FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

CURSOS INSTITUCIONALES

SCT DIRECCION GENERAL DE CARRETERAS FEDERALES

U N I X

12, 19, 26 Noviembre, 3 y 10 diciembre de 1994

- Material Didáctico -

Palacio de Minería
México, D.F.

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 1993 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. Se pueden mencionar a productos como **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 de paso a sistemas totalmente portables. El sistema operativo UNIX se ha convertido en uno de los entornos de programación más populares y utilizados del mundo. Actualmente, miles de computadoras, que van desde las microcomputadoras hasta las supercomputadoras, utilizan UNIX. Se puede afirmar que UNIX es el sistema operativo que se ha establecido como sistema abierto hasta el momento.

¿A qué se debe el éxito del sistema operativo UNIX? Además de que se cuenta con las especificaciones de diseño de UNIX, podemos distinguir dos razones principales por las cuales se ha hecho tan popular. Primero, esta escrito en lenguaje C, lo que lo hace portátil; segundo, es un buen sistema operativo, su ambiente de programación es de extraordinaria riqueza y productividad.

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

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 código de máquina del sistema resultó mayor que la versión en lenguaje ensamblador, pero parte del aumento se debió a la adición del apoyo a la multiprogramación y a la posibilidad de compartir procedimientos reentrantes.

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. La compatibilidad entre las versiones AT&T y BSD sigue siendo hasta cierto punto cuestionable. Las versiones BSD son equiparables con las versiones AT&T, ya que ambas incorporan las mejores innovaciones del otro sistema a sus propias versiones.

El primer sistema UNIX 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.

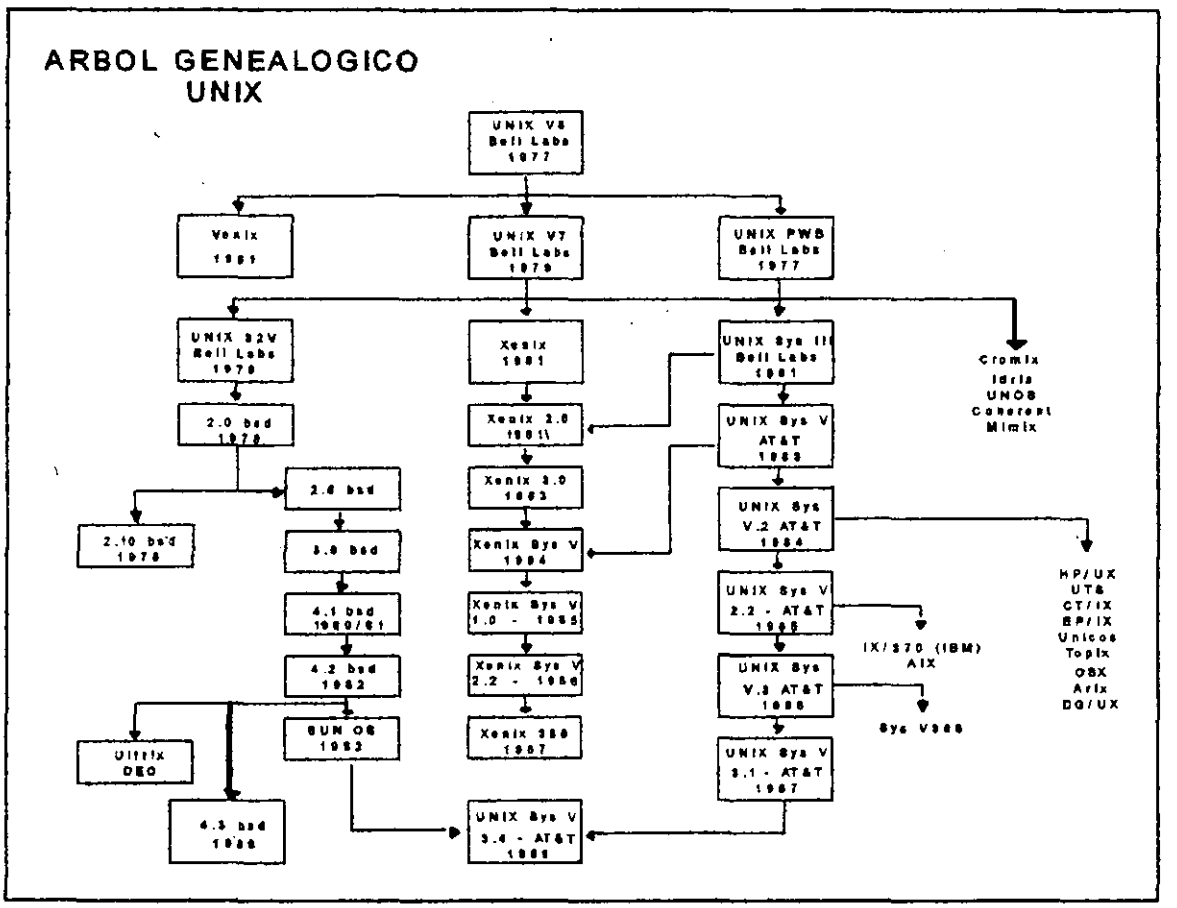


Fig. 1. Arbol genealógico de UNIX

2. Características y componentes de UNIX

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

- Sistema **multiusuario** y **multitarea**.

- Las especificaciones de diseño están disponibles públicamente, lo cual hace que se adapte a exigencias particulares.

- Esta escrito en un lenguaje de alto nivel, lo que lo hace portátil.

- Enfoque de programación.

- 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.

Los componentes o la arquitectura de alto nivel de UNIX se pueden representar por el diagrama de la figura 2. Si vemos el sistema como un conjunto de capas, el sistema operativo propiamente dicho conforma la capa más interna, esta parte del sistema operativo es la que se comunica directamente con el hardware y se le conoce comúnmente como **kernel** o **núcleo**. Entre las funciones del núcleo se pueden mencionar las siguientes: planificación de tareas, administración de recursos del sistema, administración de procesos y manejo de memoria.

En la siguiente capa se encuentran las utilerías y programas de aplicación vinculados con el sistema que se encargan de ejecutar una variedad de rutinas y funciones especiales de mantenimiento del sistema. Estas utilerías se comunican con el kernel por medio de 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.

