Sistema Operativo Unix

Nivel Básico

Contenido

1.-INTRODUCCION AL UNIX **3**
1.1.- Los Sistemas Operativos en la Historia 4
1.2.- Ejemplo de versiones de UNIX 6

2.- CONCEPTOS BASICOS **7**
2.1.- CUENTAS *7*
2.11.-Usuario *7*
2.1.2.-Administradores *8*
2.1.3.-Sesiones de ejemplo *8*

2.2.-ARCHIVOS Y DIRECTORIOS 11
2.2.1.-Archivos especiales *12*
2.2.2.-Nombres de archivos *12*
2.2.3.-Directorios *12*
2.2.4.-Rutas *13*
2.2.5.-Ejemplos de nombres de archivo *14*
2.2.6.-Manejo de Archivos *14*

2.3.- METACARACTERES 20

3.- PERMISOS **22**

4.- PROCESOS **24**

5.- C-Shell (csh) **27**
5.1.- History *27*
5.2.- Alias *28*
5.3.- Jobs *28*

6.- MANUALES EN LINEA **30**

7.- INDICE DE COMANDOS MAS UTILIZADOS EN UNIX **31**

8.-ANOTACIONES **33**

1.- Introducción al UNIX

El UNIX es un Sistema Operativo que se aventaja con cualquier otro Sistema Operativo en los siguientes factores:

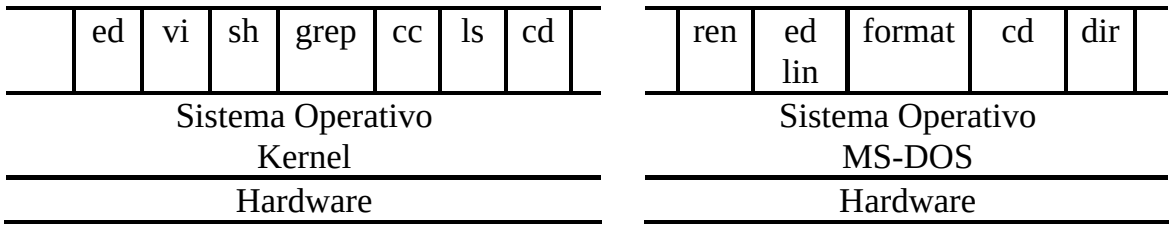
⇒ **Multiusuario :** Múltiples usuarios pueden trabajar a la vez.
⇒ **Multiprogramación :** Cada usuario puede correr más de un programa a la vez; además de los procesos del sistema.
⇒ **Multisesión**: Un usuario puede entrar varias veces a la máquina.

Nota:

Multiprogramación	**Multiproceso**
→ Varios procesos corriendo	→ Varios procesos corriendo
→ Sólo un proceso se ejecuta a la vez	→ Sólo un proceso se ejecuta a la vez
→ Varios procesos en memoria principal	→ Sólo un proceso en menú principal (se hace swap)

La idea de hacer todo "multi" es la de tener factor de utilización elevado de todos los recursos de la máquina (CPU, Impresoras, Disco, etc).

Diagrama del Sistema Operativo Unix (analogía con MS-DOS).



Como Unix está escrito en un lenguaje de alto nivel (Lenguaje C), es relativamente fácil portar el sistema operativo a distintos tipos de máquinas. Muchos fabricantes de computadores reconocieron que era más fácil portar Unix a sus máquinas que desarrollar un sistema operativo completo, con todos sus utilitarios asociados.Con esto hay implementaciones de Unix para todo el rango de máquinas, desde computadores personales hasta supercomputadores. Tanto es así que los actuales esfuerzos internacionales de estandarización en el área de sistemas operativos se basan en Unix.

1.1.- Los Sistemas Operativos en la Historia

Existe una relación estrecha entre los SO. y la plataforma de hardware, a lo largo de las diferentes generaciones de computadores.

La primera generación de computadores (1945 - 1955)

- Caracterizada por los grandes tamaños : construcciones en base a tubos de vacío y tableros para conexiones.
- Capacidad muy limitada y lentas.
- El mismo grupo de personas, diseñada, construía, programaba, operaba y mantenía las máquinas.
- No existía SO. los programas se introducían bit a bit. Luego se automatiza la función cuando en 1950, se introduce la lectora de tarjetas perforadas.
- Nombres destacados : Howard H. Aiken (primer computador electrónico MARK-1, Universidad de Harvard), John von Neumann, William Mauchly y J. Prespert Echert (primer computador electrónico ENIAC, Universidad de Pensilvania).

La segunda generación de computadores (1955 - 1965)

- Caracterizada por la introducción de los transistores que hizo a los computadores más confiables; se pudieron comercializar.
- Se separaron las funciones de diseño, construcción, programación, operación y mantenimiento. El programador deja de tener contacto con la máquina.
- El procedimiento a seguir era : diseñar el programa, perforar tarjetas, cargar el programa, esperar la salida, impresión de resultados (en línea sistema on-line).
- Para automatizar la tarea del operador, nace el primer SO, el “Monitor Residente”.
- Luego surge el sistema de procesamiento por lotes. Sólo se cargaban programas escritos en Fortran o Cobol para luego procesarlos en serie, mediante un lenguaje de comandos JCL (Job Control Lenguaje).
- Con la introducción de la unidad de cinta, surge el sistema fuera de línea (off-line) $\Rightarrow$ mayor utilización del procesador.

La tercera generación de computadores (1965 - 1980)

- Caracterizada por los circuitos integrados (computadores de tamaño más pequeño que en la primera generación) y la multiprogramación.
- Las técnicas de buffering, un antecesor del DMA (Direct Memory Access).
- Aparecen los discos magnéticos que permiten una lectura y escritura en forma “simultánea”, una ventaja frente a las unidades de cinta.
- Las técnicas de SPOOL (Simultaneous Peripheral Operation On-Line) permiten que la salida de un programa se escriba a disco para luego ser impresa en línea.
- La necesidad del mercado por ciclos de procesamiento fue aumentado, lo que derivó en la necesidad de máquinas más grandes y compatibles entre sí.

- IBM introduce en el mercado su sistema 360, una serie de máquinas con software compatible que van desde una 1401 hasta que aquellas que eran más poderosas que la 7094).
- La idea de la “familia 360”, resultó en un SO. muy grande y complejo, lleno de errores y difícil de depurar.
- La multiprogramación fue uno de los aportes más significativos de esta generación de computadores. Soporte de hardware para la protección de diferentes programas residentes en memoria simultáneamente.
- La necesidad de reducir los tiempos de respuesta entre la entrada de datos y los resultados, fue uno de los impulsores de los sistemas de tiempo compartido.
- CTSS desarrollado por MIT, fue uno de los primeros intentos de SO. de tiempo compartido, sus sucesores son, TSS de IBM, Multics que se caracterizó por ser el primer SO. escrito en un lenguaje de alto nivel (MIT, Bell Labs, General Electric), CP/CMS que luego derivó en el SO. VM de IBM, y Unix (Bell Labs).
- Aparece el concepto de memoria virtual (CTSS, Multics y CP/CMS); se podían ejecutar programas mayores que la memoria real disponible.
- Surge la ingeniería de software; la comprensión de que el software debía ser diseñado de modo que fuera confiable, comprensible y fácil de mantener, mediante el uso de métodos disciplinados y estructurados en el construcción de programas.
- Los SO. de esta época se caracterizaron por tener múltiples modos de operación que comprendían el procesamiento por lotes, de tiempo compartido y de tiempo real.
- Gran desarrollo de los minicomputadores de la familia DEC PDP, a un costo del orden del 5% del valor de una 7094 de IBM.
- Aparece el uso de los estándares en los protocolos de comunicación como TCP/IP del departamento de Defensa de Estados Unidos y el uso en redes de área local del estándar Ethernet desarrollado en el Centro de Investigación de Palo Alto de Xerox (PARC).

