EVALUACION DEL PERSONAL DOCENTE

CURSO: SISTEMA OPERATIVO UNIX PARTE I - COMANDOS Y UTILERIAS
FECHA: DEL 1 AL 22 DE FEBRERO DE 1994.

CONFERENCISTA	DOMINIO DEL TEMA	USO DE AYUDAS AUDIOVISUALES	COMUNICACION CON EL ASISTENTE	PUNTUALIDAD
ING. JOSE A. CHAVEZ FLORES (COORD)				
ING. EDWING. NAVARRO PLIEGO				
ING. ERNESTO SILVA ACEVEDO				

EVALUACION DE LA ENSEÑANZA

1.- ORGANIZACION Y DESARROLLO DEL CURSO:

2.- GRADO DE PROFUNDIDAD LOGRADO EN EL CURSO:

3.- ACTUALIZACION DEL CURSO:

4.- APLICACION PRACTICA DEL CURSO:

EVALUACION DEL CURSO

CONCEPTO	CALIF.
CUMPLIMIENTO DE LOS OBJETIVOS DEL CURSO	
CONTINUIDAD EN LOS TEMAS	
CALIDAD DEL MATERIAL DIDACTICO UTILIZADO	

ESCALA DE EVALUACION: 1 A 10

Introducción

La importancia del establecimiento de estándares dentro del desarrollo de sistemas de computación es indiscutible. En los últimos años, la tecnología de hardware ha evolucionado notablemente, a tal grado que en 1994 podemos encontrar microcomputadoras con velocidades de procesamiento de datos comparables a las de las poderosas mainframes y estaciones de trabajo que muchas veces sobrepasan las capacidades de las minicomputadoras. Este hecho ha provocado que la necesidad de intercambio de información sea más grande, lo que ha ocasionado el surgimiento de las famosas redes de computadoras, que permiten la interconexión remota de computadoras de diferentes tecnologías.

Dentro de este panorama, es necesario establecer una serie de estandáres 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.

Eejemplos de sistemas abiertos lo son: **TCP/IP** (*Transmission Control Protocol/Internet Protocol*) en el área de protocolos de comunicación entre computadoras y a **X-Window System** dentro de las interfases gráficas 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 dé 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.

1. 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 PDP-7 de 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 la 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 sistema UNIX captó inmediatamente la atención de los investigadores de la AT&T quienes 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.

El primer sistema UNIX fue la versión seis que constituyo un desarrollo interno de los Laboratorios Bell en el año 1977. 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 1981, 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.

La figura 1 describe el árbol genealógico del sistema operativo UNIX.

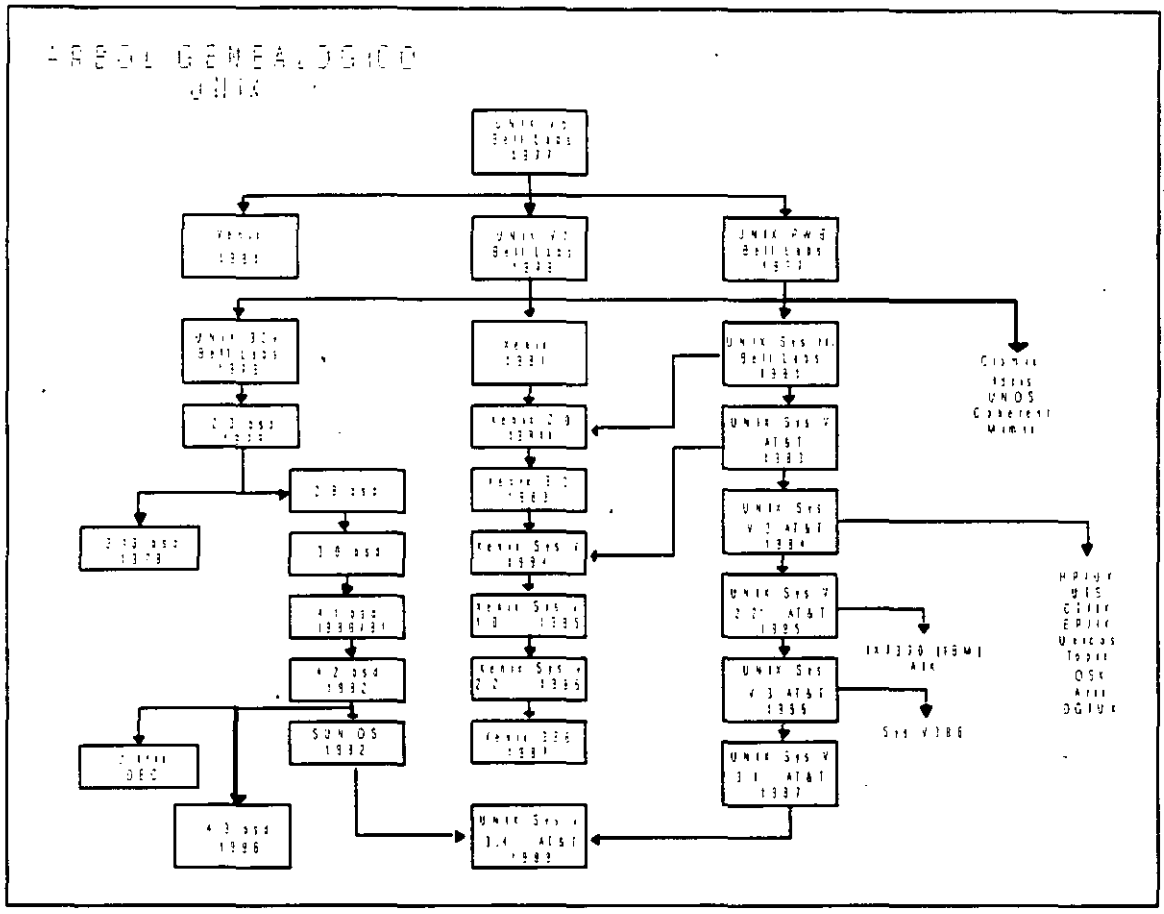


Fig. 1. Arbol genealógico de UNIX

2. Características de UNIX

- 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.

- Redireccionamiento, filtros e interconexiones.

- Sistema de archivos sencillo y eficiente.

- Un manejo de archivos consistente.

- La interfase con los dispositivos periféricos se maneja igual que un archivo.

- Esconde la arquitectura del hardware que lo utiliza.

- Es fácil de utilizar.

3. Componentes de UNIX

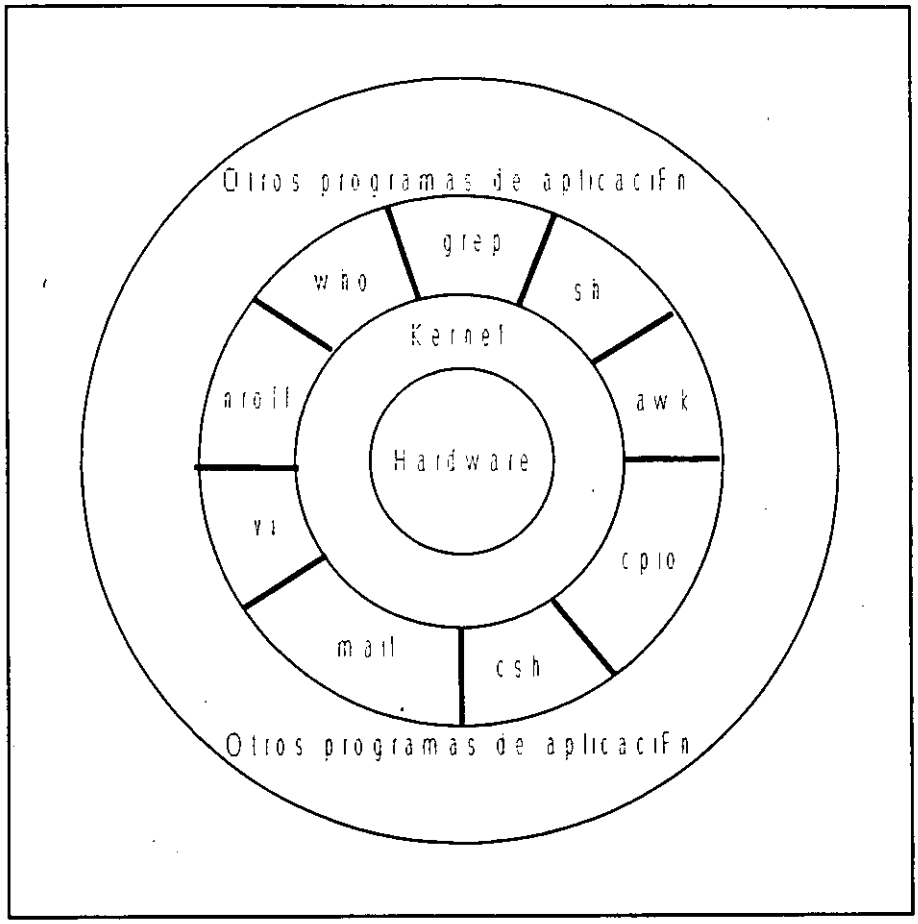


Fig. 2. Arquitectura de UNIX

Kernel o núcleo: esta parte del sistema operativo es la que se comunica directamente con el hardware. Entre sus funciones se pueden mencionar las siguientes: planificación de tareas, administración de recursos del sistema, administración de procesos y manejo de memoria.

Utilerías y programas de aplicación: Se encargan de ejecutaruna variedad de rutinas y funciones especiales de mantenimiento del sistema. Estas utilerías se comunican con el kernel por medio de una interfase desarrollada en lenguaje C que se conoce como **llamadas al sistema**. Muchas de las utilerías y programas forman parte de la configuración estándar de UNIX y son conocidos comúnmente como comandos.

Shell: Este programa se ejecuta inmediatamente después de que se abre una sesión de UNIX, es un proceso que interpreta los comandos que teclea el usuario y ejecuta las acciones asociadas 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, C-shell, Reduced shell, Korn shell* 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 los usuarios.

3. Estandarización de la interfase de UNIX

Actualmente se están desarrollando una gran cantidad de aplicaciones para ambientes UNIX, aplicaciones que van desde editores, hojas de cálculo, manejadores de bases de datos distribuidas, software de CAD/CAM, hasta aplicaciones tan especializadas como simuladores médicos y mecánicos, software gráfico de red, etc. Esto ha originado que los desarrolladores de software tengan necesidad de que la interfase para accesar los servicios del hardware por medio del sistema operativo sea estándar y de esta forma poder desarrollar más fácilmente sistemas portables de una plataforma de hardware a otra.

Los servicios que proporciona el kernel de UNIX se pueden accesar a través de una interfase en lenguaje C formada por funciones denominadas **llamadas al sistema**, dichas funciones deben de ser definidas de una forma semejante en cada una de las implementaciones de UNIX.

El estándar conocido como **IEEE 1003.1 POSIX** describe cada llamada al sistema en detalle, incluyendo una sinopsis, la descripción de la llamada, de los parámetros que recibe, los valores de regreso y los posibles errores que se generan.

El estándar de POSIX incluye llamadas al sistema para manejo de señales, intercomunicación de procesos, manejo del sistema de archivos, manejo de dispositivos, administración de la memoria, comunicaciones, etc.

Actualmente, como estrategia de migración a UNIX, muchos sistemas propietarios incluyen POSIX en su configuración base.

Capítulo 1
Inicio de sesión

Para poder trabajar en una máquina con sistema operativo UNIX, es necesario disponer de una clave de usuario y una contraseña asociada.

