Universidad **T**ecnológica **N**acional
F acultad **R**egional **R**esistencia
Departamento **d**e **I**ngeniería **e**n **S**istemas **d**e **I**nformación

French 414 -TE 362-4432928
-TEL-FAX 362-4432683 (Int. 219)
(3500) Resistencia Chaco

Guía de Laboratorio

SISTEMA DE ARCHIVOS

Título: LABORATORIO SISTEMAS OPERATIVOS

Asignatura: Sistemas Operativos

Área: Computación

Equipo docente:

Director De Cátedra

Nombre y Apellido: Ing. Liliana CUENCA PLETSCH

Categoría docente: Prof. Titular concursada

e-mail: cplr@frre.utn.edu.ar – cplr@arnet.com.ar

Docentes por División

División "A"

Profesor

1º Cuatrimestre

Nombre y Apellido: Ing. Liliana CUENCA PLETSCH

2º Cuatrimestre

Nombre y Apellido: Dr. Sergio GRAMAJO

Categoría docente: Prof. Adjunto Concursado e-mail:

Auxiliar/es

Nombre y Apellido: Ing. RISTOFF Alberto

Categoría docente: J.T.P. Concursado

e-mail: ristoffalberto@hotmail.com

División "B"

Profesor

1º Cuatrimestre

Nombre y Apellido: Ing. Liliana CUENCA PLETSCH

2º Cuatrimestre

Nombre y Apellido: Dr. Sergio GRAMAJO

Auxiliar/es

Nombre y Apellido: Ing. ROA Jorge Alejandro

Categoría docente: Auxiliar de Trabajos Prácticos de 1º Concursado

e-mail: jorge@internea.com.ar

French 414 -TE 362-4432928
-TEL-FAX 362-4432683 (Int. 219)
(3500) Resistencia Chaco

BIBLIOGRAFÍA

a) **OBLIGATORIA:**

TANENBAUM Andrew S.: *"*Sistemas Operativos Modernos"*. ".* Segunda Edición, México, Prentice Hall Hispanoamericana.

SILBERSCHATZ A. y. **otros***: "*Sistemas Operativos – Conceptos Fundamentales*".* Tercera Edición o superior, España, Reverté S.A..

TANENBAUM Andrew S.: *"*Sistemas Operativos – Diseño e Implementación*".* Segunda Edición o superior, México, Prentice Hall Hispanoamericana.

STALLINGS William : *"*Sistemas Operativos*".* Cuarta Edición o superior, México, Prentice Hall Hispanoamericana.

b) **EXTRA:**

CARRETERO PEREZ J, CARBALLEIRA F., ANASAGASTI P., PEREZ COSTOYA F.: "Sistemas Operativos - Una Visión Aplicada" . Mc Graw Hill. 2001

French 414 -TE 362-4432928
-TEL-FAX 362-4432683 (Int. 219)
(3500) Resistencia Chaco

Trabajo con sistema de archivos:

Objetivo:

Lograr un primer contacto con el sistema de archivos de un sistema operativo , comprender como se realiza la administración de archivos en el S.O. Linux.
Reconocer el árbol de directorios, su estructura típica, qué es el path absoluto y el path relativo. Lograr un manejo de los comandos básicos relacionados con directorios y archivos.

Conocimientos previos:

• Teoría de Sistemas de Archivos.

3. Archivos y directorios

3. 1. Definiciones básicas

Archivo

Toda la información, ya sean textos, imágenes, o información para la configuración del sistema, se almacena en "archivos", que a su vez se guardan en "directorios". Los archivos son la estructura empleada por el sistema operativo para almacenar información en un dispositivo físico como un disco duro, un pendrive, un CD-ROM. Con todas las herramientas y programas existentes se puede acceder a estos ficheros para ver su contenido o modificarlo.

La base del sistema de archivos de Linux, es obviamente el archivo o fichero.

Todos los archivos de Linux tienen un nombre, el cual debe cumplir unas ciertas reglas:

- Un nombre de archivo puede tener entre 1 y 255 caracteres.
- Se puede utilizar cualquier carácter excepto la barra inclinada (*/*). No es recomendable emplear los siguientes caracteres: = \ ^ ~ ' " ` * ; - ? [ ] ( ) ! & ~ < >, ya que éstos tienen un significado especial en Linux.

