|n
```

orden mail: envía un mensaje de correo electrónico

La orden mail se utiliza para enviar mensajes de correo electrónico (simples) desde una consola.

Uso:

mail usuario[(EN)maquina]

Además esta orden también se utiliza para leer el correo que se recibe, en este caso llamándola como mail.

Véase también mailx. Para obtener mas detalles consulte la correspondiente página del manual.

Orden date: muestra las hora y fecha actuales

La orden date se utiliza para mostrar la hora de sistema con un determinado formato o bien pra fijarla.

Uso:

```
date [opciones]   [+formato]
date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]
```

donde "formato" es una cadena de caracteres donde se pueden incluir caracteres fijos que aparecerán tal cual los ponemols y otros especiales, precedidos por % que se sustituyen por un valor de la fecha u hora. Los valores que podemos usar como formato y sus equivalencias son:

Carácter	Significado	Carácter	Significado

		r	
%%	Un % literal	%a	Día de la semana abreviado y según locales (lun,mar,...)
%A	Día de la semana según locales (lunes, martes, ...)	%b	Mes abreviado según locales (ene, feb,...)
%B	Mes según locales (enero, febrero,...)	%c	Hora y fecha según locales(sab 04 nov 1998 19:22:45 CET)
%C	Dígitos de la centruria (00-99)	%d	Día del mes (01-31)
%D	fecha en formato mm/dd/aa	%e	Día del mes rellenando ceros a la izquierda con espacios (1-31)
%F	Equivalente a %Y-%m-%d	%H	Hora de 00-23
%I	Hora de 01-12	%j	Día del año 001-365
%k	Hora 0-23	%l	hora 1-12
%m	Mes 01-12	%M	minuto 00-59
%n	Nueva línea	%s	timestamp (segundos desde 1/1/1970)
%S	Segundos 00-59	%u	Dia de la semana 1=lunes
%y	Dos últimos dígitos del año	%Y	Cuatro dígitos del año

Normalmente los valores numéricos se rellenan con ceros a la izquierda, pero podemos añadir modificadores para sustituir estos ceros por espacios o eliminarlos:

- elimina los ceros a la izquierda

_ sustituye los ceros a la izquierda por espacios.

Por ejemplo:

```
$ date +%d
07
$ date +%_d
 7
$ date +%-d
7
```

Ejemplo:

```
$ date
mié ene  6 19:53:26 CEST 1999
```

Como djimos, la cadena de formato está constituida por caracteres fijos que aparecen tal cual se ponen y otros caracteres precedidos por el carácter % que se sustituyen por el valor correspondiente de la fecha. La cadena de formato va precedida de un signo +.

Por ejemplo:

```
$ date "+%Y%m%d"
20011207
$ date "+ año %y mes %m dia %d"
 año 01 mes 12 dia 07
```

Para obtener mas detalles consulte la correspondiente página del manual.

Comentario: Esta orden es interesante para generar nombres de ficheros que contengan la fecha actual, para guardar copias de seguridad, por ejemplo.

Orden echo: muestra en pantalla el resto de la línea

La orden echo se utiliza para mostrar una cadena de caracteres en pantalla.

Uso:

```
 echo  [-ne] [cadena ...]
```

-n Elimina los retornos de carro finales

-e Usa el carácter de barra invertida (\ ) como carácter de escape. Es decir "\ n" se considera como un salto de línea.

La orden echo interpreta cualquier cadena que comience por «$» como una variable. Las variables se expanden cuando la cadena va encerrada entre comillas dobles (""). Todo lo que vaya comprendido entre comillas simples se interpreta tal cual. También podemos imprimir un carácter $ protegiéndolo con el carácter de escape, que vimos anteriormente (\ ). (Véanse los metacaracteres de la shell).

La verdadera importancia de la orden echo la encontraremos en el desarrollo de programas de shell.

orden sort: ordenar el contenido de un fichero

La orden sort permite ordenar uno o varios ficheros y mostrar el resultado en la salida estándar.

Uso:

sort [opciones] ficheros

Las opciones más importantes:

Opción	Significado	Opción	Significado
-b	Ignora los espacios iniciales	-d	Considera sólo letras y números
-f	No distingue May y Min	-g	Ordenación numérica
-M	Ordenación por meses 'JAN' < 'DEC'	-r	Ordenación inversa
-c	Comprueba si está ordenado	-k p1[,p2]	Ordena desde p1 hasta p2
-m	une varios ficheros ordenados	-n	Cadenas como números

Ejemplo:

Para mostrar ordenado el fichero /etc/services, pondríamos:

```
sort /etc/services
```

Orden more: muestra un fichero en pantalla

La orden more se utiliza para mostrar en fichero pantalla a pantalla cuando su contenido sobrepasa los límites de visualización.

Uso:

more fichero

Esta orden es muy parecida a less

Orden cal: muestra un calendario

Uso:

cal [mes] año

Por ejemplo, tendremos la siguiente salida

```
$ cal 1 99
      enero 99
do lu ma mi ju vi sá
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

Orden expr: evalúa una expresión entera

La orden expr evalúa una expresión entera y la muestra. Si la expresión es lógica devuelve un 1 en caso de ser verdadera y un 0 en caso de ser falsa.

Uso:

expr expresión

Los operadores que admite la orden expr son los siguientes:

Aritméticos *, +, -, /, %

Comparar cadenas :

Lógicos <, <=, =, !=, >=, >

And y Or &, |

Hay que tener en cuenta que ciertos caracteres tienen un significado especial para la shell; cuando los usemos tendremos que protegerlos con el carácter de escape \ .

Ejemplos:

```
$ expr 20 + 15
$ expr 30 * 6
$ expr 29 % 3
$ expr 101 / 7
$ expr 5  \< 8
```

$ expr 5 \> 8

Orden diff: muestra diferencias entre ficheros

La orden diff muestra la diferencia entre dos ficheros.

Uso:

diff [opciones] fichero-origen fichero-destino

Los argumentos también pueden ser dos directorios, en cuyo caso compara sus contenidos. Consultar la página del manual para tener más información.

Cuando comparamos dos ficheros podemos utilzar la salida para crear un fichero de parche para aplicar a otros ficheros idénticos al original.

find Localiza ficheros

La orden find se utiliza para localizar ficheros contenidos en los distintos discos del sistema. Esta orden dispone de muchas opciones por lo que es conveniente consultar la página del manual

Uso:

find [ruta] [expresión]

ruta indica un directorio donde comenzamos la búsqueda.

Entre las opciones disponibles de find podemos citar:

Opción	Significado	Opción	Significado
		-type tipo	tipo puede ser b, c d o f para referirnos a dispositivos de bloque (b), carácter (c), directorio (d) o fichero regular (f).
-exec orden	ejecuta una orden para cada fichero encontrado.	-user usuario	busca ficheros propiedad del usuario.
-group grupo	igual que user pero para el grupo.	-regex expreg	Ficheros que verifiquen la expresión regular indicada.
-depth	Procesa antes el contenido de un directorio antes que el propio directorio.	-maxdepth nivel	Indica el número de directorios que desciende en la búsqueda.
-mindepth nivel	No aplica acciones en niveles inferiores al indicado	-empty	Fichero o directorio vacío
-mount o -xdev	Sólo un sistema de ficheros	-newer fich	Fichero modificado con posterioridad al fichero indicado.
-perm modo	Busca elementos con esos permisos. Los permisos se considera desde un cero. Si modo va prrecedido de - se exigen todos lso permisos. Si va prpecedidio de un + entonces se exije alguno.	-name expr	indica el nombre de fichero que queremos buscar. Puede ser un nombre fijo o una plantilla formada por * y ?, en este último caso, la plantilla deberá estar comprendida entre comillas dobles para evitar que la shell las interprete antes de llamar a la orden find
-uid n	Ficheros con uid n	-user user	Ficheros propiedad de user

-gid n	Ficheros con gid n	-group grp	Ficheros del grupo grp
-nouser	Ficheos sin usuario	-nogroup	Ficheros sin grupo
-amin n	Accedico hace n minutos	-a newer fic	Último acceso posterior a fic
-atime n	Accedico hace 24*n horas	-cmin n	Cambio estado hace n minutos
-cnewer file	Cambio estado posterior a file	-ctime n	Cambio estado hace 24*n horas
-mtime n	Contenido modificado hace 24*n horas	-used n	Último acceso hace n días
( )	Subexpresión	! ( ) -not ( )	Niega la subexpresión
expr1 -a expr2	Las dos expresiones se verifican (and)	expr1 -o expr2	Se verifica una de las dos

Los valores numéricos se pueden especificar como:

+n mayor que n

-n menor que n

n igual a n

La orden find es muy útil para localizar ficheros con características muy concretas, por ejemplo todo tipo de permisos (opción -perm) modificados en los últimos días (-mtime), accedidos en los últimos días (-atime) etc. Es muy útil sobre todo en aspectos de seguridad del sistema.

Ejemplos de uso:

Borrar todos los ficheros terminados en ??? del directorio activo:

```
find . -name ?*~? -type f -exec rm {} \;
```

Buscar todos los ficheros del directorio activo modificados con posterioridad al fichero datos.txt:

```
find . -type f -newer datos.txt
```

Buscar todos los ficheros llamados .profile que haya a partir del directorio activo:

$ find . -name .profile

Borrar todos los ficheros llamados core:

```
$ find / -name core -type f -exec rm {} \;
```

Buscar todos los ficheros propiedad de root o de pepe

```
$ find /tmp  \(-user root -o -user pepe \)
```

Buscar todos los ficheros que empiecen por print

```
$ find / -name "print*"
```

Buscar en el directorio /var todos los directorios más recientes que /etc/passwd

```
$find /var  -type d -newer /etc/passwd
```

Orden ps: muestra lista de procesos

Uso:

ps [axu...]

La orden ps muestra la lista de procesos del sistema y algunas de sus características, dependiendo de las opciones que le añadamos. Entre las múltiples opciones disponibles vamos a citar:

-a Muestra también los procesos de otros usuarios

-x muestra procesos que no están controlados por ninguna terminal

-u formato usuario: muestra el usuario y la hora de inicio Pero no sólo estas opciones están disponibles, también hay otras que pueden ser muy útiles en ciertas circunstancias. Es conveniente consultar la página del manual.

Este sería un ejemplo, sin opciones:

```
$ ps
  PID TTY          TIME CMD
26861 ttyp4    00:00:00 bash
 4767 ttyp4    00:00:00 ps
```

Otro ejemplo podría ser:

```
$ ps x
  PID TTY      STAT   TIME COMMAND
  686 tty1     S      0:00 -bash
26812 tty1     S      0:00 sh /usr/X11R6/bin/stkde -- -bpp 32 -cc 4
26813 tty1     S      0:00 xinit /etc/X11/xinit/xinitrc.kde -- -bpp 32 -cc 4
26817 tty1     S      0:15 kwm
26836 tty1     S      0:00 kaudioserver
26837 tty1     S      0:00 kwmsound
26838 tty1     S      0:26 kfm
26839 tty1     S      0:00 krootwm
```

26840 tty1 S 0:26 kpanel

La orden ps proporciona una información muy interesante sobre los procesos que tenemos en ejecución. Podemos saber el pid del proceso qué programa o originó el proceso, cuanta memoria ocupa, cuanta CPU consume, cuanto tiempo de ejecución lleva, ...

Orden sleep: genera un proceso durante cierto tiempo

La orden sleep genera un proceso que dura el tiempo especificado. El tiempo se puede especificar en segundos (s), que es el valor que se toma si se omite la letra. También en minutos (m), horas (h) o días (d).

Uso:

sleep numero[smhd]

Esta orden puede parecer una tontería, pero se puede combinar con otras órdenes para demorar cierto tiempo el comienzo de su ejecución. Ejemplo:

sleep 2h; ppp-off

Orden stty: parámetros del terminal

Uso:

```
stty [valores...]
```

stty toma una serie de argumentos que modifican las características del terminal. Si añadimos un `[-]' antes del argumento, la característica se deshabilita, si no se habilita.

Por ejemplo:

```
$ stty sane
```

repondría en el terminal sus valores originales.

Orden head: muestra las primeras línea de un fichero

Uso:

head [opciones...] fichero

La orden head nos permite ver las primeras líneas de un fichero, y nos puede ser útil para identificar ficheros por su contenido.

Por ejemplo, queremos saber la versión del fichero /etc/sendmail.cf y sabemos que está en las tres primeras líneas. Entonces pondríamos:

$ head -3 /etc/sendmail.cf

tail: Muestra las últimas línea de un fichero

Uso:

tail [opciones...] fichero

Por ejemplo, para mostrar las dos últimas línea del fichero /etc/group pondríamos:

```
$ tail -2 /etc/group
```

La orden tail nos permite ver las primeras líneas de un fichero, y nos puede ser útil para ir viendo las incidencias que se anotan en un fichero de registro. Por ejemplo, es frecuente en ciertas situaciones que el administrador tenga una consola virtual para ejecutar la orden:

$ tail -f /var/log/messages

Con la opción -f hacemos que tail muestre las líneas según vayan escribiendo en el fichero /var/log/messages. Para cancelar la ejecución de esta orden (con la opción -f) tendremos que pulsar Ctrl C.

Comentario:

Esta orden se utiliza con bastante frecuencia para ir mostrando en pantalla ficheros de registros de incidencias par ir viendo en el momento lo que ocurre con el sistema.

Orden touch: actualiza las fechas de un fichero

Uso:

touch [opciones...] fichero

La orden touch modifica las fechas de acceso y modificación de un fichero. En el caso de que el fichero no exista, la orden touch crea un fichero vacío.

touch dispone de diversas opciones. Para verlas, consultar la página del manual.

Muchos programas que sólo pueden tener una copia ejecutándose en el sistema utilizan touch para crear un fichero que indica que el proceso está activo; antes de terminar la ejecución se debe borrar este fichero.

Orden tty: muestra el terminal

Uso:

tty

La orden tty muestra la terminal actualmente conectada a la salida estándar.

Orden wc: cuenta información sobre ficheros

Uso:

wc [opciones...] fichero

La orden wc cuenta, sobre un fichero de texto, líneas, caracteres y palabras. No nos confundamos, wc significa word count

Si no ponemos opciones, wc contará líneas, palabras y caracteres. Si queremos limitar las operaciones de wc disponemos de las siguientes opciones:

-c Cuenta caracteres

-w Cuenta palabras

-l Cuenta líneas

Ejemplos

Que un usuario modifique su contraseña de acceso

$ passwd

La orden passwd modifica la contraseña del usuario que la ejecuta. El administrador puede usarla poniendo a continuación un nombre de usuario para modificar las contraseñas de otras cuentas del sistema.

Averiguar los usuarios que están en este momento conectados al sistema.

$ who

Averiguar qué usuario tengo activo en una determinada consola o terminal.

Para resolver esta cuestión nos podemos basar en el caso anterior. La orden who también muestra la consola junto al nombre de usuario, por lo que bastaría poner:

$ who

Enviar un mensaje al usuario juan que está actualmente conectado.

$ write juan "Dame la dirección de corre de Antonio, please"

La orden write sólo es válida cuando el otro usuario está conectado al sistema. Si quisiéramos enviar mensajes a usuarios que no están conectados tendríamos que usar el correo electrónico.

Desactivar la recepción de mensaje en nuestra terminal

```
$ mesg n
```

La orden mesg básicamente lo que hace es modificar los permisos del correspondiente dispositivo /dev/ttyN

Mostrar las fecha y hora actuales

$ date

Mostrar ordenados los ficheros /etc/passwd y /etc/group

```
$ sort /etc/passwd
```

$ sort /etc/group

Mostrar el fichero /etc/services pantalla a pantalla

$ more /etc/services

Mostrar el calendario del 1 de enero de 1999

$ cal 1 1999

Calcular las siguientes operaciones:

110+38, resto de 100/7, si 3<8, si 8<3 y si 8>3

Puestas en el mismo orden quedarían:

```
expr 110 + 38
 expr 100 % 7
 expr 3  \  < 8
 expr 8  \  < 3
```

expr 8 \ > 3

Aquí podemos observar como hemos incluido el carácter \ para proteger los símbolos < y >. Estos dos símbolo tienen un significado propio, por lo que es necesario "protegerlos" para evitar que tengan este comportamiento (redirigir las entrada y salida estándares). También hay que tener en cuenta los espacios.

