

Universidad Nacional Autónoma de México

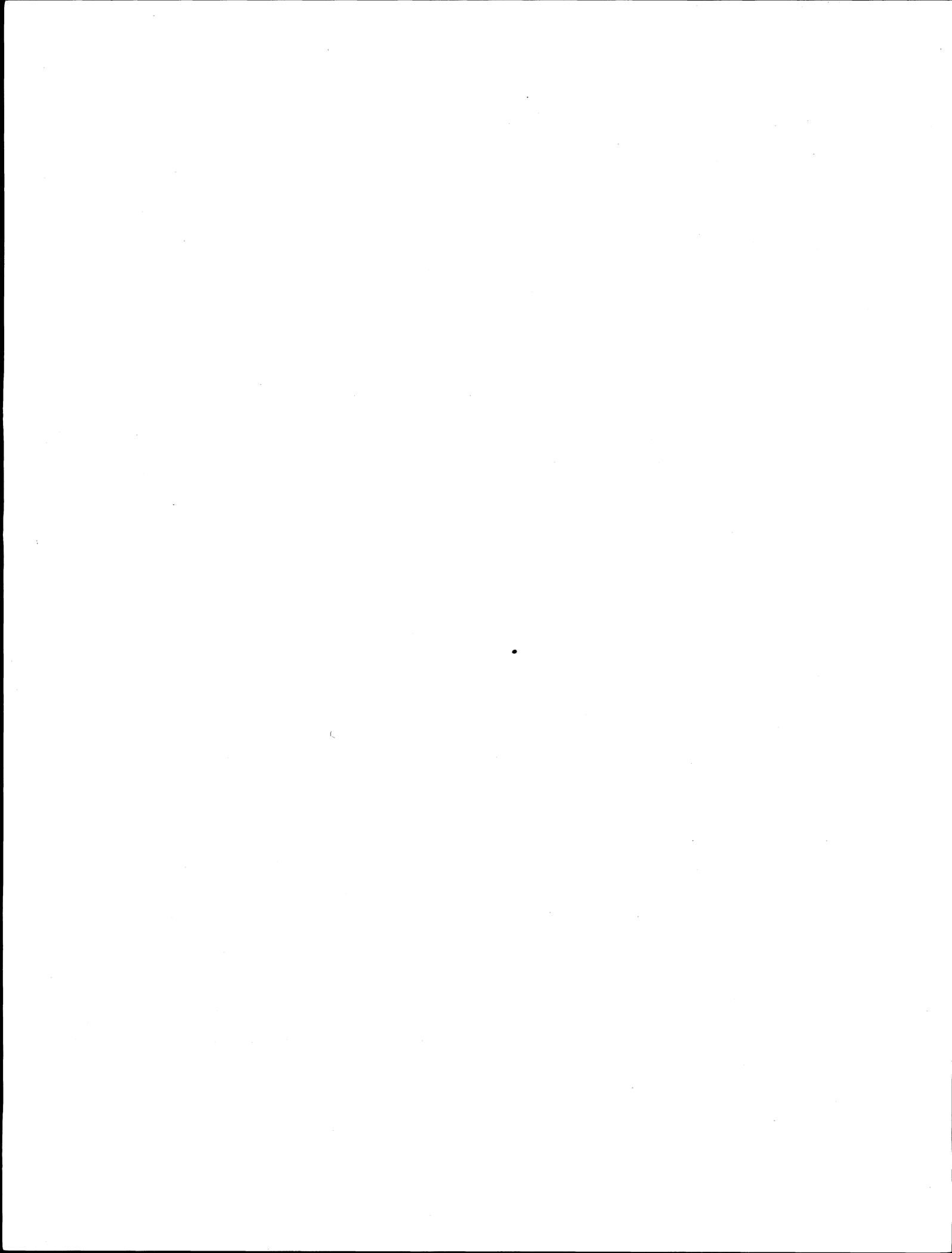
Facultad de Ingeniería

Secretaría General

CECAFI

UNIX

**Ricardo Alvarez Quiroz
Luis Alfonso Amezcua Aragón
Octubre, 1993**

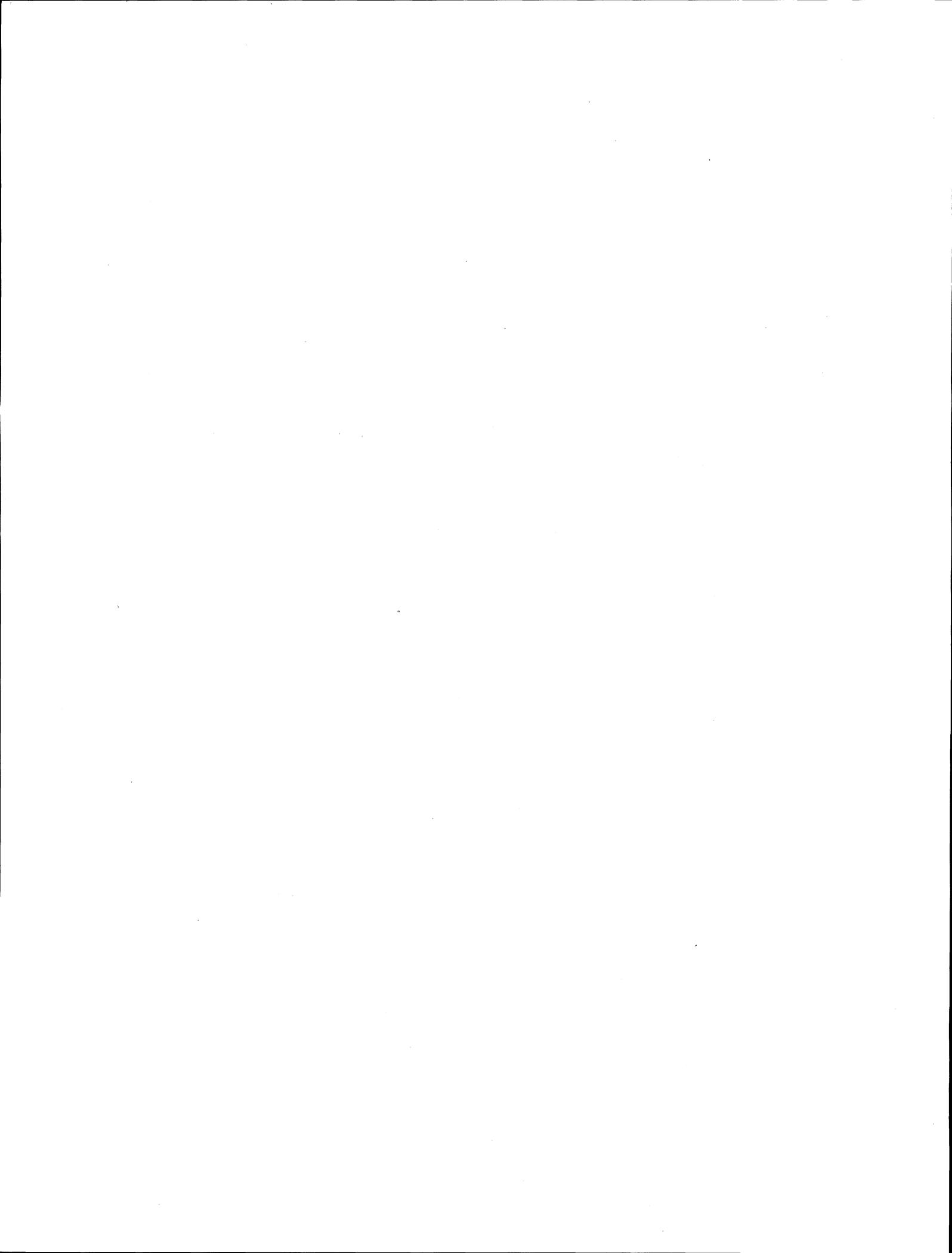


CONTENIDO

I. Introducción	1-1
Historia de Unix	1-2
Características y componentes de Unix	1-4
Inicio de sesión (logging in)	1-7
Sesión gráfica	1-10
Usando la línea de comandos	1-11
Fin de sesión (logging out)	1-14
II. El sistema de archivos	2-1
Archivos	2-1
Organización de archivos	2-2
Elección de nombres de archivo	2-3
Directorios	2-6
Tipos de archivos del sistema Unix	2-7
Archivos ordinarios	2-8
Directorios	2-8
Ligas	2-8
Ligas simbólicas	2-9
Archivos especiales	2-9
La estructura jerárquica de los archivos	2-10
Nombres de rutas	2-11
Nombres de rutas relativas	2-13
Especificación del directorio actual de trabajo	2-13
Especificación del directorio padre	2-13
III. Trabajando con archivos y directorios	3-1
Listado del contenido de un directorio	3-1
Examen de archivos	3-5
Combinación de archivos utilizando cat	3-7
Creación de un archivo con cat	3-9
Cambio de directorios	3-10

IV. Operaciones con archivos y directorios	
Manipulación de archivos	4-1
Movimiento y renombrado de archivos	4-1
Prevención de errores con mv	4-2
Creación de un directorio	4-4
Movimiento de directorios	4-4
Copia de archivos	4-5
Copia del contenido de un directorio	4-5
Eliminación de archivos	4-6
Eliminación de múltiples archivos.	4-7
Restauración de archivos	4-7
Ligas entre archivos	4-10
Ligas simbólicas	4-10
Permisos	4-12
La orden chmod	4-14
Fijación de las autorizaciones absolutas	4-17
Utilización de umask para fijar autorizaciones	4-19
Cambio de propietario de un archivo	4-21
Búsqueda de archivos	4-23
Ejecución de find en background	4-24
Otros criterios de búsqueda	4-26
	4-27
V. Filtros e interconexión de comandos	
Redireccionamiento de entrada/salida	5-1
Filtros	5-1
Interconexión de comandos	5-3
Expresiones regulares con grep y egrep	5-10
	5-12
VI. Edición de archivos	
El editor vi	6-1
Configuración del editor	6-1
Macros	6-6
	6-9
VII. Programación con AWK	
Sintaxis	7-1
Parámetros	7-1
Operaciones aritméticas	7-3
Control de flujo	7-8
Arreglos	7-11
La nueva versión de AWK	7-15
	7-18

VIII. Shell Scripts	8-1
Ejecutando Shell Scripts.	8-3
Usando Variables dentro de los shell script.	8-5
Diferencias entre ' , " y `	8-9
Usando Archivos Temporales.	8-10
Control de Flujo.	8-12
Accesando la Lista de Argumentos.	8-25
Usando Variables de Ambiente.	8-28
Búsqueda de Comandos	8-30
Características Especiales de las Variables de Shell	8-31
Depurando los shell scripts	8-33
Usando Metacaracteres.	8-35
IX. Manejo de Procesos	9-1
El shell	9-1
Archivos de configuración de C shell	9-2
Agrupando comandos del shell	9-5
El comando ps	9-9
Ejecución de procesos en Background	9-11
Deteniendo y reanudando procesos	9-13
X. Manejo de Cintas y Discos	10-1
Usando el comando tar	10-2
Uso del comando cpio	10-4
Transferencia de archivos entre Unix y MS-DOS	10-6
Impresión de archivos	10-9
XI. Comunicándose con otros usuarios	11-1
Correo Electrónico.	11-1
Enviando Correo.	11-2
Editando el correo.	11-2
Funciones ~(Tilde) de Escape Adicionales	11-3
Configurando el Archivo de Correo .mailrc	11-4
El Comando set	11-5
Alias	11-5
Leyendo el Correo	11-6
Comandos de Mail adicionales	11-7
El archivo "Mailbox"	11-10
Folders	11-10
La Utilería write	11-12
Negando la escritura de mensajes	11-13
El editor ed	A-1



I. Introducción

El desarrollo tecnológico en los últimos años a sido muy acelerado, tanto que en pocos años, lo que eran antes las veloces computadoras, basadas en tecnologías muy novedosas para su época, ahora son máquinas dignas de un museo de computación. Tecnologías como los procesadores 80286, 80386, VAX y otros, han sido substituidas por tecnologías basadas en procesadores RISC. Sin embargo, dentro de todo este cambio tecnológico, se a mantenido vigente durante mucho tiempo un sistema operativo que se ajusta al hardware y no el hardware a él, lo cual lo a convertido en unos de los sistemas operativos más usados. Además, la necesidad de grandes intercambios de información ha ocasionado el surgimiento de las redes de computadoras, que permiten la interconexión local y remota de computadoras de diferentes tecnologías.

Dentro de este panorama, es necesario establecer una serie de estándares que permitan el desarrollo de sistemas portables, así como el intercambio de información. Al conjunto de dichos estándares se les conoce actualmente como *sistemas abiertos* (Open Systems). Un sistema abierto es aquel que ha sido adoptado en diferentes arquitecturas de hardware y software debido a que sus especificaciones de diseño son públicas y por lo tanto se ha estandarizado. Se pueden mencionar a productos como el conjunto de protocolos *TCP/IP* (Transmission Control Protocol/Internet Protocol) en el área de redes y *X Window System* dentro de las interfases de usuario.

En el campo de los sistemas operativos, también es necesario establecer un estándar que permita a máquinas con diferentes arquitecturas utilizar un mismo sistema operativo, para que de esta forma todos los desarrollos de aplicaciones se den en un mismo ambiente y se de paso a sistemas totalmente portables. El sistema operativo Unix se ha convertido en uno de los entornos de programación más populares y utilizados del mundo. Actualmente, miles de computadoras, que van desde las microcomputadoras hasta las supercomputadoras, utilizan Unix. Se puede afirmar que Unix es el sistema operativo que se ha establecido como sistema abierto hasta el momento.

Podemos distinguir dos razones principales por las cuales se ha hecho tan popular. Primero, esta escrito en lenguaje C, lo que lo hace portátil; su ambiente de programación es de extraordinaria

UNIX

riqueza y productividad lo cual lo convierte en un buen sistema operativo.

La eficiencia de Unix radica en su enfoque de la programación, que constituye una filosofía de cómo utilizar la computadora, y a su consistencia, debido a que el sistema operativo fue diseñado por un grupo muy pequeño de personas.

Historia de Unix

Entre 1965 y 1969, los Laboratorios Bell de AT&T participaron, junto con General Electric y el Instituto Tecnológico de Massachusetts, en el desarrollo del sistema Multics. El sistema fue originalmente diseñado para operar en una computadora GE-645; sin embargo el sistema era demasiado grande y complejo, por lo que los Laboratorios Bell abandonaron el proyecto en 1969.

Sin embargo, el grupo de investigadores de los Laboratorios Bell que participo en el proyecto original, se propuso crear un sistema operativo que fuera lo suficientemente cómodo y rápido y que además facilitara la investigación y desarrollo de programas. La primera implementación de Unix se hizo en una computadora PDP-7 de Digital Equipment Corporation (DEC) y se escribió en lenguaje ensamblador. Participaron Ken Thompson, Rudd Canaday, Doug McIlroy, Joe Ossanna y Dennis Ritchie.

La popularidad de Unix se extendió dentro de los Laboratorios Bell y poco tiempo después era conocido por la mayoría de los investigadores de dichos Laboratorios. Thompson propuso la posibilidad de transportar el nuevo sistema operativo a otras máquinas; este trabajo requería de rehacer el 90% del trabajo realizado y así poder justificar la adquisición de una máquina DEC PDP 11-20. En 1970 Thompson con ayuda de Ritchie trasladó Unix a la PDP-11.

Dennis Ritchie tuvo un papel muy importante en la codificación del núcleo de Unix en lenguaje C, en 1972. Esto ayudó a hacer más portátil y comprensible al sistema. El código de máquina del sistema resultó mayor que la versión en lenguaje ensamblador, pero parte del aumento se debió a la adición del apoyo a la multiprogramación y a la posibilidad de compartir procedimientos reentrantes.

Introducción

El sistema Unix captó inmediatamente la atención de los investigadores de la AT&T y se comenzaron a utilizar máquinas PDP-11 con Unix para el desarrollo de sistemas telefónicos.

En 1975, Unix se había hecho muy popular en las universidades, ya que se instaló en ellas con fines educativos. Esto dio lugar a una serie de innovaciones e implementaciones de Unix dentro de las cuales se encuentra la realizada en la Universidad de California en Berkeley. Esta versión denominada BSD (Berkeley Software Distribution) fue la más difundida y utilizada dentro de la comunidad universitaria de Estados Unidos. La compatibilidad entre las versiones AT&T y BSD sigue siendo hasta cierto punto cuestionable. Las versiones BSD son equiparables con las versiones AT&T, ya que ambas incorporan las mejores innovaciones del otro sistema a sus propias versiones.

El primer sistema Unix en salir al mercado fue la versión cinco. La versión seis apareció en 1975; sin embargo, la versión oficial de Unix fue la séptima, liberada en 1979. Las versiones más populares fueron la sexta y la séptima. Siguiendo una reorganización interna del soporte del sistema Unix, AT&T cambió su numeración a Sistema III y Sistema V; sin embargo el Sistema V dominó ampliamente al Sistema III. El Sistema IV fue utilizado internamente en los Laboratorios Bell, pero se consideró un producto de transición que nunca fue soportado públicamente. A finales de los ochenta, AT&T normalizó el nombre de Sistema V versión 2 y Sistema V versión 3 (SVR2 y SVR3).

En 1980, Microsoft desarrollo XENIX, que constituye la versión de Unix para microcomputadoras con microprocesadores de 16 bits.

En 1980 DEC (Digital Equipment Corporation) sacó al mercado su versión de Unix denominada Ultrix, la cual incorpora todas las facilidades de la versión 4.2 BSD e incorpora mejoras en el área de comunicaciones. En 1983 apareció en el mercado la versión 4.3 de BSD.

Sun Microsystems desarrolló su versión de Unix denominada SunOS basada en la versión 4.2 BSD, la cual soporta facilidades para ambientes gráficos de ventanas e interfase de ratón en un sistema interconectado de estaciones de trabajo.

Las versiones descendientes de BSD y AT&T están siendo constantemente mejoradas; permitiendo integrar las diferentes versiones en una sola. Se espera que las diferentes versiones converjan en una versión única de Unix que pueda funcionar en cualquier arquitectura de computadora.

Características y componentes de Unix

Unix es un sistema operativo con características muy especiales, lo que le ha valido el reconocimiento por parte de los diseñadores de sistemas. Dentro de sus características principales, podemos mencionar las siguientes:

- Sistema *multiusuario* y *multitarea*.
- Las especificaciones de diseño están disponibles públicamente, lo cual hace que se adapte a exigencias particulares.
- Esta escrito en un lenguaje de alto nivel, lo que lo hace portátil.
- Enfoque de programación.
- Hace uso de redireccionamiento de entrada salida, filtros e interconexiones.
- Sistema de archivos sencillo y eficiente.
- Un manejo de archivos consistente.
- Las interfases con los dispositivos periféricos se manejan igual que un archivo.
- Esconde la arquitectura del hardware que lo utiliza.
- Es fácil de utilizar.

Los componentes o la arquitectura de alto nivel de Unix se pueden representar por el diagrama de la figura 1. Si vemos el sistema como un conjunto de capas, el sistema operativo propiamente dicho conforma la capa más interna, esta parte del sistema operativo es la que se comunica directamente con el hardware y se le conoce comúnmente como *kernel* o núcleo. Entre las funciones del

núcleo se pueden mencionar las siguientes: planificación de tareas, administración de recursos del sistema y manejo de memoria.

En la siguiente capa se encuentran las utilerías y programas de aplicación vinculados con el sistema que se encargan de ejecutar una variedad de rutinas y funciones especiales de mantenimiento del sistema. Estas utilerías se comunican con el kernel por medio de unas instrucciones que se conocen como *llamadas al sistema*. Muchas de estas utilerías y programas forman parte de la configuración estándar de Unix y se conocen como *comandos*.

Uno de los programas de aplicación más importantes es el intérprete de comandos o *shell*. Este programa se ejecuta inmediatamente después de que se abre una sesión de Unix, es el proceso que interpreta los comandos que teclea el usuario y ejecuta los procesos asociados con dichos comandos; de esta forma se lleva a cabo la comunicación entre el usuario y el sistema operativo.

Existen varios tipos de shell entre los que se encuentran *Bourne shell* (sh), *C-shell* (csh), *Reduced shell* (rsh), *Korn shell* (ksh), *Key shell* (keysh) y *Visual shell*. El usuario puede desarrollar su propio interprete de comandos para que sea este el que lo comunique con el núcleo.

En la capa superior de la arquitectura de Unix se encuentran los programas de aplicación que no están comprendidos dentro de la configuración estándar de Unix, es decir, programas creados por nosotros los usuarios.

Cabe recordar cuatro filosofías de trabajo dentro de los sistemas Unix, si lo tenemos en mente, nuestro trabajo será muy fácil de realizar:

- **Unix fué desarrollado por programadores para programadores.**
Es debido a esto que Unix ofrece un rico entorno de programación para la gente que desea desarrollar aplicaciones en este sistema operativo.
- **Unix asume que uno sabe lo que hace.**
Esto es, Unix no preguntará (a menos que se lo solicitemos) sobre si deseamos borrar algún archivo o ejecutar cierto comando. Unix obedece las instrucciones que le damos y las ejecuta al pie de la letra.
- **Unix todo lo ve como un archivo.**
Tanto nuestra información, como los dispositivos asociados a una terminal o disco, son vistos como un archivo dentro de Unix. Esto proporciona una gran ventaja, debido a que no es

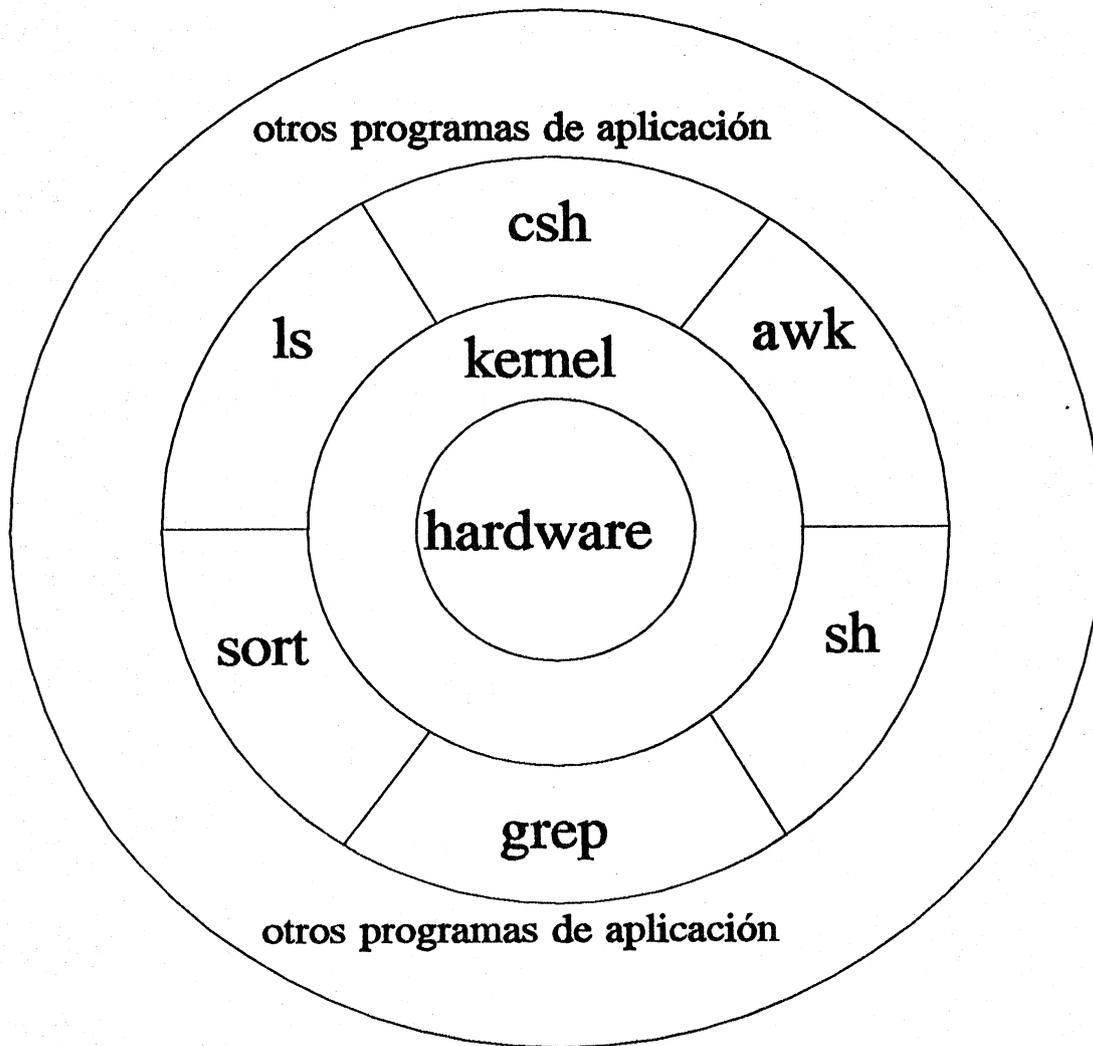


Figura 1. Arquitectura de Unix.

necesario aprenderse los nombres complicados de los dispositivos o la forma en que estos operan, simplemente mandamos leer o escribir al archivo asociado a dicho dispositivo, y listo!.

- Si un comando no manda mensajes de error a la salida estándar, éste se ejecuto satisfactoriamente. Los comandos en Unix, a menos que ocurra un error durante su ejecución, no mandan mensajes sobre su ejecución a la salida estándar (definida generalmente como el monitor).

Inicio de sesión (logging in)

Aquí se describirá la forma de iniciar una sesión en el sistema operativo HP-UX¹, introduciendo algunos conceptos básicos.

Es necesario disponer de una terminal o alguna de las estaciones de trabajo (o Work Station 'WS'). También es necesario disponer de una clave de usuario y una contraseña asociada también denominada "password".

Debido a que HP-UX es un sistema multiusuario, lo primero que hay que hacer para trabajar en él, es identificarnos como un usuario del sistema; este proceso se le conoce como *logging-in* y se lleva a cabo en una pantalla donde se presenta el mensaje *login*:

¹ En las estaciones de trabajo Hewlett Packard (HP) el sistema operativo HP-UX es la versión de Unix de HP, mientras que para las estaciones de trabajo Digital (DEC), la versión de Unix es llamada Ultrix. De aquí en adelante se usarán los términos Unix, Ultrix y HP-UX indistintamente, solamente cuando sea necesario, se aclarará sobre que sistema operativo se está hablando.

UNIX

Una vez que disponemos de una terminal, tal vez sea necesario presionar <RETURN> una o dos veces para establecer la línea de comunicación con la computadora; enseguida deberá aparecer la petición de login. Para identificarnos con el sistema, lo primero que hay que hacer, es proporcionar el nombre de la clave asignada, de esta forma, si la clave es vlu000, se deberá teclear ésta a la petición de login²:

login: vlu000

El usuario proporciona una cuenta válida

Si cometemos errores mecanográficos al momento de teclear la clave, estos se pueden corregir con el uso de la tecla <Backspace>; de lo contrario usaremos la tecla <Return>. Después de haber presionado <Return> la computadora nos pedirá que le proporcionemos nuestra contraseña secreta o password, mediante un letrero que dice: 'password:', mientras tecleamos esta contraseña, por razones de seguridad, esta será desplegada como espacios en blanco mientras la vamos tecleando. Es posible también utilizar la tecla <Backspace> para corregir algún error que pudiésemos cometer mientras escribimos nuestro password.

Login: vlu000

Password: _

Mientras teclea su password, este aparece como una serie de espacios en blanco.

En las estaciones de trabajo Hewlett Packard para borrar un carácter ya sea al estar escribiendo nuestra clave o password hacemos uso de la tecla '#' y para borrar toda la línea usamos la tecla '@' ya que la tecla <Backspace> no borra lo ya escrito.

La contraseña nos sirve para identificarnos como un usuario autorizado a utilizar la clave proporcionada. Una vez que hemos proporcionado la contraseña, se debe teclear <Return>. Si por alguna razón la contraseña no se dio correctamente, la computadora desplegará un mensaje de error. Si esto nos pasa, solamente hay que intentar otra vez el mismo procedimiento. Este procedimiento continuará hasta que se establezca la contraseña correcta.

Cuando la clave y contraseña hayan sido aceptados, normalmente se desplegarán los mensajes de bienvenida y copyright.

² De aquí en adelante, todo lo que el usuario necesite teclear se escribirá en 'negritas', asumiendo además que deberá presionar la tecla <RETURN> al final de cada comando, para iniciar la ejecución del mismo. En algunos casos, puede haber comentarios al comando del lado derecho del documento.

Cada terminal o despliegue gráfico tiene un *tipo* asociado, el cual debe de conocer HP-UX para poder comunicarse con la terminal correctamente. HP-UX nos preguntará cual es el tipo de terminal en donde nos encontramos trabajando. En caso de no estar trabajando sobre una terminal HP, podremos proporcionarle en este momento el tipo de terminal que estamos usando, por ejemplo una terminal vt100, una terminal wayse85, superbrian, etc.

```
TERM = (hp)
```

En caso de estar en una terminal³ HP o si no sabe que tipo de terminal estamos usando, presionemos simplemente la tecla <Return>. El sistema automáticamente fijará valores para la terminal que son apropiados para la mayoría de las terminales.

Si no especificamos ningún valor a la pregunta que nos hace el sistema para establecer el tipo de terminal en la cual estamos trabajando, el valor de la variable TERM será fijado a hpterminal, con lo cual no podremos aprovechar al máximo las capacidades gráficas de nuestra terminal HP. Es por esto que es necesario saber que tipo de terminal se tiene. HP-UX soporta varios tipos de terminales. La base de datos /usr/lib/terminfo le dice a HP-UX la manera de comunicarse con cada tipo de terminal.

³ Si la terminal se comporta de manera extraña, especialmente mientras hace uso del editor vi, la causa puede ser el tipo de terminal que haya definido. Si esto pasa, verifique y si es necesario, vuelva a establecer el valor de la variable de ambiente TERM.

UNIX

La siguiente tabla muestra los valores para las terminales y despliegues gráficos más comunes de los equipos HP. Cuando se presenta más de una opción, todas las demás opciones son equivalentes.

Si está utilizando una ...	Fije el valor de TERM a ...
terminal	El número del modelo: por ejemplo 2622, hp2622, 262x, o 2392
Vectra	2392
Despliegue de media resolución gráfica (512x600 pixels)	3001 o hp3001
Despliegue de alta resolución gráfica (1024x768 pixels)	300h o hp300h
Estación de despliegue HP 98721 (1280x1024 pixels)	98550, hp98550, 98550a o hp98550a
Estación de despliegue HP 98721 (1280x1024 pixels)	98720, hp98720, 98720a, hp98720a, 98721, hp98721, 98721a o hp98721a

Una vez que entramos a sesión, aparecerá el prompt de comandos. El prompt de comandos (o simplemente prompt) indica que el sistema está listo para recibir comandos. Los prompts típicos son \$ ó %.

Sesión gráfica

En caso de haber establecido una sesión desde la consola de las estaciones de trabajo, estos mensajes serán seguidos de una serie de ventanas que forman parte de nuestro ambiente de trabajo. Este ambiente de trabajo es conocido en las estaciones de trabajo HP, como VUE (Visual User Environment). Dentro de este conjunto de ventanas se puede identificar una que dice "CONSOLE" o "HP

Terminal". Es en estas ventanas donde daremos los comandos a la computadora. En la parte inferior de la pantalla se podrá observar un panel con varias divisiones; este panel es llamado el administrador del espacio de trabajo (Work space manager). Con este panel se puede configurar el medio ambiente sobre el cual estamos trabajando, esto es, mientras que las ventanas se pueden considerar como si estuvieran una sobre otra, o distribuidas sobre la pantalla, los espacios de trabajo se consideran como varios ambientes, donde en el primero se tiene una configuración muy específica tanto en ventanas como en colores y aplicaciones, en el segundo ambiente de trabajo se puede tener otra configuración muy diferente a la del primer espacio de trabajo y así cada uno de los seis espacios de trabajo disponibles. El lector interesado puede profundizar más sobre este medio ambiente gráfico consultando los manuales: *HP Visual User Environment User's Guide* y *HP Visual User Environment System Administration Manual* que se encuentran disponibles en el Departamento de Servicios de Cómputo del CECAFI.

Usando la línea de comandos

Para ejecutar o correr un comando, se teclea el nombre del comando después del prompt⁴ y se presiona la tecla <Return>. El comando comenzará entonces a ejecutarse, cuando el comando termine, el prompt volverá a aparecer. Para corregir errores mientras tecleamos el comando, usamos la tecla <Backspace>.

Comenzaremos introduciendo algunos comandos básicos del sistema. Para determinar cómo HP-UX nos identifica dentro del sistema, ejecute el comando *whoami*:

```
% whoami
vlu000                               El sistema nos reconoce como el usuario vlu000
%
```

⁴ Por propósitos de claridad y de consistencia, de aquí en adelante se utilizará el prompt '%'.

UNIX

Para obtener la fecha que tiene establecida el sistema se utiliza el comando `date`:

```
% date
Sun Apr 25 10:55:14 CDT 1993          10:55 del 25 de Abril de 1993
%
```

El comando `cal` nos presenta un calendario del mes actual:

```
% cal
      April 1993
S  M Tu  W Th  F  S
      1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30
%
```

Para saber cuáles usuarios están en sesión en ese momento, usamos el comando `who`:

```
% who
vlu000  console           Apr 16 15:15   :02
charly  ttys0              Apr 16 15:13   :02
cursol15 ttyp2             Apr 15 18:39  47:12
gaby    ttyp3             Apr 16 15:47
ideas   ttyp2             Apr 15 18:58  18:36
raq     ttyp4             Apr 16 15:42
root    ttypa             Apr 16 11:24  18:22
%
```

La primera columna indica el nombre del usuario, la segunda es el nombre del archivo asociado con la terminal desde la cual está conectado el usuario. El resto de la línea indica la fecha y hora en la que el usuario entró en sesión.

Generalmente en todos los sistema Unix, se cuenta con una ayuda en línea. Si deseamos obtener información acerca de algún comando podemos consultar el manual en línea con el comando `man`. Por ejemplo, para obtener información del comando `date` escribiremos lo siguiente:

```
% man date
```

Esto desplegará las hojas del Manual de Unix relacionadas con dicho comando.

Una tarea importante al trabajar en Unix, es establecer una nueva contraseña. La contraseña junto con la clave de usuario son la forma de dar seguridad de acceso al sistema. Si alguien descubre o conoce alguna contraseña, puede entrar a la clave asociada sin ningún problema y modificar la información del dueño de dicha clave e incluso borrar la información que ahí se encontrase o cambiar la contraseña, con lo cual, el dueño legítimo de la clave tendrá negado el acceso al sistema.

Si nuestra clave inicialmente no tiene contraseña, no se desplegará el mensaje de "Password:" al inicio de la sesión. Para establecer o modificar la contraseña se utiliza el comando `passwd`. Es muy importante establecer una contraseña que podamos recordar fácilmente; pero que sea difícil de imaginar por los demás. Generalmente el sistema exige que la contraseña tenga al menos seis caracteres, es recomendable que esta contenga un dígito u otro carácter no alfabético, lo que proporciona un rango amplio y complejo de contraseñas, dando mayor seguridad a las cuentas del sistema.

Si se desea establecer una contraseña para una clave que no la tiene, se utiliza el comando `passwd` de la siguiente forma:

% passwd

New password:

Mientras se escribe el nuevo password este no se despliega

Si nuestra cuenta ya disponía de algún password, entonces el sistema como medida de seguridad, nos preguntará por el password actual de nuestra cuenta, para cerciorarse de que somos los dueños de la misma. Después de haber verificado nuestro password, procederá a preguntarnos cual deseamos que sea nuestro nuevo password y después de esto, nos pedirá que lo re-escribamos para estar seguros de que fue tecleado correctamente:

% passwd

Old password:

Proporcionar el password actual

New password:

Teclear el nuevo password

Re-enter your new password:

Verificar el password proporcionado

Si alguna de las dos entradas al establecer el nuevo password no coinciden, se rechazará la contraseña y esta no quedará establecida.

Por razones de seguridad, es recomendable cambiar la contraseña periódicamente. Para el administrador del sistema esta tarea es obligatoria.

Fin de sesión (logging out)

Para salir de sesión, lo más común es presionar la combinación de teclas <Ctrl><d> o bien, escribiendo desde la línea de comandos el comando *exit*:

```
% exit
```

Esto funcionará siempre y cuando hayamos establecido una sesión de tipo texto (sin el ambiente gráfico VUE), de lo contrario, el efecto que tienen estas dos formas de salir, será el de cerrar la ventana activa y NO EL DE SALIRSE DE SESIÓN.

Para salirse de sesión si se estableció una sesión gráfica, será necesario colocar el cursor del mouse sobre el botón que se encuentra localizado en la esquina inferior derecha del *Work space manager* y presionar el botón izquierdo del mouse. Una vez hecho esto puede aparecer una caja de diálogo, donde se nos pregunte si deseamos salir de sesión o cancelar la operación, en donde deberemos dar nuestra elección.

II. El sistema de archivos

Sobre una buena parte del Sistema Unix subyace un modelo simple. Aprendiendo los componentes de este modelo dispondremos de una forma útil de pensar en la manera de trabajar con un Sistema Unix. Un pilar básico del Sistema Unix es el sistema de archivos jerárquico. El sistema de archivos proporciona una forma potente y flexible de organizar nuestra información contenida en la computadora.

Aunque muchas de las características del sistema de archivos se inventaron originalmente para el Sistema Unix, su estructura ha resultado ser tan útil que se ha adoptado por muchos otros sistemas operativos. Por ejemplo, si estamos familiarizados con MS-DOS, ya conocemos algo del sistema de archivos de Unix, porque DOS adoptó muchos de sus atributos importantes.

Archivos

Un archivo es la estructura básica utilizada para almacenar información en Unix. Conceptualmente un archivo es similar a un documento de papel. Técnicamente un archivo es una secuencia de bytes que se almacenan en algún lugar de un dispositivo de memoria, tal como un disco. Un archivo no tiene por qué ser almacenado en un único sector físico de un disco. De esta forma un archivo puede contener cualquier clase de información que se pueda representar como una secuencia de bytes. Un archivo puede almacenar manuscritos y otros documentos de procesamiento de textos, instrucciones o programas para la propia computadora, una base de datos de información comercial organizada, una descripción de mapa de bits de una imagen de pantalla o un mensaje de fax, o cualquier otra clase de información almacenada como una secuencia de bytes sobre la computadora.

De la misma forma que un documento tiene un título, un archivo tiene un título denominado nombre de archivo. El nombre de archivo identifica de manera unívoca al archivo. Para trabajar con un archivo, sólo necesitamos recordar el nombre de archivo. El sistema

operativo Unix conoce donde está localizado el archivo, además de guardar otro tipo de información sobre el archivo como su tamaño, protecciones, etc.

Organización de archivos

Todo el trabajo de los usuarios de un Sistema Unix está almacenado en archivos. No lleva mucho tiempo para un usuario medio generar docenas, cientos, o incluso miles de archivos. Con miles de archivos, ¿cómo se puede asegurar nombrar de manera correcta uno nuevo (unívocamente)? Una vez nombrado, ¿cómo se puede asegurar el encontrarlo posteriormente?

En poco tiempo podemos tener cientos de archivos con nuestro trabajo. Para encontrar una información específica deberemos buscar a lo largo de los cientos de nombres de archivos, creyendo recordar cómo hemos nombrado al archivo que contiene ésta información. ¿Se encuentra nuestro trabajo de redes del mes de Abril en el archivo `redes.art` o en `redes_news.doc`, o en algún otro archivo?

En algunos sistemas podría ser difícil e incluso imposible encontrar un archivo. El Sistema Unix tiene varias capacidades que facilitan mucho este trabajo. Algunas de estas capacidades tienen que ver con la naturaleza básica del sistema de archivos y se tratarán en este capítulo. Otras tienen que ver con los programas de utilidad disponibles en el sistema Unix, y se tratarán en los capítulos posteriores.

Elección de nombres de archivo

Un nombre de archivo puede ser casi cualquier secuencia de caracteres. (Generalmente dos nombres de archivos se consideran el mismo si coinciden en los primeros catorce caracteres, de manera que debemos tener cuidado si excedemos de este número de caracteres. Sin embargo, esto no ocurre para cierto tipos de archivos). El Sistema Unix coloca pocas restricciones sobre cómo nombrar archivos. Podemos utilizar caracteres ASCII en un nombre de archivo excepto la diagonal (/), que tiene un significado especial en el sistema de archivos de Unix. La diagonal actúa como un separador entre directorios y archivos dentro del sistema.

Aunque cualquier carácter ASCII, diferente de la diagonal, puede ser utilizado en un nombre de archivo, hay que tratar de utilizar caracteres alfabéticos (letras y números) cuando nombremos archivos. Podemos encontrar problemas cuando utilicemos o tratemos de visualizar nombres de archivos que contienen caracteres no alfanuméricos (tales como el carácter de interrogación '?', los signos '+' y '-', o caracteres de control entre otros).

Cuando creamos archivos y directorios, estamos trabajando con un programa importante del sistema Unix denominado: shell. Debemos evitar utilizar caracteres en los nombres de archivos que tengan significado especial para el intérprete de comandos (shell). Los siguientes caracteres pueden ser utilizados en nombres de archivos, pero es mejor evitarlos:

!	(admiración)	@	(arroba)
#	(signo de libra)	\$	(signo de dólar)
&	(ampersand)	^	(a c e n t o circunflejo)
(,)	(paréntesis)	{,}	(llaves)
' , "	(comillas simples o dobles)	*	(asterisco)
;	(punto y coma)	?	(interrogación)
	(cauce)	\	(backslash)
<, >	(símbolo menor y mayor que)	La tecla	SPACEBAR
La tecla	TAB	La tecla	BACKSPACE

UNIX

Cada uno de estos caracteres tiene un significado especial para el shell, y este significado tiene que ser desactivado si deseamos utilizarlos en nombres de archivos.

Más adelante veremos cómo desactivar el significado especial de los caracteres, por ahora simplemente evitemos utilizarlos en los nombres de archivos. También debemos evitar utilizar + (mas) y - (menos) porque estos caracteres tienen significados especiales para el shell cuando se utilizan antes de los nombres de archivos en líneas de comandos.

Al asignar nombres a archivos hay que tener en cuenta lo siguiente:

- Unix hace distinción entre letras mayúsculas y minúsculas. Esto significa que los archivos denominados **NOTES**, **Notes** y **notes** son diferentes.
- No existen convenciones para los nombres de archivos, por lo tanto se puede asignar o no una extensión a un tipo de archivo.
- No existen diferentes versiones de archivos ni tampoco archivos de respaldo.
- Cuando un nombre de archivo comienza con "." (punto), el archivo tiene significado especial para el sistema y generalmente se encuentra oculto.

A continuación se muestra una tabla con algunos de los subdirectorios y archivos más relevantes dentro de un sistema Unix.

/	directorio raíz del sistema de archivos.
/bin	utilerías y comandos esenciales.
/dev	archivos de dispositivos.
/etc	archivos de administración y usos diversos
/lib	bibliotecas.
/usr ó /users	sistema de archivos del usuario.
/etc/adm	directorio con información administrativa.
/tmp	archivos temporales.
/etc/passwd	archivo de usuarios válidos del sistema.
/etc/shutdown	comando para dar de baja el sistema.
/etc/reboot	comando para dar de alta el sistema.
/etc/init	comando que crea el primer proceso en el procedimiento de boot.
/usr/include	archivos de encabezados para C.
/hp-ux	archivo ejecutable (sistema operativo).
/ultrix	" " " "
/vmunix	" " " "
/unix	" " " "
/usr/games	directorio de programas de juegos.

