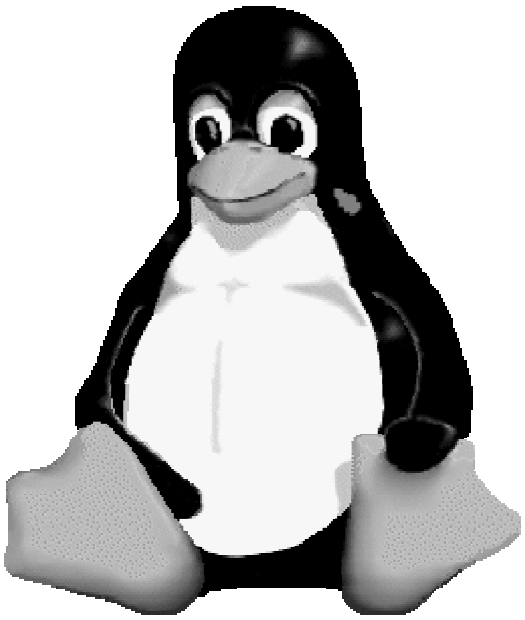
TUTORIAL DE LINUX



El presente tutorial ha sido desarrollado como guía de aprendizaje de la operación básica del Sistema Operativo **Linux**.

Está basado, fundamentalmente en el capítulo 3 del libro "**Linux: Installation and Getting Started**" (Linux: Instalación y Primeros Pasos), de Matt Welsh, traducido al español por el **Proyecto LuCAS**. Dicho libro, además de abundante material en castellano sobre Linux y temas relacionados puede ser obtenido de forma gratuita en http://lucas.ctv.es .

1 ¿Qué es Linux?

UNIX es uno de los sistemas operativos más populares del mundo debido a su extenso soporte y distribución. Originalmente fue desarrollado como sistema multitarea de tiempo compartido para mini ordenadores y mainframes a mediados de los 70 en los laboratorios de AT&T, y desde entonces se ha convertido en uno de los sistemas más utilizados.

¿Cuál es la verdadera razón de la popularidad de **UNIX**? Muchos hackers[1] consideran que **UNIX** es el auténtico y único sistema operativo. El desarrollo de **Linux** parte de un grupo en expansión de hackers de **UNIX** que quisieron hacer su sistema operativocon sus propias manos.

Existen numerosas versiones de **UNIX** para muchos sistemas, desde ordenadores personales hasta supercomputadoras como la Cray Y−MP. La mayoría de las versiones de **UNIX** para ordenadores personales son muy costosas.

Linux es una versión de **UNIX** de libre distribución, inicialmente desarrollada por Linus Torvalds en la Universidad de Helsinki, en Finlandia. Fue desarrollado con la ayuda de muchos programadores y expertos de **UNIX** a lo largo y ancho del mundo, gracias a la presencia de Internet. Cualquiera puede acceder a **Linux** y desarrollar nuevos módulos o cambiarlo a su antojo. El núcleo de **Linux** no utiliza ni una sola línea del código original del **UNIX** de **AT&T** o de cualquier otra fuente de propiedad comercial, y buena parte del software para **Linux** se desarrolla bajo las reglas del proyecto de **GNU** de la Free Software Foundation, Cambridge, Massachusetts[2].

En Marzo de 1992 apareció la primera versión “oficial” de **Linux**. Hoy es ya un clon de **UNIX** completo, capaz de ejecutar X Window, TCP/IP, Emacs, UUCP y software de correo y News. Mucho software de libre distribución ha sido ya portado a **Linux**, y están empezando a aparecer aplicaciones comerciales. El hardware soportado es mucho mayor que en las primeras versiones (inclusive se han desarrollado versiones de **Linux** para otras plataformas además de las IBM−PC compatibles, como Macintosh, SGI, Sparc, Alpha, MIPS, etc.). Mucha gente ha ejecutado tests de rendimiento en sus sistemas Pentium corriendo **Linux** y se han encontrado que son comparables a las estaciones de trabajo de gama media de Sun Microsystems y Digital.

1.1 Características del sistema

Linux implementa la mayor parte de las características que se encuentran en otras implementaciones de **UNIX**, más algunas otras que no son habituales.

Es un sistema operativo completo con multitarea y multiusuario (como cualquier otra versión de **UNIX**). Esto significa que pueden trabajar varios usuarios simultáneamente en él, y que cada uno de ellos puede tener varios programas en ejecución.

El sistema **Linux** fue desarrollado buscando la portabilidad del código fuente: Encontrará que casi todo el software gratuito desarrollado para **UNIX** se compila en **Linux** sin problemas. Y todo lo que se hace para **Linux** (código del núcleo, drivers, librerías y programas de usuario) es de libre distribución.

[1] El término ***hacker*** es comunmente utilizado para referirse a los fanáticos de la programación y las computadoras. Muchas veces se comete el error de llamar hackers a los piratas informáticos.

[2] Básicamente, lo que estas reglas establecen, es que el software en cuestión debe ser distribuído incluyendo todo el código fuente y la documentación. Establece además que cualquier persona puede modificar el software de acuerdo a sus necesidades e inclusive puede redistribuirlo, siempre y cuando lo haga en las mismas condiciones en que lo recibió, es decir, de forma gratuita y sin ocultar el código fuente.

Linux implementa todo lo necesario para trabajar en red con TCP/IP (el protocolo de Internet). Desde manejadores para las tarjetas de red más populares hasta PPP, que permite acceder a una red TCP/IP por el puerto serie (comúnmente, utilizando un Modem y la línea telefónica). Y también se han portado los clientes de TCP/IP, como FTP, telnet, NNTP (News Groups), SMTP y POP3 (E−mail) y HTTP (Web).

2 Conceptos básicos de UNIX

Bajo **UNIX**, para que los usuarios puedan identificarse en el sistema, deben presentarse (*login*), proceso que consta de dos pasos: Introducir el nombre de usuario (el nombre con que será identificado por el sistema), y una palabra clave (*password*), la cual es su llave personal secreta para entrar en la cuenta. En nuestros ejemplos supondremos que el nombre de usuario es ***larry***

En los sistemas **UNIX** tradicionales, el administrador del sistema asignará el nombre de usuario y una palabra clave inicial en el momento de crear la cuenta de usuario. Además, cada sistema **UNIX** tiene un nombre del sistema (*hostname*) asignado, que le da nombre a la máquina. El nombre del sistema es usado para identificar computadoras en una red, pero incluso aunque la máquina no esté en red, debería tener su nombre. En nuestros ejemplos, el nombre del sistema será "*mousehouse*"

2.1 Presentación en el sistema (*loggin in*)

En el momento de presentarse en el sistema, verá la siguiente línea de comandos en la pantalla:

mousehouse login: [3]

Ahora, introduzca su nombre de usuario y pulse **[Return]**[4]. En nuestro ejemplo, debería teclear lo siguiente:

mousehouse login: larry
Password:

Ahora introduzca la palabra clave. Esta no será mostrada en la pantalla conforme se va tecleando, por lo que debe teclear cuidadosamente. Si introduce una palabra clave incorrecta, se mostrará el siguiente mensaje:

Login incorrect

y deberá intentarlo de nuevo.

Una vez que ha introducido correctamente el nombre de usuario y la palabra clave, está oficialmente "presentado" en el sistema y libre para comenzar a trabajar, según los derechos de acceso que le brinde su cuenta, como veremos más adelante.

2.2 Consolas virtuales

[3] Usaremos la siguiente convención: El texto presentado por la computadora será escrito en letras ***itálicas*** , en tanto que lo escrito por el usuario se mostrará en texto corriente.
[4] En algunos teclados ésta tecla puede aparecer como **[Enter]** o **[Intro]**.