Localizar los ficheros resolv.conf, profile, hosts.allow y sendmail.cf.

```
$ find / -name resolv.conf -type f
 $ find / -name profile -type f
 $ find / -name hosts.allow
 $ find / -name sendmail.cf -type f
```

Localizar todos los directorios llamados usr y bin.

```
$ find / -name usr -type d
```

$ find / -name bin -type d

Localizar el fichero llamado aliases y mostrarlo en pantalla.

$ find /etc -name aliases -type f -exec cat {} \ ;

Mostrar la lista de procesos propiedad del usuario que ejecute la orden.

$ ps

Mostrar la lista de todos los procesos activos en la máquina, indicando el usuario que es su propietario.

$ ps axu

Poner a dormir un proceso durante 5 segundos.

$ sleep 5

Mostrar las cinco primeras líneas del fichero /etc/inetd.conf.

$ head -n 5 /etc/inetd.conf

Mostrar las últimas líneas del fichero /etc/inittab.

```
$ tail /etc/inittab
```

Crear un fichero vacío llamado nuevo en nuestro directorio personal.

```
$ cd
```

$ touch nuevo

Contar las líneas, palabras y caracteres que contiene el fichero /etc/passwd.

```
$ wc -l /etc/passwd
 $ wc -w /etc/passwd
```

$ wc -c /etc/passwd

Gestión de procesos

introducción

Como ya debemos saber, un proceso es un programa en ejecución con recursos asignados. También sabemos que un proceso puede tener distintos estados que se pueden clasificar según distintos criterios, algunos de los cuales no son incompatibles entre sí. Como ahora lo que nos interesa es el aspecto puramente práctico, vamos a distinguir tres estados:

- Primer plano: Un proceso que se ejecuta bloqueando para él la terminal desde la que e lanzó. Un proceso se lanza en primer planos simplemente introduciendo su nombre (y la ruta de acceso si fuera necesario) en el indicador de la línea de órdenes y pulsando intro.
- Segundo plano: Un proceso que se ejecuta sin bloquear la terminal, aunque sí puede escribir en ella los resultados de su ejecución. Un proceso se lanza en segundo plano poniendo al final de la línea de órdenes el símbolo & separado por al menos un espacio del nombre del programa.
- Detenido: Podemos detener un proceso y que se quede en espera en el sistema hasta que demos la orden para que continúe su ejecución. En el presente texto nos vamos a referir a procesos detenidos por orden directa del usuario. No vamos a hacer referencia a procesos suspendidos por causas internas del sistema operativo. En general disponemos de la orden ps que nos proporciona información sobre los procesos, como por ejemplo hora de inicio, uso de memoria, estado de ejecución, propietario y otros detalles.

Además tenemos que tener en cuenta otra característica: cada proceso tiene un propietario que generalmente es el usuario que lo ejecuta. Además al proceso se le aplican los mismos permisos que tenga el usuario propietario, es decir el proceso sólo podrá acceder a la información a la que pueda acceder el propio usuario con sus permisos.

Operaciones con procesos

Operaciones con procesos en primer plano

Con un proceso en primer plano podemos realizar dos acciones desde la terminal que tiene asociada.

Matar el proceso: C-c (Ctrl-c) Sólo podremos matar procesos sobre los que tengamos permiso. Si intentamos matar el proceso de otro usuario el sistema no nos lo permitirá, salvo a root.

Parar el proceso: C-z (Ctrl-z)

En el primer caso se cancela el proceso y se liberan todos los recursos que tuviera asignados. En el segundo caso sólo se detiene la ejecución del proceso, conservando su estado y sus recursos para poder continuar en el momento que se dé la orden adecuada.

Ejemplo: Vamos a lanzar un proceso que dura 20 segundos y a continuación lo vamos a matar:

En primer lugar vemos la lista de procesos.

```
$ ps
```

Creamos el proceso y observamos como la terminal se queda bloqueada durante la ejecución.

```
$ sleep 20
```

Ctlr-c Matamos el proceso y observamos como se devuelve el control de la terminal y aparece el indicador de la línea de órdenes.

Y ahora observamos como los procesos que aparecen son los mismos que al principio.

```
$ps
```

Ahora vamos a repetir la operación, pero deteniendo el proceso

En primer lugar vemos la lista de procesos.

```
$ps
```

Creamos el proceso y observamos como la terminal se queda bloqueada durante la ejecución.

```
$ sleep 20
```

C-z (Control-z) Detenemos el proceso y observamos como se devuelve el control de la terminal y aparece el indicador de la línea de órdenes.

Y ahora observamos como entre los procesos aparece ahora sleep 20.

```
$ps
```

Procesos en segundo plano

Hasta ahora hemos visto una serie de pasos para poder enviar un proceso para que continúe su ejecución en segundo plano; le enviamos una señal para que se detenga y luego otra para que continúe su ejecución. Pero este proceso es incómodo, tenemos una forma más simple de enviar un programa para que se ejecute directamente en segundo plano. Esto es con el operador & tras el nombre del programa y su argumentos. Un ejemplo simple de esta acción sería:

```
$ sleep 200 &
 $ ps
```

Al lanzar el proceso, nos aparece por pantalla algo parecido a:

```
[1]   2 035
```

donde: [1] es el número de trabajo

2035 es el número de proceso.

Podemos observar como el indicador de la línea de órdenes no aparece inmediatamente. Sin embargo esto no supone que la terminal de salida no sea desde la que se lanzó el proceso; es decir, el proceso seguirá mostrando su salida en la pantalla desde la que dimos la orden de ejecución.

Ejemplos:

```
$ sleep 200 &
 $ sleep 201 &
 $ ps
```

y podemos observar como los procesos continúan activos.

```
$ sleep 10 &
```

Ahora esperamos 10 segundos y el sistema nos avisa que ha terminado.

```
[3]+  Done                    sleep 10
```

Orden kill: envío de señales a procesos

Las órdenes kill y killall se utilizan para enviar señales a un proceso. Esta orden la podemos usar de la siguiente forma:

```
$ kill -señal lista_pid_proceso
```

donde el valor de lista_pid_proceso tendremos que averiguarlos utilizando la orden ps. En cuanto a señal puede ser bien su valor numérico o bien su valor simbólico, omitiendo el SIG inicial.

Las señales más habituales son:

SIGHUP (1) Colgar. Ciertos procesos utilizan la señal SIGHUP para indicar al proceso que relea su fichero de configuración. Este es el caso, por ejemplo, de los procesos init y inetd.

SIGCONT (18) Continuar. Cuando un proceso detenido recibe esta señal continúa su ejecución.

SIGSTOP (19) Cuando un proceso en estado preparado recibe esta señal se detiene.

SIGINT (2) Termina un proceso y desaparece.

SIGTERM (15) Mata un proceso de forma controlada.

SIGKILL (9) Esta señal mata un proceso incondicionalmente. Para enviar una señal a un proceso disponemos de dos órdenes que realizan esa labor:

Por ejemplo:

$ kill -9 337

o lo que sería lo mismo

$ kill -KILL 337

Orden killall

Esta orden es ligeramente diferente a la orden kill por dos motivos; en primer lugar utiliza el nombre de proceso en lugar del pid, y además le envía la señal a todos los procesos que tengan el mismo nombre. Es decir:

```
$ killall -señal nombre_proceso
```

Por lo demás, su comportamiento es idéntico, por lo que serían equivalentes

```
$ kill -HUP 1
```

$ killall -HUP init

al haber un único proceso init, con pid igual a 1.

También hay que tener en cuenta que todos los procesos no están preparados para recibir todas las señales. Algunas señales siguen un tratamiento estándar, otras las gestiona el propio proceso y otras son simplemente ignoradas.

Ejemplos:

$ ps

```
$ sleep 100 (pulsar C-z)
 $ ps
 $ kill -CONT pid    # pid se sustituye por el valor obtenido en ps    (pulsar C-z)
 $ kill -9 pid
 $ ps     $ killall -HUP init
```

y ahora se puede repetir lo mismo con killall. Por ejemplo:

```
$ ps
 $ sleep 100
 $ C-z
 $ ps
 $ killall -CONT pid
 $ C-z
 $ killall -9 pid
 $ ps
```

Gestión de trabajos

Cuando lanzamos un proceso en segundo plano obtenemos un PID y un número de trabajo. El PID es el número de proceso y es es método que utiliza el sistema operativo para identificar de forma única al proceso. En cambio el número de trabajo es un identificador de uno o varios procesos correspondientes a un usuario.

Anteriormente vimos que C-z detiene (suspende) un proceso y lo deja en segundo plano. También vimos como podíamos enviarle un señal para que continuara su ejecución en segundo plano. Ahora lo que vamos ver son los mecanismos para realizar una gestión más completa de esos trabajos.

Orden fg (foreground): traer a primer plano

La orden fg se utiliza como:

$ fg [%num_trabajo]

y se utiliza para traer a primer plano un trabajo que está en segundo plano, bien esté activo o bien esté detenido.

Ejemplo:

```
$ sleep 200  (y pulsar C-z)
$ sleep 300  (y pulsar C-z)
$ fg %1      (y pulsar C-z)
$ fg %2
```

Orden bg (background): continuar en segundo plano

Hasta ahora hemos alternado entre un proceso detenido y un proceso en primer plano, y podíamos hacer que continuara enviándole una señal. Tenemos otra forma para hacer que un proceso detenido en segundo plano continúe su ejecución. La orden bg se utiliza como:

$ bg [%num_trabajo]

y se utiliza para poner en ejecución en segundo plano un trabajo que está en segundo plano detenido.

Ejemplo:

```
$ sleep 200  (y pulsar C-z)
$ ps
$ bg
$ sleep 201 &
$ ps
$ fg %2      (y pulsar C-z)
$ fg %1      (y pulsar C-z)
$ bg %1
$ bg %2
```

$ ps

Orden jobs: mostrar lista de trabajos

Con la orden jobs podemos obtener una lista de los trabajos que hemos lanzado en el sistema. La orden jobs se utiliza como:

```
$ jobs
```

Al usarla nos aparece algo como

```
$ jobs
[1] Running sleep 100 &
[2] Running sleep 101 &
[3]- Running sleep 102 &
[4]+ Running sleep 103 &
```

donde entre corchetes tenemos el número de trabajo, y los signos + y menos indican:

+ El trabajo es el primero de la lista

- El trabajo es el segundo de la lista

Ejemplo:

```
   $ sleep 500 &
    $ sleep 450 ( pulsamos Ctlr-z)
[2]+ Stopped sleep 450
    $ jobs
[1]- Running sleep 500 &
[2]+ Stopped sleep 450
    $ bg
[2]+ sleep 450 &
```

y vemos como obtenemos la lista de trabajos y su estado

orden kill %: eliminar un trabajo

Con la orden

```
   $ kill %nº_trabajo ...
```

podemos matar un trabajo en ejecución.

nohup

La orden nohup lanza un proceso y lo independiza del terminal que estamos usando. Los procesos se organizan de forma jerárquica, de forma que si abandonamos la shell que nos conectó al sistema (abandonamos la sesión de trabajo) automáticamente se matarán todos los procesos que dependan de ella. Pero en muchas ocasiones no puede interesar lanzar un proceso y dejarlo en ejecución aun cuando hayamos cerrado la sesión de trabajo. Para esto se usa la orden nohup. Esta orden se usa como:

```
   $ nohup orden [argumentos]
```

Entrada y salida

Dispositivos estándares

Cualquier proceso que se lanza, por omisión, dispone de tres ficheros abiertos que son sus canales de entrada y salida estándares y salida de errores. La entrada estándar, el canal 0, también llamado stdin, por omisión se asocia al teclado. La salida estándar, el canal 1 o stdout por omisión se asocia a la pantalla (el terminal). La salida de errores, el canal 2 llamado stderr se asocia por omisión también a la pantalla al igual que la salida estándar.

Resumiendo, existen tres ficheros estándares asociados al teclado y a la pantalla.

Cada proceso tiene la posibilidad de alterar estos valores estándares y usar como entrada o salida aquéllos que se defina. También nosotros, desde la línea de órdenes tenemos la posibilidad de alterar ese comportamiento como veremos más adelante.

Otros dispositivos

En el sistema existen otros dispositivos asociados a distintos componentes de hardware del sistema. Los dispositivos se almacenan como ficheros en el directorio /dev. En la parte d administración del sistema se verán con más detalle los dispositivos, ahora no limitamos a una breve visión a nivel de usuario.

Como ejemplo de dispositivos podemos citar:

Discos duros IDE

```
/dev/hdXY
    ^ ^
    | |Y número de partición
    | X Número IDE a=1 master b=1 esclavo c=2 master d=2 esclavo
    hd = hard disk
```

Es decir, un dispositivo ide es un fichero ubicado en el directorio /dev y cuyo nombre comienza por hd. A continuación una letra a,b,c o d para indicar el disco duro correspondiente. Esta denominación hace referencia al disco duro completo, por ejemplo /dev/hda sería el primer disco duro IDE. Si el dispositivo es un lector de CDROM o DVD, es suficiente para identificar al dispositivo. Pero si el dispositivo es un disco duro además tendremos que añadirle un número que indica el número de partición, portanto /dev/hda3 indicaría la tercera partición del primer disco duro.

Discos flexibles

```
/dev/fdX
     ^ ^
     | |X número de unidad 0 = primera 1, = segunda
     | fd = floppy disk
```

Es decir /dev/fd0 sería la primera unidad de disquetes y /dev/fd1 sería la segunda si está presente.

Puertos serie

```
/dev/ttySn
     ^ ^
     | |n número de puerto
     | S = serie
   tty = terminal
```

Por ejemplo /dev/ttyS0 sería lo que msdos conoce como COM1, el primer puerto serie.

Otros dispositivos

/dev/psaux puerto ps2

/dev/audio sonido

/dev/dsp sonido

/dev/lp0 primer puerto de impresora

/dev/null dispositivo nulo

Este último dispositivo se utiliza para descartar salidas. Todo lo que se envía a /dev/null simplemente se pierde.

Redirección

En muchos casos nos va a interesar alterar las entradas y salidas estándares de un programa para que la información venga de otro origen distinto del habitual, o que la salida en vez de mostrarse en la pantalla se guarde en un dispositivo o fichero.

Para realizar esta redirección tendremos que usar los metacaracteres:

> redirige la salida la salida estándar a un fichero nuevo. Si el fichero existe se borra el contenido previo.

< redirige la entrada estándar por un fichero.

2> redirige la salida de errores a un fichero nuevo. Si el fichero existe se borra el contenido previo.

>> redirige la salida la salida estándar añadiéndola al final de un fichero. Si el fichero no existe lo crea.

2>> redirige la salida de errores añadiéndola al final de un fichero. Si el fichero no existe lo crea.

Ejemplos:

Realizar los siguientes ejemplo en la shell:

Comprobamos las diferencias entre y >>

```
$ echo ?Primera Línea? > nuevo
$ cat nuevo
$ echo ?Segunda línea? >> nuevo
$ cat nuevo
$ echo ?tercera linea? > nuevo
$ cat nuevo
$ echo ?Primera Línea? > nuevo
$ cat nuevo
$ echo ?Segunda línea? >> nuevo
$ cat nuevo
```

Vemos como copiar ficheros con >

```
$ cat nuevo > nuevo.1
$ cat nuevo.1
```

Enviamos un fichero de texto por correo

```
$ mail usuario -s ?prueba con mail? <nuevo.1
```

Comprobamos la salida de errores

```
$ aaa
$ aaa > nuevo.2
$ cat nuevo.2
$ aaa 2> nuevo.2
$ cat nuevo.2
```

Como untroducir textos multilínea

```
$ cat <<! >nuevo.3
escribir
```

```
varias
líneas
y terminar con un ! sólo en una línea
!
$ cat nuevo.3
```

Tuberías o pipes

Anteriormente vimos que cada proceso dispone de tres canales estándar para comunicarse con el exterior, las entrada y salida estándares y la salida de errores. En cualquier sistema Unix se puede hacer que la salida de una determinada orden sea la entrada estándar de otra, lo que le confiere a las órdenes Unix una enorme potencia. Ese es el motivo por el que las órdenes Unix sólo realizan estrictamente lo necesario, sin más salida de información adicional que pudiera interferir en la subsiguiente utilización de esta información.