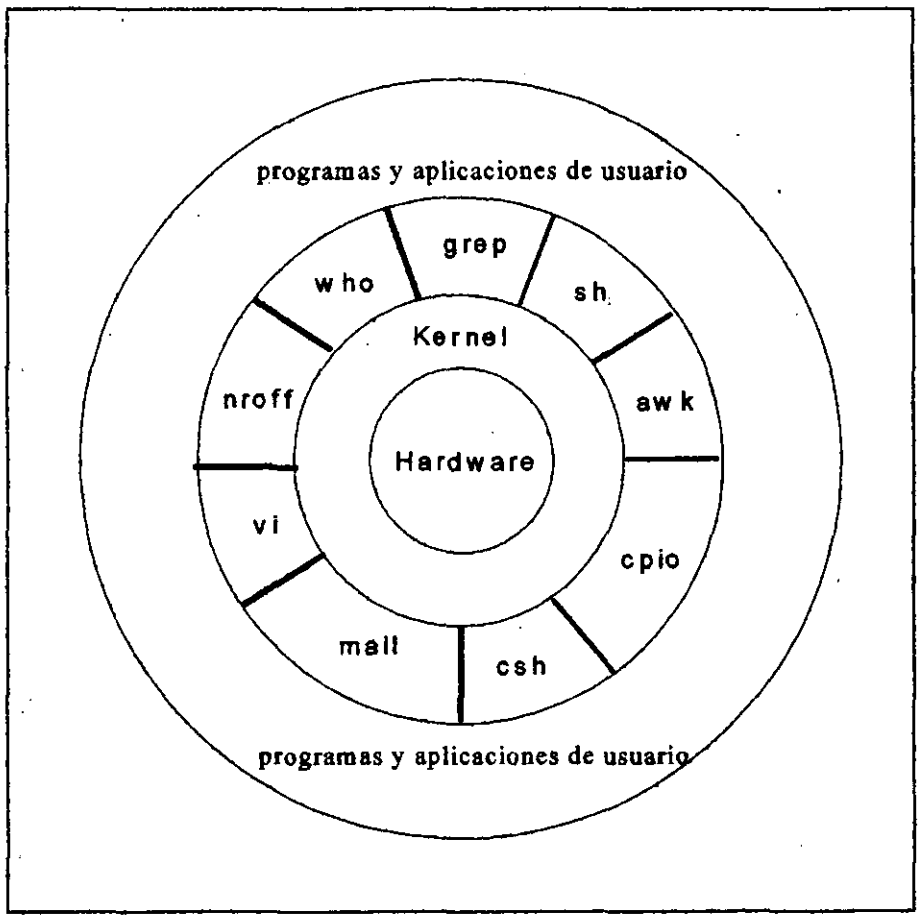


Fig. 2. Arquitectura de UNIX

Uno de los programas de aplicación más importantes es el intérprete de comandos o **shell**. Este programa se ejecuta inmediatamente después de que se abre una sesión de UNIX, es 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

En este capítulo se describirá la forma de iniciar una sesión en el sistema operativo UNIX, introduciendo algunos conceptos y comandos básicos.

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:

La contraseña sirve para identificarse como un usuario autorizado para utilizar la clave anteriormente tecleada. Al momento de introducir la contraseña, esta no se desplegará en pantalla; sin embargo los errores mecanográficos también se pueden corregir con la tecla <Backspace>. Una vez que se ha dado la contraseña, se debe teclear <Return>. 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
%
```

donde hp2648 es el pseudonimo de una terminal gráfica de Hewllett Packard.

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
%
```

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
%
```

que muestra la información del usuario que esta trabajando en la sesión desde la cuál se ejecuto el comando.

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**. Por ejemplo, para obtener información del comando date:

```
% man date
```

desplegará las hojas del Manual de UNIX relacionadas con dicho comando.

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 alguien descubre o conoce alguna contraseña, puede entrar a la clave asociada sin ningún problema y modificar la información del dueño de dicha clave e incluso cambiar la contraseña, con lo cual, el dueño legítimo de la clave tendrá negado el 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**. Es muy importante establecer una contraseña que el usuario pueda recordar fácilmente; pero que sea difícil de imaginar por los demás. Generalmente el sistema exige que la contraseña tenga al menos seis caracteres, es recomendable que esta contenga un dígito u otro carácter no alfabético, lo que proporciona un rango amplio y complejo de contraseñas, dando mayor seguridad 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:
```

en este momento se deberá teclear la contraseña que se desea establecer, la cual no se visualizará:

% passwd
Enter new password:
Verify:

enseguida se debe teclear la nueva contraseña una segunda vez, para verificar que se ha introducido correctamente. Si las dos entradas no coinciden, se rechazará la contraseña y esta no quedará establecida.

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 password:

Cuando se introduzca la contraseña actual correctamente, passwd permitirá establecer una nueva:

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

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

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 el sistema operativo.

2. Cambie su password.

3. Verifique que el password que acaba de establecer sea correcto.

4. ¿Qué usuarios estan conectados al sistema?

5. ¿Qué programa es el que se ejecuta durante su sesión y cuál es su función?

Capítulo 2

El sistema de archivos

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

El sistema de archivos de UNIX tiene las siguientes características:

- 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.

La figura 2.1 muestra la representación de árbol de un sistema de archivos común en UNIX. 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.

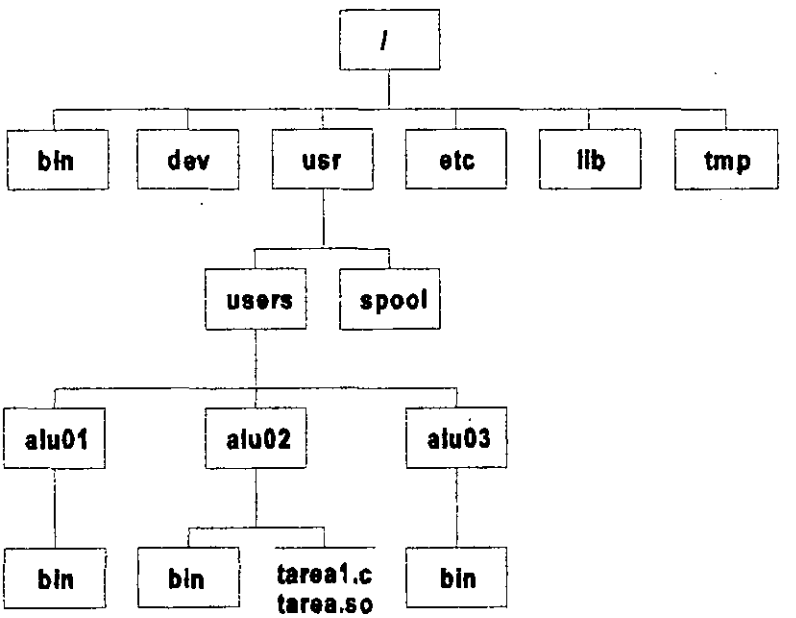


Fig. 2.2 Ejemplo de un Sistema de Archivos

Los directorios que se encuentran debajo del directorio raíz normalmente se encuentran en todas las implementaciones de UNIX. En la figura 2.2 se da una descripción general de algunos de los archivos y directorios más comunes e importantes.

/	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 reinicializar el sistema.
/etc/init	comando que crea el primer proceso en el procedimiento de boot.
/usr/include	archivos de encabezados.
/unix	archivo ejecutable del sistema operativo.
/usr/games	directorio de programas de juegos.

Fig. 2.2. Archivos y directorios importantes.

1. 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.