Una vez que se tenga acceso a una terminal, tal vez sea necesario presionar <RETURN> una o dos veces para establecer la línea de comunicación con la máquina; enseguida deberá aparecer el siguiente mensaje:

login:

Debido a que UNIX es un sistema multiusuario, lo primero que hay que hacer para trabajar en él, es identificarse; esto se lleva a cabo mediante un identificador único (el nombre de la clave asignada), de esta forma, si la clave es curso101, se deberá teclear esta después del mensaje de **login**:

login: curso101

si existen errores mecanográficos al momento de teclear la clave, estos se pueden corregir con el uso de la tecla <Backspace>; de lo contrario se deberá teclear la tecla <Return>, enseguida aparecerá un mensaje solicitando la contraseña:

login: curso101
Password:

Si por alguna razón la contraseña no se dio correctamente, el sistema desplegará otra vez el mensaje de login:

login: curso101
Password:
Login incorrect:
login:

Este procedimiento continúa hasta que se establece la contraseña correcta.

Cuando la clave y contraseña hayan sido aceptados, se desplegarán algunos mensajes iniciales, como los siguientes:

ULTRIX V4.0 (rev. 179) System #2: fri Nov 21 16:54:12 CST 1990

You have mail
Sun Feb 10 10:43:23 CDT 1992

%

Estos mensajes varían de acuerdo al tipo de sistema con el que se trabaje. Algunas líneas del mensaje pueden ser de bienvenida, algunas otras de configuración de la sesión o tal vez recordatorios.

Después de que finaliza el desplegado inicial, el sistema devuelve el control imprimiendo un prompt el cuál varía de acuerdo a la configuración de la sesión o al tipo de shell que se este utilizando, aquí utilizaremos el siguiente: **%**. Después de que aparece el prompt se estará a la espera de que se tecleen comandos para que se lleve cabo alguna acción.

El sistema UNIX es compatible con una gran variedad de terminales; sin embargo, es necesario configurar el medio ambiente del sistema operativo de acuerdo al tipo de terminal con que se esta trabajando; para ello se define una variable denominada TERM a la cual se le asigna el nombre o un pseudonimo de la terminal que se esta utilizando, si se esta utilizando C-shell se deberá teclear:

```
% setenv TERM hp2648
%
```

si se utiliza Bourne o korn shell:

```
% set TERM hp2648
% export TERM
%
```

Comenzaremos introduciendo algunos comandos básicos del sistema. Para obtener la fecha que tiene establecida el sistema se utiliza el comando **date**:

```
% date
Sun Feb 10 10:55:14 CDT 1992
%
```

El comando **who** nos dice quién está en sesión en ese momento:

```
% who
acf          tty01     Feb 10 08:35
curso101     tty05     Feb 10 10:43
jess         tty03     Feb  9 09:17
edwin .  tty11         Feb 10 07:12
ernesto  tty13         Feb  9 10:45
%
```

La primera columna indica el nombre del usuario, la segunda es el nombre del archivo asociado con la terminal desde la cual esta conectado el usuario. El resto de la línea indica la fecha en la que el usuario entró en sesión. Otra opción de who es:

```
% who am i
curso101     tty05  Feb 10 10:43
%
```

Generalmente en todos los sistema UNIX se cuenta con ayuda en línea. Si se desea obtener información acerca de determinado comando se puede consultar el manual con ayuda del comando **man**.

```
% man date
```

Una tarea importante al trabajar en un medio ambiente 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 la clave de usuario, inicialmente, no tiene contraseña, probablemente no se desplegará el mensaje de "Password:" al inicio de la sesión.

Para establecer o modificar la contraseña se utiliza el comando **passwd**.

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 al sistema.

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

```
% passwd
Enter new password:
Verify:
```

Para establecer una nueva contraseña, también se utiliza passwd; sin embargo es necesario saber la contraseña anterior, ya que el comando la requiere como entrada:

```
% passwd
Old pasword:
Enter new password:
Verify:
%
```

El comando para terminar una sesión de UNIX es **logout** (para C-shell) o **exit** (para Bourne y Korn shell).

LABORATORIO

1. Haga una sesión en UNIX.

2. Cambie su password.

3. ¿Cuantos usuarios estan en sesión?

4. Verifique la fecha del sistema.

5. Obtenga información del comando who.

Capítulo 2

El sistema de archivos

La estructura que permite organizar la información de UNIX se conoce como **sistema de archivos.**

- Es sencillo y fácil de utilizar

- No impone estructura alguna a los archivos, ni asigna significado a su contenido

- El contenido de los archivos depende únicamente del programa o utilería que lo utilice como entrada.

- Un archivo en UNIX puede contener cualquier tipo de caracteres y su estructura depende del uso que se le de. Puesto que los tipos de archivos no son determinados por el sistema de archivos, el núcleo no puede decirnos cuál es el tipo de un archivo en particular, ya que no lo conoce.

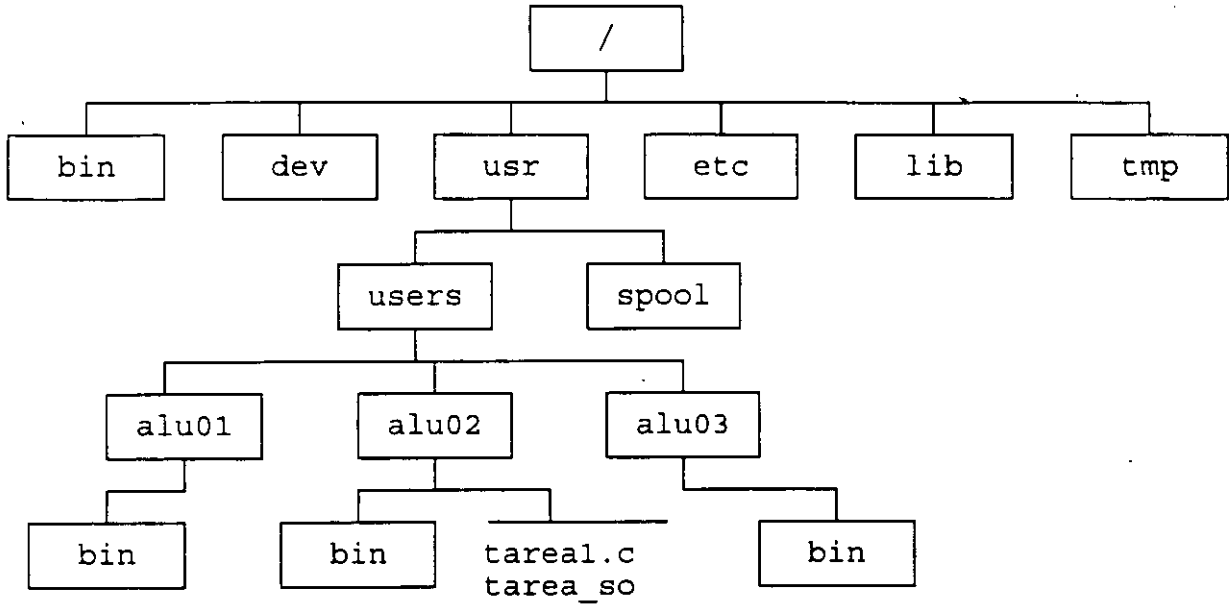
1. Características del sistema de archivos

- Una estructura jerárquica arborescente, en donde el nodo principal es el directorio llamado raíz (representado como "/") y cada uno de los niveles del árbol representan directorios.

- Consistencia en el manejo de archivos.

- Protección para archivos.

- Manejo de los dispositivos periféricos por medio de archivos.



Ejemplo de un sistema de archivos.

El nombre de un directorio o archivo contiene toda la ruta que se sigue para llegar a él, comenzando con el directorio principal ("/"). Nombres de directorios son: /etc, /usr/users/alu01/bin, /bin, /usr/spool.

Los directorios que se encuentran debajo del directorio raíz normalmente se encuentran en todas las implementaciones de UNIX.

/	directorio raíz del sistema de archivos.
/bin	utilerías y comandos estándar.
/dev	archivos de dispositivos.
/etc	archivos de administración y usos diversos
/lib	bibliotecas.
/usr	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.
/unix	archivo ejecutable del sistema operativo.
/usr/games	directorio de programas de juegos.

Archivos y directorios importantes.

2. Directorios y nombres de archivos.

Para nombrar a un archivo se deben tomar en cuenta ciertas convenciones:

- La longitud máxima del identificador es de 255 caracteres para las versiones de UNIX basadas en BSD y de 14 para las basadas en AT&T.

- Puede contener cualquier carácter. Se recomienda utilizar solamente letras, números y los caracteres "." y "_".

- Nombrar un archivo con espacios en blanco o los caracteres *, ?, [,], -, $, ', `, ", & y ! puede acarrear problemas, ya que estos tienen significado especial para el intérprete de comandos del sistema.

- Se hace distinción entre letras mayúsculas y minúsculas.

- 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 escondido.

Podemos determinar el tipo de un archivo con ayuda del comando **file**:

```
% file tarea datos script
tarea:          ASCII text
datos:          empty
script:         C_shell commands
% file archivo.txt
archivo.txt:    English txt
% file a.out
a.out:          pure executable
%
```

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 c-shell también tiene una marca especial al comienzo.

3. Creación de archivos

Al directorio de inicio de sesión se le conoce como **directorio de HOME** y es ahí donde el usuario puede comenzar a crear sus archivos y directorios ordinarios.

El **directorio de trabajo** es donde el usuario se encuentra en algún momento dentro del sistema de archivos. El comando **pwd** (*print working directory*) despliega el nombre del directorio de trabajo:

```
% pwd
/usr/users/alu01
%
```

Para cambiar el directorio de trabajo se utiliza el comando **cd** (*change directory*), por ejemplo:

```
% cd /bin
```

El comando cd sin parámetros retornará al usuario al directorio de HOME.

```
% cd /
% pwd
/
% cd
% pwd
/usr/users/alu01
%
```

Se puede omitir la ruta que se sigue para llegar al directorio de trabajo y solamente especificar la ruta establecida a partir del directorio de trabajo:

% cd bin

Con el ejemplo anterior se asume la ruta que se sigue para llegar al directorio de trabajo actual como parte del nombre del directorio al que se hace referencia, con ello se utiliza lo que se conoce como **rutas relativas** para nombrar directorios o archivos. Cuando se hace referencia a un archivo o directorio con todo su nombre comenzando con el directorio raíz, se utilizan **rutas absolutas.**

Al directorio de trabajo se le conoce también como **.** (punto), al **directorio padre** como **..** (dos puntos).

Ejemplo:

% cd ..

o

% cd ../..

otra opción para cambiarse a /usr/users/alu01/bin desde /usr/users/alu01:

% cd ./bin

Las rutas relativas se pueden utilizar al nombrar archivos y directorios con cualquier comando.

En el sistema UNIX, un archivo puede ser creado de muy diferentes formas, con un editor, un procesador de textos, o generado a partir de un programa. Una forma sencilla de crear un primer archivo, es con ayuda del comando **cat**:

```
% cat
 Este texto constituye una entrada de datos
 para el comando cat. Una vez que se de control-d, se
 marcara el fin de la entrada y esta será desplegada
 en pantalla.
ctrl-d
 Este texto constituye una entrada de datos
 para el comando cat. Una vez que se de control-d, se
 marcara el fin de la entrada y esta será desplegada
 en pantalla.
%
```

Se puede manipular el comando cat de tal forma que la entrada no se despliegue en pantalla, sino que se almacene en un archivo. esto se logra con el operador de redireccionamiento ">":

```
% cat > archivo1
 Este texto constituye una entrada de datos
 para el comando cat. Una vez que se de control-d, se
 marcara el fin de la entrada y esta será almacenada
 en el archivo con nombre archivo1.
 ctrl-d
%
```

4. El comando ls

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 /usr/users/alu01, ls listará los archivos que se encuentran en él:

```
% ls
tarea1.c
tarea_so
%
```

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

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

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

El comando **cat** también sirve para desplegar la información contenida en un archivo, esto se logra dándole como entrada un archivo de datos:

```
% cat archivo1
 Este texto constituye una entrada de datos
 para el comando cat. Una vez que se de control-d, se
 marcara el fin de la entrada y esta será almacenada
 en el archivo con nombre archivo1.
%
```

Otra forma de desplegar un archivo, pero por pantallas es con ayuda del comando **more**:

```
% more archivo1
```

El comando more tiene asociados una serie de comandos que se pueden invocar al momento que se esta desplegando un archivo, algunos de ellos son los siguientes:

COMANDO	FUNCION
<RETURN>	Despliega la siguiente linea
<space>	Despliega las siguientes 20 líneas
q	Salir de more
b	Despliega las 20 líneas anteriores
=	Despliega el número de la línea actual
h o ?	Ayuda

Resumen de comandos de more.

