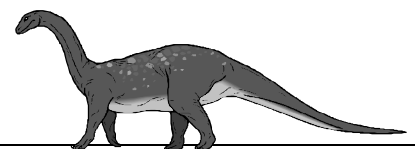


Sistemas Operativos I ***Manual de prácticas***

Grupo de Sistemas Operativos
(DSIC/DISCA)

Práctica 2: Introducción a UNIX (II)



PRÁCTICA 2: INTRODUCCIÓN A UNIX II

INTRODUCCIÓN

Después de estudiar algunas órdenes básicas junto con el manejo de ficheros y directorios en la práctica 1, continuamos con el aprendizaje de 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 el modo de trabajo interactivo, el usuario introduce los datos desde teclado, y observa la salida en pantalla.
- En el modo de operación por lotes (no interactivo), tanto las entradas como las salidas se realizan sobre ficheros.

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

En UNIX, existen tres ficheros 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 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 fichero, 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 diagnóstico coincide por omisión, con la salida estándar.

Observe el resultado de la ejecución de las siguientes órdenes:

```
$ cat /etc/passwd
...
$ cat
Hola, procura no \\
aporrear el teclado.\\
<Ctrl>-d
...
```

La segunda ejecución de `cat` repetirá los mensajes escritos en la entrada estándar, al no haberse especificado ningún fichero.

REDIRECCIÓN DE LA SALIDA ESTÁNDAR

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

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

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

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

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

```
$ > vacio
```

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

```
$ cat > texto
cat es una orden útil para
generar ficheros nuevos.
<Ctrl>d
$ cat texto
...
```

Si queremos que la salida de un programa se añada al contenido actual de un fichero, 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 fichero, 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 -3 < salida.dat
```

La orden tail filtra su entrada, dejando pasar únicamente las últimas líneas (en este ejemplo las tres últimas líneas). Como la entrada estándar está redirigida al fichero 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 fichero.
- 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.

Veamos:

```
$ 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
...
```

NOTA.- el símbolo > que aparece al principio de cada nueva línea mientras editamos el texto indica que el sistema sigue esperando datos

SALIDA DE DIAGNÓSTICO

De la misma manera que podemos redirigir la salida estándar, es posible almacenar la salida de error en un fichero¹, usando 2>.

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

¹ noexisto.dat se considera un archivo inexistente.

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 fichero 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 (en este caso `wc -l` que cuenta líneas), que se le pasa como parámetro, y despliega en la salida de error una contabilidad de tiempos sobre su ejecución².

```
$ time wc -l /etc/passwd
...
$ /bin/time3 wc -l /etc/passwd > num_users
...
$ cat num_users
...
$ /bin/time wc -l noexisto.dat > num_users 2>&1
$ cat num_users
...
```

TUBERÍAS

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

```
$ who | sort
...
```

Ahora, la salida estándar de `ls` se usa como entrada estándar de `sort` que es un ejemplo de filtro que permite ordenar la información (en un apartado posterior se describe éste y otros ejemplos más de filtros). 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
...
```

El ejemplo anterior conecta la salida de `ls-l` (contenido detallado del directorio actual) hacia `sort`, que ordena las líneas resultantes y las envía a `more`, encargado de mostrar el resultado pantalla a pantalla (tras cada pantalla espera a que el usuario pulse una tecla).

² Es necesario que el usuario se asegure de invocar a `time` con su ruta de acceso completa o bien que el intérprete de órdenes tiene su camino de búsqueda asignado (probar con “which time”).

³ Compruebe previamente si la orden `time` se encuentra en ese directorio o en `/usr/bin`. Para ello observe el resultado de “which time”.

En ocasiones interesa bifurcar el flujo de datos para que una determinada salida sirva como entrada a un fichero y simultáneamente se dirija hacia su salida. La posibilidad que UNIX ofrece es el uso de la orden `tee` (inserción de una T):

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

En el ejemplo anterior, la salida del `ls -l` la ha dejado `tee` en el fichero `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.

FILTROS BÁSICOS

A continuación se describen algunas órdenes que resultan útiles para el trabajo con ficheros y que se combinan con el uso de los mecanismos de redirección y tuberías. Estos filtros utilizan algunos ficheros de ejemplo para actuar sobre ellos. Inicialmente tales ficheros no se encuentran en su directorio de trabajo. Para copiarlos, utilice la siguiente orden:

```
$ cd
$ cp -R /practicas/asignaturas/sol/pr2/* .
```

Veamos cuáles son los filtros más importantes:

SORT

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

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

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, con la opción `-k desde,hasta`:

- A partir del segundo campo -Sólo el segundo campo

```
$ sort -k2 gente           $ sort -k2,2 gente
Jack Austen      120      Jack Austen      120
Jane Bailey     121      Jane Bailey     121
```