Algunos subdirectorios y archivos relevantes dentro de un sistema Unix.

Como ya hemos mencionado, Unix no impone estructura alguna a los archivos, todos son tratados de la misma forma ya que el sistema los ve solamente como una secuencia de bytes. Sin embargo, podemos determinar el tipo de un archivo con ayuda del comando *file*. Este comando examina el archivo y de acuerdo al tipo de caracteres que contenga determina si se trata de un archivo de texto, de un archivo con código ejecutable, de un directorio o de otro tipo, por ejemplo:

```
% file a.out
a.out:          pure executable
% file mbox3 netmail
mbox3:         ascii text
netmail:       commands text
```

UNIX

```
% file *
Mail:          directory
bin:           directory
c:             directory
ftp:           directory
gif:           directory
gws:           directory
mbox3:         ascii text
motif:         directory
netmail:       commands text
a.out          pure executable
phigs:         directory
redes:         directory
sherilyn:     lex command text
starb:         directory
txt:           directory
```

Para determinar los tipos, *file* no toma en cuenta los nombres de archivo, ya que las convenciones con los nombres, por el hecho mismo de no ser más que convenciones, no son confiables. En vez de eso, el comando *file* lee unos cuantos bytes al principio del archivo y busca indicios que indiquen el tipo del archivo en cuestión.

Algunas veces los indicios para identificar un archivo son obvios. Un programa ejecutable se marca con un carácter especial al principio. Un archivo con comandos de *cs* también tiene una marca especial al comienzo, etc.

Directorios

Cuando todo se almacena como un archivo, en poco tiempo, tendremos cientos de archivos con sólo una forma primitiva para organizarlos. Imaginemos la misma situación con los archivos de papel, todos los archivos juntos en un montón, o un único armario, sin ordenar excepto por el nombre dado a la carpeta.

¿Cómo resolvemos este problema con archivos normales? Una forma de empezar consiste en definir un conjunto de materias y mantener junta la información relativa a cada materia (por ejemplo, matemáticas). Seguidamente podemos crear sub-categorías dentro de

estas materias (por ejemplo, algebra). Después podríamos enlazar algunos archivos a otros incluyendo una referencia tal como "véanse también los archivos gráficos"). También se pueden incluir copias de archivos (por ejemplo, un documento puede existir en "ecuaciones" y "desarrollos"). Además necesitaremos alguna forma de cambiar fácilmente esta organización de archivos. Si varios archivos están agrupados, podremos crear una nueva categoría y copiar cosas en ella.

La estructura del sistema de archivos de Unix se construyó para poder utilizar estos principios de ordenación. La posibilidad de reunir archivos en grupos denominados directorios, nos permite, clasificar nuestros trabajos en grupos significativos y después utilizar estos grupos para organizar los archivos.

Los directorios proporcionan la forma de clasificar y establecer categorías en la información. Básicamente un directorio es un contenedor de un grupo de archivos organizados de la forma deseada. Si se imagina un archivo como algo análogo a un documento de la oficina, un directorio sería una carpeta de archivos o un cajón de la mesa. Por ejemplo, podemos decidir crear un directorio para mantener todos los archivos que contengan nuestras cartas. Un directorio denominado *cartas* podría contener este material, manteniendo estos archivos separados de aquellos que contengan ecuaciones, notas, correo y programas.

Sobre Unix, un directorio también puede contener otros directorios. Un directorio dentro de otro directorio se denomina *subdirectorio*. Podemos subdividir un directorio en tantos subdirectorios como deseemos, y cada uno de ellos puede contener tantos subdirectorios como queramos.

Tipos de archivos del sistema Unix

El archivo es la unidad básica del Sistema Unix, hay cuatro tipos diferentes de archivos: archivos ordinarios, directorios, vínculos simbólicos y archivos especiales. Además los archivos pueden tener más de un nombre, conocidos como vínculos o '*ligas*'.

Archivos ordinarios

Como usuarios, la información con la que trabajamos será almacenada como archivo ordinario. Los archivos ordinarios son agregados de caracteres tratados como una unidad por Unix. Un archivo ordinario puede contener caracteres ASCII normales tales como texto de cartas o programas. Los archivos ordinarios pueden crearse, cambiarse, o borrarse cuando queramos.

Directorios

Un directorio es un archivo que mantiene otros archivos y contiene información sobre las localizaciones y atributos de éstos. Por ejemplo, un directorio incluye una lista de todos los archivos y subdirectorios que éste contiene, así como sus direcciones, características, tipos de archivos (si son archivos ordinarios, vínculos simbólicos, directorios, o archivos especiales), y otros atributos.

Ligas

Una liga no es una clase de archivo, sino un segundo nombre para un archivo. Si dos usuarios necesitan compartir la información de un archivo, ellos pueden tener copias separadas de este archivo. Un problema al mantener copias separadas es que las dos copias pueden rápidamente perder la consistencia. Por ejemplo, un usuario puede realizar cambios que el otro podría no conocer. Una liga proporciona la solución a este problema. Con una liga, dos usuarios pueden compartir un único archivo. Ambos usuarios parecen tener copias del archivo, pero solamente existe un archivo con dos

nombres. Los cambios que cualquier usuario realiza, tienen lugar sobre la versión común. Esta liga no solamente ahorra espacio al tener una única copia de un archivo, sino que asegura que la copia que cada uno utiliza es la misma.

Ligas simbólicas

Las ligas las podemos utilizar para asignar más de un nombre a un archivo. Pero tienen algunas limitaciones importantes. No se pueden utilizar para asignar a un directorio más de un nombre. Y no se pueden utilizar para vincular nombres de archivos sobre sistemas de archivos o computadoras diferentes.

Estas limitaciones pueden eliminarse utilizando ligas simbólicas. Una liga simbólica es un archivo que sólo contiene el nombre de otro archivo. Cuando el sistema operativo opera sobre una liga simbólica, éste se dirige al archivo al que apunta la liga. Las ligas simbólicas no sólo se pueden utilizar para asignar más de un nombre a un directorio, también pueden ser utilizados por ligas que residan en sistemas de archivos físicos diferentes. Esto hace posible que un árbol de directorio lógico incluya archivos que residen sobre computadoras diferentes que están conectadas a través de una red. (Las ligas también se denominan ligas duras (hard links) para distinguirlas de las ligas simbólicas).

Archivos especiales

Los archivos especiales constituyen una característica no usual del sistema de archivos Unix. Un archivo especial representa un dispositivo físico. Puede ser un terminal, un dispositivo de comunicaciones, o una unidad de almacenamiento como un disco. Desde nuestra perspectiva de usuarios, Unix trata los archivos especiales como archivos ordinarios; esto es, puede leer o escribir los dispositivos exactamente como lee y escribe los archivos ordinarios. Se pueden tomar los caracteres pulsados en el teclado y escribirlos de la misma forma en un archivo ordinario o en una pantalla. Unix toma estas órdenes de lectura y escritura y produce

la activación del hardware conectado al dispositivo que se desee acceder.

Esta forma de tratar el hardware del sistema tiene una consecuencia importante para los usuarios de Unix. Puesto que Unix trata casi todo como si fuese un archivo, no necesitamos aprender las particularidades del hardware de la computadora. Una vez que aprendemos a manejar los archivos del sistema Unix, sabemos como manejar todos los objetos del sistema Unix. Utilizaremos la misma orden (*ls*) para ver si podemos leer o escribir en un archivo, una terminal, o un disco.

La estructura jerárquica de los archivos

Debido a que los directorios pueden contener otros directorios, que a su vez pueden contener otros directorios, el sistema de archivos del Unix se denominan sistema de archivos jerárquico. De hecho, dentro de Unix, no existe limitación del número de archivos y directorios que se pueden crear en un directorio propio. Los sistemas de archivos de este tipo se conocen como sistemas de archivos con estructura en árbol, porque cada directorio le permite bifurcar hacia otros directorios y archivos. Los sistemas de archivos con estructura de árbol se dibujan normalmente de arriba abajo, con la raíz del árbol en la parte superior del dibujo. La figura 2, muestra una posible estructura arborea bajo Unix.

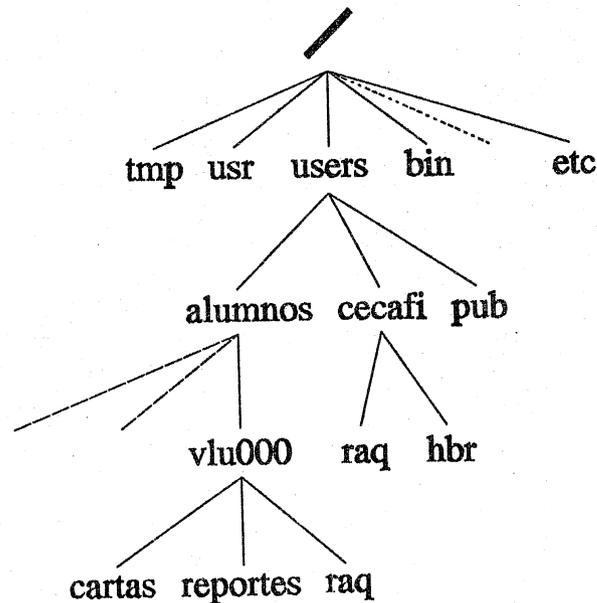
La raíz del directorio completo está en la parte superior del dibujo. Se denomina directorio raíz, o solamente raíz. La raíz se representa con un / (slash).

En la figura anterior, la raíz (/) contiene un subdirectorio llamado *users*, este a su vez contiene otro subdirectorio llamado *alumnos*. El subdirectorio *alumnos* cuenta a su vez con varios subdirectorios donde cada uno de estos, es asignado a un usuario del sistema para que dentro de estos subdirectorios, el usuario pueda guardar su información y/o generar la estructura arborea que requiera.

Cuando establecemos una sesión con la computadora, esta nos coloca automáticamente en un subdirectorio de la estructura arborea. Este subdirectorio es denominado *home directory*. Cada usuario del sistema Unix tiene un único *home directory*. El *home directory* de un usuario se representa con un símbolo que se ve con varios subdirectorios donde cada uno de estos es asignado a un usuario del sistema para que dentro de estos subdirectorios, el usuario pueda guardar su información y/o generar la estructura arborea que requiera.

Cuando establecemos una sesión con la computadora, esta nos coloca automáticamente en un subdirectorio de la estructura arborea. Este subdirectorio es denominado *home directory*. Cada usuario del sistema Unix tiene un único *home directory*. El *home directory* de un usuario se representa con un símbolo que se ve con varios subdirectorios donde cada uno de estos es asignado a un usuario del sistema para que dentro de estos subdirectorios, el usuario pueda guardar su información y/o generar la estructura arborea que requiera.

Figura 2. Posible estructura arbórea dentro del sistema de archivos Unix.



directory de cada usuario es asignado por el administrador del sistema, también conocido como el *super usuario* o *root*, debido a que el tiene todos los permisos y privilegios para poder modificar la información o configuración del sistema operativo y usuarios.

En la figura 2, dentro del subdirectorio `/users/alumnos` el usuario `vlu000` tiene asociado su home directory (`vlu000`); en este directorio hay otros dos subdirectorios (`cartas` y `reportes`) y un archivo llamado `raq`. En estos subdirectorios puede haber más subdirectorios o archivos.

Nombres de rutas

Vemos que en la figura 2 hay dos archivos con el mismo nombre, pero en posiciones diferentes (`/users/alumnos/vlu000/raq` y `/users/cecafi/raq`) en el sistema de archivos. La posibilidad de tener nombres de archivos idénticos para archivos situados en diferentes posiciones del sistema es una de las virtudes del sistema Unix. De esta forma podemos evitar nombres difíciles y artificiales en su archivo y agruparlos de manera que sea fácil de

UNIX

recordar y que tenga algún sentido. Resulta fácil reagrupar o reorganizar sus archivos creando nuevas categorías y subdirectorios.

Con el mismo nombre para archivos diferentes (raq), ¿cómo se diferencia uno de otro?. Unix permite especificar nombres de archivos incluyendo los nombres de los directorios en los que están ubicados. Este tipo de nombre de archivo se denomina 'ruta' o *path*, porque se trata de una lista del camino a lo largo del árbol de directorios que hay que seguir para llegar al archivo. Por convenio el sistema de archivos comienza en la raíz (/), y los nombres de directorios y archivos dentro del path se separan por diagonales. Por ejemplo, el path para uno de los archivos raq es

```
/users/alumnos/vlu/raq
```

y el nombre del archivo para el otro es

```
/users/cecafi/raq
```

Los path's que van desde la raíz hasta un archivo se denominan rutas absolutas. La especificación de un nombre de camino completo constituye una forma ambigua de mandar un archivo. Esto puede resultar molesto en algunos sistemas. Puesto que los archivos y directorios se utilizan para establecer categorías y organizar la información, es frecuente tener muchos niveles de directorios en un path. Es común tener de cinco a diez niveles. Un path absoluto en un sistema de este tipo podría ser

```
/users/cecafi/raq/redes/rfc/internet/seccb
```

La utilización de esta ruta absoluta requiere una buena memoria y un buen rato de tecleado. En una ruta absoluta la primer diagonal (/) hace referencia a la raíz del sistema de archivos, las siguientes diagonales separan los nombres de directorios, y la última separa el nombre del archivo del nombre del directorio. Más adelante veremos la forma de utilizar variables del medio ambiente para especificar rutas o path's de una manera más fácil y corta.

Nombres de rutas relativas

No siempre hay que especificar las rutas absolutas para hacer referencia a los archivos. Como simplificación conveniente, también podemos especificar el camino de un archivo en forma relativa al directorio sobre el cual estamos trabajando, conocido comúnmente como el '*directorio actual de trabajo*'. Tales rutas se denominan rutas relativas. Por ejemplo, si nos encontramos trabajando en el directorio `/users`, los dos archivos denominados `raq` tienen como rutas relativas `alumnos/vlu000/raq` y `cecafi/raq`.

Especificación del directorio actual de trabajo

Un único punto (`.`) se utiliza para hacer referencia al directorio actual de trabajo. La ruta `./vlu000/raq` es la ruta del archivo `raq` que se encuentra en el subdirectorio `vlu000` mientras nosotros nos encontramos trabajando en el subdirectorio `/users`.

Especificación del directorio padre

Dos puntos (`..`) se utilizan para hacer referencia al directorio padre del actual. El directorio padre es el siguiente de más alto nivel en el árbol de directorios. Para el directorio `cartas` del ejemplo, el directorio padre es `vlu000`. Para el directorio `vlu000`, el padre es `/users`. Puesto que el sistema de archivos es jerárquico, todos los directorios tienen un directorio padre excepto el raíz, el cual es el directorio a partir de donde se genera la estructura arbórea en Unix. El punto-punto se puede utilizar muchas veces para hacer referencia a cosas situadas por arriba en el sistema de archivos.

UNIX

El siguiente, por ejemplo,

```
../..
```

hace referencia al padre del padre del directorio actual. Si usted se encuentra en `/users/alumnos/vlu000/cartas`, entonces `../..` es lo mismo que el directorio `/users`, ya que `/users` es el padre del subdirectorio `vlu000`, el cual es el padre del subdirectorio `cartas`.

La notación `../..` se puede utilizar de forma repetida. Por ejemplo,

```
../../../../..
```

hace referencia al padre del padre del padre del directorio actual. Si se encuentra en el directorio `cartas`, los primeros `..` hacen referencia a su padre `vlu000`; los segundos `..` hacen referencia a su padre `alumnos`, los terceros hacen referencia a `/users`; y los cuartos `..` a su padre `/`, la raíz.

Puesto que muchas órdenes de Unix (tales como `cat` y `ls`, tratadas más adelante) operan sobre el directorio actual de trabajo, resultará útil conocerlo. La orden `pwd` (print working directory) nos dice cuál es el directorio actual de trabajo. Por ejemplo.

```
% pwd
/users/cecafi/raq
%
```

nos dice que nos encontramos colocados en el subdirectorio `users/cecafi/raq`.

El sistema de archivos

Nos puede ser útil saber en todo momento en que subdirectorio nos encontramos trabajando, para esto, podemos incluir esta línea dentro de nuestro archivo `.login` o `.cshrc`:

```
alias cd 'cd \!*; set prompt="$cwd> "'
```

la cual define el prompt como el directorio actual de trabajo cada que nos cambiemos de subdirectorio con el comando `cd`, por ejemplo, suponiendo que ésta línea ya ha sido añadida a cualquiera de los dos archivos antes mencionados:

<code>login: raq</code>	El usuario raq entra a sesión
<code>passwd:</code>	
<code>...</code>	
<code>% pwd</code>	Desplegar el directorio actual de trabajo
<code>/users/cecafi/raq</code>	
<code>% cd redes</code>	Cambiarse al subdirectorio redes
<code>/users/cecafi/raq/redes> cd ..</code>	Cambiarse al directorio padre
<code>/users/cecafi/raq></code>	De nuevo en /users/cecafi/raq



III. Trabajando con archivos y directorios

Hasta ahora hemos visto lo que son los archivos y directorios, pero no tenemos ningún procedimiento para examinar su contenido. Dos de las tareas más básicas que habrá que hacer, son visualizar el contenido de un archivo y visualizar el contenido de un directorio para ver qué contienen. Unix ofrece varios programas de utilidad que permiten hacer esto, y con frecuencia se trata de los programas más utilizados en Unix.

Listado del contenido de un directorio

El comando `ls` lista los nombres de los archivos de un directorio, si no se especifica el o los directorios de los que se desea obtener la información, toma por omisión el directorio de trabajo. Por ejemplo, si el directorio de trabajo es `/users/cecafi/raq`, `ls` mostrará los archivos que se encuentran en él:

```
% ls
Mail bcklog12-04-93:15:59 bin c ftp gif gws lsl mbox3
motif netmail phigs redes sherilyn starb txt
```

El comando `ls` puede desplegar información de uno o más directorios:

```
% ls /bin /dev
/bin:
adb
awk
...

/dev:
MAKEDEV
console
...
%
```

UNIX

Observemos que `ls` sin argumentos simplemente lista los nombres; no dice si los nombres hacen referencia a archivos o directorios. Si emitimos la orden `ls` con un argumento que es el nombre de un subdirectorio del directorio actual, `ls` nos proporciona una lista del contenido de ese directorio, por ejemplo:

```
% ls txt
art2ckfi.wp
artic002.wp
comments_rfc1034
fibras_opticas.wp
hosts.log
listusr
msgbat
music
netmsg
readme
relación
reporte
work_station
%
```

Si el archivo o directorio no existe, `ls` da un mensaje de error, tal como:

```
% ls vicky

vicky not found
%
```

Podemos ver si en nuestro directorio actual de trabajo existe un archivo con un nombre específico, para esto, suministramos el nombre como argumento del comando `ls`:

```
% ls netmail
netmail
%
```

Trabajando con archivos y directorios

El comando `ls`, al igual que la mayoría de los comandos, tiene opciones que pueden ser usadas para alterar su comportamiento normal. Dichas opciones se especifican después del comando con un signo menos - seguido de una letra que hace referencia a su significado. En la mayoría de los comandos las opciones se pueden combinar. Por ejemplo:

```
% ls -l                               Despliega un listado de archivos en forma 'larga'
total 16
drwx----- 2 raq  cecafi      24 Feb  8 18:22 Mail
drwx----- 2 raq  cecafi     1024 Mar 31 10:55 bin
drwx----- 2 raq  cecafi     1024 Mar 11 18:40 c
drwx----- 4 raq  cecafi     4096 Mar 26 13:10 ftp
...
Se despliegan más archivos y subdirectorios

-rwx----- 1 raq  cecafi      137 Apr 16 15:36 netmail
drwx----- 3 raq  cecafi     1024 Mar  9 11:19 phigs
drwxr-xr-x  2 raq  cecafi     1024 Apr 15 13:20 redes
-rwx----- 1 raq  cecafi    78758 Dec 11 16:08 sherilyn
drwx----- 2 raq  cecafi     1024 Mar 12 21:01 starb
drwx----- 2 raq  cecafi     1024 Apr 15 15:20 txt
```

La opción `-l` (long) del comando `ls` nos da mayor información sobre los archivos y directorios. El primer carácter nos dice si se trata de un archivo ordinario (`-`), directorio (`d`), liga (`l`) o archivo de dispositivo (`c` ó `b`). Luego se muestran los permisos para el archivo, el siguiente campo especifica el número de ligas del archivo, después aparecen los campos que indican el dueño y grupo al cual pertenece el archivo, tamaño, fecha de última modificación y nombre del archivo. El significado de cada uno de los campos que se muestran con esta opción se indica en la siguiente figura.

```
      -rwxr-xr-x  1 raq  cecafi    137 Apr  16 15:36 netmail
      |_____|_____|_____|_____|_____|_____|_____|_____|
      |_____|_____|_____|_____|_____|_____|_____|_____|
tipo de archivo |
permisos       |
número de ligas|
dueño         |
grupo        |
tamaño en bytes|
fecha de última modificación |
nombre       |
```

Figura 3. Información de archivos y directorios con la opción `l` del comando `ls`

UNIX

Con la opción `-a` (all) se muestran archivos de propósito especial cuyo nombre normalmente comienza con `.` (punto). Los archivos con nombres que comienzan con un punto están ocultos en el sentido de que normalmente no se visualizan cuando se listan los archivos de un directorio. Supongamos que al listar los archivos de nuestro directorio actual de trabajo vemos lo siguiente:

```
% ls
Mail      bin      c        ftp      netmail  phigs    redes
sherilyn starb    txt
```

El ejemplo anterior nos muestra que nuestro directorio actual de trabajo contiene los archivos denominados *Mail*, *bin*, *c*, *ftp*, *netmail*, *phigs*, *redes*, *sherilyn*, *starb* y *txt*. Pero puede que éstos no sean todos los archivos de este directorio. Puede haber archivos ocultos que no aparecen en este listado.

Ejemplos comunes de archivos ocultos son: *.cshrc* el cual fija el entorno de trabajo, los archivos *.newstime*, que indica la hora de las últimas noticias leídas y el archivo *.mailrc*, que lo utiliza el comando de correo electrónico *mail*. Estos archivos los utiliza regularmente el sistema, pero nosotros sólo los editaremos o leeremos de vez en cuando. Para evitar confusión, *ls* supone que no queremos que se listen los archivos ocultos a menos que explícitamente lo solicitemos. Es decir, *ls* no visualiza ningún nombre de archivo que comience con un `.` (punto).

Para ver todos los archivos de este directorio, utilice `ls -la` (la *l* es opcional):

```
% ls -la
total 24
drwxr--r-- 16 raq   cecafi   3072 Apr 19 16:04 .
drwxr-xr-x 142 root   root     3072 Mar 23 16:05 ..
-rw-r--r-- 1 raq   cecafi   291 Feb 18 19:21 .Autost
-rw-r--r-- 1 raq   cecafi   819 Mar 31 15:46 .cshrc
drwx----- 2 raq   cecafi  1024 Mar 24 13:18 .elm
-rwxr-xr-x 1 raq   cecafi   253 Apr 16 15:50 .history
-rw-r--r-- 1 raq   cecafi   389 Mar 30 20:45 .login
-r----- 1 raq   cecafi    41 Mar  2 10:20 .netrc
-r----- 1 raq   cecafi   110 Mar  4 10:33 .rhosts
drwxr-xr-x 20 raq   cecafi  1024 Feb 25 21:12 .vue
drwx----- 2 raq   cecafi    24 Feb  8 18:22 Mail
drwx----- 2 raq   cecafi  1024 Mar 31 10:55 bin
drwx----- 2 raq   cecafi  1024 Mar 11 18:40 c
drwx----- 4 raq   cecafi  4096 Mar 26 13:10 ftp
-rwx----- 1 raq   cecafi   137 Apr 16 15:36 netmail
drwx----- 3 raq   cecafi  1024 Mar  9 11:19 phigs
```

Trabajando con archivos y directorios

```
drwxr-xr-x   2 raq   cecafi   1024 Apr 15 13:20 redes
-rwx-----   1 raq   cecafi   78758 Dec 11 16:08 sherilyn
drwx-----   2 raq   cecafi   1024 Mar 12 21:01 starb
drwx-----   2 raq   cecafi   1024 Apr 15 15:20 txt
%
```

El ejemplo muestra ocho archivos ocultos. Además muestra el directorio actual y su directorio padre como . (punto) y .. (punto-punto), respectivamente.

Se puede hacer que ls ordene temporalmente los archivos con la opción **-t** (time). `ls -t` imprime los nombres de los archivos ordenados según el momento de su creación o de la última modificación. Con esta opción, los archivos que se modificaron más recientemente se listan en primer lugar. Esta forma de listado facilita la búsqueda del archivo en el que se trabajó por última vez. Esto es particularmente valioso en un gran directorio o en uno que contenga muchos archivos con nombres similares.

Si deseamos, podemos invertir una ordenación utilizando la opción **-r** (reverse). Por sí mismo, `ls -r` lista los archivos en orden alfabético inverso. Combinada con la opción **-t** lista en primer lugar los archivos más antiguos, y en último lugar los más actuales.

Examen de archivos

La forma mas simple y mas básica de inspeccionar un archivo es con la orden `cat` (de concatenado). Este comando toma cualquier archivo que le especifiquemos y lo visualiza sobre la pantalla. Por ejemplo, para visualizar sobre la pantalla el contenido del archivo `/etc/passwd`, utilizamos el comando `cat` como a continuación se muestra. Este archivo (`/etc/passwd`) contiene un listado de todas las cuentas válidas dentro del sistema así como información adicional para cada cuenta como su password, el nombre del usuario, etc.:

```
% cat /etc/passwd           Desplegar el contenido del archivo /etc/passwd
root:bUImryUSt.:0:3:Administrador del sistema,CECAFI,,:/bin/csh
daemon:*:1:5:::/bin/sh
bin:*:2:2::/bin:/bin/sh
adm:OWW0VVJKv9EKw:4:4::/usr/adm:/bin/sh
uucp:*:5:3::/usr/spool/uucpublic:/usr/lib/uucp/uucico
```

UNIX

```
lp:*:9:7::/usr/spool/lp:/bin/sh
hpdb:*:27:1:ALLBASE:/:/bin/sh
aec000:pYF3g2:677:20:alicia e. c.,,,:/users/alumnos/aec000:/bin/csh
afg000:*:374:20:adrian farias gonzalez,,,:/users/alumnos/afg000:/bin/csh
afg001:*:373:20:agustin farias gonzalez,,,:/users/alumnos/afg001:/bin/csh
aos000:nMU0Q2h.ifPTM:628:20:alfredo osorio ers/alumnos/aos000:/bin/csh
raq:rTvlU94SUp0:754:21:ricardo alvarez quiroz,,,:/users/cecafi/raq:/bin/csh
...
%
```

Puesto que Unix trata un terminal de la misma forma que trata un archivo, podemos enviar la salida de cat a un archivo de la misma forma que a la pantalla. Por ejemplo,

```
% cat netmail > copia
```

copia el contenido del archivo netmail al archivo copia. El carácter '>' proporciona una forma general de redireccionar la salida estándar (por lo general definida como el monitor) para enviar la salida de una orden a un archivo.

En el ejemplo anterior si no existe un archivo denominado copia en el directorio actual, el sistema crea uno. Si ya existe un archivo con este nombre, la salida de cat sobre escribirá ese archivo, esto es, su contenido original es sustituido.

Recordemos: en Unix los archivos incluyen a los dispositivos, como en el caso de un terminal. Para la mayor parte de las órdenes, incluyendo cat, la pantalla es el dispositivo por defecto adonde se envía la salida. Para comprobar esto, podemos hacer el siguiente ejercicio, el comando tty nos proporciona el nombre del dispositivo asociado a nuestra terminal. Una vez que sepamos cual es este archivo, utilizaremos el comando cat para redireccionar algún archivo al archivo asociado a nuestra terminal, lo que hará que el archivo sea desplegado finalmente en nuestra terminal:

```
% tty
/dev/ttypl
% cat /etc/passwd > /dev/ttypl
root:bUImryUSt.:0:3:Administrador del sistema,CECAFI,,:/:/bin/csh
daemon:*:1:5:::/bin/sh
bin:*:2:2::/bin:/bin/sh
adm:OWW0VVJKv9EKw:4:4::/usr/adm:/bin/sh
uucp:*:5:3::/usr/spool/uucppublic:/usr/lib/uucp/uucico
lp:*:9:7::/usr/spool/lp:/bin/sh
hpdb:*:27:1:ALLBASE:/:/bin/sh
aec000:pYF3g2:677:20:alicia e. c.,,,:/users/alumnos/aec000:/bin/csh
afg000:*:374:20:adrian farias gonzalez,,,:/users/alumnos/afg000:/bin/csh
afg001:*:373:20:agustin farias gonzalez,,,:/users/alumnos/afg001:/bin/csh
aos000:nMU0Q2h.ifPTM:628:20:alfredo osorio ers/alumnos/aos000:/bin/csh
raq:rTvlU94SUp0:754:21:ricardo alvarez quiroz,,,:/users/cecafi/raq:/bin/csh
```

Trabajando con archivos y directorios

En ocasiones lo que queremos es añadir la información al final de un archivo, entonces, haremos lo siguiente:

```
% cat mbox3 >> mbox
```

El conjunto de caracteres '>>' en el ejemplo anterior añade el contenido del archivo denominado *mbox3* al final del archivo denominado *mbox*, sin hacer ningún otro cambio a *mbox*. En este caso si *mbox* no existe; el sistema lo creará. Esta posibilidad de añadir la salida a un archivo existente es otra forma de redirección de archivos. Como redirección simple, ésta opera con casi todas las órdenes, no solo con *cat*.

Combinación de archivos utilizando *cat*

Podemos utilizar *cat* para combinar en uno diferentes archivos. Por ejemplo, consideremos un directorio que contiene el material que se está utilizando en la escritura de un capítulo:

```
% ls
seccion.1      macros      seccion2
capitulo.1    nombres    seccion3
capitulo.2    seccion1   sed-info
```

Se puede combinar todas las secciones de un capítulo con *cat*:

```
% cat seccion1 seccion2 seccion3 > capitulo_3
```

Este copia cada uno de los archivos *seccion1*, *seccion2* y *seccion3*, en este orden sobre el nuevo archivo *capitulo_3*. Esto se puede describir como una concatenación de los archivos en uno, de aquí el nombre *cat*.

UNIX

Para casos como éste, Unix proporciona un símbolo comodín '*' que facilita la especificación de un número de archivos con nombres similares. En el siguiente ejemplo la orden

```
% cat seccion* > capitulo_3
```

tendría el mismo efecto que la orden del ejemplo anterior. Concatena todos los archivos del directorio actual cuyos nombres comienzan con *seccion* en *capitulo_3*.

El símbolo comodín sustituye a cualquier cadena de caracteres. Cuando se utiliza como parte de un nombre de archivo en una orden, este patrón es sustituido por los nombres de todos los archivos del directorio actual que emparejan con él, listados en orden alfabético. En el ejemplo anterior, *seccion** empareja con *seccion1*, *seccion2* y *seccion3*, y así lo haría también *se**, pero *se** también coincidiría con el archivo *sed-info*.

Cuando utilizamos comodines, debemos asegurarnos de que el patrón comodín empareja con los archivos que deseamos. Es una buena idea utilizar primero el comando *ls* para probarlo. Por ejemplo,

```
% ls secc*
seccion1   seccion2   seccion3
```

pone de manifiesto que resulta seguro utilizar *secc** en la orden.

También puede utilizar *** para simplificar la escritura de órdenes, aun cuando no se esté utilizando para que empareje más que con un archivo. La orden

```
% cat *1 *2 > temp
```

es mucho más fácil que:

```
% cat seccion1 seccion2 > temp
```

Creación de un archivo con cat

Todos los ejemplos que hemos visto con cat utilizaban esta orden para copiar uno o más archivos normales en otro archivo, o en la pantalla (la salida por defecto). Pero el concepto de archivo en el Sistema Unix es muy general, y existen otras posibilidades. De la misma forma que la pantalla es la salida por defecto para cat y otras órdenes, el teclado es la entrada por defecto. Si no especificamos ningún archivo, cat simplemente copiará todo lo que se teclee a la salida. Esto nos proporciona una forma para crear archivos simples sin utilizar un editor. La orden

```
% cat > memo
```

```
La junta con el director esta programada el día 25 de  
este mes. Llevar reporte de avance.
```

```
<CTRL><d>
```

```
%
```

envía todo lo que se teclea al archivo memo. Se envía una línea cada vez que se pulsa RETURN. Se puede utilizar BACKSPACE para corregir lo que se ha tecleado sobre la línea actual, pero no se pueden corregir líneas anteriores. Cuando hemos terminado de escribir, tecleamos <CTRL><d>⁵ (las teclas Control y d en forma simultánea). Esto termina la orden cat y cierra el archivo memo.

La utilización de esta forma de cat (cat > memo) crea el archivo memo si no existe y sobrescribe sobre su contenido (reemplaza) si ya existe. También podemos utilizar cat para añadir texto al final de un archivo. Por ejemplo,

```
% cat >> memo
```

```
...
```

```
^d
```

```
%
```

⁵ CTRL-d es la marca fin de archivo (EOF) de Unix. De aquí en adelante se utilizará el símbolo '^' para representar la tecla <CTRL>.

recogerá todo lo que escribamos desde el teclado y lo añadirá al final del archivo memo. De nuevo se necesita terminar pulsando ^d.

Cambio de directorios

Nos podemos mover entre directorios utilizando la orden `cd` (change directory). Si actualmente estamos en el directorio de trabajo, `/users/cecafi/raq`, y deseamos cambiarnos al directorio Mail, tecleamos lo siguiente:

```
% cd Mail
```

Si conocemos dónde está cierta información en un Sistema Unix, podemos movernos directamente ahí especificando el path absoluto o relativo de ese directorio:

```
% cd /users/pub
% pwd
/users/pub
% ls
archiees ftp-list interest-groups
...
% cd ../cecafi/raq
% pwd
/users/cecafi/raq
% cd ../../..
% pwd
/
%
```

Si nos encontramos en cualquier parte del sistema de archivos y deseamos cambiarnos a nuestro home directory, utilizamos el comando `cd` sin parámetros.

```
% pwd
/usr/bin/X11
% cd
/users/cecafi/raq
```

Nos encontramos en otra parte de la estructura
Nos cambiamos a nuestro home directory
Estamos ya en nuestro home directory

Con frecuencia las personas que utilizan el Sistema Unix se imaginan a sí mismas situadas (lógicamente) en un lugar del sistema

de archivos. Como subrayamos en el ejemplo, en cualquier instante dado estamos en un directorio, y cuando utilizamos `cd`, nos movemos a otro directorio. Como ocurre con muchas otras utilidades, el nombre de la orden `cd`, se utiliza como un verbo por muchos usuarios. Los usuarios primero hablan de "cambiar el directorio a la raíz", después dicen "haz un `cd` a la raíz", y después simplemente utilizan la ordena como un verbo, "`cd` a la raíz". Un síntoma de lo bien que está entendiendo el Sistema Unix es su habilidad para pensar en la posición en un sistema de archivos en términos espaciales y entender el uso de las órdenes y utilidades como verbos en una sentencia.

En este capítulo vimos los conceptos fundamentales del Sistema Unix relativos a los archivos y directorios. Aprendimos a listar los archivos en un directorio utilizando la orden `ls`. También aprendimos a ver el contenido de un archivo utilizando la orden `cat`, así como a movernos entre la estructura arbórea del sistema de archivos con la orden `cd`.

Aprendimos a utilizar la orden `cat` para crear archivos. Sin embargo, se trata de una forma primitiva de crear un archivo. En el capítulo VI veremos el editor de texto `vi` que se utiliza para crear y modificar archivos.

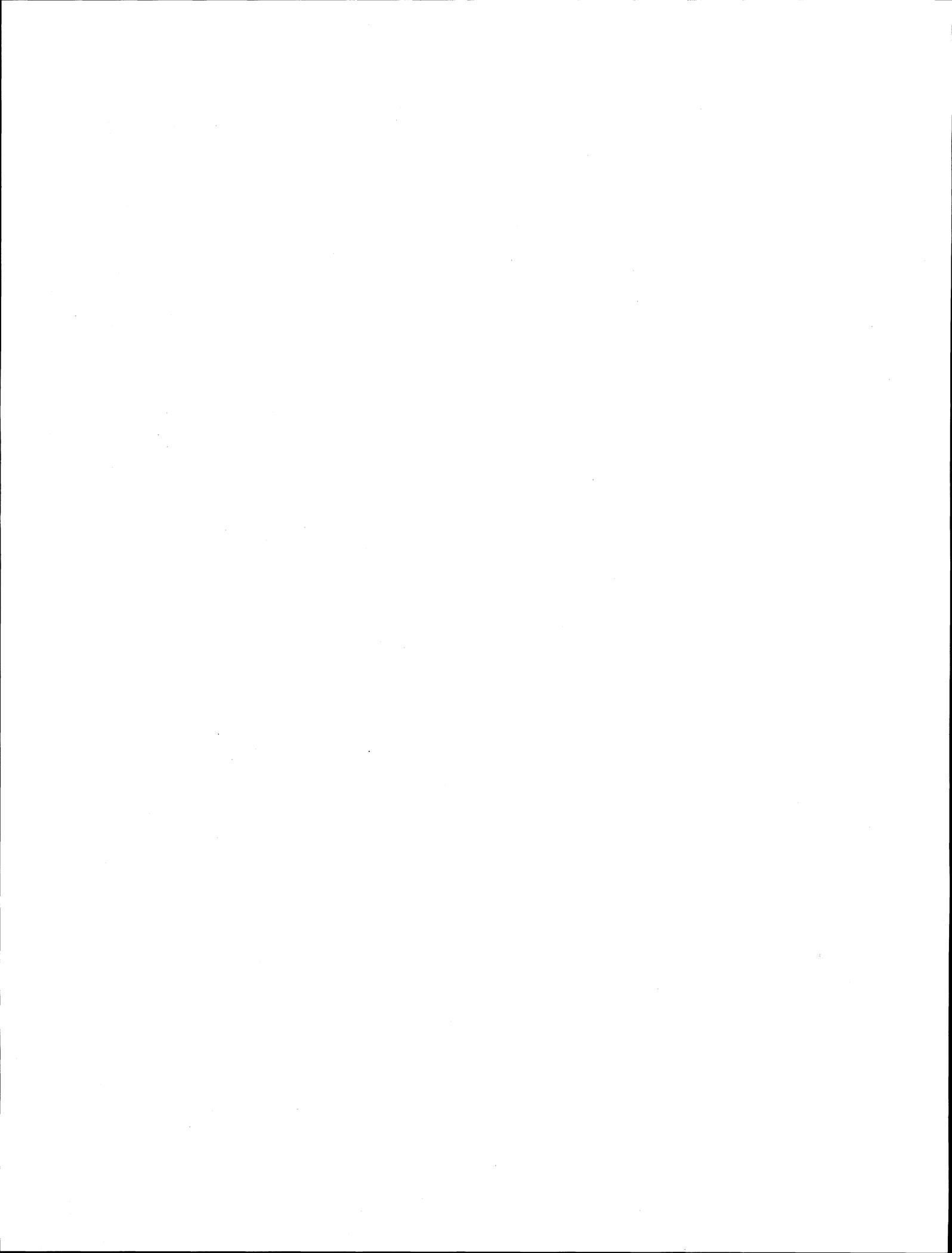
Como averiguar más

Lo que sigue es una lista de buenas referencias.

Christian, Kaarew. *The Unix Operating System, Second Edition*. New York: Wiley, 1988.

Groff, James R., and Paul N. Weinberg. *Understanding Unix, A Conceptual Guide, Second Edition*. Carmel, IN: Que Corporation. 1988.

Morgan, Rachel, and Henry McGilton, *Introducción al Unix Sistema V*. McGraw-Hill, 1988.



IV. Operaciones con archivos y directorios

El capítulo III introdujo los conceptos básicos del sistema de archivos de Unix y describió alguna de las órdenes que se utilizan para operar con archivos y directorios. Este capítulo nos ayudará a crear, modificar y manipular los archivos y directorios.

En este capítulo comenzaremos introduciendo órdenes suplementarias para operar con archivos y directorios. En particular aprenderemos órdenes para copiar, renombrar, mover y borrar archivos y crear y eliminar directorios. También aprenderemos a utilizar las opciones de la orden `ls` para obtener información sobre los archivos y sus contenidos y para controlar el formato de la salida producida por `ls`.

Aprenderemos las autorizaciones de archivos que se utilizan para restringir quién puede utilizar los archivos. Existen autorizaciones para lectura, escritura y ejecución de archivos. Pueden fijarse de manera independiente para el propietario del archivo, un grupo de usuarios y todos los demás. Veremos lo que significan estas autorizaciones, además de averiguar qué autorizaciones tiene un archivo y cómo cambiarlas.

Este capítulo también introduce un conjunto de órdenes que se pueden utilizar como herramientas para operar con archivos, incluyendo paginadores para visualizar archivos sobre la terminal y órdenes para encontrar archivos, obtener información sobre su contenido y para imprimirlos.

Manipulación de archivos

Unix proporciona una forma de establecer categorías y organizar los archivos sobre los que se trabaja. Se puede alterar el sistema de archivos añadiendo o eliminando nuevos archivos y directorios y moviendo archivos de un directorio a otro. Las órdenes que proporcionan las operaciones básicas de manipulación de archivos:

borrado, renombrado, y movimiento de archivos, están entre las más utilizadas. Esta sección trata estas órdenes básicas de manipulación de archivos del Sistema Unix.

Movimiento y renombrado de archivos

Para mantener organizado el sistema de archivos se necesita mover y renombrar archivos. Por ejemplo, se puede utilizar un directorio para borradores de documentos y moverlos desde éste hasta un directorio final cuando estén acabados. Se puede renombrar un archivo para darle un nombre que resulta más informativo o más fácil de recordar, o que refleja los cambios en su contenido o estado. Un archivo se mueve desde un directorio hasta otro con el comando *mv*. Por ejemplo, la siguiente orden mueve el archivo nombres desde el directorio actual (/users/cecafi/raq) hasta el subdirectorio /users/cecafi/raq/people

```
% pwd
/users/cecafi/raq
% mv nombres people
%
```

Si utiliza *ls* para comprobarlo, verá que existe un archivo con ese nombre en el subdirectorio *people*.

```
% ls people/nombres
nombres
```

Es posible mover varios archivos simultáneamente hacia el mismo directorio destino, nombrando en primer lugar todos los archivos a mover y dando al final el nombre del destino. Por ejemplo, la siguiente orden mueve tres archivos al subdirectorio denominado *Capítulo1*.

```
% mv seccion1 seccion2 seccion3 Capítulo1
```

Naturalmente podemos hacer esto más fácil utilizando el símbolo comodín, * (asterisco). Como se explicó en el Capítulo III, un asterisco por sí mismo empareja con todos los nombres de archivos del directorio actual, y si se utiliza con otros caracteres en una palabra, empareja con cualquier cadena de caracteres. Por ejemplo, el patrón *.a empareja con los nombres de

Operaciones con archivos y directorios

los archivos del directorio actual que finalizan en .a, incluyendo nombres tales como **temp.a**, **Libro.a** y **123.a**. Por tanto, si los únicos archivos que tienen nombres que comienzan con sección son los del ejemplo anterior, la siguiente orden tiene el mismo efecto.

```
% mv sec* Ca*1
```

Lo que acabamos de hacer es un ejemplo de cómo se pueden utilizar los comodines para simplificar la especificación de nombres de archivos en los comandos.