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

Ejemplo:

```
% 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.

2. Creación de archivos

Cuando se abre una sesión de UNIX, se comienza en un lugar dentro del sistema de archivos; el directorio de inicio de sesión se conoce como **directorio de HOME** y es ahí donde el usuario puede comenzar a crear sus archivos y directorios ordinarios. Si el sistema de archivos no impone restricciones, el usuario se puede cambiar a cualquier directorio dentro del sistema de archivos, éste nuevo directorio se conoce como **directorio de trabajo** y es el directorio en donde el usuario se encuentra dentro del sistema de archivos. El comando **pwd** (*print working directory*) despliega el nombre del directorio de trabajo:

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

Todos los archivos y directorios del sistema tienen asociados una serie de permisos de lectura, escritura y ejecución que pueden imponer restricciones cuando un usuario desea cambiarse a un directorio. Si un directorio no esta protegido de esta forma, uno se puede cambiar a él con el comando **cd** (*change directory*), por ejemplo:

```
% cd /bin
```

ocasiona que el directorio /bin sea el nuevo directorio de trabajo.

El comando cd sin parámetros retornará al usuario al directorio de HOME. La siguiente secuencia de comandos ejemplifica el retorno al directorio de HOME:

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

Cuando se desea cambiar de directorio de trabajo, tal vez sea demasiado latoso tener que especificar el nombre completo de un directorio, por ejemplo: si el directorio de trabajo es /usr/users/alu01 y se desea cambiar a /usr/users/alu01/bin. Cuando ello sucede 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 para llegar a un archivo o directorio cualquiera. Para el ejemplo planteado anteriormente, una opción podría ser la siguiente:

```
% cd bin
```

de esta forma 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**.

Otro nombre con el que se designa al directorio de trabajo es . (punto). El **directorio padre** de un directorio, es aquel directorio del nivel superior inmediato dentro de la jerarquía del sistema de archivos que forma parte de la ruta del directorio. Desde cualquier directorio, un directorio padre puede ser designado por .., esto implica que un directorio tiene asociado un nombre .. desde cada uno de sus directorios hijos. De esta forma, estando en /usr/users/alu01, se puede cambiar al directorio padre con el siguiente comando:

```
% cd ..
```

y al directorio /usr con:

% cd ../..

otra opción para cambiarse a /usr/users/alu01/bin desde su directorio padre sería:

% 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**, el cual recibe un flujo de datos de entrada (por omisión se proporciona desde el teclado, especificando el termino del flujo de datos al presionar simultáneamente las teclas <Ctrl> y <d>) y despliega dicha entrada en pantalla:

```
% 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.
%
```

El comando cat normalmente almacena en un buffer su entrada y la despliega por bloques. Si se cometen errores mecanográficos al momento de teclear la entrada, uno puede corregir solamente los de la línea actual antes de teclear <ENTER>. Esto se debe a que cuando un programa lee de la terminal, el núcleo envía al programa cada una de las líneas sólo cuando se da el carácter nueva-linea.

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
%
```

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.
%
```

Cuando se trata de un archivo grande, cat lo despliega en pantalla sin pausa alguna. Para detener el desplegado se puede utilizar la combinación de teclas <Ctrl><s> y <Ctrl><q> para continuarlo. Otra forma de hacerlo es con ayuda del comando **more**:

```
% more archivo1
```

more nos mostrará las primeras 20 líneas del archivo, para desplegar las siguientes 20 líneas se teclea la barra espaciadora, <ENTER> ocasiona que se muestre una línea más. Presionando **h** se obtiene una lista de comandos que se utilizan con **more**. Un resumen de ellos se muestra en la tabla 2.1.

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

Tabla 2.1. Resumen de comandos de more.

Un directorio, como ya se había mencionado, es un archivo, que contiene los nombres de los archivos que se encuentran bajo él. 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 ls, al igual que la mayoría de los comandos, tiene opciones que pueden ser usadas para alterar su comportamiento normal. Dichas opciones se especifican después del comando con un signo menos - seguido de una letra que hace referencia a su significado. En la mayoría de los comandos las opciones se pueden combinar.

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 nos da mayor información sobre los archivos y directorios. El primer carácter nos dice si se trata de un archivo ordinario (-), directorio (d), liga (l) o archivo de dispositivo (c). Luego se muestran los permisos para el archivo, el siguiente campo especifica el número de ligas del archivo, después aparecen los campo que indican el dueño del archivo, tamaño, fecha de última modificación y nombre del archivo. El significado de cada uno de los campos que se muestran en esta opción se indica en la figura 2.3.

```
-rwxr-xr-x  1 alu01      5120 Apr  1  19:20 tarea1.c
  1    2   3  4          5         6          7
```

1	tipo de archivo
2	permisos
3	número de ligas
4	dueño
5	tamaño en bytes
6	fecha de última modificación
7	nombre

Fig. 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) .

3. 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]
ln [opciones] lista_archivos directorio

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).

Se puede accesar y modificar el contenido de un archivo mediante cualquiera de sus ligas. Los archivos de directorio almacenan los nombres de archivos (ligas) que contienen, además de su i-nodo asociado.

4. 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
```

El primer argumento es el archivo que se desea copiar; el segundo es el nombre del nuevo archivo que es copia del primero. Si tarea1.bak ya existe, este se borra y luego se crea la copia (el sistema no manda ningún mensaje de advertencia). Si el segundo argumento es un directorio, en este se creará el archivo copia con el mismo nombre que el archivo original:

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

creará un archivo llamado tarea1.c en el subdirectorio bin del directorio de trabajo. 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
```

Se debe especificar cada uno de los archivos que se desean copiar, para hacer una copia de todos los archivos que se encuentran en un directorio hacia otro, se pueden utilizar ciertos caracteres que tienen significado especial para el shell y que se conocen como **metacaracteres**:

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

El primer comando copia todos los archivos del directorio de trabajo hacia el subdirectorio bin; el segundo copia todos los archivos cuyo nombre sea "tarea" seguido de un carácter y terminados con ".c" (tarea1.c, tareax.c, tareas.c, etc). El último comando va ha hacer una copia de los archivos cuyo nombre comience con "tarea" seguido de dos números (los corchetes especifican rangos de caracteres) y terminen con ".c" (tarea01.c, tarea23.c, etc.).

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

provoca que el nuevo nombre del archivo sea tarea2.c El comando mv también puede utilizarse para "mover" un archivo o varios hacia un directorio:

% mv tarea1.c ./bin/tarea2.c

El comando anterior mueve el archivo tarea1.c al subdirectorio bin, pero además su nombre en ese directorio será archivo2.c. Con mv se pueden mover una lista de archivos a un directorio, ya que obedece a las mismas reglas de sintaxis de cp.

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

Esta instrucción borra al subdirectorio bin solamente si se encuentra vacío. El comando rm nos permite borrar toda la estructura de árbol que se encuentra debajo de un directorio:

% rm -r bin

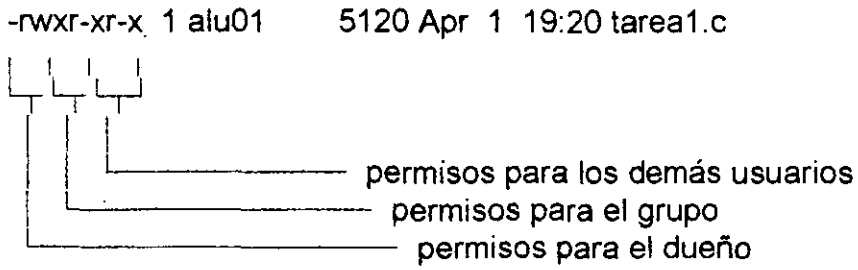
Con la opción **r** (recursivo) se borran todos los subdirectorios y archivos en varios niveles que pueda tener el directorio bin, si para estos se tiene permiso de escritura.

5. 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
```

La cadena rwxr-xr-x representa los permisos asociados con el archivo. Los tres primeros caracteres indican los permisos para el dueño del archivo, los tres siguientes indican los permisos para los usuarios que forman parte del grupo del dueño y los últimos tres indican permisos para todos los demás usuarios:



Los tres caracteres de cada grupo de permisos indican permisos de lectura, escritura y ejecución. El permiso de lectura significa que el sujeto (dueño, usuario de grupo o cualquier usuario) puede leer el archivo y se especifica con **r**. El permiso de escritura indica que el archivo se puede modificar y se señala con **w**. El permiso de ejecución significa que el archivo puede ser ejecutado, es decir su contenido es interpretado como una serie de instrucciones, este se indica con **x**. 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. En el siguiente ejemplo:

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

al archivo tarea1.c se le asignan los permisos para el dueño como rwx, al grupo se le añade el permiso de escritura y a los demás usuarios se les cancela el permiso de ejecución.

