Sistemas Operativos Multimedia

Introducción a UNIX (II)

Una vez que se ha estudiado algunas órdenes básicas junto con el manejo de archivos y directorios en la práctica anterior, continuamos con el aprendizaje de conceptos y órdenes útiles para el trabajo con sistemas UNIX. Concretamente se van a estudiar órdenes para la redirección de la E/S, la gestión de procesos y las copias de seguridad.

REDIRECCIÓN DE E/S

Cuando se ejecuta un programa en UNIX, el proceso resultante, usualmente espera una entrada de datos y normalmente produce alguna salida de resultados. En un modo de trabajo interactivo, el usuario espera interactuar con el sistema usando los dispositivos de entrada/salida. En el modo de operación por lotes, el usuario no interacciona con el programa, luego tanto las entradas como las salidas se espera que se realicen sobre archivos.

La solución que UNIX (y la mayoría de los sistemas operativos) adopta es tratar los dispositivos como archivos. Por cada dispositivo conectado al sistema existe un archivo (o varios) al cual el usuario se refiere cuando desea acceder al dispositivo.

En UNIX, existen tres archivos estándar que se refieren a la E/S por defecto que usan los procesos.

- Entrada estándar: (*stdin*) Representa al dispositivo del cual un programa espera leer su entrada.
- Salida estándar: (*stdout*) Representa al dispositivo sobre el cual un programa espera escribir su salida.
- Salida de error o diagnóstico: (*stderr*) Representa al dispositivo sobre el cual un programa escribirá los mensajes de error.

En general, si un programa espera una entrada y no se especifica ningún nombre de archivo, la entrada estándar para este programa se toma como el teclado de la terminal del usuario. Normalmente, el resultado de la ejecución de un programa se lanza sobre la salida estándar que coincide con la pantalla de la terminal del usuario. La salida de error coincide por omisión, con la salida estándar.

REDIRECCIÓN DE LA SALIDA ESTÁNDAR

Si un nombre de archivo está precedido por el símbolo >, la salida estándar de un programa se *redirige* hacia el archivo.

```
$ ls -l > dir.dat
$ cat dir.dat
...
```

Si el archivo especificado no existía antes de la orden, se crea y si ya existía, se reemplazará su contenido.

```
$ ls -l > salida.dat
$ cat salida.dat
$ who > salida.dat
$ cat salida.dat
...
```

Según esto, podemos usar este método para crear un archivo vacío. Así:

```
$ > vacio
```

Para crear un archivo con un determinado contenido, podemos usar la orden `cat` redirigida:

```
$ cat > texto
cat es una orden útil para
generar archivos nuevos.
<Crtl>d
$ cat texto
...
```

Si queremos que la salida de un programa se añada al contenido actual de un archivo, usaremos el símbolo `>>`. Por ejemplo...

```
$ ls -l > salida.dat
$ cat salida.dat
$ echo Ultima linea >> salida.dat
$ cat salida.dat
```

Si queremos que la salida de un programa se "pierda", es decir, no aparezca ni en la pantalla ni se almacene en ningún archivo, podemos usar el dispositivo `/dev/null`, así:

```
$ ls -l > /dev/null
```

REDIRECCIÓN DE LA ENTRADA ESTÁNDAR

Al igual que se puede redirigir la salida estándar, se puede hacer lo propio con la entrada estándar. Para ello usaremos el símbolo <.

```
$ tail -5 < salida.dat
```

En este caso, el proceso asociado a la orden `tail` no actúa sobre un archivo específico, sino que se aplica a la entrada estándar. Como la entrada estándar está redirigida al archivo `salida.dat`, y la salida estándar por defecto está asociada a la pantalla, el resultado de la orden será equivalente al siguiente mandato:

```
$ tail -5 salida.dat
```

La doble redirección de entrada `<< word` se interpreta como sigue: el intérprete de órdenes lee de la entrada estándar hasta una línea que sea la palabra `word`, o el fin de archivo. El documento resultante se convierte en la entrada estándar.