No existe una orden independiente en el sistema Unix para renombrar un archivo. El renombramiento de un archivo es parte de su movimiento. Se puede renombrar un archivo cuando se le mueve a un nuevo directorio incluyendo el nuevo nombre de archivo como parte del destino. Por ejemplo, la siguiente orden coloca el archivo *notas* en el directorio *Capitulo3* y le da el nuevo nombre de *seccion4*.

```
% mv notas Capitulo3/seccion4
```

Compare esto con lo siguiente, que mueve *notas* a *Capítulo3* pero mantiene el antiguo nombre, *notas*:

```
% mv notas Capitulo3
```

Para renombrar un archivo en el directorio actual también se utiliza *mv*, pero con el nombre de archivo nuevo como destino. Por ejemplo, lo siguiente renombra el archivo *repasso* a *intro*:

```
% mv repaso intro
```

Para resumir, cuando utilizamos *mv*, se nombra en primer lugar el archivo a mover, después el destino. El destino puede ser un directorio y un nombre de archivo, un nombre de directorio solo, o un nombre de archivo solo. El nombre de un directorio destino puede ser una ruta absoluta o relativa al directorio actual, por ejemplo, uno de sus subdirectorios. Si el destino no es el nombre de un directorio, *mv* lo utiliza como nuevo nombre para el archivo del directorio actual. El movimiento de archivos es muy rápido en Unix. El contenido actual de un archivo no se mueve, sólo se modifica una entrada en una tabla que dice al sistema en qué directorio se encuentran los datos. Por lo tanto, el tamaño del archivo no influye en el tiempo de la orden *mv*.

Prevención de errores con mv

Cuando utilizamos mv, debemos tener cuidado de no cometer una serie de errores comunes. Por ejemplo, cuando movemos un archivo a un nuevo directorio, es bueno comprobar primero que no existe ya un archivo con ese nombre. Si existe, mv sobrescribirá en él el nuevo archivo. La opción `-i` (interactivo) hace que mv le informe cuando en un movimiento existe la posibilidad de sobrescribir un archivo existente. Se visualiza el nombre del archivo seguido por una interrogación. Si queremos confirmar el movimiento, presionamos la tecla `y`. Cualquier otra entrada (incluyendo `n`) aborta el movimiento. El ejemplo siguiente nos muestra lo que sucede si trata de utilizar mv `-i` para renombrar el archivo `totales` a `datos` cuando el archivo `datos` ya existe.

```
% mv -i totales datos
mv: overwrite datos?
```

Creación de un directorio

Se puede crear nuevos directorios en el sistema de archivos con la orden `mkdir` (make directory). EL siguiente ejemplo muestra como utilizarla:

```
% pwd
/users/cecafi/raq
% ls
bob compra fred
% mkdir Nuevo
% ls
bob compra fred Nuevo
```

En el ejemplo anterior nos encontramos trabajando en el directorio `/users/cecafi/raq`, que contiene los archivos `bob`, `compra` y `fred`, utilizamos el comando `mkdir` para crear un nuevo directorio (llamado `Nuevo`) dentro de `/users/cecafi/raq`.

Movimiento de directorios

La orden `mv` se puede utilizar para mover un directorio y todos sus archivos y subdirectorios, de la misma forma que se mueve un único archivo. Por ejemplo, si el directorio `Notas` contiene nuestro trabajo actual sobre un documento, podemos moverlo a un directorio en el que mantengamos las versiones finales, en este ejemplo a el subdirectorio `Final`, como se muestra aquí:

```
% ls Final
Análisis
% mv Notas Final
% ls Final
Notas Análisis
%
```

Copia de archivos

Un motivo corriente para copiar un archivo es la seguridad, de manera que uno pueda modificar un archivo sin preocuparse de la pérdida del original. El comando `cp` es similar a `mv`, excepto que copia archivos en lugar de moverlos o renombrarlos. La sintaxis del comando `cp` es el siguiente:

```
cp archivo_a_copiar archivo_destino
```

El destino puede ser un directorio, un directorio y archivo, o un archivo en el directorio actual. El siguiente comando hace una copia de seguridad de archivo `doc` y nombra la copia como `doc.bckup`.

```
% cp doc doc.bckup
```

Después de haber utilizado el comando `cp` existen dos copias separadas del archivo en el mismo directorio. El original queda inalterado. El contenido del archivo copiado es idéntico al original.

Recordemos que si el directorio destino ya contiene un archivo denominado doc.bckup, la copia lo sobrescribirá. El parámetro `-i` (interactivo) nos protege las sobre-escrituras accidentales sobre archivos existentes cuando se utiliza `cp`, se recibirá un aviso antes de sobrescribir un archivo existente. Por ejemplo, si existe un archivo denominado data.2 en el directorio actual, `cp` nos avisa de que lo sobrescribirá y nos pregunta si queremos seguir adelante:

```
% cp -i data data.2
cp: overwrite data.2 ?
```

Para continuar y sobre-escribirlo, presionamos la tecla `y`. Cualquier otra respuesta, incluyendo `n` o `RETURN`, deja el archivo sin copiar.

Para crear una copia de un archivo con el mismo nombre que el original en otro directorio, utilizamos el nombre del directorio como destino:

```
% cp ftpcmd redes          Copia el archivo ftpcmd al subdirectorío redes
```

En los sistemas Unix `cp` no se puede utilizar para concatenar; en su lugar se utilizamos la orden `cat` y las redirecciones de salida o entrada estándar del shell.

Copia del contenido de un directorio

Hasta ahora hemos supuesto que estamos copiando un archivo ordinario sobre otro archivo. El parámetro `-r` del comando `cp`, nos permite copiar una estructura de directorio completa en otro directorio. Si `proy-dir` es un directorio, el siguiente comando copia todos los archivos y subdirectorios de `proy-dir` sobre el directorio `new-proy`:

```
% cp -r proy-dir new-proy
```

Eliminación de archivos

Algunas ocasiones, algunos de los archivos que se encuentren en nuestra cuenta ya no nos serán de utilidad por varias razones, entonces. para eliminarlos del sistema de archivos utilizamos el comando `rm`. Por ejemplo, supongamos que el archivo `msgbat` ya no nos es útil y deseamos borrarlo, entonces podemos utilizar los siguientes comandos:

```
% ls                               Verificamos que exista el archivo msgbat
listusr msgbat music netmsg readme
% rm msgbat                         Borramos el archivo msgbat
% ls                                 Vemos que ha sido eliminado
listusr music netmsg readme
%
```

Eliminación de múltiples archivos.

La orden `rm` acepta varios argumentos y toma varias opciones. Si le especificamos más de un nombre de archivo, elimina todos los archivos nombrados. El siguiente comando elimina los cuatro archivos que quedan en el directorio:

```
% rm listusr music netmsg readme
% ls
%
```

Recordemos que es posible eliminar varios archivos con nombres similares utilizando los caracteres comodines para especificarlos con un único patrón. El siguiente comando eliminará todos los archivos del directorio actual

```
% rm *
```

No haga esto a menos que esté seguro!.

La mayoría de los usuarios hemos borrado alguna vez archivos accidentalmente. Tales accidentes ocurren frecuentemente como consecuencia de errores de tecleado cuando se utiliza el comodín * para especificar patrones de archivos en rm. En el ejemplo anterior, si se pulsa accidentalmente la BARRA DEL ESPACIADO entre el asterisco y la extensión, y tecleamos:

```
% rm * .rlf
```

borraremos accidentalmente todos los archivos del directorio actual. Tal como se ha tecleado, este comando elimina todos los archivos (*) y después tratará de eliminar un archivo denominado .rlf.

Si queremos evitar el borrado accidental de archivos, podemos utilizar rm con la opción -i (interactivo). Cuando se utiliza esta opción, rm imprime el nombre de cada uno de los archivos y espera una respuesta antes de eliminarlos. Para continuar con el borrado de un archivo, tecleé y. Si responde n o pulsa RETURN, el archivo no se eliminará. Por ejemplo, en un directorio que contiene los archivos bob, fred y compra, la opción interactiva de rm nos permite lo siguiente:

```
% ls
bob fred compra
% rm -i *
bob: y          Presionamos la tecla 'y' para borrar el archivo bob.
compra: <RETURN> No deseamos borrar el archivo compra.
fred: y         También deseamos borrar el archivo fred.
% ls
compra
```

rm nos pide la disposición de cada uno de los archivos en el directorio. Nuestra respuesta hace que rm elimine los archivos bob y fred, pero no compra. Si después de esto utilizamos el comando ls, comprobaremos que sólo permanece compra.

Para borrar un subdirectorio, se utiliza el comando *rmdir* (remove directory), pero antes de hacerlo, debemos tomar en cuenta dos puntos:

- 1.- No debemos de estar colocados dentro del subdirectorio que deseamos eliminar.
- 2.- El subdirectorio a eliminar no debe de contener ningún archivo o subdirectorio.

Operaciones con archivos y directorios

si lo anterior se cumple, podemos entonces utilizar el comando `rmdir` de la siguiente forma:

```
% rmdir subdirectorio_a_eliminar
```

Por ejemplo, deseamos eliminar de la estructura arbórea el subdirectorio `tmp`, utilizamos entonces los siguientes comandos:

```
% ls -la          Vemos cuales son los archivos y subdirectorios actuales
-rw-rw----- 1 raq cecafi 2194 Apr 15 13:19 mbox3
-rw-rw-rw-   1 raq cecafi 149 Apr 19 16:26 file
-rwx-----  1 raq cecafi 137 Apr 16 15:36 netmail
-rwx-----  1 raq cecafi 78758 Dec 11 16:08 sherilyn
drwx-----  2 raq cecafi 24 Feb 8 18:22 Mail
drwx-----  2 raq cecafi 1024 Apr 15 15:20 txt
drwx-----  2 raq cecafi 1024 Mar 4 17:38 tmp
% rmdir tmp      Eliminamos el subdirectorio tmp que está vacío
% ls -la        Verificamos que haya sido removido el subdirectorio tmp
-rw-rw----- 1 raq cecafi 2194 Apr 15 13:19 mbox3
-rw-rw-rw-   1 raq cecafi 149 Apr 19 16:26 file
-rwx-----  1 raq cecafi 137 Apr 16 15:36 netmail
-rwx-----  1 raq cecafi 78758 Dec 11 16:08 sherilyn
drwx-----  2 raq cecafi 24 Feb 8 18:22 Mail
drwx-----  2 raq cecafi 1024 Apr 15 15:20 txt
```

Si el subdirectorio que deseamos remover no estuviera vacío, podemos utilizar el comando `rm` con los parámetros `-r` (recursivo) y `-f` (forzar) para la eliminación de todo archivo y subdirectorio contenido dentro de él. **ATENCIÓN:** Estos parámetros deben de usarse con MUCHO cuidado ya que de lo contrario podríamos borrar información muy valiosa y recordemos que una vez borrados los archivos, estos ya no pueden ser recuperados⁶. Supongamos que deseamos borrar el subdirectorio `ftp`, el cual a su vez contiene diversos archivos y subdirectorios usados en forma temporal mientras se transfiere software de la red Internet, entonces podríamos utilizar el siguiente comando:

```
% rm -fr ftp      Borra el subdirectorio ftp y todos los archivos
                  subdirectorios que dentro de él existan.
```

⁶ En las work Station HP es posible recuperar un archivo que ha sido borrado siempre y cuando se haya establecido una sesión gráfica a través del VUE y que el archivo se haya borrado utilizando el administrador de archivos 'file manager'. Para recuperar un archivo borrado de esta forma, se deberá de seleccionar el icono del bote de la basura colocado en la esquina inferior derecha e interactuar con este programa, para recuperar el archivo.

Es posible combinar las opciones recursiva (-r) e iterativa (-i) para ir paso a paso a través de todos los archivos y directorios, eliminándolos o dejándolos de uno en uno.

Restauración de archivos

Cuando eliminamos un archivo, éste desaparece. Si cometemos un error, nuestra mayor esperanza es que el archivo se encuentre disponible en algún otro lugar del sistema (sobre una cinta o disco). En estos casos podemos llamar al administrador del sistema y preguntar cómo restaurar el archivo. Si ha sido respaldado, se podrá restaurar. Los sistemas difieren ampliamente en la forma y la frecuencia en que realizan copias de seguridad. En algunos sistemas se realizan copias de seguridad de todos los archivos cada día y se guardan durante un número de días, semanas o meses. En otros sistemas las copias de seguridad son menos frecuentes, quizá semanales. En estaciones de trabajo personales las copias de seguridad tienen lugar cuando se deseé. En cualquier caso se perderán todos los cambios realizados desde la última copia de seguridad.

Ligas entre archivos

Cuando copiamos un archivo, estamos creando otro archivo con el mismo contenido que el original. Cada copia ocupa espacio adicional en el sistema de archivos, y cada una se puede modificar independientemente de las otras. En Unix, existen unas estructuras llamadas *i-nodo* que son generadas al momento de instalar el sistema operativo. Estas estructuras son las que almacenan la información referente a cada archivo, como son: el número de veces que un archivo es referenciado en un directorio (número de ligas), Los permisos de acceso al archivo, el tamaño del archivo, el lugar del disco en donde comienza el archivo, la fecha de la última modificación, acceso y cambio de protección del archivo. Cada que se genera un nuevo archivo en disco, se utiliza una o más de estas estructuras (dependiendo del tamaño del archivo, por lo general se utiliza un solo *i-nodo* por archivo). En el supuesto caso de que se

Operaciones con archivos y directorios

acabasen estas estructuras i-nodo y todavía hubiese espacio en disco, ya no sería posible crear un solo archivo más a menos de que se borrasen algunos ya existentes para liberar i-nodos, pues no habría forma de saber donde empezarían dichos archivos.

A veces resulta útil tener un archivo que sea accesible desde varios directorios, pero que permanezca siendo sólo uno. Esto, puede reducir la cantidad de espacio en disco utilizado para almacenar información redundante y puede facilitar el mantener la consistencia de los archivos utilizados por varias personas.

Por ejemplo, supongamos que estamos trabajando con un grupo de personas y necesitamos compartir información contenida en un único archivo de datos que cada uno de nosotros puede actualizar. Cada uno de nosotros necesitamos tener acceso fácil al archivo, y deseamos que las adiciones o cambios que realicemos sobre el archivo estén disponibles inmediatamente al otro. Un caso donde puede ocurrir esto es en una lista de nombres y direcciones que dos ó mas de dos personas utilizan y que cualquiera de los usuarios puede ampliar ó editar en caso que la información en ésta cambie. Cada usuario necesita acceder a una versión común del archivo en un lugar conveniente.

El comando `ln` crea ligas (conocidas comúnmente como ligas duras) entre archivos de un directorio, lo que permite hacer accesible un único archivo a dos o más posiciones del sistema de archivos. El formato para hacer una liga entre dos archivos es el siguiente:

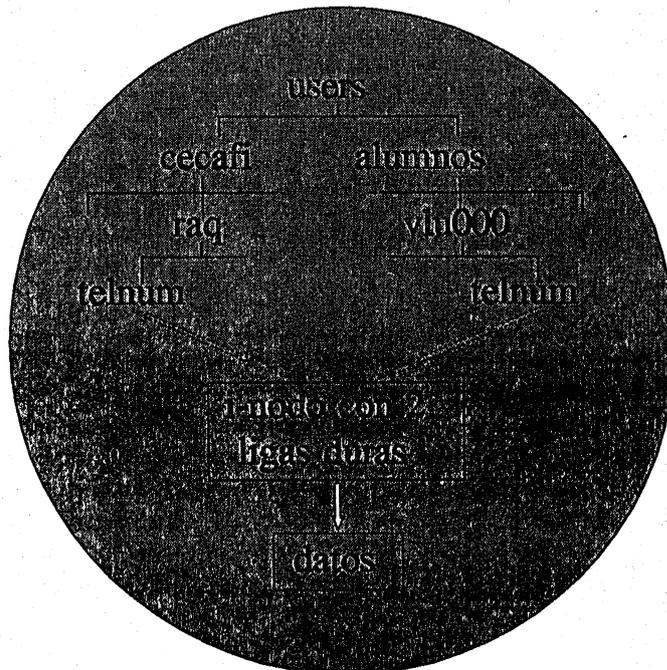
```
ln archivo_existente liga_al_archivo_existente
```

El ejemplo siguiente, liga al archivo `telnum` en el directorio `/users/cecafi/raq` con un archivo del mismo nombre en el subdirectorio `/users/alumnos/vlu000`.

```
% pwd
/users/cecafi/raq
% ln telnum /users/cecafi/vlu000/telnum
```

La utilización de `ln` para crear un vínculo (o liga), origina una segunda entrada de directorio, pero hay realmente un único archivo. Ahora si se añade una nueva línea de información al archivo `telnum` en el directorio `/users/cecafi/raq`, esto repercute en el archivo vinculado en el directorio de `vlu000`.

Cualquier cambio en el contenido del archivo vinculado afecta a todos los vínculos. Si sobrescribimos la información del archivo, la información de la liga en el subdirectorio `/users/cecafi/vlu000`



también se sobrescribe. Para una descripción de la forma de evitar estropear archivos, vea la opción noclobber de C shell (csh) y de Korn shell (ksh), que se describen en la ayuda en línea con los comandos:

```
% man csh
% man ksh
```

Con la orden `rm` podemos eliminar los archivos vinculados sin afectar a los otros. Por ejemplo, si se elimina la copia vinculada de `telnum`, la copia de `vlu000` queda inalterada. Cuando el número de ligas para un archivo llega a cero, el archivo es removido del sistema de archivos.

Al crear ligas con el comando `ln` sin parámetros, tengamos en cuenta lo siguiente:

- Las ligas deben de hacer referencia a un archivo, no a un subdirectorio.
- Comparten el mismo i-nodo.

Operaciones con archivos y directorios

- Deben de hacerse entre archivos del mismo sistema de archivos.
- Son ventajosas puesto que existe una sola copia del archivo, ahorrando así espacio en disco y proporcionan a los usuarios la copia más actualizada.

Ligas simbólicas

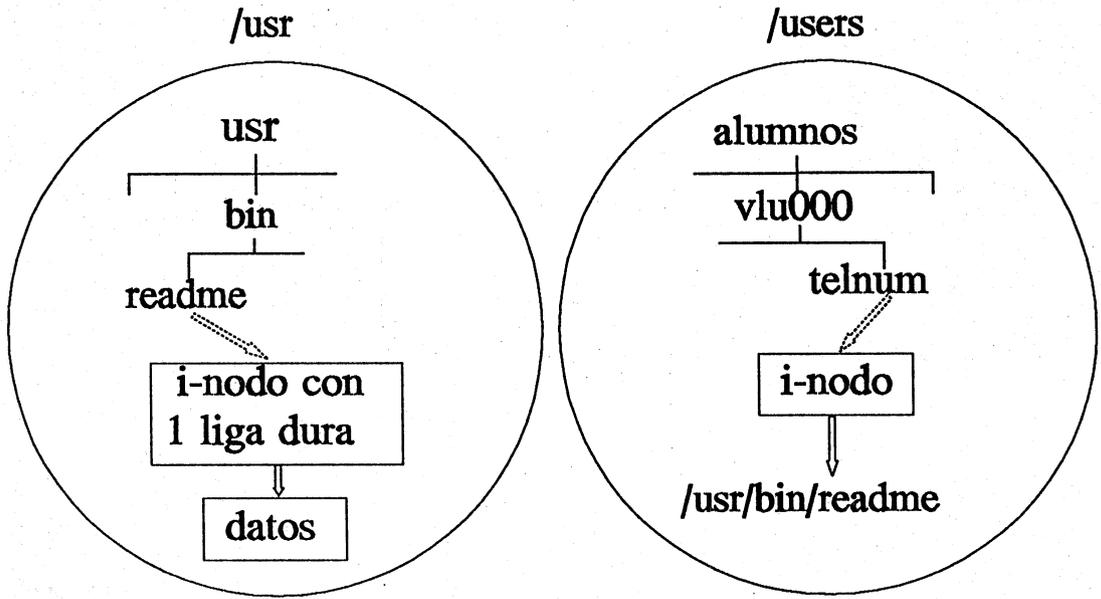
La orden `ln` sin parámetros, puede vincular archivos solamente dentro de un mismo sistema de archivos. En Unix, se pueden vincular archivos entre diferentes sistemas de archivos utilizando la opción `-s` (simbólico) del comando `ln`. El siguiente ejemplo muestra cómo podemos utilizarse esta característica para vincular un archivo del sistema de archivos `/usr` con un archivo en nuestro subdirectorio del sistema de archivos `/users`.

```
% pwd
/users/alumnos/raq
% ln -s /usr/bin/readme telnum
```

Además de permitir vínculos entre sistemas de archivos, las ligas simbólicas, permiten vincular directorios de la misma forma que archivos regulares.

Por ejemplo, el siguiente listado muestra los archivos contenidos en un subdirectorio `x`:

```
% ls -la
total 186
drwx----- 2 raq cecafi 1024 May 11 17:12 .
drwx----- 16 raq cecafi 3072 May 7 15:32 ..
-rwxr-xr-x 1 raq cecafi 4693 Mar 1 13:52 art2ckfi.wp
-rwxr-xr-x 1 raq cecafi 17321 Mar 1 13:55 artic002.wp
-rw-rw-rw- 1 raq cecafi 2928 Feb 25 20:50 comentarios_rfc1034
-rwxr-xr-x 1 raq cecafi 40639 Mar 11 18:37 fibras_opticas.wp
-rwx----- 2 raq cecafi 1342 Mar 24 15:11 listusr
-rw-r----- 1 raq cecafi 2484 Feb 15 10:25 msgbat
-rw-r--r-- 2 raq cecafi 2794 May 11 17:10 music
-rw-rw-rw- 1 raq cecafi 1582 Feb 25 20:53 netmsg
-r--r--r-- 1 raq cecafi 3489 Feb 12 11:49 readme
-rwxr-xr-x 1 raq cecafi 4024 Mar 1 14:00 reporte
```



Archivo conteniendo la ruta del archivo

Para hacer una liga dura y una liga suave sobre el archivo music damos los siguientes comandos:

```
% ln music ligal
% ln -s music liga2
```

Se crea una liga 'dura' llamada ligal.
Se crea una liga 'suave' llamada liga2.

Operaciones con archivos y directorios

Ahora utilizamos el comando `ls -lai` para ver los tamaños de los archivos que acabamos de crear (ligas). El parámetro `-i` nos permite visualizar el número del i-nodo de cada archivo:

```
% ls -lai
```

```
total 186
114702 drwx----- 2 raq cecafi 1024 May 11 17:12 .
 14336 drwx----- 16 raq cecafi 3072 May 7 15:32 ..
114776 -rwxr-xr-x 1 raq cecafi 4693 Mar 1 13:52 art2ckfi.wp
114779 -rwxr-xr-x 1 raq cecafi 17321 Mar 1 13:55 artic002.wp
114781 -rw-rw-rw- 1 raq cecafi 2928 Feb 25 20:50 comentarios_rfc1034
114782 -rwxr-xr-x 1 raq cecafi 40639 Mar 11 18:37 fibras_opticas.wp
114784 -rw-r--r-- 2 raq cecafi 2794 May 11 17:10 ligal
114773 lrwxrwxrwx 1 raq cecafi 5 May 11 17:12 liga2 -> music
 34974 -rw----- 2 raq cecafi 1342 Mar 24 15:11 listusr
114783 -rw-r----- 1 raq cecafi 2484 Feb 15 10:25 msgbat
114784 -rw-r--r-- 2 raq cecafi 2794 May 11 17:10 music
114785 -rw-rw-rw- 1 raq cecafi 1582 Feb 25 20:53 netmsg
114790 -r--r--r-- 1 raq cecafi 3489 Feb 12 11:49 readme
114791 -rwxr-xr-x 1 raq cecafi 4024 Mar 1 14:00 reporte
```

Podemos observar que existen dos archivos más llamados `ligal` y `liga2`, sin embargo, `ligal` es del mismo tamaño que el archivo `music` puesto que es una liga dura, o sea un mismo archivo referenciado bajo dos nombres diferentes, de hecho, podemos observar el número del i-nodo (primer columna) y observaremos que es el mismo i-nodo para los archivos `music` y `ligal`, mientras que el archivo `liga2`, nos indica que es un 'apuntador' o una liga suave al archivo `music`. Observemos el número del i-nodo del archivo `liga2` y veremos que es diferente al número del i-nodo del archivo `music`, esto es debido a que una liga suave, ocupa otra estructura i-nodo mientras que una liga dura ocupa el mismo i-nodo que el archivo original. Observemos además que la tercera columna (número de ligas), ahora se ha incrementado para el archivo `music` (pues este ya tiene otra referencia que es `ligal`); en el caso de `liga2` no es así debido a que es otra estructura i-nodo que de momento solo esta referenciada una sola vez.

Permisos

Existen tres clases de permisos o autorizaciones para archivos, los cuales corresponden con las tres clases de usuario: el propietario (o usuario) del archivo, el grupo al que pertenece el propietario y los otros usuarios del sistema. Al hacer uso del comando `ls` con

UNIX

el parámetro `-l`, las tres primeras letras del campo de autorización hacen referencia a las autorizaciones del propietario, las tres siguientes a los miembros del grupo del propietario y las últimas a los otros usuarios, por ejemplo:

```
% ls -l netmail
-rwxr-xr-- 1 raq cecafi 137 Apr 16 15:36 netmail
```

tipo de archivo									
permisos									
número de ligas									
dueño									
grupo									
tamaño en bytes									
fecha de última modificación									
nombre del archivo									

Información de archivos y directorios con la opción `l` del comando `ls`

En la entrada correspondiente al archivo denominado `netmail` en el ejemplo anterior, las tres primeras letras `rwx`, muestran que el propietario del archivo puede leerlo (`r`), escribirlo (`w`) y ejecutarlo (`x`).

El segundo grupo de tres caracteres, `r-x`, indican que los miembros del grupo (todos aquellos que pertenezcan al grupo `cecafi`), pueden leer y ejecutar el archivo, pero no pueden escribirlo. Los últimos tres caracteres, `r--`, muestran que todos los demás solo pueden leer el archivo, pero no escribirlo ni ejecutarlo.

Si tenemos autorización de lectura en un archivo, podemos visualizar su contenido. Autorización de escritura significa que podemos alterar su contenido. Autorización de ejecución significa que puede ejecutar el archivo como un programa.

Con respecto a directorios, la autorización de lectura permite listar el contenido del directorio, la autorización de escritura permite crear o eliminar archivos o directorios dentro de ese directorio, y la autorización de ejecución permite moverse en ese directorio con la orden `cd` y convertirlo en el directorio actual.

Existen otros códigos que se utilizan en los campos de autorización y no se ilustran en el ejemplo anterior. Por ejemplo,

la letra *s* (set user ID o set group ID) puede aparecer donde existe una *x* en el campo de autorización de usuario o de grupo. Esta *s* hace referencia a una clase especial de autorización de ejecución que es importante para programadores y administradores del sistema. Desde el punto de vista del usuario, la *s* es esencialmente lo mismo que la *x* en ese lugar. También el carácter */* puede aparecer en lugar de una *r*, *w* ó *x*. Esto significa que el archivo estará bloqueado durante el acceso, de manera que los otros usuarios no podrán acceder a él mientras se esté utilizando.

La orden *chmod*

Unix nos permite fijar las autorizaciones de cada uno de los archivos. Sólo el propietario de un archivo puede modificar las autorizaciones. Nosotros podemos manipular independientemente las autorizaciones de propietario, de grupo y otros usuarios, para permitir o impedir la lectura, escritura o ejecución para nosotros, nuestro grupo y los demás usuarios.

Para alterar la autorización de un archivo, se utiliza la orden *chmod* (change mode). Con *chmod* especificamos en primer lugar las autorizaciones que se van a cambiar: *u* para el propietario, *g* para el grupo y *o* para otros. En segundo lugar se especifica cómo se cambian: *+* (añadir autorizaciones) ó *-* (quitar autorizaciones) para leer (*r*), escribir (*w*) o ejecutar (*x*). En tercer lugar especificamos el archivo al que hacen referencia los cambios.

UNIX

En el siguiente ejemplo se solicita el listado de los archivos en nuestro directorio de trabajo en forma extendida, se cambian sus autorizaciones utilizando la orden `chmod` y se lista de nuevo para visualizar los cambios en las autorizaciones:

```
% ls -la
-rw-rw----- 1 raq cecafi      2194 Apr 15 13:19 mbox3
-rw-rw-rw-   1 raq cecafi      149 Apr 19 16:26 file
-rwx-----  1 raq cecafi      137 Apr 16 15:36 netmail
-rwx-----  1 raq cecafi     78758 Dec 11 16:08 sherilyn
-rwxr-xr-x   1 raq cecafi      253 Apr 16 15:50 .history
drwx-----  2 raq cecafi       24 Feb  8 18:22 Mail
drwx-----  2 raq cecafi     1024 Apr 15 15:20 txt
drwx-----  2 raq cecafi     1024 Mar  4 17:38 tmp
% chmod go-rw file
% ls -la
-rw-rw----- 1 raq cecafi      2194 Apr 15 13:19 mbox3
-rw-----   1 raq cecafi      149 Apr 19 16:26 file
-rwx-----  1 raq cecafi      137 Apr 16 15:36 netmail
-rwx-----  1 raq cecafi     78758 Dec 11 16:08 sherilyn
-rwxr-xr-x   1 raq cecafi      253 Apr 16 15:50 .history
drwx-----  2 raq cecafi       24 Feb  8 18:22 Mail
drwx-----  2 raq cecafi     1024 Apr 15 15:20 txt
drwx-----  2 raq cecafi     1024 Mar  4 17:38 tmp
```

El comando `chmod` en el ejemplo anterior elimina (-) las autorizaciones de lectura y escritura (`rw`) al grupo y a los demás (`go`) correspondiente al archivo `file`. Cuando utilizamos una orden como ésta, decimos "cambiamos el modo correspondiente al grupo y otros; quité las autorizaciones de lectura y escritura de archivo `file`".

También podemos añadir autorizaciones con el comando `chmod`.

```
% chgrp ugo+rwx netmail sherilyn
% ls -la
-rw-rw----- 1 raq cecafi      2194 Apr 15 13:19 mbox3
-rw-----   1 raq cecafi      149 Apr 19 16:26 file
-rwxrwxrwx   1 raq cecafi      137 Apr 16 15:36 netmail
-rwxrwxrwx   1 raq cecafi     78758 Dec 11 16:08 sherilyn
-rwxr-xr-x   1 raq cecafi      253 Apr 16 15:50 .history
drwx-----  2 raq cecafi       24 Feb  8 18:22 Mail
drwx-----  2 raq cecafi     1024 Apr 15 15:20 txt
drwx-----  2 raq cecafi     1024 Mar  4 17:38 tmp
```

El comando anterior añade (+) autorizaciones de lectura, escritura y ejecución (`rwx`) para el usuario, el grupo y otros (`ugo`) en los archivos `netmail` y `sherilyn`. Al utilizar este comando no

debemos dejar ningún espacio en blanco entre las letras de las opciones correspondientes a `chmod` pero si es posible especificar más de un archivo con el mismo modo de protección, en este caso, los archivos `netmail` y `sherilyn`.

Una característica de `chmod` es la opción `-R` (recursivo), que aplica los cambios a todos los archivos y subdirectorios de un directorio. Por ejemplo, el siguiente comando hace que todos los archivos y subdirectorios de `txt` puedan ser leídos por el propietario:

```
% chmod -R u+r txt
%
```

Fijación de las autorizaciones absolutas

La forma de la orden `chmod` descrita aquí, utilizando la notación `ugo[+/-]rwx`, la cual permite cambiar las autorizaciones en relación a su valor actual. Como propietario del archivo, podemos añadir o quitar autorizaciones según lo deseemos. Otra forma de la orden `chmod` permite fijar las autorizaciones directamente, utilizando un código numérico para especificarlas.

Este código representa la autorización de un archivo mediante tres números octales: uno para las autorizaciones del propietario, otro para las autorizaciones del grupo, y otro para los demás. Por ejemplo, el siguiente comando fija las autorizaciones de lectura, escritura y ejecución para el propietario, no permitiendo a ningún otro hacer nada con el archivo `Mail`:

```
% chmod 700 Mail
```

UNIX

Para entender mejor cómo se representan las autorizaciones con este código, veamos la siguiente tabla:

<u>Permiso de</u>	<u>Propietario</u>	<u>Grupo</u>	<u>Otros</u>
Lectura	4	0	0
Escritura	2	0	0
Ejecución	1	0	0
<hr/>			
Suma	7	0	0

La tabla anterior nos muestra cómo "700" representa las autorizaciones del subdirectorio Mail. Cada columna de la tabla hace referencia a uno de los usuarios (propietario, grupo y otros). Si un usuario tiene autorización de lectura, se suma 4; para fijar la autorización de escritura, se suma 2, y para fijar la autorización de ejecución, se suma 1. La suma de los números de cada columna es el código de autorización del usuario.

En la siguiente tabla se muestra cómo la orden

```
% chmod 750 Mail
```

fija la autorización de lectura, escritura y ejecución para el propietario, la de lectura y ejecución para el grupo no dejando ningún permiso para los demás:

<u>Permiso de</u>	<u>Propietario</u>	<u>Grupo</u>	<u>Otros</u>
Lectura	4	4	0
Escritura	2	0	0
Ejecución	1	1	0
<hr/>			
Suma	7	5	0

Operaciones con archivos y directorios

La siguiente tabla muestra cómo la orden

```
% chmod 774 Mail
```

fija las autorizaciones de lectura, escritura y ejecución para el propietario y el grupo, y sólo permite la autorización de lectura a otros:

<u>Permiso de</u>	<u>Propietario</u>	<u>Grupo</u>	<u>Otros</u>
Lectura	4	4	4
Escritura	2	2	0
Ejecución	1	1	0
<hr/>	<hr/>	<hr/>	<hr/>
Suma	7	7	4

Se puede utilizar chmod para fijar las autorizaciones relativa o absoluta de cualquier archivo del que seamos propietario. Utilizando comodines se podemos fijar las autorizaciones de conjuntos de archivos y directorios, Por ejemplo, el comando siguiente elimina las autorizaciones de lectura, escritura y ejecución de grupo y de otros en todos los archivos del directorio actual:

```
% chmod go-rwx *
```

Para fijar las autorizaciones de todos los archivos del directorio actual de manera que pueda leer, escribir y ejecutar sólo el propietario (se niega la autorización a cualquier otro miembro del grupo y otros usuarios), teclee:

```
% chmod 700 *
```

Utilización de umask para fijar autorizaciones

El comando chmod permite alterar las autorizaciones archivo por archivo. La orden umask permite hacer esto automáticamente cuando se crea cualquier archivo o directorio.

UNIX

Umask permite especificar las autorizaciones de todos los archivos que se crean después de emitir la orden umask. En lugar de tratar con autorizaciones sobre archivos individuales, se determinan con una sola orden las autorizaciones de todos los archivos futuros. La especificación de autorizaciones con umask es un poco más complicada, pero esto se facilita si recordamos dos puntos.

- umask utiliza un código numérico para representar autorizaciones absolutas de la misma forma que chmod. Por ejemplo, 777 significa autorización de lectura, escritura y ejecución para el usuario, el grupo y los otros (rwxrwxrwx).
- Se especifican las autorizaciones diciéndole a umask que reste de las autorizaciones completas (rwxrwxrwx) el valor deseado.

Por ejemplo, después de emitir la orden siguiente, todos los nuevos archivos de esta sesión tomarán las autorizaciones rwxr-xr-x:

```
% umask 022
```

Para ver cómo funciona el ejemplo anterior, tenga en cuenta que corresponde a un valor numérico de 755, y 755 es simplemente el resultado de restar la "máscara", 022 en este ejemplo, de 777.

Por ejemplo, para asegurarnos que sólo uno pueda leer, escribir o ejecutar sus archivos, podemos colocar la siguiente línea dentro de su archivo .cshrc si su intérprete de comandos es csh (C shell), o bien dentro de su archivo .profile si su intérprete de comandos es sh (Bourne o Korn shell):

```
umask 077
```

Lo anterior equivale a utilizar chmod 700 o chmod go-rwx, pero estas protecciones se aplicará a todos los archivos que creemos en la sesión actual después de emitir la orden umask.

Cambio de propietario de un archivo

Cada archivo tiene un propietario, normalmente la persona que lo creó. Cuando creamos un archivo, somos el propietario. El propietario de un archivo normalmente tiene más autorizaciones para manipular el archivo que los otros usuarios.

A veces es necesario cambiar el propietario de un archivo; por ejemplo, cuando se adquiere la responsabilidad de un archivo que previamente pertenecía a otro usuario. Incluso si otra persona nos "da" un archivo: moviéndolo a nuestro directorio, esto no hace que seamos el propietario. Una forma de convertirse en propietario de un archivo es hacer una copia, cuando hacemos una copia nos convertimos en el propietario de la copia. Por ejemplo, supongamos que copiamos un archivo que pertenece al usuario khr, desde el home directory de khr hasta nuestro directorio de trabajo:

```
% cp /users/alumnos/khr/contents contents
```

Ahora, si utilizamos `ls -l` para sacar un listado largo del original en el directorio khr y de nuestra copia, veremos que ambos tienen la misma longitud (porque su contenido es el mismo), pero el original tiene como propietario a khr, mientras que la copia nos muestra que somos es el propietario del archivo (en este caso la cuenta vlu000). Por ejemplo:

```
% ls -l /users/alumnos/khr/contents contents
-rw-r--r--  1 khr      users      1040 Jul 23 15:56 contents
-rw-r--r--  1 vlu000  users      1040 Aug 28 12:34 contents
```

La utilización de `cp` para cambiar la pertenencia de un archivo es también útil cuando ya tenemos un archivo de otro usuario (cuando está en nuestro directorio). Si khr utiliza `mv` para colocar un archivo en nuestro directorio, khr permanece como propietario. Si queremos tener el completo control sobre ese archivo, podemos hacer una copia y después borrar el original.

Esta forma de cambiar la propiedad de un archivo tiene dos desventajas. En primer lugar crea un archivo extra (la copia), cuando lo que se quiere es dar al archivo una nueva propiedad. Y mas importante, el cambio de propiedad mediante copia funciona

cuando el nuevo propietario copia el archivo del antiguo propietario, lo que requiere que el primero tenga permiso de lectura. Una forma más simple y directa de transferir la propiedad consiste en utilizar la orden *chown* (change owner).

La orden *chown* tiene dos argumentos: el nombre del nuevo propietario y el nombre del archivo. Por ejemplo, lo siguiente hace que *chr* otorgue la propiedad del archivo *contents* al usuario *vlu000*:

```
% chown vlu000 /users/alumnos/vlu000/contents
```

Sólo el propietario de un archivo (o el superusuario) puede utilizar el comando *chown* para cambiar la pertenencia de archivos.

Chmod incluye la opción *-R* (recursivo) que se puede utilizar para cambiar la propiedad de todos los archivos de un directorio. Si *Admin* es uno de nuestros subdirectorios, podemos convertir a *chr* en propietario de éste y de todos sus archivos y subdirectorios con el siguiente comando:

```
% chown -R chr Admin
```

Búsqueda de archivos

Con la orden *find* se puede explorar cualquier parte del sistema de archivos buscando todos los que tengan un nombre o alguna característica en particular. Esto es extremadamente potente, y a veces puede ser un salvavidas. En esta sección estudiaremos la forma de utilizar esta orden para búsquedas simples.

Un ejemplo de problema frecuente que *find* puede ayudar a resolver es el de localizar un archivo que se tiene mal colocado. Por ejemplo, si queremos encontrar un archivo denominado *new_data* pero no recordamos dónde lo dejamos, podemos utilizar *find* para buscar por todo o parte de nuestro directorio.

La orden *find* busca en uno o más directorios, incluyendo todos sus subdirectorios. Hay que decirle a *find* en qué directorio comenzar la búsqueda. Por ejemplo, para que busque en todos los subdirectorios, primero nos colocamos en nuestro *home directory* y después le decimos a *find* que busque desde el directorio actual de

Operaciones con archivos y directorios

trabajo hacia abajo. El siguiente ejemplo busca a partir del home directory del usuario **raq** el archivo **new_data** e imprime el nombre de la ruta absoluta de todos los archivos que encuentre con ese nombre:

```
% cd                               Nos cambiamos al home directory
% pwd
/users/cecafi/raq
% find . -name new_data -print
/users/cecafi/raq/txt/new_data      Archivo encontrado
/users/cecafi/raq/ftp/cmds/new_data  Archivo encontrado
%
```

En el ejemplo anterior existen dos archivos denominados **new_data**, uno en el subdirectorío **txt** y otro en el subdirectorío **ftp/cmds**. Este ejemplo ilustra la forma básica de la orden **find**. El primer argumento es el nombre del directorio en el que comienza la búsqueda. En este caso se trata del directorio actual de trabajo (representado por el punto). La segunda parte de la orden especifica el nombre de lo que debe buscar precedido del parámetro **-name**, y la tercera parte dice a **find** que imprima (**-print**) el nombre del camino completo de todos los archivos que encuentre.

Obsérvese que hay que incluir la opción **-print**. Si no se hace, **find** llevará a cabo la búsqueda, pero no notificará sobre ningún archivo que encuentre.

Para buscar sobre todo el sistema de archivos, podemos comenzar en el directorio raíz, representado por **/**:

```
% find / -name new_data -print
```

Esta buscará un archivo denominado **new_data** en todo el sistema de archivos. También tengamos en cuenta que se puede tardar un buen rato en finalizar una búsqueda en el sistema de archivos completo (entre 1 a 2 minutos, o más, dependiendo del tamaño del file system y la cantidad de archivos que este contenga).

Se puede decir a **find** que busque en varios directorios dándole cada uno de los directorios como argumento. El siguiente comando busca en el directorio actual y en sus subdirectorios; después busca en **/tmp/project** y sus subdirectorios:

```
% find . /tmp/project -name new_data -print
```

Con **find** se pueden utilizar símbolos comodines para buscar archivos en caso de no conocer los nombres exactos. Por ejemplo, si no estamos seguros de si el archivo que está buscando se denomina

new_data, new.data o ndata, pero sabemos que termina en data, podemos utilizar el patrón ***data** como nombre de búsqueda encerrado entre comillas. Por ejemplo:

```
% find -name "*data" -print
```

Cuando se utiliza un comodín con el argumento **-name**, hay que ponerlo entre comillas. Si no lo hacemos, el proceso de comparación del nombre de archivo puede reemplazar ***data** con los nombres de todos los archivos del directorio actual que finaliza en "data".

Ejecución de find en background

Si es necesario se puede buscar por todo el sistema diciendo a find que comience en directorio raíz (/). Find puede tardar mucho tiempo en buscar un gran directorio y sus subdirectorios, y asimismo, en el sistema completo de archivos, comenzando en /. Si necesitamos ejecutar una comando como éste que tarda mucho tiempo, podemos utilizar la característica multitarea de Unix y ejecutarlo como un trabajo asíncrono o subordinado (conocido muy frecuentemente como **background**), que nos permite continuar haciendo otras tareas al tiempo que find lleva a cabo su búsqueda.