  Nota: *Para emplear ficheros con estos caracteres o espacios hay que introducir el nombre del fichero entre comillas.*

- Se pueden utilizar números exclusivamente si así se desea.
- Las letras mayúsculas y minúsculas se consideran diferentes, y por lo tanto no es lo mismo carta.txt que Carta.txt ó carta.Txt. Se puede emplear un cierto criterio de "*tipo*" (extensión) para marcar las distintas clases de archivos (a modo referencial), empleando una serie de caracteres al final del nombre que indiquen el tipo del que se trata. Así, los archivos de texto, HTML, las imágenes PNG o JPEG tienen extensiones .txt, .htm (o .html), .png y .jpg (o .jpeg) respectivamente.

Pese a esto Linux sólo distingue tres tipos de archivos:

- **Archivos o ficheros ordinarios**: Son los mencionados anteriormente.
- **Directorios (o carpetas):** Es un archivo especial que agrupa otros ficheros de una forma estructurada.
- **Archivos especiales:** Son la base sobre la que se asienta Linux, puesto que representan los dispositivos conectados a un ordenador, como puede ser una impresora. De esta forma introducir información en ese archivo equivale a enviar información a la impresora. Para el usuario estos dispositivos tienen el mismo aspecto y uso que los archivos ordinarios.

Notal: Todo en Linux son archivos (si, hasta los directorios).

Directorio

Un directorio es un conjunto de archivos, que a su vez pueden contener otros directorios. Los directorios también poseen permisos, y eso es muy importante para evitar que los usuarios sin experiencia, borren o modifiquen algo que no deban. El árbol de directorios nos ayuda a saber dónde se encuentra un archivo.

A diferencia de los sistemas **MS-DOS/Windows**, en los sistemas UNIX no se dividen los directorios entre las distintas unidades físicas (C:, D:, etc.). Al contrario, para UNIX todos son ficheros y directorios. Éste considerará un disco duro o una disquetera como directorios. Por tanto, no tiene sentido escribir en la consola "C:", sino que nos iremos al directorio asociado a esa partición.

Todos los ficheros y directorios de un sistema UNIX cuelgan de un directorio principal llamado "***raíz***", que se representa como "**/**". El directorio raíz es la base para todo el arbol de directorios, es allí donde están contenidos todos lo directorios del sistema.

Cuando el usuario accede a una sesión, Linux "envía" al usuario a su directorio de trabajo, que es su directorio personal (*/home/nombre-usuario*). En donde el usuario tiene la libertad absoluta para hacer lo que quiera con sus ficheros y directorios ubicados ahí. Sin embargo no podrá hacer todo lo que quiera en el directorio de otro usuario, ya que Linux tiene un sistema de permisos que concede o restringe libertades sobre los directorios y ficheros que hay en Linux. Aunque existe un usuario "**root**", que sí tiene permiso de hacer cualquier cosa en Linux.

En qué directorio me encuentro…?

Basta con escribir el comando **pwd** (*Print Work Directory*).

Ejemplo:

[felix@localhost felix]$ pwd */home/felix/*

Para cambiar de directorio...

Emplearemos el comando **cd** (*Change Directory*)

Ejemplo:

[felix@localhost Comandos]$ cd /etc Nos vamos al directorio **/etc**

Nota: Si simplemente escribimos **cd**, sin especificar el nombre del directorio, esto será igual que escribir **cd /home/nombre-usuario**, es decir, me envía a mi propio directorio de trabajo.

Directorio HOME:

Directorio asignado a un usuario al crearse la cuenta del mismo. Sobre este directorio el usuario puede crear, modificar y borrar archivos y directorios. No confundir con el directorio /home.

Por ejemplo, el directorio HOME del usuario a123456 será /home/a123456

El camino o path

En cualquier sistema operativo moderno la estructura de archivos es jerárquica y depende de los directorios. En general la estructura del sistema de archivos se asemeja a una estructura de árbol, estando compuesto cada nodo por un directorio o carpeta, que contiene otros directorios o archivos.