Es interesante comprobar cómo la orden `cat` puede usarse para crear un archivo nuevo, tecleándolo en la entrada estándar y controlando su final por la aparición de determinada cadena de caracteres.

```
$ cat << ”final_archivo” > texto.txt
La orden cat es simple
pero bien usada, puede
servirnos como editor de textos
de emergencia (si no hay otro disponible)
final_archivo
$ cat texto.txt
...
```

SALIDA DE DIAGNÓSTICO

De la misma manera que podemos redirigir la salida estándar, es posible almacenar la salida de error en un archivo usando `2>`.

En el siguiente ejemplo el archivo `noexisto.dat` no existe.

```
$ cat salida.dat noexisto.dat > sal.dat 2> err.dat
$ cat sal.dat
...
$ cat err.dat
...
```

Como se puede suponer, UNIX se refiere a los dispositivos de salida usando los números correspondientes delante de símbolo `>`. Si no se especifica ningún número delante de `>` UNIX toma `1` por omisión. Compruebe que la ejecución de la siguiente orden es equivalente a la anterior.

```
$ cat dir.dat noexisto.dat 1> sal.dat 2> err.dat
...
```

Si se quiere lanzar la salida de error hacia el mismo archivo que se lanza la salida estándar, se usa la nomenclatura `2>&1`. También se puede usar `1>&2` para añadir la salida estándar a la salida de error. Veamos un ejemplo con la orden `time` que ejecuta una orden, que se le pasa como parámetro, y despliega en la salida de error una contabilidad de tiempos sobre su ejecución.

TUBERÍAS

UNIX también permite usar la salida estándar de un proceso como la entrada estándar de otro, tal y como si colocáramos uno a continuación de otro. A una secuencia de órdenes enlazadas de este modo, se le llama tubería (*pipeline*). Para conectar dos procesos con una tubería, usaremos el símbolo `|`

```
$ cat salida.dat | wc -l
...
```

Ahora, la salida estándar de `ls` se usa como entrada estándar de `sort` que es un ejemplo de filtro que permite ordenar la información. Es intuitivo pensar que las tuberías nos permiten construir procesos que operen en un "flujo" de datos. Con el estudio de los "filtros" de UNIX esta posibilidad se convierte en un recurso muy apreciado. Se pueden conectar varios procesos en una misma orden:

```
$ ls -l | sort | more
...
```

Para poder construir una "fontanería" completa, es necesario poder bifurcar el flujo de datos mediante la inserción de una "T". La posibilidad que UNIX ofrece es el uso de la orden `tee`, que permite almacenar su entrada en un archivo y dirigirla hacia su salida. Veamos:

```
$ ls -l | tee dir.dat | sort
...
$ cat dir.dat
...
```

En el ejemplo anterior, la salida del `ls –l` la ha dejado `tee` en el archivo `dir.dat` y a su vez se ha pasado como entrada para que fuera procesada por `sort`. Normalmente, `tee` se aplica para bifurcar aquellas salidas que ha costado bastante producir y que deben ser procesadas varias veces mediante mandatos distintos.

PROCESAMIENTO DE ARCHIVOS

A continuación se describen algunas órdenes que resultan útiles para el trabajo con archivos y que se combinan con el uso de los mecanismos de redirección y tuberías. Estos filtros utilizan algunos archivos de ejemplo para actuar sobre ellos. Inicialmente tales archivos no se encuentran en su directorio de trabajo.