Uniendo órdenes

Para unir la salida de una orden con la entrada de otra utilizamos el metacaracter de shell '|', poniendo a la izquierda la primera orden que queremos ejecutar y a la derecha la orden que tiene que ejecutarse con la información de la orden anterior.

El carácter de tubería realiza una doble redirección: toma la salida estándar de la orden que hay a su izquierda para pasarla como entrada estándar de la orden que hay a su derecha.

Podemos unir tantas órdenes como nos interese.

Vemos unos ejemplos:

```
$ ps axu | more
```

En este caso, la salida de la orden 'ps' se pasa como entrada a la orden more lo que nos permite verla poco a poco. Pero en Unix se pueden hacer cosas más complejas.

Por ejemplo:

```
$ ps axu | wc -l
```

En este caso la salida de ps se pasa a la orden 'wc -l' que cuenta líneas. De esta forma estamos contando el número de líneas que muestra la orden ps, es decir el número de procesos que hay actualmente ejecutándose en el sistema.

Ejercicios:

Contar el número de usuarios que hay conectados en el sistema:

Para ver los usuario que hay conectados ejecutamos

```
who
```

y como aparece uno por línea, parasaber el número de usuarios lo que hacemos es contar las líneas.

```
who|wc -l
```

Contar el número de ficheros que hay en el directorio actual:

Como en el caso anterior, lo que vamos a hacer es contar el número de líneas:

```
ls -a|wc -l
```

orden tee: extrae la salida estándar

La orden tee copia la entrada estándar en la salida estándar y en uno o más ficheros. Se utiliza cuando queremos pasar información a una orden y además almacenarlos resultados intermedios en un fichero; por ejemplo:

```
$ cat /etc/passwd | tee claves | wc -l
```

Ejercicios

Averiguar cuantos ficheros hay en el directorio actual, sin incluir los ocultos

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

Averiguar cuantos ficheros hay en el directorio actual, incluyendo los ocultos

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

Copiar el fichero /etc/hosts en el directorio actual a la vez que lo vemos por la pantalla

```
$ cat /etc/hosts | tee ./hosts
```

Duplicar un disquete, suponiendo que tenemos dos unidades de discos flexibles

```
$ cat /dev/fd0 > /dev/fd1
```

Duplicar un disquete suponiendo que la máquina tiene una sola unidad:

```
$ cat /dev/fd0 > disco.img
$ cat disco.img > /dev/fd0
$ rm disco.img
```

Añadir la línea ?192.168.1.101 pci.bez.es? al fichero hosts del directorio personal

```
echo ?192.168.1.101 pci.bez.es? >> ~/hosts
```

Guardar la fecha y la lista de usuarios conectados en este momento en un fichero llamado conectados

```
$ date >> conectados
$ who >> conectados
```

Obtener una lista de usuarios conectados ordenada alfabéticamente

```
$ who | sort
```

Averiguar cuantos usuarios distintos hay en el sistema

```
$ cat /etc/passwd | wc -l
```

Mostrar sólo la quinta línea del fichero /etc/passwd

```
head -5 /etc/passws|tail -1
```

Expresiones regulares

Son muchas las situaciones en las que tendremos que realizar operaciones de transformación y búsquedas de textos en el trabajo cotidiano de un sistema informático. Si somos programadores nos encontraremos con la necesidad de comprobar si una cadena de texto es efectivamente una dirección de correo electrónico, una URL, un número entero, flotante, una fecha, un número de pasaporte, de cuenta bancaria, etc. Muchas de estas comprobaciones pueden costar un esfuerzo adicional, muchas líneas de programación que como veremos podremos evitar. Si somos administradores tampoco o van a resultar menos útiles las expresiones regulares; por ejemplo, podemos buscar ficheros que contengan algún dato en concreto, generar páginas web automáticamente a partir de datos del sistema como cuotas, ficheros de log, etc, podremos realizar ediciones masivas de ficheros.

Merece la pena el esfuerzo de aprender a usar expresiones regulares por el trabajo que nos pueden ahorrar en proyectos futuros. Es más que probable que quien no sepa expresiones regulares no las eche de menos en su trabajo cotidiano, pero lo que es cierto es que quien las domina puede ahorrar bastente trabajo y esfuerzos.

Introducción

Definición

Una expresión regular es una plantilla formada por unos metacaracteres con significado propio que se utiliza para especificar patrones regulares sobre cadenas de texto. Como esta definición no aclara mucho los conceptos vamos a intentar aclarar más las cosas con algunos ejemplos prácticos. Un ejemplo de algo parecido a un expresión regular serían las plantillas para nombres de ficheros usando * y ?; en este caso, *.txt expresaría todos los ficheros cuyo nombre termina en .txt, pero una expresión regular va mucho más alla. Una expresión regular se puede construir, para concordar con, por ejemplo, una dirección de correo electrónico, un número, una fecha, una url, una etiqueta HTML, etc., es decir, cualquier cadena que cumpla unas determinadas características de regularidad.

Existen distintas formas de aplicar expresiones regulares y puede que no todos los programas las admitan en su totalidad. Más adelante, en los ejemplos veremos ciertos programas para ver qué podemos aplicar en cada caso y por qué se producen errores de interpretación. Podemos especificar la sintaxis, POSIX, tradicional o Unicode. Nos vamos a centrar en la sintaxis POSIX, aunque también veremos a la vez algo de la sintaxis tradicional. Aquí vamos a ver una visión general y completa de las expresiones regulares.

Como pueden servir la expresiones regulares

Las expresiones regulares nos van a resultar de suma utilidad para realizar operaciones sobre cadenas de caracteres, tanto de comprobación como de edición, con una potencia imposible de obtener por lo sistemas tradicionales que proporcionan los editores de texto.

Por ejemplo, si queremos encontrar en una página web todos los colores (bgcolor) de las celdas (<td>)de una tabla tendríamos la siguiente expresión:

/<td[^>]*?bgcolor=.?#([0-9a-fA-F]{6})[^>]*>/

Otro ejemplo, muy simple y mejorable, para verificar una dirección simple de correo electrónico:

```
[A-z0-9_]+(EN)([A-z0-9]+\.[a-z]{2,4}
```

Al principio parece un dialecto del idioma del planeta Vulcano, pero poco a poco iremos familiarizandonos con él y comprendiendo el significado de su sintaxis.

Metacaracteres de expresiones regulares

Existen ciertos caracteres que tienen un significado específico dentro de una expresión regular. Estos caracteres especiales, o metacaracteres, se pueden aplicar a definiciones de carácter para especificar con qué caracteres se concuerda, cuantificadores para especificar el modelo de repetición de los caracteres y también anclajes para especificar una posición dentro de un texto.

Estos caracteres, como hemos visto, se denominan metacaracteres porque tienen un significado que va más allá del símbolo que representa y tiene un comportamiento especial en una expresión regular:

Definiciones de carácter

Primeramente pasamos a la definición de caracteres en una expresión regular. Vamos a detallar los metacaracteres que permiten especificar una plantilla que concuerde con un texto de unas características concretas. Dentro de las definiciones de caracteres podremos especificar caracteres genéricos, rangos de caracteres, exclusión de caracteres, literales y otras características adicionales.

Metacarácter	Significado
.	Concuerda con cualquier carácter (salvo fin de línea en modo multilínea)
()	Subexpresión o grupo
[ ]	Conjunto de caracteres
[-]	Rango de caracteres
[^]	Excepto ese conjunto de caracteres
|	Permite una alternativa para elegir entre dos expresiones
/ /	Delimita una expresión regular

\	Protege el siguiente metacarácter

Las expresiones permiten también especificar caracteres especiales no imprimibles:

Metacarácter	Significado
\a	pitido, el carácter BEL (07 en hexadecimal)
\e	escape (1B en hexadecimal)
\cx	"control-x", donde x es el carácter correspondiente
\f	nueva página (0C hexadecimal)
\n	nueva línea (0A hexadecimal)
\r	retorno de carro (0D hexadecimal)
\t	tabulador (09 hexadecimal)
\xhh	carácter con código hh hexadecimal
\ddd	carácter con código ddd en octal

Ejemplos:

Expresión	Significado
/[a-z]/	una letra minúsculas. El "-" indica un rango, que en este caso comienza en "a" y termina en "z".

/[A-Z]/	una letras mayúscula
/[0-9]/	un dígito
/[,'¿!¡;:.?]/	un carácter de puntuación
/[A-Za-z]/	una letra salvo acentuadas y ñ
/[A-Za-z0-9]/	una letra, salvo acentuadas y ñ, o un dígito
/[^a-z]/	cualquier carácter salvo una letras minúscula
/[^0-9]/	Cualquier carácter salvo un número.

Clases

Podemos especificar clases de caracteres según varias sintaxis, POSIX, tradicional o Unicode.

Según la sintaxis de clases POSIX, podemos indicar [:clase:] donde clase puede ser alguna de las siguientes expresiones:

Clase	Significado
[:alpha:]	carácter alfabético
[:alnum:]	carácter alfanumérico
[:ascii:]	carácter ascii
[:blank:]	espacio, incluye tabulador (también \s según la sintaxis tradicional)
[:cntrl:]	carácter de control

[:digit:]	un dígito (también \d según la sintaxis tradicional)
[:graph:]	carácter gráfico
[:lower:]	letra minúscula
[:print:]	carácter imprimible
[:punct:]	carácter de puntuación
[:space:]	espacio (también \s según la sintaxis tradicional)
[:upper:]	letra mayúscula
[:word:]	palabra (también \w según la sintaxis tradicional)
[:xdigit:]	dígito hexadecimal

En la sintaxis tradicional, además tenemos:

Metacarácter	Significado
\D	cualquier carácter que no sea un dígito decimal (equivaente a [^:digit:])
\S	cualquier carácter que no sea un espacio en blanco (euivalente a [^:blank:]
\w	cualquier carácter de de una palabra
\W	cualquier carácter que no sea de una "palabra"

Literales

Cualquier otro carácter especificado en unaexpesión regular se representa a sí mismo. Por ejemplo, la expresión regular:

```
/To:[ ]*[^(EN)]+(EN)midominio\.dom$/
```

consta de:

dos literales :to: y (EN)midominio.dom (el "." va protegido).

un anclaje, el $ que indica final de línea

dos conjuntos de caracteres [ ] que representa un espacio en blanco y [^(EN)] que indica todo lo que no sea (EN).

los cuantificadores * que indica en este caso posibles espacios en blanco y + que indica repetición de una o más veces el carácter anteriorior, en este caso todo lo que no sea (EN). Los metacaracteres * y + se analizan con más detalles en su apartado específico.

Metacarácter como literal

Cuando queramos utilzar un metacarácter como literal tendremos que protegerlo con una contrabarra ("\"). Ya hemos visto varios ejemplos, como el caso de /midominio\.dom/. En este caso el carácter "." no tiene su valor propio de una expresión regular de ser cualquier carácter, sino que al ir protegido representa el .

Un caso particular lo podemos encontrar si usamos expresiones regulares dentro de una shell, por ejemplo un script de shell. En este caso deberíamos realizar en algunos casos una doble protección con la contrabarra, una para protegerlo en shell y otra para potegerlo en la expresión regular. Por ejemplo:

Si ponemos:

```
A="La variable \$A está definida"
```

En realidad estamos almacenando, "La variable $A está definida". Evidentemente esta cadena como expresión regular no tiene el signo $ protegido.

Si quisiéramos almacenar "La variable \$A está definida" entonces deberíamos poner:

```
A="La variable \\\$A está definida"
```

Cuantificadores

Los cuantificadores son metacaracteres de la expresiones regulares que indican las características de repetición de caracteres o grupos. Ya hemos visto en el ejemplo

```
/To:[ ]*[^(EN)]+(EN)midominio\.dom$/
```

como nos interesaba un expresión que después de To: pudiera haber cualquier serie de espacios en blanco, incluido ninguno, y a continuación de los espacios una serie de caracteres salvo "(EN)". Todas estas repeticiones las podremos expresar mediante los cuantificadores que describimos a continuación:

Metacarácter	Concuerda con
*	repetición de 0 o más veces el carácter o subexpresión previos
+	repetición de 1 o más veces el carácter o subexpresión previos
?	repetición de 0 o una vez el carácter o subexpresión previos
{n}	repetición de n veces el carácter o subexpresión previos
{n,}	repetición de n veces el carácter o subexpresión previos
{n,m}	repetición entre n y m veces el carácter o subexpresión previos

Para aclararnos un poco:

```
"*" equivale a "{0,}"
"+" equivale a "{1,}"
"?" equivale a "{0,1}"
```

Un cuantificador hace que la concordancia sea siempre con el mayor numero de veces posibles. Si queremos que sea con el menor posible le añadimos un "?".

Anclajes

Los puntos de anclaje, especifican una condición que se verifica en un punto particular del

texto, sin corresponder a ningún carácter de la cadena de entrada concreta, esto es, una posción en el texto:

Metacarácter	Significado
^	Concuerda con el principio de línea
$	Concuerda con el final de línea
< >	Concuerda con una palabra
\b	limites de palabra
\B	no sean limites de palabra
\A	inicio de la cadena de entrada (independiente del modomultilínea)
\Z	fin de la cadena de entrada o nueva línea delante del fnal
\z	fin de la cadena de entrada (independiente de modomultilínea)

Ejemplos

Expresión regular	Significado
/^En un.*/	Una línea que comienza por "En un".
/^En un.*apache$/	Una línea que comienza por "En un" y termina en "apache".
/^En un.*[Aa]pache$/	Una línea que comienza por "En un" y termina en "apache" o "Apache".

/^(http|ftp):\/\/.*\.com\/$/	Una línea que comienza por "http" o "ftp" y termina en ".com/".
/<td[^>]*>/	Cumple "<td lo quesea>"
/[ ][ ]*/	Cualquier secuencia de espacios en blanco
/[ ]+/	Cualquier secuencia de espacios en blanco. Igual que la anterior.
/^[^#]*$/	Especifica una línea que NO comience por el carácter "#".
/^[^#]*tty$/	Especifica una línea que NO comience por el carácter "#" y termine en "tty".
/^Expresiones Regulares Pag\. 2$/	Es una expresión regular que indica una línea que es exactamente "Expresiones Regulares Pag. 2". Empieza y acaba en esa frase.
/^[ ]*Expresiones Regulares Pag\. 2$/	Es una expresión regular que indica una línea que empieza por una serie de espacios en blanco y termina en "Expresiones Regulares Pag. 2"
/^[:blank:]*Expresiones Regulares Pag\. 2$/	Equivalente a la anterior, pero además de espacio incluye tabuladores
/^\s*Expresiones Regulares Pag\. 2$/	Igual que el anterior
/^Expresiones[ ]* Regulares Pag\. 2$/	Especifica una línea que empieza por "Expresiones" después una serie de espacios en blanco y termina en "Regulares Pag. 2".

Observamos como en todas las expresiones regulares anteriores hemos puesto \., El signo "." tiene su significado propio, cualquier carácter por lo que si queremos incorporar el punto como tal tenemos que protegerlo con "\".

Alternativas

Una alternativa es el equivalente a un O lógico. El simbolo de la alternativa es | como veíamos anteriormente.

Por ejemplo /(http|ftp)/ especifica "http o ftp".

Imaginemos que queremos un expresión que verifique una línea que termina en un nombre de fichero ejecutable dos/win32. Los ejcutables los vamos a limitar a .bat, .exe y .com, con lo que quedaría:

```
/.*(\.bat|\.exe|\.com)$/
```

con lo que estamos especificando una línea que termina en .bat, .exe o .com.

Construyendo expresiones regulares: ejemplos

Sintaxis de fecha del tipo dd/mm/aa

En este ejemplo sólo pretendemos comprobar que la sintaxis de la fecha es correcta, es decir que estábien construida independientemente de que la fecha sea realmente válida.

En una primera aproximación sólo comprobaremos que tenemos seis dígitos separados por "/".

```
/[0-9]{2}\/[0-9]{2}[0-9]{2}\/[0-9]{2}/
```

Usamos como separador de la expresión regular el carácter "/". Entonces los caracteres "/" propios del formato de fecha tienen que ir protegidos mediante la contrabarra ("\"). El resto de la expresión consiste en indicar que tenemos dos dígitos.