La cuarta generación de computadores (1980 - 1990)

- Caracterizada por la utilización de los circuitos de integración a gran escala LCI (Large Scale Integración), y la aparición de los computadores personales.
- La reducción en el costo de los computadores hacen posible la adquisición masiva de equipos por parte de la universidades e industria.
- Se hacen populares las estaciones de trabajo.
- Surge la industria del software de aplicación; un ambiente amistoso para el uso de sistemas como también para el desarrollo de programas.
- En las computadoras personales que utilizan el procesador Intel 8088, y sus sucesores,80286, 80386 y 80486, domina el SO. MS-DOS de la Microsoft. En las estaciones de trabajo el SO. predominante es UNIX.
- Se hicieron populares las aplicaciones en redes de computadores como el correo electrónico, la transferencia de archivos y el acceso a bases de datos remotas.
- Proliferaron las aplicaciones tipo cliente/servidor.

Las tendencias en la década del 90.

- Una característica fundamental será la computación distribuida, mediante el uso de plataformas multiprocesadoras y procesadores conectados en red.
- Desarrollo de una red de servicios integrados ISDN (Integrated Services Digital Network) para la transferencia de datos con gran cantidad.
- Dispositivos en entrada más rápidos y eficientes como los sistemas de reconocimiento automático de voz.
- Calidad fotográfica en el tratamiento de imágenes.
- Gran desarrollo en aplicaciones en Multimedia (Datos + Sonido + Imágenes), como por ejemplo el vídeo tridimensional con sonido para la administración de un aeropuerto.
- Los sistemas abiertos. Sistemas de computación y/o de comunicación, cuyas especificaciones están ampliamente disponibles, aceptadas y estandarizadas. Los elementos principales son :

1. Normas de comunicación abierta, como el modelo de referencia OSI desarrollado por la Organización Internacional de Normas.
2. Normas de SO. abiertos como UNIX.
3. Normas de interfaces de usuario abiertas, como el sistema de ventanas X, desarrollado por MIT.
4. Normas de aplicaciones de usuario abiertas, adoptadas por varias corporaciones como X/Open y la OSF.

1.2. - Ejemplo de versiones de UNIX

A través de la historia ha sido difícil conciliar a una sóla versión del UNIX. Sin embargo se distinguen dos grandes tendencias: BSD y SYS V.

Las diferencias intervienen desde la forma de ejecutar ciertos comandos hasta la forma en que el Sistema Operativo maneja los procesos.

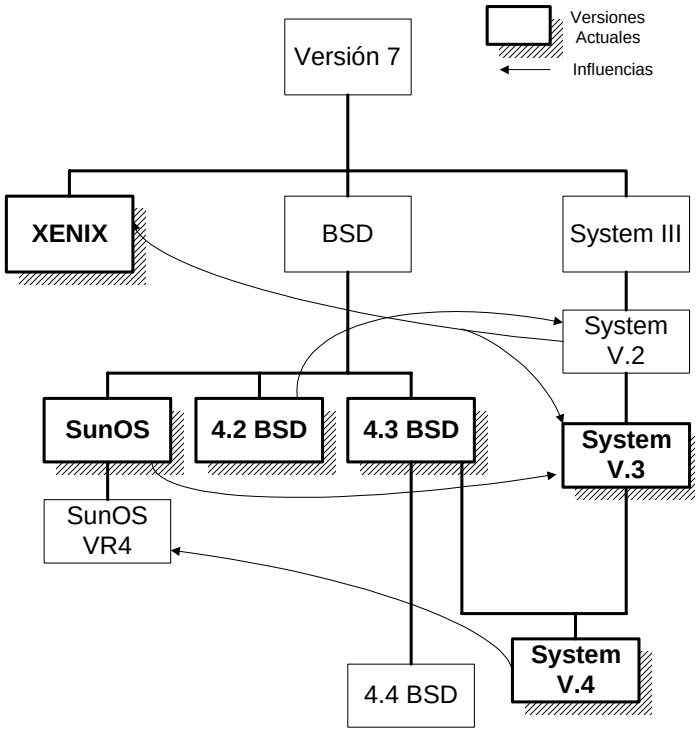
Cabe destacar que las versiones comerciales del UNIX son a veces mezclas de las dos tendencias complicando aún más el aprendizaje.

A continuación se entrega una lista de Sistema Operativo Unix más conocidos :

NOMBRE	**FABRICANTE**	**BSD**	**SYSV.**
Sun OS	SUN Microsystems	✓	
Solaris	SUN Microsystems		✓
HPUX 8 y 9	Hewlett Packard	✓	
HPUX 10			✓
Linux	Investigadores (Gratis)	✓	
OSF/1	Digital (DEC)	✓	

Otros UNIX son :

IRIX (Silicon Graphics), AIX (IBM), Ultrix (VAX).



2.- Conceptos Básicos

Este capítulo explica los conceptos básicos que Ud. necesita saber para trabajar en el ambiente Unix. Después de leer este texto Ud. entenderá los sistemas de archivos y los directorios, su organización y los nombres utilizados. Sabrá como se ingresa un comando y como los resultados de éste pueden ser redireccionados.

Es importante que lea este capítulo antes de continuar con los restantes.

2.1.- Cuentas

El sistema conoce a los usuarios, tanto usuarios comunes como administradores, a través de cuentas, que discutiremos a continuación.

2.1.1.- Usuarios

Una cuenta debe ser creada antes de que Ud. pueda conectarse al sistema Unix.
Estas cuentas poseen la siguiente información.

- **Login.** Este es el nombre por el cual Ud. es conocido en el sistema. Este nombre debe ser ingresado al comenzar cada nueva sesión de trabajo, así el sistema saber quien está trabajando en él.

- **Password** (Clave)**.** Para aumentar el nivel de seguridad en el equipo, cuando Ud. ingresa al sistema éste le pide una palabra clave. Sólo si ella es entregada en forma correcta Ud. puede ingresar al sistema. La password es solicitada inmediatamente después del *login.*

- **Identificación de usuario (UID)** A cada usuario le corresponde un número único que lo identifica dentro del sistema, así se evita que otro usuario tenga derechos sobre los archivos de otro.

- **Identificación del grupo. (GID)** Cada usuario es conocido por el sistema como un ser independiente que pertenece a uno o más grupos. Ser miembro de un grupo es importante por razones de seguridad y administración.

- **Directorio personal** (home directory)**.** Este es el lugar, dentro del sistema de archivos, donde Ud. puede guardar archivos personales. Cuando Ud. ingresa al sistema se ubica en este lugar.

- **Login Shell.** Este es el programa que lee y ejecuta los comandos Unix que Ud. ingrese. En la mayoría de los casos, su *login shell* será uno llamado *Bourne Shell* usa el signo peso ($) para presentarse. Sin embargo, Ud. podría tener configurado el *C-Shell,* en cuyo caso aparecerá en la pantalla el signo de porcentaje (%). Otra posibilidad es que posea el *Visual Shell,* que es una interfaz que funciona presentando menús. En todo caso, independiente de cual sea la presentación (prompt) que tenga una máquina nosotros usaremos la expresión **Unix prompt** para referirnos a cualquier tipo de presentación, sea ésta un signo peso o un signo de porcentaje.

2.1.2.- Administradores

Además de las cuentas normales Unix posee una cuenta de superusuario. El superusuario recibe el nombre de root. Esta cuenta es la encargada de contener al administrador del sistema, para que realice sus tareas, ya que a través de ella se puede leer y editar cualquier archivo del sistema, así como, ejecutar cualquier programa.