El camino o path de un fichero o directorio es la secuencia de directorios que se ha de recorrer para acceder a un determinado fichero separados por /. Supongamos la estructura de archivos de la siguiente figura:

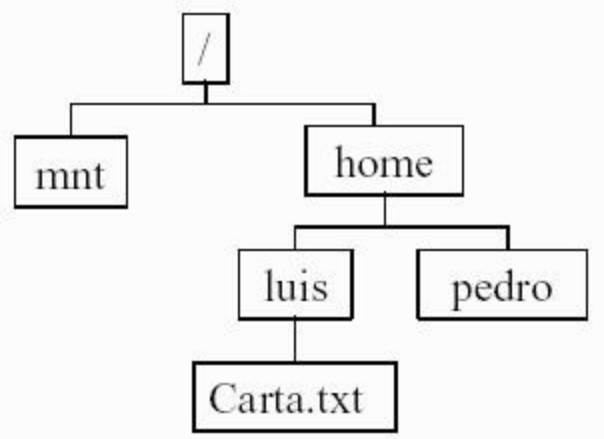


Figura 2

Existen dos formas del path o camino:

- el camino absoluto que muestra toda la ruta a un fichero, */home/luis/Carta.txt*.
- el path relativo a un determinado directorio, por ejemplo si no encontramos en el directorio */home*, el path relativo al fichero *Carta.txt es luis/Carta.txt*

*Ejemplo: si el usuario **a123456** se encuentra en su directorio HOME **(/home/a123456**) y en ese directorio existe un archivo llamado **fich**, podemos referirnos a dicho archivo de varias formas:*

Path absoluto: ***/home/alumnos/a123456/fich***

Path relativo:

fich

./fich
../a123456/fich
../../alumnos/a123456/fich

Para complicar aun más las cosas, todos los directorios contienen dos directorios especiales:

- El directorio actual, representado por *el punto* (**.**)
- El directorio padre, representado por *dos puntos* (**..**)

Estando en el directorio **/home/pedro** se puede acceder a **Carta.txt** con **/home/luis/Carta.txt** (*path absoluto*) o bien **../luis/Carta.txt** (path relativo). En **luis** como **./ Carta.txt** o simplemente Carta.txt

Estructura básica del árbol de directorios

/ directorio raíz.
/bin comandos para usuarios comunes.
/dev conjunto de archivos especiales destinados a la gestión de periféricos.
/etc archivos empleados en la administración del sistema.
/home directorio que almacena los directorios personales de los usuarios comunes del sistema.
/mnt punto de montaje de otros sistemas de archivos.
/proc contiene archivos destinados a la comunicaciones entre el HW y el kernel de Linux.
/sbin programas relacionados con la administración del sistema.
/tmp directorio temporal (se borra cada vez que se arranca el sistema).
/usr contiene diversos subdirectorios con herramientas para los usuarios (**/usr/bin**, **/usr/sbin)**, documentación de paquetes instalados (/usr/doc), archivos cabecera para los programas en C (**/usr/include**), las librerías del compilador de C (**/usr/lib**).

/var complementa a **/usr**. Su contenido es variable, de ahi su nombre

Otros comandos básicos relacionados con archivos y directorios

mkdir *directorio*: Crea el directorio indicado.

rmdir *directorio*: Elimina el directorio indicado (si está vacío y no es el directorio de trabajo).

cat *archivo* Permite visualizar por pantalla el contenido del archivo indicado. Podemos redireccionar la salida estándar (la pantalla) mediante el símbolo >.

$ **cat archivo >nuevo** De esta manera el archivo no se muestra por pantalla sino que se crea un archivo llamado nuevo que es copia de archivo.

Pueden crearse pequeños archivos de texto mediante **$ cat >nuevo**

Una vez pulsada la tecla Enter podemos empezar a añadir líneas de texto al nuevo archivo. Para finalizar pulsamos la combinación de teclas [Control+D].

ls Muestra los archivos del directorio actual en orden alfabético.

Atención: El directorio actual aparece como **.** y el directorio padre como **..**

Puede tener como argumento un nombre de archivo o directorio. Pueden usarse patrones de archivos y distintas opciones que veremos en profundidad en la siguiente práctica.

cp [-r] *fuente destino* Permite copiar uno o varios archivos al destino indicado. La opción -r permite hacer copias de sub-árboles de directorios completos.