La consola del sistema (o terminal) está formada por el monitor y teclado conectado directamente a la computadora[5]. **Linux**, proporciona acceso a consolas virtuales (o VC's), las cuales le permitirán tener mas de una sesión de trabajo activa a la vez desde una única consola.

Para demostrar esto, ingrese en su sistema (como hemos visto antes). Ahora pulse **[ALT]**+**[F2]**. Debería ver la pregunta *login*: de nuevo. Está viendo la segunda consola virtual ya que ha entrado en el sistema por la primera. Para volver a la primera VC, pulse **[ALT]**+**[F1]**[6].

Un sistema **Linux** recién instalado probablemente le permita acceder a las primeras seis VC's, usando **[ALT]**+**[F1]** a **[ALT]**+**[F6]**, pero es posible habilitar hasta 12 VC's, una por cada tecla de función del teclado.

Mientras que el uso de VC's es algo limitado (después de todo, sólo puede mirar una por vez), esto debería darle una idea de las capacidades multiusuario del sistema. Mientras está trabajando en la VC #1, puede conmutar a la VC #2 y comenzar a trabajar en otra tarea, mientras el sistema continúa ejecutando la tarea de la VC #1.

2.3 Intérpretes de comandos y comandos

Un intérprete de comandos (también conocido como "shell") es simplemente un programa que toma la entrada del usuario (p. ej. las órdenes que teclea) y las traduce a instrucciones del Sistema Operativo. Esto puede ser comparado con el *COMMAND.COM* de *DOS*[7], el cual efectúa esencialmente la misma tarea. El intérprete de comandos es sólo una de las interfaces con **UNIX**. Hay muchas interfaces posibles, como la GUI[8] *X Window*, la cual permite ejecutar comandos usando el ratón y el teclado.

Tan pronto como ingresa en el sistema, se ejecuta un intérprete de comandos y Ud. ya puede teclear órdenes al sistema. Veamos un ejemplo. Aquí, ***larry*** entra en el sistema y es situado en el intérprete de comandos:

mousehouse login: larry
Password:
Welcome to Mousehouse!

/home/larry#

"/home/larry#" es el *prompt* (o indicador) del intérprete de comandos, indicando que está listo para recibir órdenes. Tratemos de decirle al sistema que haga algo interesante:

/home/larry# make love
*make: *** No way to make target 'love'. Stop.*
/home/larry#

[5] Como vemos, en las computadoras personales (PC) la consola es única, en tanto que en los sistemas basados en Minicomputadoras y Supercomputadoras, existe la posibilidad de conectar múltiples terminales a la unidad central de procesamiento.

[6] Esto significa mantener presionada la tecla **[Alt]** y, sin soltarla, presionar la tecla **[F1]** liberándola antes que a **[Alt]**.

[7] DOS (Disk Operating System, o Sistema Operativo de Disco): Sistema Operativo desarrollado a principios de los '80 por la empresa Microsoft junto con IBM para la IBM PC. Fue durante muchos años el sistema dominante en dicha plataforma. Originariamente recibió el nombre de PC-DOS y luego se le llamó MS-DOS. Existen además versiones de DOS producidas por de otras empresas, como DR-DOS (conocido despues como Novell DOS y actualmente como Open DOS).

[8] GUI (Graphic User Interface, o Interfaz Gráfica de Usuario): Es un ambiente gráfico que permite la operación del sistema a través de un dispositivo de puntero (generalmente un Mouse).

Bien, como resulta que *make* es el nombre de un programa ya existente en el sistema, el intérprete de comandos lo ejecuta (desafortunadamente, el sistema no está siendo muy amigable).

Esto nos lleva a una cuestión importante: ¿Qué es una orden? ¿Qué ocurre cuando tecleamos “make love”? La primera palabra de la orden, “make”, es el nombre de la orden a ejecutar. El resto de la orden es tomado como argumentos (o parámetros) de la orden. Por ejemplo:

/home/larry# cp foo bar

Aquí, el nombre de la orden es “cp”, y los argumentos son “foo” y “bar”.

Cuando se teclea una orden, el intérprete de comandos hace varias cosas. Primero, busca el nombre de la orden y comprueba si es una orden interna (es decir, una orden que el propio intérprete de comandos sabe ejecutar por si mismo). Hay bastantes órdenes de ese tipo que veremos mas adelante. El intérprete de comandos también comprueba si la orden es un “alias” o nombre sustituto de otra orden. Si no se cumple ninguno de estos casos, el intérprete de comandos busca el programa y lo ejecuta pasándole los argumentos especificados en la línea de comandos.

En nuestro ejemplo, el intérprete de comandos busca el programa llamado *make* y lo ejecuta con el argumento *love*. *make* es un programa usado a menudo para compilar programas grandes, y toma como argumentos el nombre de un “objetivo” a compilar. En el caso de “make love”, ordenamos a *make* que compile el objetivo *love*. Como *make* no puede encontrar un objetivo de ese nombre, falla enviando un mensaje de error y volviendo al intérprete de comandos.

¿Qué ocurre si tecleamos una orden y el intérprete de comandos no puede encontrar el programa de ese nombre? Bien, probémoslo:

/home/larry# eat dirt
eat: command not found
/home/larry#

Bastante simple, si no se puede encontrar el programa con el nombre dado en la orden (aquí “eat”), se muestra un mensaje de error que debería de ser autoexplicativo. A menudo verá este mensaje de error si se equivoca al teclear una orden (por ejemplo, si hubiese tecleado “mkae love” en lugar de “make love”).

2.4 Salida del sistema

Antes de proseguir, deberíamos ver cómo salir del sistema. Desde la línea de comandos usaremos la orden para salir. Hay otras formas, pero esta es la más simple:

/home/larry# exit

2.5 Cambiando la palabra clave

La primera vez que ingrese al sistema lo hará utilizando la palabra clave asignada por el administrador, pero es altamente recomendable que la cambie de inmediato (además, se recomienda realizar este procedimiento de vez en cuando). La orden *passwd* le pedirá su palabra clave vieja y la nueva. Volverá a pedir una segunda vez la nueva para validarla. Tenga cuidado de no olvidar su clave, si eso ocurre, deberá pedirle al administrador del sistema que la modifique por usted.

2.6 Archivos y directorios

En la mayoría de los sistemas operativos (**UNIX** incluido), existe el concepto de archivo[9], el cual es un conjunto de información al que se le ha asignado un nombre.

Ejemplos de archivo son un mensaje de correo, o un programa que puede ser ejecutado. Esencialmente, cualquier cosa almacenada en el disco es guardada en un archivo individual[10].

Los archivos son identificados por sus nombres. Por ejemplo, el archivo que contiene su historial podría ser grabado con el nombre *history−paper*. Estos nombres usualmente identifican el archivo y su contenido de alguna forma significativa para usted. No hay un formato estándar para los nombres de los archivos como lo hay en *DOS* y en otros sistemas operativos; en general estos pueden contener cualquier caracter (excepto /), y están limitados a 256 caracteres de longitud.

Con el concepto de archivo aparece el concepto de directorio. Un directorio es simplemente una colección de archivos. Puede ser considerado como una "carpeta" que contiene muchos archivos diferentes. Los directorios también tienen nombre con el que los podemos identificar. Además, los directorios mantienen una estructura de árbol; es decir, directorios pueden contener otros directorios.