La forma indicada anteriormente para establecer permisos es un poco complicada. Otra forma más sencilla es ver los permisos como una secuencia de 9 bits asociados con cada permiso. Un bit encendido indica que el permiso está establecido; cuando el bit está

apagado no existe permiso. Cada uno de los grupos de tres bits se pueden representar con un número decimal no mayor a 7. Con esta notación, 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.

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

CURSOS INSTITUCIONALES

" U N I X "

DIRECCION GENERAL DE CARRETERAS FEDERALES S.C.T.

12, 19, 26 de noviembre y 3 -10 de diciembre de 1994

C A P I T U L O 3

REDIRECCIONAMIENTO, FILTROS E INTERCONEXION DE COMANDOS

JESSICA BRISEÑO

Palacio de Minería

1994

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. En este capítulo se discutirán los filtros más utilizados, así como también la forma de utilizarlos en interconexiones.

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); sin embargo, tanto la entrada de datos como la salida se pueden cambiar, de manera que los comandos puedan tomar datos de entrada de un archivo y su salida se pueda escribir a un archivo o sea tomada como entrada a otro comando.

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

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

El comando ls -la, normalmente va ha desplegar un listado completo de todos los archivos que se encuentran en el directorio de trabajo; cuando se utiliza el operador de redireccionamiento, la salida que produce el comando se va ha almacenar en el archivo denominado listado. En la pantalla de la terminal no aparecerá nada, ya que el flujo de datos de salida se mando hacia el archivo:

```
% 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
```

El archivo será creado en caso de que no exista; si ya existe, su contenido será reemplazado. De esta forma, 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.

De esta forma, el comando:

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

crea el archivo tareas en caso de que no exista, de lo contrario se va ha anexar el contenido de los archivos tarea1, tarea2 y tarea3 al final del contenido del archivo tareas.

El **redireccionamiento de entrada** se da cuando se sustituye la entrada estándar por un archivo. El redireccionamiento de entrada utiliza el operador <. Por ejemplo, para redireccionar la entrada del comando cat:

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

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

#include <stdio.h>

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

El comando anterior toma su entrada del archivo tareas y despliega en la salida estándar su contenido. Supongase que se tiene un programa llamado nomina al cual se le

proporcionan como entrada una lista de empleados y genera como salida un reporte en pantalla. Se podrían hacer una serie de manipulaciones para que la lista de empleados se proporcionará por medio de un archivo y el reporte quedará almacenado en otro:

```
% nomina    mpleados > reporte
%
```

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

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. La siguiente línea de comando:

```
% tail tareas
```

desplegará las últimas diez líneas del archivo tareas. El comando tail también permite que se le indique cuantas líneas se desea desplegar. Así, para desplegar las últimas 20 líneas del archivo tareas, se deberá ejecutar el siguiente comando:

```
% tail -20 tareas
```

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

```
% tail +10 tareas
```

desplegará el archivo tareas a partir de la línea 10.

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
```

muestra las 20 primeras líneas del archivo 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. El siguiente ejemplo muestra el funcionamiento de wc:

```
% 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

por ejemplo, el siguiente comando:

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

da como resultado el número de líneas y palabras en el archivo tarea1.c.

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. Por ejemplo, supongase que se desea ordenar el archivo nombres, cuyo contenido se muestra en la figura 3.1, tomando como llave de ordenamiento el primer apellido; para este caso el comando a ejecutar sería:

```
% sort +1 nombres
Aaron Arcos Tapia
Norberto Arrieta Marquez
Beatriz Barrera Hernandez
Jessica Briseño Cortes
Antonio Chavez Flores
Gabriela Magallanes Gonzalez
Gerardo Meza Gil
Edwin Navarro Pliego
%
```

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

ARCHIVO: *nombres*

```
Antonio Chavez Flores
Norberto Arrieta Marquez
Aaron Arcos Tapia
Jessica Briseño Cortez
Gabriela Magallanes Gonzalez
Gerardo Meza Gil
Edwin Navarro Pliego
Beatriz Barrera Hernandez
```

ARCHIVO: *frutas*

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

Fig. 3.1. Contenido de los archivos nombres y frutas

Otras opciones de sort se muestran en la tabla 3.1

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.

Tabla 3.1. Opciones de sort.

El siguiente ejemplo:

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

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

```
% 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. Recordando que con la opción l del comando ls el cuarto campo que se despliega indica el tamaño en bytes, se puede, con los siguientes comandos, obtener el resultado deseado:

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

Sin embargo, el archivo creado no tiene ninguna utilidad después de que se ha obtenido el resultado. Si el usuario del sistema realizará algunas otras aplicaciones de este tipo, tendría que estar creando archivos temporales, los cuales resultarían innecesarios. 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:

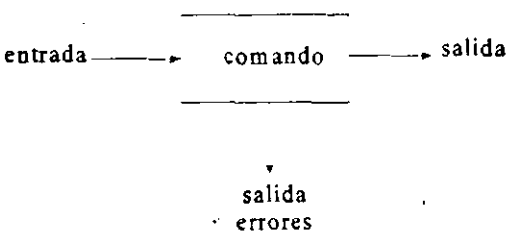
```
% ls -la /bin | sort +3n
```

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. Este tipo de programas deben operar correctamente y generalmente, el esquema para invocarlos es el siguiente:

comando [argumentos] [lista de archivos]

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

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



Cuando 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; esto se muestra en la figura 3.2.

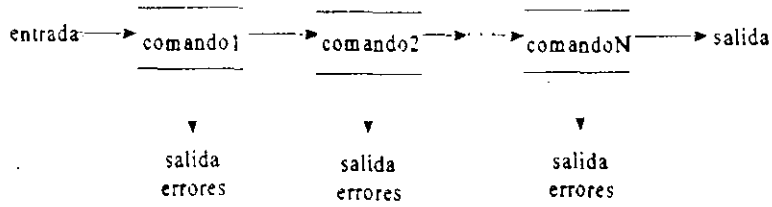


Fig. 3.2. Interconexión de comandos.

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. Por ejemplo, para saber si el usuario con clave curso101 se encuentra en sesión, se podría teclear el siguiente comando:

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

El patrón que se especifica en grep, se conoce como expresión regular. Una expresión regular es un patrón que representa o describe a un conjunto de cadenas. En el ejemplo

anterior, la expresión regular era sencilla; sin embargo, las expresiones regulares se especifican utilizando caracteres que tienen significado especial dentro de la misma expresión regular. A estos caracteres se 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. Por ejemplo, para listar todos los directorios del directorio de trabajo, se utilizaría el siguiente comando:

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

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

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

Se ha mencionado que el carácter **.** representa a cualquier carácter y que para representar a uno en especial basta con representarlo tal como es en la expresión. Si 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.

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
Beatriz Barrera Hernandez
Jessica Briseño Cortes
Antonio Chavez Flores
%
```

El comando grep acepta las opciones que se muestran en la tabla 3.2.