Maryann Clark	101	Maryann Clark	101
Steve Daniels	111	Steve Daniels	111
Sylvia Dawson	110	Sylvia Dawson	110
Henry Morgan	112	Henry Morgan	112
Hank Parker	114	Hank Parker	114
Sally Smith	113	Charlie Smith	122
Charlie Smith	122	Sally Smith	113
Bill Williams	100	Bill Williams	100

En el primer ejemplo se ordena por apellido y luego por número, y en el segundo sólo por apellido. La diferencia puede observarse en la penúltima y antepenúltima líneas

Ordenación numérica. (opción n)

- Interpretación ASCII

- Interpretación numérica

\$ sort -k3 puntos		\$ sort -k3n puntos	
Steve Daniels	11	Bill Williams	2
Sally Smith	14	Jane Bailey	2
Hank Parker	18	Jack Austen	3
Maryann Clark	18	Henry Morgan	5
Bill Williams	2	Sylvia Dawson	7
Jane Bailey	2	Charlie Smith	9
Jack Austen	3	Steve Daniels	11
Henry Morgan	5	Sally Smith	14
Sylvia Dawson	7	Hank Parker	18
Charlie Smith	9	Maryann Clark	18

Orden inverso. (opción r)

\$ sort -k3n puntos		\$ sort -k3nr puntos	
Bill Williams	2	Hank Parker	18
Jane Bailey	2	Maryann Clark	18
Jack Austen	3	Sally Smith	14
Henry Morgan	5	Steve Daniels	11
Sylvia Dawson	7	Charlie Smith	9
Charlie Smith	9	Sylvia Dawson	7
Steve Daniels	11	Henry Morgan	5
Sally Smith	14	Jack Austen	3
Hank Parker	18	Bill Williams	2
Maryann Clark	18	Jane Bailey	2

SPLIT

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

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

División por defecto: el prefijo por defecto es x, y el número de líneas 1000 (crea ficheros xaa, xab, xac, xad... con 1000 líneas cada uno)

\$ 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 fichero. La sintaxis en este caso es como sigue:
cut [opciones] nombre ...

Cortando columnas. Opción -c

```
$ cut -c16,17,18 gente
122
112
...
$ cut -c7-18 gente
$ cut -c14- gente
```

Cortando campos (por defecto, cut asume que los campos se encuentran separados por tabuladores). Opción -f

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

El ejemplo anterior es poco ilustrativo, porque los campos del fichero gente no se encuentran separados por tabuladores, sino por blancos (con lo que cut interpreta que existe un único campo por línea). Es posible indicar de forma explícita el carácter usado como separador entre campos (opción -d)

```
$ cut -d" " -f1,2 gente
Charlie Smith
```

```
Henry Morgan
...
```

Ejemplo

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

PASTE

El mandato une las líneas de varios ficheros. Si los ficheros de entrada son a,b,c, la n-esima línea del nuevo fichero se forma como
 an tabulador bn tabulador cn
siendo xn la n-esima línea del fichero x.

Sintaxis

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

Uniendo ficheros

```
$ 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 (el primer carácter se usa para separar los dos primeros campos, el segundo para los dos siguientes, etc.)

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

WC

El mandato `wc` cuenta del número de líneas (`-l`), palabras (`-w`) y caracteres (`-c`) de un fichero 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 tabulador⁴):

```
$ 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-DOS⁵:

```
$ 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 metacaracteres⁶, tanto en las expresiones como en la lista de nombres de ficheros. La forma general del mandato es la siguiente:

⁴ Véase en el manual de `tr` como se identifican los caracteres especiales.

⁵ En MS-DOS cada línea finaliza con un par de caracteres (salto de línea y retorno de carro), pero en UNIX se utiliza únicamente el carácter de salto de línea. Para realizar la conversión es suficiente borrar los caracteres de retorno de carro, que se representan como `\r`

```
grep [opciones] expresión [fichero] ...
```

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 ficheros pero no las líneas.
- -v Muestran las líneas donde **no** se produce la coincidencia.

Por ejemplo:

```
$ 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*⁷, el programa `sh`⁸ que se encarga de lanzar a ejecución dichos procesos.

El `sh` es una orden del sistema UNIX que consiste en un intérprete de un lenguaje de programación que lee su entrada desde una terminal o desde un fichero. Cuando un usuario comienza una sesión de trabajo, el UNIX arranca un programa `sh` que funciona de forma interactiva con su terminal.

Para arrancar un proceso, basta que el usuario teclee el nombre del fichero asociado en su terminal.

```
$ date
```

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

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

⁶ El concepto es similar al utilizado en el sistema de ficheros, aunque mucho más potente. Ej.- en el sistema de ficheros utilizamos ‘?’ para indicar un carácter cualquiera, y ‘*’ para indicar una secuencia de caracteres cualesquiera: esto permite utilizar `ls prueba *` para listar todos los ficheros que empiezan con la palabra prueba, `ls *.txt` para listar todos los que acaban en .txt, `ls ej?.obj*` para todos los formados como ej seguido de cualquier carácter, .obj, y cualquier tira, etc.