Un archivo puede ser referenciado por su nombre con camino, el cual esta constituido por su nombre, antecedido por el nombre del directorio que lo contiene. Por ejemplo, supongamos que ***larry*** tiene un directorio de nombre *papers* que contiene tres archivos: *history−final*, *english−lit* y *masters−thesis* (cada uno de los tres archivos contiene información sobre tres de los proyectos en los que ***larry*** esta trabajando). Para referirse al archivo *english−lit*, ***larry*** puede especificar su camino:

```
papers/english−lit
```

Como puede ver, el directorio y el nombre del archivo van separados por un caracter /. Por esta razón, los nombres de archivo no pueden contener este caracter. Los usuarios de *DOS* encontrarán esta convención familiar, aunque en ese sistema operativo se usa el caracter \.

Como hemos mencionado, los directorios pueden anidarse uno dentro de otro. Por ejemplo, supongamos que ***larry*** tiene otro directorio dentro de *papers* llamado *notes*, y dentro de ese directorio, tiene un archivo llamado *cheat−sheet*. El camino de este archivo sería:

```
papers/notes/cheat−sheet
```

Por lo tanto, el camino realmente es la "ruta" (*path*) que se debe recorrer para localizar a un archivo. El directorio sobre un subdirectorio dado es conocido como el directorio padre. Aquí, el directorio *papers* es el padre del directorio *notes*.

2.7 El árbol de directorios[11]

La mayoría de los sistemas **UNIX** tienen una distribución de archivos estándar, de forma que los recursos y archivos puedan ser fácilmente localizados. Esta distribución forma el árbol de directorios, el cual comienza en el directorio "/", también conocido como "*directorio raíz*".

[9] Suele utilizarse alternativamente la palabra "fichero", que proviene de la traducción literal del vocablo inglés "file".

[10] Como sinónimos de "grabar" información en un archivo, suelen utilizarse las palabras "salvar" o "guardar".

[11] Los usuarios de MS−DOS notarán que no existe en UNIX el concepto de "unidad" o "disco". La raíz del árbol de directorios es única. Las distintas unidades aparecerán como directorios en el árbol.

Directamente por debajo (dentro) de / hay algunos subdirectorios importantes: */bin*, */etc*, */dev* y */usr*, entre otros. Estos a su vez contienen otros directorios con archivos de configuración del sistema, programas, etc.

En particular, cada usuario tiene un directorio "home". Este es el directorio en el que el usuario guardará sus archivos. En los ejemplos anteriores, todos los archivos de ***larry*** (como *cheat−sheet* y *history−final*) estaban contenidos en el directorio "home" de ***larry***. Usualmente, los directorios "home" de los usuarios cuelgan de */home*[12] y son denominados con el nombre del usuario al que pertenecen. Por lo tanto, el directorio "home" de ***larry*** es */home/larry*.

En la **figura 1** se muestra un árbol de directorio de ejemplo. Este debería darle una idea de cómo está organizado en su sistema el árbol de directorios.

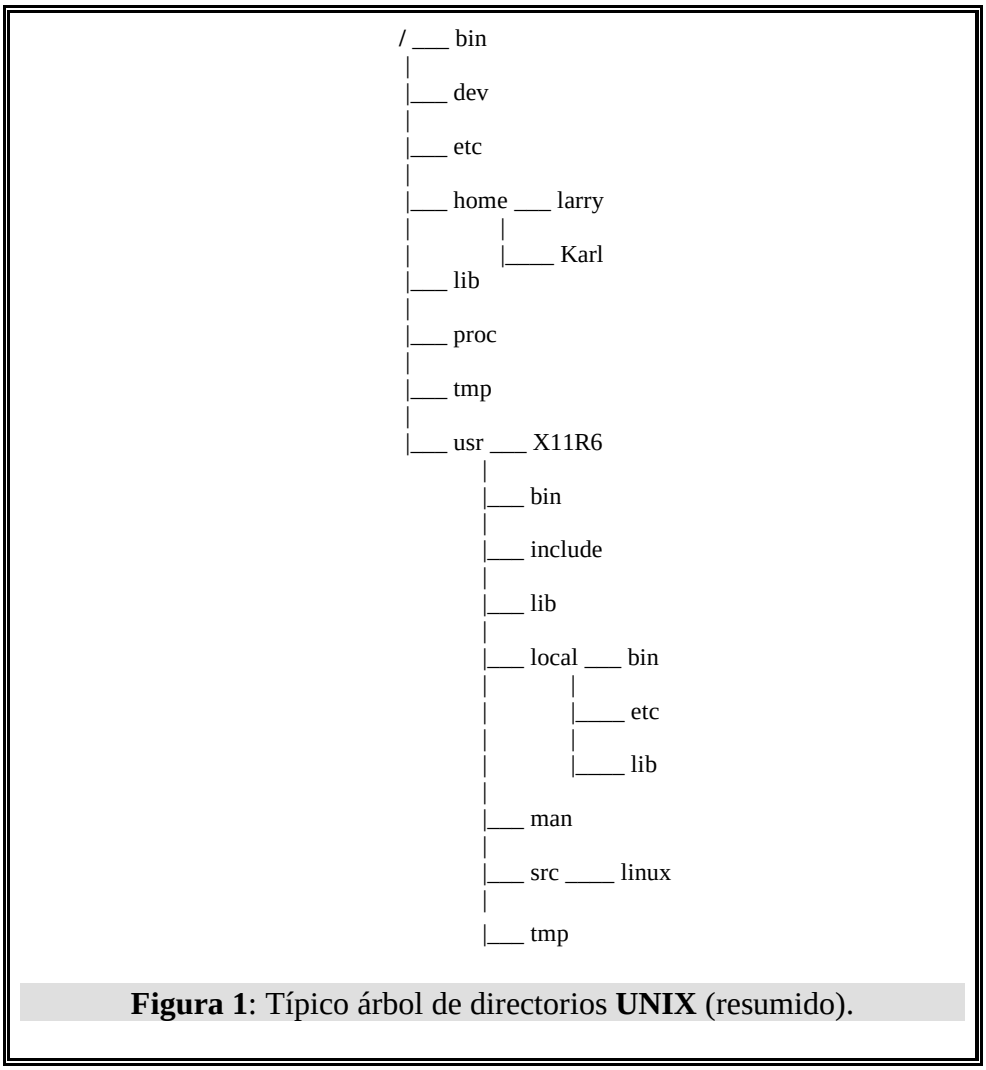


Figura 1: Típico árbol de directorios **UNIX** (resumido).

2.8 Directorio de trabajo actual

Las órdenes que teclee al intérprete de comandos son dadas en términos de su directorio de trabajo actual. Puede pensar en su directorio actual de trabajo como en el directorio en el que actualmente está situado. Cuando entra en el sistema, su directorio de trabajo se inicializa a su

[12] Notar la diferencia que existe entre "home" y ***/home*** . Con "home" nos referimos al directorio personal de un usuario determinado (en el caso de ***larry*** se corresponde con el directorio ***/home/larry***), en tanto que ***/home*** es el directorio en el cual se encuentran todos los directorios personales.

directorio home (*/home/larry* en nuestro caso). Cuando referencie a un archivo puede hacerlo con relación a su directorio de trabajo actual, en lugar de especificar el camino completo del archivo.

Veamos un ejemplo: ***larry*** tiene el directorio *papers*, y *papers* contiene el archivo *history−final*.

Si ***larry*** quiere ver el contenido de ese archivo, puede usar la orden:

/home/larry# more /home/larry/papers/history−final

La orden *more* simplemente muestra el archivo, pantalla a pantalla. Pero como el directorio de trabajo actual de ***larry*** es */home/larry*, podría haberse referido al archivo de forma relativa a su directorio de trabajo actual. La orden sería:

/home/larry# more papers/history−final

Por lo tanto, si comienza el nombre de un archivo (como *papers/history−final*) con un caracter distinto a "/", el sistema supone que se está refiriendo al archivo con su posición relativa a su directorio de trabajo. Esto es conocido como "camino relativo".