Si, destino es un archivo, fuente debe ser un sólo archivo. Si destino es un directorio, fuente puede ser varios archivos (una lista de archivos separados por blancos o utilizando máscaras).

rm *archivo* Borra el archivo indicado. La opción -r permite un borrado recursivo de directorios incluso si estos no están vacíos (¡cuidado!).

mv *fuente destino* Permite mover un archivo a otro directorio y/o renombrarlo.

Enlaces e inodos

En Linux, cada archivo en el sistema está representado por un inodo. Un inodo no es más que un bloque que almacena información de los archivos, de esta manera a cada inodo podemos asociarle un nombre. A simple vista pareciera que a un mismo archivo no podemos asociarle varios nombres, pero gracias a los enlaces esto es posible.

Enlaces físicos o duros

Un enlace físico no es más que una etiqueta o un nuevo nombre asociado a un archivo. Es una forma de identificar el mismo contenido con diferentes nombres. Éste enlace **no es una copia separada del archivo anterior** sino un nombre diferente para el mismo contenido.

Para crear un enlace físico en Linux, ejecutamos un comando como:

```
$ ln archivo.txt nuevo_nombre.txt
```

El enlace aparecerá como otro archivo más en el directorio y apuntará al mismo contenido de *archivo.txt*. Cualquier cambio que se haga se reflejará de la misma manera tanto para *archivo.txt* como para *nuevo_nombre.txt*.

Un enlace se puede borrar usando el comando *rm* de la misma manera en que se borra un archivo, sin embargo **el contenido del inodo no se eliminará mientras haya un enlace físico que le haga referencia**. Esto puede tener varias ventajas, pero también puede complicar la tarea de seguimiento de los archivos. Un enlace físico tampoco puede usarse para hacer referencia a directorios o a archivos en otros equipos.

Enlaces simbólicos

Un enlace simbólico también puede definirse como una etiqueta o un nuevo nombre asociado a un archivo pero a diferencia de los enlaces físicos, **el enlace simbólico no contiene los datos del archivo**, simplemente apunta al nombre de un archivos. Tiene mucha similitud a un *acceso directo* en Windows o un *alias* en OS X.

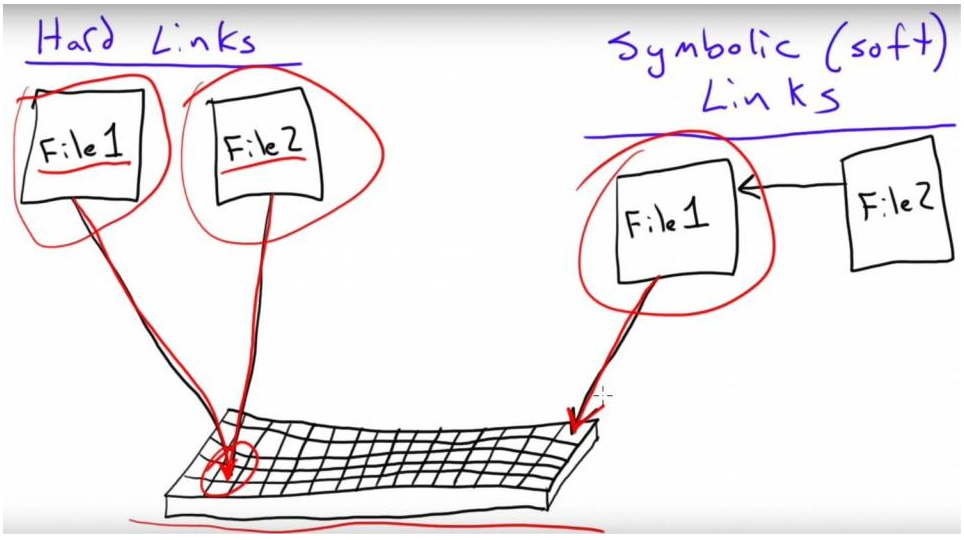
Para crear un enlace simbólico del archivo *archivo.txt* a *nuevo_nombre.txt*, ejecutamos:

```
$ ln -s archivo.txt nuevo_nombre.txt
```

Éste enlace también aparecerá como otro archivo más en el directorio y apuntará al mismo contenido de *archivo.txt*, reflejando todos los cambios que se hagan tanto para *archivo.txt* como para *nuevo_nombre.txt*.