2.1.3.- Sesiones de ejemplo

Lo primero que se debe hacer para comenzar a trabajar es conectarse a la cuenta definida para el usuario, por lo general se dispone de una cuenta por persona. Al acercarnos a un terminal debe verse un mensaje del tipo:

Login:

Esto nos indica que el computador está en espera de una conexión, en este momento podemos escribir nuestro **username** (nombre de usuario), por ejemplo:

login: alumnoXX

Luego si pulsamos la tecla **ENTER** nos responde con el mensaje:

password:

Lo que nos está pidiendo es una palabra clave (similar al sistema utilizado por los cajeros automáticos), cada usuario puede definirse su propia password que debe ser conocida sólo por él, así impide que otras personas ocupen su cuenta, lean, modifiquen o borren sus archivos o lean su correo personal. En este momento debemos ingresar nuestra password (la primera vez es asignada por el administrador del computador), la cual no se muestra en la pantalla mientras se escriba, así se evita que otras personas lo puedan leer. En caso que algo falle, el computador no muestra el mensaje:

login incorrect

Y vuelve a preguntar por un username, esto puede significar que se ha cometido un error al digitar el nombre del usuario o la password por lo que volvemos a intentar. Si luego de varios intentos aún no logramos conectarnos es recomendable ir a hablar con el administrador.

Debemos notar que Unix es case sensitive, es decir, considera las letras mayúsculas y minúsculas distintas, por ejemplo, considera las palabras hola, Hola, HOLA y HoLa distintas. Por ello debemos poner especial atención al ingresar nuestra identificación, pues es causa común de errores.

En caso de haber ingresado los datos correctamente, aparecerán en la pantalla algunos mensajes del administrador (si existen) y luego el **prompt** del sistema (mensaje que nos indica que el computador está esperando un comando), algo como:

- maquina@login**$**

La primera forma, en la práctica, ya no es muy utilizada, generalmente uno quiere más información en el **prompt**, como por ejemplo: el computador en que estamos trabajando, la cuenta en que estamos, el directorio que estamos utilizando, etc.

En este momento se está ejecutando en nuestra cuenta el **shell** que es un programa que interpreta los comandos y los ejecuta, él se encarga de la comunicación entre el usuario y el computador, de otra forma, sería casi imposible darle órdenes. Por lo general, se dispone en Unix de varios **Shell**, como por ejemplo, el Bourne Shell (sh) que es el más básico y proporciona menos facilidades para el usuario, y el **C-Shell** (csh).

La forma de indicarle al computador lo que queremos hacer es por medio de órdenes o comandos que se escriben a continuación del **prompt**, las que se indican de la forma:

$ Comando opciones parámetros

Las opciones son modificadores para los comandos, que no siempre es necesario darlas, cada opción se representa, en la mayoría de los casos, con una sola letra precedida de un signo -. Los parámetros son información que el comando necesita, lo que depende del comando en particular. Por ejemplo:

$ wc -l carta documento ficha salida.dat
$ ls -a -l -F carta documento salida. dat
$ cal 1 1996

En el primer caso, el comando ejecutado es **wc** con la opción **l** y se le dan los parámetros carta, documento, ficha y salida.dat. En el segundo ejemplo, el comando es ls, se dan las opciones **a**, **l**, y **F** y los parámetros carta, documento y salida.dat. En la práctica se prefiere, si es posible, juntar las opciones en un solo -, así, este ejemplo puede ser escrito de forma totalmente equivalente como:

$ ls -alF carta documento salida.dat

Para el tercer ejemplo, el comando es cal, al cual no se le da ninguna opción y se le dan los parámetros 1 y 1996.

Para cambiar la password (palabra secreta) existe el comando passwd, el cual nos pide ingresar la password actual y luego la nueva dos veces, para evitar posibles errores. Este comando no requiere de opciones ni parámetros, se da de la forma:

$ passwd

Luego de esto, para ingresar al sistema, habrá que escribir la nueva password.

Es muy importante que luego de terminar nuestro trabajo nos desconectemos del computador, con ello evitamos además que otra persona pueda ver nuestro trabajo, correo personal, etc., además deja el terminal disponible para que pueda ser usado por otra persona. La desconexión se hace por medio del comando **logout, exit, o ^d**:

$ exit

Luego de esto el terminal debiera mostrar la pantalla de presentación que tenía al momento de comenzar la sesión.

Importante: No debe apagar el terminal antes de dar el comando **exit,** ya que con ello no se realiza la desconexión, por lo tanto, su cuenta sigue activa.

2.2.- Archivos y Directorios

El archivo es la unidad fundamental del sistema de archivos de Unix. Existen tres tipos de archivos: los archivos ordinarios (usualmente llamados archivos), los archivos especiales, y los directorios. Cada uno de estos son descritos en las siguientes secciones.

- **Archivos Ordinarios**. Los archivos ordinarios son usualmente los documentos programados, programas, datos. Los archivos ejecutables, o programas computacionales son considerados también archivos ordinarios.

  Todos los archivos ordinarios poseen los siguientes atributos :

  ⇒ Un nombre de archivo (no necesariamente único)
  ⇒ Un número único dentro del sistema de archivos, llamado número del inodo
  ⇒ Un tamaño medido en bytes (caracteres)
  ⇒ Los instantes de creación del archivo, último acceso (sea para leer o grabar) y del último cambio realizado
  ⇒ Un conjunto de permisos de acceso a él
  ⇒ Un propietario y un grupo al que pertenece

Protección de Archivos. Sobre un sistema multiusuario, a menudo, es necesario proteger ciertos archivos, controlar el acceso a ellos por parte de los usuarios. Los archivos se protegen a través de la asignación de adecuados "permisos de Acceso". Unix provee tres niveles de permisos de acceso:

- **read (r)** Teniendo permiso de lectora (read) sobre un archivo, el usuario puede ver el contenido de él con los comandos cat y more. Sin embargo, si un archivo tiene permiso de solo lectura (read-only) él no puede ser editado.

- **write (w)** Teniendo permiso de escritura (write) el usuario puede editar y borrar el archivo.

- **execute (x)** Si el archivo es un programa y el usuario tiene permiso de ejecución sobre él, entonces el usuario puede correr el programa. Ud. no puede ejecutar un programa sobre el que no posee permiso de ejecución. Si es un directorio, entonces Ud. puede entrar a él.

Los permisos de acceso son asignados por el propietario del archivo (por omisión el propietario es el creador). Cualquier combinación de los tres tipos de acceso está permitido. Por lo tanto, el propietario del archivo determina que otros usuarios pueden leer, escribir y/o ejecutar un archivo. Tenga presente que **el superusuario tiene permiso de lectura, escritura y ejecución sobre todos los archivos en el sistema.**

El mecanismo de seguridad de archivos en Unix es muy flexible. Los permisos de acceso están separados para el **propietario** del archivo, para el **grupo** y para el **resto**. Un caso típico es aquel donde el propietario tiene permiso de lectura y escritura, el grupo solo puede leer el archivo, y el resto no tiene ningún acceso a él.

2.2.1.- Archivos especiales

Cada dispositivo físico en el sistema, tales como disquetteras, impresoras, terminales, etc. reciben el nombre de archivos especiales. Para efectos de este curso solo es necesario saber de su existencia.

2.2.2.- Nombre de archivos

Un nombre de archivo es una secuencia de 1 a 255 caracteres tales como letras, dígitos y algunos caracteres especiales (_). Cada archivo, directorio y dispositivo posee un nombre de archivo. Si bien se pueden usar distintos caracteres para especificar un nombre es recomendable restringirlos al conjunto alfanumérico más el punto (.).

Los nombres de archivo debieran ser descriptores de su contenido. Por ejemplo, un archivo que contenga ordenes de compra podría llamarse ordenes. Dentro de un directorio los nombres son únicos para cada archivo, pero dentro de diferentes directorios pueden existir nombres iguales. Esto es posible porque si tienen el mismo nombre, poseen una ruta de acceso diferente (es diferente el pathname).