Por otra parte, si comienza el nombre del archivo con "/", el sistema interpreta esto como un camino completo es decir, el camino al archivo completo desde el directorio raíz, /. Esto es conocido como "camino absoluto".

2.9 Refiriéndose al directorio home

Bajo tcsh y bash,[13] el directorio "home" puede ser referenciado usando el caracter de la tilde (~). Por ejemplo, la orden:

/home/larry# more ~/papers/history−final

es equivalente a:

/home/larry# more /home/larry/papers/history−final

El caracter "~" es simplemente sustituido por el intérprete de comandos con el nombre del directorio home. Además, también puede especificar otros directorios home de usuarios con la tilde. El camino "~Karl/letters" es traducido por el intérprete de órdenes a "/home/Karl/letters" (si /home/Karl es el directorio home de Karl). El uso de la tilde es simplemente un atajo; no existe ningún directorio llamado "~", es simplemente una ayuda sintáctica proporcionada por el intérprete de comandos.

[13] ***tcsh*** y ***bash*** son dos intérpretes de comandos que corren bajo **Linux**. Un intérprete de comandos es el programa que lee las órdenes del usuario y las ejecuta; la mayoría de los sistemas **Linux** habilitan ***bash*** para las nuevas cuentas de usuario.

3 Primeros pasos en UNIX

Antes de comenzar es importante destacar que todos los nombres de archivos y comandos son “case−sensitive” (que hacen diferencia entre mayúsculas y minúsculas, lo cual no ocurre en sistemas operativos como DOS). Por ejemplo, el comando “make” es diferente a “Make” o “MAKE”. Lo mismo ocurre en el caso de nombres de archivos o directorios.

3.1 Moviéndonos por el entorno

Ahora que ya podemos presentarnos como usuarios y sabemos como indicar archivos con su camino completo, ¿cómo podemos cambiar nuestro directorio de trabajo?

La orden para movernos por la estructura de directorios es *cd*, abreviación de “cambio de directorio”. Hay que destacar, que la mayoría de las órdenes **UNIX** más usadas son de dos o tres letras. La forma de uso de la orden *cd* es:

cd <directorio>

donde <directorio> es el nombre del directorio al que queremos ir.

Como dijimos, al entrar al sistema comenzamos en el directorio “home”. Si ***larry*** quiere ir al subdirectorio *papers*, debería usar la orden:

/home/larry# cd papers
/home/larry/papers#

Como se puede ver, el prompt de ***larry*** cambia para mostrar su directorio actual de trabajo. Ahora que ya está en el directorio *papers* puede ver el contenido del archivo *history−final* con el comando:

/home/larry/papers# more history−final

Ahora ***larry*** está en el subdirectorio *papers*. Para volver al directorio padre de éste, usará la orden:

/home/larry/papers# cd ..
/home/larry#

(Nótese el espacio entre “cd” y “..”). Cada directorio tiene una entrada de nombre “..” la cual se refiere al directorio padre. De igual forma, existe en cada directorio la entrada “.” la cual se refiere a sí mismo. Así que el siguiente comando nos deja donde estamos:

/home/larry/papers# cd .
/home/larry/papers#

También pueden usarse nombres con el camino absoluto en la orden *cd*. Para ir al directorio de Karl con *cd*, introduciremos la siguiente orden:

/home/larry/papers# cd /home/Karl
/home/Karl#

También, usando “cd” sin argumentos nos llevará a nuestro directorio de origen:

/home/Karl# cd
/home/larry#

3.2 Mirando el contenido de los directorios

El simple movimiento por el árbol de directorios es poco útil, necesitamos un nuevo comando, *ls*. Esta orden muestra en pantalla la lista de archivos y directorios, si no se indica otra cosa, del directorio activo. Por ejemplo:

/home/larry# ls
Mail
letters
papers
/home/larry#

Aquí podemos ver que ***larry*** tiene tres entradas en su directorio actual: *Mail*, *letters* y *papers*. Esto no nos dice demasiado ¿son archivos o directorios? Podemos usar la opción "−F" de la orden *ls* para obtener mas información.

/home/larry# ls −F
Mail/
letters/
papers/
/home/larry#

Por el caracter / añadido a cada nombre sabemos que las tres entradas son subdirectorios. La orden "ls −F" puede también añadir al final "*", esto indica que es un archivo ejecutable (programa). Si "ls −F" no añade nada, entonces es un archivo normal, es decir no es ni un directorio ni un ejecutable.

Por lo general cada orden **UNIX** puede tomar una serie de opciones definidas en forma de argumentos. Éstos usualmente comienzan con el caracter "−", tal como vimos antes con "ls –F". La opción "−F" le dice a *ls* que dé información sobre el tipo de cada entrada.

Si a *ls* le pasamos un nombre de directorio, mostrará el contenido de ese directorio:

/home/larry# ls −F papers
english−lit
history−final
masters−thesis
notes/
/home/larry#

Para ver un listado más interesante, veamos el contenido de directorio del sistema */etc*.

/home/larry# ls /etc

Images	*ftpusers*	*lpc*	*rc.new*	*shells*
adm	*getty*	*magic*	*rc0.d*	*startcons*
bcheckrc	*gettydefs*	*motd*	*rc1.d*	*swapoff*
brc	*group*	*mount*	*rc2.d*	*swapon*
brc~	*inet*	*mtab*	*rc3.d*	*syslog.conf*
csh.cshrc	*init*	*mtools*	*rc4.d*	*syslog.pid*
csh.login	*init.d*	*pac*	*rc5.d*	*syslogd.reload*

default *initrunlvl* *passwd* *rmt* *termcap*
disktab *inittab* *printcap* *rpc* *umount*
fdprm *inittab.old* *profile* *rpcinfo* *update*
fstab *issue* *psdatabase* *securetty* *utmp*
ftpaccess *lilo* *rc* *services* *wtmp*

/home/larry#

Los usuarios de *DOS* notarán que los nombres de los archivos pueden ser más largos que 8.3 caracteres y pueden contener puntos en cualquier posición. Incluso es posible que un archivo contenga más de un punto en su nombre[14].

Vayamos al directorio raíz con "*cd* .." y desde allí vayamos al directorio */usr/bin*:

/home/larry# cd ..
/home# cd ..
/# cd usr
/usr# cd bin
/usr/bin#

También podemos movernos dentro de directorios en múltiples pasos, como en cd /usr/bin. Trate de moverse por varios directorios usando *ls* y *cd*. En algunos casos podrá encontrarse el desagradable mensaje de error "*Permission denied*". Esto simplemente es debido a cuestiones de seguridad del **UNIX**. Para moverse o listar un directorio debe de tener permisos para poder hacerlo. Hablaremos sobre ello más adelante.

3.3 Creando directorios nuevos

Es el momento de aprender a crear directorios. Para ello se usa la orden *mkdir*. Pruebe lo siguiente:

/home/larry# mkdir foo
/home/larry# ls −F
Mail/
foo/
letters/
papers/
/home/larry# cd foo
/home/larry/foo# ls
/home/larry/foo#

Acaba de crear un directorio nuevo y moverse a él. Como no hay ningún archivo en el directorio nuevo, veamos como copiar archivos desde un lugar a otro.