Sobre un enlace simbólico también se pueden usar todos los comandos básicos de archivos (rm, mv, cp, etc). sin embargo cuando el archivo original es borrado o movido a una ubicación diferente el enlace dejará de funcionar y se dice que el enlace **está roto**.

Un enlace simbólico permite enlazar directorios y, usando NFS, también permite enlazar archivos fuera del equipo.



French 414 -TE 362-4432928
-TEL-FAX 362-4432683 (Int. 219)
(3500) Resistencia Chaco

Redireccionamiento de Entrada y Salida Estándar

La **entrada estándar** representa los datos que necesita una aplicación para funcionar, como por ejemplo un archivo de datos o información ingresada desde la terminal y es representado en la terminal como el **tipo 0**.

La **salida estándar** es la vía que utilizan las aplicaciones para mostrarte información, allí podemos ver el progreso o simplemente los mensajes que la aplicación quiera darte en determinado momento y es representado en la terminal como el **tipo 1**.

El **error estándar** es la forma en que los programas te informan sobre los problemas que pueden encontrarse al momento de la ejecución y es representado en la terminal como el **tipo 2**.

Entonces una redirección consiste en **trasladar la información**, por ejemplo, de la salida estándar a la entrada estándar o del error estándar a la salida estándar. Esto lo logramos usando el símbolo **>**.

Por ejemplo, para redireccionar la salida de un comando y "volcarla" en un archivo bastaría con ejecutar:

```
ls -la > archivo.txt
```

Sin embargo, cada vez que ejecutemos ese comando el contenido de *archivo.txt* será reemplazado por la salida del comando *ls*. Si queremos agregar la salida del comando al archivo, en lugar de reemplazarla, entonces ejecutamos:

```
ls -la >> archivo.txt
```

Se pueden dar otros usos al redireccionamiento de la salida estándar como por ejemplo:

$ cat archivo > nuevo De esta manera el archivo no se muestra por pantalla sino que se crea un archivo llamado nuevo que es copia de archivo.

Entonces pueden crearse pequeños archivos de texto mediante **$ cat > nuevo**

Una vez pulsada la tecla Enter podemos empezar a añadir líneas de texto al nuevo archivo. Para finalizar pulsamos la combinación de teclas [Control+D].

Lo interesante es que, además de la salida estándar, también podemos redireccionar el error estándar y la entrada estándar.

Permisos de archivos y directorios

En Linux, todo archivo y directorio tiene tres niveles de permisos de acceso: los que se aplican al propietario del archivo, los que se aplican al grupo que tiene el archivo y los que se aplican a todos los "otros" usuarios del sistema. Veamos los permisos listando un directorio con

```
ls -l
```

Veremos algo como

```
-rwxrwxr--  1 sergio ventas    9090 sep  9 14:10 presentacion
-rw-rw-r--  1 sergio sergio 2825990 sep  7 16:36 reporte1
drwxr-xr-x  2 sergio sergio    4096 ago 27 11:41 videos
```

La primera columna, por ejemplo, de la primera línea (-rwxrwxr--) es el tipo de archivo y sus permisos, la siguiente columna (1) es el número de enlaces al archivo, la tercera columna (sergio) representa al propietario del archivo, la cuarta columna (ventas) representa al grupo al que pertenece al archivo y las siguientes son el tamaño, la fecha y hora de última modificación y por último el nombre del archivo o directorio.

El primer caracter al extremo izquierdo, representa el tipo de archivo, los posibles valores para esta posición son los siguientes:

- - un guión representa un archivo común (de texto, html, mp3, jpg, etc.)
- d representa un directorio
- l link, es decir un enlace o acceso directo
- b binario, un archivo generalmente ejecutable

Los siguientes 9 restantes, representan los permisos del archivo y deben verse en grupos de 3.

Los tres primeros representan los permisos para el propietario del archivo. Los tres siguientes son los permisos para el grupo del archivo y los tres últimos son los permisos para el resto del mundo u “otros”.