Para ejecutar una orden en modo asíncrono o background, el comando a ejecutar se finaliza con un ampersand (&). El siguiente comando ejecuta a find en modo subordinado buscando en el sistema de archivos completo por el archivo new_date y enviando su salida al archivo 'found'.

```
% find / -name new_date -print > found &
```

La ventaja de ejecutar una orden en background es que nos es posible ejecutar otro comando sin necesidad de esperar a que finalice la tarea en background.

Observemos que en este ejemplo la salida de find fue dirigida a un archivo, en lugar de ser visualizada en la pantalla. Si no hacemos esto, la salida aparecerá en la pantalla mientras se está realizando otra cosa: por ejemplo, mientras estamos editando un documento. Esto no suele ser lo que se desea.

Otros criterios de búsqueda

Los ejemplos vistos hasta el momento han mostrado cómo utilizar `find` para buscar un archivo teniendo un nombre dado. Hay muchos otros criterios que podemos utilizar para buscar archivos. La opción `-perm` hace que `find` busque archivos que tienen un patrón particular de autorizaciones (utilizando el código de autorizaciones octal descrito anteriormente en `chmod`). La opción `-type` permite especificar el tipo de archivo a buscar. Para buscar un directorio, utilizamos `-type d`. La opción `-user` restringe la búsqueda a archivos que pertenecen a un usuario particular. La opción `-size` permite buscar archivos de determinado tamaño, la opción `-mtime` y `-atime` permiten encontrar aquellos archivos que no han sido modificados o accedidos desde determinada fecha respectivamente, etc.

Se puede combinar estas y otras opciones de `find`. Por ejemplo, el siguiente comando dice a `find` que busque un nombre de archivo que comienza en `garden`, perteneciente al usuario `vlu000`.

```
% find . -name "garden*" -user vlu000 -print
```

Para encontrar aquellos archivos que no han sido accedidos o modificados en los últimos 90 días, podemos utilizar el siguiente comando:

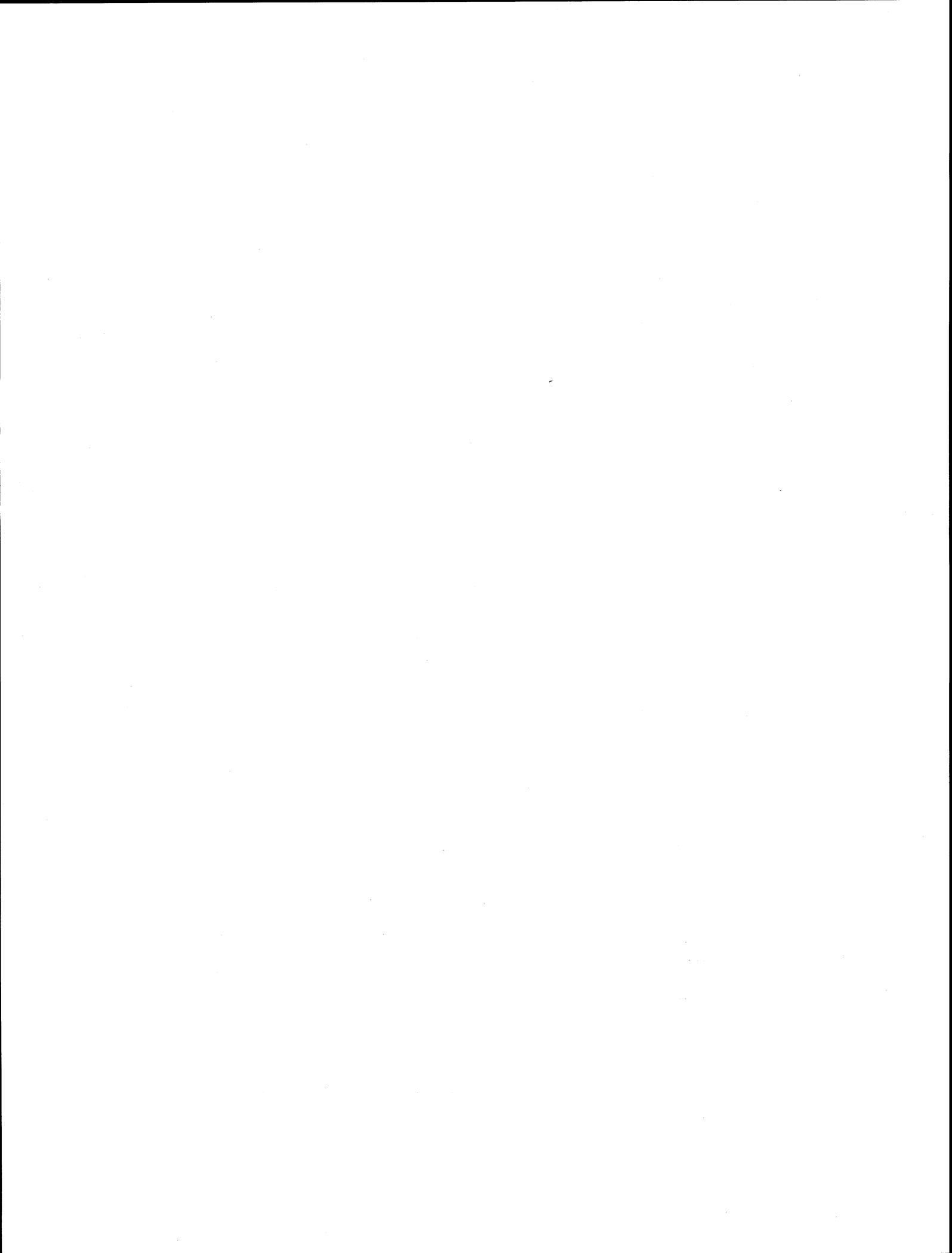
```
% find . -atime +90 -mtime +90 -print
```

Para encontrar aquellos archivos de tamaño mayor a 500 Kb y el resultado dejarlo en el archivo `big_files`, utilizamos el comando:

```
% find . -size +1000 -print > big_files
```

Pero el comando `find` puede hacer más que imprimir el nombre del archivo que encuentra. Por ejemplo, podemos decirle que ejecute una orden en cada archivo que empareje con el patrón de búsqueda. Para este y otros usos avanzados de `find`, hay que consultar las páginas del manual de `find` en la ayuda en línea con el comando:

```
% man find.
```



V. Filtros e interconexión de comandos

Una de las características principales de Unix es su enfoque de programación. Dentro de este enfoque tenemos lo que se conoce como *interconexiones*, que permiten dirigir la salida de un comando como entrada de otro, con lo cual podemos obtener por medio de comandos aparentemente sencillos resultados que en otros sistemas se realizan con comandos muy especializados.

Algunos de los comandos que ayudan a realizar esta interconexión, son los llamados *filtros*, que son comandos que leen alguna entrada, realizan una serie de transformaciones sobre esta (la filtran) y generan una salida sin modificar la entrada original de datos. En este capítulo se discutirán los filtros más utilizados, así como también la forma de utilizarlos en interconexiones.

Redireccionamiento de entrada/salida

La mayoría de los comandos y utilerías de Unix toman el flujo de datos de entrada, por omisión, de la *entrada estándar* (definida como el teclado) y producen una salida de datos hacia la *salida estándar* (definida como el monitor); sin embargo, tanto la entrada como la salida estándar se pueden cambiar, de manera que los comandos puedan tomar datos de entrada de un archivo y su salida se pueda escribir a un archivo o sea tomada como la entrada de otro comando.

Cuando se modifica la salida estándar de datos se dice que se está haciendo un *redireccionamiento de salida*. El redireccionamiento puede ser dirigido hacia un archivo con ayuda del operador `>`. El ejemplo siguiente, muestra como se realiza el redireccionamiento de salida:

```
% ls -la > listado
```

UNIX

El comando `ls -la`, normalmente va a desplegar un listado completo de todos los archivos que se encuentran en el directorio de trabajo; cuando se utiliza el operador de redireccionamiento, la salida que produce el comando se va a almacenar en el archivo denominado listado. En la pantalla de la terminal no aparecerá nada, ya que el flujo de datos de salida se redireccionó hacia el archivo listado. Si después de esto deseamos ver el contenido del archivo listado, podemos utilizar el comando `cat`:

```
% cat listado
```

```
-rw-rw---- 1 raq cecafi      2194 Apr 15 13:19 mbox3
-rw----- 1 raq cecafi       149 Apr 19 16:26 file
-rwxrwxrwx 1 raq cecafi       137 Apr 16 15:36 netmail
-rwxrwxrwx 1 raq cecafi    78758 Dec 11 16:08 sherilyn
-rwxr-xr-x 1 raq cecafi       253 Apr 16 15:50 .history
drwx----- 2 raq cecafi        24 Feb  8 18:22 Mail
drwx----- 2 raq cecafi     1024 Apr 15 15:20 txt
drwx----- 2 raq cecafi     1024 Mar  4 17:38 tmp
-rw-rw-rw- 1 raq cecafi       840 Apr 10 14:30 tarea1
-rw-rw-rw- 1 raq cecafi       480 Apr 11 18:52 tarea2
-rw-rw-rw- 1 raq cecafi       260 Apr 12 17:27 tarea3
```

El archivo será creado en caso de que no existiera, si ya existe, su contenido será reemplazado. Gracias a esta característica, es posible con ayuda del comando `cat` almacenar el contenido de varios archivos en uno sólo como se explicó en el capítulo anterior usando los operadores de redireccionamiento `>` y `>>`.

El *redireccionamiento de entrada* se da cuando se sustituye la entrada estándar de un comando, haciendo que esta sea tomada de otro lugar, por ejemplo un archivo, o de la salida de otro comando. El redireccionamiento de entrada utiliza el operador `<`. Por ejemplo, para redireccionar la entrada del comando `cat`:

```
% cat < tarea1.c
/*  Tarea No. 1
   Programa que escribe el letrero: "Hola mundo"
*/
#include <stdio.h>

main() {
    printf("Hola mundo\n");
}
%
```

El comando anterior toma su entrada del archivo `tareas` y despliega en la salida estándar, en este caso el monitor, su

contenido. Supongamos que tenemos un programa llamado *nómina* al cual se le proporcionan como entrada una lista de empleados y genera como salida un reporte en pantalla. Se podrían hacer una serie de manipulaciones para que la lista de empleados se proporcionará por medio de un archivo y el reporte quedará almacenado en otro, de esta forma podíamos escribir comandos como el siguiente:

```
% nomina < empleados > reporte  
%
```

El archivo *empleados* contiene una lista de empleados y en el archivo *reporte* se almacenará el reporte que normalmente se produce en pantalla al ejecutar el programa *nómina*.

Filtros

Un filtro en Unix es un programa o comando que recibe un flujo de datos de entrada, realiza una transformación, manipulación o selección de ella (filtra precisamente la información de entrada) y genera una salida de datos sin alterar la entrada original.

Se puede utilizar el comando *cat* para visualizar archivos completos. Pero con frecuencia lo que realmente se necesita es ver las primeras líneas o las últimas de un archivo. Por ejemplo, si un archivo de base de datos se actualiza periódicamente con nueva información de contabilidad, se puede necesitar ver si se han realizado las actualizaciones más recientes. Otros ejemplos son la comprobación de las últimas líneas de un archivo para ver si ha sido ordenado, y la lectura de las primeras líneas de cada uno de los archivos para ver cual cuál contiene la versión más reciente de un apunte. Los comandos *tail* y *head* están especialmente diseñados para este tipo de trabajos.

UNIX

Por ejemplo, el comando `tail` es un filtro que despliega las últimas diez líneas del flujo de datos de entrada sin modificar esta. El siguiente comando, visualizará las últimas 10 líneas del archivo llamado `transactions`:

```
% tail transactions
# 1 /users/pub/story
# 1 /users/pub/tareas.ed
# 1 /users/pub/ti.n
# 1 /users/pub/tlock
# 1 /users/pub/trapl.ksh
# 1 /users/pub/ul.n
# 1 /users/pub/uno
# 1 /users/pub/vshld102.zip
# 1 /users/pub/waiter
# 1 /users/pub/xx
```

El comando `tail` también permite le indiquemos cuantas líneas deseamos desplegar. Así, para desplegar las tres últimas líneas, se proporciona a `tail` un argumento numérico `-3`:

```
% tail -3 transactions
# 1 /users/pub/vshld102.zip
# 1 /users/pub/waiter
# 1 /users/pub/xx
%
```

para desplegar las últimas 20 líneas del archivo `tareas`, deberémos ejecutar el siguiente comando:

```
% tail -20 tareas
```

También podemos utilizar `tail` para desplegar archivo a partir de una línea específica, sustituyendo el signo `(-)` por un `(+)`. El comando:

```
% tail +10 tareas
```

desplegará el archivo `tareas` a partir de la línea 10.

Filtros e interconexión de comandos

El comando `head` al contrario que `tail`, muestra las primeras 10 líneas de un archivo. Por ejemplo, la siguiente orden visualiza las diez primeras líneas del archivo `transactions`:

```
% head transactions
# 1 /
# 1 /users
# 1 /users/pub
# 1 /users/pub/2
# 1 /users/pub/ADD
# 1 /users/pub/AYUDA
# 1 /users/pub/DELETE
# 1 /users/pub/DELETE.ksh
# 1 /users/pub/HOSTNAME.LIS
# 1 /users/pub/HOSTS.TXT
%
```

También podemos indicar cuantas líneas se desean desplegar, el comando:

```
% head -16 /etc/passwd          Desplegar las primeras 16 líneas.
root:GLPI/Ygll4uD.:0:3:Administrador del sistema,CECAFI,,:/bin/csh
ceo:*:0:3:./bin/ksh
daemon*:1:5:./bin/sh
bin*:2:2:./bin:/bin/sh
adm:rKfKRrC4raarc:4:4:./usr/adm:/bin/sh
uucp*:5:3:./usr/spool/uucppublic:/usr/lib/uucp/uucico
lp*:9:7:./usr/spool/lp:/bin/sh
hpdb*:27:1:ALLBASE:./bin/sh
raq:OwQEp3w:574:22:Ricardo Alvarez Quiroz,6228100:/users/cecafi/raq:/bin/csh
edwin:gDQhTAP2:567:22:Edwin Navarro Pliego,6228100,./users/cecafi/edwin:/bin/csh
amezcua:3O13ytUiw:578:22:luis a. amezcua a.,6228100:/users/cecafi/amezcua:/bin/sh
gaby:.N4bXzHhMYhak:277:22:Gabriela Magallanes,6228100,./users/cecafi/gaby:/bin/sh
hbr:PoXl/G.tGQkS.:279:22:Hector Badillo Rojas,6228100,,./users/cecafi/hbr:/bin/sh
nam*:200:22:norberto arrieta,,./users/cecafi/nam:/bin/csh
samuel:4GOpj52uTneNI:205:20,,,,./users/cecafi/samuel:/bin/csh
rna000*:644:20:raquel negrete aranda,,./users/alumnos/rna000:/bin/csh
```

despliega las primeras 16 líneas del archivo `/etc/passwd`.

Otro filtro interesante es `wc` (word counter), el cuál nos permite hacer un conteo de las líneas, palabras y caracteres de un flujo de datos proporcionado como entrada. El siguiente ejemplo muestra el funcionamiento de `wc`:

```
% wc tareal.c nomina
10   20   130 tareal.c
5    40   153 nomina
15   60   283 total
```

UNIX

Esta salida nos indica que el archivo `tareal.c`, contiene 10 líneas, 20 palabras y 130 caracteres, el archivo `nomina` contiene 5 líneas, 40 palabras y 153 caracteres; estos en total son 15 líneas, 60 palabras y 283 caracteres.

Por omisión, `wc` cuenta el número de líneas, palabras y caracteres en el flujo de datos de entrada; sin embargo, se pueden especificar las siguientes opciones, o una combinación de ellas para afectar la forma en la que el comando `wc` mostrará su información de salida:

```
l   cuenta el total de líneas
w   cuenta el total del palabras
c   cuenta el total de caracteres
```

por ejemplo, el comando:

```
% wc -lw tareal.c
10  20 tareal.c
```

da como resultado solamente el número de líneas y palabras en el archivo `tareal.c`.

Otro filtro muy útil es el comando `sort`, el cual nos permite clasificar registros de un flujo de datos de entrada. La clasificación se hace tomando como base el orden lexicográfico de cada uno de los caracteres en un registro; sin embargo, se puede llevar a cabo el ordenamiento tomando un campo específico como llave de ordenamiento. Los registros de entrada son vistos como un conjunto de campos separados por espacios en blanco o tabuladores. Para los siguientes ejemplos, desplegaremos primero el contenido de los archivos `nombres` y `frutas`, posteriormente manipularemos la información de estos archivos utilizando el comando `sort`.

```
% cat nombres
Badillo Rojas Hector
Lopez Uresti Vicky
Alvarez Quiroz Ricardo
Navarro Pliego Edwin
Magallanes Gonzalez Gabriela
Barrera Hernandez Beatriz
Meza Gil Gerardo
% cat frutas
manzana          4500
naranja          1500
pera             6000
melón            2000
sandía           900
```

Filtros e interconexión de comandos

guayaba	1200
fresa	5000
mango	1100
%	

Ahora, supongamos que se desea ordenar el archivo nombres tomando como llave el primer apellido; para este caso el comando a ejecutar es:

```
% sort nombres
Alvarez Quiroz Ricardo
Badillo Rojas Héctor
Barrera Hernández Beatriz
López Uresti Vicky
Magallanes González Gabriela
Meza Gil Gerardo
Navarro Pliego Edwin
%
```

Ahora, si deseásemos ordenar este archivo tomando como llave el nombre de la persona, daríamos el siguiente comando:

```
% sort +2 nombres
Barrera Hernández Beatriz
Navarro Pliego Edwin
Magallanes González Gabriela
Meza Gil Gerardo
Badillo Rojas Héctor
Alvarez Quiroz Ricardo
López Uresti Vicky
%
```

El parámetro +2 especifica el campo que se tomará como llave del ordenamiento; para el comando sort, el primer campo se indica como +0.

Algunos otros parámetros para el comando sort se muestran a continuación:

PARAMETRO FUNCION

f	Considera letras mayúsculas y minúsculas por igual.
n	Toma el campo a clasificar como un valor numérico.
r	El ordenamiento se realiza en orden inverso.
t	Especifica un separador de campo.
u	Elimina registros repetidos.
m	Ordena por el método de la mezcla archivos previamente clasificados.

Tabla 5.1. Parámetros del comando sort.

El siguiente ejemplo:

```
% sort +1 frutas
mango          1100
guayaba        1200
naranja        1500
melón          2000
manzana        4500
fresa          5000
pera           6000
sandia         900
%
```

no produce el resultado deseado, puesto que el ordenamiento se hace tomando el orden lexicográfico de los caracteres que componen un campo, de tal forma, que el segundo campo es tomado simplemente como una secuencia de caracteres y no como un valor numérico. Para que el resultado sea correcto debemos especificar un ordenamiento numérico tomando como llave el segundo campo:

```
% sort +1n frutas
sandia         900
mango          1100
guayaba        1200
naranja        1500
melón          2000
manzana        4500
fresa          5000
pera           6000
%
```

Filtros e interconexión de comandos

Si nosotros quisiéramos ordenar el archivo /etc/passwd, utilizando el comando sort, tomando como llave el tercer campo el cual es la identificación de cada usuario dentro del sistema, o uid por sus siglas en inglés, y además, deseamos ordenar este campo en forma descendente para ver primero los uids mas grandes y al último los más pequeños, entonces, podríamos pensar que el comando sería el siguiente:

```
% sort +2nr /etc/passwd
adm:rKfKRRc4raarc:4:4::/usr/adm:/bin/sh
amezcua:3R5Ff0l:578:22:luis alfonso amezcua a.,,,:/users/cecafi/amezcua:/bin/sh
bin:*:2:2::/bin:/bin/sh
ceo:*:0:3:::/bin/ksh
daemon:*:1:5:::/bin/sh
edwin:geAp2:567:22:Edwin Navarro Pliego,6228100,:/users/cecafi/edwin:/bin/csh
gaby:.N4bXzhak:277:22:Gabriela Magallanes,6228100,:/users/cecafi/gaby:/bin/sh
hbr:PoXl/G.tGQks.:279:22:Hector Badillo Rojas,6228100,:/users/cecafi/hbr:/bin/csh
hpdb:*:27:1:ALLBASE:/bin/sh
lp:*:9:7::/usr/spool/lp:/bin/sh
nam:*:200:22:norberto arrieta m, soporte,20950,:/users/cecafi/nam:/bin/csh
raq:OyoWwlp3w:574:22:Ricardo Alvarez Quiroz,6228100:/users/cecafi/raq:/bin/csh
rna000:*:644:20:raquel negrete aranda,,,:/users/alumnos/rna000:/bin/csh
root:GLPI/Ygil4uD.:0:3:Administrador del sistema,CECAFI,,,:/bin/csh
samuel:4GOpj52uTneNI:205:20:,,,:/users/cecafi/samuel:/bin/csh
uucp:*:5:3::/usr/spool/uucpublic:/usr/lib/uucp/uucico
```

...

Lo cual es incorrecto, ya que sort ordeno nuevamente por el campo número 0, el nombre de la cuenta, esto sucede debido a que sort espera que el separador de campos por omisión sea el espacio en blanco o el tabulador, y como aquí los campos están separados por dos puntos (:), sort ordena el único campo que ve, el cual es todo el registro.

Para ordenar correctamente este archivo en base al tercer campo, hay que especificarle al comando sort cual va a ser el separador de campos del archivo, en este caso hacemos uso del parámetro `-t`: como sigue:

```
% sort +2nrt: /etc/passwd
oca000:fedFmz6UU:701:20:ochoa cruz ana luisa,,,:/users/alumnos/oca000:/bin/csh
rcr001:*:689:20:ricardo g. cota robles,,,:/users/alumnos/rcr001:/bin/csh
mtg000:*:688:20:mayra tovar garcia,,,:/users/alumnos/mtg000:/bin/csh
hcz000:*:687:20:hector m. cabrera zu&iga,,,:/users/alumnos/hcz000:/bin/csh
afh000:*:685:20:ana lucia franco hdez.,,,:/users/alumnos/afh000:/bin/csh
lrh000:*:682:20:leopoldo ruiz huerta,,,:/users/alumnos/lrh000:/bin/csh
poo000:*:681:20:patricia ortiz ortiz,,,:/users/alumnos/poo000:/bin/csh
```

...

%

Interconexión de comandos

Al utilizar redireccionamiento de entrada/salida vimos que se pueden crear archivos que contengan la salida de un comando; de esta forma, se podría tomar dicho archivo como entrada a otro comando. Por ejemplo, supóngase que se desea obtener en pantalla el desplegado de los archivos que existen en el directorio /bin ordenados por su tamaño en forma ascendente. Recordemos que con la opción -l del comando ls, el quinto campo que se despliega indica el tamaño en bytes, (en Ultrix es el cuarto campo, ya que el comando ls -l no despliega el grupo al cual pertenece el archivo. Para esto hay que especificar el parámetro -g). Se puede, con los siguientes comandos, obtener el resultado para el enunciado anterior:

```
% ls -la /bin > temporal
% sort +5n temporal
```

Sin embargo, el archivo creado no tiene ninguna utilidad después de que se ha obtenido el resultado deseado. De esta forma si realizamos algunas otras aplicaciones de este tipo, tendríamos que estar creando archivos temporales, los cuales resultarían innecesarios una vez que se hubiésemos obtenido el resultado esperado. Unix nos da la facilidad de ahorrar la creación de los archivos temporales, redireccionando o conectando la salida de un programa como entrada a otro mediante una interconexión | (o 'pipe' [tubería]); de esta forma, para obtener el resultado anterior bastaría con ejecutar el siguiente comando:

```
% ls -la /bin | sort +5n
```

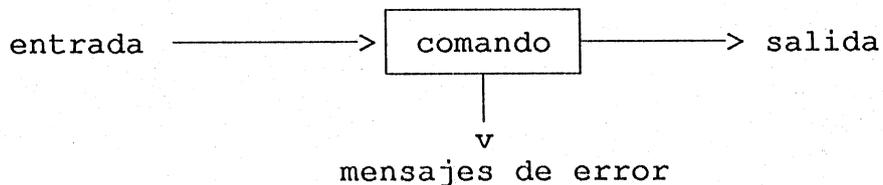
Todos los programas que reciben como entrada un flujo de datos ya sea de la entrada estándar o de un archivo, lo pueden hacer de otro programa a través de una interconexión y la salida de ellos puede pasar como entrada a otro comando o redireccionarse hacia un archivo. Este tipo de programas deben operar correctamente y generalmente, el esquema para invocarlos es el siguiente:

```
comando [argumentos] [lista de archivos]
```

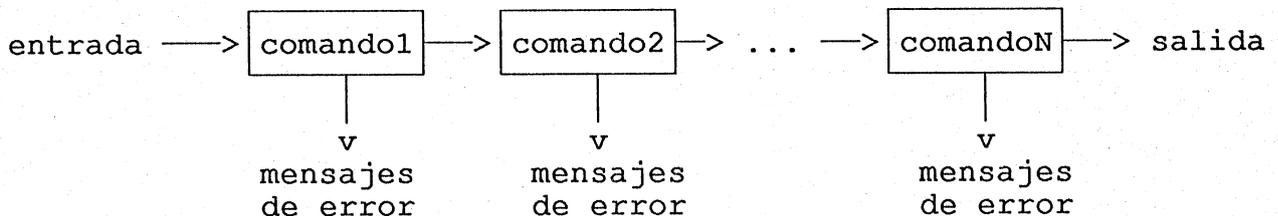
Filtros e interconexión de comandos

donde los corchetes indican que se trata de parámetros opcionales. De esta forma, el flujo de datos de entrada para un comando puede provenir de un conjunto de archivos, de la entrada estándar o de una interconexión; la salida es generada, por omisión, hacia la salida estándar, para que de esta forma se pueda redireccionar hacia un archivo o hacia una interconexión.

Los mensajes de error de los comandos no se mezclan con la salida normal, sino que son enviados a la *salida de errores* (definida como el monitor); de esta forma, los mensajes de error no se pierden al utilizar interconexiones. El siguiente diagrama ilustra el diseño de este tipo de comando:



Cuando interconectamos comandos, la única salida que se puede manipular, ya sea para almacenarse en un archivo o para desplegarla en pantalla, es la del último comando interconectado; esto se muestra en el siguiente diagrama:



De esta forma podemos formar comandos muy potentes simplemente interconectando las salidas y entradas de comandos mucho más sencillos, por ejemplo, el siguiente comando:

```
% ps -ef | grep raq | sort +ln | more
```

nos mostrará todos los procesos (*ps -ef*) del usuario *raq* (*grep raq*) ordenados por el segundo campo (*sort +ln*), el cual es la identificación de cada proceso dentro del sistema, y en caso de que el número de procesos que tenga ejecutándose el usuario *raq* sea grande (más de 24 líneas), la salida será mostrada pantalla por pantalla gracias al comando *more*. Para más información sobre el

comando ps, consulte la ayuda en línea de este comando con la orden:

```
% man ps
```

el comando grep es cubierto dentro del siguiente tema de este capítulo.

Expresiones regulares con grep y egrep

Otro filtro de gran utilidad es `grep` (global regular expression printer), el cual busca la ocurrencia de un patrón en el flujo de datos de entrada y despliega las líneas donde encuentra dicho patrón. Por ejemplo, para saber si el usuario con clave `curso101` se encuentra en sesión, se podría teclear el siguiente comando:

```
% who | grep curso101
curso101  tty01    Apr 10 16:40
%
```

El patrón que se especifica en `grep`, se conoce como expresión regular. Una expresión regular es un patrón que representa o describe a un conjunto de caracteres. En el ejemplo anterior, la expresión regular era sencilla; sin embargo, las expresiones regulares se especifican utilizando caracteres que tienen significado especial dentro de la misma expresión regular. A estos caracteres se le denomina metacaracteres, por ejemplo: `^` indica inicio de línea mientras que `$` indica fin de línea.

Cuando se utilizan metacaracteres en la expresión regular, esta debe encerrarse entre apóstrofes. Por ejemplo, para listar todos los directorios del directorio de trabajo, podemos utilizar el siguiente comando:

```
% ls -la | grep '^d'
```

Recordemos que el comando `ls` con la opción `-l`, proporciona varios campos de información para los archivos y que el primer carácter del primer campo indica el tipo de archivo del que se trata (`d` para directorios, `-` para archivos ordinarios, `l` para ligas, etc.).

Filtros e interconexión de comandos

El metacarácter . (punto) se utiliza para representar a un carácter cualquiera, * indica cero o más repeticiones del carácter anterior (más adelante veremos que se utiliza también para representar cero o más ocurrencias de una cadena), de tal forma que la expresión regular `a.*b` representa a cadenas que comienzan con el carácter `a` seguidas de una secuencia de caracteres (no importando cuales, incluyendo la cadena vacía) y terminan con `b`.

Hemos mencionado que el carácter . representa a cualquier carácter y que para representar a uno en especial basta con representarlo tal como es en la expresión. Si queremos especificar un subconjunto de caracteres especificamos este rango entre corchetes, por ejemplo, `[a-z]` representa a cualquier letra minúscula, y `[a-zA-Z]` a cualquier letra mayúscula o minúscula.

Si deseamos obtener una lista ordenada, del archivo `nombres`, de las personas cuyo primer apellido comience con las tres primeras letras del alfabeto; la lista deberá estar ordenada por apellido paterno. En este caso, el comando adecuado sería el siguiente:

```
% grep '^[abcABC]' nombres | sort
Alvarez Quiroz Ricardo
Badillo Rojas Héctor
Barrera Hernández Beatriz
%
```

El comando `grep` acepta las opciones que se muestran a continuación o una combinación de ellas:

- c Produce como salida el número de líneas en donde fue encontrado el patrón.
- f Especifica un archivo del cual se lee el patrón.
- n Genera como salida la línea en donde se encontró la ocurrencia del patrón, precedida por su número de línea.
- s Suprime los mensajes de salida, excepto los mensajes de error producidos por archivos inexistentes o no leíbles.
- v Despliega las líneas que no se ajustan a la expresión regular proporcionada.

Existen otros dos comandos cuyo funcionamiento es muy similar a grep y también utilizan expresiones regulares: **egrep** y **fgrep**. El primero de ellos utiliza expresiones regulares verdaderas, con algunos metacaracteres adicionales y el segundo comando no utiliza metacaracteres, pero con la opción **-f** se pueden buscar varios patrones fijos simultáneamente en un mismo archivo. Para mayor información de estos comandos, consultar la ayuda en línea con los comandos:

```
% man egrep
% man fgrep
```

Las expresiones regulares que se le proporcionan a egrep, pueden utilizar como metacaracteres a los paréntesis para agrupar expresiones; por ejemplo: **(ab)*** representa a la cadena vacía y a cadenas que tienen la secuencia ab, como la cadena ababababab. El comando egrep también utiliza al metacaracter **|** como operador or; por ejemplo la expresión **(ab|xy)*** representa a la cadena vacía o a una cadena con secuencias de ab ó xy, como la cadena xyxyabxyabab.

Supongamos que deseamos buscar los archivos o directorios en el directorio de trabajo que tienen protecciones de lectura y escritura o sólo lectura para todos los usuarios, el comando utilizado sería el siguiente:

```
% ls -la | egrep '^[-d].....(rw-|r--)'
```

Otro conjunto de metacaracteres importantes son **+** y **?**, el primero indica una o más ocurrencias del carácter o cadena anteriores (no incluye a la cadena vacía); el segundo indica una ocurrencia o la cadena vacía.

La siguiente tabla resume los metacarateres para grep y egrep.

METACARACTER	SIGNIFICADO
c	Cualquier carácter que no sea especial se representa a sí mismo.
^	Inicio de línea.
\$	Fin de línea.
*	Cero o más repeticiones del carácter o cadena anteriores.
+	Una o más repeticiones del carácter o cadena anteriores, no incluye a la cadena vacía (solamente en egrep).
?	Cero o una repetición del carácter o cadena anteriores (solamente en egrep).
[]	Representa al rango de caracteres encerrado en los corchetes, por ejemplo [a-z]; [^a-z] es la negación del rango.
exp1 exp2	Representa a la expresión regular uno o a la expresión regular dos.

Tabla 5.2. Algunos parámetros de la familia grep

Con estos metacaracteres podemos especificar patrones de búsqueda bastante útiles y de manera muy sencilla. Por ejemplo, supongamos la existencia del archivo price.veg, el cual se muestra a continuación:

```
% cat /users/pub/price.veg
lettuce          .89
tomatoes         0.69
broccoli         .79
cauliflower     .89
parsley          2.14
avocado          1.19
carrots          .69
celery           .59
string-beans    1.29
onions           0.29
asparagus       1.39
corn             0.85
cabbage         .99
lima-beans      1.29
yellow-beans    1.17
spinach         0.88
```

```
mushrooms      1.09
peppers        1.39
potatoes       .39
artichokes     1.59
black-beans    1.89
black-eyed-peas 1.49
```

Si por ejemplo, quisiéramos encontrar aquellos vegetales que empiecen con la letra s, podemos utilizar el siguiente comando:

```
% grep '^s' /users/pub/price.veg
string-beans   1.29
spinach        0.88
%
```

Para encontrar los vegetales cuyo precio termina en 9:

```
% grep '9$' /users/pub/price.veg
lettuce        .89
tomatoes       0.69
broccoli       .79
cauliflower    .89
avocado        1.19
carrots        .69
celery         .59
string-beans   1.29
onions         0.29
asparagus     1.39
cabbage        .99
lima-beans     1.29
mushrooms     1.09
peppers        1.39
potatoes       .39
artichokes     1.59
black-beans    1.89
black-eyed-peas 1.49
%
```

Aquellos vegetales cuyo precio termine en 09, 39 o 79:

```
% grep '[037]9$' /users/pub/price.veg
broccoli       .79
asparagus     1.39
mushrooms     1.09
peppers        1.39
potatoes       .39
%
```

Filtros e interconexión de comandos

Para obtener un listado de aquellos vegetales que empiecen con cualquier letra de la a, a la m, sin importar si son mayúsculas o minúsculas:

```
% grep '^[a-m|A-M]' /users/pub/price.veg
lettuce          .89
broccoli         .79
cauliflower     .89
avocado         1.19
carrots         .69
celery          .59
asparagus      1.39
corn            0.85
cabbage         .99
lima-beans     1.29
mushrooms      1.09
artichokes     1.59
black-beans    1.89
black-eyed-peas 1.49
%
```

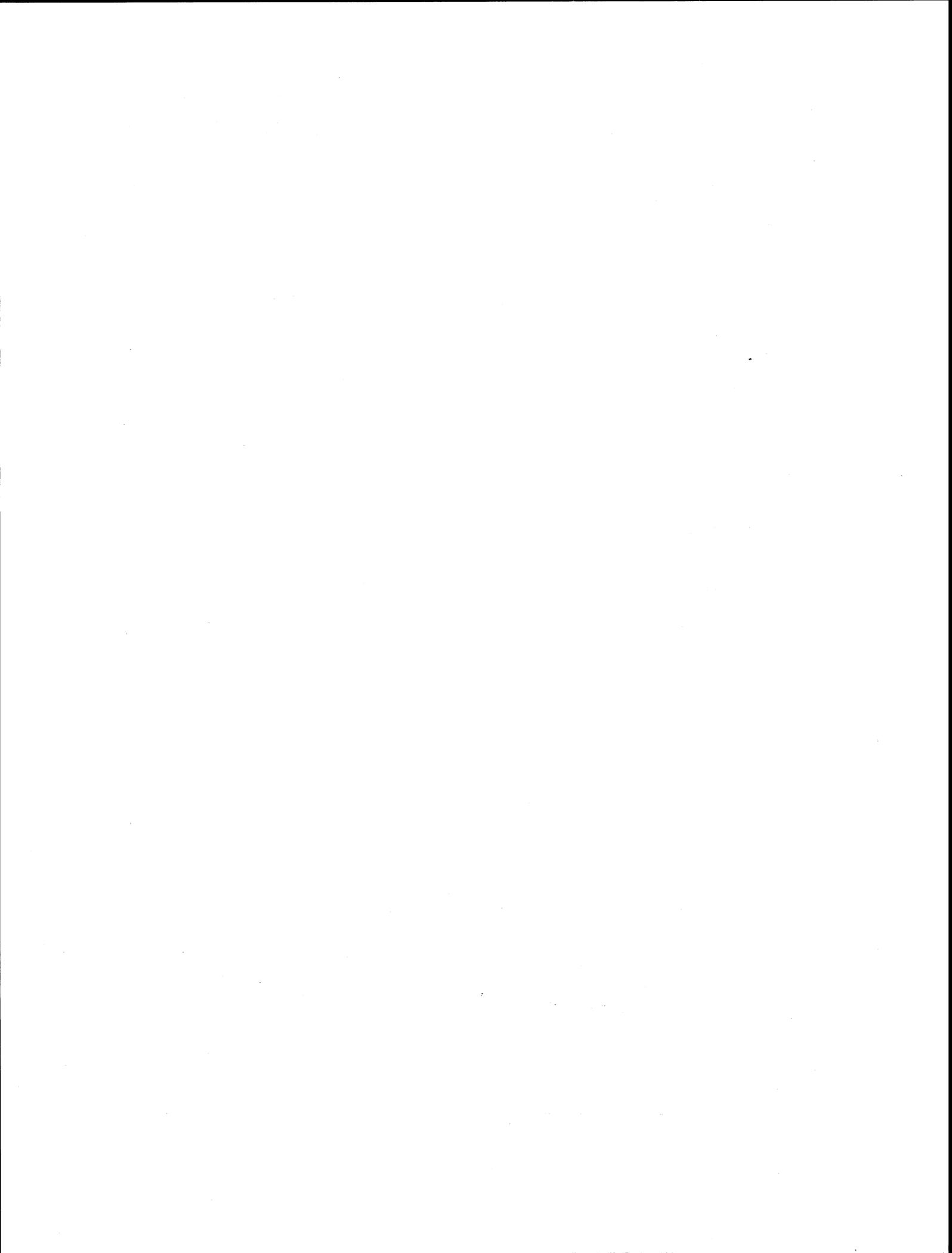
Aquellos vegetales cuyo precio esté entre 1.00 y 1.50:

```
% grep '1\.[1-4][1-9]$' /users/pub/price.veg
avocado         1.19
string-beans   1.29
asparagus      1.39
lima-beans     1.29
yellow-beans   1.17
peppers        1.39
black-eyed-peas 1.49
```

Los vegetales que empiezan con alguna vocal:

```
% grep '^[aeiou|AEIOU]' /users/pub/price.veg
avocado         1.19
onions          0.29
asparagus      1.39
artichokes     1.59
```

Y así podrían seguir los ejemplos utilizando el comando grep. Si observamos, las expresiones regulares de búsqueda, son bastante sencillas pero bastante poderosas, nos permiten encontrar patrones de caracteres muy fácilmente.



VI. Edición de archivos

En el capítulo dos indicamos como crear archivos a través de redireccionamiento de salida; sin embargo este procedimiento, aunque bueno para comenzar a crear archivos, tiene muchas limitaciones. Por una parte, sólo se puede modificar la línea que se esta proporcionando, una vez creado el archivo, este no se puede modificar. Una forma más cómodo de crear archivos y de manipular el contenido de ellos es utilizar un editor de texto.

La mayoría de los sistemas Unix poseen varios editores de texto con diferentes características; dos de los más populares son los editores *vi* y *emacs*. El editor que forma parte de la configuración estándar de Unix es el editor de línea *ed*. ¿Porque editor de línea? el término se debe a que solamente podemos editar una línea a la vez.

En este capítulo describimos el funcionamiento y configuración del editor *vi*; esto debido a que generalmente está disponible en todos los sistemas con sistema operativo Unix.

El editor *vi*

El editor *vi* o editor Visual es un editor orientado a pantalla, es decir, aprovecha las ventajas de las terminales para mostrar los cambios que se realizan en los archivos al mismo tiempo que estos se efectúan. Además permite visualizar el archivo en pantalla desplazándolo a través de esta.

Para invocar el editor *vi*, ejecutamos el siguiente comando:

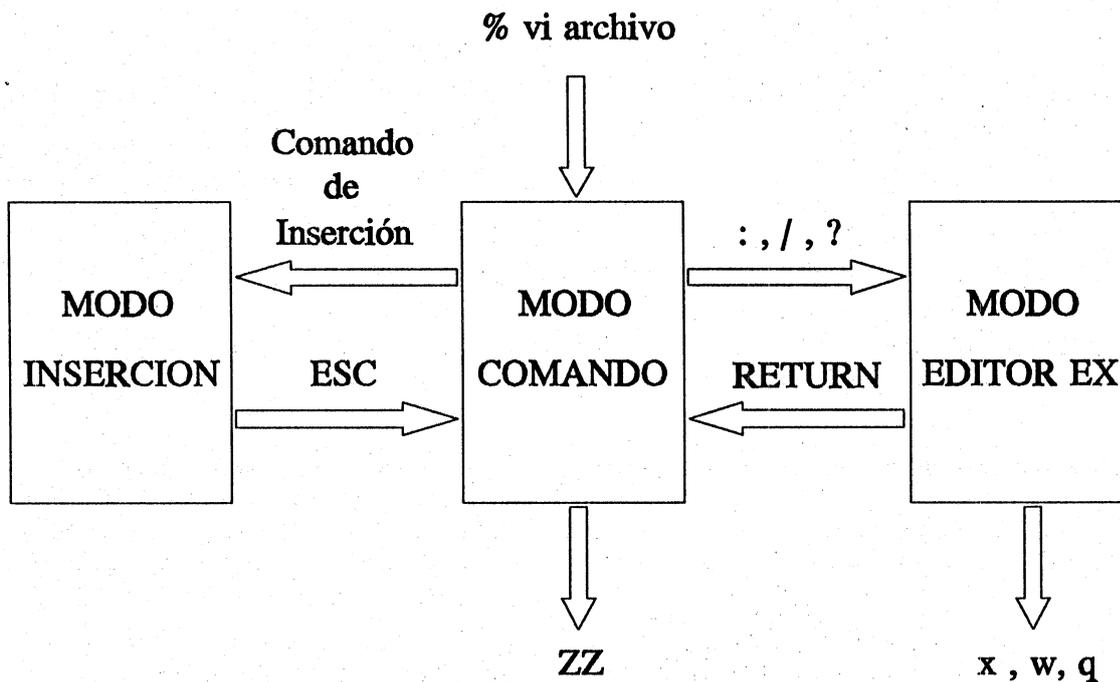
```
% vi [archivo]
```

si omitimos el nombre de archivo entramos a editar un archivo nuevo.

Cuando comenzamos a editar con *vi* se entra en el modo comando. Este modo permite el movimiento del cursor a través del archivo que

se va desplazando por la pantalla. La tabla 6.1 nos muestra algunos de los comandos para movimiento del cursor. La mayor parte de ellos son movimientos sencillos, no es necesario dar ejemplos de ellos, solamente se ejemplificarán aquellos para los cuales sea necesario.

Figura 5. Esquema del editor vi.