SORT

El mandato clasifica un archivo de texto. La sintaxis es:

```
sort [opciones] nombre...
```

Supongamos que tenemos el archivo `gente` con el siguiente contenido (`nombre+' '+apellido+'  '+código`)

```
$ cat gente
Maryann Clark  101
Bill Williams  100
Hank Parker    114
Sylvia Dawson  110
```

```
Henry Morgan   112
Jack Austen    120
Sally Smith    113
Jane Bailey    121
Charlie Smith  122
Steve Daniels  111
```

Clasificación por defecto. Ordenación alfabética a partir del primer carácter de la línea

```
$ sort gente
Bill Williams     100
Charlie Smith     122
Hank Parker       114
Henry Morgan      112
Jack Austen       120
Jane Bailey       121
Maryann Clark     101
Sally Smith       113
Steve Daniels     111
Sylvia Dawson     110
```

Las líneas se dividen en campos delimitados por blancos. La clasificación puede realizarse utilizando números de campo:

- A partir del segundo campo

```
$ sort --key=2 gente
Jack Austen          120
Jane Bailey          121
Maryann Clark        101
Steve Daniels        111
Sylvia Dawson        110
Henry Morgan         112
Hank Parker          114
Sally Smith          113
Charlie Smith        122
Bill Williams        100
```

-Sólo el segundo campo

```
$ sort --key=2,2 gente
Jack Austen                120
Jane Bailey                121
Maryann Clark              101
Steve Daniels              111
Sylvia Dawson              110
Henry Morgan               112
Hank Parker                114
Sally Smith                122
Charlie Smith              113
Bill Williams              100
```

- A partir del tercer campo

```
   $ sort -b --key=3      gente
   Bill Williams          100
   Maryann Clark          101
   Sylvia Dawson          110
   Steve Daniels          111
   Henry Morgan           112
   Sally Smith            113
   Hank Parker            114
   Jack Austen            120
   Jane Bailey            121
   Charlie Smith          122
```

- Interpretación ASCII

```
$ sort –b --key=3 puntos
Steve Daniels              11
Sally Smith                14
Hank Parker                18
Maryann Clark              18
Bill Williams               2
Jane Bailey                 2
Jack Austen                 3
Henry Morgan                5
Sylvia Dawson               7
Charlie Smith               9
```

- Interpretación numérica

```
$ sort –b --key=3n puntos
Bill Williams  2
Jane Bailey    2
Jack Austen    3
Henry Morgan   5
Sylvia Dawson  7
Charlie Smith  9
Steve Daniels  11
Sally Smith    14
Hank Parker    18
Maryann Clark  18
```

Orden inverso. Opción -r

```
$ sort –b --key=3nr puntos
Hank Parker       18
Maryann Clark     18
Sally Smith       14
Steve Daniels     11
Charlie Smith     9
Sylvia Dawson     7
Henry Morgan      5
Jack Austen       3
Bill Williams     2
Jane Bailey       2
```

SPLIT

El mandato split divide un archivo de texto en partes. La sintaxis es de la forma:

```
split [-líneas] nombre [prefijo]
```

División por defecto. Crea archivos xaa ,xab, xac, xad... con 1000 líneas

```
$ wc -l /etc/termcap
  14625 /etc/termcap
$ split /etc/termcap
$ ls x??
xaa xab ... xao
$ wc -l x??
   1000 xaa
   1000 xab
    ...
    625 xao
  14625 total
```

Indicando el número de líneas

```
$ rm x??
$ split -5000 /etc/termcap
$ ls x??
xaa xab xac
```

Indicando un prefijo diferente

```
$ split /etc/termcap trozo_
$ ls trozo_??
trozo_aa trozo_ab trozo_ac ... trozo_ao
```

CUT

El mandato cut recorta columnas o campos de un archivo. La sintaxis en este caso es como sigue:

```
cut [opciones] nombre ...
```

Cortando columnas (por defecto 1 carácter). Opción -c

```
$ cut –c16,17,18 gente
122
112
...
$ cut –c16-18 gente
$ cut –c16- gente
```

Cortando campos. Opción –f
Por defecto, `cut` asume que los campos se encuentran separados por tabuladores. Como están separados por espacios no realizará nada.

```
$ cut –f1,2 gente
Charlie Smith     122
Henry Morgan      112
...
```

Ahora bien, si se indica que el separador sea el espacio:

```
$ cut -d" " –f1,2 gente
Charlie Smith
Henry Morgan
...
```

Se puede obtener tres archivos a partir del inicial:

```
$ cut –c16- gente > tels
$ cut -d" " –f1 gente > noms
$ cut -d" " -f2 gente > aps
```