Ejemplos:

% ls -la

```
drwx——  3 alu01      33792 Sep  5  1990 .
drwxr-xr-x 64 root    5120 May  1  1990 ..
drwx——  1 alu01     33792 Sep  5  1990 .cshrc
drwx——  1 alu01     5120 Sep  5  1990 .exrc
drwx——  1 alu01     33792 Sep  5  1990 .login
-rwx——  1 alu01     5120 Sep  1  1990 .logout
-rwx——  1 alu01     51200 Sep  1  1990 .profile
drwxr-xr-x  2 alu01   33792 Apr  5  11:35 bin
-rwxr-xr-x  1 alu01    5120 Apr  1  19:20 tarea1.c
-rwxr-xr-x  1 alu01    51200 Apr  1  15:19 tarea_so
```

% ls -l

```
drwxr-xr-x  2 alu01   33792 Apr  5  11:35 bin
-rwxr-xr-x  1 alu01    5120 Apr  1  19:20 tarea1.c
-rwxr-xr-x  1 alu01    51200 Apr  1  15:19 tarea_so
%
```

La opción **l** del comando ls muestra la siguiente información:

- El primer carácter nos dice si se trata de un archivo ordinario (-), directorio (d), liga (l) o archivo de dispositivo (c).

- El segundo campo muestra los permisos para el archivo.

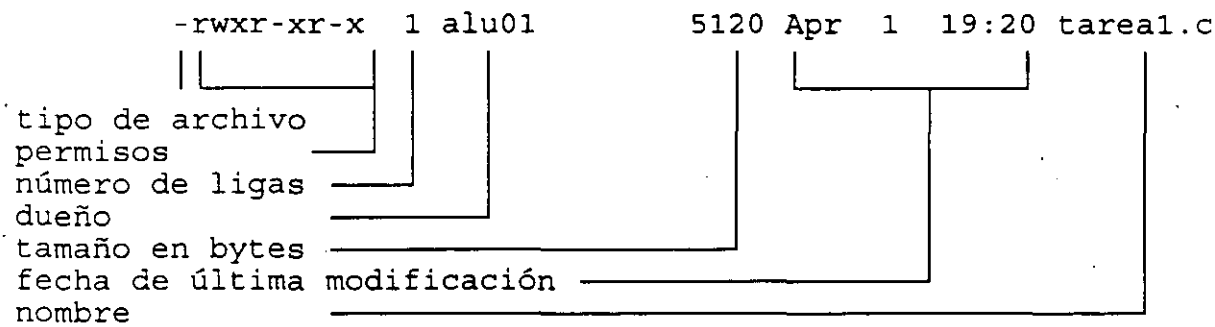
- Número de ligas del archivo.

- Dueño del archivo.

- Tamaño.

- Fecha de última modificación.

- Nombre del archivo. El significado de cada uno de los campos que se muestran en esta opción se indica en la figura 2.3.



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

Con la opción **a** se muestran archivos de propósito especial cuyo nombre normalmente comienza con . (punto) .

5. I-nodos

La representación y manejo interno de los archivos en UNIX se lleva a cabo mediante unas estructuras especiales llamadas **i-nodos**. Cuando un programa desea manipular un archivo, el núcleo lee de disco el i-nodo asociado con dicho archivo. Un i-nodo tiene la siguiente información referente a un archivo:

- Número del i-nodo.

- Identificador del usuario propietario del archivo.

- Identificador de grupo del propietario.

- Protecciones del archivo.

- Número de ligas al archivo. Las ligas son los diferentes nombres por los que se puede acceder a un archivo.

- Tamaño del archivo.

- Vector de direcciones de disco de los bloques que conforman el archivo.

- Fecha de la última lectura o ejecución del archivo.

- Fecha de la última modificación al archivo.

- Fecha de la última modificación del i-nodo.

Se puede acceder a un archivo físicamente por nombres diferentes. Un nombre de archivo constituye una liga mediante la cual se accesa la información que representa el archivo. Para crear una liga se utiliza el comando **ln**, que tiene la siguiente sintaxis:

ln [opciones] nombre1 [nombre2]

Para crear una liga para el archivo tarea_so se ejecuta el comando:

% ln tarea_so tarea_so.ln

Dos ligas a un archivo apuntan al mismo i-nodo, ls con la opción **i** nos permite obtener el i-nodo asociado a cada liga:

```
% ls -li
6320 drwxr-xr-x  2 alu01      33792 Apr  5  11:35 bin
6616 -rwxr-xr-x  1 alu01       5120 Apr  1  19:20 tarea1.c
5738 -rwxr-xr-x  2 alu01      51200 Apr  1  15:19 tarea_so
5738 -rwxr-xr-x  2 alu01      51200 Apr  1  15:19 tarea_so.ln
%
```

El primer campo de los registros desplegados por el comando anterior indica el i-nodo asociado con la liga y el tercero el número de ligas del archivo (para tarea_so y tarea_so.ln es 2).

6. Manipulación de archivos

Analicemos algunos comandos que nos permitirán manipular archivos y directorios en el sistema de archivos de UNIX. El primer comando nos permite, **cp**, nos permite hacer una copia de un archivo:

```
% cp tarea1.c tarea1.bak
```

para copiar a un directorio:

```
% cp tarea1.c bin
```

El comando cp también acepta una lista de archivos a copiar; en este caso el último argumento especifica el lugar donde se desean copiar y debe ser un nombre de directorio:

```
% cp tarea1.c tarea_so bin
```

En los comando para manipulación de archivos se puede hacer uso de **metacaracteres**:

```
% cp * bin
% cp tarea?.c bin
% cp tarea[0-9][0-9].c bin
```

Para cambiar el nombre a un archivo se utiliza el comando **mv**, que cambia el identificador para un archivo, conservando el mismo i-nodo:

```
% mv tarea1.c tarea2.c
```

El comando **rm** borra archivos. Estrictamente hablando lo que hace rm es borrar ligas. Si un archivo tiene varias ligas, el archivo puede ser accesado por medio de cualquiera de sus ligas restantes, el archivo se borra realmente cuando se borra su última liga. Se puede especificar una lista de archivos a borrar:

```
% rm tarea1.bak
% rm ./bin/*
```

Cuando un directorio se encuentra vacío, este se puede borrar con el comando **rmdir**:

```
% rmdir bin
```

El comando rm nos permite borrar toda la estructura de árbol que se encuentra debajo de un directorio:

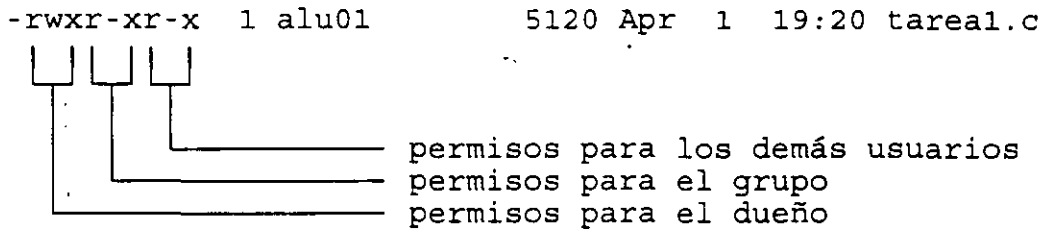
```
% rm -r bin
```

6. Protección de archivos

Todos los archivos y directorios del sistema tienen muchos atributos asociados además de su nombre. El comando ls nos permite ver algunos de ellos:

```
% ls -l
drwxr-xr-x  2 alu01      33792 Apr  5  11:35 bin
-rwxr-xr-x  1 alu01       5120 Apr  1  19:20 tarea1.c
-rwxr-xr-x  1 alu01      51200 Apr  1  15:19 tarea_so
```

Los permisos se indican en el segundo campo:



Los tres caracteres de cada grupo de permisos indican permisos de lectura, escritura y ejecución.

Cuando un archivo no tiene algún permiso, este se indica con -

De esta forma, la cadena rwxr-xr-x indicaría que el dueño tiene permisos de lectura, escritura y ejecución sobre el archivo; los usuarios pertenecientes al grupo del dueño tiene permisos de lectura y ejecución, pero no de escritura; y los demás usuarios solamente tienen permisos de lectura y ejecución.

Para cambiar los permisos de un archivo del cual se es dueño se utiliza el comando **chmod** (*change mode*). Para indicar los permisos se utilizan los caracteres **u**, **g** y **o** para permisos de dueño, grupo y otros usuarios respectivamente; también se utilizan los caracteres **w**, **r** y **x** para indicar permisos de escritura, lectura y ejecución. El carácter = significa establecer un permiso, + aumentarlo y - para suprimirlo. Ejemplo:

```
% chmod u=rwx,g+w,o-x tarea1.c
```

Otra forma más sencilla es ver los permisos como una secuencia de 9 bits asociados con cada permiso. Cada uno de los grupos de tres bits se pueden representar con un número decimal no mayor a 7.

Con la notación anterior, un 7 para cada grupo (777) indicaría conceder todos los permisos al dueño, grupo y demás usuarios, en el comando chmod se indicaría de la siguiente forma:

```
% chmod 777 tarea1.c
```

Para establecer los permisos a rwxr—— utilizaríamos la secuencia de dígitos octales 740.

LABORATORIO

1. Haga una liga para el archivo nombres (ln nombres nombres.ln). Una vez creada la liga, con ayuda del comando ls, obtenga la información del inodo y tamaño del archivo al cual apuntan nombres y nombres.ln. ¿Qué observa?

2. Lista el contenido de nombres.ln, enseguida modifique nombres y vuelva a listar el contenido de nombres.ln. ¿Qué concluye?

3. Copie nombres.ln a un archivo con nombre nombres.bak. Con ls obtenga el tamaño e i-nodo de dichos archivos..

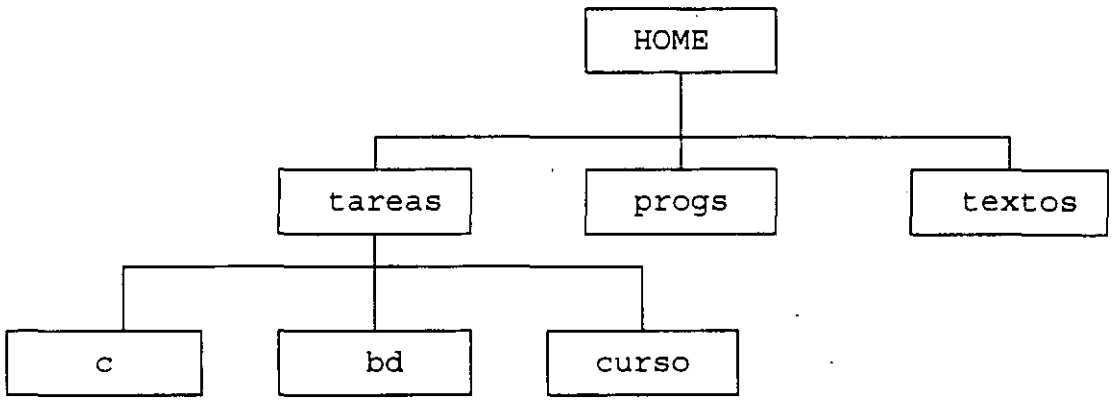
4. Con el comando mv renombre nombres.ln a nombres. Con ls obtenga el tamaño e i-nodo de nombres y comparelos con los que se habían obtenido para nombres.ln.