Cuando un nombre de archivo comienza con un (.) (como en .arch), se dice que es un archivo oculto (hidden), ya que no es desplegado al utilizar comando ls. Sin embargo el comando **ls -a**, despliega estos archivos.

Es importante destacar que el signo de interrogación (?), el asterisco (*). los paréntesis de llave ({}), y los corchetes ([ ]) nunca deben ser usados en nombres de archivos pues tienen un significado especial para el sistema.

2.2.3.-Directorios

Un directorio es el lugar donde los archivos están almacenados (conceptualmente, no físicamente). Un directorio posee información sobre los archivos de sistema que se encuentran dentro de él. Contiene sus nombres y sus números de inodo. Unix usa internamente inodos para organizar su sistema de archivos. Hay un inodo por cada archivo. Los inodos contienen información sobre los archivos. La información incluye el tipo de archivos, su tamaño, la ubicación, último acceso y última modificación además de los permisos ya mencionados.

De igual forma que los archivos ordinarios, los archivos de directorio pueden ser protegidos utilizando permisos de acceso adecuados (lectura, escritura y ejecución).

Dado que un sistema Unix existen múltiples usuarios trabajando sobre múltiples proyectos, el número de archivos puede crecer rápidamente. Como una forma de ayuda a controlar este crecimiento el sistema utiliza un esquema organizacional llamado estructura del árbol.

Todos aquellos archivos que poseen características comunes pueden ser agrupados bajo un directorio. Un directorio puede contener archivos y otros directorios, estos últimos reciben el nombre de subdirectorio. Un subdirectorio puede contener, a su vez, subdirectorios y archivos. El comando **cd** es usado para desplazarse de un directorios a otro.

En este árbol de archivos típico, la raíz se encuentra en el lugar tope. Los directorio corresponden a nodos en el árbol y los archivos ordinarios corresponden a las hojas del mismo. La siguiente figura representa una estructura de árbol.

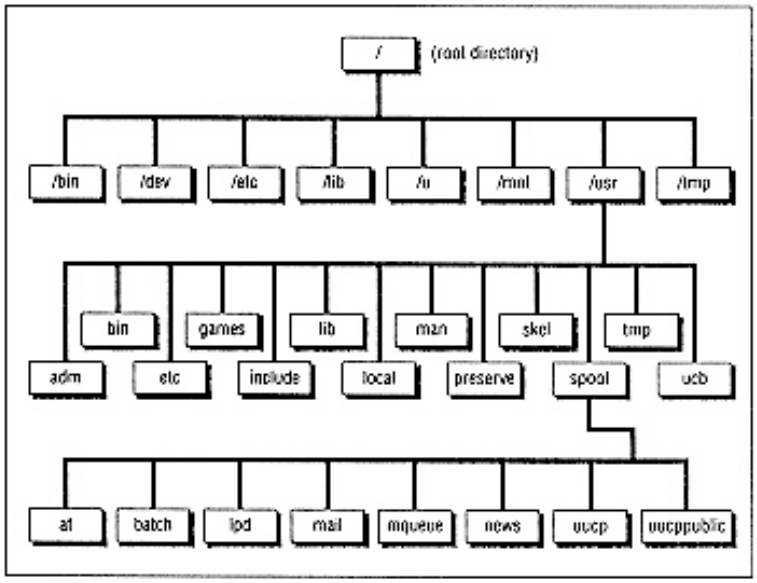


Figure 2-6. BSD Directory Structure

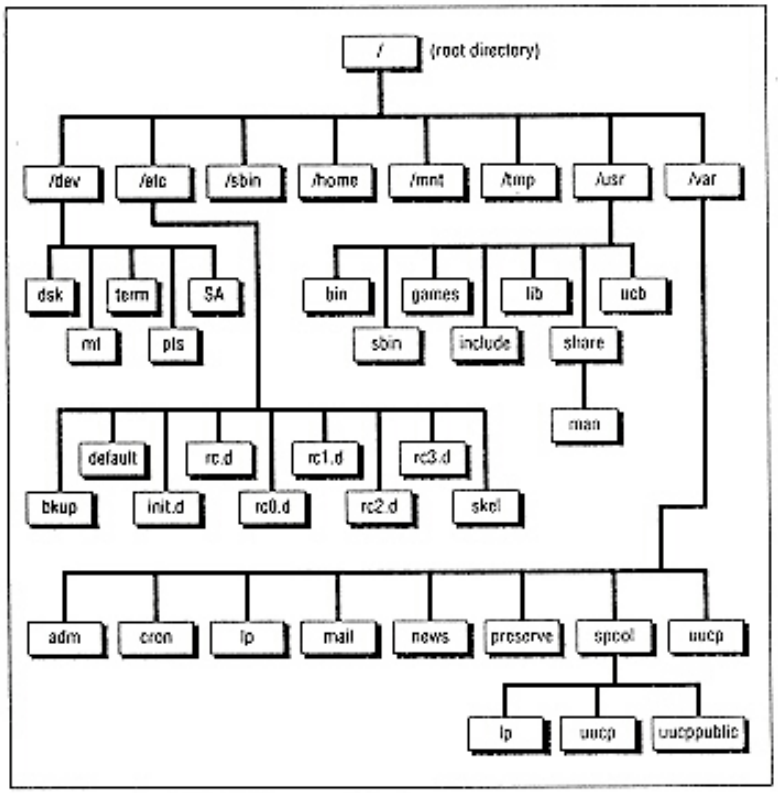


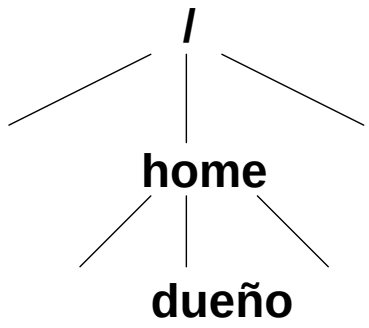
Figure 2-7. Typical V.4 Directory Structure

En la figura anterior, los nombres sbin, usr, dev, home y tmp representan directorios. Son los nodos en el árbol. En el tope del árbol se encuentra el directorio raíz (root), el cual recibe el nombre de slash

Es posible identificar cualquier archivo en forma inequívoca, si se describe toda la ruta (path) entre la raíz y el archivo a identificar, incluyendo su nombre.

Por ejemplo : **/usr/news/text.**

El directorio del usuario. Cada usuario de Unix posee un directorio personal (home directory). Este es el lugar donde Ud. puede guardar sus archivos. Dentro del directorio personal se pueden construir subdirectorios, que serán de su propiedad y estarán bajo su control. Es común que los directorios personales se encuentren bajo el directorio usr, tal como se puede apreciar en la siguiente figura.



2.2.4.- Rutas (pathnames)

Una ruta (pathname) es una secuencia de nombres de directorio seguida por un nombre de archivo, cada uno separado de los anteriores por un slash (/). Si la ruta comienza en slash, especifica un archivo que se encuentra comenzando la búsqueda en la raíz del árbol. En caso contrario, el archivo se encuentra buscando en el directorio actual (current directory) del usuario (también llamado directorio de trabajo, working directory). El comando **pwd** sirve para desplegar el nombre del directorio de trabajo en pantalla.

Una ruta que comienza con slash se llama ruta completa (full pathname) o absoluta (absolute pathname). La ruta absoluta indica donde se encuentra el archivo en el sistema. Una ruta absoluta es única: no hay dos archivos, directorios o dispositivos con exactamente la misma ruta absoluta. Si la ruta no comienza con slash, se llama relativa (relative pathname), ya que especifica una ruta relativa al directorio actual.