PASTE

El mandato une las líneas de varios archivos. Sintaxis

```
paste [opciones] nombre ...
```

Uniendo archivos

```
$ paste noms aps tels
Charlie Smith     122
Henry   Morgan    112
Maryann Clark     101
...
```

Especificando separadores distintos del tabulador. Opción -d

Utilizando el espacio en blanco

```
$ paste -d" " noms aps tels
Charlie Smith 122
Henry Morgan 112
Maryann Clark 101
...
```

Utilizando varios separadores

```
$ paste -d", " noms aps tels
Charlie,Smith 122
Henry,Morgan 112
Maryann,Clark 101
...
```

WC

El mandato wc cuenta el número de líneas (`-l`), palabras (`-w`) y caracteres (`-c`) de un archivo de texto. La sintaxis es la siguiente:

```
wc [opciones] nombre.....
```

```
$ wc gente
   10    30   190 gente
$ wc -l gente
   10 gente
$ wc -lc gente
   10   190 gente
$ wc -wcl gente
   30   190    10 gente
```

TR

Este mandato permite cambiar o traducir los caracteres procedentes de la entrada de acuerdo a reglas que se especifican. El formato general es:

```
tr [opciones] cadena_1 cadena_2
```

Ejemplos de utilización de este mandato son:

- Para cambiar un carácter por otro: por ejemplo, el utilizado como separador entre campos de un archivo (':') con otro (por ejemplo, el tabulador):

```
$ tr '\t' : < puntos
```

- Para cambiar un conjunto de caracteres: para poner en mayúsculas todos los caracteres que aparecen en un archivo:

```
$ tr '[a-z]' '[A-Z]' < puntos
```

- Eliminar los caracteres de control de fin de línea que pueden aparecer al utilizar archivos en formato de texto MS-DOS:

```
$ cat nom_fich_entrada | tr -d '\r' > nom_fich_salida
```

GREP

El mandato **grep** (véase también *fgrep* y *epreg*) permite realizar búsquedas de líneas que contengan texto que identifique a un objetivo o patrón que se especifica. Se pueden utilizar para extraer información de los archivos, buscar líneas que se relacionen con un elemento particular y para localizar archivos que contengan una palabra clave particular.

Los patrones a buscar se pueden realizar con metacaracteres, tanto en las expresiones como en la lista de nombres de archivos. La forma general del mandato es la siguiente:

```
grep [opciones] expresión [archivo] ...
```

Si se quiere buscar más de una palabra (una frase) separadas por espacios en blanco, o se utilizan caracteres comodín, es necesario encerrar la expresión entre comillas.

Algunas de las opciones de este mandato son:

- `-i` La búsqueda no es sensible a mayúsculas/minúsculas
- `-n` Muestra el número de línea donde se ha encontrado la coincidencia.
- `-l` Muestra los nombres de los archivos pero no las líneas.
- `-v` Muestran las líneas donde **no** se produce la coincidencia.

```
$ grep Smith gente
Charlie Smith   122
Sally Smith     113
```

GESTIÓN DE PROCESOS

Las órdenes del usuario y las tareas del sistema, se traducen en la ejecución de procesos, tal como el usuario ha tenido oportunidad de comprobar en la anterior práctica y parte de ésta. Para ello, el sistema ofrece al usuario un *intérprete de órdenes o comandos* que se encarga de lanzar a ejecución dichos procesos.

Para lanzar un proceso, basta que el usuario teclee el nombre del archivo asociado en su terminal. Hasta que no termine la ejecución del proceso no se activará el shell para solicitar más órdenes. Esta forma de ejecutar procesos se denomina secuencial, primer plano o **foreground**.

```
$ date
```

Si se desea ordenar a UNIX que ejecute varios procesos de forma secuencial, podemos invocarlos separados por punto y coma "`;`".

```
$ date;ps;who
```

El proceso `ps` comenzará a ejecutarse cuando termine la ejecución de `date` y `who` cuando lo haga `ps`. A esto se le llama proceso secuencial.