5. ¿Qué concluye con respecto a los comandos cp, mv y ln?

6. Por medio de redireccionamiento obtenga un archivo llamado dir.bin que contenga la información de los archivos que se encuentran en el directorio /bin.

7. Obtenga una copia de /etc/passwd. Dicha copia será el archivo passwd en su HOME.

8. Haga la siguiente estructura de archivos con los comandos adecuados:



Capítulo 3

Redireccionamiento, 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** o **pipes**, 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 sencillos que leen alguna entrada, realizan una serie de transformaciones sobre esta y generan una salida sin modificar la entrada original de datos.

1. 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).

Cuando se modifica la salida estándar de datos se dice que se esta haciendo un **redireccionamiento de salida**. Ejemplo:

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

El contenido del archivo generado se puede conocer de la siguiente forma:

```
% cat listado
drwx——  3 alu01   33792 Sep  5  1990 .
drwxr-xr-x 64 root   5120 May  1  1990 ..
drwx——  1 alu01   33792 Sep  5  1990 .cshrc
drwx——  1 alu01   5120 Sep  5  1990 .exrc
drwx——  1 alu01   33792 Sep  5  1990 .login
-rwx——  1 alu01   5120 Sep  1  1990 .logout
-rwx——  1 alu01   51200 Sep  1  1990 .profile
drwxr-xr-x  2 alu01  33792 Apr  5  11:35 bin
-rwxr-xr-x  1 alu01   5120 Apr  1  19:20 tarea.c
-rwxr-xr-x  2 alu01   51200 Apr  1  15:19 tarea_so
-rwxr-xr-x  2 alu01   51200 Apr  1  15:19 tarea_so.ln
```

Es posible con ayuda del comando cat almacenar el contenido de varios archivos en uno sólo:

```
% cat tarea1 tarea2 tarea3 > tareas
%
```

El operador de redireccionamiento > > funciona de manera semejante a como lo hace >, excepto que > > no reemplaza el contenido del archivo a donde se redirecciona la salida, sino que conserva su contenido y además anexa al final del archivo la salida que se esta redireccionando.

Ejemplo:

```
% cat tarea1 tarea2 tarea3 > > tareas
%
```

El **redireccionamiento de entrada** se da cuando se sustituye la entrada estándar por un archivo.

Ejemplos:

```
% cat < tarea1.c
/*  Tarea No. 1

        Programa que manda un letrero de "hola mundo"
*/

#include <stdio.h>

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

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

2. 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 estos datos y genera una salida sin alterar la entrada original.

El comando **tail** es un filtro que despliega las últimas diez líneas del flujo de datos de entrada.

Ejemplo:

% tail tareas

El comando tail también permite que se le indique cuantas líneas se desea desplegar.

% tail -20 tareas

También se puede utilizar tail para desplegar el contenido de un archivo a partir de una línea específica:

% tail +10 tareas

El comando **head** funciona de una forma semejante a como lo hace tail, solamente que head despliega las diez primeras líneas del flujo de datos de entrada:

% head tareas

También se puede indicar cuantas líneas se desean desplegar:

% head -20 tareas

Otro filtro interesante es **wc** (*word counter*), el cuál permite hacer un conteo de las palabras, líneas y caracteres de un flujo de datos proporcionado como entrada.

Ejemplo:

```
% wc tarea1.c nomina
10      20      130 tarea1.c
5       40      153 nomina
15      60      283 total
%
```

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:

l	cuenta líneas
w	cuenta palabras
c	cuenta caracteres

Ejemplo:

```
% wc -lw tarea1.c
10      20 tarea1.c
%
```

Para los ejemplos posteriores se utilizarán los siguientes archivos:

ARCHIVO: *nombres*

Antonio Chavez Flores
Norberto Arrieta Marquez
Jessica Briseño Cortez
Aaron Arcos Tapia
Ernesto Silva Acevedo
Edwin Navarro Pliego

ARCHIVO: *frutas*

manzana	4500
naranja	1500
pera	6000
platano	800
melon	2000
sandia	900
guayaba	1200
fresa	5000
mango	1100

El comando **sort** permite ordenar 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 especifico como llave de ordenamiento. Los registros de entrada son vistos como un conjunto de campos separados por espacios en blanco o tabuladores.

Ejemplo:

```
% sort +1 nombres
Aaron Arcos Tapia
Norberto Arrieta Marquez
Jessica Briseño Cortes
Antonio Chavez Flores
Edwin Navarro Pliego
Ernesto Silva Acevedo
%
```

El parámetro +1 especifica el campo que se tomará como llave del ordenamiento; el primer campo se indica como +0.

Otras opciones de sort se muestran a continuación:

OPCION	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.

Opciones de sort.

Ejemplos:

```
% sort +1 frutas
mango       1100
guayaba     1200
naranja     1500
melon       2000
manzana     4500
fresa       5000
pera        6000
platano     800
sandia      900
%
```

```
% sort +1n frutas
platano     800
sandia      900
mango       1100
guayaba     1200
naranja     1500
melon       2000
manzana     4500
fresa       5000
pera        6000
%
```

3. Interconexión de comandos

Al utilizar redireccionamiento de entrada/salida se vio 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, supongase que se desea obtener en pantalla el desplegado de los archivos que existen en el directorio /bin ordenados por su tamaño.

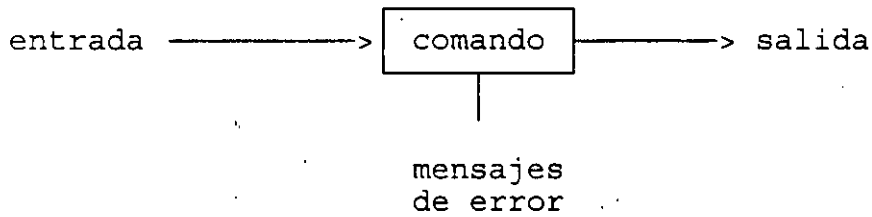
```
% ls -la /bin > temporal
% sort +3n temporal
```

El sistema operativo 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; de esta forma, para obtener el resultado anterior bastaría con ejecutar el siguiente comando:

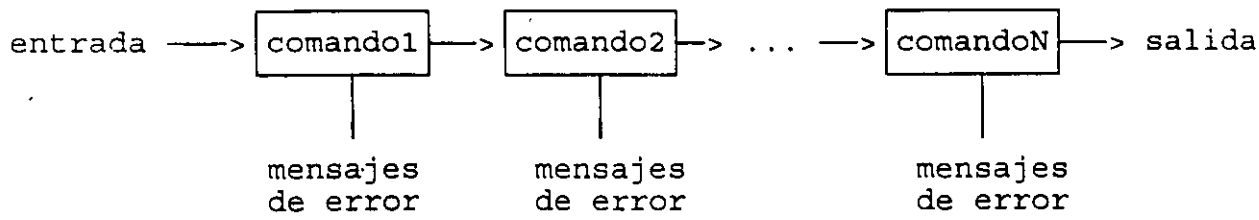
```
% ls -la /bin | sort +3n
```

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.

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 se interconectan 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.



Interconexión de comandos.

4. 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.

Ejemplo:

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

El patrón que se especifica en grep, se conoce como expresión regular.

Las expresiones regulares se especifican utilizando caracteres que tienen significado especial dentro de la misma expresión regular. A estos caracteres se les denomina metacaracteres, por ejemplo: ^ indica inicio de línea y $ fin de línea.

Cuando se utilizan metacaracteres en la expresión regular, esta debe encerrarse entre apóstrofes.

Ejemplo:

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

El metacaracter . se utiliza para representar a un carácter cualquiera, * indica cero o más repeticiones del carácter anterior.

Si se quiere especificar un subconjunto de caracteres se especifica este rango entre corchetes, por ejemplo, **[a-z]** representa a cualquier letra minúscula, **[a-zA-Z]** a cualquier letra mayúscula o minúscula.

Ejemplo:

Supongase que se desea 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 +1
Aaron Arcos Tapia
Norberto Arrieta Marquez
Jessica Briseño Cortes
Antonio Chavez Flores
%
```

El comando grep acepta las sigueintes opciones:

OPCION	FUNCION
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	No genera ninguna salida, salvo que se trate de mensajes de error.
v	Despliega las líneas que no se ajustan a la expresión regular proporcionada.

Tabla 3.2. Opciones de grep.

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.

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.

Metacaracteres para grep y egrep.

Ejemplo:

Suponga que se desea 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--)'
```

Capítulo 4

Edición de archivos

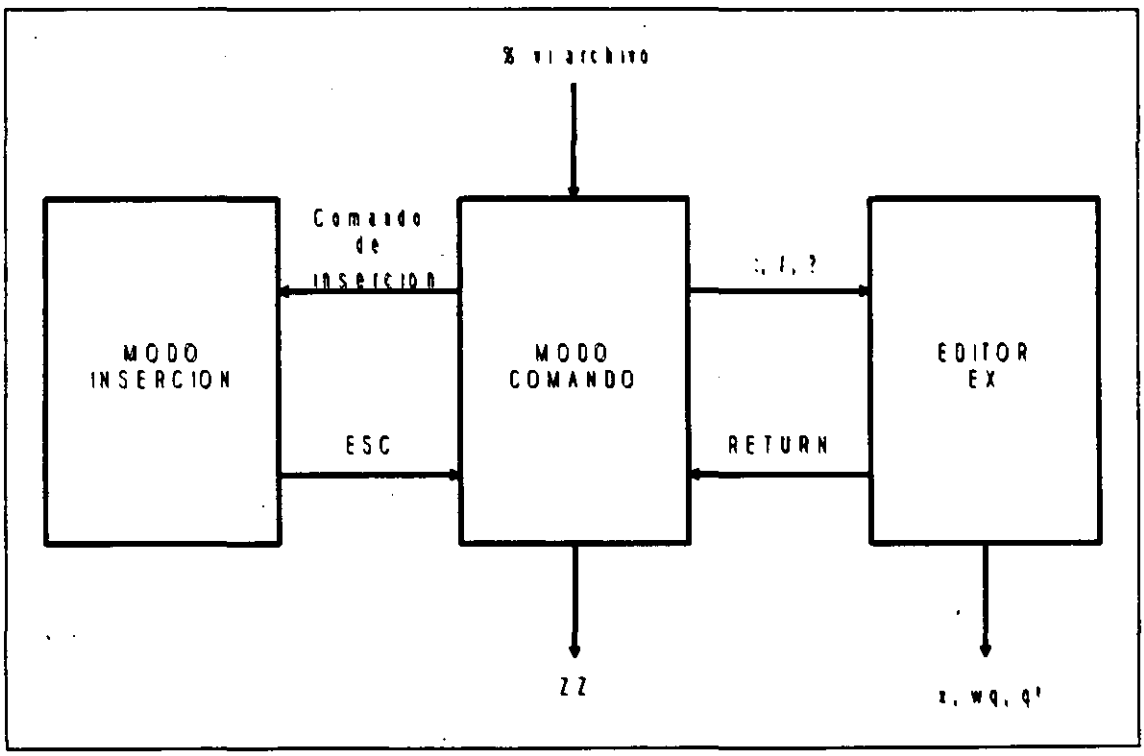
La mayoría de los sistemas UNIX poseen varios editores de texto con diferentes características; dos de los más populares son **vi** y **emacs**; sin embargo, el editor que forma parte de la configuración estándar de UNIX es **ed**, un editor de línea que es utilizado cuando se instala el sistema operativo.

Este capítulo describe el funcionamiento y configuración del editor vi; esto debido a que generalmente esta disponible en todos los sistemas con sistema operativo UNIX.

1. 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 desplazandolo a través de esta.

El editor vi sigue un esquema como el mostrado en la siguiente figura:



Esquema del editor vi.

La invocación para el editor vi es la siguiente:

% vi archivo

La siguiente tabla 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 se crea necesario.

COMANDO	DESCRIPCION
k, -	Mueve el cursor una línea arriba tratando de conservar la posición dentro de la columna en la que se encuentra.
j, RETURN. +	Mueve el curso una línea abajo.
h, Ctrl-h	Mueve el cursor hacia la izquierda.
l	Mueve el cursor hacia la derecha.
0	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.
Ctrl-f	Posiciona el cursor en la próxima página de texto. Se entiende por página de texto las líneas que aparecen en pantalla.
Ctrl-b	Mueve el cursor a la página anterior.

Comandos del editor vi para movimiento del cursor.

Comandos 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.
nr	Reemplaza n caracteres a partir de la posición del cursor.
ncw	Reemplazo n palabras, por el texto que se introduce, hasta abandonar el modo de inserción.
ns	Reemplaza n caracteres.
S	Reemplaza la línea sobre la que se encuentra el cursor.

Comandos de inserción y reemplazo.

Comandos de borrado:

COMANDO	DESCRIPCION
nx	Borra n caracteres.
ndw	Borra n palabras.
ndd	Borra n líneas.
nX	Borra n caracteres a la izquierda del cursor.
D	Borra toda la línea y la deja en blanco.

Comandos de borrado.

Cuando se borran caracteres, líneas o palabras con alguno de los comando 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. Los comandos de manejo de bloque se muestran a continuación:

COMANDO	DESCRIPCION
nyy	Guarda en la memoria temporal las n 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.

Comandos de manejo de bloques.

Comando de manipulación de archivos:

COMANDO	DESCRIPCION
:w	Archiva el texto actual.
:w nombre	Graba el texto en el archivo que se especifica.
:wq	Archivar el texto actual y salir 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 nombre	Edita el archivo que se especifica.
:e +nombre	Edita el archivo que se especifica y coloca el cursor al final del archivo.
:r nombre	Lee el archivo especificado y lo coloca a partir de la posición del cursor.
:!comando	Ejecuta un comando del shell y regresa a vi.

Comandos de manipulación de archivos.

Comandos de Búsqueda:

COMANDO	DESCRIPCION
/patrón	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.
?patrón	Igual que el comando anterior, solamente que la búsqueda es en sentido inverso.
n	Búsqueda de la siguiente ocurrencia del patrón.
N	Búsqueda de la siguiente ocurrencia del patrón en sentido inverso.

Comandos de búsqueda.

2. Configuración del editor

Dentro del medio ambiente del editor existen una serie de variables que ayudan a adaptar el editor a las necesidades del usuario. Para ver el valor de esas variables, en modo comando, se teclea:

:set all

A continuación se muestran algunas variables de configuración del editor.

VARIABLE	DESCRIPCION
autoindent	Permite que las línea de un párrafo conserve el margen de la primera línea del párrafo.
noautoindent	Desactiva la variable autoindent.
tabstop=num	Permite fijar la cantidad de espacios que representan la tecla <TAB>.
redraw	Cuando esta variable esta activada, el editor vi redibujará el texto después de un cambio como borrar una línea o insertar caracteres.
noredraw	Desactiva la variable redraw.
list	Esta variable permite que se visualicen los caracteres de control.
nolist	Desactiva la variable list.
number	Muestra los números de línea.
nonumber	Desactiva la variable number.
term=tipo	Establece un tipo de terminal para el editor.

Variables de configuración del editor vi.

El valor para cada una de estas variables se puede establecer desde la sesión de vi o bien, por medio del archivo de configuración .exrc. Para indicar que se desea activar una variable:

set variable

y para asignar un valor a una variable:

set variable = valor

Capítulo 5

Programación con AWK

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

La sintáxis de awk es la siguiente:

awk programa [lista_archivos]

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 }

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. Por otra parte, $0 identifica a todo el registro. El separador de campos por omisión es el blanco.

Cuando se utilizan expresiones regulares como patrones en awk, estas se encierran entre diagonales; por ejemplo, si se desea listar los nombres de los directorios en el directorio de trabajo actual, se podría ejecutar el siguiente comando:

```
% ls -la | awk '/^d/ { print $8 }'
```

Para el archivo nombres, mencionado en el capítulo 3, si se quiere obtener un listado de los nombres comenzando con sus apellidos, se podría teclear el siguiente comando:

```
% awk '{ print $2,$3,$1 }' nombres
Chavez Flores Antonio
Arrieta Marquez Norberto
Briseño Cortez Jessica
Arcos Tapia Aaron
Silva Acevedo Ernesto
Navarro Pliego Edwin
%
```

La salida de print puede ser dirigida a múltiples archivos; por ejemplo, se quieren mandar los apellidos del archivo nombres al archivo x y los nombres al archivo y, el comando a ejecutar sería:

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

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

printf cadena_formato, arg1, arg2, ..., argn

La cadena de formato contiene caracteres ordinarios, que son copiados a la salida, y especificaciones de conversión, cada una de las cuales causa la conversión de los siguientes argumentos sucesivos de printf. Cada una de estas especificaciones comienzan con % y terminan con uno de los caracteres mostrados en la siguiente tabla.

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

Conversiones para printf

Entre el % y el caracter de conversión puede aparecer, en orden:

- Un signo menos, que indica especificación 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 numérico, o el número máximo de una cadena de caracteres.

Ejemplo :

```
% awk '{printf"%3d %-10s %-10s %-10s\n",NR,$1,$2,$3}' nombres
  1 Antonio    Chavez     Flores
  2 Norberto   Arrieta    Marquez
  3 Jessica    Briseño    Cortez
  4 Aaron      Arcos      Tapia
  5 Ernesto    Silva      Acevedo
  6 Edwin      Navarro    Pliego
%
```

En el ejemplo anterior, se manda a la salida de datos la variable predefinida NR, la cual almacena el número de registro o línea que se procesa en ese momento. Cada vez que awk procesa una línea diferente NR 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	*Caracter separador de campos.*
RS	*Caracter separador de líneas de entrada (por omisión es el caracter de fin de línea).*
NR	*Número de la línea de entrada actual.*
OFS	*Caracter separador de campos de salida.*
FILENAME	*Nombre del archivo de entrada actual.*

Variables predefinidas en awk.

Existen dos patrones especiales en awk, el primero de ellos es BEGIN, con el cual se especifican las acciones que se deberán realizar antes de que se comience a procesar la primera línea de entrada; el otro 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.

Ejemplos :

1.

```
% awk 'END { printf "\t%d\n", NR }' nombres
        6
%
```

2.

```
% awk 'BEGIN { FS=":" } { print $1 }' /etc/passwd
```

1. 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 se manejan variables de tipo numérico y cadena; sin embargo, no es necesario definirlas, ya que se definen al momento de utilizarse.

Existen una serie de funciones para manipulación de variables, constantes numéricas o cadenas, estas se muestran en la siguiente tabla.

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

Funciones predefinidas en awk

Ejemplo :

Consideremos el ejemplo de un programa cuyo funcionamiento es igual al de wc cuando se le proporciona un flujo de datos de entrada proveniente de la entrada estándar o de un sólo archivo:

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

*por lo tanto se tienen los operadores + =, -=, * =, % = y / =.*

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, se utilizaría 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 no sea mayor a 80 caracteres sería:

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

¿que operadores se evalúan primero?

La presedencia de los operadores se muestra en la siguiente tabla:

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.*

Operadores en awk por orden creciente de precedencia.

2. 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.

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

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

Ejemplo :

El programa prog1.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. La forma de invocarlo, para el directorio de trabajo, es la siguiente:

```
% ls -la | awk -f prog1.awk
```

```
{       if ( NR == 1 )
                next
        if ( NR == 2 ) {
                min = $4
                max = $4
                archMax = $8
                archMin = $8
        }
        else    {
                if ( $4 > max ) {
                        max = $4
                        archMax = $8
                }
                if ( $4 < min ) {
                        min = $4
                        archMin = $8
                }
        }
}
END {
        printf "Archivo mas grande: %s %7.2f kbytes\n",          archMax,max/1000
        printf "Archivo mas pequenio: %s %7.2f kbytes\n",        archMin,min/1000
}
```

Ejemplo 2 :

Implementación con awk del comando wc.

```
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
}
```

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
```

Con la proposición while se puede implementar el comportamiento del for de la siguiente forma:

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

3. Arreglos

El lenguaje de programación de awk 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.

Ejemplo:

```
# Programa de ordenamiento de las líneas de entrada

{       linea[NR] = $0 }

END {
        for(i = 1;i < NR;i + +)
                for(j = NR;i < j;j–)
                        if (linea[j-1] > linea[j]) {
                                aux = linea[j]
                                linea[j] = linea[j-1]
                                linea[j-1] = aux
                        }
        for(i = 1;i < = NR;i + +)
                print linea[i]
}
```

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 ¿como se recorre el arreglo, si los índices no llevan un orden?.

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
```

Ejemplo :

```
# Programa que obtiene la frecuencia de las palabras
# de unt texto.


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

4. 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. El comando awk es una liga a nawk o oawk.

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

- Se pueden definir funciones.

- Se pueden hacer interconexiones de comandos en el programa.

- Se pueden borrar a tiempo de ejecución arreglos asociativos.

- El parser para el nuevo AWK elimina algunas ambiguedades que se permiten en la versión anterior.

Capitulo 6

Manejo de procesos

Como se ha mencionado anteriormente, UNIX es un sistema operativo multitarea, en donde aparentemente se pueden ejecutar varios procesos al mismo tiempo.

Decimos que aparentemente, ya que el sistema operativo solamente atiende a un procesos en pequeños intervalos de tiempo; sin embargo la rapidez con que atiende a cada uno de los procesos da la impresión de que todos los procesos se ejecutan al mismo tiempo.

En UNIX se puede tener cierto control sobre los procesos del usuario, es decir se puede monitorear su desempeño, controlar el tiempo de CPU que utiliza, suspender o terminar un proceso, etc.

¿Qué es un proceso?

Un proceso consiste de un espacio en memoria y un conjunto de estructuras de datos que lo identifican.

El espacio en memoria que el sistema operativo marca para uso exclusivo de un proceso contiene espacio para el código del programa, para las variables que el proceso utiliza y para el stack.

Las estructuras de datos, que son almacenadas en el núcleo del sistema operativo, contienen información como:

- El mapa de direcciones de memoria.

- El estado.

- La prioridad de ejecución.

- Registro de los recursos utilizados.

- El dueño del proceso.

Parámetros de un proceso

Muchos de los parámetros asociados a un proceso afectan directamente la ejecución de este: archivos que puede accesar, salida de datos, etc. estos se encuentran almacenados ya sea en el kernel en las estructuras de datos o bien en su espacio de memoria. A continuación se explican los más importantes.

Identificador de Proceso (PID)

El identificador de proceso (PID, *Process IDentification*) es un número único que el núcleo asigna a un proceso para su identificación. El número es asignado conforme los procesos se van creando.

Identificador del Proceso Padre (PPID)

El Identificador del Proceso Padre (PPID, *Parent Process IDentification*) es el PID del proceso que creó el proceso en cuestión.