COMANDO	DESCRIPCION
k,	Mueve el cursor una línea arriba tratando de conservar la posición dentro de la columna en la que se encuentra.
-	Mueve el cursor una línea arriba y coloca el cursor en la primer columna.
j, RETURN	Mueve el cursor una línea abajo tratando de conservar la posición dentro de la columna en la que se encuentra.
+	Mueve el cursor una línea abajo y coloca el cursor en la primer columna.
h, ^h	Mueve el cursor hacia la izquierda.
l	Mueve el cursor hacia la derecha.
^	Mueve el cursor al principio de la línea.
\$	Mueve el cursor al final de la línea.
w, W	Mueve el cursor a la siguiente palabra.
b, B	Mueve el cursor a la palabra anterior.
e, E	Mueve el cursor al final de la palabra en la que se encuentra el cursor o de la siguiente.
)	Mueve el cursor a la próxima frase.
(Mueve el cursor a la frase anterior.
}	Mueve el cursor al próximo párrafo.
{	Mueve el cursor al párrafo anterior.
nG	Posiciona el cursor en la línea n del texto.
G	Mueve el cursor al final del texto.
^f	Posiciona el cursor en la próxima página de texto. Se entiende por página de texto las líneas que aparecen en pantalla.
^b	Mueve el cursor a la página anterior.

Tabla 6.1. Comandos del editor vi para movimiento del cursor.

Para comenzar a editar propiamente el archivo se debe pasar a modo de inserción a través del comando `i`, una vez que se ejecuta este comando, todo el texto que tecleemos se introduce en el texto. La mayoría de los sistemas Unix no permiten moverse a través de la pantalla una vez que se entra en este modo, a menos que se regrese al modo comando. Para volver a modo comando debemos presionar la tecla `<ESC>`.

La tabla 6.2 muestra los comandos más comunes de inserción y reemplazo.

COMANDO	DESCRIPCION
a	Comienza la inserción de texto a la derecha de donde se encuentra el cursor.
i	Comienza la inserción en la posición del cursor.
I	Inserción al comienzo de la línea.
A	Inserción al final de la línea.
O	Abre una línea antes de aquella en la que se encuentra el cursor y activa modo de inserción.
o	Abre una línea después de aquella en la que se encuentra el cursor y activa el modo de inserción.
R	Activa el modo de inserción a partir de la posición del cursor sobrescribiendo el texto actual.
[n]r	Reemplaza <i>n</i> caracteres a partir de la posición del cursor con la siguiente tecla que se presione.
[n]cw	Reemplazo <i>n</i> palabras, por el texto que se introduce, hasta abandonar el modo de inserción.
[n]s	Reemplaza <i>n</i> caracteres.
S	Reemplaza la línea sobre la que se encuentra el cursor.
[n]x	Borra <i>n</i> caracteres.
[n]dw	Borra <i>n</i> palabras.
[n]dd	Borra <i>n</i> líneas a partir de la que está posicionado el cursor.
[n]X	Borra <i>n</i> caracteres a la izquierda del cursor.
D	Borra a partir de donde se encuentra el cursor hasta el final de la línea.
u	Restaura lo último borrado o la última inserción.
U	Restaura la línea al estado que se encontraba antes de la última edición.

Tabla 6.2. Comandos de inserción, reemplazo y borrado.

Cuando se borran caracteres, líneas o palabras con alguno de los comandos de borrado vistos anteriormente, vi guarda en una memoria temporal el último conjunto de caracteres borrados, los cuales constituyen un bloque y se pueden colocar sobre otra parte del texto por medio de algún comando de manejo de bloques.

COMANDO	DESCRIPCION
[n]yy	Guarda en la memoria temporal las <i>n</i> líneas que se encuentran a partir de la posición del cursor.
p	Escribe el texto que se encuentra en la memoria temporal en la posición siguiente del cursor si se trata de caracteres o palabras o en la línea abajo del cursor en caso de líneas.
P	Escribe el texto que se encuentra en la memoria temporal en la posición del cursor si se trata de caracteres o palabras o en la línea arriba del cursor en caso de líneas.

Tabla 6.3. Comandos de manejo de bloques.

El editor incluye un conjunto de comandos para manipulación de archivos y búsquedas, los cuales son mostrados en las siguientes tablas. Cuando se ejecutan estos comandos la posición del cursor se cambia a la última línea.

COMANDO	DESCRIPCION
:w	Archivar el texto actual.
:w <i>nombre</i>	Grabar el texto en el archivo que se especifica con <i>nombre</i> .
:wq	Archivar el texto actual y sale de la sesión de vi.
:q	Salir de sesión de vi (antes se debieron archivar los cambios hechos al texto).
:q!	Salir de vi sin archivar los cambios.
:e <i>nombre</i>	Edita el archivo que se especifica en <i>nombre</i> .
:e + <i>nombre</i>	Edita el archivo que se especifica en <i>nombre</i> y coloca el cursor al final del archivo.
:r <i>nombre</i>	Lee el archivo especificado en <i>nombre</i> y lo coloca a partir de la siguiente línea de la posición del cursor.
:! <i>comando</i>	Ejecuta un <i>comando</i> del shell y al término de su ejecución regresa a vi.
:n	Editar el siguiente archivo de la línea de argumentos.

Tabla 6.4. Comandos de manipulación de archivos en ex.

COMANDO	DESCRIPCION
<code>/patrón</code>	Busca el patrón a través del texto a partir de la posición del cursor hacia abajo y en forma circular, es decir, si no lo encuentra y llega al final de texto comienza en la primera línea hasta llegar a la posición del cursor.
<code>?patrón</code>	Igual que el comando anterior, solamente que la búsqueda es en sentido inverso.
<code>n</code>	Búsqueda de la siguiente ocurrencia del patrón en el sentido de la búsqueda.
<code>N</code>	Búsqueda de la siguiente ocurrencia del patrón en sentido inverso de la búsqueda.

Tabla 6.5. Comandos de búsqueda en ex.

Configuración del editor

Dentro del medio ambiente del editor existen una serie de variables que nos ayudan a adaptar el editor a nuestras necesidades. Para ver el valor de esas variables, en modo comando, tecleamos:

```
:set all
```

Estas variables se establecen de tres formas: en forma de switch, esto es, estas variables pueden activarse si estaban desactivadas, o desactivarse en caso de estar activas (también se les conoce como variables tipo 'toggle'). Aquellas que requieren un valor numérico y aquellas que requieren un valor alfanumérico. Para establecer el valor de alguna de estas variables, podemos hacerlo de las siguientes formas:

```
:set opción
:set noopción
:set variable=valor
```

La siguiente tabla nos muestra algunas variables de configuración del editor.

VARIABLE	DESCRIPCION
<code>autoindent</code>	Permite que las línea de un párrafo conserve el margen de la primera línea del párrafo.
<code>noautoindent</code>	Desactiva la variable <code>autoindent</code> .
<code>tabstop=num</code>	Permite fijar la cantidad de espacios que representan la tecla <TAB> a <i>num</i> .
<code>redraw</code>	Cuando esta variable esta activada, el editor vi redibujará el texto después de un cambio como borrar una línea o insertar caracteres.
<code>noredraw</code>	Desactiva la variable <code>redraw</code> .
<code>list</code>	Esta variable permite que se visualicen los caracteres de control.
<code>nolist</code>	Desactiva la variable <code>list</code> .
<code>number</code>	Muestra los números de línea.
<code>nonumber</code>	Desactiva la variable <code>number</code> .
<code>term=tipo</code>	Establece un <i>tipo</i> de terminal para el editor.
<code>warn</code>	El usuario es advertido si trata de ejecutar un comando del shell y no a escrito los últimos cambios del archivo.
<code>nowarn</code>	Desactiva el efecto de la variable <code>warn</code> .

Tabla 6.6. Variables de configuración del editor vi.

Para visualizar el valor de una determinada opción se hace uso del comando:

```
:set opción?
```

y para desplegar solo aquellas que han sido cambiadas en las sesión tecleamos:

```
:set
```

El valor para cada una de estas variables se puede establecer desde la sesión de vi haciendo uso del comando `set`, o bien, por medio del archivo de configuración `.cshrc` para quienes trabajamos con el intérprete de comandos `csh`, o en el archivo `.exrc` para aquellos que trabajen en Bourne o Korn shell. Dentro de estos dos

últimos archivos, deberemos de establecer las opciones deseadas como el valor de la variable de ambiente EXINIT. Por ejemplo, si se desea que al entrar al editor se fijen de manera automática las variables `redraw`, `autoindent`, `warn` y se fije el intérprete de comandos a `cs`h, podemos incluir la siguiente línea dentro del archivo `.cshrc` si trabajamos con el intérprete de comandos C shell:

```
setenv EXINIT 'set redraw ai warn sh=/bin/csh'
```

y para aquellos que trabajen en Bourne o Korn shell:

```
EXINIT='set redraw ai warn sh=/bin/csh'  
export EXINIT
```

El archivo `.cshrc` debe estar en el home directory, de esta forma cada vez que se invoca a el editor `vi`, se lee este archivo y se configura automáticamente el ambiente del editor. Una vez que se ha creado la sesión de `vi`, el usuario puede configurar su ambiente desde la sesión según sus necesidades.

Macros

El editor vi permite la creación de macro instrucciones (llamadas simplemente macros) para aquellos casos en los cuales los comandos que usamos dentro de la edición de un documento son repetitivos y/o largos. Para definir una macro dentro del editor, hacemos uso del comando *map* con la siguiente sintaxis:

```
:map nombre definición<RETURN>
```

en donde *nombre* es el identificador que se le asignará a la macro, este nombre es de un solo carácter, y *definición* es el cuerpo de la macro. La definición de la macro no debe de ser mayor de 100 caracteres. Si definición llevase algún carácter especial, este deberá de ir precedido de dos caracteres **^V**, los espacios en blanco y tabuladores dentro de la definición no necesitan ir precedidos de **^V**.

Un ejemplo muy práctico de macros es definir a la tecla *q* las funciones de salvar el archivo a disco y salir del documento; esto se hace dando el siguiente comando:

```
:map q :wq^V^V<RETURN><RETURN>
```

Esto hará que de aquí en adelante, cada vez que se presione la tecla *q*, será como si se hubiesen presionado las cuatro teclas **:wq<RETURN>**. La secuencia **^V** es necesaria ya que sin ella, el primer retorno de carro hubiese terminado el comando **:** en lugar de ser parte de la definición del comando *map*. También podrá notar que existen dos **^V** ya que dentro del editor vi es necesario teclear esto dos veces para obtener uno solo. Mientras que el primer **<RETURN>** es parte de la definición del comando *map*, el segundo **<RETURN>** termina el comando.

Para borrar una macro solamente hay que usar el comando *unmap* de la siguiente forma:

```
:unmap nombre
```

UNIX

Otro ejemplo de macro es definir alguna tecla para unir dos líneas en una sola. Si observamos el archivo /etc/group ya sea con el comando cat o con el comando more, verá que este archivo contiene a todos los usuarios del sistema separados en grupos, y cada grupo contiene en una sola línea a todos sus integrantes. Esta tarea generalmente se hace al ejecutar el comando J (join) estando en modo comando, así que si estamos editando un archivo que contiene una larga serie de nombres, uno por renglón, y queremos colocarlos todos dentro de una sola línea dentro del texto, podemos hacer la siguiente macro:

```
:map f Jx<RETURN>
```

esto hace que cada que se presione la tecla f, tendrá el mismo efecto que haber presionado las teclas J (para unir los renglones) y x (para eliminar el espacio en blanco que por omisión coloca el comando J). Por ejemplo si nuestro archivo es el siguiente:

```
users*:20:  
rcr001,  
mtg000,  
hcz000,  
lpn000,  
afh000,  
...
```

y estando colocados en el primer renglón, al presionar la tecla J pasaría lo siguiente:

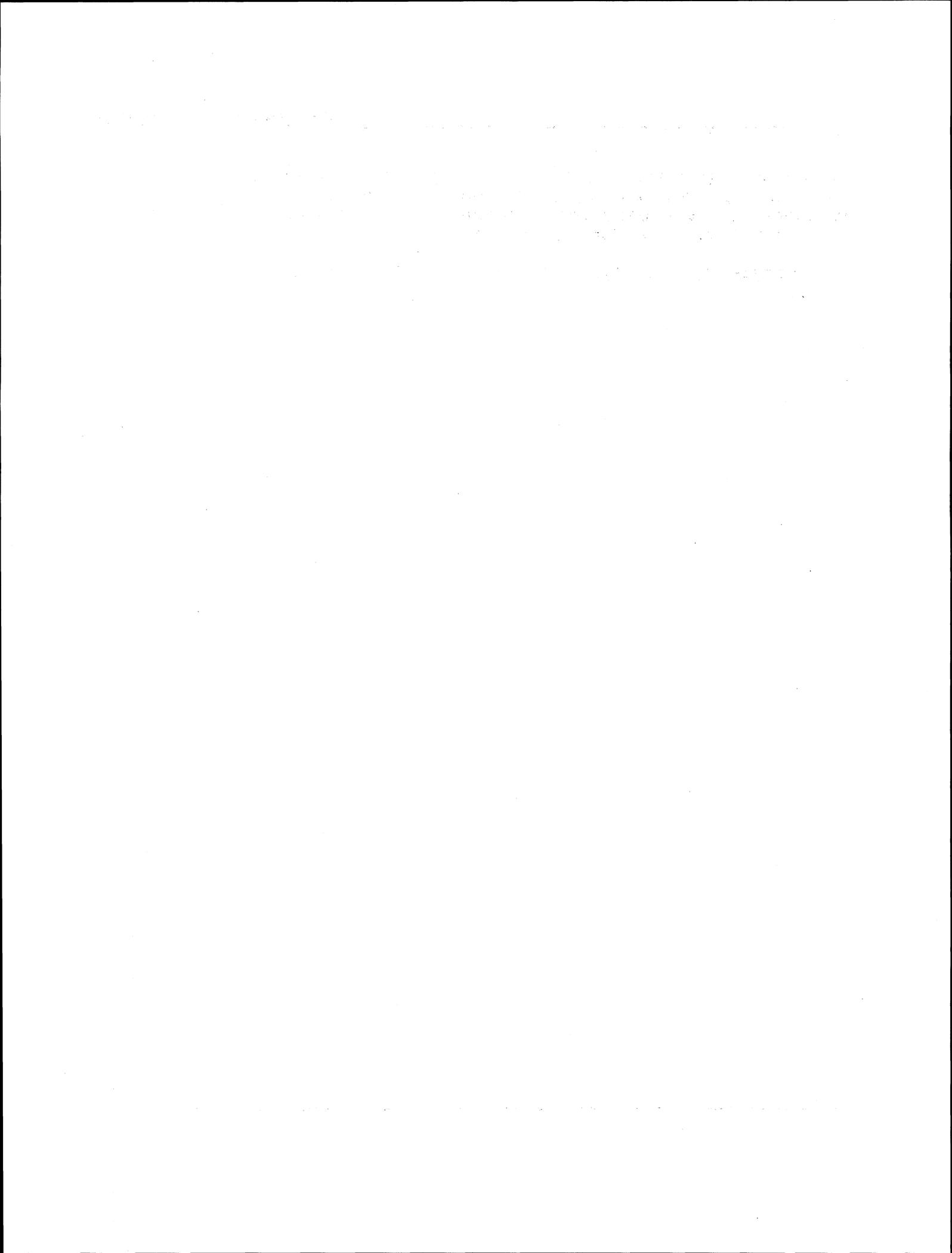
```
users*:20: rcr001,           Se presionó solamente la tecla J.  
mtg000,  
hcz000,  
lpn000,  
afh000,  
...
```

note el espacio en blanco que deja el comando J entre los dos puntos y rcr001. Para borrar este espacio en blanco ahora sería necesario presionar la tecla x. Pero si nosotros ya definimos la macro anterior, basta presionar la tecla f para obtener el mismo resultado:

```
users*:20:rcr001,mtg000,     Se presiona solamente la tecla f.  
hcz000,  
lpn000,  
afh000,  
...
```

y mejor aún, podemos repetir la macro tantas veces como queramos presionando primero el número de veces que queremos se repita esta, por ejemplo, si presionamos ahora 3f haremos que la macro f se repita 3 veces, dejando el siguiente resultado:

```
users*:20:rcr001,mtg000,hcz000,lpn000,afh000,  
...
```



VII. Programación con AWK

Uno de los filtros más importantes es `awk` (el nombre se debe a las iniciales de sus creadores: Alfred V. Aho, Peter J. Weinberger y Brian W. Kernighan), el cuál es un localizador de patrones y lenguaje de programación que examina un flujo de datos de entrada y compara cada línea de entrada con el conjunto de patrones especificados. Para cada patrón especificado, se ejecuta una serie de acciones indicadas a través de un lenguaje de programación.

Sintaxis

Los patrones pueden ser expresiones regulares como las utilizadas en `grep` y `egrep`. El programa consiste de una serie de instrucciones cuya sintaxis es muy parecida a la del lenguaje C. El programa puede ser especificado en la línea de comandos, o bien puede estar contenido dentro de un archivo e indicarlo con la opción `-f` en la línea de comandos. La sintaxis para `awk` es la siguiente:

```
awk programa [lista_archivos]
```

o bien:

```
awk -f archivo_de_programa [lista_archivos]
```

Cuando el programa se especifica en la línea de comandos, este debe encerrarse entre apóstrofes. Un programa de `awk` es una secuencia de patrones asociados con una serie de acciones:

```
patrón { acciones }  
patrón { acciones }
```

...

En `awk` el patrón o la acción pueden ser omitidas, pero nunca ambas. El patrón selecciona los registros o líneas para los cuales se van a ejecutar las acciones asociadas, si dicho patrón no se especifica, se seleccionan todos los registros del flujo de datos de entrada; por otra parte, si no se especifican acciones

UNIX

asociadas, se imprime toda la línea para las líneas seleccionadas en el patrón. El programa de awk es procesado para cada una de las líneas de entrada. Aunque el patrón o las acciones son opcionales, las acciones se encierran entre llaves para distinguirlas de los patrones.

Por ejemplo, el siguiente programa en awk:

```
/x/ { print }
```

imprime cada línea de entrada que contenga el carácter x dentro de ella.

Casi al igual que sort, awk toma el flujo de datos de entrada como una secuencia de registros divididos en campos e identifica al primer campo como \$1, al segundo como \$2 y así sucesivamente. \$0 identifica a todo el registro. De la misma forma que el comando sort, awk utiliza, por omisión, el blanco como separador de campos.

Cuando se utilizan expresiones regulares como patrones en awk, estas se encierran entre diagonales. Por ejemplo, si deseamos listar los nombres de los subdirectorios en el directorio actual de trabajo, podemos ejecutar el siguiente comando:

```
% ls -la | awk '/^d/ { print $9 }'
```

donde la acción indicada, es la de imprimir el campo nueve (\$9), el cual representa el nombre del archivo (en este caso el nombre del subdirectorio) de cada una de las líneas de salida del comando `ls -la`⁷ seleccionadas con la expresión regular `/^d/`.

⁷ Recuerde que en HP-UX el comando `ls -la` imprime el grupo al cual pertenece el archivo, haciendo que el nombre del archivo sea el noveno campo. En Ultrix el comando `ls -la` no imprime el grupo al cual pertenece el archivo, haciendo que el campo del nombre del archivo, sea el octavo campo del listado.

Parámetros

Para el archivo nombres, mencionando anteriormente (página 5-6) si queremos obtener un listado de los nombres presentando primero su nombre, apellido paterno y apellido materno, podemos usar el siguiente comando:

```
% awk '{ print $3,$1,$2 }' nombres
Héctor Badillo Rojas
Vicky López Uresti
Ricardo Alvarez Quiroz
Edwin Navarro Pliego
Gabriela Magallanes González
Beatriz Barrera Hernández
Gerardo Meza Gil
%
```

La instrucción print causa que se haga una copia a la salida estándar de los parámetros listados, si estos se separan por comas, se está indicando que en la salida se separen con el separador de campos de salida por omisión, el cual es el carácter blanco. Cada vez que se manda una instrucción print, se manda a la salida el carácter de fin de línea. La salida de print puede ser dirigida a múltiples archivos; por ejemplo, si quisiéramos mandar los apellidos del archivo nombres al archivo ape y los nombres al archivo nom, el comando a ejecutar sería:

```
% awk '{ print $1,$2 > "ape"; print $3 > "nom" }' nombres
```

Los nombres de los archivos deben de ir entre comillas. Para separar instrucciones en una misma línea se utiliza el carácter ;. En un programa que este almacenado en un archivo cada instrucción se puede colocar en una línea, sin necesidad de separarlas con ;. En el ejemplo anterior podemos utilizar >> en lugar de > para producir el mismo resultado que cuando se hace redireccionamiento de salida.

UNIX

Para tener un mejor control del formato que se da a la salida, podemos utilizar la instrucción `printf`, la cual tiene la siguiente sintaxis:

```
printf "formato", expresión1, expresión2, ..., expresiónn
```

la cual formatea las expresiones de la lista de acuerdo a las especificaciones de la cadena de formato y las imprime. La cadena de formato sigue las mismas convenciones que las de la instrucción `printf` del lenguaje de programación C.

Por ejemplo, supongamos la existencia del siguiente archivo llamado `countries`:

```
% cat countries
Russia      8650 262  Asia
Canada      3852 24   North America
China       3692 866  Asia
USA         3615 219  North America
Brazil      3286 116  South America
Australia   2968 14   Australia
India       1269 637  Asia
Argentina   1072 26   South America
Sudan       968  19   Africa
Algeria     920  18   Africa
%
```

la instrucción:

```
{ printf "%10s %6d %6d\n", $1, $2, $3 }
```

imprimiría `$1` como una cadena de 10 caracteres justificada a la derecha, los campos segundo y tercero formarían una tabla de dos columnas de hasta 6 dígitos cada una. Para producir la salida descrita, hacemos uso del siguiente comando:

```
% awk '{ printf "%10s %6d %6d\n", $1, $2, $3 }' countries
Russia      8650 262
Canada      3852 24
China       3692 866
USA         3615 219
Brazil      3286 116
Australia   2968 14
India       1269 637
Argentina   1072 26
Sudan       968  19
Algeria     920  18
%
```

Si la cadena de formato contiene caracteres ordinarios, estos son copiados al flujo de salida. Esta cadena, además puede contener especificaciones de conversión, cada una de las cuales causa la conversión impresa de las expresiones contenidas en la lista. Cada una de las especificaciones de conversión comienzan con el carácter % y terminan con un carácter de conversión. Los caracteres de conversión se muestran en la tabla 7.1. Entre el % y el carácter de conversión puede aparecer, en orden:

- Un signo menos, que indica justificación a la izquierda del argumento convertido.
- Un número que indica el ancho mínimo del campo.
- Un punto, que separa el ancho de campo de la precisión.
- Un número que indica el número de dígitos después del punto decimal para un valor numérica, o el número máximo de una cadena de caracteres.

CARACTER	FORMA EN LA QUE ES IMPRESO EL ARGUMENTO
d	Número decimal.
o	Número en formato octal.
x	Número en formato hexadecimal.
c	Carácter.
s	Cadena de caracteres.
f	Número fraccionario.

Tabla 7.1. Conversiones para printf

Por ejemplo, el siguiente programa:

```
% awk '{printf"%3d %-10s %-10s %-10s\n",NR,$3,$1,$2}' nombres
1 Héctor      Badillo      Rojas
2 Vicky       López        Uresti
3 Ricardo     Alvarez      Quiroz
4 Edwin       Navarro      Pliego
5 Gabriela    Magallanes   González
6 Beatriz     Barrera      Hernández
7 Gerardo     Meza         Gil
%
```

da formato a cada uno de los campos de salida. Printf no manda al flujo de salida el carácter de fin de línea como lo hace print, este se debe especificar en la cadena de control y se representa como \n. En el ejemplo anterior, se manda a la salida de datos una variable predefinida de awk: NR, la cual almacena el número de registro o línea de entrada que procesa en ese momento el programa; cada que awk procesa una línea diferente esta variable cambia su valor. Otras variables predefinidas en awk se muestran en la siguiente tabla:

VARIABLE	DESCRIPCION
NF	Número de campos de la línea de entrada.
FS	Separador de campos de entrada. Por omisión es un blanco o tabulador.
RS	Carácter separador de líneas de entrada (por omisión es el carácter de nueva línea).
NR	Número actual de la línea de entrada.
OFS	Separador de campos de salida. Por omisión es un blanco.
OFMT	Formato de impresión de números con la declaración print. Por omisión es %.6g
FILENAME	Nombre del archivo de entrada actual.

Tabla 7.2. Variables predefinidas en awk.

Existen dos patrones especiales en awk, el primero de ellos es *BEGIN*, con el cual especificamos las acciones que se deberán realizar antes de que se comience a procesar la primera línea de entrada; el otro patrón es *END* que especifica las acciones que se realizan después de haber leído la última línea de entrada. El patrón *BEGIN* es muy utilizado para hacer inicialización de variables.

Para ilustrar los patrones *BEGIN* y *END*, consideremos los siguientes ejemplos; primero, se obtendrá el número de líneas procesadas:

```
% awk 'END { printf "\t%d\n", NR }' nombres
7
%
```

El resultado anterior se pudo haber obtenido también con el comando *wc* con la opción *-l*, tomando como entrada un sólo archivo.

El segundo ejemplo consiste en obtener las claves válidas en el sistema junto con el home directory de cada una de estas cuentas. Para ello debemos tomar en cuenta lo siguiente: existe un archivo llamado /etc/passwd en donde se registran las claves del sistema, dicho archivo contiene registros con información para cada una de las claves: el registro consiste de varios campos separados por el carácter :, el significado de los campos se muestra en el siguiente registro de ejemplo:

```
curso101:l5eT9R5Ggrxfs:310:15:clave para cursos:/users/cecafi/curso101:/bin/csh
Clave |
Contraseña encriptada |
Identificador de usuario |
Identificador de grupo |
Comentario |
Home directory |
Shell |
```

Por lo tanto, para obtener el resultado deseado, se daría el siguiente comando:

```
% awk 'BEGIN { FS=":" } { print $1,$6 }' /etc/passwd
root /
ceo /
daemon /
bin /bin
adm /usr/adm
uucp /usr/spool/uucppublic
lp /usr/spool/lp
hpdb /
raq /users/cecafi/raq
edwin /users/cecafi/edwin
amezcua /users/cecafi/amezcua
gaby /users/cecafi/gaby
hbr /users/cecafi/hbr
nam /users/cecafi/nam
samuel /users/cecafi/samuel
rna000 /users/alumnos/rna000
...
%
```

Operaciones aritméticas

El lenguaje de programación de awk, emplea un conjunto de operadores para llevar a cabo operaciones aritméticas entre variables. Los operadores aritméticos son los siguientes:

+	Suma
-	Resta
*	Multiplicación
/	División
%	Residuo

En awk podemos manejar variables de tipo numérico y cadena; sin embargo, no es necesario definir las, ya que se definen al momento de utilizarse. Tampoco es necesario inicializarlas, ya que por omisión, las variables numéricas se inicializan en cero y las tipo cadena lo hacen con la cadena vacía. Dependerá del contexto tratar a una variable como un número o como una cadena, en casos ambiguos, el valor de cadena se utiliza a menos que los operandos sean numéricos. Existen una serie de funciones para manipulación de variables o constantes numéricas o cadenas, estas se muestran a continuación:

FUNCION	DESCRIPCION
<code>cos(val)</code>	Coseno de <i>val</i> .
<code>sin(val)</code>	Seno de <i>val</i> .
<code>exp(val)</code>	Exponencial de <i>val</i> .
<code>int(val)</code>	Obtiene la parte entera de <i>val</i> .
<code>log(val)</code>	Logaritmo natural de <i>val</i> .
<code>length(cad)</code>	Longitud de la cadena <i>cad</i> .
<code>substr(cad, m, n)</code>	Obtiene una subcadena de <i>m</i> caracteres de la cadena <i>cad</i> , comenzando en la posición <i>n</i> .
<code>index(cad1, cad2)</code>	Devuelve la posición a partir de la que se encuentra <i>cad2</i> en <i>cad1</i> , o cero si no se encuentra.
<code>sprintf(f, e1, ...)</code>	Devuelve una cadena con el formato especificado en la cadena <i>f</i> , tomando los argumentos <i>e1, ...</i>

Tabla 7.3. Funciones predefinidas en awk

Consideremos el ejemplo de un programa cuyo funcionamiento es igual al de `wc` cuando se le proporciona un flujo de datos de entrada de un archivo (más adelante generalizaremos el ejemplo para cuando se toman varios archivos de entrada). El programa es el siguiente:

```
% cat wc_awk
{
  caracteres = caracteres + length($0) + 1
  palabras = palabras + NF
}
END {
  printf "\t%d\t%d\t%d\n", NR, palabras, caracteres
}
%
```

Para hacer uso de este programa, podemos correrlo sobre si mismo obteniendo el siguiente resultado:

```
% awk -f wc_awk wc_awk
7      19      128
%
```

indicándonos que el archivo tiene 7 líneas, 19 palabras y 128 caracteres.

Las operaciones de asignación como la siguiente:

```
palabras = palabras + NF
```

en las cuales se modifica una variable con una operación sobre la misma, se pueden escribir en forma compacta de la siguiente forma:

```
palabras += NF
```

en términos generales, si se tiene una expresión de la forma:

```
var = var op exp
```

donde:

```
var = nombre de variable
op  = algún operador aritmético
exp = expresión
```

se puede transformar a:

```
var op = exp
```

UNIX

por lo tanto, se tienen los operadores +=, -=, *=, %= y /=, equivalentes a: a=a+b, a=a-b, a=a*b, a=a%b, y a=a/b respectivamente.

Los patrones en awk, pueden involucrar operadores aritméticos; pero también pueden ser expresiones que incluyan otro tipo de operadores, como lógicos, relacionales o de evaluación de expresiones regulares. Por ejemplo, para especificar un patrón que seleccione los registros de entrada que tienen cinco campos, utilizaríamos la siguiente expresión:

```
NF == 5
```

Un patrón que seleccione registros cuya longitud máxima sea de 80 caracteres:

```
length($0) <= 80
```

Para seleccionar registros que tengan solamente 10 caracteres se podrían utilizar alguno de los siguientes patrones:

```
$0 == /^.....$/  
length($0) == 10
```

Un patrón que seleccione los registros pares con cinco campos cuya longitud total sea menor a 80 caracteres sería:

```
NR % 2 == 0 && NF == 5 && length($0) <= 80
```

En el ejemplo anterior, nos podría surgir la pregunta ¿qué operadores se evalúan primero?, para contestarla hay que saber que todos los operadores tienen cierta precedencia que indica el orden de evaluación. Si queremos romper dicha precedencia, podemos utilizar paréntesis. La siguiente tabla muestra los operadores válidos en awk en orden creciente de precedencia.

OPERADOR	DESCRIPCION
= += -= *= %= /=	Asignación
	OR lógico. Se aplica la regla del corto circuito.
&&	AND lógico. Se aplica la regla del corto circuito.
!	NOT lógico. Negación.
> < >= <= == != ~ !~	Operadores relacionales. Los dos últimos se aplican en expresiones regulares, para denotar correspondencia y no correspondencia.
+ -	Suma y resta.
* / %	Multiplicación, división y residuo.
++ --	Incremento y decremento unitario.

Tabla 7.4. Operadores en awk por orden creciente de precedencia.

Control de flujo

Existen algunas proposiciones en awk para especificar un orden en la realización de las operaciones de un programa, dichas proposiciones son las estructuras de control de flujo de la programación estructurada.

UNIX

La primera de ellas es la proposición if-else, la cual tiene la siguiente sintaxis:

```
if (expresión)
    proposición1
else
    proposición2
```

donde las proposiciones 1 y 2 son expresiones o un bloque que consiste en expresiones encerradas entre llaves {} y la parte else es opcional. La expresión se evalúa, si es verdadera (si el resultado de la expresión tiene un valor diferente de cero), se ejecuta la *proposición1*; si es falsa (el valor de la expresión es cero) y si existe la parte else, se ejecuta la *proposición2*.

El siguiente programa en awk, obtiene el archivo más grande así como el más pequeño al pasarle como flujo de datos de entrada un listado con información de los archivos de un directorio.

```
% cat size_awk
{
    if ( NR == 1 )
        next
    if ( NR == 2 ) {
        min = $5
        max = $5
        archmax = $9
        archmin = $9
    }
    else {
        if ( $5 > max ) {
            max = $5
            archmax = $9
        }
        if ( $5 < min ) {
            min = $5
            archmin = $9
        }
    }
}
END {
    printf "Archivo más grande: %s %7.2f Kbytes\n", archmax, max/1024
    printf "Archivo mas pequeño: %s %7.2f Kbytes\n", archmin, min/1024
}
%
```

La forma de invocarlo, para obtener el archivo más grande y el más pequeño, por ejemplo, dentro del subdirectorio /etc sería:

```
% ls -la /etc | awk -f size_awk
Archivo mas grande: update 700.00 kbytes
Archivo mas pequeño: lanscan_data 0.00 kbytes
%
```

El comando `ls` genera, como ya se había visto, un listado con información de los archivos; sin embargo, la primera línea de salida es un encabezado que indica el total de bloques de disco (de 512 Kb) que ocupan todos los archivos, por lo cual esta primera línea no debe procesarse. La instrucción `next` provoca que se lea el siguiente registro en el flujo de datos de entrada. Nuevamente, debemos tener en cuenta que este programa hace referencia a los campos 5 y 9 de la salida producida por el comando `ls -la` del sistema operativo HP-UX, si se desea ejecutar este mismo programa sobre la estación de trabajo DEC 3100, se deberán cambiar los campos \$5 por \$4 y \$9 por \$8 que en Ultrix son el tamaño y nombre del archivo respectivamente, o bien, utilizar el comando `ls -lag` en lugar de `ls -la`.

El siguiente programa es una implementación con `awk` del comando `wc` cuya entrada puede ser uno o varios archivos:

```
% cat wc_awk
BEGIN {
    archivo = FILENAME }
{
    if ( archivo != FILENAME )
    {
        printf"%8d%8d%8d %s\n",NR-1,palabras,caracteres,archivo
        palabras = 0
        caracteres = 0
        NR = 1
    }
    caracteres += length($0) + 1
    palabras += NF
    archivo = FILENAME
    totalCar += length($0) + 1
    totalPal += NF
    totalLin++
}
END {
    printf"%8d%8d%8d %s\n",NR,palabras,caracteres,FILENAME
    printf"%8d%8d%8d total\n",totalLin,totalPal,totalCar
}
%
```

Supongamos que ejecutamos este programa sobre un conjunto de archivos, el resultado sería algo parecido a lo siguiente:

```
% awk -f wc_awk awk*
 2      4      32 awk1
18     36     266 awk2
 5     10      70 awk3
 4      8      58 awk4
14     28     205 awk6
 7     14     116 awk8
50    100     747 total
%
```

El comando `awk`, también incluye proposiciones que permiten realizar ciclos, dichas proposiciones son `while` y `for`. La sintaxis para cada una de ellas se muestra a continuación:

```
while (expresión)
    proposición
```

```
for(expresión1;expresión2;expresión3)
    proposición
```

Para el caso de `while`, se evalúa la *expresión*, si esta es verdadera se ejecutan las instrucciones que forman la *proposición* y se vuelve a evaluar la *expresión*. El ciclo continúa hasta que la *expresión* sea falsa, momento en el cual la ejecución del programa sigue después de la *proposición*. Con la proposición `while` se puede implementar el comportamiento del `for` de la siguiente forma:

```
expresión1
while (expresión2) {
    proposición
    expresión3
}
```

La *expresión1* en el `for` generalmente es una inicialización y solamente se realiza una vez, al comienzo del `for`; después se evalúa la *expresión2* si esta es verdadera se ejecutan las instrucciones que forman la *proposición* y finalmente se realiza la *expresión3*, para posteriormente volver a evaluar la *expresión2*. El ciclo continúa hasta que la *expresión2* sea falsa.

Arreglos

El lenguaje de programación de awk nos permite manejar arreglos. Al igual que las variables, los arreglos no se necesitan inicializar y pueden ser arreglos numéricos o de cadenas de caracteres. La inicialización funciona igual que con las variables. El siguiente programa sort.awk ordena las líneas de entrada por el método conocido como "la burbuja"; en dicho ejemplo se almacenan las líneas de entrada en un arreglo de cadenas de caracteres.

```
% cat sort.awk
# Programa de ordenamiento de las líneas de entrada
{
    línea[NR] = $0
}
END {
    for(i=1;i<NR;i++)
        for(j=NR;i<j;j--)
            if (línea[j-1] > línea[j]) {
                aux = línea[j]
                línea[j] = línea[j-1]
                línea[j-1] = aux
            }
        for(i=1;i<=NR;i++)
            print línea[i]
    }
}
```

Si ejecutamos el comando `ls -la`, obtenemos una salida parecida a la siguiente:

```
% ls -la
total 20
drwxrwxrwx  3 root    root    3072 May  4 20:50 .
drwxr-xr-x 18 root    root    2048 May  3 21:19 ..
drwxrwxrwx  2 bin     sys     1024 May  3 16:54 .X11-unix
-rw-rw-rw-  1 root    sys      0 Mar  9 21:36 gated.conf
-rw-rw-rw-  1 root    sys      0 Mar  9 21:36 gated.pid
-rw-rw-rw-  1 root    sys     123 May  4 20:48 logprogl
-rw-rw-rw-  1 root    sys     533 May  4 20:51 logsort
-rw-rw-rw-  1 root    sys     617 May  4 20:46 progl.awk
-rw-rw-rw-  1 root    sys     435 May  4 20:50 sort.awk
```

UNIX

Ahora, si ejecutamos el mismo comando y la salida la redireccionamos como entrada del comando awk con el programa anterior, obtenemos lo siguiente:

```
% ls -la /tmp | awk -f sort
-rw-rw-rw- 1 root    sys      0 Mar   9 21:36 gated.conf
-rw-rw-rw- 1 root    sys      0 Mar   9 21:36 gated.pid
-rw-rw-rw- 1 root    sys     123 May  4 20:48 logprogl
-rw-rw-rw- 1 root    sys     435 May  4 20:50 sort.awk
-rw-rw-rw- 1 root    sys     617 May  4 20:46 progl.awk
drwxr-xr-x 18 root    root    2048 May  3 21:19 ..
drwxrwxrwx 2 bin     sys    1024 May  3 16:54 .X11-unix
drwxrwxrwx 3 root    root    3072 May  4 20:50 .
total 18
```

lo cual al parecer está mal, pero no es así, recordemos que al hacer las comparaciones de las cadenas de caracteres, estas comparaciones se hacen en base al orden lexicográfico donde el carácter - es "menor" que cualquiera de las letras, 18 es "menor" que 2 por empezar con '1', etc.

Normalmente, los índices de los arreglos son valores enteros; sin embargo, el lenguaje de programación de awk permite manejar como índice cualquier valor. Cuando los índices de los arreglos no son valores enteros se habla de los llamados arreglos asociativos. Cuando se manejan arreglos asociativos, podría surgir la pregunta ¿cómo se recorre el arreglo, si los índices no llevan un orden?. Los índices para los arreglos asociativos tienen un orden imprevisto y awk utiliza un esquema de hashing para garantizar que el acceso a cualquier elemento del arreglo tarde más o menos el mismo tiempo. Para recorrer todos los elementos del arreglo se utiliza una variante de la proposición for, cuya sintaxis es la siguiente:

```
for(índice in arreglo)
    proposición
```

En esta proposición for cada uno de los valores de los índices se van asignando en cada iteración a la variable *índice*, hasta que el arreglo denominado *arreglo* se recorre completamente. El orden en que se van obteniendo los índices es impredecible.

Un ejemplo que podría aclarar el uso de los arreglos asociativos, lo constituye el programa que obtiene la frecuencia que tiene cada una de las palabras de un texto. En principio sabemos sabe cuantas palabras contiene un texto cualquiera, si se contara únicamente con arreglos con índices enteros se debería de tener un mapeo de un índice a una palabra; sin embargo, con los

arreglos asociativos esto es innecesario ya que las mismas palabras son los índices y los elementos de los arreglos contienen la frecuencia. Este programa es el siguiente:

```
% cat frecuencia.awk
{
for ( i=1; i <= NF; i++ )
    palabras[$i]++
}
END {
    for ( i in palabras )
        printf "%s %d\n",i,palabras[i]
}
%
```

Si corremos este programa sobre el archivo: /usr/man/man1/vuehelp.1 obtendríamos lo siguiente:

```
% awk -f frecuencia.awk /usr/man/man1/vuehelp.1
root 1
server 5
viewed 1
created, 2
serves 1
(called 1
1X 1
Search 3
cause 1
opens 1
.
.
.
%
```

Lo que nos indica que el archivo vuehelp.1 tiene una sola vez la palabra root, cinco veces la palabra server, etc.

La nueva versión de AWK

Las implementaciones más recientes de Unix, incluyen una nueva versión de AWK, comúnmente llamada "nuevo AWK". Esta nueva versión, es compatible con la descrita en este capítulo. En algunos sistemas, se cuenta con las dos versiones: **nawk** y **oawk**, que representan a la nueva versión y a la descrita en este capítulo respectivamente.

Algunas extensiones de la nueva versión de AWK se mencionan a continuación:

- Se pueden hacer interconexiones de comandos en el programa.