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.

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.

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

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--)'
```

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

La tabla 3.3 resume los metacarateres para grep y egrep.

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

Tabla 3.3. Metacaracteres para grep y egrep.

LABORATORIO

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

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

3. Obtenga los primeros 10 registros de passwd.

4. ¿Cuantas claves de usuarios estan registradas?

5. Ordene el archivo de family por el campo de apellido en forma descendente.

6. Obtenga la lista de usuarios agrupados por su número de grupo.

7. Ordene el archivo de price.fruit por precio.

8. Con grep obtenga todos los usuarios del sistema que no tienen passwdord.

9. Obtenga todos los usuarios con clave de curso.

10. Del archivo de price.fruit obtenga los vegetales con precio entre 1.00 y 150.

11. Imprima los vegatables cuyo nombre tiene seis letras exactamente.

12. Con una interconexión de comandos, obtenga el número de directorios de /usr/users.

Página intencionalmente blanca.

Capítulo 4

Edición de archivos

En el capítulo se indica como crear archivos a través de redireccionamiento de salida; sin embargo este procedimiento, aunque bueno en un principio, tiene muchas limitaciones. Por una parte, sólo se puede modificar la línea que se esta editando y una vez creado el archivo este no se puede modificar. Un modo más cómodo de crear archivos y de manipular el contenido de ellos es utilizar un editor de texto.

La mayoría de los sistemas UNIX poseen varios editores de texto con diferentes características; dos de los más populares son **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 figura 4.1. La invocación para el editor vi es la siguiente:

% vi archivo

si se omite el nombre de archivo se entra a editar un archivo nuevo.

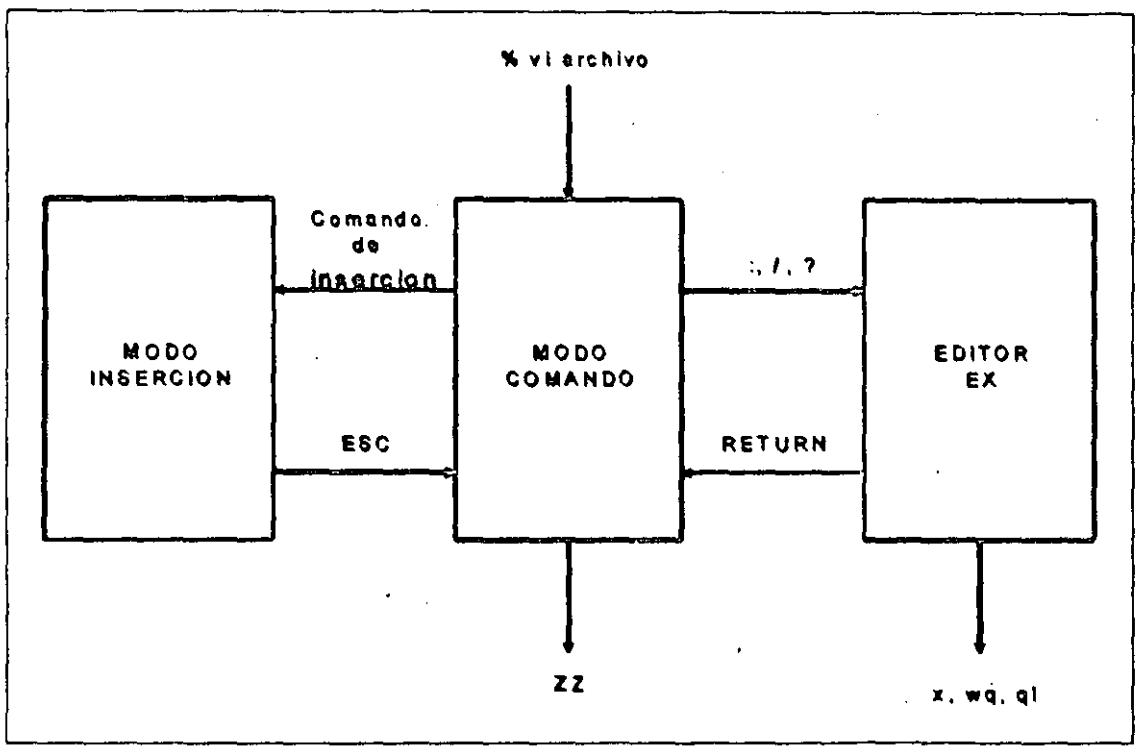


Fig. 4.1. Esquema del editor vi.

Cuando se comienza a editar con vi se entra en el modo comando. Este modo permite el movimiento del cursor a través del archivo que se va desplazando por la pantalla. La tabla 4.1 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.

Tabla 4.1. Comandos del editor vi para movimiento del cursor.

Para comenzar a editar el archivo se debe pasar a modo de inserción a través de un comando, una vez que se ejecuta este comando, todo el texto que se teclea se introduce en el texto. La mayoría de los sistemas UNIX no permiten moverse a través de la pantalla una vez que se entra en este modo. Para volver a modo comando se

debe utilizar la tecla <ESC>. La tabla 4.2 muestra los comando más comunes de inserción y reemplazo.

COMANDO	DESCRIPCION
a	Comienza la inserción de texto a la derecha de donde se encuentra el cursor.
i	Comienza la inserción en la posición del cursor.
I	Inserción al comienzo de la línea.
A	Inserción al final de la línea.
O	Abre una línea antes de aquella en la que se encuentra el cursor y activa modo de inserción.
o	Abre una línea después de aquella en la que se encuentra el cursor y activa el modo de inserción.
R	Activa el modo de inserción a partir de la posición del cursor sobrescribiendo el texto actual.
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.

Tabla 4.2. Comandos de inserción y reemplazo.

En modo comando se pueden utilizar, también comandos de borrado, algunos de ellos se muestran en la tabla 4.3.

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.

Tabla 4.3. 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 en la tabla 4.4.

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.

Tabla 4.4. Comandos de manejo de bloques.

El editor incluye un conjunto de comandos para manipulación de archivos y búsquedas, los cuales son mostrados en las tablas 4.5 y 4.6 respectivamente. Cuando se ejecutan estos comandos la posición del cursor se cambia a la última línea..

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.

COMANDO	DESCRIPCION
: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.

Tabla 4.5 Comandos de manipulación de archivos.

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.

Tabla 4.6. 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

La tabla 4.7 muestra 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.

Tabla 4.7. 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

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

Página intencionalmente blanca.

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

CURSOS INSTITUCIONALES

" U N I X "

12, 19, 26 de Noviembre, 3 y 10 de Diciembre de 1994

SECRETARIA DE COMUNICACIONES Y TRANSPORTES

PROGRAMACION CON AWK

Ing. Jessica Briseño Cortés
Palacio de Minería
1994

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. **A**ho, Brian **W**. Kernighan y Peter J. **W**einberger), 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.

Los patrones pueden ser expresiones regulares como las utilizadas en grep y egrep. El programa consiste de una serie de instrucciones cuya sintaxis es muy parecida a la del lenguaje C. El programa puede ser especificado en la línea de comandos, o bien por medio de un archivo indicandolo con la opción **f** en la línea de comandos:

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 }
...

El patrón selecciona los registros o líneas para los cuales se van a ejecutar las acciones, si dicho patrón no se especifica, se seleccionan todos los registros del flujo de datos de entrada; por otra parte, si no se indica acción alguna, se imprimen todos los caracteres de las lineas seleccionadas en el patrón. El programa de awk es procesado para cada una de las líneas de entrada.

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 }'
```

La acción indicada, es la de imprimir el campo ocho (en este caso el nombre del directorio) de cada una de las líneas seleccionadas con la expresión regular.