2.2.5.- Ejemplos de nombres de archivos

Entre los directorios y archivos comúnmente encontrados en sistemas Unix se cuentan:

/	El directorio raíz.
/bin	El directorio que contiene los comandos de uso más frecuente.
/usr	El directorio que contiene programas para usuarios. El directorio /usr/bin contiene otros comandos que no están en /bin.
/home	El directorio que contiene los directorios de los usuarios.
/dev	El directorio que contiene archivos especiales correspondiente a los dispositivos del sistema.
/dev/console	El archivo especial que corresponde a la consola del sistema.
/dev/ttyxx	El archivo especial asociado a una puerta del sistema. XX es un número, como **006 o la**. La mayor parte de las puertas están asignadas a terminales.
/lib	Un directorio que contiene “bibliotecas” (libraries) usadas para desarrollo del sistema.
/usr/local	Directorio donde se almacenan programas instalados por el administrador para satisfacer necesidades locales.
/usr/lib	Directorio que contiene bibliotecas para desarrollo de aplicaciones Unix
/tmp	El directorio para archivos temporales.
bin/script	Un nombre relativo. Se refiere al archivo **script** en el subdirectorio bin del directorio actual.
archivo1	El nombre de archivo1 en el directorio actual.

Todos los archivos y directorios, con la única excepción del directorio raíz, tienen un directorio “padre”. Este es el directorio inmediatamente por encima del archivo o directorio dado. Para el directorio actual hay notaciones especiales en Unix:

.	El directorio actual. Por ejemplo, ./**archivito**, es lo mismo que archivito, un nombre en el directorio actual.
..	El padre del directorio actual. Por ejemplo, ../.. se refiere al directorio dos más arriba que el actual.

2.2.6.- Manejo de Archivos

Hemos visto algunos comandos de Unix, ls, lp, rm. El comando ls lista nombres de archivos, el comando lp sirve para imprimir archivos, y rm borra archivos. En esta sección veremos algunos comandos adicionales, además de formalizar algo la descripción de los comandos anteriores.

⇒ **Manipulando Directorios**

Para crear un directorio nuevo se usa el comando **mkdir**. Este comando se usa de la siguiente forma:

mkdir nombre

En ese caso, queda creado un directorio llamado nombre dentro del directorio actual. Para ser más preciso, nombre es la ruta del directorio a crear, puede ser tanto una ruta absoluta como una relativa.

Por ejemplo, considere los comandos siguientes :

cd
mkdir algo
ls -l

El primer cd lo lleva al directorio propio, en el cual tiene pleno derecho a crear nuevos subdirectorios; el comando **mkdir** crea un directorio llamado algo y, finalmente, ls muestra el contenido del directorio actual. La opción **-l** de **ls** hace que muestre información más detallada, resultando algo como :

```
 total 1
drwxr-xr-x   2      dueño        512          Jul    10     11:14  algo/
```

Para eliminar un directorio se usa **rmdir** directorio. Eso sí, el directorio en cuestión debe estar vacío. Siguiendo el ejemplo anterior, podemos hacer lo siguiente :

mkdir otro
touch otro/nada

El comando **"mkdir** *otro"* crea un directorio adicional, y el comando **touch** entonces crea el archivo **"nada"** dentro de él. Ahora, al hacer :

ls -l

el sistema corresponde algo similar a:

```
total 2
drwxr-xr-x   2      dueño        512    Jul    10     11:14  algo/
drwxr-xr-x   2      dueño        512    Jul    10     11:39  otro/
```

Para ver el contenido del nuevo directorio se ejecuta:

ls -l otro

Se obtiene:

```
total 0
-rw-r--r--    1    dueño    0    Jul    10    11:39   nada
```

Nótese que el archivo nada tiene largo 0, vale decir, no contiene nada. Si ahora se ejecuta :

rmdir algo otro

Este comando instruye al sistema a eliminar ambos directorios. El sistema responde:

rmdir: otro: Directory not empty

al ejecutar:

ls -l

el sistema responde:

```
total 1
drwxr-xr-x    2    dueño    512    Jul    10    11:40   otro/
```

Vale decir, el directorio algo, que estaba vacío, fue eliminado; mientras el directorio otro, que contiene el archivo **"nada",** permanece. Continuado el ejemplo, tenemos:

rm otro/nada
rmdir otro

Ahora sí desaparece el directorio otro, como es fácil de verificar.

⇒ **Moviendose entre directorios**

Ejecute el comando **"pwd"**

$ pwd
/home/usuario

Cree un nuevo subdirectorio

$ mkdir nuevo

Para entrar y salir de un directorio, se ocupa el comando **"cd"**

```
$ cd  nuevo
$ pwd
/home/usuario/nuevo
$ cd . .
$ pwd
/home/usuario
```

Note que entre el **"cd"** y el **". ."** existe un espacio necesario, a diferencia del MS-DOS.

⇒ **Redireccionamientos**

Una de las gracias de Unix es la posibilidad de redireccionar entradas y salidas de los comandos. En particular, se puede:

- Enviar la salida de un comando a un archivo
- Agregar la salida de un comando a un archivo
- Enviar la salida de un comando a otro comando como entrada
- Tomar la entrada de un comando desde un archivo

Es importante recalcar que éstos pueden aplicarse a cualquier comando de exactamente la misma forma. Trataremos cada una de estas en la que sigue.

Para enviar la lista de los archivos en /etc. al archivo listita puede ejecutarse:

```
ls /etc > listita
ls -l listita
```

El signo mayor (>) indica al sistema que la salida del comando se envíe al archivo mencionado, luego el primer **ls** no despliega nada. El segundo **ls** desplegará información sobre el archivo listita, algo como:

```
-rw-r--r--   1   dueño   865   JUL   13   10:28   listita
```

Al hacer **cat** listita rápidamente pasa una larga lista de archivos. En realidad, lo que se obtiene es exactamente lo mismo que con **ls** /etc. Para poder ver esta lista con más calma, puede usarse **more** *listita*, que despliega el archivo listita una página a la vez. Al presionar la barra de espacio avanza a la página siguiente, al presionar <RETURN> avanza una línea.

Se obtiene el mismo resultado anterior usando:

```
more < listita
```

Acá se invoca a **more**, pero no se le indica un archivo a desplegar. En tal caso, **more** despliega lo que se le entrega en la entrada; y el signo menor que (<) da el archivo listita como entrada a **more**.

El efecto de los comandos anteriores, que podemos resumir mediante :

ls /etc > listita
more listita

podría haberse logrado también mediante :

ls /etc | more

En este caso estamos indicando mediante la barra vertical (|) (también llamada pipe, ya que crea una cañería de conexión entre comandos) que la salida de ls se entregue como entrada a **more**.

Una manera simple de crear un archivo es usando **cat**, aprovechando que si no se dan archivos a **cat**, éste escribe en la salida lo que viene de su entrada, en este caso el teclado :

cat > archivo
bla bla bla
bla bla
bla
y bla
^d

En lo anterior, **^d** representa presionar juntas las teclas marchadas Control de d. Para Unix esto representa fin de archivo. Para corroborar lo hecho, con **cat** *archivo* (o **more** archivo para los más avezados) se comprueba el contenido del archivo. Nótese que si el archivo ya existe, >lo borra y luego escribe.

Fuera de lo anterior, se puede agregar la salida de un comando al final de un archivo mediante >>. Usando el mismo archivo anterior:

cat >> archivo
Ahora agregamos texto adicional
al archivo.
Al diferencia del texto anterior,
esta, esta en castellano.
^d

Usando **cat** o **more** puede verificarse el nuevo contenido del archivo.

Nótese que usar cat para escribir un archivo puede ser útil en algunas situaciones particulares, aunque Unix tiene facilidades mucho más elaboradas para crear archivos. Estamos usando este simple por ahora, más adelante veremos el uso del editor **vi**.

⇒ Copiar y Mover

Otros comandos importantes son los que permiten copiar y mover archivos. Por copiar entendemos crear un archivo, cuyo contenido es idéntico al original. El comando para esto es:

"cp fuente destino"

En cambio, mover un archivo significa registrarlo bajo una ruta (posiblemente sólo cambiarle el nombre). Acá el comando es:

" mv fuente destino"

En ambos casos, si el destino es un directorio, el archivo queda con el nombre original bajo el directorio destino. Además, en esa situación pueden copiarse o moverse múltiples archivos con un comando.