Si pretendemos que varios procesos se ejecuten simultáneamente, podemos invocarlos seguidos del signo &. Por ejemplo, se puede utilizar la orden sleep que hace que la terminal quede bloqueada durante el tiempo especificado en el argumento. En cambio si lo ejecutamos con el signo &, podremos seguir introduciendo órdenes.

```
$ sleep 20 &
[1] 4533
$ date&ps&who
.....
```

Los procesos `ps, date` y `who` se ejecutarán simultáneamente compartiendo el tiempo de CPU y devolviendo el control al shell antes de terminar su ejecución. A esto se le llama proceso no secuencial, segundo plano o **background**.

Al ejecutar un comando en segundo plano, el shell muestra el identificador de trabajo (JID) entre corchetes y el identificador de proceso (PID) del proceso creado. Mediante el JID se permite gestionar procesos (eliminar, suspender, reanudar, etc.).

Para generalizar podemos decir que una *tubería* es una secuencia de órdenes separadas por el símbolo `|`. Una lista de órdenes es una secuencia de una o más *tuberías* separadas por alguno de los símbolos `;`, &, &&, `||`. El intérprete de órdenes interpreta listas de órdenes.

El significado de los símbolos & y `;` ya lo conocemos, veamos los otros. El símbolo && hace que la lista que le sucede se ejecute solo si la que le precede se ejecuta con éxito (devuelve cero). El símbolo `||` hace que la lista que le sucede se ejecute solo si la ejecución de la que le precede ha producido un error (devuelve valor distinto de cero). Los símbolos & y `;` tienen igual precedencia, que es menor que la de && y `||`. Los paréntesis se pueden usar para alterar la precedencia.

```
$ ls|wc&
 .....
```

```
$ ls;wc <who >archivo_que_existe
 .....
$ sleep 5;pwd&
.....
$ (sleep 5;pwd)&
.....
$ cat archivo_que_no_existe && cat archivo_que_existe
.....
$ cat archivo_que_existe && cat archivo_que_no_existe
.....
$ cat archivo_que_no_existe || cat archivo_que_existe
.....
$ cat archivo_que_existe || cat archivo_que_no_existe
```

En lo que sigue, veremos algunas órdenes útiles para manejar los procesos UNIX.

ORDEN PS

UNIX es sistema operativo multiproceso, lo que quiere decir que en un sólo ordenador se pueden ejecutar a la vez varios procesos (del mismo o de diferentes usuarios). Cada comando o programa que se está ejecutando es un proceso. Los procesos se identifican en la máquina UNIX mediante un número llamado **identificador de proceso** (PID).

Para crear un nuevo proceso debe existir otro previo (proceso padre) que se clona así mismo y, posteriormente cambia su imagen en memoria por la del nuevo proceso. El nuevo proceso guarda el **identificador del proceso padre** (PPID)

El intérprete de comandos es un proceso. Cada vez que ejecuta un comando crea un nuevo proceso para ese comando, y espera a que termine para aceptar nuevos comandos. Por lo tanto, cada comando que invocamos es un nuevo proceso.

Existen varios comandos u órdenes que permiten manejar los procesos.

La orden `ps` sirve para listar los procesos que están ejecutándose en un momento determinado. Con él podemos averiguar no sólo qué procesos hay, sino también su PID, su estado y otras propiedades.

```
$ ps
     PID    TTY        TIME  COMMAND
   29437   pts/1  00:00:00  bash
    6816   pts/1  00:00:00  ps
```

Algunas opciones (-f, -e, -u, …) proporcionan información más detallada (ver las páginas de `man`)

```
$ ps -f
...
```

La información principal que suministra `ps` es el PID (identificador del proceso), la terminal desde donde se ha lanzado el proceso (TTY) y el tiempo de CPU que lleva consumido.

```
$ (sleep 10;who)|(sleep 15;wc)&
```

```
$ ps
 ...
$ ps
```

Todos los términos de una tubería se ejecutan en procesos separados. Además, cuando se utilizan paréntesis el shell crea un shell hijo que ejecuta la parte de la orden encerrada entre los paréntesis.