Identificador del Usuario (UID)

El Identificador del Usuario (UID, *User IDentification*) es un número con el que se

identifica al usuario que creo el proceso. Este usuario es el dueño del proceso y es el único que puede cambiar los parámetros de este.

Efectivo Identificador de Usuario (EUID)

El Efectivo Identificador de Usuario (EUID, *Effective User IDentification*) es el UID utilizado para determinar los recursos que puede accesar el proceso.

Prioridad

La prioridad de un proceso determina el tiempo de CPU que dicho proceso puede utilizar.

El CPU determina el siguiente proceso a ejecutarse en base a las prioridades.

Para sistemas BSD las prioridades de un proceso están en el rango +19 a -19 y en sistemas ATT están en 40 a -20, en donde las prioridades más altas son las negativas.

Todos los procesos de usuario tienen un prioridad por omisión. El dueño de un proceso solo puede disminuir la prioridad de este

Un proceso solo puede cambiar su prioridad de acuerdo a como lo pueda su dueño.

Un proceso hereda la prioridad del proceso que lo crea.

Terminal asociada

Todos los procesos en UNIX tienen asociada una terminal desde la cual se llevan a cabo las operaciones de entrada y hacia la cual se dirigen las salidas del proceso (esto a menos que haya redireccionamiento).

Estados de un proceso

Debido a que un proceso no se encuentra en ejecución en todo momento, se definen cinco posibles estado para un proceso:

Ejecución: un proceso en ejecución es un proceso que ha adquirido tiempo de CPU en ese instante, así como también todos los recursos necesarios para entrar en operación.

Sleeping: un proceso en estado sleeping es aquel que espera la ocurrencia de un evento; mientras tanto, el proceso no puede solicitar tiempo de CPU.

Swapp: los procesos en estado de Swapp son aquellos que han sido transladados de memoria a disco. Este estado de un proceso se genera principalmente en sistemas que no tienen memoria virtual.

Zombie: los procesos Zombie son procesos que no tienen nada; pero

por alguna razón el kernel guarda información de ellos.

Stop: un proceso en estado de Stop es aquel que ha sido marcado por el núcleo para que no pueda entrar en ejecución a menos que se le mande una señal de continuación. Los procesos son enviados a este estado cuando se les manda una señal de STOP.

Manejo de señales

Las señales son el medio por el cual se pueden controlar los procesos o bien, se puede llevar a cabo una comunicación entre ellos.

Cuando un proceso recibe una señal, pueden suceder dos cosas: si el proceso cuenta con una rutina o manejador para esa señal, esta es ejecutada; de otra forma, el núcleo proporciona un manejador para esta señal.

Cuando existe una manejador para una señal se dice que la señal se "atrapa".

Para prevenir que deliberadamente se envíen señales a un proceso, este puede ignorar o bloquear ciertas señales. Existen dos señales que no pueden ser atrapadas, ignoradas o bloqueadas: KILL y STOP.

Existen 17 señales estándar para versiones AT&T y 31 para sistemas BSD, cada una de ellas tiene asociado un identificador y un nombre simbólico.

Número	Nombre	Descripción
2	SIGINT	Interrupción
3	SIGQUIT	Terminación
4	SIGILL	Instrucción ilegal
9	SIGKILL	Destrucción
10	SIGBUS	Error de bus
12	SIGSYS	Llamada al sistema con argumentos no válidos
14	SIGALARM	Alarma

Señales estándar para sistemas AT&T

Número	Nombre	Descripción
2	SIGINT	Interrupción
3	SIGQUIT	Terminación
4	SIGILL	Instrucción ilegal
9	SIGKILL	Destrucción
10	SIGBUS	Error de bus
12	SIGSYS	Llamada al sistema con argumentos no válidos
14	SIGALARM	Alarma
17	SIGSTOP	Detención
19	SIGCONT	Continuación
20	SIGCHLD	Estado del proceso hijo ha cambiado
25	SIGXFSZ	Tamaño de archivo excedido
29	SIGLOST	Recurso dañado

Señales estándar para sistemas BSD

Envío de señales

La comunicación entre procesos se puede llevar a cabo por medio del envió de señales.

El comando **kill** permite enviar una señal a un proceso en particular.

Para enviar una señal a un proceso, es necesario ser el dueño de dicho proceso, solamente el superusuario puede mandar señales a cualquier proceso.

La sintaxis del comando kill es la siguiente:

kill [-señal] PID

La señal se puede especificar por su número o nombre, PID es el Identificador del Proceso al que se le envía la señal. Si no especifica señal, se envía la señal 15 (TERM).

Modificación de la prioridad de un proceso

La prioridad de un proceso puede ser cambiada para aumentar o disminuir el tiempo de CPU que recibe.

Los usuarios normales solamente pueden decrementar la prioridad de sus procesos, el superusuario es quien puede cambiar la prioridad de cualquier proceso en cualquier sentido.

En sistemas BSD la prioridad de un proceso puede determinarse al momento de crear el proceso, con ayuda del comando **nice**, el cuál tiene la siguiente sintaxis:

nice [nivel] comando [argumentos]

Si no se especifica un nivel de prioridad se crea un proceso con prioridad 10.

Si un nivel de prioridad se especifica, este se añade el nivel por omisión para determinar el nuevo nivel de prioridad.

El nivel de prioridad por omisión es 0. Por ejemplo, para incrementar la prioridad a un proceso se deberá especificar un nivel negativo:

% nice -10 comando
%

Cuando se trabaja con csh, el comportamiento de nice varía un poco, en este caso el comando es opcional y el nivel de prioridad es interpretado relativo al nivel del proceso padre.

Cuando no se especifica proceso en el comando nice, se cambia el nivel de prioridad para el proceso de shell actual (por lo tanto para sus procesos hijos).

Cuando no se especifica un nivel de prioridad, se incrementa en 4.

En sistemas BSD, la prioridad de un proceso también puede cambiarse una vez que este se ha creado; esto se logra con ayuda del comando **renice**. La sintaxis para el comando es la siguiente:

renice nivel [-p PID ...] [-g pgroup] [-u usuario]

El nivel especifica el nuevo nivel de prioridad. Con la opción **p** se especifican los PID's de los procesos afectados, la opción **g** sirve para indicar un grupo, con lo cuál se afectan los procesos que pertenecen al grupo especificado. Por último la opción **u** permite listar el usuario para el cual sus procesos serán afectados.

En sistemas AT&T, el comportamiento del comando nice es diferente. En este caso el nivel de prioridad se interpreta de acuerdo al nivel de prioridad actual del proceso. Por ejemplo:

```
% nice -5 nomina
%
```

incrementa el nivel de prioridad en 5.

Si no se especifica nivel, este se incrementa en 10. Solamente el superusuario puede bajar el nivel de prioridad (en realidad se incrementa la prioridad de los procesos):

% nice –10 nomina

Cuando se especifican niveles de prioridad de -100 o menos, los procesos se ejecutan en tiempo real, es decir se les asignan todo el tiempo de CPU.

Procesos en paralelo

Cada vez que se proporciona un comando al shell, este lo interpreta, lo ejecuta y cuando termina devuelve el control al usuario para que este pueda proporcionar un segundo comando.

En realidad cada vez que se proporciona un comando al shell, este una vez que lo interpreta crea un proceso hijo para ejecutar el comando.

UNIX es un sistema operativo multitarea, por lo que se pueden mandar a ejecutar procesos al mismo tiempo (aunque en realidad no sucede así).

La forma de crear procesos en paralelo es con el terminador &.

Cuando se finaliza un comando con &, el shell crea un proceso hijo como es usual, pero no espera a que este termine, en lugar de ello, el shell muestra el PID del nuevo proceso y de inmediato muestra el indicador para otro comando:

```
% nomina &
2764
%
```

Si el proceso en paralelo requiere de alguna entrada, se le envía una señal de

STOP.

La salida generada por el proceso en paralelo siempre es enviada a la salida estándar mientras el usuario que arranco el proceso continué en sesión.

Cuando el usuario que inició el proceso paralelo termina su sesión de UNIX, el proceso paralelo (hijo del shell de inicio del usuario) es adoptado por el proceso *init* (cuyo PID es 1).

Cuando es adoptado el proceso, este sigue conservando todos sus parámetros, salvo que ya no tendrá una terminal asociada y su PPID será 1. Por otra parte, la salida generada por este proceso se perderá.

En sistemas AT&T cuando un shell es terminado se envía una señal de HUP a todos sus procesos hijos, incluyendo los que fueron ejecutados en paralelo.

Si se desea que un proceso paralelo continué después de que termina el shell desde el cuál se ejecutó, se deberá utilizar el comando **nohup** de la siguiente forma:

nohup comando &

Este comando ignora la señal NOHUP.

Monitoreo de procesos

El comando **ps** permite obtener información para los procesos existentes en el sistema.

El comando ps incluye opciones, las cuales pueden variar de sistema a sistema.

```
% ps
PID  TTY      TIME COMMAND
5643 console  0:02 -csh
6214 console  0:00 ps
```

```
% ps -elf
 F S     UID   PID  PPID  C PRI NI      ADDR   SZ    WCHAN    STIME TTY       TIME COMD
 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 lcsp
 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/nktl_daemon
 1 S    root   604     1  0 154 20 40013880   22 40044f22   Aug  3  ?        0:00 /etc/rlbdaemon
 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/snmpd
 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
```

La información que se presenta es, entre otra:

Estado

S	Durmiendo (Sleeping)
W	En espera (Waiting)
R	En ejecución (Running)
Z	Terminado (Terminated)
T	Detenido (Stopped)
X	Creciendo (Growing)

UID

PID

PPID

LABORATORIO

1. Cree tres nuevos procesos: csh, ksh y sh y analice la tabla de procesos.

2. Mande a ejecutar un proceso en background, anote su PID y PPID, termine su sesión y vuelva a crear una. Busqué el proceso en background y vea cuál es su PID y PPID ¿Qué diferencias encuentra?

3. ¿Puede mandar la señal 9 al proceso anterior? ¿Por qué?

Capítulo 7

Correo Electrónico y comunicación entre usuarios

Correo Electrónico

Uno de los servicios más utilizados dentro de un ambiente UNIX, es sin duda el correo electrónico.

Por medio de este servicio, los usuarios pueden establecer un canal de comunicación para envío de mensajes con usuarios de la misma máquina, o bien con usuarios de alguna otra maquina conectada por medio de una red.

El comando **mail** permite utilizar el servicio de correo electrónico, para enviar mensajes y leer los recibidos.

Las implementaciones de mail varían en los diferentes sistemas UNIX. Normalmente los sistemas BSD proporcionan una versión mucho más amigable para la lectura de mensajes recibidos; sin embargo, los sistemas de versiones AT&T incluyen en sus comandos una versión de mail muy semejante a las de los sistemas BSD denominada mailx (no confundir con xmail, que es la versión de mail para ambiente X-Windows).

1. Envío de mensajes

Para enviar un mensaje a un usuario, no es necesario que este se encuentre en sesión en ese momento, el daemon de mail guarda los mensajes dirigidos hacia un usuario en algún lugar para que este los pueda leer en algún momento que entre a sesión.

Para enviar un mensaje se deberá proporcionar el comando mail y el nombre del usuario destinatario, a continuación se ejemplifica el envío de un mensaje por medio de mail:

```
% mail jess
Subject: Proximo curso

    Le comunico a usted que el dia 25 de marzo del año en
curso inicia su curso de Dise&o de Bases de Datos. Esperamos
su puntual asistencia.

                        ATT.

                    J. Antonio Chavez

^D
%
```

El comando mail despliega un mensaje (Subject:) en donde se deberá indicar una especie de titulo para el mensaje. Quien recibe el mensaje tendrá en este titulo una especie de referencia acerca del contenido del mensaje.