Esta expresión no verificará fechas del tipo 2/3/07, ya que según la definición obligamos a tener dos dígitos por componente. Para mejorarla exigiremos que los componentes correspondientes al dia y al mes puedan tener uno o dos dígitos:

```
/[0-9]{1,2}\/[0-9]{1,2}[0-9]{2}\/[0-9]{2}/
```

No obstante seguirían siendo valores de fecha aceptados 9/99/07. Para afinar un poco más la expresión tendríamos que exigir que el mes no pudiera ser mayor que 12 y el día no mayor

que 31:

En el caso del mes pondríamos

```
\/([1-9]|1[012])\/
```

es decir o tenenemos un solo dígito que va del 1 al 9 o tenemos dos, de forma que el primer es un 1 fijo y el segundo puede ser 0, 1 ó 2.

En el caso del día, y con una razonamiento similar pondríamos:

```
\/([1-9]|[12][0-9]|3[01])\/
```

es decir el día es un solo dígito del 0 al 9, o dos dígitos donde el primero es 1 ó 2 y el segundo cualquiera entre el 0 y el 9 y por último un dígito que puede ser 3 y seguido de un 1 o un 2.

Ahora todo junto quedaría como

```
/\/([1-9]|[12][0-9]|3[01])\/\/([1-9]|1[012])\/[0-9]{2}/
```

Es importante observar como el carácter "|" permite seleccionar alternativas de concordancia.

Por último, si quisiéramos que también fuer válida una fecha como 01/02/08, la expresión quedaría como:

```
/\/(0?[1-9]|[12][0-9]|3[01])\/\/(0?[1-9]|1[012])\/[0-9]{2}/
```

Es decir, añadiendo "0?" estamos diciendo que el "0" puede o no estar presente.

Dirección de correo electrónico

Vamos a definit una expresión regular para comprobar una dirección de correo electrónico:

En primer lugar definimos todo lo que hay a la izquierda de la (EN):

Será una serie de caracteres alfanuméricos, un posible carácter de separación y otra serie de caracteres alfanuméricos opcional. Con esto se verifican valores como:

aaa

aaa-bbb

aaa.bbb

aaa_bbb

aaa_bbb_ccc

Esta primera parte quedaría como:

```
([0-9a-zA-Z]+)([._-]([0-9a-zA-Z]+))*
```

A continuación, después del literal (EN) el nombre del dominio que se construye de una forma similar al nombre del usuario y al que le añadimos un dominio de nivel superior que consta de 2 a 4 caracteres.

La parte del dominio sería:

```
([0-9a-zA-Z]+)([._-]([0-9a-zA-Z]+))*[.]([0-9a-zA-Z]){2,4}
```

Ahora todo junto, y le añadimos el comienzo y fin de línea:

```
"^([0-9a-zA-Z]+)([._-]([0-9a-zA-Z]+))*(EN)([0-9a-zA-Z]+)([._-]([0-9a-zA-Z]+))*[.]([0-9a-zA-Z]){2,4}$"
```

Referencias

Cuando utilizamos subexpresiones, podemos utilizar el valor encontrado correspondiente a la subexpresión posteriormente, normalmente cuando realizamos sustituciones en cadenas de texto. Las subexpresiones se ordenan de izquierda a derecha correspondiendo \1 a la primera subexpresión, \2 a la segunda y así sucesivamente.

Por ejemplo, si la expresión regular es:

```
([A-Za-z]+)([0-9]+)
```

entonces \1 se refiere al conjunto de letras que han verificado la expresión regular (primer paréntesis) y \2 a una serie de dígitos.

Además el carácter "&" como referencia equivale a toda la expresión regular.

Modos de búsqueda

Existen varios métodos de búsqueda

/i evita la distinción entre mayúsculas y minúsculas.

/s activa modo "línea única". En este modo los saltos de líneas verifican el punto.

/m activa modo "multilínea". En este modo los saltos de líneas verifican el $ y el carácter siguiente el ^.

/g global, hace referencia a todas las apariciones de la expresión regular. Si se omite sólo hacemos referencia a la primera.

Ejemplos

Expresión para comprobar una dirección de correo electrónico:

```
"^([0-9a-zA-Z]+)([._-]([0-9a-zA-Z]+))*(EN)([0-9a-zA-Z]+)([._-]([0-9a-zA-Z]+))*[.]
([0-9a-zA-Z]){2}([0-9a-zA-Z])?$"
```

Expresión de un hiperenlace

```
"<a[ ]{1,}href=[\"']{0,}([a-zA-Z0-9/:~._#]{0,})[\"']{0,}[^>]{0,}>([^<]{0,})</a>"
```

Tratamiento de ficheros de texto

Los sistemas Unix, y Linux en particular disponen de herramientas avanzadas que permiten la manipulación de ficheros de texto para poder extraer información y modificarlos. Esto es realmente importante ya que la mayoría de los ficheros de configuración de un sistema Unix son ficheros de texto que habitualmente tendremos que manipular.

Orden grep: buscar en un fichero

La orden grep busca una expresión regular en un fichero mostrando el resultado en la salida estándar.

La orden grep se usa como:

```
$ grep [opciones] expresión_regular [lista_ficheros ...]
```

Consultar la página de manual correspondiente para tener más detalle sobre las opciones disponibles de grep.

Un uso habitual de grep es usar como entrada la salida estándar de otra orden para filtrar la información y obtener sólo aquéllas líneas que nos puedan interesar.

A continuación vemos algunos ejemplos prácticos:

Ejemplos

Buscar las líneas que contengan

La palabra pc seguida de dígitos

```
$ grep pc[1-9]* /etc/hosts
```

La palabra procmail en el fichero /etc/mail/sendmail.cf

```
$ grep procmail /etc/mail/sendmail.cf
```

La palabra no en /etc/passwd

$ grep no.* /etc/passwd

Líneas que comienzan por n en /etc/passwd

```
$ grep ^n.* /etc/passwd
```

Líneas qu terminan en false en /etc/passwd

```
$ grep false$ /etc/passwd
```

Líneas que comiencen por "p" o "n" de /etc/passwd

```
$ grep '^p\|^n' /etc/passwd
```

La palabra tcp en el fichero /etc/inetd.conf

```
$ grep tcp /etc/inetd.conf
```

Líneas que comiencen por 1 en el fichero /etc/inittab

```
$ grep ^1 /etc/inittab
```

Líneas que conengan "in." en /etc/inetd.conf

```
$ grep ?in\.? /etc/inetd.conf
```

La palabra nobody en la lista de procesos

```
$ ps axu | grep nobody
```

La palabra httpd en la lista de procesos

```
$ ps axu | grep httpd
```

Cuantas veces está activo el proceso https

```
$ ps axu | grep httpd | wc -l
```

Quien usa la terminal tty1

```
$ who | grep tty1
```

Lista de ficheros que contenan el texto .bashrc

```
$ ls -la | grep \.bashrc
```

Lista de ficheros con permiso de lectura y escritura para su propierario

```
$ ls -la | grep ^-rw
```

Lista de enlaces simbólicos

```
$ ls -la | grep ^l
```

Lista de directorios

```
$ ls -la | grep ^d
```

Orden egrep: busca en una lista de ficheros

La orden egrep permite realizar un grep en múltiples ficheros. Vea la opción -E de grep.

Por ejemplo, si queremos buscar la palabra ?fail? en cualquier fichero del directorio /var/log, pondríamos:

```
$ egrep fail /var/log/*
```

Observamos que pueden aparecer línea de error diciendo que ha intentado buscar en un directorio. Si queremos descartar estas líneas de error pondríamos lo siguiente:

```
$ egrep fail /var/log/* 2>/dev/null
```

Orden sed: editor no interactivo

sed es un editor de texto no interactivo. Su principal utilidad es poder editar ficheros de textos desde un programa. Hay que observar que el resultado de la edición se muestra en la salida estándar.

La orden sed dispone de diversas opciones, pero aquí sólo vamos a ver una, la sustitución de un texto por otro.

En general la orden sed se usa como:

```
$ sed orden_sed fichero
```

Donde:

orden es una instrucción de sed. Entre las órdenes está 's' que se utiliza para sustituir texto. En este caso la orden queda como:

```
s/expr_reg_origen/nuevo valor/
```

Si al final añadimos 'g' indicamos que haga una sustitución global en todo el fichero.

Vamos a ver un ejemplo:

Supongamos que tenemos un fichero de texto llamado ftexto con el siguiente contenido:

```
$ cat ftexto
 enero, febrero, noviembre, abril
 a12, a3, s34, e56, 123,
 lunes
 martes 23
 37
```

Veamos los resultado que obtendríamos con diferentes órdenes de sed:

```
$ sed "s/12/zzzzzzz12zzzzzzzz/g"
 enero, febrero, noviembre, abril
 azzzzzzz12zzzzzzzz, a3, s34, e56,
 zzzzzzz12zzzzzzzz3, lunes
 martes 23
 37
```

Otro ejemplo:

```
$ sed "s/noviembre/diciembre/" ftexto
 enero, febrero, diciembre, abril
 a12, a3, s34, e56,
 123, lunes
 martes 23
 37
```

También podemos hacer que el texto buscado aparezca en el texto sustituido, usando &, lo que veíamos como referencias en las expresiones regulares. Por ejemplo

```
$ sed "s/[0-9][0-9]/& dos numeros/g" ftexto
 enero, febrero, noviembre, abril
 a12 dos numeros, a3, s34 dos numeros, e56 dos numeros,
 12 dos numeros3, lunes
 martes 23 dos numeros
 37 dos numeros
```

Como la orden sed escribe sus resultados en la salida estándar, si queremos modificar un

fichero tendremos que hacerlo en varios pasos redirigiendo la salida estándar.

Por ejemplo, si queremos que en el nuevo fichero ftexto donde pone lunes ponga domingo tendríamos que poner:

```
$ sed "s/lunes/domingo/" ftexto >ftexto.nuevo
$ cp ftexto.nuevo ftexto
$ rm ftexto.nuevo
```

Ejemplos

```
$ echo "aabbccdd"|sed "s/a/x/"
xabbccdd
$ echo "aabbccdd"|sed "s/a/x/g"
xxbbccdd
$ echo "aabbccdd"|sed "s/a\+/x/"
xbbccdd
$ echo "aabbccdd"|sed "s/a*/x/"
xbbccdd
```

En el primer ejemplo observamos como sólo sustituye la primera "a" por la "x". En el sigundo sí ha sustituido todas las apariciones de "a" al haber añadido la opción /g.

En el tercer ejemplo se sustituye una o más apariciones de "a" por "x".

En el último obsevamos el resultado esperado, sustituir una secuencia de "a" por una x.

Palabras

```
echo "esto si es asi"|sed "s/\<si\>/no/g"
esto no es asi
```

En el anterior ejemplo vemos como sustituir una palabra por otra en un texto sin que afecte a las demás. Hemos sustituido la palabra "si" por "no" sin que afecte a la palabra "asi".

Mayúsculas

```
echo "esto si es asi"|sed "s/\<Si\>/no/gi"
esto no es asi
$ echo "esto si es asi"|sed "s/\<Si\>/no/g"
esto si es asi
```

Vemos como al añadir la opción /i no se distinguen mayúsculas de minúsculas mientras que si la omitimos hacemos distinción.

Referencias

```
echo "esto si es asi"|sed "s/\(.*es\)\(.*\)/\2 \1 /g"
 asi esto si es
```

```
echo "esto si es asi"|sed "s/\(.*es\)(.*\)/## & ##/g"
## esto si es asi ##
```

Orden cut: corta texto

La orden cut corta un trozo de línea de su entrada estándar de acuerdo con las opciones especificadas.

La orden cut se usa como:

```
$ cut opciones fichero
```

y muestra los resultados en la salida estándar.

Las opciones más importantes de esta orden son:

-c n1-n2 Corta caracteres desde el que ocupa la posición n1 hasta la posición n2.

-d separador Indica el carácter que actúa como separador

-f c1 [,c2,...] Indica qué campos queremos cortar. c1 es un número que representa el orden del campo que queremos cortar. Se supone que los campos están separados por el carácter indicado con la opción -d

Ejemplos:

```
$ cut -d: -f1 /etc/passwd
  $ who | cut -c 10-17
  $ ls -la | cut c 2-10
  $ ls -la | grep ^d | cut -c 2-10
  $ cut -d: -f 1,3 /etc/group
  $ ps axu | grep root | cut -c 1-6
  $ cat /etc/hosts | grep ^[0-9] | cut -c 1-6
  $ grep udp /etc/inetd.conf | cut -d: -f 1,2,3,4
  $ grep ^[^#] /etc/inittab | cut -d: -f 2,4,5
  $ cut -d: -f 1,6 /etc/passwd > lista
```

Ordenes spell, aspell: corrector ortográfico

Estas órdenes verifican la ortografía de un fichero de texto respecto a un diccionario y un conjunto de reglas de formación de palabras.

Estas órdenes hay que instalarlas explícitamente en un sistema. Muchos programas, sobre todo editores de texto, utilizan estas órdenes del sistema.

Para más detalles consultar las páginas del manual.

Permisos y propietarios

Con anterioridad ya habíamos hablado de los permisos y de los propietarios de un fichero o directorio; ahora entramos en más detalles.

Existen órdenes para modificar, tanto los permisos como los propietarios de un fichero. También podremos establecer los criterios sobre los permisos de los ficheros de nueva creación.

Propiedad

Cualquier sistema operativo multiusuario tiene que proporcionar mecanismos para que cada usuario pueda poseer su propios datos. En el caso de los sistema Unix y Linux en particular a cada fichero o directorio se le puede asignar un usuario propietario y un grupo propietario. La lista de usuarios está en el fichero /etc/passwd y la lista de grupos está en el fichero /etc/group. Cada usuario tiene un grupo principal ya además puede pertenecer a cualquier otro grupo.

Combinando el propietario, el grupo de un fichero y los grupos a los que pertenecen los diversos usuario se puede delimitar perfectaemente el acceso a los datos almacenados. Además los sistemas Unix existen ciertos usuarios y grupos estándares que se utilizan para realizar diversas tareas administrativas. En el apartado de administraciñón de usuarios se verá esto con más detalle.

En general, el propietario y el grupo de los ficheros y directorios se heredan del usuario que los crea.

En muchos casos nos interesa modificar el propietario o el grupo de un fichero; para realizar estas acciones Linux (Unix en general) disponen de las órdenes adecuadas.

Orden chown: cambia el propietario

Para cambiar el propietario de un fichero tenemos que usar la orden chown (change owner). La orden chown se utiliza de la siguiente forma:

```
$ chown nuevo_propietario fichero_o_dir
```

La orden chown también se puede utlizar para cambiar también el grupo de un fichero o directorio; esto se hace con la siguientes sintaxis de la orden chown:

```
$ chown nuevo_usuario.nuevo_grupo fichero_o_dir
```

o bien

```
$ chown nuevo_usuario:nuevo_grupo fichero_o_dir
```

poniendo el nuevo usuario y el nuevo grupo separados por un punto. De esta forma ahora el fichero o el directorio pertenecen al nuevo usuario y al nuevo grupo. Si omitimos el usuario pero ponemos el punto o los dos puntos sólo se cambia el grupo.

Orden chgrp: cambia el grupo de un fichero o directorio

La orden chgrp cambia el grupo de un fichero o directorio. Se utiliza como

```
$ chgrp nuevo_grupo fichero_o_dir
```

Esta orden tiene la misma funcionalidad que

```
$ chown :nuevo_grupo fichero_o_dir
```

Permisos

chmod

La orden chmod cambia los permisos de un fichero o directorio. Se usa como:

$ chmod [opciones] permiso fichero

Donde permiso se puede especificar de dos formas distintas:

a)

> En octal, donde un 0 representa un permiso denegado y un 1 un permiso concedido. El número octal está formado por tres dígitos; el de más a la izquierda representa representa los permisos del usuario propietario del fichero o (directorio). El dígito central representa los permisos del grupo propietario del fichero y el dígito de la derecha representa los permisos que se conceden al resto de los usuarios del sistema. Cada uno de los dígitos octales se descompone en tres dígitos binarios donde de izquierda a derecha significan lectura (r), escritura (w) y ejecución (x).