3.4 Copia de archivos

La copia de archivos es efectuada por la orden *cp*:

[14] El Sistema Operativo DOS impone como limitación a los nombres de archivos una longitud de 8 caracteres mas 3 de extensión (mediante la cual se identifica el tipo). Además restringe la utilización de determinados caracteres (por ejemplo el punto, que se usa como separador entre el nombre y la extensión).

```
/home/larry/foo# cp /etc/termcap .
/home/larry/foo# cp /etc/shells .
/home/larry/foo# ls −F
shells termcap
/home/larry/foo# cp shells bells
/home/larry/foo# ls −F
bells shells termcap
/home/larry/foo#
```

La orden *cp* copia los archivos listados en la línea de comandos al archivo o directorio pasado como ultimo argumento. Nótese que usamos el directorio "." para referirnos al directorio actual.

3.5 Moviendo archivos

La orden *mv* mueve archivos en lugar de copiarlos. La sintaxis es muy sencilla:

```
/home/larry/foo# mv termcap sells
/home/larry/foo# ls −F
bells sells shells
/home/larry/foo#
```

Nótese como *termcap* ya no existe, en su lugar está el archivo *sells*. Esta orden puede usarse para renombrar archivos, como acabamos de hacer, pero también para mover archivos a directorios diferentes.

Nota: *mv* y *cp* sobreescribirán los archivos destino (si ya existen) sin consultar. Sea cuidadoso cuando mueva un archivo a otro directorio: puede haber ya un archivo con el mismo nombre que será sobreescrito y su contenido se perderá para siempre.

3.6 Borrando archivos y directorios

Para borrar un archivo, use la orden *rm*.

```
/home/larry/foo# rm bells sells
/home/larry/foo# ls −F
shells
/home/larry/foo#
```

En el directorio *foo* sólo ha quedado el archivo *shells*. Nótese que *rm* por defecto no preguntará antes de borrar un archivo, sea cuidadoso.

Una orden relacionada con *rm* es *rmdir*. Esta orden borra un directorio, pero sólo si está vacío. Si el directorio contiene archivos o subdirectorios, nos informará del error.

3.7 Mirando los archivos

Las órdenes *more* y *cat* son usadas para ver el contenido de archivos. *more* muestra el archivo pantalla a pantalla mientras que *cat* lo muestra entero de una vez.

Para ver el contenido del archivo *shells* podemos usar la orden

```
/home/larry/foo# more shells
```

Durante la ejecución de *more* pulse **[Espacio]**[15] para avanzar a la página siguiente y **[b]** para volver a la página anterior. **[q]** finalizará la ejecución de *more*. Hay otros comandos disponibles, los citados son sólo los más básicos.

Salga de *more* y pruebe "cat /etc/termcap". El texto probablemente pasará demasiado rápido como para poder leerlo. El nombre "cat" viene de "concatenar", que es para lo que realmente sirve el programa. La orden *cat* puede ser usada para concatenar el contenido de varios archivos y guardar el resultado en otro archivo. Esto se verá mas adelante.

3.8 Obteniendo ayuda en línea

Prácticamente cada sistema **UNIX** proporciona una utilidad conocida como "páginas de manual". Estas páginas contienen documentación en línea para todas las órdenes del sistema, recursos, archivos de configuración, etc. La orden usada para acceder a las páginas de manual es *man*. Por ejemplo, si está interesado en conocer otras opciones de la orden *ls*, puede escribir:

/home/larry# man ls

y le será mostrada la página de manual para *ls*.

Desafortunadamente para los principiantes, la mayoría de las páginas de manual contienen detalles técnicos de la orden sin ningún ejemplo ni explicación adicional acerca de su uso. Pese a esto, estas páginas son una gran fuente de información que permiten refrescar la memoria si olvidamos la sintaxis de un comando. Pruebe *man* con los comandos que ya hemos tratado y con los que vayamos introduciendo. Notará que alguno de los comandos no tiene página de manual. Esto puede deberse a diferentes motivos. En primer lugar, puede que las páginas no hayan sido escritas aún. En segundo lugar, la orden puede ser interna del intérprete de comandos, o un "alias" (renombre de otro comando), en cuyo caso no tendrán una página propia. Un ejemplo es la orden *cd*, la cual es interna del intérprete de comandos.

4 Sumario de Órdenes Básicas

Esta sección introduce algunas de las órdenes básicas más útiles de un sistema **UNIX**, incluidas las ya cubiertas en las secciones anteriores.

Nótese que las opciones usualmente comienzan con "−" y en la mayoría de los casos se pueden añadir múltiples opciones de una letra con un único "−". Por ejemplo, en lugar de usar "ls −l −F" es posible usar "ls −lF". En lugar de listar todas las opciones disponibles para cada uno de los comandos sólo hablaremos de aquellas más útiles o importantes. De hecho, la mayoría de las órdenes tienen un gran número de opciones. Puede usar *man* para ver las páginas de manual de cada orden, la cual mostrará la lista completa de opciones disponibles.

Nótese también, que la mayoría de las órdenes toman una lista de archivos o directorios como argumentos, denotados como "<archivo1> . . . <archivoN>". Por ejemplo, la orden *cp* toma como argumentos la lista de archivos a copiar, seguidos del archivo o directorio destino. Cuando se copia más de un archivo, el destino debe ser un directorio.

---

cd Cambia el directorio de trabajo actual.
Sintaxis: cd <directorio>
<directorio> es el directorio al que cambiamos. ("." se refiere al directorio actual, ".." al directorio padre.)
Ejemplo: **cd ../foo** pone ../foo como directorio actual.

---

[15] Ésta es la llamada "barra espaciadora", que sirve para insertar un espacio o blanco.

ls Muestra información sobre los archivos o directorios indicados.
Sintaxis: ls <archivo1> <archivo2> ...<archivoN>
Donde <archivo1> a <archivoN> son los archivos o directorios a listar.
Opciones: Ésta orden tiene gran cantidad de opciones. Las más usadas son: −F (muestra información sobre el tipo de archivo) y −l (da un listado "largo" incluyendo tamaño, propietario, permisos, etc.)
Ejemplo: **ls −lF /home/larry** mostrará el contenido del directorio /home/larry.

---

cp Copia archivo(s) en otro archivo o directorio.
Sintaxis: cp <archivo1> <archivo2> ...<archivoN> <destino>
Donde <archivo1> a <archivoN> son los archivos a copiar y <destino> es el archivo o directorio destino.
Ejemplo: **cp ../frog joe** copia el archivo ../frog al archivo o directorio joe.

---

mv Mueve archivo(s) a otro archivo o directorio. Es equivalente a una copia seguida del borrado del original. Puede ser usado para renombrar archivos, como el comando *RENAME* de *DOS*.
Sintaxis: mv <archivo1> <archivo2> ...<archivoN> <destino>
Donde <archivo1> a <archivoN> son los archivos a "mover" y <destino> es el archivo o directorio destino.
Ejemplo: **mv ../frog joe** mueve el archivo ../frog al archivo o directorio joe.

---

rm Borra archivos. Nótese que cuando los archivos son borrados en UNIX, son irrecuperables (a diferencia de *DOS*, donde usualmente se puede recuperar un archivo borrado).
Sintaxis: rm <archivo1> <archivo2> ...<archivoN>
Donde <archivo1> a <archivoN> son los nombres de los archivos a borrar.
Opciones: −i pedirá confirmación antes de borrar un archivo.
Ejemplo: **rm −i /home/larry/joe /home/larry/frog** borra los archivos joe y frog en /home/larry.

---

mkdir Crea directorios nuevos.
Sintaxis: mkdir <dir1> <dir2> ...<dirN>
Donde <dir1> a <dirN> son los directorios a crear.
Ejemplo: **mkdir /home/larry/test** crea el directorio test colgando de /home/larry.