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
Arcos Tapia Aaron
Briseño Cortez Jessica
Magallanes Gonzalez Gabriela
Meza Gil Gerardo
Navarro Pliego Edwin
```

Barrera Hernandez Beatriz
%

La instrucción print causa que se haga una copia a la salida de los parámetros listados, las comas en print son reemplazadas a la salida por el separador de campos de salida. Cada vez que se manda una instrucción print, se manda a la salida el carácter de fin de línea. La salida de print puede ser dirigida a múltiples archivos; por ejemplo, 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 separar instrucciones en una misma línea se utiliza ;. En un programa que este almacenado en un archivo cada instrucción se puede colocar en una línea, sin necesidad de separarlas con ;. En el ejemplo anterior se puede utilizar **>>** en lugar de **>** para producir el mismo resultado que cuando se hace redireccionamiento de salida.

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, arg_1, arg_2, ..., arg_n

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 tabla 5.1.

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.

Tabla 5.1. 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.

Por ejemplo, el siguiente programa:

```
% awk '{printf"%3d %-10s %-10s %-10s\n",NR,$1,$2,$3}' nombres
1 Antonio    Chavez     Flores
2 Norberto   Arrieta    Marquez
3 Aaron      Arcos      Tapia
4 Jessica    Briseño    Cortez
5 Gabriela   Magallanes Gonzalez
6 Gerardo    Meza       Gil
7 Edwin      Navarro    Pliego
%
```

da formato a cada uno de los campos de salida. El printf no manda al flujo de salida el caracter de fin de línea, este se debe especificar en la cadena de control y se representa como **\n**. En el ejemplo anterior, se manda a la salida de datos 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 tabla 5.2.

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.

Tabla 5.2. 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.

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

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

el resultado anterior se pudo haber obtenido también con el comando wc con la opción l, tomando como entrada un sólo archivo. El segundo ejemplo consiste en obtener las

claves válidas en el sistema, para ello se deberá tener el siguiente antecedente: existe un archivo llamado /etc/passwd en donde se registran las claves del sistema, dicho archivo contiene un registro con información para cada una de las claves, el registro consiste de varios campos separados por el caracter :, el significado de los campos se muestra en el siguiente registro de ejemplo:

```
curso101:l5eT9M0Ggrxfs:310:15: clave para cursos:/usr/users/curso101:/bin/csh
    1          2       3  4        5                  6           7
```

1. Clave
2. Contraseña encriptada
3. Identificador de usuario
4. Identificador de grupo
5. Comentario
6. Directorio de HOME
7. Shell

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

```
% 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. Tampoco es necesario inicializarlas, ya que por omisión, las variables numéricas se inicializan en cero y las tipo cadena lo hacen con la cadena vacía. Dependerá del contexto tratar a una variable como un número o como una cadena, en casos ambiguos, el valor de cadena se utiliza a menos que los operandos sean numéricos. Existen una serie de funciones para manipulación de variables, constantes numéricas o cadenas, estas se muestran en la tabla 5.3.

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, ...

Tabla 5.3. Funciones predefinidas en awk

Consideremos el ejemplo de un programa cuyo funcionamiento es igual al de wc cuando se le proporciona un flujo de datos de entrada proveniente de la entrada estándar o de un sólo archivo (más adelante generalizaremos el ejemplo para cuando se toman varios archivos de entrada). El programa se muestra en la figura 5.1.

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

Fig. 3.1. Programa que cuenta líneas, palabras y 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
```

En el ejemplo anterior, podría surgir la pregunta ¿que operadores se evalúan primero?, para contestarla hay que saber que todos los operadores tienen cierta precedencia que indica el orden de evaluación. Si se quiere romper dicha precedencia, se pueden utilizar paréntesis. La tabla 5.4 muestra los operadores válidos en awk en orden creciente de precedencia.

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

Tabla 5.4. 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
```

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

El programa prog1.awk mostrado en la figura 5.2 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
}
```

Fig. 5.2. Listado del programa prog1.awk

El comando ls genera, como ya se habia visto, un listado con información para los archivos; sin embargo, la primera línea de salida es un encabezado que indica el total de archivos listados, por lo cual esta primera línea no debe procesarse. La instrucción **next** provoca que se lea el siguiente registro en el flujo de datos de entrada.

Otro programa ejemplo se muestra en la figura 5.3 y es una implementación con awk del comando wc.

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
```

```
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

}
```

Fig. 5.3. Implementación de wc con awk.

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

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

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

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.
La figura 5.4 muestra el programa sort.awk que ordena las líneas de entrada por el método conocido como "la burbuja"; en dicho ejemplo se almacenan las líneas de entrada en un arreglo de cadenas de caracteres.

```
# 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]
}
```