Por ejemplo: 764 indica

Propietario 7 = 111 = rwx

Grupo 6 = 110 = rw-

Otros 4 = 100 = r-

b)

> En formato simbólico, de la forma "d+4#4-p", donde "d" es el destinatario del permiso y puede ser "u" (usuario propietario), "g" (grupo propietario) y "o" (otros, el resto de los usuarios); también se puede poner "a" que significa todos (all). Un signo "+" activa un

permiso y un signo "-" lo desactiva. El siguiente carácter "p" representa el permiso que puede ser r (lectura), w (escritura) y x (ejecución) o una combinación de ellos.

Por supuesto, el fichero indica quien recibe los permisos.

Realizar las siguentes operaciones en su sistema linux y ver como se modifican los permisos:

$ touch mifi

$ chmod ug+wr mifi

$ ls -la mi*

$ chmod o-wrx mifi

$ ls -la mifi

$ chmod 700 mifi

$ ls -la

$ chmod 731 mifi

$ ls -la

y podremos ver como van cambiando los permisos del fichero con las sucesivas ejecuciones de la orden chmod.

Si a la orden chmod le añadimos ls opción -R modifica los permisos de forma recursiva, es decir incluyendo toda la rama del árbol de directorios a partir de donde se ejecuta la orden.

Hay que observar que el comportamiento de los distintos permisos es distinto para ficheros y directorios:

- El permiso de lectura en un fichero implica poderlo leer mientras que en un directorio significa mostrar su contenido.
- El permiso de escritura en un fichero indica poderlo borrar o modificar mientras que para un directorio indica crear ficheros en él o borrar los existentes.
- El permiso de ejecución para un fichero implica que se pueda ejecutar como programa, mientras que un directorio significa poder entrar en él.

Los bits SUID y SGID

Existen otros dos permisos, propios de los ficheros ejecutables que modifican las características de los procesos generados al ejecutarlos si estos permisos están activos. Antes de entrar en más detalles vamos a aclarar dos conceptos sobre los procesos que se lanzan:

- usuario real: el usuario que está conectado y que lanza el proceso.
- usuario efectivo: el usuario que es propietario de un proceso, y cuyos permisos le son

aplicables.

Esto es importante porque algunos programas necesitan ejecutarse como un usuario distinto al usuario que lo lanza. Por ejemplo, el fichero (/etc/passwd) es propiedad del usuario root y del grupo root y tienen los permisos rw-rw-r- o rw-r-r-, lo que quiere decir que cualquiera puede leerlo pero sólo el root puede escribir en él. Pero un usuario para poder modificar su clave de acceso tiene que escribir en el fichero. Pero claro, al ejecutar el programa (passwd), este proceso es propiedad del usuario que lo lanza y en consecuencia no puede modificarlo. Entonces ¿cómo podemos solucionar esto? Pues haciendo que el proceso generado por la orden passwd sea propiedad del root y activando el bit SUID. De esta forma cuando la orden passwd genera un proceso se genera como un proceso que es propiedad del propietario del fichero, y no propiedad del usuario que lo lanza.

Hay que limitar en la medida de lo posible los fichros ejecutables con el bir SUID activo, sobre todo si los ficheros son propiedad del root, ya que pueden originar problemas de seguridad, que alguien consiga acceso de root de forma indebida.

El permiso SGID es igual el SUID salvo que se aplica al grupo en lugar de la usuario. No son incompatibles.

Los bits SUID y SGISD los podemos asignar como:

$ chmod u+s fichero

$ chmod g+s fichero

El sticky bit

Este es un bit que tiene un significado para los directorios. Cuando este bit está activo, hace que un usuario sólo pueda borrar los ficheros que son de su propiedad en dicho directorio. Esto es particularmente útil en el directorio /tmp.

El sticky bit se activa como:

$ chmod +t directorio

Ejercicios:

Vemos ahora un lista de ejercicios para comprobar como actúan las últimas órdenes.

Ejercicio 1

Crear un fichero llamado borrar:

```
$ touch borrar
```

Verificar los permisos y los propietarios y grupo.

```
$ ls -la bo*
```

Asignar los permisos necesarios para que podamos borrar el fichero si lo cambiamos de propietario.

```
$ chmod g+w borrar
```

Intentar ponerle al fichero los todos los permisos a todo el mundo:

```
$ chmod a+wrx borrar
```

Borrar el fichero:

```
$ rm borrar
```

Ejercicio 2

Crear un fichero llamado eliminar

```
$ touch eliminar
```

Verificar los permisos del fichero

```
$ ls -la el*
```

Asignar los permisos necesarios para poderlo borrar si cambiamos el grupo y propietario.

```
$ chmod o+wr eliminar
```

Asignar permisos de ejecución a eliminar.

```
$ chmod u+x eliminar
```

Borrar el fichero

```
$ rm eliminar
```

Ejercicio 3

Crear un fichero llamado pr1

```
$ touch pr1
```

Asignarle el permiso de ejecución para todos los usuarios:

```
$ chmod a+x pr1
```

Añadir una línea al fichero que sea ?sleep 100?:

```
$ echo ?sleep 100? >> pr1
```

Ejecutar pr1 en segundo plano:

```
$ ./pr1 &
```

Ver quien es el propietario del proceso generado:

```
$ ps axu | grep pr1
```

Copiar el fichero pr1 en otro llamado pr2

```
$ cp pr1 pr2
```

Asignarle el bit SUID a pr2

```
$ chmod +s pr2
```

Verificar que se ha asignado el permiso

```
$ ls -la pr*
```

Lanzar ambos procesos en segundo plano

```
$ ./pr1 &; ./pr2 &
```

Comprobar los propietarios de ambos procesos

```
$ ps axu | grep ?pr[12]?
```

Shell

La shell es el intérprete de órdenes que utiliza Linux, bueno Unix en general. No hay una sola shell, sino que cada usuario puede elegir la que quiera en cada momento. Además las shells funcionan como lenguajes de programación de alto nivel, lo que se estuaciará en un capítulo posterior.

En principio hay dos familias de shell, las shell de Bourne y las shell C. Dentro de cada una

de ellas hay diferentes vesiones, como por ejemplo ksh (Korn shell) o bash (Bourne Again Shell),. Nosotros en Linux vamos a usar esta última, aunque gran parte de las cosas son aplicables a otras shell.

Definiciones

Variables de shell

Una variable de shell es una zona de almacenamiento de información con un nombre que la identifica. Para no liarnos mucho, es similar a un variable en lenguaje C. Podemos asignar un valor a una variable mediante el operador ?=?. Si la variable no existe el operador ?=? la crea. No se debe dejar espacio entre el nombre de la variable y el operador de asignación (=).

Para usar el valor de una variable es necesario anteponerle el símbolo ?$?.

Ejemplos:

```
$ AA=Hola que tal
$ echo AA
$ echo $AA
```

Como norma de estilo, se suelen usar letras mayúscular para definir los nombres de las variables.

Variables de entorno

Son variables que tienen un significado propio para la shell o algún otro programa. Ciertos programas leen el contenido de las variables de entorno par mosificar su comportamiento, entre ellos la propia shell.

Entre la variables de entorno más importantes podemos citar:

- PATH Indica la ruta de búsqueda de programas ejecutables. Está constituida por una lista de directorios separados por ?:?. El directorio actual, de forma predeterminada, no viene incluida en PATH.
- PS1 Especifica el indicador del sistema. Lo habitual es que PS1 sea ?$? para usuarios normales y ?#? para root.
- PS2 Especifica el indicador secundario del sistema. Aparece cuando no se ha completado una orden.
- LANG Especifica el lenguaje que se aplica al usuario. Para español se utiliza ?es?.

- LC_ALL Contiene el idioma y se utoiliza para usar los valores locales como mensajes del sistema, símbolo monetario, formato de fecha, formato de números decimales y otras características.
- TERM Indica el tipo de teminal que se está utilizando. Por ejemplo, si estamos en la consola el valor de TERM será ?linux?, para usar las carácterísticas del teclado. Si entramos por telnet desde w9x entonces probablemente tengamos que poner vt100.
- EDITOR Especifica el editor por omisión del sistema. Lo habitual en los sistema Unix es que el editor por omisión sea ?vi?.
- DISPLAY Especifica qué equipo muestra la salida que se efectúa en modo gráfico. Ese equipo deberá tener un servidor gráfico.
- LD_LIBRARY_PATH Esta variable se utiliza para definir rutas alternativas de búsqueda de para bibliotecas de funciones del sistema.
- PWD Contiene el directorio de trabajo efectivo.

Con la orden env (environment) podemos comprobar el valor de las variables de entorno del sistema. Para modificarlas basta asignarle un nuevo valor.

Ejemplo:

```
$ env
PWD=/home/usuario/public_html/docs/shell
LTDL_LIBRARY_PATH=/home/usuario/.kde/lib:/usr/lib
HOSTNAME=www.bdat.com
LD_LIBRARY_PATH=/home/usuario/.kde/lib:/usr/lib
QTDIR=/usr/lib/qt-2.2.2
CLASSPATH=./:/usr/lib/jdk118/lib/classes.zip
LESSOPEN=|/usr/bin/lesspipe.sh %s
PS1=[\u(EN)\h\W]$
KDEDIR=/usr
BROWSER=/usr/bin/netscape
USER=usuario
MACHTYPE=i386-redhat-linux-gnu
LC_ALL=es_ES
MAIL=/var/spool/mail/usuario
EDITOR=joe
LANG=es
COLORTERM=
DISPLAY=:0
LOGNAME=usuario
SHLVL=4
LC_CTYPE=iso-8859-1
SESSION_MANAGER=local/www.bdat.com:/tmp/.ICE-unix/965
SHELL=/bin/bash
HOSTTYPE=i386
OSTYPE=linux-gnu
```

```
HISTSIZE=100
HOME=/home/usuario
TERM=xterm
PATH=/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/usr/lib/jdk118/bin:./:/sbin:/usr/sbin:/usr/local/sbin:/opt/netscape
GROFF_TYPESETTER=latin1
LESSCHARSET=latin1
```

Ficheros ejecutables

Un fichero ejecutable es un fichero que tenga asignado el permiso de ejecución.

Los ficheros ejecutables pueden ser de dos tipos:

Binarios, cuando son resultado de la compilación de un programa, por ejemplo en C.

Programas de shell, llamados scripts o guiones. Son ficheros de texto que contienen órdenes de shell y son interpretados por una shell.

Como ya dije anteriormente, un sistema unix reconoce un fichero como ejecutable por disponer del permiso de ejecución activo, no por ningún tipo de extensión en el nombr del fichero.

Shell y subshell

Podemos abrir una nueva shell simplemente ejecutando el fichero binario que contiene la shell.

También, cuando ejecutamos un script se abre una nueva shell que es la encargada de ir interpretando las diferentes órdenes y de mantener el valor de las variables definidas. Esta subshell tiene como tiempo de vida el tiempo de ejecución del script. Además esta subshell hereda el valor de parte de las variables de entorno, pero en su propio espacio de memoria, por lo que las modificaciones de cualquier variable de la nueva shell no tendrá repercusión en la shell padre.

Cerrar una shell

Para cerrar una shell usamos la orden exit. Esta orden termina la ejecución de la shell y vuelve a la shell padre. Si ejecutamos la orden exit sobre la shell inicial que abrimos al entrar al sistema (llamada login shell) entonces terminamos la sesión de trabajo.

Variables exportadas

En el apartado anterior, veíamos que una subshell hereda parte de las variables definidas en la la shell padre. Para que las shell hijas de una dterminada shell hereden una variable es necesario indicarlo explícitamente con la orden export. Por ejemplo, si queremos que el valor

de la variable EDITOR pase a las subshell que sean hijas de la shell activa, tendremos que poner

```
$ export EDITOR
```

En la shell bash, podemos realizar en una sola operación la asignación de una valor y exportarla, por ejemplo

```
$ export EDITOR=vi
```

No todas las shell permiten esta operación

Ejemplos:

Asignar la palabra contenido a una variable llamada AA

```
$ AA=contenido
```

Mostrar el contenido de la variable en pantalla.

```
$ echo $AA
```

Abrir una shell hija de la anterior.

```
$ /bin/bash
```

Mostrar de nuevo el contenido de la variable

```
$ echo $AA
```

Salir de la subshell activa

```
$ exit
```

Mostrar de nuevo el contenido de la variable

```
$ echo $AA
```

Exportar la variable

```
$ export AA
```

Abrir una shell hija de la anterior.

```
$ /bin/bash
```

Mostrar de nuevo el contenido de la variable

```
$ echo $AA
```

Las comillas en la shell

En las operaciones con la shell distinguimos tres tipos de comillas con distintas funcionalidades: las comillas dobles, las comillas simples y las comillas invertidaa, el acento grave francés. A continuación describimos las funcdiones:

' Engloban un literal. La shell no trata de interpretar nada de lo que haya comprendido entre estas comillas.

? La shell expande las variables que haya especificadas sustituyendo su nombre por el contenido. Para imprimir un $ es necesario protegerlo con una \.

` La shell intenta ejecutar lo que haya comprendido entre estas comillas.

Ejemplo:

Asignamos una valor a una variable:

```
$  AA=?ls -la?
```

y observamos la diferencia entre:

```
echo '$AA'
echo "$AA"
echo `$AA`
```

En bash podemos sustituir las comillas invertidas por el operador $().

Personalización de la shell

Cada vez que se inicia una shell, se lee un fichero de configuración. No es un fichero complejo, es simplemente un fichero con ódenes que se se ejecutan automáticamente cada vez que se inicia una nueva shell.

Diferentes shell utilizan diferentes ficheros de configuración. Las shell C suelen llamar a este fichero .login. Las shell de Bourne suelen llamar a este fichero .profile. En la shell bash, además del .profile, tenemos también el fichero .bashrc.

En este fichero vamos a ejecutar las órdenes y asiganar valores a variables necesarios para adaptar la shell a nuestras necesidades. En general, todo aquéllo que queramos que se ejecute cada vez que entremos al sistema. Por ejemplo, nos puede interesar añadir a este

fichero una línea como:

```
$ export LANG=es
```

para hacer que nuestro idioma predeterminado sea el español.

También podemos poner . (directorioactivo) en la ruta de búsqueda de programas ejecutables:

```
$ export PATH=$PATH:.
```

La shell bash también usa el fichero .inputrc para configurar el teclado.

El editor vi

Intoducción

El editor vi (visual) es el programa de edición común a todos los sistemas Unix. Aún cuando es posible utilizar otros editores, es necesario conocer su funcionamiento en determinadas situaciones críticas.

Modos de trabajo

El editor vi tiene dos modos de trabajo:

- *Modo edición*: En este modo se puede introducir texto en el fichero que estamos editando. Para salir del modo inserción tenemos que pulsar la tecla Esc.
- *Modo orden*: En este modo vi acepta órdenes. Las órdenes son letras con algún posible arguemento.

Terminales

Para el correcto funcionamiento del vi tenemos que asegurarnos que las características de la terminal son las adecuadas. Por ejemplo, si trabajamos con un terminal vt100 deberíamos poner:

```
$ TERM=vt100
$ export TERM
```

La línea inferior de la pantalla la usa vi como línea de estado, para escribir algunos mensajes y para escribir las órdenes.

Salir de vi

Para salir de vi hay que estar en modo orden, por lo que se tendrá que pulsar Esc.