---

rmdir Esta orden borra directorios vacíos. Al usar rmdir, el directorio de trabajo actual no debe de estar dentro del directorio a borrar.
Sintaxis: rmdir <dir1> <dir2> ...<dirN>
Donde <dir1> a <dirN> son los directorios a borrar.
Ejemplo: **rmdir /home/larry/papers** borra el directorio /home/larry/papers si está vacío.

---

man Muestra la página de manual del comando o recurso (cualquier utilidad del sistema que no es un comando, como funciones de librería) dado.
Sintaxis: man <command>
Donde <command> es el nombre del comando o recurso sobre el que queremos obtener la ayuda.
Ejemplo: **man ls** muestra ayuda sobre la orden ls.

---

more Muestra el contenido de los archivos indicados, una pantalla cada vez.
Sintaxis: more <archivo1> <archivo2> ...<archivoN>
Donde <archivo1> a <archivoN> son los archivos a mostrar.
Ejemplo: **more papers/history−final** muestra por la pantalla el contenido del archivo papers/history−final.

---

cat Oficialmente usado para concatenar archivos, cat también es usado para mostrar el contenido completo de un archivo de una vez.
Sintaxis: cat <archivo1> <archivo2> ...<archivoN>
Donde <archivo1> a <archivoN> son los archivos a mostrar.
Ejemplo: **cat letters/from−mdw** muestra por la pantalla el contenido del archivo letters/from−mdw.

---

echo Simplemente envía al terminal los argumentos pasados.
Sintaxis: echo <arg1> <arg2> ...<argN>
Donde <arg1> a <argN> son los argumentos a mostrar.

Ejemplo: **echo "Hola mundo'**' muestra la cadena "Hola mundo".

---

grep Muestra todas las líneas de un archivo dado que coinciden con un cierto patrón.
Sintaxis: grep <patrón> <archivo1> <archivo2> ...<archivoN>
Donde <patrón> es una expresión regular y <archivo1> a <archivoN> son los archivos donde buscar.
Ejemplo: **grep loomer /etc/hosts** mostrará todas las líneas en el archivo /etc/hosts que contienen la cadena "loomer".

---

5 Caracteres comodín

Una característica importante de la mayoría de los intérpretes de comandos en **UNIX** es la capacidad para referirse a más de un archivo usando caracteres especiales. Estos llamados "comodines" le permiten referirse a, por ejemplo, todos los archivos cuyo nombre contiene una determinada secuencia de caracteres.

El comodín "*" hace referencia cualquier cadena de caracteres en el nombre del archivo. Cuando se usa el caracter "*" para referirse al nombre de un archivo, el intérprete de comandos lo sustituye por todas las combinaciones posibles provenientes de los archivos en el directorio al cual nos estamos refiriendo.

Veamos un ejemplo rápido. Supongamos que ***larry*** tiene los archivos *frog*, *joe* y *stuff* en el directorio actual.

/home/larry# ls
frog joe stuff
/home/larry#

Para listar todos los archivos con la letra "o" en su nombre, hemos de usar la orden:

/home/larry# ls *o*
frog joe
/home/larry#

Como puede verse, el comodín "*" ha sido sustituido con todas las combinaciones posibles que coincidían de entre los archivos del directorio actual.

El uso de "*" solo, simplemente se refiere a todos los archivos, puesto que todos los caracteres coinciden con el comodín.

/home/larry# ls *
frog joe stuff
/home/larry#

Veamos unos pocos ejemplos más.

/home/larry# ls f*
frog
/home/larry# ls *ff
stuff
/home/larry# ls *f*
frog stuff
/home/larry# ls s*f
stuff
/home/larry#

El proceso de la sustitución de "*" en nombres de archivos es llamado *expansión de comodines* y es efectuado por el intérprete de comandos. Esto es importante: las órdenes individuales, como *ls*, nunca ven el "*" en su lista de parámetros. Es el intérprete quien expande los comodines para incluir todos los nombres de archivos que se adaptan. Luego la orden:

/home/larry# ls *o*

es expandida para obtener

/home/larry# ls frog joe

Una nota importante: El comodín "*" **no** coincidirá con nombres de archivos que comiencen con un punto (".”). Estos archivos son tratados como **ocultos**. Los archivos de este tipo simplemente no son mostrados en un listado normal de *ls* y no son afectados por el uso del comodín "*".

He aquí un ejemplo. Ya hemos mencionado que cada directorio tiene dos entradas especiales: "." que hace referencia al directorio actual y ".." que se refiere al directorio padre. De cualquier forma, cuando use *ls* esas dos entradas no se mostrarán.

/home/larry# ls
frog joe stuff
/home/larry#

Si usa el parámetro −*a* con *ls* podrá ver nombres de archivos que comienzan con ".". Observe:

/home/larry# ls −a
. .. .bash_profile .bashrc frog joe stuff
/home/larry#

Ahora podemos ver las dos entradas especiales, "." y "..", así como otros dos archivos "ocultos": *.bash_profile* y *.bashrc* . Estos dos archivos son usados en el arranque por *bash* cuando ***larry*** realiza el *login*.

Otro caracter comodín es "?". Este caracter comodín sólo expande un único caracter. Luego "ls ?" mostrará todos los nombres de archivos con un caracter de longitud, y "ls termca?" mostrará "termcap" pero no "termcap.backup". Aquí tenemos otro ejemplo:

/home/larry# ls j?e
joe
/home/larry# ls f??g
frog
/home/larry# ls ????f
stuff
/home/larry#

Como puede ver, los caracteres comodín le permiten referirse a más de un archivo a la vez. En el resumen de órdenes dijimos que *cp* y *mv* pueden copiar o mover múltiples archivos de una vez. Por ejemplo:

/home/larry# cp /etc/s* /home/larry

copiará todos los archivos de /etc que comiencen por “s” al directorio /home/larry.

6 Fontanería UNIX[16]

6.1 Entrada y salida estándar

Muchos comandos **UNIX** toman sus datos de entrada de la llamada “entrada estándar” y envían sus resultados a la “salida estándar” (a menudo abreviadas como “*stdin*” y “*stdout*” respectivamente). Usualmente el sistema está configurado de forma que la entrada estándar es el teclado y la salida estandar la pantalla.

Veamos un ejemplo con el comando *cat*. Normalmente *cat* lee datos de los archivos cuyos nombres se pasan como argumentos en la línea de comandos y envía estos datos directamente a la salida estándar. Luego, al usar el comando:

/home/larry/papers# cat history−final masters−thesis

se mostrará por pantalla el contenido del archivo *history−final* seguido por el contenido del archivo *masters−thesis*.

Si a *cat* no se le pasan nombres de archivos como parámetros, leerá datos de *stdin* y los enviara a *stdout*. Veamos un ejemplo:

/home/larry/papers# cat
Hi there !!!
Hi there !!!
Bye.
Bye.
[Ctrl]+**[D]**
/home/larry/papers#

Como se puede ver, cada línea que el usuario teclea (impresa en itálica) es inmediatamente reenviada al monitor por *cat*. Cuando se está leyendo de la entrada estándar, los comandos reconocen el fin de la entrada de datos cuando reciben el caracter *EOT* (*end−of−text*, fin de texto). Normalmente es generado con la combinación **[Ctrl]**+**[D]**.