ORDEN KILL

Normalmente no se necesita conocer el PID de un proceso, pero a través de él se puede acceder a un proceso mediante el envío de señales. Se puede considerar una señal como un mensaje con un número.
Cuando se envía una señal con la orden `kill` hay que identificar el proceso destinatario (mediante su PID) y el número de señal a enviar. Cuando un proceso recibe una señal pueden ocurrir dos cosas:

- Si el número de señal era esperado por el proceso, éste efectúa alguna acción en respuesta a la señal, y después continúa con su actividad normal.
- Si el número de señal no era esperado por el proceso, éste termina (muere) de forma inmediata.

Por lo tanto, una de las utilidades del envío de señales es matar procesos. Observe la ejecución de la siguiente secuencia de órdenes.

```
$ (sleep 30;echo Han pasado 30 segundos)&
 [1] 6814
$ ps
    PID TTY          TIME COMMAND
  29437 pts/1  00:00:00 bash
   6814 pts/1  00:00:00 bash
   6815 pts/1  00:00:00 sleep 30
   6816 pts/1  00:00:00 ps
$ kill –9 6814
 ...
```

Para ver las señales disponibles podemos utilizar la opción -l.:

```
$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGIOT       7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     17) SIGCHLD
18) SIGCONT     19) SIGSTOP     20) SIGTSTP     21) SIGTTIN
22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO
30) SIGPWR
 ...
```

Por defecto `kill` utiliza la señal SIGTERM, habitualmente utilizada para abortar procesos. Como -s es la opción más habitual, también se puede omitir, escribiendo el nombre o número de la señal precedido de un guión. Los siguientes ejemplos envían la señal de muerte al proceso cuyo *PID* es el 2543 (todas las órdenes son equivalentes):

```
$ kill -9 2543
$ kill -s 9 2543
```

```
$ kill -SIGKILL 2543
$ kill -s SIGKILL 2543
 ...
```

Normalmente kill se emplea para matar procesos, de ahí su nombre.

ORDEN NOHUP

Otra orden útil para el manejo de procesos en *background* es `nohup`. Como ya hemos visto, al acabar una sesión (con `<Ctrl>-d` o `exit`) todos los procesos que se lanzaron en esa sesión morirán, al ser todos ellos hijos del proceso `sh` que se ejecuta al iniciar la sesión.

Puede ocurrir que nos interese que algún proceso no muera (es decir, simplemente nos interesa que el sistema no envíe la señal SIGHUP a ese proceso) al acabar la sesión al ser su ejecución muy costosa. Esto se hace con `nohup`, pasándole como argumento la línea de mandatos que no deba recibir la señal SIGHUP al terminar el shell. Veamos:

Construyamos primero con `vi`, `joe` o la orden `cat` un sencillo programa UNIX en un archivo que llamaremos `diferido` con el siguiente contenido:

```
sleep 90
who
```

fijemos el permiso de ejecución con:

```
$ chmod +x diferido
```

Comprobemos el funcionamiento de `nohup`.

```
$ nohup diferido > salida &
 [1] 45321
$ ps
...
$ exit
 ...
 ... Volver a entrar
$ ps ax | grep 45321
...
```

ORDEN TOP

`top` es una orden de Linux parecido a ps. Muestra información sobre los procesos activos en la máquina, con la diferencia de que el listado se refresca automáticamente a intervalos regulares de tiempo.

ALMACENAMIENTO Y COMPRESIÓN DE ARCHIVOS

A continuación se describen algunos mandatos que resultarán útiles para archivar o guardar información en un sistema UNIX. Posteriormente se explicará como pueden ser utilizados de cara a realizar copias de seguridad en dispositivos de almacenamiento (disquetes).

TAR

Este mandato permite almacenar varios archivos en uno solo. Su sintaxis consiste en:

tar [opciones] [nombres de archivos ...]