Fig. 5.4. Programa sort.awk

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

for(índice in arreglo)
proposición

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

Un ejemplo que podría aclarar el uso de los arreglos asociativos, lo constituye el programa que obtiene la frecuencia que tiene cada una de las palabras de un texto. En principio no se sabe cuantas palabras contiene un texto cualquiera, si se contara únicamente con arreglos con índices enteros se tendría que tener un mapeo de un índice a una palabra; sin embargo, con los arreglos asociativos esto es innecesario ya que las mismas palabras son los índices y los elementos de los arreglos contienen la frecuencia. De esta forma el programa que realiza tal función se muestra en la figura 5.5.

```
# Programa que obtiene la frecuencia de las palabras
# de una entrada de datos

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

Fig. 5.5. Programa que obtiene la frecuencia de las palabras del flujo de datos de entrada.

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.

Página intencionalmente blanca.

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

CURSOS INSTITUCIONALES

" U N I X "

SECRETARIA DE COMUNICACIONES Y TRANSPORTES

12, 19 y 26 de noviembre, 3 y 10 de diciembre de 1994

EL USO DEL SHELL

ING. JESSICA BRISEÑO
Palacio de Minería
1994

El uso del shell

Los componentes o la arquitectura de alto nivel de UNIX se pueden representar por el diagrama de la figura 2. Si vemos el sistema como un conjunto de capas, el sistema operativo propiamente dicho conforma la capa más interna, esta parte del sistema operativo es la que se comunica directamente con el hardware y se le conoce comúnmente como **kernel** o **núcleo**. Entre las funciones del núcleo se pueden mencionar las siguientes: planificación de tareas, administración de recursos del sistema, administración de procesos y manejo de memoria.

En la siguiente capa se encuentran las utilerías y programas de aplicación vinculados con el sistema que se encargan de ejecutar una variedad de rutinas y funciones especiales de mantenimiento del sistema. Estas utilerías se comunican con el kernel por medio de 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.

Uno de los programas de aplicación más importantes es el intérprete de comandos o **shell**. Este programa se ejecuta inmediatamente después de que se abre una sesión de UNIX, es 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.

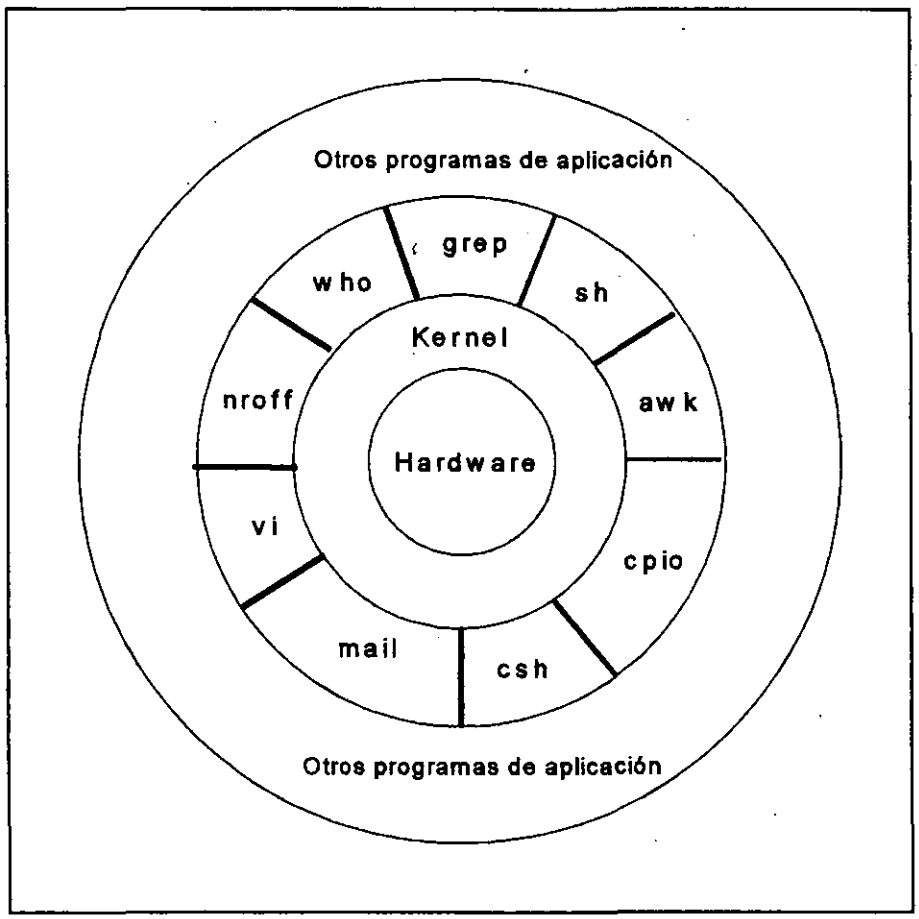


Fig. 2. Arquitectura de UNIX

Cuando se inicia una sesión Unix, el sistema operativo proporciona un shell por default, este shell se denomina login shell. El login shell se indica en el archivo /etc/passwd y no necesariamente debe de ser un shell, podría ser cualquier programa ejecutable.

Mientras uno se encuentra en sesión se puede mandar a ejecutar algún otro shell para realizar algunos trabajos, pero estos procesos serán procesos '*hijos*' de nuestro login shell.

Si se desea podemos cambiar de login shell, ésto se hace mediante el comando *chsh*, al cual se le proporciona la cuenta a la que se le desea cambiar de login shell y la ruta completa de donde se encuentra localizado el intérprete de comandos (shell). Por ejemplo, el usuario raq desea cambiar su login shell a C shell (csh), entonces debe teclear lo siguiente:

```
$ chsh acf /bin/csh
```

Esto funciona para las versiones de Unix system V. En Ultrix basta con dar el comando chsh y presionar return. La computadora nos informará cual es nuestro login shell actual y nos preguntará cual deseamos que sea nuestro nuevo login shell, donde basta con teclear la abreviación del shell que deseamos. Por ejemplo:

```
$ chsh
Changing login shell for acf
shell [/usr/bin/ksh]: csh
```

Una vez echo ésto, la próxima vez que entremos a sesión, nuestro login shell será C shell (csh).

Cada vez que un usuario entra a sesión, la computadora ejecuta algunos cuantos archivos de comandos para definir ciertas características del medio ambiente donde trabajará el usuario, es decir del shell..

Los archivos de configuración dependen del login shell, pero pueden ser utilizados para establecer el valor de variables de ambiente como podría ser el path o algunos alias o variables que se quisieran definir de manera global. Cuando se creen o modifiquen los archivos de configuración, debemos cerciorarnos de que tengan permiso de ejecución.

Agrupando comandos del shell

Hasta ahora, hemos discutido la forma de mandar a ejecutar varios comandos y/o archivos de comandos, pero de momento solo se ha mandado a ejecutar un comando a la vez. Uno puede obtener más ventajas sobre el shell desarrollando la habilidad de ejecutar varios comandos a la vez.

Siempre que usamos el shell con la entrada estándar, el teclado, y con la salida estándar, el monitor, estamos usando el shell en forma interactiva. Una de las ventajas de usar el shell en forma interactiva, es la de poder observar el resultado de los comandos en cuanto nuestra terminal los recibe. Una de las desventajas de usar el shell en forma interactiva (hasta ahora) es la de poder mandar ejecutar un sólo comando a la vez.

Si mandamos ejecutar un segundo comando antes de que el primero termine de ejecutarse, la salida del primero se mezclará muy probablemente en la terminal con nuestro segundo comando mientras lo vamos escribiendo. En el siguiente ejemplo, al escribir el comando **date** se mezcla la salida del comando **who**:

```
% who
sjg000          tty00   Apr 26 15:03
pnh000          tty02   Apr 26 14:55
kjd000          tty04   Apr 26 12:44
%
```

Muchas veces necesitamos observar el resultado de un comando para decidir cuál será el siguiente comando a ejecutar. Sin embargo, hay ocasiones en que la salida del comando anterior no tiene efecto sobre el siguiente comando a teclear. Habrá ocasiones en las que deseemos ejecutar varios comandos a la vez ya que no importa el resultado de éstos para que los demás se ejecuten. Esto es, es posible mandar a ejecutar varios comandos a la vez.

Hasta ahora el 'separador' de comandos ha sido la tecla RETURN, o ENTER. Debido a que RETURN es la tecla que le indica al shell empezar la ejecución del comando escrito en la línea de comandos, sólo se puede ejecutar un comando a la vez.

Es posible mandar a ejecutar varios comandos a la vez separando los comandos por puntos y comas (;) como se muestra más adelante. Cuando se le pasan al shell varios comandos para su ejecución separados por puntos y comas en la misma línea de comandos, ésto se conoce como un 'job'. Más adelante en este capítulo hablaremos sobre el manejo de jobs.