Por ejemplo, luego de lo anterior un **ls -l** entrega algo como:

```
total 3
-rw-r--r--   1   dueño   141   Jul   13   14:16   archivo
-rw-r--r--   1   dueño   865   Jul   13   10:28   listita
drwxr-xr-x   2   dueño   512   Jul   10   10:44   otro
```

Si ahora se ejecutan los comandos siguientes:

```
cp archivo archivo2
cp listita listita2
ls -l
```

Vale decir, se copia archivo en archivo2 y se copia listita en listita2. El sistema responde al último **ls** con algo similar a lo siguiente:

```
total 5
-rw-r--r--   1   dueño   141   Jul   13   14:16   archivo
-rw-r--r--   1   dueño   141   Jul   13   14:26   archivo2
-rw-r--r--   1   dueño   865   Jul   13   10:28   listita
-rw-r--r--   1   dueño   865   Jul   13   14:26   listita2
drwxr-xw-r   2   dueño   512   Jul   10   11:44   otro
```

Usando **cat** o **more** es fácil verificar que los archivos realmente contienen lo mismo.

Si ahora se ejecutan los comandos:

mv archivo archivo3
ls -l

se obtiene un resultado como:

```
total5
-rw-r--r--  1  dueño  141  Jul  13  14:16  archivo3
-rw-r--r--  1  dueño  141  Jul  13  14:26  archivo2
-rw-r--r--  1  dueño  865  Jul  13  10:28  listita
-rw-r--r--  1  dueño  865  Jul  13  14:26  listita2
drwxr-xw-r  2  dueño  512  Jul  10  11:44  otro
```

Nótese que desapareció archivo y apareció archivo3, y que archivo3 tiene exactamente los mismos atributos que tenía archivo. O sea, archivo cambió de nombre, pasando a ser archivo3.

El comando **mv** sirva para mover archivos, un caso muy particular de lo cual es cambiarles de nombre. Por ejemplo, si queremos darle el nombre pepito al archivo que actualmente se llama archivo3 y verificar el resultado, basta ejecutar:

mv archivo3 pepito
ls -l

con lo que se obtiene:

```
total 7
drwxr-xr-x  3  dueño  512   Aug  10  11:08  .
drwxr-xr-x  4  dueño  1024  Aug  10  11:10  ..
-rw-r--r--  1  dueño  141   Jul  13  14:26  archivo2
-rw-r--r--  1  dueño  865   Jul  13  10:28  listita
-rw-r--r--  1  dueño  865   Jul  13  14:26  listita2
drwxr-xw-r  2  dueño  512   Jul  10  11:44  otro
-rw-r--r--  1  dueño  141   Jul  13  14:16  pepito
```

2.3.- Metacaracteres

Unix permite usar abreviaturas para referirse a un conjunto de nombres de archivo. Suponga, por ejemplo, que usted está escribiendo el apunte para un curso de Unix. Los distintos capítulos del apunte podrán estar en archivos **Unix1, Unix2, Unix3**, y así sucesivamente. Incluso podría dividir un capítulo en varios archivos, como ser **Unix2.1, Unix2.2**, etc.

Si quiere imprimir el apunte completo, podría usar el siguiente comando :

lp Unix1 Unix2. 1 Unix2. 2 ...

Escribir tantos nombres de archivos es tedioso, además que probablemente lleve a cometer errores. Para situaciones como éstas hay un conjunto de caracteres que hacen de "comodines" (wildcard). Discutiamos el significado de :

*	Calza con cero o más caracteres
[ ]	Calza con uno de los caracteres incluidos entre los corchetes
?	Calza con un caracter cualquiera

En el ejemplo anterior, podría haber escrito también:

lp Unix*

El asterisco (*) significa "cero o más caracteres de cualquier tipo", de manera que esto se traduce en "envíe todos los archivos cuyos nombres comienzan con Unix a la impresora". Así tiene una manera cómoda para imprimir todo su apunte.

Estas abreviaturas no son propias del comando **lp**, pueden usarse con cualquier comando. Para desplegar en pantalla los nombres de los archivos que componen su apunte puede escribir :

ls Unix*

Por lo demás, el asterisco no tiene por qué aparecer en la última posición del nombre. El comando :

ls *.c

lista los nombres de archivos que terminan en .c.

El asterisco no es la única alternativa. Si sólo quiere imprimir los capítulos 1 a 4 y 9, pueden dar el comando :

lp Unix [1-49]*

Nótese que se pueden abreviar rangos consecutivos, como 1-4 en el ejemplo. Lo mismo vale para letras, aunque Unix hace una distinción entre las letras mayúsculas y minúsculas. Así todas las letras se escribiría [a-zA-A].

El signo de interrogación (?) calza a cualquier caracter. Si queremos información acerca de los primeros archivos de cada capítulo de nuestro apunte, podemos decir :

ls Unix?.1

Hay situaciones en las que uno quiere “apagar” el significado especial de los caracteres anteriores (típicamente porque de “alguna misteriosa manera” apareció un archivo llamado a*...). Para resolver este problema específico, podemos escribir :

rm “a*”

El comando **rm** borra archivos, y el usar los apóstrofes alrededor del nombre indica que debe considerarse tal cual.

3- Permisos

Cada archivo y directorio tienen asociados diferentes permisos, estos permisos pueden ser de lectura, escritura y ejecución, y pueden ser datos sobre tres universos de usuarios: el dueño (quien creó el archivo), el grupo (aquellos usuarios que fueron definidos en el mismo grupo que el dueño del archivo) y el resto de los usuarios. Estos permisos se pueden ver con el comando **ls** y la opción **-l**.

ls -l

con lo que se obtiene:

```
total 31
drwxr-xr-x   2   juan    512    Nov  10  20:11  bin
-rw-rw-rw-   1   juan    315    Nov  10  20:36  carta.Erika
-rwxr-xr-x   1   juan    155    Nov  12  14:00  fondos
-rwxr-xr-x   1   juan     30    Nov  10  20:10  lbs
-rw-r--r--   1   juan     19    Nov   4  20:56  notas
drwxr-xr-x   2   juan    512    Nov  10  20:11  personal
-rw-rw-rw    1   juan    23800  Nov  14  16:41  unix.apunte
drwxr-xr-x   2   juan    512    Nov  10  21:22  varios
```

El conjunto de letras que aparecen a la izquierda corresponde a los permisos, cada letra tiene un cierto significado.

r: lectura
w: escritura
x: ejecución

La primera letra señala si el archivo es un directorio (**d**) o archivo (-), el siguiente grupo de tres caracteres corresponde a los permisos para el dueño del archivo, los siguientes 3 a los del grupo y los últimos 3 a los del resto de los usuarios.

d	**rwx**	**rwx**	**rwx**
Directorio	**Dueño**	**grupo**	**Resto de usuarios**

Cuando en lugar de la letra correspondiente aparece un signo ‘-‘ significa que dicho permiso no está asignado, así por ejemplo **“unix.apunte”** tiene permisos de lectura y escritura

para el dueño, el grupo, y el resto de los usuarios, por su parte, el archivo **"notas"** puede ser leído y escrito por el dueño y sólo leído por el grupo y el resto de los usuarios.