Después de teclear dicho encabezado, se procede a proporcionar el texto del mensaje indicando el fin de este con Ctrl-d.

Una de las desventajas del procedimiento descrito anteriormente es que no se hace una edición del contenido del mensaje, es decir no se pueden corregir las líneas tecleadas anteriormente, solamente la línea actual.

Se puede crear un archivo con ayuda de un editor y posteriormente redireccionarlo como entrada al comando mail:

```
% mail jess < aviso
%
```

El mensaje enviado de esta forma no tiene asociado un encabezado para ello se puede utilizar la opción s:

```
% mail -s "Aviso de inicio de curso" < aviso
%
```

Otra alternativa es utilizar los comandos del modo tilde.

Los comandos tilde se invocan cuando se proporciona el mensaje indicándolos con una tilde (~) en la primera columna seguida del comando deseado.

COMANDO	FUNCION
~r archivo	Lee un archivo y lo incluye en el mensaje.
~v	Entra a un sesión de vi para editar el mensaje.
~!comando	Ejecuta un comando del shell.
~e	Edita el mensaje.
~w archivo	Escribe el mensaje al archivo.
~s texto	Establece un nuevo titulo para el mensaje.

Comandos del modo tilde.

Cuando se recibe el mensaje, este se guarda en un archivo perteneciente al destinatario del mensaje, para que este lo pueda leer en algún momento.

El directorio en donde se localiza este archivo depende de la implementación de la que se trate, en algunos es en /usr/mail en otros en /usr/spool/mail y tiene como nombre el login del usuario.

En sistemas AT&T el archivo de mensaje se llama mbox y se localiza en el HOME DIRECTORY del destinatario.

2. Lectura de mensajes

Cuando se recibe un mensaje por medio de mail, el sistema operativo lo notifica cuando se inicia una sesión de UNIX con el siguiente mensaje:

You have mail

Para leer los mensajes que se han recibido se utiliza el mismo comando mail sin argumentos. Después de ello aparecerá una lista con los mensajes recibidos:

% mail

```
Mail version 2.18 5/19/83.  Type ? for help.
"/usr/spool/mail/acf": 3 messages 2 new
    1 jess      Wed Jan 20 18:47  25/615 "Saludo"
>N  2 nam       Fri Feb 12 14:28  17/294 "libro shell"
>N  3 acf@cancun.cecafi.unam.mx Sat Feb 13 20:13  12/290 "Archivos"
>N  4 aaron     Fri Feb 19 11:32  23/315 "UNIXWORLD"
&
```

La lista de mensajes desplegada por mail incluye la siguiente información:

```
      N  2 nam    Fri Feb 12 14:28  17/294 "libro shell"

Estado
Num. de mensaje
Remitente
Fecha de llegada
Líneas/caracteres del mensaje
Asunto
```

El estado del mensaje puede ser alguno de los siguientes:

N Se acaba de recibir
U El mensaje no se acaba de recibir, pero no se ha leído
R Se acaba de leer

Cuando se despliegan los mensajes aparece el indicador '&', en ese momento se pueden introducir comandos de mail para la manipulación de los mensajes.

COMANDO	FUNCION
type [lista]	Despliega el contenido de los mensajes de la lista. Cuando no se proporciona una lista se despliega el mensaje actual indicado por '>'.
header	Despliega la lista de los mensajes.
h+	Despliega la siguiente página de lista de mensajes
h-	Despliega la página anterior de lista de mensajes
=	Muestra el número de mensaje actual
next	Despliega el siguiente mensaje
edit [lista]	Edita los mensajes de la lista o el actual
n	Despliega el mensaje n
save [lista] arch	Almacena los mensajes de la lista en el archivo
reply [mensaje]	Responde al emisor del mensaje, así como a los que lo recibieron
Reply [mensaje]	Responde únicamente al emisor del mensaje

Comandos de mail para lectura de mensajes.

3. Configuración de mail

El archivo .mailrc permite la configuración de una sesión de mail, este archivo se lee cada vez que se envian mensajes.

El archivo .mailrc debe estar localizado en el directorio HOME.

Este archivo puede contener entre otras cosas, definición de variables, editor, alias, etc.

Las variables se establecen de la siguiente forma:

```
set ask
set askcc
```

Hay dos tipos de opciones que podemos fijar con el comando set:

1. Prender una opción:

```
set opción
```

2. Asignar un valor a una opción:

```
set opción=valor
```

Para apagar una opción: unset opción

Además de personalizar la cuenta de mail, podemos usar el archivo .mailrc para definir un alias.

Un alias es un identificador que representa a uno o varios destinatarios.

Para definir un alias, en .mailrc:

```
alias instructores  jessica  antonio  edwin  ernesto
```

De esta forma el comando:

```
% mail instructores < listaGrupo
%
```

envía por mail el archivo listaGrupo a los usuarios jessica, antonio, edwin y ernesto.

Comunicación entre usuarios

En un ambiente multiusuario y tal vez de redes, es muy usual que un usuario quiera establecer comunicación con otro que se encuentra trabajando en la misma máquina.

UNIX permite establecer una comunicación interactiva con usuarios de la misma máquina, y de otras que se encuentren conectadas a la red. Esta comunicación interactiva muchas veces suple a las llamadas telefónicas.

El comando **write** envía un mensaje a un usuario que tenga establecida una conexión con el sistema. El mensaje es tomado originalmente de la entrada estándar:

```
% write nam
        Mensaje Urgente, puedes contestarme?
%
```

Si en ese momento, el usuario al que se le envía el mensaje (en este caso nam) ha dado permisos para que le manden mensajes a su terminal, el recibirá el siguiente mensaje no importando la actividad que este realizando:

```
Message from acf (tty01) [Fri May 13 12:12.56]
        Mensaje Urgente, puedes contestarme?
```

Aunque el usuario destinatario recibe un mensaje, este no se mezcla con lo que este realizando en ese momento.

Hasta este momento la comunicación es solamente de acf a nam.

Si el usuario destinatario no desea que le envíen mensajes puede establecer permisos de no escritura sobre la terminal en la que se encuentra trabajando, esto se puede llevar a cabo mediante el siguiente comando:

```
% mesg n
%
```

Sin embargo, el comando **mesg** no existe en algunos sistemas y por lo tanto el comando equivalente sería aquel que cambiará los permisos a la terminal en la que se encuentra trabajando:

```
% chmod 700 /dev/tty05
%
```

Si el usuario destinatario desea establecer una comunicación en dos direcciones lo puede hacer por medio del mismo comando write:

% write acf

Cual es tu mensaje URGENTE!!!
Tengo mucho trabajo.

%

Si el usuario que inicio la conversación no ha terminado el comando write, puede una vez que reciba el mensaje de contestación mandar un segundo mensaje; el destinatario original puede también enviar un segundo mensaje.

De esta forma los dos usuarios pueden establecer una conversación.

Una vez establecida una conversación los dos usuarios tienen que acordar en establecer un protocolo para especificar la terminación de los mensajes, de otra forma los mensajes que envía uno se pueden mezclar con los que se reciben haciendo difícil la lectura y escritura de estos.

La entrada de write se puede direccionar o bien puede ser tomada de una interconexión de comandos.

Algunos sistemas BSD incluyen el comando **talk** que permite llevar a cabo una conversación, pero con una interfase más amigable que evita el tener que establecer un protocolo de terminación de mensajes.

Capitulo 8

Manejo de periféricos

El respaldo de la información es una tarea básica del administrador del sistema.

Sin embargo, en algunas ocasiones es necesario que como usuarios podamos realizar nuestros propios respaldos en el momento deseado ya sea a unidades de cinta o disco.

En la mayoría de los sistemas UNIX es posible, además de hacer respaldo con los comandos propios del sistema operativo, almacenar información en formatos reconocidos por otros sistemas operativos, por ejemplo MS-DOS.

Para hacer respaldos y/o transferir información desde un sistema UNIX hacia una unidad de almacenamiento secundario se puede hacer uso de los comandos tar (tape archiver), cpio (copy input-output).

Como se menciono en el capítulo "El sistema de archivos", para todos los dispositivos conectados a la computadora (terminales, unidades de cinta, unidades de disco, impresoras, etc.) existe asociado un archivo de interfase que determina la forma en como el sistema operativo accesa dicho dispositivo.

Generalmente estos archivos se encuentran en el directorio /dev.

Los dispositivos sobre los cuales se pueden llevar a cabo respaldos pueden ser:

- Floppy diskettes
- Cartridge tapes
- TK's
- Data Cartridge

Antes de realizar un respaldo conviene conocer lo siguiente:

1.- El archivo asociado a la unidad de disco o cinta que deseamos accesar.

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.

A continuación se mencionan algunos ejemplos:

/dev/rmt0h	Unidad de cinta en Ultrix
/dev/rdsk/0s0	Unidad de disco flexible en HP-UX
/dev/rfd048ds9	Primera unidad de disco flexible de baja densidad (360 K) en SCO UNIX
/dev/rfd1135ds18	Segunda unidad de disco flexible de 1.44 Mbytes en SCO UNIX

En el caso de SCO UNIX, la información de los archivos de dispositivos se puede encontrar en el archivo /etc/default/tar.

Comando tar

El comando **tar** (tape archiver) permite crear y restaurar respaldos de archivos a y desde cintas magnéticas o discos flexibles.

Algunas opciones del comando tar 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.

Ejemplos:

% tar -cvf /dev/rfd0135ds8 *

% tar -cv *

% tar -cvf /users/sagitario/cinta *.c

% tar -tvf /users/sagitario/cinta

% tar -xvf /dev/rfd0135ds18

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.

-o	-i	DESCRIPCIÓN
-c	-c	Escribir el encabezado en formato ASCII. Si es usado con -o deberá de ser usado con -i.
-x	-x	Manejo de archivos especiales.
-v	-v	Despliega los archivos copiados.
	-u	Restaurar incondicionalmente los archivos especificados. Si el archivo ya existe, esta opción lo sobrescribe.
	-m	Retiene la fecha de modificación de los archivos.
	-d	Restaura la estructura de directorios según se necesite.

Por ejemplo, para respaldar en disco de 3 1/2" todos nuestros programas de C, podemos usar el siguiente comando:

```
% find . -name '*.c' -print | cpio -ocx > /dev/rfd0135ds18
```

Para restaurar los archivos de un respaldo, usamos el comando cpio de la siguiente manera:

```
% cpio -icxudm < /dev/rfd0135ds18
```

Si solo deseamos ver el contenido de una cinta o disco, usamos a cpio de la siguiente manera:

```
% cpio -ict < /dev/rfd0135ds18
```

Para restaurar un solo archivo de un respaldo, usamos el comando cpio de la siguiente manera:

```
% cpio -icxudm '*archivo*' < /dev/rfd0135ds18
```

Transferencia de archivos entre Unix y MS-DOS

En la mayoría de los sistemas UNIX existen comandos para transferencias de archivos de MS-DOS a UNIX y viceversa.

En SCO UNIX existen los comandos: dosdir, doscp, dosls, dosmkdir, etc., que permiten accesar la información de un disco flexible en formato MS-DOS, o bien la partición de MS-DOS.