```
% who;date;echo "hola"
sjg000          tty00   Apr 26 15:03
pnh000          tty02   Apr 26 14:55
kjd000          tty04   Apr 26 12:44
Mon Apr 26 15:15:32 GMT 1993
hola
```

Además, podemos continuar escribiendo un comando en la siguiente línea, utilizando diagonales invertidas (\) al final de cada línea seguida de la tecla RETURN. Con ésto se consigue 'escapar' el significado de la tecla RETURN ante el shell para que éste no mande a ejecutar los comandos escritos hasta el momento, y poder seguir escribiendo en la siguiente línea. Esto es especialmente útil cuando se trata de comandos largos como por ejemplo:

```
% grep "estados" capítulo1.txt capítulo2.txt \<RETURN>
capítulo3.txt ../pubicaciones.old contenido.list
```

Combinaciones de puntos y comas y antidiagonales pueden ser usadas para escribir comandos que no estén limitados por la longitud de la línea de comandos como se muestra a continuación:

```
% echo "La fecha y hora actual es: "; date; \<RETURN>
echo "Lista de los usuarios en sesion:";\<RETURN>
who; echo "Unix posee varias caracteristicas interesantes."
La fecha y hora actual es:
Mon Apr 26 17:11:02 GMT 193
Lista de los usuarios en sesion:
sjg000          tty00  Apr 26 15:03
pnh000          tty02  Apr 26 14:55
kjd000          tty04  Apr 26 12:44
Unix posee varias caracteristicas interesantes.
%
```

Es importante notar las diferencias entre usar punto y coma (;) para separar comandos y las interconexiones (|) o pipes. Las interconexiones le dicen al shell que la salida del primer comando la use como la entrada del segundo comando. Un punto y coma entre dos comandos le dice al shell que ejecute el primer comando y después de éste ejecute al segundo comando. Esto es, mientras que con interconexiones los procesos se realizan en forma concurrente, con puntos y comas se ejecutan de manera secuencial.

Cuando se usan interconexiones (|) entre comandos en lugar de puntos y comas, solo es desplegada la salida del último comando. Esto es debido a que la interconexión toma la salida del comando que se encuentra antes de la interconexión y esta se convierte en la entrada del comando que se encuentra después de la interconexión. Si el comando que se encuentra después de la interconexión generalmente no toma su entrada del teclado, la interconexión no tiene sentido. Por ejemplo:

```
% date | who | pwd | ls
ascii   calendar        mbox        plum        settings
bin         misc            practice    shell
% date ; who ; pwd; ls
Mon Apr 26 17:15:00 GMT 1993
sjg000          tty00  Apr 26 15:03
kjd000          tty04  Apr 26 12:44
/users/jess
ascii   calendar        mbox        plum        settings
bin         misc            practice    shell
%
```

Si por el contrario, usamos puntos y comas en lugar de utilizar interconexiones, los resultados pueden ser confusos. Por ejemplo, los siguientes comandos **date** y **who** son ejecutados, pero el resultado del comando who no es ordenado como se esperaba. En lugar de ésto, da la impresión de que la terminal se 'bloqueo' (queda esperando indefinidamente), lo cual se debe a que el comando sort está esperando datos de la entrada estándar (el teclado). Debemos, entonces, presionar la tecla de interrupción o la de fin de archivo para recibir nuevamente el prompt. Si se usan interconexiones, la salida del comando who entonces es ordenada como se esperaba y el prompt es regresado inmediatamente después.

```
% date ; who ; sort
Mon Apr 26 17:15:00 GMT 1993
sjg000              tty00   Apr 26 15:03
pnh000              tty02   Apr 26 14:55
kjd000              tty04   Apr 26 12:44
^D
%
% date ; who | sort
Mon Apr 26 17:15:43 GMT 1993
kjd000              tty04   Apr 26 12:44
pnh000              tty02   Apr 26 14:55
sjg000              tty00   Apr 26 15:03
%
```

También podemos agrupar comandos entre paréntesis. El efecto que esto tiene es el de generar un nuevo shell ('hijo' o sub-shell) a nuestro shell actual, y mandar a ejecutar los comandos tecleados en este nuevo sub-shell. Por ejemplo, supongamos que nos encontramos trabajando en el subdirectorio /users/alumnos/vlu000 viendo algunos programas fuente, y deseamos examinar el contenido de algunos archivos en el subdirectorio /users/pub. Entonces podríamos ejecutar el comando siguiente:

```
% (cd /users/pub; grep 'printf' snmpd.c ifconfig.c)
```

Cuando el prompt aparezca de nuevo, el directorio de trabajo permanecerá igual que antes de haberse ejecutado el comando anterior. Esto es debido a que el comando cd y el comando grep fueron ejecutados en el nuevo sub-shell. En cuanto éstos terminaron, el nuevo sub-shell también terminó y el control fue regresado al shell que mandó a ejecutar el comando. En ese shell (el actual) no se ejecutó nunca ninguna orden de cambiar de

directorio, es por eso que al regresar el control, se sigue trabajando en el mismo directorio.

También podemos utilizar el agrupamiento entre paréntesis para realizar direccionamientos de salida estándar y de error a la vez. Cosa que no es posible realizar utilizando en forma independiente los símbolos de redireccionamiento. Por ejemplo, para redireccionar la salida estándar de un comando grep al archivo *salida* y la salida de errores al archivo *diagfile* se puede utilizar el comando siguiente:

% (grep 'mount' /etc/* > salida) >& diagfile

Cabe notar lo siguiente:

1.- La salida estándar del comando grep se mando al archivo salida:

2.- La salida de errores (**>&**) del comando encerrado entre paréntesis se mando al archivo diagfile:

Shell scripts

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

Los comandos que se escriben dentro de un shell script son los mismos que se usan desde la línea de comandos, sin embargo, debido a que existen varios intérpretes de comandos como son el C-shell (csh), el Bourne-shell (sh), el Korn-shell (ksh) y otros más, ciertas instrucciones, relacionadas principalmente con el control de flujo, deberán de ser escritas con la sintaxis propia del intérprete de comandos encargado de ejecutar nuestro archivo de comandos.

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

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

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

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

Algunos puntos a destacar sobre este archivo son:

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

- El símbolo de número (#) es usado para introducir comentarios. Los comentarios no son interpretados por el shell como comandos. Son ignorados.

- Cada línea de comentario es encerrada entre apóstrofos. Así se asegura que el comentario sea efectivamente tratado como comentario.

- Para que el shell script sea ejecutado por el intérprete de comandos C-shell, es muy importante que el primer carácter del primer renglón contenga el símbolo de número (#). Si no es así el intérprete que ejecutaría nuestro archivo de comandos sería el Bourne-shell.

- Un comentario que está en la misma línea de un comando es llamado un comentario en línea (in-line comment). Hay que asegurarse de separar el comando del comentario con un separador de comandos punto y coma (;). Si no se usa, el comentario es interpretado como un argumento del comando.

- La opción -n del comando echo le dice al shell no imprimir un retorno de carro al final de la línea impresa.

Ejecutando Shell Scripts.

Hay tres formas de ejecutar un shell script:

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

Invocando un subshell.

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

```
% csh nombre_archivo
```

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

```
% csh cld
```

Corriendo el script en el shell Actual.

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

```
% source nombre_archivo
```

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

```
% source cld
```

Haciendo el Script Ejecutable.

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

```
% chmod ugo+x cld
% cld
...
%
```

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

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

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