Por ejemplo, el siguiente mandato

```
$ tar cvf backup.tar prueba1
```

permite almacenar todos los archivos del directorio `prueba1` (respetando su estructura) en el archivo `backup.tar`. El primer argumento de la orden (`cvf`) representa alguna de las opciones disponibles. En este caso, la opción `c` indica que se creará un nuevo archivo. La opción `v` permite visualizar los archivos y directorios que se van almacenando, mientras que `f` indica que el siguiente argumento (`backup.tar`) será el nombre utilizado para el nuevo archivo. El resto de argumentos (en el ejemplo están limitados al directorio `prueba1`) consistirán en los archivos o directorios que se pretende archivar.

Si se ejecuta el mandato

```
$ tar xvf backup.tar
```

se extraerá la información procedente del archivo `backup.tar` y se depositará en el directorio actual. Si los archivos han sido almacenados con nombres de ruta absolutos, la operación de extracción utilizará también esos mismos nombres de ruta.

El mandato

```
$ tar tvf backup.tar
```

puede utilizarse para obtener información sobre el contenido del archivo `backup.tar`, sin necesidad de extraerla. De esta forma se pueden consultar los nombres de los archivos, y en qué orden fueron almacenados.

GZIP

A diferencias de algunas utilidades de archivo disponibles en otros sistemas operativos, `tar` no comprime de forma automática la información de los archivos conforme los archiva. Para ello se utilizan mandatos como `gzip`, que permiten comprimir cualquier clase de archivos (no sólo archivos tar). Por ejemplo, el mandato

```
$ gzip backup.tar
```

comprimirá el archivo `backup.tar`, y lo sustituirá por el nombre de archivo `backup.tar.gz`. Es decir, la versión comprimida del archivo original. Para recuperar el contenido original se utilizará el siguiente mandato

```
$ gzip –d backup.tar.gz
```

Otro mandato similar a `gzip` es `compress`. La diferencia consiste en la extensión que se añade al archivo comprimido (`.Z`). El mandato para invertir el resultado de `compress`, es `uncompress`. En el caso que el tamaño del archivo comprimido resulte aún demasiado grande para su manejo se puede recurrir a utilidades como `split` descrita en el apartado de FILTROS.

ZIP/UNZIP

Es una orden específica de Linux similar a `gzip` (consultan el `man`).

USO DE DISQUETES PARA COPIAS DE SEGURIDAD

Los disquetes son el medio habitual en UNIX para realizar copias de seguridad, sobre todo si no se dispone de algún dispositivo de almacenamiento con mayor capacidad. En este punto habrá que distinguir varios métodos para realizar las copias de seguridad, en función de formato que tenga el disquete.

En primer lugar, si el disquete está formateado en MS-DOS, existe la opción en algunos sistemas UNIX que permite su acceso y uso para almacenamiento. Para ello se utilizará un mandato como

```
$ mount /dosA
```

Esta orden `mount` permite que el dispositivo disquete (`/dev/fd0`) sea accesible a partir del directorio `/dosA`. Por tanto órdenes como

```
$ ls /dosA
...
```

proporcionarán el contenido del disquete, una vez montado. Asimismo podrán realizarse operaciones de copia o similar, tal como se haría con otros directorios en UNIX. Es importante que al final de la sesión de trabajo con el disquete, se aplique la operación de desmontaje consistente en

```
$ umount /dosA
```

En caso contrario, ello puede suponer que las operaciones realizadas no queden reflejadas en el contenido del disquete. También hay que indicar que la operación de montaje puede tener una versión abreviada, según la configuración del sistema presente en el laboratorio.

La opción alternativa consiste en trabajar con disquetes que tengan el mismo formato del sistema UNIX, que se esté utilizando. En el caso de Linux, resulta habitual trabajar con el formato denominado `ext2`. El procedimiento será montar el dispositivo disquete en un punto del sistema de archivos UNIX, de forma parecida a como se hacía con un disquete MS-DOS.

```
$ mount /floppy
```

A veces las órdenes mount y umount están protegidas para que únicamente las pueda ejecutar el superusuario.