El cambio de estos permisos es hecho mediante el comando **chmod** el cual tiene un formato de la forma:

chmod [quienes] que cuales archivo1 archivo2...

quienes:

u: usuario (dueño)
g: grupo
o: otros
a: todos (usuario, grupo y otros)

que:

+: agregar permisos
-: quitar permisos
=: asignar permisos

cuales:

r: lectura
w: escritura
x: ejecución

Las indicaciones de **quienes, que** y **cuales** permisos deben ser dadas en un sólo parámetro (sin espacios entre ellos), así, si quisiéramos quitar el permiso de escritura para el grupo y otros usuarios al archivo **unix.apunte** se puede hacer con el comando:

chmod go-w *unix.apunte*

Si quisiéramos dejar todos los archivos (y directorios) con permiso de lectura y escritura para el grupo se daría el comando:

chmod g=rw *

Similarmente, si queremos agregarle permiso de ejecución para el dueño al archivo *comandos* se puede dar el comando:

chmod u+x *comandos*

En caso que la indicación de quienes sea omitida, se asume que se refiere a todos, por lo que, si queremos dejar el archivo *unix. apunte* sólo un permiso de lectura para todos los usuarios, basta dar el comando:

chmod =r *unix.apunte*

Otra forma de especificar los permisos de un archivo es utilizando números octales, lo que consiste en sumar simultáneamente un 4 para lectura, un 2 para escritura y un 1 para ejecución.

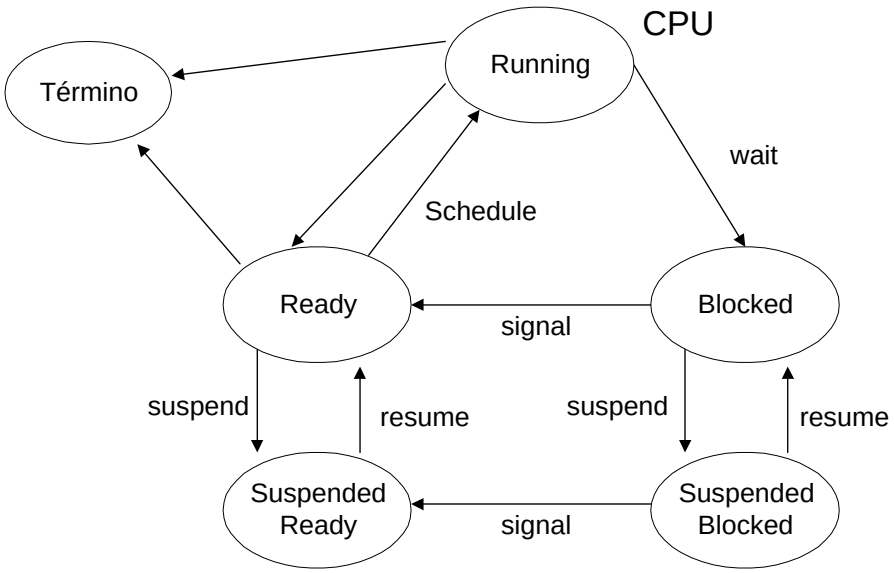
Ejemplos:

chmod 666 archivo1 **equivale a** **chmod = rw archivo1**
chmod 755 archivo2 **chmod = rx archivo2 y**
chmod u + w archivo2

4.-Procesos

Como bien se dijo al comienzo, Unix es un sistema operativo multitarea, es decir, que cada usuario puede estar ejecutando varios comandos a la vez, esto es útil sobre todo cuando se tienen programas que demoran mucho en ejecutar (como una simulación o la obtención de resultados por métodos numéricos). A cada programa que se está ejecutando en el computador se la llama **Proceso**.

Todo proceso pasa por el siguiente ciclo:



Hasta el momento, en cada comando que hemos visto, Unix espera que termine para ejecutar el siguiente. Es posible indicar a Unix que luego del comando, sin importar que no haya terminado, despliegue el prompt para poder ejecutar otro comando más (en forma simultánea), esto se hace colocando el caracter '&' (Ampersand) al final del comando, así por ejemplo, si tenemos el programa calculo_largo y no queremos esperar hasta que termine para poder ejecutar otros comandos, podemos dar dicho comando de la forma:

$ calculo_largo &

Pero probablemente este programa escribe sus resultados en la pantalla, lo que interfiere con los siguientes comandos que ejecutemos (sólo en su presentación en pantalla), por lo que se recomienda, en este caso, redireccionar la salida del programa.

$ calculo_largo >salida.calculo &

Para consultar sobre los procesos que tenemos ejecutando en nuestra cuenta podemos utilizar el comando **ps**:

$ ps

con lo que se obtiene:

PID	**TT**	**STAT**	**TIME**	**COMMAND**
140	00	**IW**	0:00	-bash
139	01	**IW**	0:02	-bash
157	01	**S**	0:00	vi unix.apunte
163	02	**S**	0:03	calculo_largo
171	01	**R**	0:00	ps
141	02	**IW**	0:00	-bash

En la primera columna **(PID)** nos muestra un número identificador del proceso, en la columna **TT** se señala el terminal que está ocupando dicho comando (en este caso se está ocupando más de un terminal), la columna **STAT** nos muestra el estado de la ejecución, la columna **TIME** muestra el tiempo de microprocesador que ha utilizado y finalmente el comando que estamos ejecutando.

Tal vez estemos ejecutando un programa que demora mucho y hemos olvidado colocar el caracter **'&'** al final de la línea, podemos recuperar el control suspendiendo dicho proceso (detener temporalmente su ejecución) pulsando las teclas **Control** (Ctrl) y sin soltarla pulsar la tecla **Z,** con ello, la ejecución de nuestro programa queda detenida y nos aparece el prompt, en él podemos enviar una señal avisándole al proceso que queremos que se siga ejecutando, estas señales enviadas por el comando **kill** de la forma:

kill -señal *pid1*

Para este caso enviamos la señal **CONT** (continuar), suponiendo que el número del proceso (**PID**) era 163, el comando sería:

kill -CONT 163

Otro caso común es cuando ejecutamos un comando por error y está demorando demasiado, entonces queremos eliminarlo, generalmente sirve pulsar las teclas **Control** y C, lo que envía una señal de interrupción (**INT**) al proceso que se está ejecutando, si esto falla habrá que enviarle una señal terminar (**TERM**) desde otro shell (una conexión a la misma cuenta desde otro terminal, por ejemplo), la señal **TERM** es la que el comando **kill** utiliza cuando no se le dice que señal enviar, por lo que bastaría con el comando:

kill 163

Si esto también fallara, tenemos un último recurso, la señal destruir (**KILL**):

kill -KILL 163

Las versiones más antiguas de Unix no aceptan la indicación de la señal con letras, por lo que debe ser indicada con números según la siguiente tabla:

STOP	=	17
CONT	=	19
TERM	=	15
INT	=	2
KILL	=	9

Con esto : kill **-KILL** 163 es equivalente a kill -9 163.

5.- C-Shell (csh)

El C-Shell es uno de los tanto intérpretes de comando que existen para UNIX. Son equivalentes al “commando.com” del MS-DOS y entre los destacados están:

sh	Bourne Shell
csh	C-Shell
ksh	Korn Shell
bash	Bourne Again Shell

Otros son: tcsh yVisual shell.

C-Shell es un intérprete de comandos (Shell), más poderoso que el standard **Bourne Shell** provisto con Unix.

En este capítulo se comentarán algunas de las facilidades proporcionadas por C-Shell, tales como: History, que es una forma cómoda de repetir comandos y Jobs, una forma más fácil de manejar procesos, pero un tanto más restringida.

5.1.- History

Esta capacidad que incorpora C-Shell nos permite usar palabras de los comandos escritos previamente, en este capítulo veremos sólo su uso más sencillo que consiste en repetir algún comando.

Podemos ver la lista de comandos que “recuerda” C-Shell dando el comando:

history

el que nos lista lo siguiente:

1 ls -F
2 cat carta.Erika
3 rm reunion.memo
4 cd apuntes
5 vi unix.txt

Se dispone de dos formas para repetir alguno de estos comandos, la primera requiere que demos el comando **history** y nos fijemos en el número que aparece a la izquierda del comando que deseamos repetir. Para repetir el comando **cd apuntes** basta el comando de repetición de la forma:

!4

La segunda alternativa es más amigable, sólo necesita las primeras letras del comando, buscando hacia atrás el primer comando que coincida con ellas, así, para repetir el comando **cd apuntes** se puede indicar de la forma:

!cd

O equivalentemente:

!c

ya que es último comando realizado que comienza con la letra **'c'**.