Suponiendo que se desea generar una lista de Continentes-población, ordenadas alfabéticamente por Continente. El siguiente programa **awk** acumula los valores de la población del 3er campo para cada nombre de continente contenido en el cuarto campo dentro de un arreglo llamado **pop**. Después de esto imprime cada continente con su respectiva población y la salida la 'pasa' al comando **sort**.

```
BEGIN { FS = "\t" }
        { pop[$4] += $3 }
END    { for (c in pop)
        print c ":" pop[c] | "sort" }
```

Corriendo este programa **awk** sobre el archivo **countries** visto anteriormente se obtendría lo siguiente:

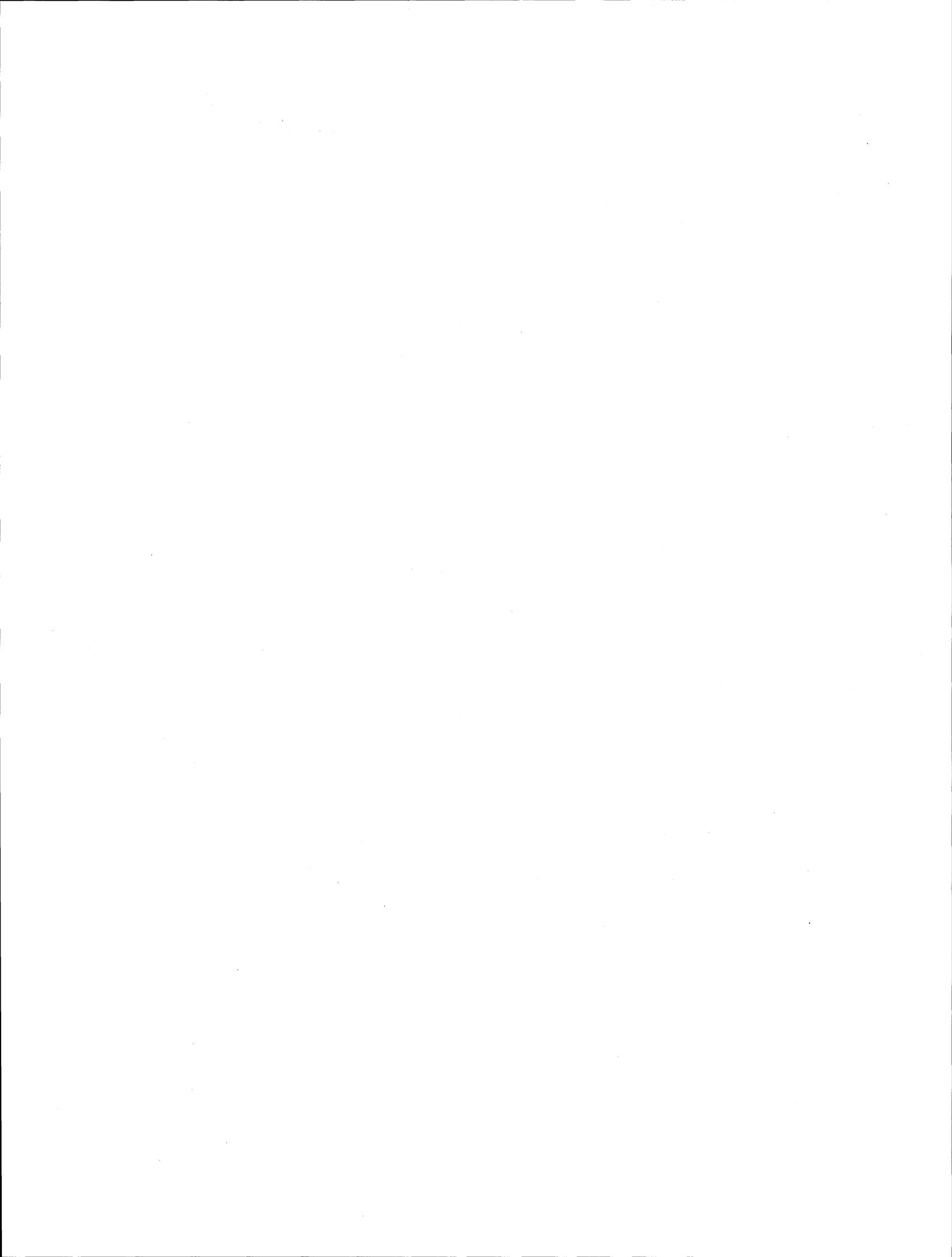
```
Africa:37
Asia:1765
Australia:14
North America:243
South America:142
```

- Se pueden borrar a tiempo de ejecución arreglos asociativos.
- El parser para el nuevo AWK elimina algunas ambigüedades que se permiten en la versión anterior.
- Se pueden definir funciones. Una función se define como:

```
function nombre(lista_de_argumentos) {  
    expresiones  
}
```

Por ejemplo se puede definir la función factorial de la siguiente forma:

```
function fact(n) {  
    if (n <= 1)  
        return 1  
    else  
        return n * fact(n-1)  
}  
{ print $1 "! es " fact ($1) }
```



VIII. Shell Scripts

Los archivos de comandos (Shell scripts) son programas que contienen instrucciones en un lenguaje de comandos que pueden ser tecleados directamente en el prompt o pueden ser incluidos en un archivo para ser ejecutados en cualquier momento.

Ciertas tareas requieren que se proporcione una secuencia de comandos cada vez que se ejecutan. Agrupando estos comandos en un shell script, los podemos ejecutar con un solo comando, ahorrando tiempo y esfuerzo así como incrementando la exactitud.

Los shell scripts son también conocidos como programas de shell (shell programs) o procedimientos de shell (shell procedures). Para la creación de shell scripts todo lo que debemos hacer es crear un archivo que contenga comandos de Unix ordinarios en el orden en que normalmente se proporcionarían.

Los comandos que se escriben dentro de un shell script son los mismos que se usan desde la línea de comandos, sin embargo, debido a que existen varios intérpretes de comandos como son el C-shell (csh), el Bourne-shell (sh), el Korn-shell (ksh) y otros más, ciertas instrucciones, relacionadas principalmente con el control de flujo, deberán de ser escritas con la sintaxis propia del intérprete de comandos encargado de ejecutar nuestro archivo de comandos. Los ejemplos mostrados en este capítulo están escritos para el intérprete de comandos C-shell (/bin/csh), pero si se desea, todos estos ejemplos pueden ser trasladados para su ejecución bajo otro intérprete de comandos con ligeras modificaciones. Por ejemplo, mientras que en C-shell se escribe la instrucción if de la siguiente manera:

```
if (condición) then
    expresión(es)
endif
```

en Bourne-shell se escribiría:

```
if [condición]
then
    expresión(es)
fi
```

UNIX

Por ejemplo, si quisiéramos que la computadora nos diera un listado del número y nombres de los subdirectorios que se encuentran en nuestro home directory, podríamos utilizar comandos de Unix como los siguientes:

```
% cd
% ls -l | grep '^d' | wc -l      Contar el número de subdirectorios
...
% ls -l | grep '^d'            Listar solo los subdirectorios
...
%
```

Pero esto se puede hacer con un solo comando si nosotros colocamos todos estos comandos dentro de un shell script. El siguiente shell script es usado para contar y listar los subdirectorios en nuestro home directory:

```
#!/bin/csh
# 'Archivo: cld -- cuenta y lista directorios'
#
# 'Este comando cuenta el numero de subdirectorios en el'
# 'home directory de un usuario, imprime el numero, y lista'
# 'los nombres de todos los subdirectorios.'
#
cd
echo -n "Tu home directory es "
pwd
echo -n "El numero de subdirectorios en tu home directory es "
ls -l | grep '^d' | wc -l; # 'cuenta numero de directorios'
echo "Estos son:"
ls -l | grep '^d';        # 'lista los directorios'
```

Algunos puntos a destacar sobre este archivo son:

- Es importante documentar los programas propiamente. Es una muy buena costumbre colocar al inicio del archivo la ruta completa del intérprete de comandos que ejecutará nuestro shell script. En este caso se trata del intérprete C-shell (/bin/csh). Esto es muy útil, sobre todo cuando después de un tiempo editamos nuestro archivo, esto nos recordará para que intérprete de comandos está escrito evitándonos grandes dolores de cabeza por utilizar la sintaxis de otro intérprete de comandos.

- El símbolo de número (#) es usado para introducir comentarios. Los comentarios no son interpretados por el shell como comandos. Son ignorados.
- Cada línea de comentario es encerrada entre apóstrofes. Así se asegura que el comentario sea efectivamente tratado como comentario.
- Para que el shell script sea ejecutado por el intérprete de comandos C-shell, es muy importante que el primer carácter del primer renglón contenga el símbolo de número (#). Si no es así el intérprete que ejecutaría nuestro archivo de comandos sería el Bourne-shell.
- Un comentario que está en la misma línea de un comando es llamado un comentario en línea (in-line comment). Hay que asegurarse de separar el comando del comentario con un separador de comandos punto y coma (;). Si no se usa, el comentario es interpretado como un argumento del comando.
- La opción -n del comando echo le dice al shell no imprimir un retorno de carro al final de la línea impresa.

Ejecutando Shell Scripts.

Hay tres formas de ejecutar un shell script:

- 1.- Invocar un subshell.
- 2.- Correr el script en el shell actual.
- 3.- Hacer el script ejecutable.

Invocando un subshell.

Para ejecutar un shell script en un subshell, escribimos el nombre del shell que ejecutará nuestro shell script pasándole como parámetro el nombre del script, en este caso para C-shell:

```
% csh nombre_archivo
```

donde `nombre_archivo` se refiere al nombre del archivo del shell script. Para el caso del script anterior (`cld`) lo pudimos haber mandado ejecutar con la siguiente instrucción:

```
% csh cld
```

Corriendo el script en el shell Actual.

Este método tiene el mismo efecto que ejecutar todos los comandos del shell separados por punto y comas y diagonales invertidas. Para hacer esto utilizamos el comando `source` de la siguiente manera:

```
% source nombre_archivo
```

Para el caso del script anterior (`cld`) lo pudimos haber mandado ejecutar con la siguiente instrucción:

```
% source cld
```

Haciendo el Script Ejecutable.

El tercer método de ejecutar un shell script es hacer el script ejecutable usando el comando `chmod` (agregar el permiso de ejecución), es decir para convertir los shell scripts a comandos, todo lo que se tiene que hacer es hacerlos ejecutables con este comando. Por ejemplo para convertir el shell script `cld`, en un comando que se pueda ejecutar, se debe realizar lo siguiente:

```
% ls -l cld
-rw-r--r-- 1 amezcua cecafi 613  Oct 6 12:39 cld
% chmod ugo+x cld
% ls -l cld
-rwxr-xr-x 1 amezcua cecafi 613  Oct 6 12:40 cld
% cld
...
%
```

Esta forma del comando `chmod` le dice al shell sumar (+) permiso de ejecución (x) al usuario (u), grupo (g), y otros (o).

El hacer un shell script ejecutable no tiene efecto sobre la habilidad para usar `csh`, y `source` para ejecutarlo. Todos estos comandos se comportarán exactamente como lo hicieron antes de que el archivo fuera ejecutable.

Por otra parte hay que señalar que, los comandos sólo pueden ser ejecutados si la variable path contiene a el directorio actual de trabajo (representado por un punto '.') o si se proporciona el pathname⁸ completo.

Usando Variables dentro de los shell script.

Una variable permite "llenar el espacio en blanco" en tiempo de ejecución. Es posible usar variables para almacenar información interactivamente además de ser usadas para almacenar información dentro de los shell scripts. Hay dos formas para que una variable obtenga un valor:

- A través de un sentencia de asignación que es escrita dentro del shell script.
- A través de un sentencia de lectura en la cual el usuario proporciona el valor cuando el shell script es ejecutado.

Las sentencias de asignación las podemos escribir dentro de los shell scripts utilizando la sentencia set en el siguiente formato:

```
% set <nombre variable>=<valor>
```

Una asignación de este tipo busca un lugar de almacenamiento con la etiqueta <nombre variable> y almacena ahí el valor <valor> especificado.

Los nombres de variables deben empezar con una letra y contener sólo letras, números, y subguiones. Pueden ser de hasta 20 caracteres de largo. Por convención, a las variables se les dan nombres mnemónicos. Mnemónico significa que el nombre de la variable te dice algo acerca de lo que la variable contiene.

⁸ Pathname y ruta absoluta se refieren a lo mismo.

Consideremos un shell script que usa el comando `grep` para buscar ocurrencias de palabras mal escritas. En tal script, podemos encontrar el siguiente comando:

```
set palabra="teh"
```

el cual almacena el valor "teh" en una variable llamada `palabra`. Si después en el shell script, aparece el comando:

```
set palabra="tej"
```

el contenido de la variable `palabra` será reemplazado por el valor "tej". Las comillas alrededor de "teh" y "tej" indican que el valor de adentro debe ser tomado literalmente, incluyendo cualquier carácter de espacio en blanco. Una cadena de caracteres encerrada entre comillas es conocida como una cadena literal.

Suponiendo que quisiéramos borrar una variable de su lugar de almacenamiento podemos usar el comando `unset`:

```
unset <nombre variable>
```

Además de almacenar información como el valor de una variable, se debe poder recuperar esa información. Para referirnos al valor de una variable, debemos poner el signo (\$) inmediatamente antes del nombre de la variable (por ejemplo, `$palabra`). Para imprimir el valor de una variable escribimos:

```
echo $<nombre variable>
```

El siguiente ejemplo asigna a la variable 'mensaje' la cadena de caracteres "Seleccione una opción", después, visualiza el contenido de la variable 'mensaje' y después desasigna esta variable eliminándola. Por último verifica que la variable ya no existe.

```
% set mensaje="Seleccione una opción."  
% echo $mensaje  
Seleccione una opción.  
% unset mensaje  
% echo $mensaje  
%
```

Si omitimos el signo \$, sólo el nombre de la variable será impreso:

```
% echo mensaje  
mensaje
```

Se puede imprimir a la vez el valor de más de una variable en un enunciado echo. Se puede inclusive combinar variables y texto:

```
% set primero=1
% set segundo=2
% echo "El valor de la primera variable es $primero"
El valor de la primera variable es 1
%
% echo "Los valores de la primera y segunda variables";\
echo "son $primero y $segundo respectivamente."
Los valores de la primera y segunda variables
son 1 y 2 respectivamente.
```

Hay que notar que en el ejemplo anterior, las comillas permiten que los valores de variables sean substituidos dentro de cadenas literales. Los apóstrofes no permiten esta substitución.

El <valor> en una sentencia de asignación no tiene que ser una cadena literal. Puede ser también el valor de otra variable, la salida de un comando, o cualquier combinación de estos elementos. Por ejemplo el siguiente comando:

```
% set prompt="'pwd' $user > "
```

asigna a la variable prompt el resultado de la ejecución del comando *pwd* (print work directory), así como el valor de la variable *\$user* además de un símbolo 'mayor que'.

Otra forma de asignar un valor a una variable es tener al usuario proporcionando el valor. Este tipo de asignación es conocido como una sentencia read (read statement). La sintaxis de esta sentencia es la siguiente:

```
% set <nombre variable>=$<
```

Cuando el shell encuentra una sentencia read espera hasta que escribamos el final de una línea con un RETURN, antes de continuar interpretando el shell script.

UNIX

El siguiente es un ejemplo de la implementación del comando ADD el cual nos permite agregar información a una agenda telefónica:

```
#!/bin/csh
# 'Archivo: ADD -- Agrega información a la mini agenda telefónica'
#
# 'Este comando es usado para agregar información a la'
# 'agenda telefónica contenida en el archivo: datafile. Cada'
# 'registro consiste de un nombre y apellido, seguido'
# 'de una fecha de nacimiento y un numero telefónico.'
# 'El usuario que corra este archivo de comandos le será'
# 'preguntada toda la información necesaria'
#
echo -n "Cual es el nombre de la persona a agregar? "
set nombre=$<
echo -n "Cual es el apellido de $nombre? "
set apellido=$<
echo "Teclea la fecha de nacimiento en el formato mes/día/año."
set dian=$<
echo -n "Cual es el teléfono de $nombre $apellido: "
set telefono=$<
echo "$nombre $apellido<TAB><TAB>$dian<TAB><TAB>$telefono"\
>> $HOME/datafile
sort +1 -2 $HOME/datafile > $HOME/temp
mv $HOME/temp $HOME/datafile
echo "El siguiente registro ha sido agregado a tu base de datos:"
echo "$nombre $apellido<TAB><TAB>$dian<TAB><TAB>$telefono"
```

NOTA: Dentro algunas sentencias echo en el ejemplo anterior y en algunos posteriores, aparece la secuencia de caracteres '<TAB>', estos deberán de substituirse por un tabulador en lugar de escribir literalmente esta cadena de caracteres.

Diferencias entre ' , " y `.

Estas marcas preservan los espacios en blanco en un comando echo y en expresiones regulares para grep. También previenen que el shell interprete metacaracteres de nombres de archivos.

Adicionalmente a los apóstrofes y a las comillas, hay un tercer tipo de marca que debemos conocer y que es interpretado de manera diferente que los otros dos. El tercer tipo es un apóstrofo al revés o acento grave (`). Es usado para forzar que el shell interprete un comando como en los siguientes dos ejemplos:

```
% echo "La fecha y hora actual es `date`."
La fecha y hora actual es Mon May 14 14:55:39 EDT 1993.
%
% set hoy=`date`
% echo "La fecha y hora actual es $hoy."
La fecha y hora actual es Mon May 14 14:55:39 EDT 1993.
```

Los efectos de los tres tipos de marcas se explican a continuación:

- Los apóstrofes (') protegen todo. Cualquier cosa encerrada en apóstrofes es tomada literalmente.
- Las comillas (") protegen espacios en blanco, metacaracteres de nombres de archivos, y apóstrofes. No protegen variables (precedidas por \$) o comandos encerrados por apóstrofes al revés (`). El signo de pesos y los apóstrofes al revés son siempre interpretados.
- Los acentos graves le dicen al shell que ejecute el comando encerrado adentro. Todo lo que se encuentra adentro de ellos es interpretado tal y como si fuera tecleado interactivamente.

Usando Archivos Temporales.

Cuando se usan archivos temporales hay que tener cuidado de que no exista un archivo con el mismo nombre que el del temporal en el directorio de trabajo actual. Si existe, el shell script lo borrará. Además, si a otro usuario se le ocurre crear un archivo temporal, muy probablemente use también el nombre temp, con lo cual podría borrar nuestra información si es que se usa el mismo subdirectorio para escribir este tipo de archivos temporales. Obviamente esto no es lo que deseamos.

Afortunadamente hay alternativas para crear un archivo temporal en el directorio actual o en algún otro subdirectorio. El directorio de sistema /tmp existe solamente con el propósito de proporcionar espacio para archivos temporales, ya sean estos de comandos, de información o de lo que sea.

Hay que asegurarnos que cuando algún comando cree un archivo temporal, el nombre que le demos a ese archivo sea único. Se puede crear un nombre de archivo único agregándole el PID (Process Identification) del comando al nombre del archivo. La variable de shell \$\$ contiene el valor del número de identificación del proceso en ejecución. Si le damos a un archivo creado dentro de un shell script el nombre \$0\$\$, el nombre del comando (el nombre del archivo que contiene al script) es substituido por \$0, y el PID es substituido por \$\$\$. Si se recibe un mensaje de error que diga "Variable syntax", deberemos llamar al archivo \${0}\$\$\$.

El siguiente ejemplo es una versión modificada del archivo de comandos ADD, donde hacemos uso de lo que se acaba de señalar. Esta nueva versión de ADD también verifica si la persona a dar de alta ya existe dentro del archivo \$HOME/datafile para evitar duplicar información:

```
#!/bin/csh
# 'Archivo: ADD -- Agrega información a la mini agenda telefónica'
#
# 'Este comando es usado para agregar información a la'
# 'mini agenda telefónica contenida en el archivo: datafile. Cada'
# 'registro consiste de un nombre y apellido, seguido'
# 'de una fecha de nacimiento y un número telefónico.'
# 'El usuario que corra este archivo de comandos le será'
# 'preguntada toda la información necesaria'
#
clear
echo -n "Cual es el nombre de la persona a agregar? "
set nombre=$<
echo -n "Cual es el apellido de $nombre ? "
set apellido=$<
if { grep -sq "$nombre $apellido" $HOME/datafile } then
    echo "$nombre $apellido ya se encuentra en su agenda."
else
    echo "Teclea la fecha de nacimiento en el formato mes/día/año."
    set dian=$<
    echo -n "Cual es el teléfono de $nombre $apellido: "
    set telefono=$<
    echo "$nombre $apellido<TAB><TAB>$dian<TAB><TAB>$telefono" \
        >> $HOME/datafile
    sort +1 -2 $HOME/datafile > /tmp/${0}$$
    mv /tmp/${0}$$ $HOME/datafile
    echo "El siguiente registro ha sido agregado a tu base de \
datos:"
    echo "$nombre $apellido<TAB><TAB>$dian<TAB><TAB>$telefono"
endif
```

Observemos que ahora el archivo temporal se mando al subdirectorio /tmp bajo un nombre que será único (\${0}\$\$).

Control de Flujo.

Alguna vez , quizás deseemos que la computadora ejecute algunos comandos sólo bajo condiciones particulares o para repetir ciertos comandos. Para tener a la computadora ejecutando comandos en orden no-secuencial o no-lineal, se necesita alterar el flujo de control del shell script. Las construcciones de control de flujo son comandos especiales que se pueden usar en los shell scripts (o en cualquier programa de computadora) para hacer que los comandos sean ejecutados condicionalmente o repetidamente.

Uno de los usos más importantes de la ejecución condicional en programas de computadora es el checar errores. Usando sentencias condicionales, los programas pueden checar una entrada de usuario para detectar diferentes tipos de errores y mandar un mensaje de aviso adecuado al usuario.

Los símbolos `&&` y `||` permiten ejecutar un segundo comando si el comando previo tiene éxito o falla respectivamente. Por ejemplo:

```
% echo "hola" && echo "adiós"
hola
adiós
```

```
% echo "hola" || echo "adiós"
hola
%
```

En el primer comando, le decimos al intérprete de comandos que ejecute el segundo comando (`echo "adiós"`), solo si el primer comando (`echo "hola"`) tiene éxito; por supuesto esta primer sentencia `echo` siempre tiene éxito y la segunda sentencia `echo` es ejecutada. En el segundo caso (`||`) le decimos que ejecute el segundo comando (`echo "adiós"`) solo si el primer comando (`echo "hola"`) falla. Como el primer comando `echo` no falla, el segundo comando `echo` no es ejecutado.

Pero muchas veces es necesario poder mandar ejecutar un grupo de instrucciones en caso de éxito o falla de algún comando, no solo un solo comando como en el caso anterior.

Consideremos el script de comandos DELETE que se muestra en el siguiente ejemplo, el cual tiene la finalidad de borrar un registro de nuestra agenda telefónica. Idealmente, se quiere borrar el nombre sólo si está en la base de datos e imprimir un mensaje si no está. Esta revisión se realiza mediante la estructura if-then-else, en caso de tener éxito el comando que se encuentra entre llaves {}, se ejecutarán las instrucciones grep, mv y echo; en caso de falla, se ejecutará solamente el comando echo que se encuentra después del else.

```
#!/bin/csh
# 'Archivo: DELETE -- BORRA un registro de la agenda telefónica.'
#
echo -n "Cual es el nombre de la persona a borrar? "
set nombre=$<
echo -n "Cual es el apellido de la persona a borrar? "
set apellido=$<
if { grep -sq "$nombre $apellido" $HOME/datafile } then
    grep -v "$nombre $apellido" $HOME/datafile > temp
    mv temp $HOME/datafile
    echo "$nombre $apellido ha sido borrado(a) de tu base de datos."
else
    echo "$nombre $apellido no se encuentra en su agenda."
endif
```

La estructura de control if/then permite checar que SI (IF) el primer comando tiene éxito ENTONCES (THEN) ejecute los siguientes comandos. El final de la lista de comandos es marcado con un terminador. El terminador es endif en el caso del C shell. La sintaxis es la siguiente:

```
if {<comando>} then
    <lista de comandos>
endif
```

NOTA: La colocación de la palabra then es muy importante. En C shell then debe ser parte de la misma línea del if.

La sintaxis de la construcción if/then/else es:

```
if {<comando>} then
    <lista de comandos>
else
    <lista de comandos>
endif
```

Las estructuras de control if/then e if/then/else nos permite escribir shell scripts que efectúan otros tipos de decisión. El formato más general para una sentencia condicional es:

```
if {<condición>} then
  <lista de comandos>
else
  <lista de comandos>
endif
```

La <condición> puede ser de varios tipos:

- Una prueba numérica del valor de una variable: Es \$#argv más grande que 2?
- Una prueba del valor de la cadena de una variable: Contiene \$confirma la cadena 'si'?
- Una prueba del tipo de un archivo: Es el archivo practica un directorio?
- Una prueba para ver si un archivo o variable existe?

La <condición> puede ser aún una combinación de dos o más pruebas condicionales. Si la condición se evalúa como cierta, la primera acción (uno o más comandos) después de la palabra then es realizada. Si la condición se evalúa como falsa, la acción que sigue a la palabra else es realizada. La cláusula else (la palabra else y la acción asociada) es opcional.

Las expresiones pueden ser usadas para efectuar varios tipos de pruebas y pueden tomar cualquiera de la siguientes formas:

- Para probar valores numéricos:
Por ejemplo, num1 <operador> num2. El <operador> puede ser cualquiera de los siguientes, como en (`$#argv == 0`):

OPERADOR	FUNCIÓN
<code>==</code>	igual a
<code>!=</code>	diferente a
<code><</code>	menor que
<code><=</code>	menor o igual que
<code>></code>	mayor que
<code>>=</code>	mayor o igual que

- Para probar valores de cadenas:

EXPRESIÓN	FUNCIÓN
<code>cadena1 == cadena2</code>	Verdadero si las dos cadenas son iguales
<code>cadena1 != cadena2</code>	Verdadero si las dos cadenas son diferentes
<code>cadena1 =~ cadena2</code>	Verdadero si la cadena1 concuerda el patrón de metacaracteres especificado por cadena2, como en (<code>"dos palabras" =~ *p*</code>)
<code>cadena1 !~ cadena2</code>	Verdadero si cadena1 no concuerda con el patrón de metacarateres especificado por cadena2, como en (<code>"letras !~ ???"</code>)

UNIX

- Para probar permisos, existencia, y tipo de un archivo como en (-d \$archivo) para probar si \$archivo es un directorio. También:

EXPRESION	FUNCION
-r nombrearchivo	Verdadero si el archivo tiene permisos de lectura.
-w nombrearchivo	Verdadero si el archivo tiene permisos de escritura.
-x nombrearchivo	Verdadero si el archivo tiene permisos de ejecución.
-e nombrearchivo	Verdadero si el archivo existe.
-o nombrearchivo	Verdadero si el usuario es el propietario del archivo.
-z nombrearchivo	Verdadero si el archivo está vacío (el tamaño es cero).
-f nombrearchivo	Verdadero si el archivo no es un directorio.
-d nombrearchivo	Verdadero si el archivo es un directorio.

Un lugar donde quizás se quieran realizar pruebas numéricas es adentro de cualquiera de los comandos de la mini agenda telefónica. Para hacer a los comandos más amigables al usuario, se le puede dar al usuario la opción de teclear el nombre y apellido como argumentos en la línea de comandos, o teclear simplemente el comando, en cuyo caso, se le pedirán posteriormente estos datos. Esto se ejemplifica a continuación:

```
#!/bin/csh
# 'Archivo: LOOKUP -- BUSCAR información en la agenda telefónica'
#
# 'Esta versión de LOOKUP le pregunta al usuario el nombre'
# 'de la persona a buscar si no se incluyo esta información'
# 'en la línea de comandos.'
#
clear
```

```

if ( $#argv < 1 ) then
    echo -n "Cual es el nombre de la persona a buscar? "
    set nombre=$<
    echo -n "Cual es el apellido de la persona a buscar? "
    set apellido=$<
    set nombrec="$nombre $apellido"
else
    set nombrec="$1 $2";           # 're-asignar argumentos de'
                                # 'la línea de comandos'
endif
grep "$nombrec" $HOME/datafile
if ( $status != 0 ) then        # 'si la persona no fue encontrada'
    echo "$nombrec no esta en tu base de datos."
endif

```

Observemos que el segundo if del ejemplo anterior hace uso de la variable \$status. A esta variable se le asigna el valor de 0 (cero) si el comando inmediato anterior se ejecuto satisfactoriamente. En caso contrario se le asigna un valor numérico asociado al tipo de error que haya presentado el comando durante su ejecución.

Las comparaciones son también frecuentemente usadas para probar una respuesta de usuario a una pregunta en shell scripts interactivos. El siguiente shell script ilustra este uso así como también muestra una prueba para verificar la existencia de un archivo:

```

#!/bin/csh
# 'Archivo: previ -- preguntar al usuario por archivo de backup y'
#                               'entra al editor vi'
#
# 'Este comando checa si el archivo a editar ya existe.'
# 'Si si existe, el usuario tiene la opción de crear una'
# 'copia de backup del archivo antes de entrar al editor.'
#
if (-e $1) then                # 'si el archivo existe'
    echo -n "Quieres crear una copia de $1 (s/n)? "
    set respuesta=$<
    if ($respuesta == 's') then
        cp $1 "$1.bak"; # 'Crea el backup'
        echo "Una copia de $1 ha sido salvada en $1.bak."
    endif
endif
/usr/bin/vi $1

```

Sentencias if Anidadas.

Las sentencias if anidadas pueden ser confusas cuando son involucradas cláusulas else. Una regla importante que debemos recordar es que una cláusula else es siempre relacionada a la sentencia if más reciente.

Sentencias if/then/else Anidadas.

Frecuentemente es necesario especificar varias opciones en lugar de sólo una. En shell scripts interactivos, por ejemplo, se pueden usar varias sentencias if/then/else para checar la entrada y así detectar diferentes tipos de error. Cada condición es realmente usada para asociar una acción con un tipo específico de entrada. Si alguna de estas condiciones es VERDADERA, ninguna de las restantes es probada.

El siguiente ejemplo muestra la estructura anidada para un genérico fragmento de código de chequeo de errores:

```
if (<condición error 1>) then
  echo <mensaje error 1>
else
  if (<condición error 2>) then
    echo <mensaje error 2>
  else
    if (<condición error 3>) then
      echo <mensaje error 3>
    else
      <entrada normal -- procesarla>
    endif
  endif
endif
```

Combinando Pruebas Condicionales.

Se puede lógicamente combinar pruebas condicionales así como efectuarlas separadamente. Hay tres tipos de operaciones lógicas: AND (&&), OR (||), y NOT (!). Sus efectos son resumidos en la tabla siguiente:

Operaciones Lógicas

OPERACIÓN	SINTAXIS	RESULTADO
NOT	!<expr>	Lo contrario del valor de <expr>. Si <expr> se evalúa como VERDADERO, !<expr> se evalúa como FALSO
OR	<expr1> <expr2>	Se evalúa como VERDADERO si una o ambas expresiones se evalúan como VERDADERO
AND	<expr1> && <expr2>	Se evalúa como VERDADERO sólo si ambas expresiones se evalúan como VERDADERO

Precedencia de Operaciones Lógicas.

El operador AND (&&) tiene precedencia sobre el operador OR (||). El orden de las operaciones en la evaluación de sentencias condicionales es muy similar al orden de las operaciones en aritmética. El operador AND actúa como la multiplicación, y el operador OR actúa como la adición. Se pueden inclusive usar paréntesis para cambiar el orden de evaluación.

Ciclos (looping).

Como la mayoría de los lenguajes de programación, el shell también proporciona construcciones de flujo de control llamadas loops que nos permiten repetir secciones de código.

¿Qué es un loop?

Hay dos tipos básicos de loops. El primer tipo es conocido como un loop iterativo. Este se ejecuta un número específico de veces. Los loops iterativos tienen una condición de salida ya construida. Esto significa que un loop que se ejecuta un número específico de veces termina tan pronto como ese número es alcanzado.

El segundo tipo de loop es un loop condicional. Este tipo de loop se ejecuta mientras la condición de control del loop es verdadera. Si la condición nunca es verdadera, no se ejecuta el loop. Tan pronto como la condición es falsa, el loop se deja de ejecutar. Si la condición nunca es falsa, el loop se continúa ejecutando hasta que "fuerzas externas" interfieren (por ejemplo, hasta que alguien "mata" o elimina el proceso en cuestión).

Loops Iterativos.

Para implementar un loop iterativo, se necesita la siguiente construcción:

```
foreach <variable-control-loop> ( <lista> )
    <lista de comandos>
end
```

Para la mayoría de los comandos que se escriban, quizás se quiera usar la lista de argumentos del comando para substituirlos por <lista>. Esta lista de argumentos es representada por `$argv[*]` en el C shell. Si se desea usar la lista de argumentos del comando, se debe usar explícitamente:

```
foreach <variable> ( $argv[*] )
```

El siguiente shell script permite teclear `lf` en lugar de `ls -F` para obtener una lista de archivos con directorios y archivos ejecutables específicamente marcados:

```
#!/bin/csh
# 'Archivo: lf -- lista archivos como la opción -F de ls'
#
# 'Este comando hace casi lo mismo que la opción -F'
# 'del comando ls. Su salida es una lista de los archivos'
```

```

# 'contenidos en los subdirectorios especificados en su línea'
# 'de argumentos.'
#
# 'Si una archivo es un subdirectorio, una diagonal (/) es'
# 'agregada al nombre de este, si es un archivo ejecutable,'
# 'entonces un asterisco (*) es agregado al nombre del archivo'
# 'Si no se dan argumentos, el directorio actual es listado.'
#
unset noclobber
if ( $#argv == 0 ) then # 'lista el directorio actual'
                        # 'si no hay argumentos'
    ls -F
else
    foreach file ($argv[*])
        if (-f $file) then
            ls -F $file >> /tmp/${0}$$
        else
            echo "directorio: $file"
            ls -F $file
            echo " "
        endif
    end
    if (-e /tmp/${0}$$) then
        cat /tmp/${0}$$
        /bin/rm /tmp/${0}$$
    endif
endif

```

NOTA: Cuando la variable `noclobber` está definida, se recibe un mensaje de error si se trata de agregar una salida a un archivo que no existe usando redireccionamiento (`>>`). Si se coloca el comando `unset noclobber` al principio del script, este problema puede ser evitado.

Loops Condicionales.

Los loops condicionales son implementados usando sentencias `while`. La sintaxis es la siguiente:

```

while ( <condición> )
    <lista de comandos>
end

```

UNIX

La condición es probada en la parte superior del loop. Si es verdadera, la lista de comandos (conocida como cuerpo del loop) es ejecutada. Típicamente, uno de los comandos en el cuerpo del loop cambiará el valor de prueba. El loop se continúa ejecutando mientras la condición sea verdadera.

Si se entra a un loop y ninguna de las sentencias dentro del loop cambia el resultado de la prueba de condición, el loop se continúa ejecutando infinitamente.

El siguiente script puede ser usado para implementar el comando CREATE.

NOTA: Antes de que este comando sea usado, el comando ADD debe ser modificado par checar la existencia de la base de datos.

```
#!/bin/csh
# 'Archivo: CREATE -- permite al usuario crear una'
# 'mini agenda telefónica'
#
clear
echo "Este comando permite crear una mini agenda telefónica."
echo "Toda la información necesaria le será pedida."
echo -n "Quieres crear una agenda telefónica? (s/n) ? "
set respuesta=$<
while ( $respuesta == 's' )
  ADD
  echo -n "Quieres agregar a alguien mas? (s/n) ? "
  set respuesta=$<
end
```

Otro uso común de los loops condicionales son los programas conducidos por menús. Este tipo de programas le dan al usuario una lista de opciones de la cual puede elegir. La elección es evaluada, el comando es ejecutado, y al usuario se le pide que seleccione otra opción. Cuando se escribe un programa de este tipo, hay que asegurar incluir una opción de ayuda (help) y una opción de salir (quit). La opción de ayuda vuelve a desplegar el menú en caso de que el usuario olvide los comandos. La opción de salir le permite al usuario salir del programa.

El siguiente shell script combina todas las operaciones para la mini agenda telefónica. Adicionalmente a los comandos ADD, DELETE, LOOKUP, y UPDATE, otras opciones han sido agregadas para permitirle al usuario ver el MENU de comandos, y salir (QUIT) del programa.

NOTA: El primer comando en el shell script es set noglob. Este comando es necesario para prevenir que el signo de interrogación sea interpretado como un metacaracter.

```
#!/bin/csh
# 'Archivo: info -- implementa la mini agenda telefónica'
# '
#           basada en un menú.'
#
clear
set noglob
echo -n "Bienvenido al programa de información de la "
echo "mini agenda telefónica."
set comando='?'
while ($comando != 'q' && $comando != 'Q')
  if ($comando == 'a' || $comando == 'A') then
    ADD
  else if ($comando == 'd' || $comando == 'D') then
    DELETE
  else if ($comando == 'l' || $comando == 'L') then
    LOOKUP
  else if ($comando == 'u' || $comando == 'U') then
    UPDATE
  else if ($comando == 'p' || $comando == 'P') then
    PRINT
  else if ($comando == '?') then
    MENU
  else
    echo "$comando no es un comando valido."
    echo "Teclea ? para ver una lista de comandos
disponibles."
  endif
echo -n "Introduce un comando (? para ver una lista de opciones):"
set comando=$<
end
```

```
#!/bin/csh
# 'Archivo: MENU -- imprime un menú de los comandos disponibles'
#
echo "Para ejecutar cualquier comando, teclea la primera letra del"
echo "nombre del comando. Para ver la lista de comandos, teclea"
echo "un signo de interrogación (?)."
```

UNIX

```
echo " "
echo "ADD      - agregar un nombre a la agenda telefónica"
echo "DELETE   - borrar un nombre de la agenda telefónica"
echo "LOOKUP   - buscar un nombre en la agenda telefónica"
echo "UPDATE   - actualizar información en la agenda telefónica"
echo "PRINT    - imprimir un listado de la agenda telefónica"
echo "QUIT     - salir del programa info"
echo " "
```

```
#!/bin/csh
# 'Archivo: UPDATE -- Actualizar la información en la'
# 'mini agenda telefónica'
#
# 'Este comando es usado para actualizar un registro en la'
# 'mini agenda telefónica contenida en el archivo datafile.'
# 'Cada registro consiste de un nombre y un apellido, seguido'
# 'de una fecha de nacimiento y de un número telefónico.'
# 'El usuario que corra este archivo de comandos le será'
# 'preguntada toda la información necesaria'
#
if ($#argv != 2) then
    echo -n "Cual es el nombre de la persona a actualizar? "
    set nombre=$<
    echo -n "Cual es el apellido de $nombre? "
    set apellido=$<
    set nombrec="$nombre $apellido"
else
    set nombrec="$1 $2"
endif
if { grep -sq "$nombrec" $HOME/datafile } then
    echo "El siguiente registro esta siendo actualizado:"
    grep "$nombrec" $HOME/datafile
    echo -n "Estas seguro de que lo quieres cambiar (s/n)? "
    set confirma=$<
    if ($confirma == 's') then
        grep -v "$nombrec" $HOME/datafile > /tmp/${0}$$
        echo "Teclea la nueva fecha de nacimiento (mes/día/año)"
        set dian=$<
        echo "Teclea el nuevo numero telefónico de $nombrec."
        set teléfono=$<
        echo "$nombrec<TAB><TAB>$dian<TAB><TAB>$teléfono" \
>> /tmp/${0}$$
        sort +1 -2 /tmp/${0}$$ > $HOME/datafile
        echo "El nuevo registro es:"
        echo "$nombrec<TAB><TAB>$dian<TAB><TAB>$teléfono"
    else
        echo "Tu base de datos no ha sido cambiada."
```

```
endif
else
  echo "$nombrec no esta en tu base_de_datos."
  echo "Ejecuta el comando ADD para agregar a $nombrec."
endif
```

Accesando la Lista de Argumentos.

La mayoría de los comandos se utilizan con opciones o argumentos. Una de las funciones del shell es analizar gramaticalmente cada línea de comando o dividirla en sus componentes. Estas partes incluyen al comando, sus argumentos, y apuntadores de redirección así como sus archivos correspondientes. La primera cosa que el shell hace cuando interpreta un comando es estudiarlo. La redirección es tomada en cuenta en este momento. Después, las demás palabras en la línea de comandos son asignadas a variables con nombres numéricos, comenzando con cero para el comando mismo. Así que la línea de comando:

```
% grep -n "Miguel" archivodatos personal amigos > miguel.info
```

es dividida de la siguiente forma:

```
$0 = grep      $3 = archivodatos
$1 = -n        $4 = personal
$2 = Miguel    $5 = amigos
```

Estas variables son conocidas como parámetros posicionales porque sus nombres son derivados de sus posiciones en la línea de comandos. Estas le permiten a un shell script aceptar entrada en la línea de comandos. Sus valores son asignados por el shell cuando un comando es ejecutado, y pueden ser referenciados en el código del comando. Hay que notar que el apuntador de redirección así como el nombre de archivo asociado con él, no son parámetros posicionales.

Puede haber veces que se necesiten referir todos los argumentos del comando. En este caso hay que usar la variable `$argv[*]` (en el C shell). En el ejemplo anterior, el valor de `$argv[*]` es:

```
"-n Miguel archivodatos personal amigos"
```

UNIX

La variable `$#argv`, se refiere al número de parámetros posicionales (sin incluir el comando). En el ejemplo anterior, `$#argv` es 5.

Hay algunas limitaciones en el uso de parámetros posicionales. Sólo se puede hacer referencia a parámetros posicionales con nombres de dígitos sencillos (`$1 - $9`). Usualmente no se necesitan más de nueve parámetros posicionales, pero en caso de que se necesiten más, se pueden acceder. En C shell, se tiene que usar el comando *shift* para acceder a aquellos parámetros que su referencia sea mayor a la novena posición. El efecto del comando *shift* es el de desplazar la lista de parámetros un lugar hacia la izquierda, de manera que después de haber usado *shift*, el parámetro `$1` es ahora substituido por el valor de `$2`, `$2` es substituido por `$3` y así sucesivamente. Si se volviese a hacer uso de *shift*, ahora `$1` que fue substituido por `$2` sería substituido por `$3`, `$2` que fue substituido por `$3` sería substituido por `$4` y así como parámetros haya.

Los siguientes ejemplos consisten de un shell script llamado `show`, el cual imprime los valores de hasta seis parámetros posicionales:

```
% cat show
# /bin/csh
# 'Archivo: show -- muestra numeros y valores de parámetros'
# 'posicionales'
#
echo "El comando ("'$0'") es $0."
echo "El numero de parámetros ("'$#argv'") es $#argv"
echo '$1 es' $1
echo '$2 es' $2
echo '$3 es' $3
echo '$4 es' $4
echo '$5 es' $5
echo '$6 es' $6
%
% show un ejemplo con exactamente seis parámetros
El comando ($0) es show.
El numero de parámetros ($#argv) es 6
$1 es un
$2 es ejemplo
$3 es con
$4 es exactamente
$5 es seis
$6 es parámetros
%
```

```

% show un ejemplo corto
El comando ($0) es show.
El numero de parámetros ($#argv) es 3
$1 es un
$2 es ejemplo
$3 es corto
$4 es
$5 es
$6 es
%
% show Cuando hay mas de seis parámetros los últimos \
son ignorados.
El comando ($0) es show.
El numero de parámetros ($#argv) es 10
$1 es Cuando
$2 es hay
$3 es mas
$4 es de
$5 es seis
$6 es parámetros
%
% show "Cualquier cosa entre comillas es" 'un solo parámetro'
El comando ($0) es show.
El numero de parámetros ($#argv) es 2
$1 es Cualquier cosa entre comillas es
$2 es un solo parámetro
$3 es
$4 es
$5 es
$6 es

```

Si modificamos ahora un poco el comando show para que haga desplazamientos de variables utilizando shift quedaría algo parecido a lo siguiente:

```

# /bin/csh
# 'Archivo: show -- muestra numeros y valores de parámetros'
# 'posicionales'
#
echo "El comando ("'$0'" ) es $0."
echo "El numero de parámetros ("'$#argv'" ) es $#argv"
echo '$1 es' $1
echo '$2 es' $2
echo '$3 es' $3
shift; shift;          # 'Se hacen dos desplazamientos'
echo '$4 es' $4
echo '$5 es' $5
echo '$6 es' $6

```

```
%  
% show un ejemplo con más de seis parámetros  
El comando ($0) es show.  
El numero de parámetros ($#argv) es 7  
$1 es un  
$2 es ejemplo  
$3 es con  
$4 es seis  
$5 es parámetros  
$6 es
```

Usando Variables de Ambiente.

Cuando se crean variables de ambiente se les puede dar casi cualquier nombre que se quiera. Hay algunos nombres, sin embargo, que representan variables de shell predefinidas. Estas variables son usadas por el shell para controlar el ambiente de trabajo. Un ejemplo de una variable de ambiente es el prompt del shell. El prompt en C shell está almacenado en la variable "prompt". Se puede cambiar el prompt, con un reset del valor de la variable apropiada:

```
% set prompt="<HOLA> "  
<HOLA>
```

Hay que notar el espacio después de <HOLA>. Este espacio fue añadido para que al usuario le sea más fácil distinguir los comandos que teclee desde el prompt.