Orden	Acción	Orden	Acción
ZZ	sale de vi grabando el fichero	:wq	sale de vi grabando el fichero
:q!	sale de vi sin grabar los cambios		

Introducir texto (ir a modo edición)

Orden	Acción	Orden	Acción
i	inserta texto en la posición del cursor	a	inserta texto después de la posición del cursor
A	inserta texto al final de la línea	o	Crea una nueva línea bajo la actual
O	Crea una nueva línea sobre la actua		

Borrar

Orden	Acción	Orden	Acción
x	borra el carácter sobre el cursor	d0	borra hasta princio de línea
dw	borra hasta el final de la palabra.	dnw	borra hasta el final de la

			palabra n
db	borra hasta el principio de la palabra	dd	borra la línea actual

Desplazamientos

Orden	Acción	Orden	Acción
->, espacio	espacio a la derecha	<-	espacio a la izquierda
w	palabra a la derecha	b	palabra a la izquierda
$	fin de línea	0	principio de línea
return	línea siguiente	j	línea de abajo
k	línea de arriba		

Búsquedas y sustituciones

Orden	Acción	Orden	Acción
/expreg	busca expreg hacia adelante	? expreg	busca expreg hacia atrás
/	repite la última búsqueda hacia adelante	?	repite la última búsqueda hacia atrás
s/buscado/su	sustituye la primera aparición de la palabra		

stitución[/g]	buscado reemplazándola por la parabra sustitición. Si añadimos /g al final, la sustitución es global en todo el documento.		

Otras órdenes

Orden	Acción	Orden	Acción
H	principio de la pantalla	M	mitad de la pantalla
L	final de la pantalla	nG	A la línea n
(	principio de frase	)	fin de frase
{	principio de párrafo	}	fin de párrafo
r	sustituye el carácter del cursor	R	sustituye caracteres
D	borraría hasta el final de línea	:set number	numera las líneas
:set ai	establece el sangrado automático	:!orden	ejecuta el orden de shell
:w	graba el fichero sin salir de vi.		

Ejecución y agrupación de órdenes

Una vez vistas gran parte de las órdenes de usuario, pasamos a ver una de las principales características de un sistema Unix que es la facilidad para agrupar órdenes de distintas formas para realizar tareas complejas. Por un lado en la propia línea de órdenes se pueden especificar órdenes y unir su ejecución de diversas formas.

Hasta ahora hemos visto como combinar órdenes con tubería, como redirigir las salida y

como gestionar procesos en primer y segundo planos. Ahora vamos a ver otra serie de mecanismos para ejecutar órdenes y programas y verificar el estado de conclusión de la orden.

Ejecución consecutiva

Podemos agrupar varias órdenes en una misma línea de ordenes separándolas por ?;?

La agrupación de órdenes separadas por ?;? es útil cuando tenemos que repetir una misma secuencia de órdenes varias veces.

La ejecución de cada una de las órdenes se realiza cuando ha concluido la anterior, e independiente de que el resultado haya sido correcto o no.

Esto nos puede resultar útil para demorar la ejecución de una o varias órdenes un determinado tiempo, por ejemplo

```
$ sleep 300 ; ps axu
```

y la orden ps axu se ejecutaría a los 300 segundos.

Ejecución condicional

Otra situación algo más elaborada que la anterior es ejecutar una orden condicionada a la terminación correcta o no de una orden previa.

Esta funcionalidad nos la proporcionan los operadores ?&&? y ?||?.

Operador &&

El primer operador, ?&& ? separa dos órdenes de forma que la que tiene a la derecha sólo se ejecuta cuando la de la izquierda termina correctamente, es decir

```
orden1 && orden2
```

orden2 sólo se ejecutará si orden1 terminó sin ningún error.

Por ejemplo, queremos ejecutar la orden cat fichero sólo si existe fichero; entonces tendremos que buscar una orden que termine con un error si no existe fichero, por ejemplo ls fichero y condicionar la ejecución de cat fichero a esta:

```
$ ls fichero && cat fichero
```

Otro ejemplo, para compilar los controladores de dispositivos de linux e instalarlos, lo podemos hacer como:

```
make module && make modules_install
```

es decir instalará los controladores sólo si ha conseguido compilarlos correctamente.

Operador ||

El segundo operador, ?|| ? tiene un comportamiento similar al anterior, separa dos órdenes de forma que la que tiene a la derecha sólo se ejecuta cuando la de la izquierda termina incorrectamente, es decir

```
orden1 || orden2
```

orden2 sólo se ejecutará si orden1 terminó con algún error.

Por ejemplo si no existe fichero queremos crearlo vacía y si existe no hacemos nada. Igual que en el ejemplo anterior, buscamos una orden que termine con un error si no existe fichero y condicionamos la ejecución de la orden touch fichero al error de la orden previa:

```
$ ls fichero || touch fichero
```

También, al igual que en el ejempo anterior podríamos hacer que si el proceso make modules falla, se borraran todos los ficheros temporales que se crean:

```
make modules || rm -r *.o
```

Ejecución simultánea

Otra posibilidad de ejecución también posible es lanzar varios procesos simutáneamente en segundo plano; basta escribir uno a continuación de otro en la línea de órdenes separados por ?&?. Este es el símbolo que se utiliza para indicar que el proceso se tiene que ejecutar en segundo plano, pero también actúa como separador para la ejecución de distintas órdenes.

por ejemplo :

```
[pfabrega(EN)port pfabrega]$ sleep 10 & sleep 20 & sleep 15 &
[pfabrega(EN)port pfabrega]$ ps axu
USER       PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
root         1  0.1  0.2  1408  540 ?        S    22:19   0:04 init [3]
...
pfabrega  1263  0.3  0.2  1624  516 pts/4    S    23:24   0:00 sleep 10
pfabrega  1264  0.0  0.2  1624  516 pts/4    S    23:24   0:00 sleep 20
```

```
pfabrega  1265  0.0  0.2  1624  516 pts/4    S    23:24   0:00 sleep 15
pfabrega  1266  0.0  0.3  2732  848 pts/4    R    23:24   0:00 ps axu
```

Agrupando con paréntesis

Podemos organizar la ejecución de varias órdenes agrupándolas convenientemente mediante paréntesis para modificar el orden predeterminado de ejecución. En primer lugar actúan los ; después los & y la ejecución es de izquierda a derecha. Para las ejecuciones condicionales se tiene en cuenta el valor de la variable $? que se fija por la última orden que se ejecuta.

Para alterar esta forma de ejecución podemos utilizar los paréntesis. En realidad los paréntesis fuerzan una nueva subshell para ejecutar las órdenes correspondientes.

Esta característica puede ser interesante en diferentes situaciones:

Quemos enviar una secuencia de órdene s a segundo pláno

```
(mkdir copiaseg; cp -r ./original/* ./copiaseg; rm -r ./original~; rm -r ./original.tmp) &
```

Resultado de la ejecución de una orden

Es habitual necesitar almacenar el resultado de la ejecución de una orden en una variable en lugar de que se dirija a la salida estándar o simplemente ejecutar como orden el resultado de otra orden.

Comillas invertidas ` `

Las comillas invertidas consideran una orden lo que tengan dentro y lo ejecutan devolviendo el resultado como líneas de texto a la shell. Por ejemplo:

```
$ A="ls /bin"
$ `echo $A`
arch        consolechars   ed          igawk     mount          rpm         tar
ash         cp             egrep       ipcalc    mt             rvi         tcsh
ash.static  cpio           ex          kill      mv             rview       touch
awk         csh            false       ln        netstat        sed         true
basename    date           fgrep       loadkeys  nice           setserial   umount
bash        dd             gawk        login     nisdomainname  sfxload     uname
bash2       df             gawk-3.0.6  ls        ping           sh          usleep
bsh         dmesg          grep        mail      ps             sleep       vi
cat         dnsdomainname  gtar        mkdir     pwd            sort        view
chgrp       doexec         gunzip      mknod     red            stty        ypdomainname
chmod       domainname     gzip        mktemp    rm             su          zcat
chown       echo           hostname    more      rmdir          sync
```

También podríamos haber puesto:

```
$ A="ls /bin"
$ B=`$A`
$ echo $B
arch ash ash.static awk basename bash bash2 bsh cat chgrp chmod chown consolechars
cp cpio csh date dd df dmesg dnsdomainname doexec domainname echo ed egrep ex false
fgrep gawk gawk-3.0.6 grep gtar gunzip gzip hostname igawk ipcalc kill ln loadkeys
login ls mail mkdir mknod mktemp more mount mt mv netstat nice nisdomainname ping
ps pwd red rm rmdir rpm rvi rview sed setserial sfxload sh sleep sort stty su sync
tar tcsh touch true umount uname usleep vi view ypdomainname zcat
```

El operador $()

La shell bash proporciona el operador $() similar a las comillas invertidas. Ejecuta como orden los que haya entre paréntesis y devuelve su resultado. El mecanismo de funcionamiento es idéntico, con la ventaja de poder anidar operadores.

Programas de shell

Además de las anteriores posibilidades también se pueden agrupar una serie de órdenes en un fichero de texto que se ejecutarán consecutivamente siguiendo el flujo determinado por órdenes de control similares a cualquier lenguaje de programación. Estos ficheros se conocen como scripts, guiones o simplemente programas de shell. A las órdenes agrupadas en ficheros también se le aplican todas las características descritas anteriormente. No olvidemos que para un sistema unix, una línea leída de un fichero es idéntica a una línea leída desde el teclado, una serie de caracteres terminado por un carácter de retorno de carro.

Cualquier forma de ejecución que se pueda dar en la línea de órdenes también se puede incluir en un fichero de texto, con lo que facilitamos su repetición. Y si por último añadimos las estructuras que controlan el flujo de ejecución y ciertas condiciones lógicas, tenemos un perfecto lenguaje de programación para administrar el sistema con toda facilidad. Un administrador que sabe cual es su trabajo cotidiano, realizar copias de seguridad, dar de alta o baja usuarios, comprobar que los servicios están activos, analizar log de incidencias, configurar cortafuegos, lanzar o parar servicios, modificar configuraciones, etc., normalmente se creará sus script personalizados. Algunos los utilizará cuando sea necesario y para otros programará el sistema para que se ejecuten periódicamente.

La programación en shell es imprescindible para poder administrar un sistema Unix de forma cómoda y eficiente.

Vemos un primer ejemplo:

```
    #!/bin/bash
```

echo Hola Mundo

Puestas estas dos línea en un fichero de texto con permiso de ejecución, al ejecutarlo escribiría en pantalla ?Hola Mundo?. La primera línea, como veremos con posterioridad, indica qué shell es la que interpreta el programa de shell.

subshell

Para la ejecución de programas de shell, la shell se encarga de interpretar unas órdenes en unos casos o de lanzar el programa adecuado en otros casos. En general, cuando lanzamos la ejecución de un conjunto de órdenes agrupadas en un programa de shell, se abre una nueva shell (subshell hija de la anterior) que es la encargada de interpretar las órdenes del fichero. Una vez concluida la ejecución esta subshell muere y volvemos a la shell inicial. Esto es importante tenerlo presente para saber el comportamiento de los programas. Por ejemplo, los cambios hechos en las variables de shell dentro de un programa no se conservan una vez concluida la ejecución.

Vamos a ilustrar este comportamiento con nuestro primer ejemplo de programa de shell:

Creamos un programa en un fichero de texto, lo ejecutamos, comprobamos que se crea una nueva shell y que los cambios en las variables hechos dentro del programa no se mantienen una vez concluido. Editamos un fichero llamado ?pruebashell? con el siguiente contenido:

```
echo ?******** GUION ********?
echo ?el valor previo de VAR es ** $VAR **?
VAR=?valor asignado dentro del guion?
echo ?Ahora VAR vale ** $VAR **?
ps
echo ?******** FIN DEL GUION ********?
```

Con este guion mostramos el valor previo de una variable llamada VAR, le asignamos un valor nuevo y también los mostramos. Para verificar que se lanza una nueva shell mostramos la lista de procesos con la orden ps.

Una vez editado el fichero tendremos que asignarle el permiso de ejecución

```
$ chmod u+x pruebashell
```

después asignamos un una valor a la variable VAR para comprobar como cambia. Además tendremos que exportarla par que la shell hija pueda heredarla:

```
$ export VAR=?valor previo?
```

Ahora mostramos la lista de procesos para ver cuantas shell tenemos abiertas:

```
$ ps
o bien
$ ps |wc -l
```

y a continuación ejecutamos el guion ?pruebashell?:

```
$ ./pruebashell
```

y volvemos a mostrar el contenido de la variable:

```
$ echo $VAR
```

Podremos observar como aparece una shell más. Si la variable VAR está exportada veremos como muestra el valor que asignamos antes de ejecutar el guion y como muestra el que le asignamos dentro del guion. Y al final, al mostrar la variable VAR, observamos como nos muestra el valor que tenía antes de ejecutar el guion; el guion no ha modificado la variable.

Este mismo comportamiento se puede aplicar a la orden cd. Veamos el siguiente ejemplo, un simple script que cambia de directorio.

Editamos el fichero llamado ?cambia? con el siguiente contenido:

```
echo ?cambiando de directorio?
cd /tmp
echo ?estamos en:?
pwd
```

Es decir, el script simplemente cambia al directorio /tmp.

Una vez editado le asignomos el permiso de ejecución

```
$ chmod u+x cambia
```

Ahora mostramos nuestro directorio activo

```
$ pwd
```

ejecutamos el script

```
$ ./cambia
```

y volvemos a comproba nuesto directorio activo

```
$ pwd
```

y observamos como estamos situados en el mismo directorio que antes de la ejecución del guion.

¿Por qué ocurre todo esto?, Porque todos los cambios se realizan en la subshell que ha interpretado el guion.

Comentarios y continuaciones de línea

Para añadir un comentario a un programa de shell se utiliza el carácter #. Cuando la shell interprete el programa ignorará todo lo que haya desde este carácter hasta el final de la línea.

Por otro lado si nos vemos obligados a partir un línea en dos o más, tendremos que finalizar cada línea de texto inconclusa con el carácter \. De esta forma la shell las verá todas ellas como si se tratara de una única línea.

Parámetros posicionales

En un programa de shell definen unas variables especiales, identificadas por números, que toman los valores de los argumentos que se indican en la línea de órdenes al ejecutarlo. Tras el nombre de un script se pueden añadir valores, cadenas de texto o números separados por espacios, es decir, parámetros posicionales del programa de shell, a los que se puede acceder utilizando estas variables.

La variable $0 contiene el parámetro 0 que es el nombre del programa de shell.

Las variables $1, $2, $3, $4, ... hacen referencia a los argumentos primero, segundo, tercero, cuarto, ... que se le hayan pasado al programa en el momento de la llamada de ejecución.

Por ejemplo, si tenemos un programa de shell llamado parametros con el siguiente contenido:

```
echo $0
echo $1
echo $2
echo $3
```

al ejecutarlo

```
$ parametros primero segundo tercero
prametros
primero
segundo
tercero
```

Modificación de los parámetros posicionales

Durante la ejecución de un programa de shell podría interesarnos modificar el valor de los parámetros posicionales. Esto no lo podemos hacer directamente, las variables 1, 2, ... no están definidas como tales. Para realizar estos cambios tenemos que utilizar la orden set. Esta orden asigna los valores de los parámetros posicionales a la shell activa de la misma forma que se hace en la línea de órdenes al ejecutara un programa; hay que tener en cuenta que no los asigna individualmente, sino en conjunto.

Por ejemplo

```
$ set primero segundo tercero
$ echo $1
primero
$ echo $2
segundo
$ echo $3
tercero
```

La sentencia shift

La sentencia shift efectúa un desplazamiento de los parámetros posicionales hacia la izquierda un número especificado de posiciones. La sintaxis para la sentencia shift es:

```
$ shift n
```

donde n es el número de posiciones a desplazar. El valor predeterminado para n es 1. Hay que observar que al desplazar los parámetros hacia la izquierda de pierden los primeros valores, tantos como hayamos desplazado, al superponerse los que tiene a la derecha.

Por ejemplo:

```
$ set uno dos tres cuatro
$ echo $1
uno
$ shift
$ echo $1
dos
$ shift
$ echo $1
tres
$ shift
$ echo $1
cuatro
```

Operador {}

Hemos visto la forma de acceder a los diez primeros parámetros posicionales, pero para acceder a parámetros de más de dos dígitos tendremos que usar una pareja { } para englobar el número.

```
$ echo ${10}
$ echo ${12}
```

El operador { } también se usa para delimitar el nombre de las variables si se quiere utilizarla incluida dentro de un texto sin separaciones:

```
$DIR=principal
$ DIRUNO=directorio
$ UNO=subdirectorio
$ echo $DIRUNO
directorio
$ echo ${DIR}UNO
principalUNO
```

Variables predefinidas

Además de las variables de shell propias del entorno, las definidas por el usuario y los parámetros posicionales en un shell existen otra serie de variables cuyo nombre está formado por un carácter especial, precedido por el habitual símbolo $.