5.2.- Alias

Alias es una forma de definir sinónimos a los comandos del sistema o incluso redefinir los existentes con alguna más complicada (definirlos con otros comandos es posible, pero no es gracioso cuando otra persona define el comando **cp** como **rm** sin advertirlo).

Si queremos definir un nuevo comando que borre archivos, por ejemplo: **borrar** lo podemos hacer de la forma:

alias borrar rm

Ya que rm es el comando Unix para borrar archivos.

Un caso muy utilizado es el de redefinir **ls** por **ls -f.**

Con ello, cada vez que se utilice el comando ls se entenderá como ls -f.

Se puede obtener una lista de los alias definidos dando el comando **alias** sin parámetros:

alias

Para borrar alguna definición de alias, se hace por medio del comando unalias, especificando el alias que deseamos eliminar:

unalias borrar

Borrará el alias **borrar.**

5.3.- Jobs

Esta es una forma más práctica que nos provee C-Shell para el manejo de procesos, pero sólo de aquellos ejecutándose en dicho C-Shell.

Para consultar los procesos que tenemos en dicho C-Shell se debe dar el comando:

jobs

con lo que se obtiene:

[1]	**+**	**Running**	**ls -l**
[2]		**Running**	**calculo_largo**
[3]	**-**	**Running**	**who**

Nos muestra a la izquierda (entre paréntesis de corchetes) un número de identificación del trabajo, para referenciar alguno de ellos se debe colocar dicho número precedido del signo % , también pueden ser referenciados con un signo % seguido de las primeras letras del comando (al igual que con el comando **history**, salvo que en este caso se busca hacia adelante), además pueden distinguirse un proceso marcado con el signo '+', el cual se utiliza en caso de omitir la identificación del proceso o puede especificarse como %+ o %% ; y otro marcado con el signo '-'. referenciado como %-.

Ahora veremos una lista de los comandos de ***C-Shell*** con respecto al manejo de procesos:

jobs [-1] Muestra la lista de procesos del C-Shell en curso, al dar la opción -1 muestra además el **PID** (El número identificador de proceso mostrado por el comando **ps**)

stop [%job] Detiene el proceso *job*

fg [%job] Continúa la ejecución del proceso job, pero espera hasta que éste termine (ejecución foreground)

%job Equivalente a **fg** *job*

bg [%job] Continúa la ejecución del proceso job dejando al **C-Shell** disponible inmediatamente para dar otros comandos (ejecución background)

%job & Equivalente a **bg** *job*

kill [-señal] [%job] Envía una señal al proceso job, de la misma forma explicada en el capítulo 6, salvo que la especificación del proceso puede ser realizada por esta forma sencilla provista por C-Shell

Así, si queremos detener el proceso ls -la > salida, puede ser realizado de cualquiera de las siguientes formas:

- **stop**
- **stop %%**
- **stop %+**
- **stop %1**
- **stop %ls**

Por otra parte, si queremos destruir el proceso **calculo_largo >salida.calculo_largo**, puede ser hecho de las formas:

- **kill -KILL %2**
- **kill -KILL %calc**
- **kill -KILL %c**

6.- Manuales en Línea

Unix nos provee de una ayuda que, por lo general, es bastante completa, y con ella podemos aprender muchos comandos de Unix sin necesidad de tener un pesado manual a mano. Si queremos saber más sobre el comando **ls** podemos hacerlo mediante el comando:

man ls

El comando que nos proporciona la ayuda es el comando man, al cual se le da como argumento el tópico sobre el cual queremos averiguar. Esta ayuda la muestra por medio del comando **more** (ver comando **more**), puede obtener una ayuda rápida de él pulsando la tecla **h.**

También podemos encontrar una lista de los tópicos relacionados con una palabra por medio del comando **apropos** dándole como parámetro la palabra sobre la cual queremos encontrar información, por ejemplo:

apropos gcc

con lo que obtenemos:

gcc (1) - GNU project C and C++ Compiler (v2.7)
gcc (1) - GNU project C and C++ Compiler (v2.7)

El número que muestra entre paréntesis corresponde a la sección, en caso de haber ambigüedad (el mismo tópico en dos secciones distintas), se puede dar la sección correspondiente al comando **man,** por ejemplo:

man 1 gcc

7.- Indice de comandos más utilizados en Unix

Comando	Descripción	Sintaxis
apropos	Muestra comandos asociados a la palabra especificada	**apropos** comandos
at	Ejecuta un comando o un archivo de comandos en la fecha y hora indicados	**at** hora [día] [archivo]
cal	Muestra un calendario del año y mes especificados	**cal** [[mes] año]
cat	Muestra el contenido de los archivos especificados	**cat** [archivo1 archivo2 ...]
cd	Cambia el directorio de trabajo	**cd** [directorio]
chmod	Cambia los permisos de los archivos y/o directorios especificados	**chmod** [quienes] que cuales archivos ...
comm	Compara dos archivos	**comm** [-1][-2][-3] archivo1 archivo2
cp	Copia archivos	**cp** archivo_fuente archivo_destino **cp** arch1arch2...directorio_destino
diff	Muestra las diferencias entre dos archivos	**diff** archivo1 archivo2
du	Muestra el tamaño (Bytes) que ocupa un directorio o conjunto de archivos	**du** [-s] [directorio y7o archivos]
echo	Muestra un mensaje	**echo** [-n] mensaje
expr	Evalúa una expresión	**expr** expresión
file	Indica la clasificación de los archivos	**file** archivo1 archivo2
find	Permite buscar un archivo o conjunto de archivos en una ruta especifica	**find** ruta_inicio -name nombre_a_buscar -print
grep	Muestra las líneas que cumplen con cierto patrón en los archivos	**grep** patrón archivo1 archivo2
kill	Envía una señal de término a un proceso	**kill** [-señal] proceso1 proceso2
ln	Referencia un archivo con otro nombre	**In** [-s] archivo_existente nombre_enlace
lpr	(incluso en otro directorio)	**Ipr** archivo1 archivo2 ...
ls	Imprime archivos	**ls** [-altF] [archivos y/o directorios]
mail	Muestra la lista de archivos	**mail** [usuario1 usuario2 ...]
man	Envía o recibe correo	**man** comando
mesg	Muestra manual en línea del comando	**mesg** y (Permite)
	Permite o prohibe la recepción de mensajes	**mesg** n (Phohibe)

Comando	Descripción	Sintaxis
mkdir	Crea un directorio	**mkdir** directorio
more	Muestra un archivo en forma pausada (de a una página)	**more** archivo1 archivo2 ...
mv	Mueve (o renombra) archivos	**mv** archivo_viejoarchivo_nuevo mv arch1 arch2 ... directorio
passwd	Cambia password actual	**passwd**
ps	Despliega el estado de los procesos	**ps** [-alxu]
pwd	Muestra el actual directorio de trabajo	**pwd**
rm	Borra archivos	**rm** [-ir] archivo1 archivo2
rmdir	Borra directorios	**rmdir** directorio1 directorio2 ...
sort	Ordena archivos	**sort** [-bfnr] [-tx] [campos] [archivo (s)]
talk	Conversa con otro usuario	**talk** usuario@nodo
wc	Informa el número de líneas, palabras y/o caracteres de los archivos	**wc** [-clw] archivo1 archivo2 ...
who	Muestra la lista de usuarios conectados	**who** [am `i`]
write	Envía un mensaje a otro usuario	**write** usuario [tty]

8.- Anotaciones