La siguiente tabla muestra algunas de las más importantes variables de ambiente en el shell. Así como se puede desplegar el valor de una variable utilizando sentencias echo. También se puede usar el comando set sin argumentos para desplegar los valores de TODAS las variables que están definidas.

VARIABLES DE AMBIENTE EN EL SHELL

NOMBRE VARIABLE	DESCRIPCION
cwd	Directorio actual de trabajo. Es definida por el shell cuando se ejecuta un comando cd.
history	Número de comandos a salvar en la lista de historia (history). En el C shell, también es el número a desplegar.
home	Directorio de entrada del usuario (home directory).
mail	Especifica el nombre del archivo que contiene el correo del usuario. Si la variable está definida, el shell checa aproximadamente cada 10 minutos si el archivo ha sido modificado desde la última vez que fue leído. Si es así, el mensaje "You have new mail" es impreso.
noclobber	Si la variable está definida, el shell no permite sobrescribir archivos cuando se redirecciona la entrada usando el apuntador de redirección >. También previene crear archivos usando el apuntador de redirección >>. En cualquier caso se recibe un mensaje de error.
path	Contiene los nombres de los subdirectorios en donde se buscarán los comandos que se escriban desde la línea de comandos.
prompt	Valor del prompt primario del shell (el default es %, y # es usado para superusuario)
shell	Shell de entrada del usuario (intérprete de comandos). Por ejemplo: /usr/bin/ksh, /bin/csh, /bin/sh, o /usr/bin/sh5

Búsqueda de Comandos

Cuando se manda un comando para que el shell lo ejecute, se usa el valor de la variable PATH para encontrar el(los) directorio(s) donde el código para ese comando está localizado. Esto significa que cuando se le pasa un comando al shell, este busca si existe un archivo con ese nombre en el primer directorio listado en la variable PATH, si es así, lo ejecuta. Si no, busca en el siguiente directorio de la lista. Este proceso se repite hasta que se encuentra el comando o se termina la lista de subdirectorios. Si el comando nunca se encuentra, se recibe uno de estos dos mensajes de error:

```
<command name>: not found  
<command name>: command not found
```

Por convención, los directorios de comandos en Unix son usualmente llamados bin. El nombre viene de la palabra binary y se deriva del hecho que las versiones ejecutables de la mayoría de los comandos de sistema están almacenadas en formato binario. La mayoría de los comandos de sistema están escritos en el lenguaje de programación C; la versión binaria es el estado ejecutable. Dentro de la cultura de Unix, las palabras binario y ejecutable han llegado a ser casi sinónimos.

Los usuarios y programadores de Unix usualmente almacenan sus comandos en un subdirectorio de su directorio hogar llamado bin. Después, redefiniendo el valor de la variable path para incluir el directorio de comandos propio, estos comandos se pueden ejecutar desde otros directorios.

Una de las ventajas de establecer el path con estos subdirectorios, es que se pueden escribir comandos con los mismos nombres que los comandos de sistema. Debido a que el directorio bin propio puede preceder a los directorios de sistema en la variable path, la versión propia del comando es ejecutada en lugar de la versión del sistema.

Características Especiales de las Variables de Shell

Las variables que son creadas dentro de un shell script sólo existen mientras el shell script se ejecuta. Las variables que están dentro de un shell script son llamadas variables locales, porque sus valores son locales al shell script que las usa. Si se trata de hacer referencia a ellas interactivamente, o si otro shell script trata de usarlas, éstas no existirán.

El shell trata a la sesión de terminal como si fuera un shell script. Las variables que se definen durante la sesión de terminal, incluyendo a las variables de ambiente, sólo están disponibles cuando se escriben comandos interactivamente. No se pueden modificar dentro de los shell scripts que se escriban.

```
% cat setpr
#!/bin/csh
# 'Archivo: setpr -- Muestra el valor de la variable prompt,'
# 'reassigna su valor y muestra el nuevo valor de la variable.'
#
# echo "El valor actual de la variable prompt es $prompt"
# echo -n "Escriba el nuevo prompt: "
# set prompt=$<
# echo "El nuevo valor de prompt es $prompt"
%
% setpr
El valor actual de la variable prompt es %
Escriba el nuevo prompt: LAA>
El nuevo valor de prompt es LAA>
%
```

Compartiendo Valores de Variables.

Algunas veces los comandos necesitan hacer referencia a los valores de ciertas variables de ambiente. El editor vi es un buen ejemplo de un comando así. Si el valor de la variable TERM no está definida, vi no opera como un editor de pantalla-completa.

Para compartir el valor de una variable, un shell *exporta* la variable. En C shell se usa el comando *setenv*. El valor de cualquier variable exportada es compartido por todos los procesos hijos del shell. Para exportar una variable, se debe usar la siguiente sintaxis:

```
% setenv <nombre variable> <valor>
```

En C shell, también hay un comando llamado *printenv* que imprime los valores de todas las variables que fueron definidas con el comando *setenv*. Adicionalmente, el comando *printenv* imprime algunas variables de ambiente que el C shell *exporta* automáticamente.

Se pueden exportar variables sólo al shell actual y a los "hijos" de ese shell. Ningún shell "padre" del shell actual se enterará de las variables que se exportaron.

Cabe señalar que cuando se encuentra un error en un shell script, éste se deja de ejecutar inmediatamente.

Usando Operaciones Aritméticas con Variables de Shell.

Adicionalmente a sus muchas otras capacidades, el shell puede efectuar aritmética de enteros simple. La aritmética puede ser usada en muchas formas diferentes dentro de los shell scripts. Uno de los usos más importantes es el conteo, el cual es frecuentemente usado para controlar la ejecución de los loops.

En C shell, el símbolo "at" (@) significa matemática. La siguiente tabla muestra como asignar los resultados de operaciones aritméticas a una variable llamada *numero*.

Operaciones Matemáticas

OPERACIÓN	ASIGNACIÓN VARIABLE
suma (a+b)	@ numero = a + b
resta (a-b)	@ numero = a - b
multiplicación (axb)	@ numero = a * b
división entera (parte entera de a/b)	@ numero = a / b
residuo de a/b	@ numero = a % b

El shell reconoce sólo enteros no negativos entre 0 y 32767. Si se trata de introducir un número negativo o un número con punto decimal en un shell script interactivo, se recibirá un mensaje de error. Los números más grandes que 32767 no son interpretados correctamente.

El siguiente ejemplo muestra el shell script `countdown` como un ejemplo de como la aritmética puede ser usada para controlar la ejecución de un loop:

```
% cat countdown
# 'Archivo: countdown -- cuenta en reversa del 10 al 0'
#
set contador=10
while ($contador >= 0)
    echo "          $contador"
    @ contador=$contador - 1
end
echo "* F U E G O *"
%
% countdown
          10
          9
          8
          7
          6
          5
          4
          3
          2
          1
          0
* F U E G O *
%
```

Depurando los shell scripts

Si se encuentra un error con los datos de prueba probablemente se quiera efectuar una depuración ("debug") del programa siguiendo visualmente su ejecución línea por línea. Para seguir la ejecución de los shell scripts, se debe incluir el comando `set verbose` dentro

de los scripts que provoque que el shell imprima cada línea del script según la ejecute.

El método más rápido de seguir la ejecución de un shell script es correrlo ya sea con la opción `-v` o `-x` del comando `csh`. Por ejemplo:

```
% csh -vx nombreadarchivo
```

La opción `-v` imprime las líneas de entrada del shell según son leídas, mientras que la opción `-x` imprime los comandos y sus argumentos según son ejecutados.

Substituyendo Comandos.

Como ya se mencionó, se puede asignar la salida estándar de un comando a una variable con substitución de comandos. El formato general para la substitución de comandos es:

```
% set <variable>=<'comando'>
```

donde `<'comando'>` es un comando que escribe a la salida estándar.

Se puede usar substitución de comandos para evitar volver a teclear largas listas de palabras. Por ejemplo, quizás se quiera asignar los nombres de los archivos que contengan la cadena "include" a la variable `incarchivos`. El comando `grep` tiene una opción `-l` que imprime los nombres de los archivos que contienen palabras que concuerdan con la cadena dada en la línea de comandos.

```
% set incarchivos=$(grep -l 'include' *)
```

La variable `incarchivos` entonces contiene la lista de aquellos archivos que contienen la cadena de caracteres "include":

```
% echo $incarchivos  
dos1.c doc2.c fill.c h.c list.c large.c
```

En este momento se puede usar la variable `incarchivos` como entrada a otros comandos. Después, se puede ver el contenido de los archivos pantalla por pantalla, copiarlos a un directorio de backup, y mandarlos a la impresora:

```
% more $incarchivos  
% cp $incarchivos /users/cecafi/amezcua/backup  
% lp $incarchivos
```

La substitución de comandos no está limitada a la asignación de variables. Se puede usar la substitución de comandos para substituir argumentos en la línea de comandos. Por ejemplo, el comando cat se podría ejecutar así:

```
% cat `grep -l 'include' *`
```

Usando Metacaracteres.

Los metacaracteres tienen significado especial para un shell y para ULTRIX. Permiten crear comandos cortos muy poderosos que ahorran mucho tiempo. Por ejemplo, si se quieren mover 50 archivos de un directorio a otro, se podría introducir el comando mv 50 veces, o se podrían usar metacaracteres para realizar la misma tarea con un sólo comando.

Para mover los 50 archivos en una vez, se substituiría la parte numerada del nombre del archivo con un metacaracter * en el comando mv tal y como se muestra abajo:

Método sin metacaracter

```
% mv arch.1 c/notas
% mv arch.2 c/notas
% mv arch.3 c/notas
...
% mv arch.49 c/notas
% mv arch.50 c/notas
```

Método con metacaracter

```
% mv arch.* c/notas
```

UNIX

Tres tipos de metacaracteres se pueden usar en Unix:

- * Reemplaza cualquier cadena de caracteres, incluyendo a la cadena vacía.
- ? Reemplaza un solo carácter.
- [caracteres] Reemplaza cualquier carácter entre aquellos encerrados con corchetes. Se puede especificar un rango de caracteres usando un guión (-).

IX. Manejo de Procesos

Como ya se ha visto en capítulos anteriores, el shell es un intérprete de comandos. Uno se comunica con el sistema mediante el intérprete de comandos. Cada comando escrito se almacena como un programa independiente, el shell analiza lo que se tecleó y decide que programas ejecutar. Cada usuario del sistema posee una copia del shell. Por ejemplo, cuando se escribe el comando `vi`, el editor `vi` es llamado por el shell para su ejecución. Unix incluye cuatro versiones de shells: C shell (`csh`), Bourne shell (`sh`), System V Bourne shell (`sh5`) y Korn shell (`ksh`).

El C shell toma su nombre y algunas estructuras, del lenguaje de programación C, mientras que el Bourne y Korn shell toman su nombre de sus creadores. El Korn shell es el más nuevo de todos éstos y es una derivación del Bourne shell. El Korn shell combina características de C shell y Bourne shell.

Este capítulo se basa en las características de manejo de procesos bajo el intérprete de comandos C shell.

El shell

Cuando se entra a sesión Unix nos proporciona un shell por default, este shell se denomina login shell. Mientras uno se encuentra en sesión se puede mandar a ejecutar algún otro shell para realizar algunos trabajos, pero estos procesos serán siempre 'hijos' de nuestro login shell.

Si se desea podemos cambiar de login shell, esto se hace mediante el comando `chsh`, al cual se le proporciona la cuenta a la que se le desea cambiar de login shell y la ruta completa de donde se encuentra localizado el intérprete de comandos (shell). Por ejemplo, el usuario `raq` desea cambiar su login shell a C shell (`csh`), entonces debe teclear lo siguiente:

```
$ chsh raq /bin/csh
```

Esto funciona para las versiones de Unix system V, por ejemplo las estaciones de trabajo Hewlett Packard con HP-UX. En Ultrix basta con dar el comando `chsh` y presionar return. La computadora nos informará cual es nuestro login shell actual y nos preguntará cual deseamos que sea nuestro nuevo login shell, donde basta con teclear la abreviación del shell que deseamos. Por ejemplo:

```
$ chsh
Changing login shell for raq
shell [/usr/bin/ksh]: csh
```

Una vez echo ésto, la próxima vez que entremos a sesión, nuestro login shell será C shell (`csh`).

Archivos de configuración de C shell

Cada vez que un usuario entra a sesión, la computadora ejecuta algunos cuantos archivos de comandos para definir ciertas características del medio ambiente donde trabajará el usuario.

Existe también un archivo llamado `.cshrc` (C shell restart command) en el home directory de cada usuario, el cual es leído por cada C shell en cuanto comienza su ejecución. El archivo `.cshrc` puede ser utilizado para establecer el valor de variables de ambiente como podría ser el path o algunos alias o variables que se quisieran definir de manera global. Cuando se cree o modifique el archivo `.cshrc` debemos cerciorarnos de que este tenga permiso de ejecución (`chmod u+x .cshrc`).

A continuación se muestra el contenido típico de un archivo .cshrc:

```
#!/bin/csh
# Default user .cshrc file (/bin/csh initialization).
# Usage: Copy this file to a user's home directory and edit it to
# customize it to taste. It is run by csh each time it starts up.
#
# Set up default command search path:
# (For security, this default is a minimal set.)

    set path=( /bin /usr/bin )

# Set up C shell environment:

if ( $?prompt ) then                # shell is interactive.
    set history=20                   # commands to remember.
    set savehist=20                 # number to save across sessions.
    set system='hostname'           # name of this system.
    set prompt = "$system \!: "     # command prompt.

# Sample alias:

alias h    history

# More sample aliases, commented out by default:

# alias    d    dirs
# alias    pd   pushd
# alias    pd2  pushd +2
# alias    po   popd
# alias    m    more
endif
```

Si lo deseamos, uno puede agregar comandos y/o definiciones de variables al archivo .cshrc para que éstos se ejecuten cada vez que se inicia un nuevo intérprete de comandos C shell.

Si nuestro intérprete de comandos C shell es nuestro login shell, después de haber leído el contenido del archivo .cshrc leerá, en caso de existir, el contenido del archivo .login localizado también el home directory de cada usuario y ejecutará los comandos ahí contenidos. El archivo .login se utiliza comúnmente para configurar las características de nuestra terminal y hacer definiciones con el comando **set**.

UNIX

El siguiente, es una típico archivo .login:

```
# @(#) $Revision: 64.2 $
# Default user .login file ( /bin/csh initialization )
# Set up the default search paths:
set path=(/bin /usr/bin /usr/contrib/bin /usr/local/bin .)
#
#set up the terminal
#
eval `tset -s -Q -m `:?'hp' `
stty erase "^H" kill "^U" intr "^C" eof "^D" susp "^Z" hupcl ixon
ixoff tostop tabs
#
# Set up shell environment:
#
set noclobber
set history=20
umask 002
set prompt="'hostname' `whoami`> "
date
```

Después de que el shell ejecuta los comandos contenidos en el archivo .login, presenta el prompt al usuario y espera por el siguiente comando.

Cuando uno invoca el comando *logout*, *exit* o presiona la secuencia de teclas *^D*, el shell busca en el home directory de cada usuario la existencia del archivo *.logout*. En caso de que este archivo exista, se ejecutan los comandos ahí especificados antes de terminar la sesión del usuario. El siguiente es un ejemplo de lo que puede contener un archivo *.logout*:

```
# /bin/csh
#
echo -n "Usuario saliendo de sesion: "
whoami
echo -n "Terminal: "
tty
date
echo "Adios \!"
```

Agrupando comandos del shell

Hasta ahora, hemos discutido la forma de mandar a ejecutar varios comandos y/o archivos de comandos, pero de momento solo se ha mandado a ejecutar un comando a la vez. Uno puede obtener más ventajas sobre el shell desarrollando la habilidad de ejecutar varios comandos a la vez.

Siempre que usamos el shell con la entrada estándar, el teclado, y con la salida estándar, el monitor, estamos usando el shell en forma interactiva. Una de las ventajas de usar el shell en forma interactiva, es la de poder observar el resultado de los comandos en cuanto nuestra terminal los recibe. Una de las desventajas de usar el shell en forma interactiva (hasta ahora) es la de poder mandar ejecutar un sólo comando a la vez.

Si mandamos ejecutar un segundo comando antes de que el primero termine de ejecutarse, la salida del primero se mezclará muy probablemente en la terminal con nuestro segundo comando mientras lo vamos escribiendo. En el siguiente ejemplo, al escribir el comando `date` se mezcla la salida del comando `who`:

```
% who
dasjg000          tty00      Apr 26 15:03
pnh000           tty02      Apr 26 14:55
kjd000           tty04      Apr 26 12:44
e%
Mon Apr 26 15:15:05 GMT 1993
```

Muchas veces necesitamos observar el resultado de un comando para decidir cuál será el siguiente comando a ejecutar. Sin embargo, hay ocasiones en que la salida del comando anterior no tiene efecto sobre el siguiente comando a teclear. Habrá ocasiones en las que deseemos ejecutar varios comandos a la vez ya que no importa el resultado de éstos para que los demás se ejecuten. Esto es, es posible mandar a ejecutar varios comandos a la vez.

Hasta ahora el 'separador' de comandos ha sido la tecla RETURN, o ENTER. Debido a que RETURN es la tecla que le indica al shell empezar la ejecución del comando escrito en la línea de comandos, sólo se puede ejecutar un comando a la vez.

UNIX

Es posible mandar a ejecutar varios comandos a la vez separando los comandos por puntos y comas (;) como se muestra más adelante. Cuando se le pasan al shell varios comandos para su ejecución separados por puntos y comas en la misma línea de comandos, ésto se conoce como un 'job'. Más adelante en este capítulo hablaremos sobre el manejo de jobs.

```
% who; date; echo "Los sistemas Unix son faciles de aprender y de usar."  
sjg000      tty00      Apr 26 15:03  
pnh000      tty02      Apr 26 14:55  
kjd000      tty04      Apr 26 12:44  
Mon Apr 26 15:15:32 GMT 1993  
Los sistemas Unix son faciles de aprender y de usar.
```

Además, podemos continuar escribiendo un comando en la siguiente línea, utilizando diagonales invertidas (\) al final de cada línea seguida de la tecla RETURN. Con ésto se consigue 'escapar' el significado de la tecla RETURN ante el shell para que éste no mande a ejecutar los comandos escritos hasta el momento, y poder seguir escribiendo en la siguiente línea. Esto es especialmente útil cuando se trata de comandos largos como por ejemplo:

```
% grep "estados" capítulo1.txt capítulo2.txt \capítulo3.txt ../publicaciones.old contenido.list
```

Combinaciones de puntos y comas y antidiagonales pueden ser usadas para escribir comandos que no estén limitados por la longitud de la línea de comandos como se muestra a continuación:

```
% echo "La fecha y hora actual es: "; date; \echo "Lista de los usuarios en sesion:"; \who; echo "Unix posee varias características interesantes."  
La fecha y hora actual es:  
Mon Apr 26 17:11:02 GMT 193  
Lista de los usuarios en sesion:  
sjg000      tty00      Apr 26 15:03  
pnh000      tty02      Apr 26 14:55  
kjd000      tty04      Apr 26 12:44  
Unix posee varias características interesantes.  
%
```

Es importante notar las diferencias entre usar punto y coma (;) para separar comandos y las interconexiones (|) o pipes. Las interconexiones le dicen al shell que la salida del primer comando la use como la entrada del segundo comando. Un punto y coma entre dos comandos le dice al shell que ejecute el primer comando y después de éste ejecute al segundo comando. Esto es, mientras que

con interconexiones los procesos se realizan en forma concurrente, con puntos y comas se ejecutan de manera secuencial.

Cuando se usan interconexiones (|) entre comandos en lugar de puntos y comas, solo es desplegada la salida del último comando. Esto es debido a que la interconexión toma la salida del comando que se encuentra antes de la interconexión y esta se convierte en la entrada del comando que se encuentra después de la interconexión. Si el comando que se encuentra después de la interconexión generalmente no toma su entrada del teclado, la interconexión no tiene sentido. Por ejemplo:

```
% date | who | pwd | ls
ascii      calendar      mbox          plum          settings
bin        misc           practice      shell
% date ; who ; pwd ; ls
Mon Apr 26 17:15:00 GMT 1993
sjg000          tty00          Apr 26 15:03
kjd000          tty04          Apr 26 12:44
/users/cecafi/raq
ascii      calendar      mbox          plum          settings
bin        misc           practice      shell
%
```

Si por el contrario, usamos puntos y comas en lugar de utilizar interconexiones, los resultados pueden ser confusos. Por ejemplo, los siguientes comandos **date** y **who** son ejecutados, pero el resultado del comando **who** no es ordenado como se esperaba. En lugar de esto, da la impresión de que la terminal se 'bloqueo' (queda esperando indefinidamente), lo cual se debe a que el comando **sort** está esperando datos de la entrada estándar (el teclado). Debemos, entonces, presionar la tecla de interrupción o la de fin de archivo para recibir nuevamente el prompt. Si se usan interconexiones, la salida del comando **who** entonces es ordenada como se esperaba y el prompt es regresado inmediatamente después.

```
% date ; who ; sort
Mon Apr 26 17:15:00 GMT 1993
sjg000          tty00          Apr 26 15:03
pnh000          tty02          Apr 26 14:55
kjd000          tty04          Apr 26 12:44
^D
%
% date ; who | sort
Mon Apr 26 17:15:43 GMT 1993
kjd000          tty04          Apr 26 12:44
pnh000          tty02          Apr 26 14:55
sjg000          tty00          Apr 26 15:03
```

%

También podemos agrupar comandos entre paréntesis. El efecto que esto tiene es el de generar un nuevo shell ('hijo' o sub-shell) a nuestro shell actual, y mandar a ejecutar los comandos tecleados en este nuevo sub-shell. Por ejemplo, supongamos que nos encontramos trabajando en el subdirectorio /users/alumnos/vlu000 viendo algunos programas fuente, y deseamos examinar el contenido de algunos archivos en el subdirectorio /users/pub. Entonces podríamos ejecutar el comando siguiente:

```
% (cd /users/pub; grep 'printf' snmpd.c ifconfig.c)
```

Cuando el prompt aparezca de nuevo, el directorio de trabajo permanecerá igual que antes de haberse ejecutado el comando anterior. Esto es debido a que el comando cd y el comando grep fueron ejecutados en el nuevo sub-shell. En cuanto éstos terminaron, el nuevo sub-shell también terminó y el control fue regresado al shell que mandó a ejecutar el comando. En ese shell (el actual) no se ejecutó nunca ninguna orden de cambiar de directorio, es por eso que al regresar el control, se sigue trabajando en el mismo directorio.

También podemos utilizar el agrupamiento entre paréntesis para realizar direccionamientos de salida estándar y de error a la vez. Cosa que no es posible realizar utilizando en forma independiente los símbolos de redireccionamiento. Por ejemplo, para redireccionar la salida estándar de un comando grep al archivo *salida* y la salida de errores al archivo *diagfile* se puede utilizar el comando siguiente:

```
% ( grep 'mount' /etc/* > salida ) >& diagfile
```

Cabe notar lo siguiente:

- 1.- La salida estándar del comando grep se mando al archivo salida:

```
% head salida
/etc/bcheckrc:# anything else that should be done before mounting
/etc/bcheckrc:# put root into mount table
/etc/btldinstall:# mounted on <mntdir>
/etc/btldinstall: echo "<directory on which BTL D disk is mounted>"
/etc/dfspace:# Calculate available disk space in all mounted filesystems.
/etc/termcap:# significant amount of 'nl' delay.
/etc/umountall:#  @(#) umountall.sh 23.1 91/08/29
/etc/umountall:#  Unmounts all but the root, "/", file system.
/etc/umountall:-k kill processes with files open before unmounting."
/etc/umountall:/etc/mount |
```

2.- La salida de errores (>&) del comando encerrado entre paréntesis se mando al archivo diagfile:

```
% head diagfile
grep: can't open /etc/ fst
grep: can't open /etc/aiolkinit
grep: can't open /etc/asktime
grep: can't open /etc/asktimerc
grep: can't open /etc/atab
grep: can't open /etc/auth
grep: can't open /etc/authckrc
grep: can't open /etc/badtrk
grep: can't open /etc/brand
grep: can't open /etc/chroot
```

El comando ps

El comando ps (process status) permite obtener información acerca de los procesos que se encuentran en ejecución. Esta información puede cambiar mientras el comando ps se encuentra en ejecución, por lo que ps nos da información 'instantánea' del estado del procesador en un determinado momento. Cada proceso en ejecución le pertenece a un cierto usuario el cual tiene una identificación numérica dentro del sistema conocida como UID (User IDentification).

Cada usuario es dueño de la copia del intérprete de comandos con el cual trabaja, así como también de los procesos que el shell crea para poder ejecutar sus comandos. El comando ps despliega una lista de los comandos que uno posee además de cierta información extra. Por ejemplo, el siguiente comando:

```
% ps
PID TTY      TIME COMMAND
5643 console 0:02 -csh
6214 console 0:00 ps.
```

despliega el número de identificación del proceso (PID), el nombre de la terminal desde la cual fue lanzado el proceso (TTY), el tiempo acumulado de ejecución (TIME) y el nombre del comando al cual se hace referencia (COMMAND). Estos campos pueden variar

UNIX

ligeramente si se trata de Ultrix, HP-UX, SCO-Unix o alguna otra 'variante' de Unix. Los ejemplos de este capítulo están basados en HP-UX, pero nuevamente recordamos que si se encuentra trabajando en otro sistema como por ejemplo, Ultrix, los cambios son mínimos, por ejemplo, la columna de la terminal, en lugar de decir TTY dirá TT.

```
% ps -elf
F S  UID  PID  PPID  C  PRI  NI   ADDR  SZ  WCHAN  STIME TTY  TIME  CMD
3 S  root  0    0  0 128  20 40013000  0  126ccc Dec 31 ?  0:00 swapper
1 S  root  1    0  0 168  20 40013180  41 904000 Aug  3 ?  0:00 /etc/init
3 S  root  2    0  0 128  20 400130c0  0  146e74 Aug  3 ?  0:00 vhand
3 S  root  3    0  0 128  20 40013100  0  11e0cc Aug  3 ?  0:00 statdaemon
3 S  root  200  0  0 128  20 40013140  0  146e7c Aug  3 ?  0:00 unhashdaemon
3 S  root  6    0  0 152  20 40013240  0 4003aa2c Aug  3 ?  0:00 sockregd
3 S  root  4    0  0 154  20 40013280  0  11eec0 Aug  3 ?  0:00 lcspp
3 S  root  201  0  0 152  20 400132c0  0  11e0e8 Aug  3 ?  0:00 syncdaemon
1 S  root  5643  1  3 168  20 400524c0  46 904000 10:51:00 console 0:02 -csh
1 S  root  609  1  0 154  20 400139c0  13  126cdc Aug  3 ?  0:00 /etc/syslogd
1 S  root  958  1  0 154  20 400521c0  17 147c77a Aug  3 ?  0:00 /etc/cron
1 S  root  241  1  0 127  20 40013640  19 111234 Aug  3 ?  0:00 /etc/nk11 daemon
1 S  root  604  1  0 154  20 40013880  22 40044f22 Aug  3 ?  0:00 /etc/r1bdaemon
1 S  root  244  243  0 127  20 400134c0 129 1476060 Aug  3 ?  0:00 netfmt -CF
1 S  root  614  1  0 154  20 40013b40  34  126cdc Aug  3 ?  0:00 /etc/portmap
1 S  root  617  1  0 154  20 40013ac0  20  126cdc Aug  3 ?  0:00 /etc/inetd
1 S  root  620  1  0 154  20 40013c40  8 40049e2c Aug  3 ?  0:30 /etc/rwhod
1 S  root  636  1  0 154  20 400523c0  11 40049a22 Aug  3 ?  0:00 /usr/bin/nftdaemon
1 S  root  960  1  0 155  20 40052540  12  126dbc Aug  3 ?  0:00 /etc/ptydaemon
1 S  root  643  1  0 154  20 40052500  41  126cdc Aug  3 ?  0:00 /etc/anmpd
1 S  root  963  1  0 154  20 400525c0  16  126cdc Aug  3 ?  0:00 /etc/vtdaemon
1 R  root  6226 5643  8 180  20 40052f00  16  16:40:45 console 0:00 ps -elf
3 S  root  5882  0  0 153  20 40052a40  0  11eed2 16:29:01 ?  0:00 gcsp
```

El comando ps nos proporciona bastante información acerca de los procesos que se encuentran en ese momento ejecutándose. Los códigos que se pueden encontrar en la columna de estado del proceso (S status) pueden ser alguno de los siguientes:

S Durmiendo (Sleeping)
W En espera (Waiting)
R En ejecución (Running)
Z Terminado (Terminated)
T Detenido (Stopped)
X Creciendo (Growing)

En el ejemplo anterior el único proceso que se encuentra en 'ese momento' en ejecución es el proceso número 6226 que al checar en la columna de CMD nos damos cuenta que corresponde al proceso ps.

Otra información importante es la proporcionada por las columnas *UID*, *PID* y *PPID*:

UID Es la identificación del dueño del proceso (**User IDentification**). En HP-UX es una cadena de caracteres con el nombre del usuario, en Ultrix es el número que identifica a ese usuario, el mismo que aparece en el archivo */etc/passwd* en el tercer campo.

PID Es la identificación numérica del proceso (**Process IDentification**). Si se sabe este dato es posible terminar en cualquier momento al proceso.

PPID Es la identificación numérica del padre del proceso (**Parent Process IDentification**). Con este dato es posible identificar que proceso lo lanzó, o es el 'padre' de este proceso.

Ejecución de procesos en Background

Cuando mandamos ejecutar cualquier comando en Unix, ese comando es ejecutado como un proceso subordinado de nuestro login shell, esto es, se genera un proceso hijo y el control del shell no es regresado hasta que este proceso hijo termina. Muchas veces es deseable que el proceso continúe su ejecución mientras se mandan a ejecutar otros comandos. Esto se logra añadiendo una ampersand (&) al final del comando. Por ejemplo, para hacer una compilación de un programa en C podemos utilizar el siguiente comando:

```
% cc prog.c >& prog.err &
```

En este ejemplo, los mensajes de error de la compilación se direccionan al archivo *prog.err* el cual puede ser examinado después. Esta característica de mandar a ejecutar los comandos sin que éstos interfieran con los comandos que se dan en la línea de comandos, es llamada ejecución en '**background**'.

UNIX

Supongamos que queremos obtener un reporte de todos los usuarios del sistema. El reporte contendrá el login name de cada usuario y el nombre de este. Entonces podemos mandar el siguiente comando en background:

```
% awk -F: '{print $1,$5}' /etc/passwd > usuarios &
[1] 1823
%
[1] Done      awk -F: '{print $1,$5}' /etc/passwd > usuarios
```

En el ejemplo anterior, el shell realizó lo siguiente:

- Primero analiza el comando para cerciorarse que la sintaxis es correcta.
- Después, el shell hace una copia de si mismo y este proceso hijo, hace entonces, una serie de ajustes para asegurarse de manejar correctamente la entrada/salida.
- Finalmente, el shell se reemplaza por el comando awk. Cuando el proceso hijo termina, éste se sale de ejecución.

La ejecución del login shell que lanzó al proceso en background no se suspende. El login shell en lugar de esperar a que termine el proceso hijo, imprime en la salida estándar el número del trabajo (o *job_number*) entre corchetes '[' seguido del PID del proceso hijo, y espera entonces, por más instrucciones. En este caso el *job_number* es 1 y el PID del proceso es el 1823.

En el ejemplo anterior del comando `ps (ps -elf)` se puede observar que el `csh` del usuario `root` tiene como padre al proceso número 1, el cual es en casi todas las versiones de Unix conocido como `'init'`. El proceso `init` es el segundo proceso que ejecuta el sistema y es el responsable, entre otras cosas, de generar procesos login encargados de esperar respuesta desde las terminales.

Al terminar nuestra sesión, los procesos que se han mandado en background no son eliminados junto con nuestro login shell. Estos procesos siguen corriendo, pero ahora el sistema les asigna como proceso padre al proceso `init`.

Es importante recordar que los comandos que se ejecutan en background, detendrán su ejecución en caso de necesitar leer de la entrada estándar, a menos de que se establezca la variable `notify` con el comando `set (set notify)` dentro del archivo `.login` o `.cshrc`.

Por ejemplo, el siguiente comando ordena el contenido del archivo 'registros' y la salida la manda al archivo impresión, después imprime este último archivo para posteriormente preguntarnos sobre si deseamos borrar o no el archivo impresión.

```
% sort registros > impresión; lp impresión; rm -i impresión &
[1] + Stopped (tty input)      sortrecs
%
```

En el ejemplo anterior el comando `rm -i` hace que el trabajo se detenga ya que está esperando una respuesta de la entrada estándar. Para poder proporcionarle al comando la entrada que espera, es necesario hacer que el trabajo cambie de background a foreground con el comando '`fg`' de la siguiente manera:

```
% fg %1
rm: remove impresión? y
%
```

El comando `notify %job_number` le dice al C shell que avise en el momento exacto en el cual termina el trabajo `job_number` que se está ejecutando en background. Esto también ocurre si establecemos la variable de ambiente `notify`.

Deteniendo y reanudando procesos

Cuando los comandos se ejecutan en la manera usual, esto es, esperando a que éstos terminen para poder dar el siguiente comando, se dice que se están ejecutando en *foreground*. Estos comandos que se ejecutan en foreground pueden ser detenidos mediante otros comandos o bien, mediante teclas de control.

En cambio, los comandos que al ejecutarse no esperan a terminar para regresarle al usuario el control del prompt, se dice que son comandos en background a los cuales se les puede hacer referencia por su número de trabajo o '`job_number`'.

Es posible ver cuantos comandos están corriendo en background utilizando el comando `jobs`, el cual nos dará un reporte del estado actual del trabajo así como el comando que se mandó a ejecutar.

UNIX

El comando `jobs` lista los trabajos por su número de trabajo (`job number`), el cual es usado para detenerlo (`stop`), continuarlo (`bg` o `fg`) o bien, para terminarlo (`kill`).

La siguiente tabla muestra los comandos para control de trabajos.

COMANDO	FUNCION
<code>comando &</code>	Comienza a ejecutar ese comando o trabajo en background.
<code>jobs</code>	Lista por número de trabajo, los comandos que se encuentran corriendo en background o que están suspendidos.
<code>bg %job_number</code>	Ocasiona que el comando referenciado por <code>job_number</code> continúe su ejecución en background.
<code>fg %job_number</code>	Ocasiona que el comando referenciado por <code>job_number</code> que se encuentra corriendo en background o que se encuentra suspendido, sea ahora ejecutado en foreground.
<code>^Z</code>	Suspende un comando que se encuentra corriendo en foreground.
<code>stop %job_number</code>	Suspende al comando referenciado por <code>job_number</code> .
<code>kill -sig %job_number</code>	Manda la señal <code>sig</code> al trabajo <code>job_number</code> .
<code>kill -sig PID</code>	Manda la señal <code>sig</code> al proceso referenciado por <code>PID</code> .
<code>kill -l</code>	Lista las señales que pueden ser mandadas a los procesos o a los trabajos.

Aunque el comando stop pertenece al sistema operativo Ultrix, aquellos usuarios de otros sistemas operativos pueden determinar la señal que causa la suspensión de los trabajos que están corriendo en background, utilizando el comando kill -l. Utilizando este comando se busca la señal SIGSTOP para hacer referencia a su número asociado o bien, a la señal tal cual. En HP-UX los valores de las diferentes señales que pueden ser mandadas a los procesos se muestran a continuación:

SIGHUP	01	# hangup
SIGINT	02	# interrupt
SIGQUIT	03	# quit
SIGILL	04	# illegal instruction
SIGTRAP	05	# trace trap
SIGABRT	06	# software generated abort; see abort(3C)
SIGIOT	06	# software generated signal
SIGEMT	07	# software generated signal
SIGFPE	08	# floating point exception
SIGKILL	09	# kill
SIGBUS	10	# bus error
SIGSEGV	11	# segmentation violation
SIGSYS	12	# bad argument to system call
SIGPIPE	13	# write on a pipe with no one to read it
SIGALRM	14	# alarm clock; see alarm(2)
SIGTERM	15	# software termination signal
SIGUSR1	16	# user defined signal 1
SIGUSR2	17	# user defined signal 2
SIGCHLD	18	# death of a child (see WARNINGS below)
SIGCLD	18	# death of a child (see WARNINGS below)
SIGPWR	19	# power fail (see WARNINGS below)
SIGVTALRM	20	# virtual timer alarm; see getitimer(2)
SIGPROF	21	# profiling timer alarm; see getitimer(2)
SIGIO	22	# asynchronous I/O signal; see select(2)
SIGWINDOW	23	# window change or mouse signal; see windowing package
SIGSTOP	24	# stop
SIGTSTP	25	# stop signal generated from keyboard
SIGCONT	26	# continue after stop
SIGTTIN	27	# background read attempted from control terminal
SIGTTOU	28	# background write attempted to control terminal
SIGURG	29	# urgent data arrived on an I/O channel
SIGLOST	30	# file lock lost (NFS file locking)

Para mandar una señal a un proceso se hace uso del comando kill como se muestra en la tabla anterior. Cabe mencionar que el comando kill lo que hace es mandar señales a los procesos. Una

UNIX

práctica muy común para los usuarios de Unix, es utilizar el comando `kill` para eliminar procesos que no se desea que sigan corriendo, mandándoles precisamente la señal de terminación **SIGKILL**. Esto se hace de la siguiente manera:

```
% kill -9 PID
% kill -SIGKILL %job_number
```

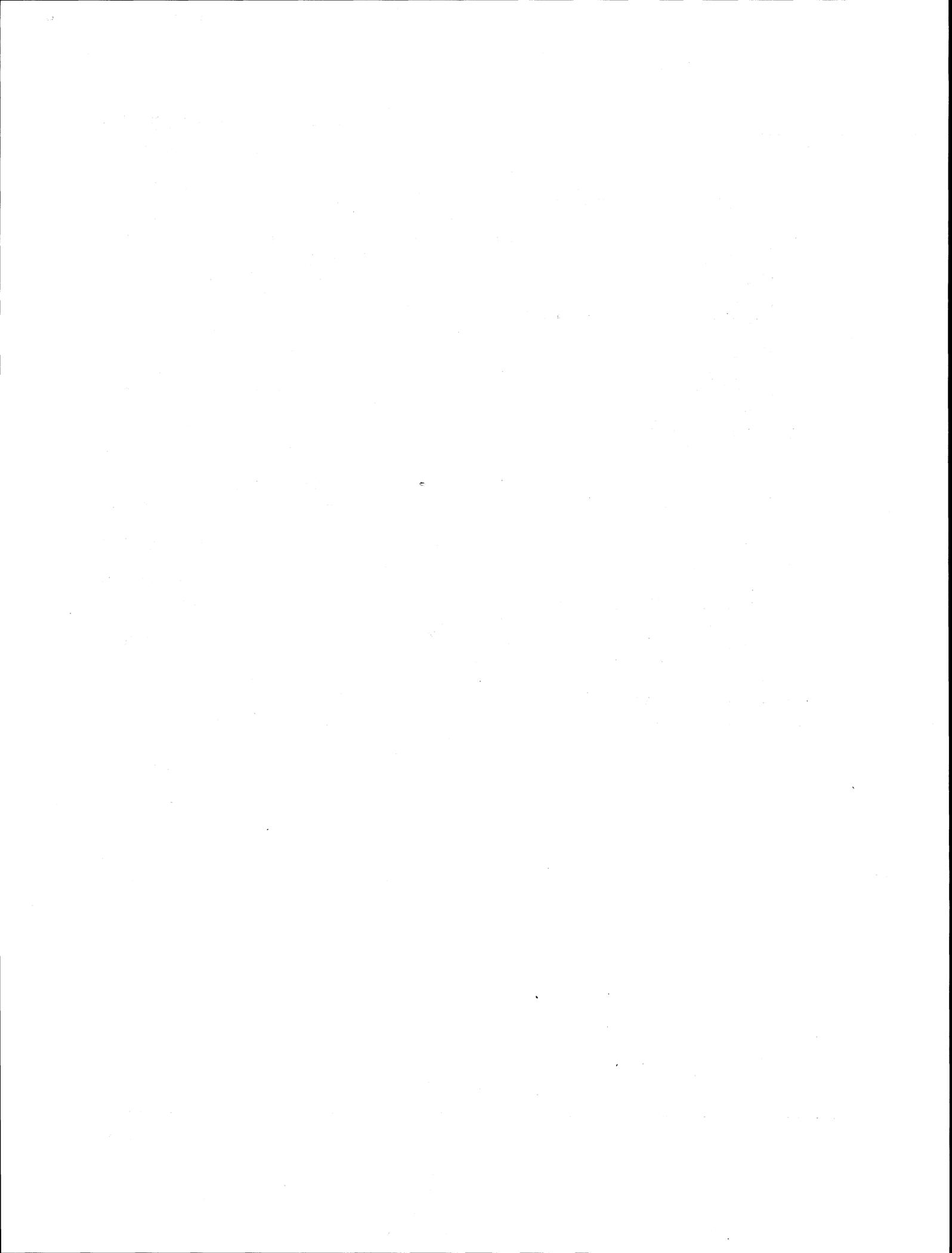
en cualquiera de los dos casos anteriores, lo que se está haciendo es mandar la señal **9** que corresponde a la señal **SIGKILL** la cual causa que el proceso termine inmediatamente.

Por último, el siguiente ejemplo nos muestra como se mandan a ejecutar varios comandos en background, los cuales son detenidos, reanudados y eliminados en su ejecución utilizando los comandos antes mencionados:

```

% ls -laR / > /tmp/biglist &
[1] 6545
% jobs                               Desplegar el estado de los procesos en background
[1] +Running                          ls -laR / > /tmp/biglist
% kill -9 %1                          Eliminar el trabajo número 1
[1] +Killed                            ls -laR / > /tmp/biglist
% ls -laR / >! /tmp/biglist &
[1] 6548
% jobs
[1] +Running                          ls -laR / > /tmp/biglist
% kill -24 %1                          Detener momentáneamente al proceso
% jobs
[1] +Stopped                          ls -laR / > /tmp/biglist
% kill -26 %1                          Reanudar el proceso
% jobs
[1] +Running                          ls -laR / > /tmp/biglist
% kill -24 %1                          Detener nuevamente el proceso
[1] +Stopped                          ls -laR / > /tmp/biglist
% bg %1                                Reinicializar el proceso mandándolo a background
[1] +Running                          ls -laR / > /tmp/biglist
% ps -f                                Ver los procesos que se encuentran corriendo
    UID    PID  PPID  C   STIME TTY      TIME COMMAND
    root   5643    1   0 10:51:00 console 0:11 -csh
    root   6544   5643  1 17:03:17 console 0:00 csh -vx cmd
    root   6604   6544  7 17:03:29 console 0:00 ps -f
    root   6548   6544  0 17:03:22 console 0:00 ls -laR /
% kill -9 6548                          Eliminar el proceso ahora por su PID
[1] +Killed                            ls -laR / > /tmp/biglist
%

```



X. Manejo de Cintas y Discos

El respaldo de la información es una tarea del administrador del sistema; pero muchas veces es necesario que como usuarios podamos realizar nuestros propios respaldos en el momento deseado ya sea a unidades de cinta o disco. Sabiendo manejar adecuadamente las unidades de cinta y disco es posible transferir información entre diferentes sistemas que manejen Unix o incluso hacer transferencias entre sistemas operativos como Unix y MS-DOS.