Variable $*

* La variable $* contiene una cadena de caracteres con todos los parámetros posicionales de la shell activa excepto el nombre del programa de shell. Cuando se utiliza entre comillas dobles se expande a una sola cadena y cada uno de los componentes está separado de los otros por el valor del carácter separador del sistema indicado en la variable IFS. Es decir si IFS tiene un valor ?s? entonces "$*" es equivalente a "$1s$2s...". Si IFS no está definida, los parámetros se separan por espacios en blanco. Si IFS está definida pero tiene un contenido nulo los parámetros se unen sin separación.

Variable $(EN)

(EN) La variable $(EN) contiene una cadena de caracteres con todos los parámetros posicionales de la shell activa excepto el nombre del programa de shell. La diferencia con $* se produce cuando se expande entre comillas dobles; $(EN) entre comillas dobles se expande en tantas cadenas de caracteres como parámetros posicionales haya. Es decir ?$(EN)? equivale a "$1" "$2" ...

Variable $#

Contiene el número de parámetros posicionales excluido el nombre del probrama de shell. Se suele utilizar en un guion de shell para verificar que el número de argumentos es el correcto.

Variable $?

? Contiene el estado de ejecución de la última orden, 1 para una terminación con error o 0 para una terminación correcta. Se utiliza de forma interna por los operadores || y && que vimos con anterioridad, se utilizar por la orden test que vermos más adelante y también la podremos usar explícitamente.

Variable $$

$ contiene el PID de la shell. En un subshell obtenida por una ejecución con (), se expande al PID de la shell actual, no al de la subshell. Se puede utilizar para crear ficheros con nombre único, por ejemplo $$.tmp, para datos temporales.

Variable $!

! Contiene el PID de la orden más recientemente ejecutada en segundo plano. Esta variable nos puede ayudar a controlar desde un guion de shell los diferentes procesos que hayamos lanzado en segundo plano.

Ejemplos

Vemos algunos ejemplos a continuación:

```
$ set a b c d e
$ echo $#
5
$ echo $*
a b c d e
$ set "a      b" c d
$ echo $#
3
$ echo $*
a     b c d
```

Otro ejemplo, si tenemos el script llamado ejvar1 con el siguiente contenido

```
ps
echo ? el PID es $$?
```

al ejecutarlo

```
$ ./ejvar1
  PID TTY          TIME CMD
  930 pts/3    00:00:00 bash
 1011 pts/3    00:00:00 bash
 1012 pts/3    00:00:00 ps
el PID es 1011
```

y vemos como muestra el PID de la shell que ejecuta el script.

Como el PID del prodceso es único en el sistema, este valor puede utilizarse para construir nombres de ficheros temporales únicos para un proceso. Estos ficheros normalmente se suelen situar en el directorio temporal /tmp.

Por ejemplo:

```
miproctemp=/tmp/miproc.$$
. . . .
rm -f $miproctemp
```

Uso de valores predeterminados de variables

Además de las asignaciones de valores a variables vista con anterioridad, que consistía en utilizar el operador de asignación (=), podemos asignarle valores dependiendo del estado de la variable.

Uso de variable no definida o con valor nulo

Cuando una variable no está definida, o lo está pero contiene un valor nulo, se puede hacer que se use un valor predeterminado mediante la siguiente la expresión:

```
$ {variable:-valorpredeterminado}
```

Esta expresión devuelve el contenido de variable si está definida y tiene un valor no nulo. Por ejemplo si la variable resultado inicialmente no esta definida:

```
$ echo ${resultado}
$ echo "E1 resultado es: {resultado:-0}"
E1 resultado es: O
$ resultado=1
```

```
$ echo "E1 resultado es: ${resultado:-0}"
E1 resultado es: 1
```

A los parámetros posicionales podemos acceder como:

```
{$1: -o}
```

Uso de variable no definida

En algunas ocasiones interesa utilizar un valor predeterminado sólo en el caso de que la variable no esté definida. La expresión que se utiliza para ello es algo diferente de la anterior:

```
${variable- valorpredeterminado}
```

Consultar el ejmplo anterior.

Uso de variable definida o con valor nulo

Existe una expresión opuesta a la anterior. En este caso, si la variable está definida y contiene un valor no nulo, entonces en vez de usarse dicho valor, se utiliza el que se especifica. En caso contrario, el valor de la expresión es la cadena nula. La expresión para esto es:

```
$ {variable: +valorpredeterminado}
```

Por ejemplo:

```
$ resultado=10
$ echo ${resultado:+5}
5
$ resultado=
$ echo ${resultado:+30}
$
```

Uso de variable no definida

En algunas ocasiones puede también interesar que el comportamiento anterior sólo suceda cuando la variable no esté definida. La expresión para ello es:

```
$ {variable+valorpredeterminado}
```

Asignación de valores predeterminados de variables

Anteriormente veíamos la forma de utilizar un valor predeterminado de las variable en ciertos casos. Ahora, además de usar el valor predeterminado, queremos asignarlo a la variable.

Asignación a variable o definida o con valor nulo

En este caso no sólo utilizamos un valor predeterminado, sino que en la misma operación lo asignamos. La expresión que se utiliza para ello es la siguiente:

```
${variable:=valorpredeterminado}
```

Si el contenido de variable es no nulo, esta expresión devuelve dicho valor. Si el valor es nulo o la variable no está definida entonces el valor de la expresión es valorpredeterminado, el cual será también asignado a la variable variable. Veamos un ejemplo en el que se supone que la variable resultado no está definida:

```
$ echo ${resultado}
$ echo "El resultado es: ${resultado:=0}"
E1 resultado es: 0
$ echo ${resultado}
0
```

A los parámetros posicionales no se le pueden asignar valores utilizando este mecanismo.

Asignación a variable no definida

Análogo al caso anterior para el caso de que la variable no esté definida. La expresión ahora es:

```
${variable=valorpredeterminado}
```

Mostrar un mensaje de error asociado a una variable

Ahora lo que pretendemos es terminar un script con un mensaje de error asociado al contenido de una variable.

Variable no definida o con valor nulo

En otras ocasiones no interesa utilizar ningún valor por defecto, sino comprobar que la variable está definida y contiene un valor no nulo. En este último caso interesa avisar con un mensaje y que el programa de shell termine. La expresión para hacer esto es:

```
$ {variable : ?Mensaje }
```

Por ejemplo:

```
$ res=${resultado:? "variable no válida"}
resultado variable no valida
```

En el caso de que la variable resultado no esté definida o contenga un valor nulo, se mostrará el mensaje especificado en pantalla, y si esta instrucción se ejecuta desde un programa de shell, éste finalizará.

Variable no definida

Análogo al caso anterior para el caso de que la variable no esté definida. La expresión para ello es:

```
${variable?mensaje}
```

Otras operaciones con variables

Ciertas shell propporcionan unas facilidades que pueden ser útiles para ahorrar código en la programación de guiones de shell, como son la eliminación o extracción de subcadenas de una variable.

Subcadenas de una variable

${variable:inicio:longitud}

Extrae una subcadena de variable, partiendo de inicio y de tamaño indicado por longitud. Si se omite longitud toma hasta el fin de la cadena original.

```
$ A=abcdef
$ echo ${A:3:2}
de
$ echo ${A:1:4}
bcde
$ echo ${A:2}
```

```
cdef
```

Cortar texto al principio de una variable

```
${variable#texto}
```

Corta texto de variable si variable comienza por texto. Si variable no comienza por texto variable se usa inalterada. El siguiente ejemplo muestra el mecanismo de funcionamiento:

```
$ A=abcdef
$ echo ${A#ab}
cdef
$ echo ${A#$B}
cdef
$ B=abc
$ echo ${A#$B}
def
$ echo ${A#cd}
abcdef
```

Cortar texto al final de una variable

```
${variable%texto}
```

Corta texto de variable si variable termina por texto. Si variable no termina por texto variable se usa inalterada.

Vemos un ejemplo:

```
$ PS1=$
$PS1="$ "
$ A=abcdef
$ echo ${A%def}
abc
$ B=cdef
$ echo ${A%$B}
ab
```

Reemplazar texto en una variable

```
${variable/texto1/texto2}
${variable//texto1/texto2}
```

Sustituye texto1 por texto2 en variable. En la primera forma, sólo se reemplaza la primera

aparición. La segunda forma hace que se sustituyan todas las apariciones de texto1 por texto2.

```
$ A=abcdef
$ echo ${A/abc/x}
xdef
$ echo ${A/de/x}
abcxf
```

Evaluación aritmética

En habitual tener que efectuar evaluaciones de expresiones aritméticas enteras durante la ejecución de un script de shell; por ejemplo para tener contadores o acumuladores o en otros casos.

Hasta ahora habíamos visto que esto lo podíamos hacer con expr, pero hay otra forma más cómoda: let

La sintaxis de let es la siguiente:

```
let variable=expresión aritmética
```

por ejemplo

```
let A=A+1
```

En algunas shell incluso podremos omitir la palabra let, aunque por motivos de compatibilidad esto no es aconsejable.

Para evaluar expresiones reales, es decir con coma decimal, tendremos que usar otros mecanismos y utilidades que pueda proporcionar el sistema. En linux disponemos de la orden bc.

Selección de la shell de ejecución

Si queremos que nuestro programa sea interpretado por una shell concreta lo podemos indicar en la primera línea del fichero de la siguiente forma

```
#!/ruta/shell
```

Por ejemplo, si queremos que sea la shell bash la que interprete nuestro script tendríamos que comenzarlo por

```
#!/bin/bash
```

En ciertas ocasiones es interesante forzar que sea una shell concreta la que interprete el script, por ejemplo si el script es simple podemos seleccoinar una shell que ocupe pocos recursos como sh. Si por el contrario el script hace uso de características avanzadas puede que nos interese seleccionar bash como shell.

Lectura desde la entrada estándar: read

La orden read permite leer valores desde la entrada estándar y asignarlos a variables de shell.

La forma más simple de leer una variable es la siguiente:

```
read variable
```

Después de esto, el programa quedará esperando hasta que se le proporcione una cadena de caracteres terminada por un salto de línea. El valor que se le asigna a variable es esta cadena (sin el salto de línea final).

La instrucción read también puede leer simultáneamente varias variables:

```
read varl var2 var3 var4
```

La cadena suministrada por el usuario empieza por el primer carácter tecleado hasta el salto de línea final. Esta línea se supone dividida en campos por el separador de campos definido por la variable de shell IFS (Internal Field Separator) que de forma predeterminada es una secuencia de espacios y tabuladores.

El primer campo tecleado por el usuario será asignado a var1, el segundo a var2, etc. Si el número de campos es mayor que el de variables, entonces la última variable contiene los campos que sobran. Si por el contrario el número de campos es mayor que el de variables las variables que sobran tendrán un valor nulo.

La segunda característica es la posibilidad incluir un mensaje informativo previo a la lectura. Si en la primera variable de esta instrucción aparece un carácter "?". todo lo que quede hasta el final de este primer argumento de read, se considerará el mensaje que se quiere enviar a la salida estándar.

Ejemplo:

```
read nombre?"Nombre y dos apellidos? " apl ap2
```

Evaluación de expresiones: test

La orden test evalúa una expresión para obtener una condición lógica que posteriormente se utilizará para determinar el flujo de ejecución del programa de shell con las sentencias correspondientes. La sintaxis de la instrucción es:

```
test expr
```

Para construir la expresión disponemos una serie de facilidades proporcionadas por la shell. Estas expresiones evalúan una determinada condición y devuelven una condición que puede ser verdadera o falsa. El valor de la condición actualiza la variable $? con los valores cero o uno para los resultados verdadero o falso.

Pasamos a describir a continuación estas condiciones, y tenemos que tener en cuenta que es importante respetar todos los espacios que aquí aparecen.

-f fichero existe el fichero y es un fichero regular -r fichero existe el fichero y es de lectura -w fichero existe el fichero y es de escritura -x fichero existe el fichero y es ejecutable -h fichero es un enlace simbólico. -d fichero existe el fichero y es un directorio -p fichero es una tubería con nombre -c fichero es un dispositivo de carácter -b fichero es un dispositivo de bloques -u fichero tiene puesto el bit SUID -g fichero tiene puesto el bit SGID	-s fichero existe el fichero y su longitud es mayor que 0 -z s1 la longitud de la cadena s1 es cero -n s1 la longitud de la cadena s1 no es cero s1=s2 la cadena s1 y la s2 son iguales s1!=s2 la cadena s1 y la s2 son distintas n1 -eq n2 los enteros n1 y n2 son iguales. n1 -ne n2 los enteros n1 y n2 no son iguales. n1 -gt n2 n1 es estrictamente mayor que n2. n1 -ge n2 n1 es mayor o igual que n2. nl -lt n2 n1 es menor estricto que n2. nl -le n2 n1 es menor o igual que n2.

Estas expresiones las podemos combinar con:

! operador unario de negación

-a operador AND binario

-o operador OR binario

Ejemplos:

```
$ test -f /etc/profile
$ echo $?
$ test -f /etc/profile -a -w /etc/profile
$ echo $?
$ test 0 -lt 0 -o -n "No nula"
$ echo $?
```

La orden test se puede sustituir por unos corchetes abiertos y cerrados. Es necesario dejar espacios antes y después de los corchetes; este suele ser unos de los errores más frecuentes.

Estructura de control

La programación en shell dispone de las sentencias de control del flujo de instrucciones necesarias para poder controlar perfectamente la ejecución de las órdenes necesrias.

Sentencia if

La shell dispone de la sentencia if de bifurcación del flujo de ejecución de un programa similar a cualquier otro lenguaje de programación. La forma más simple de esta sentencia es:

```
if   lista_órdenes
then
   lista_órdenes
fi
```

fi, que es if alrevés, indica donde termina el if.

En la parte reservada para la condición de la sentencia if, aparece una lista de órdenes separados por ";". Cada uno de estos mandatos es ejecutado en el orden en el que aparecen. La condición para evaluar por if tomará el valor de salida del último mandato ejecutado. Si la última orden ha terminado correctamente, sin condición de error, se ejecutará la lista de órdenes que hay tras then. Si esta órden ha fallado debido a un error, la ejecución

continúa tras el if.

Como condición se puede poner cualquier mandato que interese, pero lo más habitual es utilizar diferentes formas de la orden test. Por ejemplo:

```
if [ -f mifichero ]
then
        echo "mifichero existe"
fi
```

Pero también podemos poner:

```
if grep body index.html
then
        echo ?he encontrado la cadena body en index.html?
fi
```

Como en cualquier lenguaje de programación también podemos definir las acciones que se tieenen que ejecutar en el caso de que la condición resulte falsa:

```
if lista_órdenes
then
        lista_órdenes
else
        lista_órdenes
fi
```

Por ejemplo:

```
if [ -f "$1" ]
```

then

```
      pr $1
else
        echo "$1 no es un fichero regular"
fi
```

Cuando queremos comprobar una condición cuando entramos en el else, es decir, si tenemos else if es posible utilizar elif. Vemos un ejemplo:

```
if [ -f "$1" ]
then
        cat $1
elif [ -d "$1" ]
then
        ls $1/*
```

```
else
	echo "$1 no es ni fichero ni directorio"
fi
```

Sentencia while

La sentencia while tiene la siguiente sintaxis:

```
while lista_órdenes
do
	lista órdenes
done
```

La lista de órdenes que se especifican en el interior del bucle while se ejecutará mientras que lista_órdenes devuelva un valor verdadero, lo que significa que la última orden de esta lista termina correctamente.

Vemos un ejemplo:

```
I=0
while [ ${resp:=s} = s ]
do
	I=`expr $I + 1`
	echo $I
	read resp?"Quiere usted continuar(s/n)? "
done
```

Sentencia until

La sentencia until similar a while, es otro bucle que se ejecutará hasta que se cumpla la condición, es decir, hasta que la lista de órdenes termina correctamente. Su formato es el siguiente:

```
until lista_órdenes
do
	lista órdenes
done
```

Sentencia for

La sentencia for repite una serie de órdenes a la vez que una variable de control va tomando los sucesivos valores indicado por una lista de cadenas de texto. Para cada iteración la variable de control toma el valor de uno de los elementos de la lista. La sintaxis de for es la siguientes

```
for variable in lista
do
	lista mandatos
```

```
done
```

lista es una serie de cadenas de texto separadas por espacios y tabuladores. En cada iteración del bucle la variable de control variable toma el valor del siguiente campo y se ejecuta la secuencia de mandatos lista_mandatos.