Otra opción es emplear las órdenes de MTOOLS que permiten trabajar con un disquete en formato MSDOS sin necesidad de montarlo.

La orden `mdir` (MS-DOS directory) sirve para visualizar el contenido de un disquete (o en general de cualquier disco) de MS-DOS/Windows sin necesidad de montar ni desmontar la unidad de disquetes (la unidad no debe estar montada para que funcione). El listado se muestra con el mismo aspecto a como lo hace el comando dir de MS-DOS/Windows.

```
$ mdir a:/so1/*.*
$ mdir a:/
```

La orden `mcopy` (MS-DOS copy) sirve para copiar ficheros entre el árbol de directorios de UNIX y un disquete (disco en general) con formato MS-DOS/Windows sin necesidad de montar ni desmontar la unidad de disquetes (la unidad no debe estar montada para que funcione). La copia se puede realizar desde o hacia un disco MS-DOS/Windows.

```
$ mcopy a:/so1/*.* .
$ mcopy –t a:/prueba.txt practica1/.
$ mcopy –t a:/prueba.txt practica1/prueba1seguridad.txt
```

Cuando hay que copiar archivos en modo texto es necesario especificar la opción `–t`. Si no se indica se copian en binario.

Existen otras órdenes de la librería MTOOLS (`mdel, mmove, mtype, mcat, mcd, mmd, mrd, mdeltree` …)

REFERENCIAS

BIBLIOGRAFÍA

- Syed Mansoor Sarwar: El Libro de Unix, Addison Wesley, 2002.
- F. Maciá Pérez y A. Soriano Payá: El Shell Korn. Manual de Usuario y Programador, Servicio de Publicaciones de la Universidad de Alicante, 1999.
- J. Tackett y D. Gunter: Linux 4ª Edición, Prentice may, 2000.

WEB

Existe una gran cantidad de manuales y tutoriales de Unix y Linux en Internet que puedes localizar a partir de cualquier buscador (ej. www.google.com).

AUTOEVALUACIÓN

1. A partir del archivo `gente`, mostrar aquellos usuarios que contengan la secuencia de las letras ll ordenados de forma alfabética y a la vez crear un archivo llamado `ll.txt`.

2. A partir del archivo `gente`, mostrar aquellos usuarios que no tiene el dígito 0 en su código.

3. Crear con el comando `zip` en el directorio inicial el archivo `todoalu.zip` que contenga los archivos y subdirectorios del directorio inicial`.`

4. Crear el directorio `tmp` en el directorio inicial.

5. Descomprimir con el comando `unzip` el archivo `todoalu.zip` en el directorio `tmp`.

6. Crear el archivo `contenido` con el contenido del directorio inicial. A continuación, añadir al principio del archivo la fecha actual. Hacerlo todo en una sola línea de órdenes.

7. Mostrar **únicamente** los archivos que contengan como nombre la cadena `alu` ubicados en `/usr`.

8. Mostrar el nombre de los usuarios (login) dados de alta en el sistema ordenados alfabéticamente de forma descendente. Hacerlo todo en una sola línea de órdenes.

9. Ejecutar las siguientes líneas de órdenes:
   a. sleep 1000 &
   b. sleep 2000 &
   c. sleep 3000 &

10. Mostrar únicamente los procesos que está ejecutando el usuario `alu` y redirigirlos al archivo `proceso1`

11. Eliminar el proceso creado al ejecutar sleep 2000

12. Mostrar únicamente los procesos que está ejecutando el usuario `alu` y redirigirlos al archivo `proceso2`

13. Contar los usuarios que contienen el archivo `gente` cuyo apellido o nombre contienen una `S`.

14. Contar los usuarios que contienen el archivo `gente` cuyo apellido contiene una `S`.

15. Contar los usuarios que contienen el archivo `gente` cuyo nombre contiene una `S`.

16. Listar los archivos del directorio inicial ordenados por tamaño.