Veamos otro ejemplo. El comando *sort* toma como entrada líneas de texto (de nuevo leerá desde *stdin* si no se le proporcionan nombres de archivos en la línea de comandos), y devuelve la salida ordenada a *stdout*. Pruebe lo siguiente:

/home/larry/papers# sort
bananas
carrots
apples
[Ctrl]+**[D]**
apples
bananas
carrots
/home/larry/papers#

[16] Suele hablarse de la "fontanería" o "plomería" de UNIX debido a que este sistema operativo provee una serie de herramientas llamadas "caños" o "tuberías".

6.2 Redireccionando la entrada y salida

Ahora, supongamos que queremos que la salida de *sort* vaya a un archivo para poder grabar la lista ordenada de salida. El intérprete de comandos nos permite redireccionar la salida estándar a un archivo usando el símbolo “>“. Veamos como funciona:

/home/larry/papers# sort > shopping−list
bananas
carrots
apples
[Ctrl]+**[D]**
/home/larry/papers#

Como puede ver, el resultado de *sort* no se muestra por pantalla, en su lugar es grabado en el archivo *shopping−list*. Veamos su contenido:

/home/larry/papers# cat shopping−list
apples
bananas
carrots
/home/larry/papers#

Supongamos ahora que teníamos nuestra lista desordenada original en el archivo *items.* Una forma de ordenar la información y grabarla en un archivo podría ser darle a *sort* el nombre del archivo a leer en lugar de la entrada estándar y redireccionar la salida estándar como hicimos arriba.

/home/larry/papers# sort items > shopping−list
/home/larry/papers# cat shopping−list
apples
bananas
carrots
/home/larry/papers#

Hay otra forma de hacer esto. No sólo puede ser redireccionada la salida estándar, también puede ser redireccionada la entrada estándar usando el símbolo “<“.

/home/larry/papers# sort < items
apples
bananas
carrots
/home/larry/papers#

Técnicamente, “sort < items” es equivalente a “sort ítems”, pero nos permite demostrar que “sort < items” se comporta como si los datos del archivo fueran tecleados por la entrada estándar. El intérprete de comandos es quien maneja las redirecciones. *sort* no recibe el nombre del archivo a leer, desde su punto de vista, está leyendo datos de la entrada estándar como si fueran tecleados desde el teclado.

Esto introduce el concepto de “filtro”. Un filtro es un programa que lee datos de la entrada estándar, los procesa de alguna forma y devuelve los datos procesados por la salida estándar. Usando la redirección, la entrada estándar y/o la salida estándar pueden ser referenciadas desde archivos. *sort* es un filtro simple: ordena los datos de entrada y envía el resultado a la salida

estándar. *cat* es incluso más simple, no hace nada con los datos de entrada, simplemente envía a la salida cualquier cosa que le llega.

6.3 Uso de tuberías (*pipes*)

Ya hemos visto como usar *sort* como un filtro. Pero estos ejemplos suponen que tenemos los datos en un archivo en alguna parte o vamos a introducir los datos manualmente por la entrada estándar.

¿Qué pasa si los datos que queremos ordenar provienen de la salida de otro comando, como *ls*? Por ejemplo, si el contenido de nuestro directorio actual fuese:

```
/home/larry/papers# ls
english−list
history−final
masters−thesis
notes
/home/larry/papers#
```

Usando la opción −r con *sort* ordenaremos los datos en orden inverso. Si queremos listar los archivos en el directorio actual en orden inverso, una forma podría ser:

```
/home/larry/papers# ls > file−list
/home/larry/papers# sort −r file−list
notes
masters−thesis
history−final
english−list
/home/larry/papers#
```

Aquí, guardamos la salida de *ls* en un archivo, y entonces ejecutamos *sort −r* sobre ese archivo. Pero esta forma necesita crear un archivo temporal en el cual grabar los datos generados por *ls*.

La solución es usar los “pipes” (o “tuberías”). El uso de pipes es otra característica del intérprete de comandos que nos permite conectar una cadena de comandos, de manera que la *stdout* del primero es enviada directamente a la *stdin* del segundo y así sucesivamente. Para crear un pipe se usa el símbolo “|”. En nuestro ejemplo, queremos conectar la salida de *ls* con la entrada de *sort* :

```
/home/larry/papers# ls | sort −r
notes
masters−thesis
history−final
english−list
/home/larry/papers#
```

Esta forma es mas corta y obviamente mas fácil de escribir.

Veamos otro ejemplo útil. Al usar el comando:

```
/home/larry/papers# ls /usr/bin
```

se mostrará una lista de archivos demasiado extensa, parte de la cual pasará rapidamente por la pantalla ante nuestros ojos, sin que podamos leerla. En lugar de esto, usemos *more* para detener el listado cada vez que se complete la pantalla.

```
/home/larry/papers# ls /usr/bin | more
```

Ahora podemos ir avanzando línea por línea o pantalla por pantalla, cómodamente.

Como dijimos anteriormente, podemos "entubar" más de dos comandos a la vez. *head* es un filtro que muestra las primeras líneas de la entrada. Si queremos ver el ultimo archivo del directorio actual en orden alfabético, usaremos:

```
/home/larry/papers# ls | sort −r | head −1
notes
/home/larry/papers#
```

Donde "head –1" simplemente muestra la primera línea de la entrada que recibe (en este caso, el flujo de datos ordenados inversamente provenientes de *ls*, a través del *pipe*).

6.4 Redirección no destructiva

El uso de ">" para redireccionar la salida a un archivo es destructivo. En otras palabras, el comando:

```
/home/larry/papers# ls > file−list
```

sobreescribe el contenido del archivo *file−list*. Si en su lugar, usamos el símbolo ">>", la salida será añadida al final del archivo nombrado, en lugar de ser sobrescrito. El comando:

```
/home/larry/papers# ls >> file−list
```

añadirá la salida de ls al final de file−list.

7 Permisos de Archivos

7.1 Conceptos de permisos de archivos

Al ser **UNIX** un sistema multiusuario, para proteger los archivos de usuarios particulares de la manipulación por parte de otros usuarios, proporciona un mecanismo conocido como "permisos de archivos". Este mecanismo permite que archivos y directorios "pertenezcan" a un usuario en particular. Por ejemplo, como ***larry*** creó archivos en su directorio "home", ***larry*** es el propietario de esos archivos y tiene acceso a ellos.

UNIX también permite que los archivos sean compartidos entre usuarios y grupos de usuarios. Si ***larry*** lo desea, podría restringir el acceso a sus archivos de forma que ningún otro usuario pueda acceder a ellos. De cualquier modo, en la mayoría de los sistemas por defecto se permite que otros usuarios puedan leer tus archivos pero no modificarlos o borrarlos.

Como hemos explicado arriba, cada archivo pertenece a un **usuario** en particular. Por otra parte, los archivos también pertenecen a un **grupo** en particular, que es un conjunto de usuarios

definido por el sistema. Cada usuario pertenece al menos a un grupo. El administrador del sistema puede hacer que un usuario pertenezca a más de un grupo.

Los grupos usualmente son definidos por el tipo de usuarios que acceden a la computadora. Por ejemplo, en un sistema **UNIX** de una universidad, los usuarios pueden ser divididos en los grupos *estudiantes*, *dirección*, *profesores* e *invitados*. Hay también unos pocos grupos definidos por el sistema (como *bin* y *admin*) los cuales son usados por el propio sistema para controlar el acceso a los recursos, normalmente los usuarios comunes no pertenecen a estos grupos.

Los permisos están divididos en tres tipos: lectura, escritura y ejecución. Estos permisos pueden ser fijados para tres clases de usuarios: el propietario del archivo, los integrantes del grupo al que pertenece el propietario y todos los demás usuarios.

El permiso de lectura permite a un usuario leer el contenido del archivo o en el caso de un directorio, listar el contenido del mismo (usando *ls*).