Ejemplo:

```
for i in $*
do
	echo $i
done
```

y mostraríamos todos los parámetros posicionales.

```
for i in *
do
	echo $i
done
```

y mostraríamos la lista de ficheros del directorio activo.

Sentencias break y continue

La sentencia break se utiliza para terminar la ejecución de un bucle while o for. Es decir el control continuará por la siguiente instrucción del bucle. Si existen varios bucles anidados, break terminará la ejecución del último que se ha abierto.

Es posible salir de n niveles de bucle mediante la instrucción break n.

La instrucción continue reinicia el bucle con la próxima iteración dejando de ejecutar cualquier orden posterior en la iteración en curso.

Sentencia case

La sentencia case proporciona un if múltiple similar a la sentencia switch de C. El formato básico de esta sentencia es el siguiente:

```
case variable in
patrón1)
 lista_órdenes1
;;
patrón2)
lista_órdenes2
;;
```

```
...
patrónN)
lista_órdenesN;;
esac
```

La shell comprueba si variable coincide con alguno de los patrones especificados. La comprobación se realiza en orden, es decir empezando por patrón1 terminando por patrónN. En el momento en que se detecte que la cadena cumple algún patrón, se ejecutará la secuencia de mandatos correspondiente hasta llegar a ";;". Estos dos puntos y comas fuerzan a salir de la sentencia case y a continuar por la siguiente sentencia después de esac (esac es case alrevés).

Las reglas para componer patrones son las mismas que para formar nombres de ficheros, así por ejemplo, el carácter "*" es cumplido por cualquier cadena, por lo que suele colocarse este patrón en el último lugar, actuando como acción predeterminada para el caso de que no se cumpla ninguna de las anteriores. Ejemplo:

```
case "$1" in
  start)
        echo -n "Ha seleccionado start "
        ;;
  stop)
        echo -n "Ha seleccionado stop "
        ;;
  status)
        echo -n "Ha seleccionado stop "
        ;;
  restart)
        echo -n " Ha seleccionado restart "
        ;;
  *)
        echo "No es una opción válida"
        exit 1
esac
```

Terminar un programa de shell (exit)

Como hemos visto, todas las órdenes órdenes tienen un estado de finalización, que por convenio es 0 cuando la ejecución terminó correctamente, y un valor distinto de 0 cuando lo incorrectamente o con error.

Si un programa de shell termina sin errores devolverá un valor cero, pero también es posible devolver explícitamente un valor mediante la sentencia exit. La ejecución de esta sentencia finaliza en ese instante el programa de shell, devolviendo el valor que se le pose como argumento, o el estado de la última orden ejecutada si no se le pasa ningún valor.

Ejemplo:

```
if grep "$1" /var/log/messages
then
	exit 0
else
	exit 10
fi
```

Opciones en un programa de shell: getopts

En los sistema Unix es habitual poner las opciones para ejecutar una orden siguiendo unas normas:

Las opciones están formadas por una letra precedida de un guión.

El orden de las opciones es indiferente, pero suelen preceder a los ntes de los argumentos propiamente.

Cada opción puede llevar uno o más argumentos.

Las opciones sin argumentos pueden agruparse después de un único guión

Para facilitar el análisis de las opciones la shell dispone de la orden getopts. La sintaxis de getopts es la siguiente:

```
getopts cadenaopciones variable [args ...]
```

En cadenaopciones se sitúan las opciones válidas del programa. Cada letra en esta cadena significa una opción válida. El carácter ":" después de una letra indica que esa opción lleva un argumento asociado o grupo de argumentos separados por una secuencia de espacios y tabuladores.

Esta orden se combina con la sentencia while para iterar por cada opción que aparezca en la línea de órdenes en la llamada al programa. En variable se almacena el valor de cada una de las opciones en las sucesivas llamadas. En la variable OPTIND se guarda el índice del siguiente argumento que se va a procesar. En cada llamada a un programa de shell esta variable se inicializa a 1.

Además:

Cuando a una opción le acompaña un argumento, getopts lo sitúa en la variable OPTARG.

Si en la línea de mandatos apar ece una opción no válida entonces asignará "?" a variable.

Veamos un ejemplo:

```
while getopts ab:cd opcion
```

```
do
        case $opcion in
                a) echo "Opción a"
                ;;
                b) echo "Opción b con argumento OPTARG"
                ;;
                c) echo "Opción c"
                ;;
                d) echo ?Opcion d?
                ?) echo "Uso: $0 -acd -b opcion "
        esac
done
shift `expr $OPTIND - 1`
echo "resto de argumentos: $*"
```

Observamos como hemos desplazados todos los argumentos con shift para enerlos disponibles con los parámetros posicionales y descartando las opciones previamente analizadas.

Evaluación de variables: eval

La orden eval toma una serie de cadenas como argumentos:

```
eval [arg1 [arg2] ...]
```

los expande siguiendo las normas de expansión de la shell, separándolos por espacios y trata de ejecutar la cadena resultante como si fuera cualquier orden.

Esta instrucción se debería utilizar cuando:

Pretendemos examinar el resultado de una expansión realizada por la shell.

Para encontrar el valor de una variable cuyo nombre es el valor de otra variable.

Ejecutar una línea que se ha leído o compuesto internamente en el programa de shell.

```
B=ls
A=B
eval \$$A
```

y observamos como ejecuta la orden ls

Funciones

Ciertas shell como bash permiten la declaración de funciones para agrupar bloques código como en un lenguaje de programación convencional.

La forma de declarar un función es

```
function mi_funcion
{
      código de la función
}.
```

Para realizar la llamada a la función sólo tenemos que usar su nombre. Además las funciones pueden tener argumentos en su llamada. No es necesario declarar los parámetros en la declaración de la función, basta usar las variables $1, $2, etc. dentro de la definición de las instrucciones y serán los parametros en su orden correspondiente. Para llamar a una función con argumentos no se usan los habituales paréntesis.

Ejemplos:

```
#!/bin/bash
 function terminar {
    exit 0
 }
 function saludo {
    echo ¡Hola Mundo!
 }
 saludo
 terminar
```

Otro ejemplo:

```
#!/bin/bash
 function terminar {
    exit 0
 }
 function saludo {
    echo $1
 }
 saludo Hola
 saludo Mundo
 terminar
```

En este último ejemplo podemos observar el uso de una función con argumentos, tanto en la delaración como en la llamada.

Trucos de programación en shell

Vemos a continuación una serie de ejemplo genéricos para resolver cuestiones que se presentan en la programación de shell con cierta frecuencia

Script con número variable de argumentos:

Este programa de shell pretende ejecutar una serie de instrucciones para cada uno de los argumentos que se le pasen desde la línea de órdenes.

```
for i in $*
do
	instrucciones
done
```

Por ejemplo, hacemos un script que mueva todos los ficheros pasado como argumento al directorio ./papelera:

```
for i in $*
do
	if [ -f $i ]
	then
		mv $i ./papelera
	fi
done
```

Script para aplicar una serie de órdenes a cada fichero de un directorio

Muy similar al ejemplo anterior, pero sustituimos $* por simplemente * que equivale a todos los ficheros del directorio activo.

```
for i in *
do
instrucciones
done
```

Leer un fichero de texto línea a línea

Es muy habitual tener que procesar todas las línea de un fichero de texto para realizar diferentes operaciones. Vemos una primera forma:

```
while read LINEA
do
```

```
        instucciones por línea
done <fichero
```

En este caso estamos redirigiendo la entrada estándar de la orden read, que es el teclado, por un fichero. Al igual que en el caso del teclado, la lectura se realizará hasta que se encuentre un salto de línea. Observamos como la redirección se realiza tras el final de la sentencia while.

Otra forma posible para hacer esto mismo sería:

```
cat fichero|while read LINEA
do
instrucciones por línea
done
```

Este método difiere ligeramente del anterior, ya que al utilizar una tubería creamos una nueva shell con lo cual puede ocurrir que no se conserven ciertos valores de las variables de shell.

Cambiar una secuencia de espacios por un separador de campos

La salida de ciertas órdenes y ciertos ficheros separan los datos por espacios en blanco. Por ejemplo las órdenes ps o ls -la, o ifconfig.

Si queremos utilizar parte de los datos de las salidas de estas órdenes tendremos que contar las columnas en las que aparece cada dato y cortarlos con cut usando la opción -c. Pero otra opción sería sustituir toda una serie de espacios en blanco por un separador, por ejemplo ?:? o ?;?.

Por ejemplo vamos a ver como sustituir los espacios de la orden ifconfig por ?;?.

La salida normal sería:

```
$ /sbin/ifconfig
eth0      Link encap:Ethernet  HWaddr 00:90:F5:08:37:E4
          inet addr:192.168.1.5  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5103 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:10 Base address:0x3200
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:1726 errors:0 dropped:0 overruns:0 frame:0
```

```
        TX packets:1726 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
```

Ahora usamos sed para sustituir cualquier secuencia de espacios en blanco ([ ][ ]*) por un separador ?;?:

```
$ /sbin/ifconfig |sed "s/[ ][ ]*/;/g"
eth0;Link;encap:Ethernet;HWaddr;00:90:F5:08:37:E4;
;inet;addr:192.168.1.5;Bcast:192.168.1.255;Mask:255.255.255.0
;UP;BROADCAST;MULTICAST;MTU:1500;Metric:1
;RX;packets:0;errors:0;dropped:0;overruns:0;frame:0
;TX;packets:5241;errors:0;dropped:0;overruns:0;carrier:0
;collisions:0;txqueuelen:100;
;Interrupt:10;Base;address:0x3200;
lo;Link;encap:Local;Loopback;
;inet;addr:127.0.0.1;Mask:255.0.0.0
;UP;LOOPBACK;RUNNING;MTU:16436;Metric:1
;RX;packets:1773;errors:0;dropped:0;overruns:0;frame:0
;TX;packets:1773;errors:0;dropped:0;overruns:0;carrier:0
;collisions:0;txqueuelen:0;
```

Ahora podríamos cortar de forma exacta el campo que nos interese, por ejemplo:

```
$ /sbin/ifconfig | sed "s/[ ][ ]*/:/g" | grep inet | cut -f4 -d:
192.168.1.5
127.0.0.1
```

Vemos paso a paso la anterior orden compuesta:

Primero ejecutamos la orden ifconfig

Sustituimos los espacios en blanco por ?:?

Buscamos la línea que contenga la palabra inet

Cortamos el campo 4 usando ?:? como separador.

Ejemplos prácticos

Realizar una copia de seguridad incremental:

```
#!/bin/bash
fecha=`date +%Y%m%d`
mkdir /var/cseg/$fecha
for dir in /home /etc /var/log /var/www /var/lib
do
```

```
tar cfz /var/copias/$fecha/$dir-$fecha.inc.tar.gz $(find $dir -newer /var/copias/ultimo ! -type d)
done
touch /var/copias/ultimo
```

Sustituir una palabra por otra en un conjunto de ficheros del directorio activo

```
for i in $(EN)
do
perl -p -e 's/original/nueva/g' $i >$i.b
mv $i.b $i rm $i.b
done
```

Reemplazar todos los ?#abcdef? qu estén comprendidos entre <table y > por ?#ccccff?.

```
for i in $(EN)
do
perl -p -e 's/<table ([^#])*(#.....)([^>])*>/<table \1#ccccff \3>/g' $i >$i.b
mv $i.b $i rm $i.b
done
```

Ejemplos prácticos

Realizar una copia de seguridad incremental:

```
#!/bin/bash
fecha=`date +%Y%m%d`
mkdir /var/cseg/$fecha
for dir in /home /etc /var/log /var/www /var/lib
do
tar cfz /var/copias/$fecha/$dir-$fecha.inc.tar.gz $(find $dir -newer /var/copias/ultimo ! -type d)
done
touch /var/copias/ultimo
```

Sustituir una palabra por otra en un conjunto de ficheros del directorio activo

```
for i in $(EN)
do
perl -p -e 's/original/nueva/g' $i >$i.b
mv $i.b $i rm $i.b
```

```
done
```

Reemplazar todos los ?#abcdef? qu estén comprendidos entre <table y > por ?#ccccff?.

```
for i in $(EN)
do
perl -p -e 's/<table ([^#])*(#.....)([^>])*>/<table \1#ccccff \3>/g' $i >$i.b
mv $i.b $i rm $i.b
done
```

Prácticas: ejercicios propuestos

¿Que salida ocasionaría cada una de las siguientes órdenes si la ejecutamos consecutivamente?

```
$ set a b c d e f g h i j k l m n
$ echo $10
$ echo $15
$ echo $*
$ echo $#
$ echo $?
$ echo ${11}
```

Explica que realizaría cada una de las siguientes órdenes ejecutadas en secuencia

```
$ A=\$B
$ B=ls
$ echo $A
$ eval $A
```

Mostrar el último parámetro posicional

Pista: $#

Asignar el úlimo parámetro posicional a la variable ULT

Realizar un programa que escriba los 20 primeros números enteros.

Realizar un programa que numere las líneas de un fichero

Realizar un programa que tomando como base el contenido de un directorio escriba cada elemento contenido indicando si es fichero o directorio.

Realizar un programa que muestre todos los ficheros ejecutables del directorio activo.

Modificar el programa anterior para que indique el tipo de cada elemento contenido en el directorio activo: fichero, directorio, ejecutable,...

Ejercicios resueltos sobre ficheros y directorios

Guion de shell que genere un fichero llamado listaetc que contenga los ficheros con permiso de lectura que haya en el directorio /etc:

```
for F in /etc/*
do
if [ -f $F -a -r $F ]
then
echo $F >> listaetc
fi
done
```

Hacer un guion de shell que, partiendo del fichero generado en el ejercicio anterior, muestre todos los ficheros del directorio /etc que contengan la palagra ?procmail?:

```
while read LINEA
do
if grep procmail $L >/dev/null 2>&1
then
echo $L
fi
done <listaetc
```

Hacer un guion de shell que cuente cuantos ficheros y cuantos directorios hay en el directorio pasado como argumento:

```
DI=0
FI=0
for I in $1/*
do
	if [ -f $I ]
	then
		let FI=FI+1
	fi
	if [ -d $I ]
```

```
	then
		let DI=DI+1
	fi
done
```

Hacer un guion de shell que compruebe si existe el directorio pasado como argumento dentro del directorio activo. En caso de que exista, que diga si no está vacío.

```
if [ -d $1 ]
then
	echo ?$1 existe?
	N=$(ls | wc -l)
	if [ $N -gt 0 ]
	then
		echo ?S1 no está vacio, contiene $N ficheros no ocultos?
	fi
fi
```

Hacer un guion de shell que copie todos los ficheros del directorio actual en un directorio llamado csg. Si el directorio no existe el guion lo debe de crear.

```
if [ ! -d csg ]
then
	mkdir csg
fi
cp * csg
```

Hacer un script que muestre el fichero del directorio activo con más líneas:

```
NLIN=0
for I in *
do
	if [ -f $I ]
	then
		N=$(wc -l $I)
		if [ $N -gt $NLIN ]
		then
			NOMBRE=$I
			NLIN=$N
		fi
	fi
done
echo ?$NOMBRE tiene $NLIN lineas?
```

Claves

Como...

- Enlazar dos ficheros
- Ver las diferencias entre dos ficheros
- No permitie que alguien pueda ver un fichero
- No permitie que alguien pueda entrar en un directorio
- Impedir o permitir el acceso a un directorio
- Impedir o permitir la modificación de un fichero
- Comprimir ficheros y otro método para comprimir ficheros
- Comprimir directorios
- Cambiar la clave de acceso
- Ver la lista de procesos activos en el sistema
- Cortar informacion de una linea
- Sustituir una palabra en un fichero
- Buscar una palabra en un fichero
- Continuar la ejecucion de un proceso en segundo plano
- Mostrar un calendario
- Ver la fecha
- Imprimir un fichero
- Traer un proceso a primer plano