En cuanto a las letras, su significado son los siguientes:

- r read - lectura
- w write - escritura (en archivos: permiso de modificar, en directorios: permiso de crear archivos en el dir.)
- x execution - ejecución

Las nueve posiciones de permisos son en realidad un bit que está encendido (mostrado con su letra correspondiente) o esta apagado (mostrado con un guión -), así que, por ejemplo, permisos como rwxrw-r--, indicaría que los permisos del propietario (rwx) puede leer, escribir y ejecutar el archivo, el grupo (o sea los usuarios que estén en mismo grupo del archivo) (rw-) podrá leer y escribir, pero no ejecutar el archivo y cualquier otro usuario del sistema (r--), solo podrá leer el archivo, ya que los otros dos bits de lectura y ejecución no se encuentran encendidos o activados.

Llamadas al sistema para manejo de archivos

Los sistemas tipo UNIX proporcionan un conjunto de llamadas al sistema para la manipulación de ficheros. Todas las aplicaciones o utilidades que en UNIX trabajan con archivos están fundamentadas en estos servicios básicos. La biblioteca estándar de C dispone de un conjunto de funciones para utilizar directamente estas llamadas al sistema, proporcionando al programador la misma visión que sobre los recursos tiene el sistema operativo UNIX.

Estas funciones se suelen denominar "de bajo nivel". La biblioteca estándar también ofrece otras rutinas más cómodas, construidas a partir de las llamadas al sistema. (La interfaz con estos servicios se encuentra en <stdio.h>).

Este apartado explica algunas de las llamadas al sistema del UNIX que trabajan con ficheros, que les permitirán:

* Abrir y cerrar un fichero

* Crear y borrar un fichero

* Leer en un fichero

* Escribir en un fichero

* Desplazarse por un fichero

Manejo de ficheros en UNIX

El manejo de ficheros en UNIX sigue el modelo de la sesión. Para trabajar con un fichero hay primero que abrirlo con una invocación a la función open. Ésta devuelve un descriptor de fichero (*file descriptor* en inglés), un número entero que servirá de identificador de fichero en futuras operaciones. Finalmente hay que cerrar el fichero, con la función close, para liberar los recursos que tengamos asignados.

Existen al menos tres descriptores ya establecidos en la ejecución de un programa (ya los han abierto por nosotros). El descriptor 0 es la entrada estándar (normalmente el teclado), el descriptor 1 es la salida estándar (normalmente la pantalla) y el descriptor 2 el fichero estándar de visualización de errores (también la pantalla, normalmente). Los pueden considerar como simples ficheros que ya han sido abiertos, y pueden trabajar con ellos con cierta normalidad. Incluso los pueden cerrar.

Los ficheros en UNIX permiten tanto el acceso directo como el secuencial. Cada fichero abierto dispone de un puntero que se mueve con cada lectura o escritura. Hay una función especial llamada lseek para posicionar ese puntero donde se quiera dentro del fichero.

Especificación de los permisos: forma octal

Las llamadas al sistema creat y open admiten un parámetro entero en el que se especifican los permisos con los que se crea un archivo. Una de las maneras más cómodas de declararlos es mediante la representación octal.

Los permisos se forman como un número de 9 bits, en el que cada bit representa un permiso, tal y como se muestra en el cuadro (es el mismo orden con el que aparecen cuando hacemos un ls).

RWX	RWX	RWX
usuario	grupo	otros

Se toman los nueve permisos como tres números consecutivos de 3 bits cada uno. Un bit vale 1 si su permiso correspondiente está activado y 0 en caso contrario. Cada número se expresa en decimal, del 0 al 7, y los permisos quedan definidos como un número octal de tres dígitos. Para poner un número en octal en el lenguaje C, se escribe con un cero a la izquierda.

Por ejemplo, los permisos **rw-r--r-x** son el número octal 0645.

```
/* Crea un fichero con permisos RW-R--R-- */
int fd = creat ( "mi_fichero", 0644);
```

Apertura, creación y cierre de ficheros

Función open

La función **open** abre un fichero ya existente, retornando un descriptor de fichero. La función tiene este prototipo:

```
int open ( char* nombre, int modo, int permisos );
```

El parámetro **nombre** es la cadena conteniendo el nombre del fichero que se quiere abrir.