El permiso de escritura permite a un usuario escribir y modificar el archivo (inclusive, eliminarlo). Para directorios, el permiso de escritura permite crear nuevos archivos o borrar archivos ya existentes en dicho directorio.

Por último, el permiso de ejecución permite a un usuario ejecutar el archivo si es un programa o "script"[17] del intérprete de comandos. Para directorios, el permiso de ejecución permite al usuario cambiar al directorio en cuestión con *cd*.

7.2 Interpretando los permisos de archivos

Veamos un ejemplo del uso de permisos de archivos. Usando el comando *ls* con la opción "*–l"* se mostrara un listado "largo" de los archivos, el cual incluye los permisos de archivos.

```
/home/larry/foo# ls −l stuff
−rw−r−−r−− 1 larry users 505 Mar 13 19:05 stuff
/home/larry/foo#
```

El primer campo impreso en el listado representa los permisos de archivos. El tercer campo es el propietario del archivo (***larry***), y el cuarto es el grupo al cual pertenece el archivo (***users***). El último campo es el nombre del archivo (*stuff*).

La cadena "−rw−r−−r—" nos informa, por orden, de los permisos para el propietario, el grupo del archivo y cualquier otro usuario.

El primer caracter de la cadena de permisos ("−") representa el tipo de archivo. El "–" significa que es un archivo regular, "d" indicaría que se trata de un directorio. Las siguientes tres letras ("rw−") representan los permisos para el propietario del archivo, ***larry***. El "r" para lectura y "w" para escritura. Luego ***larry*** tiene permisos de lectura y escritura para el archivo *stuff*.

Como ya mencionamos, aparte de los permisos de lectura y escritura esta el permiso de ejecución, representado por una "x". Como hay un "−" en lugar del "x", significa que ***larry*** no tiene permiso para ejecutar ese archivo. Esto es correcto, puesto que *stuff* no es un programa. Por supuesto, como el archivo es de ***larry***, él puede darse a sí mismo permiso de ejecución si lo desea, como veremos más adelante.

Los siguientes tres caracteres, "r−−", representan los permisos para los miembros del grupo. El grupo al que pertenece el archivo es ***users***. Como solo aparece una "r" cualquier usuario que pertenezca al grupo ***users*** puede leer este archivo, pero no modificarlo ni ejecutarlo.

Los últimos tres caracteres, también "r−−", representan los permisos para cualquier otro usuario del sistema (que no sea ***larry*** ni pertenezca al grupo ***users***). De nuevo, como sólo está presente la "r", los demás usuarios pueden leer el archivo, pero no escribir en él o ejecutarlo.

[17] Un "script" o "guión" es un archivo que contiene una secuencia de comandos del intérprete de comandos.

Aquí tenemos otros ejemplos de permisos de grupo.

−rwxr−xr−x El propietario del archivo puede leer, escribir y ejecutar el archivo. Los usuarios pertenecientes al grupo del archivo y todos los demás usuarios pueden leer y ejecutar el archivo.

−rw−−−−−−− El propietario del archivo puede leer y escribir. Nadie más puede acceder al archivo.

−rwxrwxrwx Todos los usuarios pueden leer, escribir y ejecutar el archivo.

7.3 Dependencias

Es importante remarcar que los permisos de un archivo también dependen de los permisos del directorio en el que reside. Por ejemplo, aunque un archivo tenga los permisos "–rwxrwxrwx", otros usuarios no podrán acceder a él a menos que también tengan permiso de lectura y ejecución para el directorio en el cual se encuentra el archivo. Si *larry* quiere restringir el acceso a todos sus archivos, podría simplemente poner los permisos de su directorio "home" */home/larry* como "drwx−−−−−−". De esta forma ningún usuario podrá acceder a su directorio ni a ninguno de sus archivos o subdirectorios. Así ***larry*** no necesita preocuparse de los permisos individuales de cada uno de sus archivos.

En otras palabras, para acceder a un archivo, debes de tener permiso de ejecución de todos los directorios a lo largo del camino de acceso al archivo, además de permiso de lectura (o ejecución) del archivo en particular.

Habitualmente, los usuarios de un sistema **UNIX** son muy abiertos con sus archivos. Los permisos que se dan a los archivos usualmente son "−rw−r−−r−−", lo que permite a todos los demás usuarios leer los archivos, pero no modificarlos. Los directorios usualmente tienen los permisos "drwxr−xr−x", lo que permite que los demás usuarios puedan moverse y ver los directorios, pero sin poder crear o borrar nuevos archivos en ellos.

Muchos usuarios pueden querer limitar el acceso de otros usuarios a sus archivos. Poniendo los permisos de un archivo como "−rw−−−−−−−" no se permitirá a ningún otro usuario acceder al archivo.

Igualmente, poniendo los permisos del directorio como "drwx−−−−−−" no se permitirá a los demás usuarios acceder al directorio en cuestión.

7.4 Cambiando permisos

El comando *chmod* se usa para establecer los permisos de un archivo. Sólo el propietario puede cambiar los permisos del archivo (además, claro está, del administrador del sistema). La sintaxis de *chmod* es:

chmod {a,u,g,o}{+,−}{r,w,x} <archivos>

El primer parámetro indica a qué usuarios afecta: *all*, *user*, *group* u *other* (todos, el propietario, el grupo u otros usuarios; respectivamente). Luego se especifica si se están añadiendo permisos (+) o quitándolos (−). El tercer parámetro especifica qué tipo de permiso estamos añadiendo o quitando: *read*, *write* o *execute*. Finalmente, se indican los nombres de los archivos a afectar. Algunos ejemplos de la utilización de *chmod* son:

chmod a+r stuff Da a todos los usuarios acceso de lectura al archivo *stuff*.

chmod +r stuff	Igual al anterior. Si no se indica *a*, *u*, *g* u *o* por defecto se toma *a*.
chmod og−x stuff	Quita permisos de ejecución de *stuff* a todos los usuarios excepto al propietario.
chmod u+rwx stuff	Permite al propietario leer, escribir y ejecutar el archivo *stuff*.
chmod o−rwx stuff	Quita permisos de lectura, escritura y ejecución del archivo *stuff* a todos los usuarios menos al propietario y a los usuarios de su mismo grupo.
chmod og–r stuff sells	Quita permisos de lectura de los archivos *stuff* y *sells* a todos los usuarios excepto al propietario.

8 Utilidades

Existe un programa que es especialmente útil para la manipulación de archivos, evitándonos tener que conocer (y recordar) cada uno de los comandos. Los usuarios de *DOS* seguramente recordarán al Norton Commander (NC). Se encuentra disponible para **Linux** el Midnight Commander (MC), una réplica casi exacta del anterior en cuanto a su interfaz, pero que además provee muchas otras facilidades propias del mundo **UNIX**.
Es realmente recomendable la utilización de este programa para la realización de ciertas tareas, pero hay que tener en cuenta que de ninguna manera reemplaza al intérprete de comandos ya que no nos brindan la flexibilidad ni la cantidad de opciones de éste último.
Con relación a los editores de texto, **Linux** dispone (como todo **UNIX**) del famoso (y antiguo) *vi*, además del poderosísimo *emacs*. Ocurre que estos editores disponen de una gran cantidad de opciones, generalmente accesibles mediante combinaciones de teclas, lo cual hace que su aprendizaje sea algo lento y tedioso (aunque quienes llegan a dominarlos dicen que vale la pena el trabajo de aprender a usarlos). Afortunadamente para los usuarios que provengan de *DOS*, se encuentra disponible en **Linux** el editor *joe*, que se opera mediante las mismas combinaciones que el conocido *WordStar*.