Para hacer respaldos y/o transferir información desde y hacia las estaciones de trabajo podemos hacer uso de los comandos tar (tape archiver), cpio (copy input-output) y los comandos dosll, doscp, etc. Los primeros dos comandos se encuentran disponibles en casi todas las computadoras con Unix y nos permiten respaldar información en unidades de cinta o disco. Los comandos dosll, doscp, etc., no se encuentran en todas las versiones de Unix, aunque si es común encontrarlos en aquellas computadoras que manejen unidades de disco flexible, y nos permiten transferir información entre los sistemas operativos Unix y MS-DOS.

Antes de realizar un respaldo conviene conocer lo siguiente:

- 1.- El archivo asociado a la unidad de disco o cinta que deseamos acceder.
- 2.- La capacidad de almacenamiento de dicha unidad.
- 3.- La ubicación de los archivos a respaldar.

En los sistemas Unix los archivos asociados a las unidades de cinta y/o disco se encuentran ubicadas por lo general bajo el subdirectorio `/dev`. Si no conocemos cual es el archivo asociado a la unidad que deseamos manejar podemos preguntarle al administrador del sistema cuales son estos archivos. Por ejemplo, en la estación de trabajo cancan (132.248.54.10), el archivo asociado al disco externo de 3 1/2" es: `/dev/rdisk/0s0`, en la estación de trabajo ixtapa este archivo es: `/dev/rdisk/3s0`. Así que si nosotros mandamos a leer o escribir a esa unidad, estaremos obteniendo o mandando información a nuestro disco de 3 1/2".

Usando el comando tar

El comando **tar** (tape archiver) permite crear y restaurar respaldos de archivos a y desde cintas magnéticas o discos flexibles. El funcionamiento del comando **tar** es controlado por su primer argumento denominado 'argumento llave'. La sintaxis del comando **tar** es la siguiente:

```
tar argumento_llave [f archivo_de_dispositivo] [archivo...]
```

Algunas opciones del argumento llave son las siguientes:

- c** Crea un nuevo respaldo (volumen) y en caso de que el destino contenga información, ésta será borrada.
- r** Agrega los archivos al final del volumen.
- t** Despliega los archivos contenidos en el volumen.
- u** Los archivos son añadidos al volumen solamente si éstos son nuevos o han sido modificados.
- x** Restaura los archivos del volumen.

Por ejemplo, para hacer un respaldo de todos nuestros archivos que terminen en '.c' en un disco de 3 1/2", podemos hacer uso del comando **tar** de la siguiente forma:

```
% tar -cvf /dev/rdisk/0s0 *.c
```

en donde le estamos especificando lo siguiente:

- c** Le indica al comando **tar** que cree un nuevo volumen en esa unidad ya sea de cinta o de disco. Esto **DESTRUYE** la información que pudiera existir en esa unidad y la inicializa para escritura.
- v** Se le solicita al comando **tar** que se ejecute en modo verbose (palabra por palabra). Esto es, que imprima todas

las acciones realizadas. El comportamiento de tar por omisión, es el de no mandar mensajes a la salida estándar de lo que está haciendo.

- f Le especifica al comando tar que el siguiente parámetro es el dispositivo sobre el cual va a hacer el respaldo. En este caso el nombre de archivo /dev/rdisk/0s0 hace referencia a la unidad de disco externo de 3 1/2". Si por ejemplo nosotros queremos hacer el respaldo y dejar toda la información en un archivo llamado respaldo en nuestro home directory, se hubiera especificado de la siguiente manera: -f \$HOME/respaldo.

/dev/rdisk/0s0 En este ejemplo hace referencia a la unidad de disco externo de 3 1/2".

- *.c Hace referencia a los archivos que deseamos respaldar. Podemos hacer uso de los comodines * y ? para especificar nombres de archivos.

Una vez que hicimos ésto, nuestra información estará respaldada en disco, dándonos ésto la facilidad de poder realizar nuestros respaldos sin la necesidad de depender del operador o del administrador del sistema para proteger nuestra información.

Para agregar un archivo al final del volumen podemos usar el comando tar de la siguiente manera

```
% tar -rvf /dev/rdisk/0s0 lastfile.c
```

Para listar el contenido de alguna cinta o disco en el cual hayamos respaldado información con el comando tar, usamos el siguiente comando:

```
% tar -vtf /dev/rdisk/0s0
```

Recordemos que en estos ejemplos, el archivo /dev/rdisk/0s0 hace referencia a la unidad de disco externo de 3 1/2" de la estación de trabajo cancan (132.248.54.10). Ahora para poder recuperar un archivo o todos de un respaldo realizado con tar, podemos hacer uso del siguiente comando:

```
% tar -xvf /dev/rdisk/0s0 union.c
```

UNIX

En el ejemplo anterior se le está pidiendo que extraiga el archivo llamado *union.c*. Si quisiéramos restaurar todos los archivos podríamos haber colocado en lugar de '*union.c*' un asterisco ***.

```
% tar -xvf /dev/rdisk/0s0 *
```

Uso del comando *cpio*

Otra forma de hacer respaldos es usando el comando *cpio* (copy input output). Este comando trabaja en dos modos. Para hacer respaldos:

```
cpio -o
```

En este modo *cpio* lee la entrada estándar en busca de nombres de archivos y los copia a la salida estándar junto con su ruta de acceso e información del archivo. Para restaurar archivos usamos el comando *cpio* de la siguiente manera:

```
cpio -i
```

la cual lee la entrada estándar de la cual asume que es el resultado de un comando *cpio* previo y restaura los archivos especificados.

Algunas de las opciones que podemos usar con el comando `cpio` se muestran a continuación.

<code>-o</code>	<code>-i</code>	DESCRIPCIÓN
<code>-c</code>	<code>-c</code>	Escribir el encabezado en formato ASCII. Si es usado con <code>-o</code> deberá de ser usado con <code>-i</code> .
<code>-x</code>	<code>-x</code>	Manejo de archivos especiales.
<code>-v</code>	<code>-v</code>	Despliega los archivos copiados.
	<code>-u</code>	Restaurar incondicionalmente los archivos especificados. Si el archivo ya existe, esta opción lo sobrescribe.
	<code>-m</code>	Retiene la fecha de modificación de los archivos.
	<code>-d</code>	Restaura la estructura de directorios según se necesite.

Por ejemplo, para respaldar en disco de 3 1/2" todos nuestros archivos que terminen en `.c`, podemos usar el siguiente comando:

```
% find . -name '*.c' -print | cpio -ocx > /dev/rdisk/0s0
```

Para restaurar los archivos de un respaldo, usamos el comando `cpio` de la siguiente manera:

```
% cpio -icxudm < /dev/rdisk/0s0
```

Si solo deseamos ver el contenido de una cinta o disco, usamos a `cpio` de la siguiente manera:

```
% cpio -ict < /dev/rdisk/0s0
```

Para restaurar un solo archivo de un respaldo, usamos el comando `cpio` de la siguiente manera:

```
% cpio -icxudm '*archivo*' < /dev/rdisk/0s0
```

Aunque `cpio` hace bien su trabajo de archivar y restaurar información, esto causa un mayor desgaste en las unidades de cinta debido al avance y retroceso de la misma, debido a que las velocidades de transmisión desde la computadora hacia la unidad de

UNIX

cinta no son las mismas. Para evitar ésto podemos hacer uso del comando `tcio`, el cual crea un área temporal de almacenamiento para después transferir directamente a la unidad de cinta o disco la información y evitar el prematuro desgaste de las unidades. El comando `tcio` al igual que el comando `cpio` funciona de dos formas distintas:

```
tcio -i  <- Escribe a un dispositivo.  
tcio -o  <- Lee desde un dispositivo.
```

Las opciones para `tcio` corresponden con las mismas que `cpio`. Por ejemplo, para crear un respaldo, podemos usar el siguiente comando:

```
% find . -print | cpio -ocx | tcio -o /dev/rdisk/0s0
```

Para restaurar un respaldo:

```
% tcio -i /dev/rdisk/0s0 | cpio -icxudm
```

Para obtener un listado del contenido de un respaldo:

```
% tcio -i /dev/rdisk/0s0 | cpio -ict
```

Para restaurar un solo archivo:

```
% tcio -i | cpio -icxudm '*archivo*'
```

Transferencia de archivos entre Unix y MS-DOS

Cuando trabajamos en las Estaciones de Trabajo muchas veces generamos archivos como parte de nuestro trabajo. Estos aparte de poder imprimirlos para tener una copia en papel del archivo, los podemos transferir a discos de 3 ½" con formato de MS-DOS para tener un respaldo de nuestra información, o bien, poder seguir trabajando con ellos mientras el equipo es usado por otras personas, por ejemplo, podemos realizar la ediciones de estos archivos en una PC y cuando estén listos, compilarlos en la Estación de Trabajo, con lo cual estaríamos aprovechando mejor nuestro tiempo en ella.

Las Estaciones de Trabajo en CECAFI que cuentan con unidad de disco de 3 ½" son: Cancun (132.248.54.10), Cozumel (132.248.54.13), Ixtapa (132.248.54.11) y Mazatlan (132.248.54.12).

Recordemos que en Unix todos los dispositivos ya sean estos discos, DAT, CD-ROM o el disco interno de las máquinas, es visto como un archivo. Si nosotros mandamos a escribir al archivo asociado con la unidad que deseamos usar, en realidad estaríamos escribiendo a esa unidad la información deseada. En las Estaciones de Trabajo, los archivos relacionados con los discos de 3 ½", se encuentran localizados en el subdirectorio: /dev/rdisk y se llaman *xs0*, donde *x* es un número que puede ser del 0 al 6. Por ejemplo, en la Estación de Trabajo Cancun (132.248.54.10) el archivo a donde debemos de mandar escribir nuestra información, es: /dev/rdisk/0s0, en la Estación de Trabajo Ixtapa el archivo es: /dev/rdisk/3s0 al igual que en Mazatlan, mientras que en Cozumel el archivo es: /dev/rdisk/5s0.

La copia se realiza con el comando **doscp** en forma muy similar que con el comando **cp** (copy):

```
% cp origen destino
% doscp origen destino
```

Por ejemplo si estamos trabajando en la Estación de Trabajo Ixtapa y deseamos copiar el archivo *datos.dat* localizado en nuestro home directory, hacia nuestro disco, el comando que le daríamos sería el siguiente:

```
% doscp datos.dat /dev/rdisk/3s0:datos.dat
```

En este ejemplo, notemos los dos puntos que separan el nombre del archivo (*datos.dat*) del nombre del dispositivo (/dev/rdisk/3s0). Además, es necesario proporcionar el nombre del archivo en el destino, aquí no asume que deseamos dejarle el mismo nombre, sino que en caso de omitir el destino, nos marcaría un error de sintaxis. Tampoco se permite el uso de comodines para especificar múltiples archivos (aunque esto no es gran problema). Ahora si deseáramos transferir algún archivo generado en MS-DOS hacia las Estaciones de Trabajo, haríamos lo mismo pero intercambiando el lugar del origen con el del destino, por ejemplo:

```
% doscp /dev/rdisk/3s0:datos.dat datos.ms_dos
```

en el cual se estaría copiando el archivo *datos.dat* desde el disco de 3 ½" hacia nuestro directorio actual de trabajo bajo el nombre de *datos.ms_dos*.

UNIX

En caso de que deseáramos transferir muchos archivos hacia o desde disco, podemos utilizar un archivo de comandos para poder especificar ahí los comodines que deseemos para poder realizar la transferencia mucho más rápido. El archivo para copiar desde la Estación de Trabajo hacia disco, es el siguiente:

```
#!/bin/csh
foreach file ( archivos )
    echo "Copiando $file a /dev/rdisk/xs0:$file ..."
    doscp $file /dev/rdisk/xs0:$file
end
```

o bien, para copiar archivos desde disco hacia la Estación de Trabajo:

```
#!/bin/csh
foreach file ( archivos )
    echo "Copiando /dev/rdisk/xs0:$file a $file ..."
    doscp /dev/rdisk/xs0:$file $file
end
```

Nótese que solo se intercambió el orden del origen y destino en los dos archivos de comandos, archivos son los nombres de los archivos a copiar, aquí si se pueden utilizar comodines, y nuevamente x puede ser substituida por alguna de la unidades ya antes mencionadas.

Por último supóngase que se requiere respaldar todos los archivos con extensión '.c' en disco de 3 ½" para trabajar con ellos en una PC, entonces se puede utilizar el primer archivo de comandos de la siguiente manera (supóngase también que ahora nos encontramos trabajando en la Estación de Trabajo Cancun, con lo cual cambiaría el archivo relacionado con la unidad de disco de 3s0 a 0s0):

```
#!/bin/csh
foreach file ( *.c )
    echo "Copiando $file a /dev/rdisk/0s0:$file ..."
    doscp $file /dev/rdisk/0s0:$file
end
```

y una vez modificado nuestro archivo lo hacemos ejecutable y lo corremos de la siguiente manera (si nuestro archivo se llama copia):

```
% chmod u+x copia
% copia
```

Impresión de archivos

Para poder imprimir un archivo hacemos uso del comando `lp` seguido del nombre del archivo(s) que deseamos imprimir. Por ejemplo, si deseamos mandar a imprimir el archivo `resultados` debemos dar el siguiente comando:

```
% lp resultados
request id is text-54 (1 file)
%
```

En caso de especificarse más de un archivo en la línea de comandos, éstos serán impresos en el orden en que aparecen en la línea de comandos. El comando `lp` proporciona una identificación única (`id`) por cada impresión, ya sea un solo archivo o un conjunto de archivos. Esta identificación puede servir para poder cancelar el trabajo de impresión u obtener información sobre el mismo.

Si por alguna causa deseáramos cancelar el trabajo de impresión, podemos usar el comando `cancel` seguido del `id` que nos regresó el comando `lp`. Por ejemplo, para cancelar el trabajo de impresión del ejemplo anterior, podríamos usar el comando siguiente:

```
% cancel text-54
request "text-54" cancelled
%
```

UNIX

Si solamente deseamos obtener información sobre el trabajo de impresión, damos el comando `lpstat` seguido del id de la impresión que nos interesa. En un sistema multiusuario como lo es HP-UX puede haber más de una impresora conectada al sistema, lo cual nos daría la facilidad de poder seleccionar por que impresora deseamos mandar a imprimir nuestro trabajo. Tal vez por una de matriz de puntos para un documento no muy formal, pero por una impresora láser para aquellos documentos importantes, etc. Para averiguar que impresoras están definidas en nuestro sistema, así como los trabajos que están en esa(s) impresora(s) usamos el comando `lpstat` con el parámetro `-t` de la siguiente forma:

```
% lpstat -t
scheduler is running
system default destination: text
device for text: /dev/null
text accepting requests since Jul 28 15:47
printer text disabled since Oct 15 16:04 -
    cambio de papel
    fence priority : 0
text-56          root          priority 0   Oct 15 16:04
    passwd                                15166 bytes

printer queue for text
no entries
cozumel.cecafi.unam.mx: Warning: text is down

cozumel.cecafi.unam.mx: text: ready and waiting
%
```

XI. Comunicándose con otros usuarios

En todas partes existe información. Una clave para usar esta información efectivamente es distribuirla eficientemente. Hoy en día, los medios más eficientes para distribuir información son electrónicos.

Una de las funciones más importantes de la computadora actual es su capacidad de comunicación. Los beneficios de la comunicación electrónica incluyen: distribución más rápida de información, acceso más fácil a esa información, información actualizada, eliminación de cuellos de botella de información, y una consecuente mejora en el manejo de la información.

El sistema operativo UNIX proporciona dos maneras (la utilería **mail** y la **write**) de comunicarse electrónicamente con otros usuarios. La utilidad **mail** nos permite recibir mensajes electrónicos y mandar mensajes a usuarios que están, o no, en sesión. La utilidad **write** nos permite la comunicación con otros usuarios que están en sesión en ese momento.

Correo Electrónico.

En HP-UX, mailx (mail en Ultrix) nos permite recibir, seleccionar, ver, mandar y mantener mensajes de correo. Los mensajes pueden consistir de mensajes pequeños o de documentos completos. Como hemos mencionado, mailx permite mandar mensajes a usuarios del sistema que estén o no en sesión.

UNIX

Enviando Correo.

Para mandar correo a otro usuario (ya sea que esté o no en sesión):

Prompt del sistema: Tecleamos:

% **mailx** *usuario*, y oprimimos RETURN (donde usuario es la cuenta asociada -login- de la persona a la que deseamos mandar el correo).

Subject: El tema o título del mensaje y oprimimos RETURN; o sólo oprimimos RETURN si seleccionamos dejar el título en blanco.

Introducimos el cuerpo del mensaje y oprimimos RETURN para mover el cursor al principio de la próxima línea. Oprimimos CTRL/D para finalizar el mensaje.

Cc: Usuarios a los que debemos mandar copias al carbón adicionales. Para varios usuarios tenemos que separar los nombres con espacios o comas.

% Indica que hemos mandado exitosamente el mensaje de correo

Editando el correo.

Cuando estamos introduciendo el cuerpo del mensaje de correo, la edición está limitada a la línea actual. Esto lo podemos evitar usando cualquiera de los siguientes tres métodos:

Método 1 - Editar un archivo antes de mandarlo por correo

Simplemente tenemos que crear y editar un documento como un archivo (por ejemplo, carta.correo) y mandar ese archivo. Por ejemplo:

```
% mailx raq hugo hbr < carta.correo
```

manda el documento carta.correo a los usuarios raq, hugo y hbr. Este proceso es muy útil para mandar documentos completos a otros usuarios. Debemos notar el símbolo de redireccionamiento <.

NOTA: el campo de título (Subject) queda en blanco, a menos que se especifique en la línea de comandos el parámetro -s.

Método 2 - Editando un archivo cuando se está en Mail

Editamos el mensaje de correo mientras tecleamos, usando el editor vi. Una vez que empezamos a introducir el cuerpo del mensaje podemos fácilmente entrar al editor vi moviendo el cursor al principio de la línea y tecleando ~v (tilde v).

Cuando finalizamos la sesión de edición del vi, oprimimos ESC, y después tecleamos :wq ó :x para continuar la sesión de mail. Cualquier otra cosa que tecleemos es agragada al final del mensaje a menos que oprimamos CTRL/D al principio de una línea para acabar el mensaje.

Método 3 - Usando ~r para mandar un archivo especificado

Tenemos que crear un archivo, o asegurarnos de que el archivo que tratamos de mandar por correo existe, y después realizar la secuencia siguiente:

% mailx amezcua	Teclear el comando
Subject: Aviso <RETURN>	Introducir un título
~r aviso.txt <RETURN>	Teclear
"aviso.txt" 8/171	es desplegado
^d	
Cc: <RETURN>	Introducir otros usuarios y/o RETURN
%	

Funciones ~(Tilde) de Escape Adicionales

~v es un ejemplo de como podemos usar la tilde (~) en mailx. Estas funciones son referidas como escapes de tilde.

Los escapes de tilde son reconocidos unicamente al principio de una línea.

UNIX

Algunos de los escapes de tilde más comunes son listados a continuación:

Funciones de Escape de Tilde

Opción	Función
<code>~v</code>	Invoca al editor <code>vi</code>
<code>~t</code> usuarios(s)	Agrega nombre(s) a la lista de receptores directos
<code>~c</code> usuario(s)	Agrega nombre(s) a la lista de receptores de copias al carbón
<code>~p</code>	Imprime los campos del encabezado, seguidos por el contenido completo del mensaje hasta ese momento
<code>~r</code> archivo	Incluye el archivo especificado dentro del mensaje
<code>~s</code> cadena	Usa la cadena como el campo de título (Subject) en el encabezado del mensaje
<code>~?</code>	Despliega una lista de ayuda de los escapes de tilde

Los escapes de tilde hacen del mandar mensajes de correo una tarea más fácil y rápida.

Configurando el Archivo de Correo `.mailrc`

Si encontramos que los escapes de tilde `~s` y `~c` se usan frecuentemente, podemos configurar el correo para que automáticamente solicite el título y las copias de carbón cada vez que mandamos correo, creando el archivo de configuración `.mailrc`.

El archivo `.mailrc` debe estar localiazado en el directorio hogar. El archivo `.mailrc` no está presente cuando entramos a sesión por primera vez; debemos crearlo con el editor `vi` o tecleando las siguientes líneas:

```
% cd
% cat > .mailrc
set ask
```

```
set askcc
^D
%
```

En este ejemplo, `.mailrc` contiene dos opciones:

`ask`, la cual le dice a mail que pida un título cuando se manda un mensaje

`askcc`, la cual solicita los usuarios a los que les mandaremos copias de carbón cuando acabamos de teclear un mensaje

Cuando mandamos correo, si escogemos no tener título o no mandar copias de carbón de un mensaje dado, sólo debemos oprimir RETURN como respuesta al prompt correspondiente.

El Archivo de Correo del Sistema

Cuando la utilidad mail es llamada, antes de pasar el control al usuario, ejecuta el contenido del archivo `/usr/lib/mailx/mailx.rc` (`/usr/lib/Mail.rc` en Ultrix). Este es un archivo controlado por el administrador del sistema y contiene comandos que son aplicables a todos los usuarios del sistema.

El Comando set

Hay dos tipos de opciones que podemos fijar con el comando `set`:

1. Opciones que están apagadas o prendidas (binarias). Por ejemplo, el comando `set` prende la opción: `set opcion`

Y, el comando `unset` apaga la opción: `unset opcion`

2. Opciones que tienen un valor. Por ejemplo: `set opcion=valor`

Para especificar una opción, tenemos que editar o crear el archivo `.mailrc` para incluir la(s) línea(s) necesarias. Podemos especificar varias opciones en un solo comando `set`.

Alias

Además de personalizar la cuenta de mail, podemos usar el archivo `.mailrc` para definir un alias. Un alias es un nombre que está en lugar de uno o más nombres de usuario. El correo que mandamos a un alias es realmente mandado a la lista de usuarios reales o a una lista de distribución asociada con él.

UNIX

Podemos definir el comando `alias` para los miembros de un proyecto, para que podamos mandar correo al proyecto completo mandando correo a sólo un nombre.

Por ejemplo, supongamos que los participantes en un proyecto tienen los nombres de usuario `angel`, `andres`, `hugo` y `javier`. Para definir un alias llamado "proyecto" para ellos, debemos de introducir la siguiente línea en el archivo `.mailrc`:

```
alias proyecto angel andres hugo javier
```

Incluyendo esta línea en el archivo `.mailrc` se define `proyecto` como una lista de distribución que incluye `angel`, `andres`, `hugo` y `javier`. Para mandar correo a todos los usuarios en esta lista de distribución, tecleamos:

```
% mailx proyecto
```

Podemos usar el comando `alias` también para dar un nombre conveniente a alguien cuyo nombre de usuario es inconveniente. Por ejemplo, si un usuario llamado Ramon Gallardo tuviera como login `gallardo`, se podría usar:

```
% alias moncho gallardo
```

Este comando nos permite mandar correo al nombre más corto, `moncho`.

Cuando usamos alias personales (los alias listados en el archivo `.mailrc`) para mandar correo, aquellos que lo reciben pueden contestar a todos o sólo a algunos de los receptores listados en la lista de distribución.

Leyendo el Correo

Si tenemos correo cuando se entra a sesión, el sistema operativo manda el mensaje: "You have mail."

Para leer el correo, simplemente tecleamos el comando `mailx` y presionamos RETURN.

Cuando estamos leyendo el correo, el menú numerado de todos los mensajes de correo actuales aparece en la pantalla seguido por el prompt del mail el cual es un signo de interrogación (ampersand, `&`, en Ultrix).

Los encabezados de mensaje en el menú son listados en el orden en el cual fueron recibidos. Cuando estamos listos para regresar al prompt `%`, debemos teclear `ex`, o `x` o `q`.

Comunicándose con otros usuarios

Las entradas de mensaje están precedidas por los siguientes símbolos:

Símbolo	Significado
N	Mensaje nuevo, mailx no tiene registro de que lo hayamos visto anteriormente
U	Mensaje no leído. Hemos visto el encabezado de mensajes en el menú de correo antes, pero no lo hemos leído entonces
>	Apunta al mensaje actual. Inmediatamente después que se invoca mailx, > apunta al primer mensaje nuevo en la lista.

Una línea del menu de mail puede ser como la siguiente,

```
U 1 hbr Thu Sep 9 21:30 11/295 Software nuevo
```

donde,

U	significa que no hemos leído aún este mensaje de correo
1	indica que este es el mensaje número 1
hbr	indica quien mandó el mensaje
Thu Sep 9 21:30	muestra el día de la semana, mes, día y hora en que el mensaje fue mandado
11/295	muestra el número de líneas y caracteres en el mensaje, incluyendo al encabezado
Software nuevo	es un duplicado del título del mensaje

Comandos de Mail adicionales

La tabla siguiente contiene los comandos que un principiante necesita conocer para saber leer y manipular el correo. Antes, debemos recordar estas definiciones:

UNIX

- . Una **lst msj** es una lista de uno o más números de mensaje. En la mayoría de los casos la lista puede ser omitida, y el último mensaje escrito es usado (un rango se marca con un guión entre el límite inferior y el superior).
- . Una **lst usr** es una lista de nombres de login separados por espacios.

Comandos de Mail Comunes

Función	Comando	Resultado
Ayuda en-línea	??, help	una lista de comandos es desplegada junto con un resumen de su significado.
Terminar mail	?q, ^D	Mail es terminado; todos los cambios hechos al mailbox (borrados) son actualizados.
Leer un mensaje	?RETURN	El mensaje actual es desplegado
	?p, t	El mensaje actual es desplegado o redesplegado.
	?n, +	El próximo mensaje es desplegado.
	?-	El mensaje anterior es desplegado.
	?numero	El número de mensaje seleccionado es desplegado.
Borrar mensajes	?d lst msj	Todos los mensajes cuyos números están en lst msj son borrados. Si ninguna lista es dada, sólo el mensaje actual (señalado) es borrado.

Comunicándose con otros usuarios

	?dp,dt	El mensaje leído más recientemente es borrado y el próximo mensaje es desplegado.
	?u lst msj	Todos los mensajes cuyos números están en lst msj son desborrados y puestos de regreso en el mailbox del usuario. Sólo los mensajes de la sesión actual de correo son desborrados.
Respuesta a mensajes	?R	Permite mandar una respuesta al usuario que mandó el mensaje que se acaba de leer. El campo de título del mensaje contiene el título del mensaje al que se le está respondiendo, precedido por "Re:".
	?r	Igual que R, pero manda una respuesta a todos los receptores del mensaje original así como al que lo manda.
	?set	Lista las variables de mail.
Salvar mensajes	?s,w archivo	El mensaje es salvado en el archivo mbox en el directorio hogar.
	?pre,ho	El mensaje es guardado en el mailbox del sistema.
	?co	El mensaje es copiado a mbox.
	?s,w,co lst msj archivo	Los mensajes en lst msj son salvados en el archivo nombrado en lugar de mbox.

Listar todos los mensajes recibidos	?h, f	El menú de mensajes es desplegado. Los mensajes borrados no se incluyen en la lista. Los mensajes conservados son precedidos por una P.
	?top	Imprime las primeras cinco líneas del mensaje actual.
	?z, z-	Los encabezados son movidos una pantalla hacia arriba o una pantalla hacia abajo si la lista es muy larga para ser desplegada en una sola pantalla.

El archivo "Mailbox".

Si no leemos los mensajes y no los borramos o salvamos en un archivo específico, éstos son guardados en un archivo llamado "Mailbox" que se encuentra en /usr/mail/<usuario> cuando usamos q para salir de la utilidad de mailx. Por otra parte, existe también un archivo de nombre mbox el cual es creado por la utilidad mailx cuando leemos los mensajes y no los borramos. Si mbox ya existe cuando salimos de mailx, mailx agrega los mensajes, que leemos pero no borramos, al final del archivo mbox.

Podemos leer tanto el "Mailbox" como el archivo mbox tecleando:

```
% cat /usr/mail/<usuario> para desplegar todos los mensajes del "Mailbox" y,
```

```
% cat $HOME/mbox para desplegar los mensajes guardados en este archivo.
```

Folders.

Una vez que empezamos a usar el mailx se empiezan a acumular muchos mensajes. Podemos salvar algunos de ellos y organizarlos para un

Comunicándose con otros usuarios

acceso más sencillo. **mailx** incluye una facilidad sencilla para organizar grupos de mensajes juntos en folders.

Usando folders.

mailx nos permite crear tantos folders como necesitemos. Cada folder de mensajes es un archivo. Por conveniencia, todos los folders son guardados en un solo directorio que hayamos escogido. Debemos de especificar el directorio de folders en el archivo de opciones **.mailrc** como se muestra a continuación:

```
set folder=cartas
```

Este comando designa a "cartas" como el directorio de folders. Si el directorio de folders no empieza con "/", mail asume una ruta relativa al directorio hogar.

En cualquier parte que se espera un nombre de archivo, podemos usar un nombre de folder, precedido por '+'. Por ejemplo para colocar un mensaje en un folder con el comando **save**, podemos usar:

?s +clase

Este comando salva el mensaje actual en el folder llamado clase. Si el folder clase todavía no existe, es creado. Los mensajes que salvamos con el comando **save** automáticamente son marcados y borrados del mailbox del sistema, si salimos de mailx con la opción "q".

Para hacer una copia del mensaje en un folder provocando que el mismo sea borrado del mailbox del sistema y salvado en el archivo mbox (si se sale con "q"), usamos el comando **copy (co)**:

?co +clase

Podemos usar el comando **folder** para dirigir correo al contenido de un folder diferente. Por ejemplo,

?fold +clase

Este comando le indica a mailx leer el contenido del folder clase.

Para averiguar cual folder se está usando actualmente, usamos el comando **fold**:

?fold

Para listar el grupo actual de folders, usamos el comando **folders**:

&folders

Para empezar mailx leyendo uno de los folders, podemos usar la opción -f. Por ejemplo:

```
% mailx -f +clase
```

Este comando provoca que mailx lea el folder clase sin tomar en cuenta el mailbox del sistema.

La Utilería write.

La utilería **write** nos permite tener una conversación en-línea (interactiva) con otro usuario en el sistema, desplegando los comentarios que se escriban en la pantalla de su terminal.

Para tener una conversación en-línea:

En el prompt del sistema tecleamos:

Función

write usuario

Empieza la conversación con un "beep" en la terminal del receptor y desplegando un mensaje que informa que un mensaje será desplegado.

Tecleamos el texto del mensaje. Puede ser de cualquier longitud.

^D

Al principio de la línea, finaliza esta parte de la conversación.

Usando write para conversación en Dos-Sentidos.

El comando **write** es conveniente para comunicación en un-sentido que no requiere respuesta, pero en la mayoría de las ocasiones cuando usamos **write** vamos a querer un conversación en dos-sentidos. Para hacer ésto el receptor del mensaje debe escribir de regreso, realizando el mismo procedimiento que el que manda el mensaje pero en su propia terminal.

Para evitar confusiones que se puedan presentar en la conversación (mezcla de mensajes del emisor y del receptor), se sugiere que los

Comunicándose con otros usuarios

usuarios establezcan un protocolo (procedimiento estandar) para sus conversaciones, en el que cada usuario espera a responder hasta que la otra persona indica que su mensaje está completo.

Cuando ejecutamos el comando **write**, el sistema busca en su lista de usuarios en sesión para encontrar a la persona a la que se le está escribiendo. Si el nombre no se encuentra en la lista, recibimos un mensaje de error diciendo que ese usuario no está en sesión.

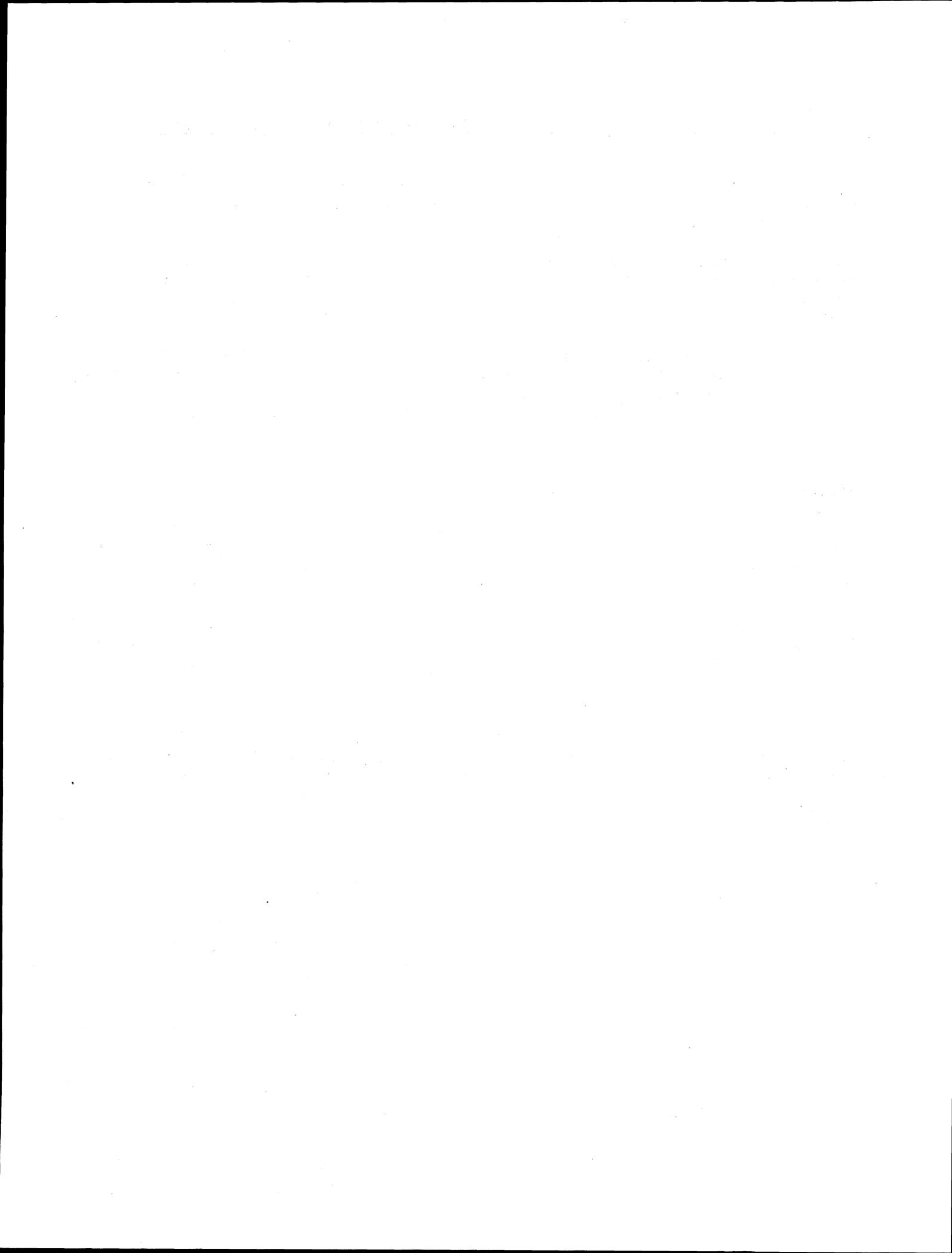
Nota: El sistema no revisa que el usuario especificado sea un usuario válido. Por lo tanto, si damos un nombre de usuario incorrecto, es decir sin cuenta en el sistema, se notifica que el usuario no está en sesión, no que no existe tal cuenta.

Negando la escritura de mensajes.

Es posible prohibir que otros usuarios nos manden mensajes a la terminal en que estamos trabajando. Si tratamos de escribir a una persona que ha deshabilitado el permiso de escritura sobre su terminal, recibimos el mensaje de error de "**Permission denied.**".

Para hacer ésto, es decir no permitir que otros usuarios nos distraigan con mensajes que interrumpen, se utiliza el comando **mesg n** en el archivo **.login** (o desde la línea de comandos). Para que podamos recibir mensajes, se introduce **mesg y**.

Si no estamos seguro de que otros usuarios pueden o no mandar mensajes a la terminal en que estamos trabajando, tecleamos **mesg** sin argumento. La respuesta es "**is y**" si estamos recibiendo mensajes, y es "**is n**" si no estamos recibiendo mensajes.



A. El editor ed

Sumario de los Comandos del Editor de línea ed

Comando	Función
% ed /bin/ed.hup	Recobra la sesión de edición si el trabajo fue salvado en /bin/ed.hup debido a que se halla "trabado" la terminal.
Invocando y saliendo de ed:	
% ed archivo	Lee una copia de un archivo existente en el buffer de la sesión de edición y despliega el número de caracteres leídos.
q	Sale de ed sin salvar el contenido del buffer.
w	Escribe (salva) el contenido del buffer en el archivo nombrado.
Obteniendo ayuda:	
h	Despliega el significado del mensaje de error "?"
f	Despliega el nombre del archivo actual.
u	Cancela el último comando.
Desplegando texto:	
p	Despliega la línea actual del buffer.
.=	Despliega el número de línea actual.
5	Despliega la línea 5
5n	Despliega la línea 5 y el número de línea.
1,5p	Despliega de la línea 1 a la 5.

UNIX

-3 o +3 Despliega 3 líneas arriba o abajo de la línea actual.

1,\$p o ,p Despliega de la línea 1 hasta la última línea del buffer.

1,\$n o ,n Despliega de la línea 1 hasta la última línea del buffer, y los números de línea.

Agregando texto:

a Agrega o crea nuevo texto, acabando con el apuntador de línea ".", en el buffer.

li Inserta texto, acabando con el apuntador de línea ".", antes de la línea 1.

Borrando texto:

d Borra la línea actual.

1,3d Borra de la línea 1 a la 3.

4,\$d Borra de la línea 4 hasta la última línea.

Moviendo líneas de texto:

1m3 Mueve la línea 1 después de la línea 3.

1,4m8 Mueve de la línea 1 a la 4 después de la línea 8.

Copiando líneas de texto:

1t3 Copia la línea 1 después de la línea 3.

1,4t8 Copia de la línea 1 a la 4 después de la línea 8.

Cambiando líneas de texto:

1,3c Reemplaza de la línea 1 a la 3 con el nuevo texto escrito acabando con el apuntador de línea ".".

Buscando texto:

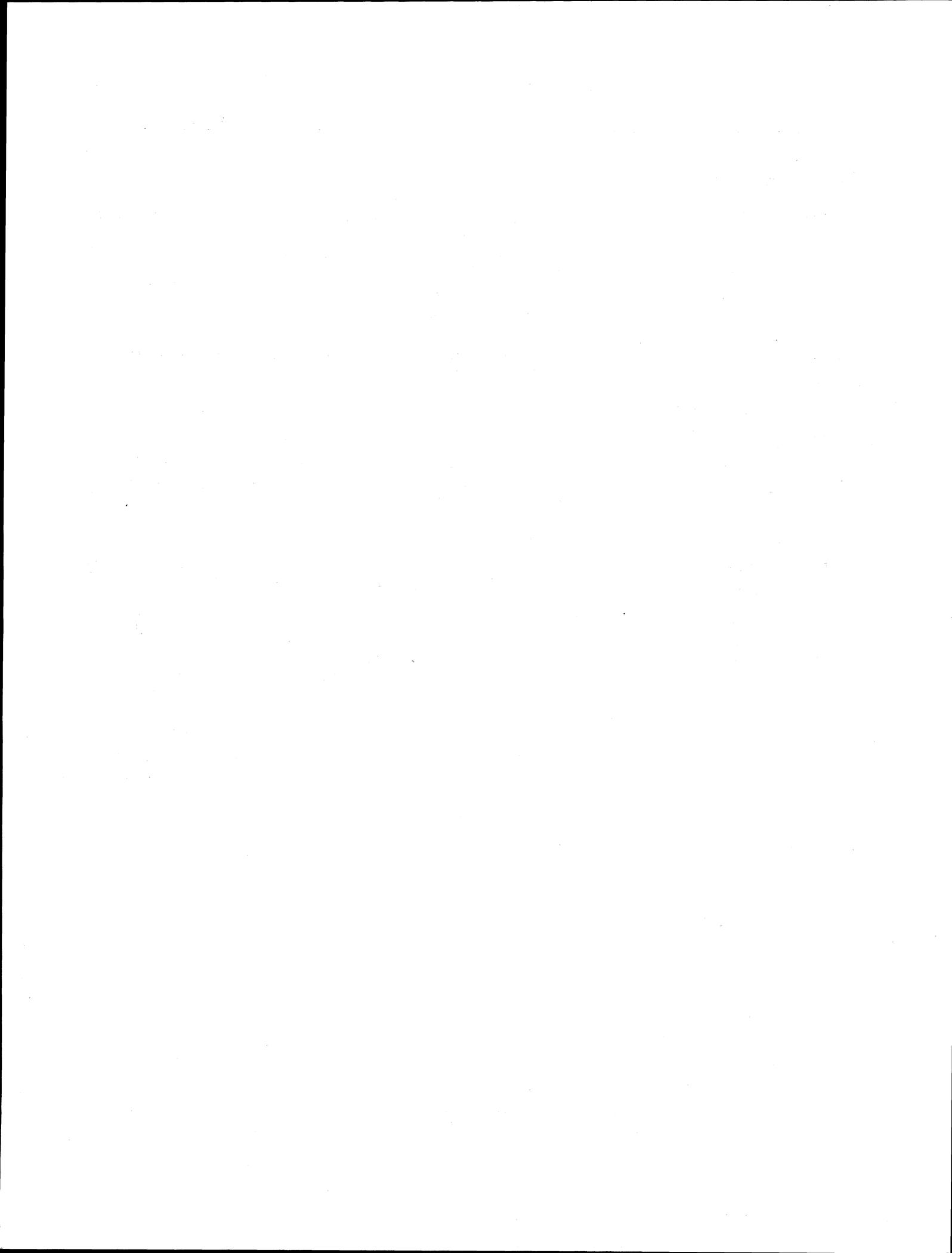
- /patron Encuentra la primera ocurrencia de **patron** hacia adelante, y despliega la línea si lo encuentra o "?" si no.
- g/patron Encuentra todas las ocurrencias de **patron** hacia adelante.
- ?patron Encuentra la primera ocurrencia de **patron** hacia atrás.

Substituyendo texto:

- s/texto_viejo
/texto_nuevo Encuentra la primera ocurrencia de **texto_viejo** y la substituye con **texto_nuevo**.
- s/texto_viejo
/texto_nuevo/g Substituye cada ocurrencia de **texto_viejo** con **texto_nuevo** en la línea actual.

Leyendo texto en el buffer:

- e *archivo* Lee una copia del archivo nombrado en el buffer, y despliega el número de caracteres leídos.
- r *archivo* Agrega una copia del archivo nombrado al archivo que está en el buffer, y despliega el número de caracteres leídos.
- f *archivo* Renombra el archivo actual.
-



**Esta obra se terminó de imprimir
en diciembre de 1993
en el taller de imprenta del
Departamento de Publicaciones de la
Secretaría de Servicios Académicos
Facultad de Ingeniería
Ciudad Universitaria, México, D.F.,
C.P. 04510**

**El tiraje consta de 100 ejemplares
más sobrantes de reposición.**