El parámetro **modo** establece la forma en que se va a trabajar con el fichero. Algunas constantes que definen los modos básicos son:

O_RDONLY	abre en modo lectura
O_WRONLY	abre en modo escritura
O_RDWR	abre en modo lectura-escritura
O_APPEND	abre en modo apéndice (escritura desde el final)
O_CREAT	crea el fichero y lo abre (si existía, se lo machaca)
O_EXCL	usado con **O_CREAT**. Si el fichero existe, se retorna un error
O_TRUNC	abre el fichero y trunca su longitud a 0

Para usar estas constantes, se ha de incluir la cabecera **<fcntl.h>**. Los modos pueden combinarse, simplemente sumando las constantes, o haciendo un "or" lógico, como en el ejemplo:

```
O_CREAT | O_WRONLY
```

El parámetro ***acceso*** sólo se ha de emplear cuando se incluya la opción **O_CREAT**, y es un entero que define los permisos de acceso al fichero creado. Consulten en la bibliografía cómo se codifican los permisos.

La función **open** retorna un descriptor válido si el fichero se ha podido abrir, y el valor -1 en caso de error.

Función creat

Si desean expresarmente crear un fichero, disponen de la llamada **creat**. Su prototipo es

`int creat ( char* `*`nombre`*`, int `*`acceso`*` );`

Equivale (más o menos) a llamar a **open (*nombre*, O_RDWR|O_CREAT, *acceso*)**. Es decir, devuelve un descriptor si el fichero se ha creado y abierto, y -1 en caso contrario.

Función close

Para cerrar un fichero ya abierto está la función **close**:

`int close ( int `*`fichero`*` );`

donde **fichero** es el descriptor de un fichero ya abierto. Retorna un 0 si todo ha ido bien y -1 si hubo problemas.

Borrado de ficheros

La función

`int unlink ( char* nombre );`

borra el fichero de ruta ***nombre*** (absoluta o relativa). Devuelve -1 en caso de error.

Lectura y escritura

Para leer y escribir información en ficheros, han de abrirlos primero con **open** o **creat**. Las funciones **read** y **write** se encargan de leer y de escribir, respectivamente:

`int read ( int fichero, void* buffer, unsigned bytes );`

`int write( int fichero, void* buffer, unsigned bytes );`

Ambas funciones toman un primer parámetro, ***fichero***, que es el descriptor del fichero sobre el que se pretende actuar.

El parámetro ***buffer*** es un apuntador al área de memoria donde se va a efectuar la transferencia. O sea, de donde se van a leer los datos en la función ***read***, o donde se van a depositar en el caso de ***write***.

El parámetro ***bytes*** especifica el número de bytes que se van a transferir.

Las dos funciones devuelven el número de bytes que realmente se han transferido. Este dato es particularmente útil en la función ***read***, pues es una pista para saber si se ha llegado al final del fichero. En caso de error, retornan un -1.

Hay que tener especial cautela con estas funciones, pues el programa no se va a detener en caso de error, ni hay control sobre si el puntero ***buffer*** apunta a un área con capacidad suficiente (en el caso de la escritura), etc., etc.

La primera vez que se lee o escribe en un fichero recién abierto, se hace desde el principio del fichero (desde el final si se incluyó la opción **O_APPEND**). El puntero del fichero se mueve al *byte* siguiente al último *byte* leído o escrito en el fichero. Es decir, UNIX trabaja con ficheros secuenciales.

Movimiento del puntero del fichero

El lenguaje C y UNIX manejan ficheros secuenciales. Es decir, conforme se va leyendo o escribiendo, se va avanzando en la posición relativa dentro del fichero. El acceso directo a cualquier posición dentro de un fichero puede lograrse con la función **lseek**.

long lseek (int *fichero*, long *desp*, int *origen*);

Como siempre, ***fichero*** es el descriptor de un fichero y abierto.

El parámetro ***desp*** junto con ***origen*** sirven para determinar el punto del fichero donde va a acabar el puntero. ***desp*** es un entero largo que expresa cuántos *bytes* hay que moverse a partir del punto indicado en ***origen***, parámetro que podrá adoptar estos valores:

0	SEEK_SET	inicio del fichero
1	SEEK_CUR	relativo a la posición actual
2	SEEK_END	relativo al final del fichero

Las constantes simbólicas se encuentran en **<stdlib.h>** y **<unistd.h>**.

El parámetro *desp* puede adoptar valores negativos, siempre que tengan sentido. Si el resultado final da una posición mayor que el tamaño del fichero, éste crece automáticamente hasta esa posición.