Impresión de archivos

Para poder imprimir un archivo hacemos uso del comando **lp** seguido del nombre del archivo(s) que deseamos imprimir.

```
% lp *.c
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.

Si por alguna causa deseáramos cancelar el trabajo de impresión, podemos usar el comando **cancel** seguido del id que nos regresó el comando lp:

```
% cancel text-54
request "text-54" cancelled
%
```

Si solamente deseamos obtener información sobre el trabajo de impresión, damos el comando **lpstat** seguido del id de la impresión que nos interesa.

Capítulo 9

El shell y la administración del ambiente de trabajo

El shell de UNIX es la interface entre el usuario y el sistema operativo.

El shell es un programa que interpreta lo que teclea el usuario y ejecuta los comandos adecuados.

Existen varios tipos de shell, incluso el usuario puede crear su propio shell si la interface que se le proporciona no le agrada. Los shell's más populares son: Bourne Shell, C Shell y Korn Shell.

El administrador del sistema asigna a cada usuario un shell con el que trabajará al iniciar una sesión; sin embargo, eso no implica que no se pueda trabajar con alguno de los otros shell's existentes.

El shell proporciona una ambiente de trabajo, el cual se puede configurar y adecuar a las necesidades del usuario.

Los shell's más utilizados generalmente reconocen estructuras de control de flujo, variables, operadores, etc., de tal forma que se pueden crear programas a los cuales tradicionalmente se les conoce como **scripts.**

En el archivo /etc/passwd se registran las claves válidas en el sistema; en este archivo se indica el shell de entrada para cada una de las claves.

La información que contiene es la siguiente:

1. Nombre de la cuenta de usuario.

2. Password encriptado.

3. Identificador de Usuario (UID).

4. Identificador del Grupo de dafault (GID).

5. Información particular del usuario.

6. Directorio de HOME.

7. Shell de inicio.

Ejemplos de registros del /etc/passwd:

```
acf:g/GQFQc2mugls:280:15:J. Antonio Chavez F., CECAFI:/usr/cecafi/acf:/bin/csh
aaron:gfL..Qc2mugxw:315:15:Aaron Arcos Tapia, CECAFI:/usr/cecafi/aaron:/bin/csh
jess:aN1/3jklmuped:516:15:C. Jessica Brisenio C., CECAFI:/usr/cecafi/jess:/bin/sh
nam:wpl109smn.sls:785:15:Norberto Arrieta M., CECAFI:/usr/cecafi/nam:/bin/csh
```

El campo siete no necesariamente tiene que ser un shell, puede ser cualquier programa, el cuál se ejecutará al momento que el usuario entre a sesión.

El usuario puede cambiar su shell de entrada con ayuda del comando chsh:

```
% chsh jess /bin/csh
%
```

En otras implementaciones de UNIX la forma de trabajar del comando chsh es la siguiente:

```
# chsh
Changing login shell for jess
shell [/usr/bin/ksh]: csh
```

Este comando no cambia inmediatamente el shell de la sesión, la siguiente que el usuario entre haga una sesión, su shell será diferente.

El shell es un programa ejecutable, cuando este se ejecuta se crea un proceso, el cuál será el proceso padre de todos los procesos que se generen en la sesión. Este proceso tiene como padre al proceso con PID 1 (INIT).

Debido a que el shell es un programa ejecutable, el usuario puede cambiar de shell simplemente ejecutando el shell, lo cuál creará un nuevo proceso:

```
% /bin/sh
$
```

```
$ /bin/csh
% /bin/ksh
$
```

Scripts

Un script de shell es un programa de shell, es un archivo que contiene instrucciones que pueden ser interpretadas por el shell.

Un script puede ser simplemente una secuencia de comandos, o bien un archivo que contenga sentencias que entienda el shell como: instrucciones de control de flujo, manejo de variables, expresiones con operadores, etc.

Los scripts de C shell deben comenzar con "#" en la primera columna de la primera línea que no sea de comentarios, esto es debido a que si no es así muy probablemente se interprete como un script de Bourne shell.

Cuando se trata de un script de Korn Shell, no existe problema ya que Bourne Shell y Korn Shell son muy semejantes.

Para poder ejecutar un script este debe de tener permisos de ejecución para quien lo desee ejecutar:

```
% chmod 711 script1
% script1
```

Ejemplo:

```
# /bin/csh
# Script de C shell para desplegar información del
# directorio de trabajo
#

echo -n "Número de directorios:"
ls -la | grep '^d' | wc -l
echo -n "Número de archivos:"
ls -la | grep '^[^d]' | wc -l
echo -n "Espacio ocupado:"
ls -la | awk '{x+=$5}END{print x}'
echo -n "La fecha es:"
date
```

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).

- El símbolo de número (#) es usado para introducir comentarios.

- 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 (;).

Variables

En una sesión de shell o en un script se pueden manejar variables.

La forma de crear una variable es la siguiente:

en csh:

% set variable=valor

en sh y ksh:

$ variable=valor

El valor de la variable puede ser número o una secuencia de caracteres (no se necesitan comillas a menos contenga espacios en blanco).

Las variables así definidas solamente tienen validez en la sesión de shell en donde se definan, es decir si se crean en un script solamente existen al momento que este se ejecuta; si estas se definen en una sesión, existen hasta que la sesión termine o bien hasta que se desactiven.

Los nombres de variables deben empezar con una letra y contener sólo letras, números, y subguiones. Pueden ser de hasta 20 caracteres de largo.

Con el comando set se muestan las variables que se tienen definidas:

```
% set
TERM=hp
HOME=/users/jess
PATH=(. /bin /usr/bin /users/sybase/bin)
HISTORY=20
%
```

Para hacer referencia al contenido de una variable, se utiliza el signo "$" antes de la variable referida. Este valor se puede utilizar en expresiones que acepte el shell. Para ver el contenido de una variable podemos utilizar el comando echo:

```
% set SALUDO="buenos dias"
% echo $SALUDO
buenos dias%
```

Para desactivar una variable se utiliza el comando unset:

```
% unset variable
```

Las variables se pueden "exportar", lo que significa que una variable puede ser utilizada desde los procesos hijos del shell en donde fué definida.

Para definir una variable de este tipo:

en csh:

```
% setenv variable valor
```

en sh y ksh:

```
$ variable=valor
$ export variable
```

La forma en como se accesan estas variables es idéntica a la forma en como se hace para variables normales.

Si se define una variable con el mismo identificador que una que proviene de un proceso padre, la primera es la que tiene validez en el proceso actual.

Suponga los siguientes scripts:

```
# script1
#
set x 10
echo $y
setenv y 20
echo $x
echo $z
script2
```

```
# script2
#
echo $y
echo $x
setenv x 30
echo $x
echo $z
setenv z 60
```

¿Qué salida se genera con la siguiente secuencia de comandos?

```
% setenv z 50
% script1
% echo $x
% echo $y
% echo $z
```

El usuario puede crear sus variables para configuración de su ambiente. Cuando se utiliza algún tipo de software normalmente hay que configurar el ambiente mediante la creación de ciertas variables.

Existen algunas variables que se inicializan al momento de login, otras se pueden inicializar en la línea de comandos y algunas otras se pueden inicializar en los archivos de configuración.

Existen un conjunto de variables que son utilizadas por el shell, estas son dependientes del shell que se este utilizando.

VARIABLE	DESCRIPCION
cwd	Directorio 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). C shell.
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.
noclober	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. Es importante incluir al directorio "." para poder ejecutar comandos del directorio de trabajo actual.
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
ps1	Prompt primario. Bourne Shell.
ps2	Prompt secundario. Bourne Shell.
LPDEST	Impresora por default.
TERM	Definición de terminal utilizada

Diferencias entre ' , " y `.

Estos caracteres tienen significado especial para el shell, aparentemente los tres pueden ser utilizados para encerrar una cadena de caracteres sin embargo tienen diferencias.

Los caracteres " y ' 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 apóstrofos 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 Mar 25 11:11:31 EDT 1993.
%
% set hoy=`date`
% echo "La fecha y hora actual es $hoy."
La fecha y hora actual es Mon Mar 25 11:11:31 EDT 1993.
% echo 'La fecha y hora actual es $hoy.'
La fecha y hora actual es $hoy.
```

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

- Los apóstrofos (') protegen de los caracteres especiales. Cualquier cosa encerrada en apóstrofos 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 la secuencia de caracteres como un comando.

Ejemplos:

```
% ls -la `awk 'BEGIN{FS=":"}{print $6}' /etc/passwd`
```

```
% set UID=`tail -1 /etc/passwd | awk 'BEGIN{FS=":}{print $3}'`
```

Archivos de configuración del shell

Cuando un usuario entra a sesión es recomendable que configure su medio ambiente, para ello existen una serie de archivos que llevan a cabo esta tarea. Además se cuenta con otros archivos que permiten configurar una sesión del editor vi o bien una de mail. Algunos de estos archivos son:

.cshrc	permite configurar un ambiente de csh.
.login	permite configurar el medio ambiente de inicio de sesión cuando el shell que atiende es csh.
.profile	tiene la misma función que .login pero cuando se trata de bourne shell o kshell.
.logout	permite ejecutar acciones cuando se termina la sesión, siempre y cuando el interprete de comandos sea csh.
.mailrc	configura una sesión de mail.
.exrc	configura una sesión de vi.
.hushlogin	es un archivo vació que impide que a un usuario le aparezcan los mensajes desplegados por /etc/motd.

Es importante que estos archivos se modifiquen para que configuren el medio ambiente de usuario adecuadamente, es conveniente:

- Incluir en el PATH los directorios de utilerías y comandos importantes.

- Definir variables como:

  - TERM
  - history
  - prompt

- Definir alias en los comandos rm y mv para evitar borrados indeseables:

```
alias rm "rm -i"
alias mv "mv -i"
```

- Establecer el prompt.

- Configurar la terminal adecuadamente.

- Mandar mensajes de bienvendida.

Ejemplos de estos archivos de configuración se muestran a continuación:

.cshrc

```
#
# Default user .cshrc file (/bin/csh initialization).
# 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 ( $?prompt ) then          # shell is interactive.
            set history=20            # previous commands to remember.
            set savehist=20           # number to save across sessions.
            set system='hostname'     # name of this system.
            set prompt = "$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
#         aliases for all shells
alias cd            'cd \!*; set prompt="$cwd> "'
alias cp            'cp -i'
alias rm            'rm -i'
alias mv            'mv -i'
alias pwd           'echo $cwd'
alias atl                   rlogin atl
set history=40
set ignoreeof
set prompt="$HOME>"
setenv SYBASE /users/sybase
setenv DSQUERY PRODUCCION
```

.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 /users/sybase/bin/sqr /users/sybase/bin
.)

#set up the terminal
setenv TERM vt100
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
banner "Bienvenido"
```

.profile

```
# @(#) $Revision: 66.1 $

# Default user .profile file (/bin/sh initialization).

# Set up the terminal:
        eval ' tset -s -Q -m ':?hp' '
        stty erase "^H" kill "^U" intr "^C" eof "^D"
        stty hupcl ixon ixoff
        tabs

# Set up the search paths:
        PATH=$PATH:.

# Set up the shell environment:
        set -u
        trap "echo 'logout'" 0

# Set up the shell variables:
        EDITOR=vi
        export EDITOR
```

.mailrc

```
unset askcc
set ask
set autoprint
```

.exrc

```
set autoindent
set nomesg
set number
set scroll=24
set shell=/bin/csh
```

Agrupación de comandos

Es posible mandar a ejecutar varios comandos a la vez separando los comandos por puntos y comas (;).

Se puede continuar escribiendo un comando en la siguiente línea, utilizando diagonales invertidas (\) al final de cada línea seguida de la tecla RETURN.

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.

Cuando se usan interconexiones (|) entre comandos en lugar de puntos y comas, solo es desplegada la salida del último comando.

¿Cuál es la salida de los siguientes comandos?

```
% ls -la | sort +5n | wc

% ls -la ; sort +5n ; wc

% date; pwd ; who

% date | pwd | who
```

LABORATORIO

1. Configure su medio ambiente de trabajo de la forma más adecuada.