⁷ En un entorno gráfico, al abrir una ventana terminal

⁸ Generalmente se habla de `sh`, pero los sistemas UNIX ofrecen distintos intérpretes de órdenes con características similares, así podemos encontrar: `sh`, `csh`, `ksh` y `bash`.

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 &
.....
$ date&ps&who
.....
```

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

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.

Compruebe la ejecución de las siguientes órdenes:

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

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

ORDEN PS

Esta orden muestra en la pantalla información sobre los procesos que se están ejecutando en el instante actual.

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

Algunas opciones, proporcionan información más detallada (ver las páginas de man)

```
$ ps -ef
...
```

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

Esta orden se usa para matar a un proceso en ejecución, para usarla, debemos conocer el PID el proceso que se desea terminar. 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
...
```

Resulta curioso el resultado de la ejecución de la siguiente secuencia:

```
$ ps
  PID TTY      TIME    COMMAND
 29437 pts/1    00:00:00 bash
  6816 pts/1    00:00:00 ps
$ kill -9 29437
.....
```

Si el shell que acabamos de eliminar tenía todavía algún proceso hijo, estos procesos descendientes reciben la señal `SIGHUP`⁹. Normalmente esto hace que los procesos descendientes sean también eliminados, aunque depende del tratamiento que cada uno de ellos haga de esta señal (si no se realiza ningún tratamiento especial, morirán).

ORDEN NICE

En ocasiones, es interesante modificar a voluntad la prioridad con la que opera un proceso. El valor de la prioridad, hace referencia a la cantidad de tiempo que se le asigna a un proceso. En UNIX existe un valor para cada proceso (NICE) que se usa para calcular la prioridad efectiva de este (PRI). Podemos ver estos valores tecleando:

```
$ ps -l
...
```

Un valor pequeño para PRI hace referencia a una elevada prioridad. Compruebe las prioridades mediante la siguiente secuencia:

```
$ wc /usr/lib/* >/dev/null 2>/dev/null&
$ ps -l
...
$ nice wc /usr/lib/* >/dev/null 2>/dev/null&
$ ps -l
...
```

ALMACENAMIENTO Y COMPRESIÓN DE FICHEROS

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 ficheros en uno solo. Su sintaxis consiste en:

```
tar [opciones] [nombres de ficheros ...]
```

Por ejemplo, el siguiente mandato

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

permite almacenar todos los ficheros del directorio `prueba1` (respetando su estructura) en el fichero `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 ficheros 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

⁹ Las versiones 2.2.xx del núcleo de Linux ya no envían la señal `SIGHUP` a los procesos descendientes. Por tanto, bajo Linux no se da esta eliminación de los procesos descendientes. Sin embargo, en otros sistemas UNIX sí puede darse.

(en el ejemplo están limitados al directorio `prueba1`) consistirán en los ficheros o directorios que se pretende archivar.

Si se ejecuta el mandato

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

se extraerá la información procedente del fichero `backup.tar` y se depositará en el directorio actual. Si los ficheros 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 diferencia de algunas utilidades de archivo disponibles en otros sistemas operativos, `tar` no comprime de forma automática la información de los ficheros conforme los archiva. Para ello se utilizan mandatos como `gzip`, que permiten comprimir cualquier clase de ficheros (no sólo archivos `tar`). Por ejemplo, el mandato

```
$ gzip backup.tar
```

comprimirá el fichero `backup.tar`, y lo sustituirá por el nombre de fichero `backup.tar.gz`. Es decir, la versión comprimida del fichero 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 fichero comprimido (`.Z`). El mandato para invertir el resultado de `compress`, es `uncompress`. En el caso que el tamaño del fichero comprimido resulte aún demasiado grande para su manejo se puede recurrir a utilidades como `split` descrita en el apartado de FILTROS.

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
```

AUTOEVALUACIÓN

1.- Utilice las utilidades de redirección para incluir en la cabecera de un fichero ya creado la fecha actual. Incluya también al final del fichero el número de líneas que lo componen. No es necesario que el fichero resultante sea el mismo que se tomó inicialmente para efectuar estas modificaciones.

2.- A partir de un fichero de nombres (p.e. gente) detecte aquellos que contengan la secuencia de letras ll y almacénelos en un nuevo fichero ordenados de forma alfabética.

3.- Usando algún sistema de ventanas “X11” cree tres ventanas “xterm” y, desde una de ellas en concreto intente identificar a las demás mediante ps y termínelas de forma selectiva usando la orden kill. ¿Cómo puede saber qué proceso corresponde a cada ventana?

4.- Utilice la orden tar para crear un fichero que contenga a todos sus ficheros. Comprima este fichero mediante la orden gzip y copie el resultado a un nuevo directorio o almacénelo en su disquete. Escriba las órdenes necesarias para obtener de nuevo su sistema de ficheros en el caso en que todos los ficheros, excepto el directorio público, hubieran desaparecido.