La función **`lseek`** devuelve un entero largo que es la posición absoluta donde se ha posicionado el puntero; o un -1 si hubo error. Obsérvese que la función **`lseek`** puede utilizarse también para leer la posición actual del puntero.

Ejemplos

En las siguientes líneas se muestran dos programas que emplean las llamadas al sistema.

Primer programa: creación y escritura

Este ejemplo crea un fichero y escribe en él algunos caracteres.

```
#include <string.h>     /* Función strlen() */
#include <fcntl.h>      /* Modos de apertura y función open()*/
#include <stdlib.h>     /* Funciones write() y close() */

main ( int argc, char* argv[] )
{
   /* Cadena que se va a escribir */
   const char* cadena = "Hola, mundo";

   /* Creación y apertura del fichero */
   int fichero = open ("mi_fichero", O_CREAT|O_WRONLY,0644);

   /* Comprobación de errores */
   if (fichero==-1)
   {
        perror("Error al abrir fichero:");
        exit(1);
   }

   /* Escritura de la cadena */
   write(fichero, cadena, strlen(cadena));
   close(fichero);
   return 0;

}
```

Segundo programa: lectura

Este programa lee diez caracteres, a partir de la posición 400, de un fichero ya existente.

```
#include <fcntl.h>      /* Modos de apertura */
#include <stdlib.h>     /* Funciones de ficheros */

main ( int argc, char* argv[] )
{
   char cadena[11];     /* Depósito de los caracteres */
   int leidos;

   /* Apertura del fichero */

   int fichero = open ("mi_fichero", O_RDONLY);

   /* Comprobación */
   if (fichero==-1)
   {
```

```
        perror("Error al abrir fichero:");
        exit(1);
   }

   /* Coloca el puntero en la posición 400 */
   lseek(fichero,400,SEEK_SET);

   /* Lee diez bytes */
   leidos = read(fichero, cadena, 10);
   close(fichero);
   cadena[10]=0;

   /* Mensaje para ver qué se leyó */
   printf ( "Se leyeron %d bytes. La cadena leída es %s\n",
leidos,cadena );

   return 0;
}
```

CONSIGNAS PRÁCTICAS

Fecha de entrega para las consignas: 25 de octubre, utilizar capturas de pantallas para los ejercicios de línea de comandos. Para programas en C, adjuntar archivos fuente con extensión .c que esté listo para ser compilado.

1. Entrar en el sistema y comprobar cuál es su directorio HOME.
2. Hacer un listado extendido (que incluya archivos ocultos) del directorio HOME.
3. Crear (mediante la orden cat) un archivo en Linux llamado dni que contenga nuestro nombre y número de documento.
4. Ir al directorio HOME. Crear los directorios datos y borrar. Obtener un listado del directorio HOME.
5. Dentro del directorio borrar, crear un subdirectorio llamado nuevo.
6. Cambiar al directorio HOME e intentar borrar el directorio borrar. Puede hacerlo ¿Qué mensaje se obtiene?.
7. Crear un enlace simbólico llamado name al archivo nombre. Luego Mover el archivo nombre al directorio datos (Explique qué ocurre con name). Copiar nombre al directorio borrar.
8. Volver al directorio HOME. Desde este directorio mostrar por pantalla el contenido de los archivos dni y nombre (de los directorio datos y borrar) usando paths absolutos y relativos.
9. Ir al directorio datos. Desde este directorio mostrar el contenido del archivo nombre del directorio borrar usando paths absolutos y relativos.
10. Dé un ejemplo de redirección de la salida de error estándar y de la entrada estándar, explique para qué pueden servir estas acciones.
11. Cambiar los permisos del directorio datos y de todos sus subdirectorios a la siguiente configuración, permiso de lectura y escritura para los tres tipos de usuarios, luego intente acceder al directorio datos y a sus subdirectorios, explique que ocurre y como lo soluciona.
12. Investigue y escriba un programa en lenguaje C que realice una redirección de la salida estándar para el texto “Estoy redireccionando la salida” hacia un archivo llamado exit.txt, el programa debe utilizar las llamadas al sistemas para gestión de archivos y debe manipular los descriptores estándares de archivos en UNIX/Linux. Luego explique detalladamente cómo, su programa, realiza esta tarea solicitada.