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

A LOS ASISTENTES A LOS CURSOS

Las autoridades de la Facultad de Ingeniería, por conducto del jefe de la División de Educación Continua, otorgan una constancia de asistencia a quienes cumplan con los requisitos establecidos para cada curso.

El control de asistencia se llevará a cabo a través de la persona que le entregó las notas. Las inasistencias serán computadas por las autoridades de la División, con el fin de entregarle constancia solamente a los alumnos que tengan un mínimo de 80% de asistencias.

Pedimos a los asistentes recoger su constancia el día de la clausura. Estas se retendrán por el periodo de un año, pasado este tiempo la DECFI no se hará responsable de este documento.

Se recomienda a los asistentes participar activamente con sus ideas y experiencias, pues los cursos que ofrece la División están planeados para que los profesores expongan una tesis, pero sobre todo, para que coordinen las opiniones de todos los interesados, constituyendo verdaderos seminarios.

Es muy importante que todos los asistentes llenen y entreguen su hoja de inscripción al inicio del curso, información que servirá para integrar un directorio de asistentes, que se entregará oportunamente.

Con el objeto de mejorar los servicios que la División de Educación Continua ofrece, al final del curso deberán entregar la evaluación a través de un cuestionario diseñado para emitir juicios anónimos.

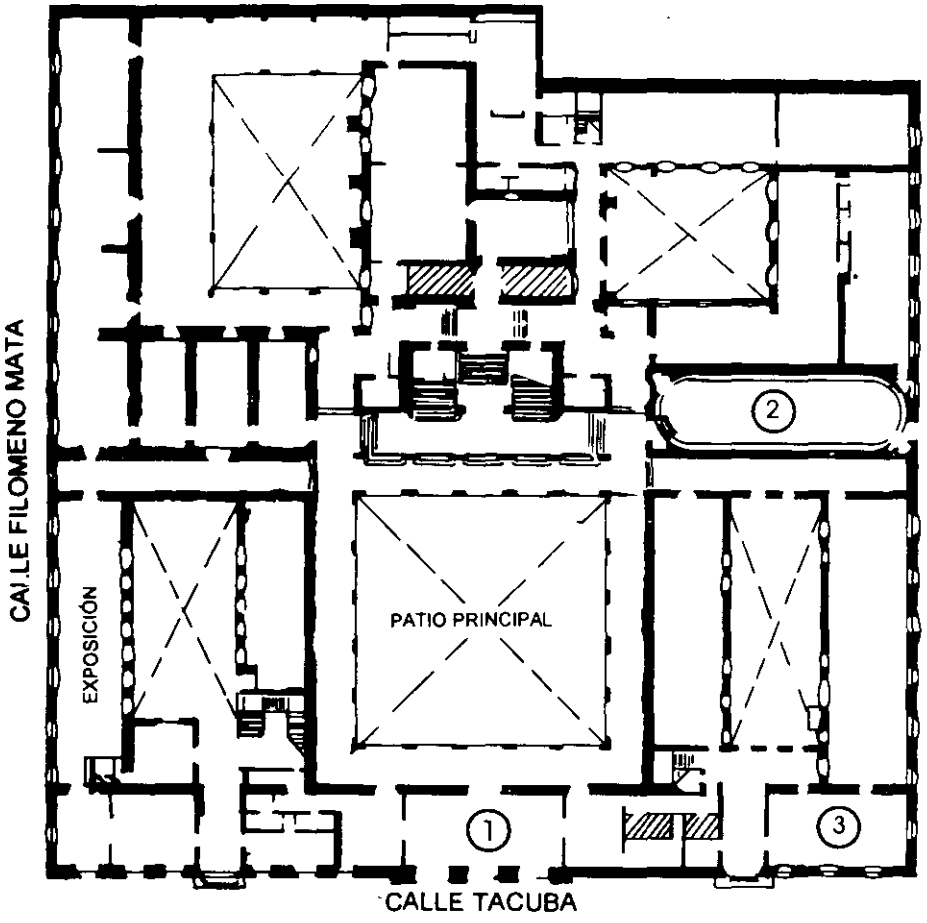
Se recomienda llenar dicha evaluación conforme los profesores impartan sus clases, a efecto de no llenar en la última sesión las evaluaciones y con esto sean más fehacientes sus apreciaciones.

Atentamente

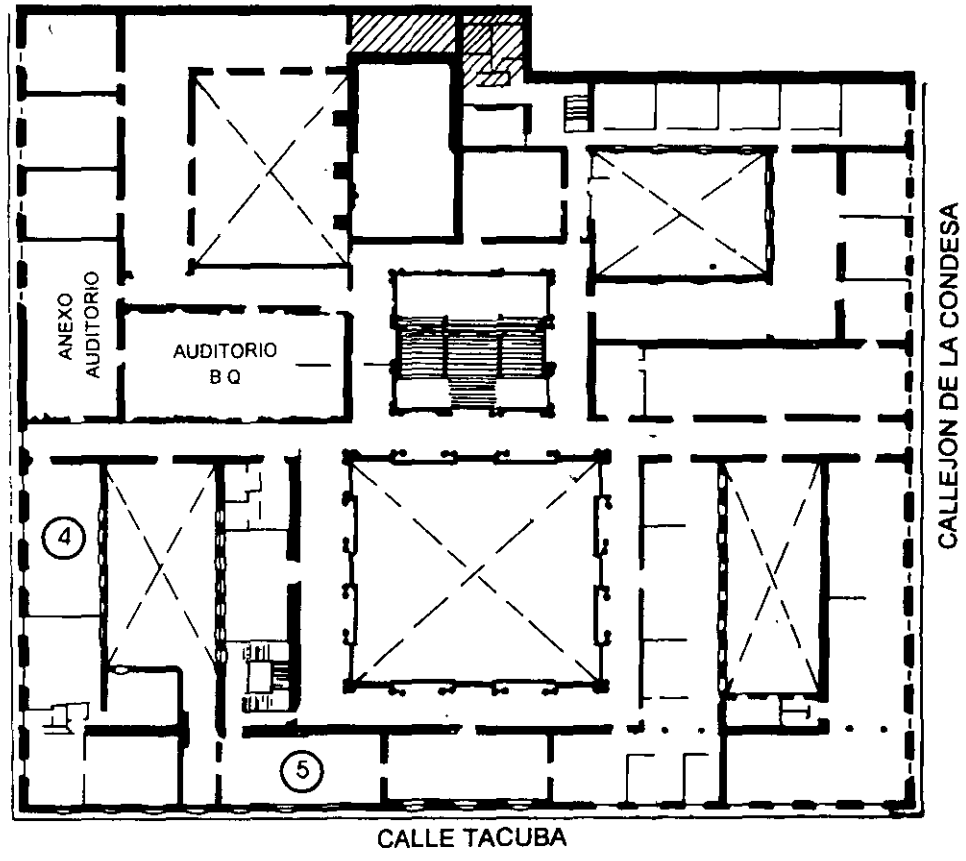
División de Educación Continua.

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. APDO. Postal M-2285
Teléfonos: 512-8955 512-5121 521-7335 521-1987 Fax 510-0573 521-4020 AL 26

PALACIO DE MINERIA

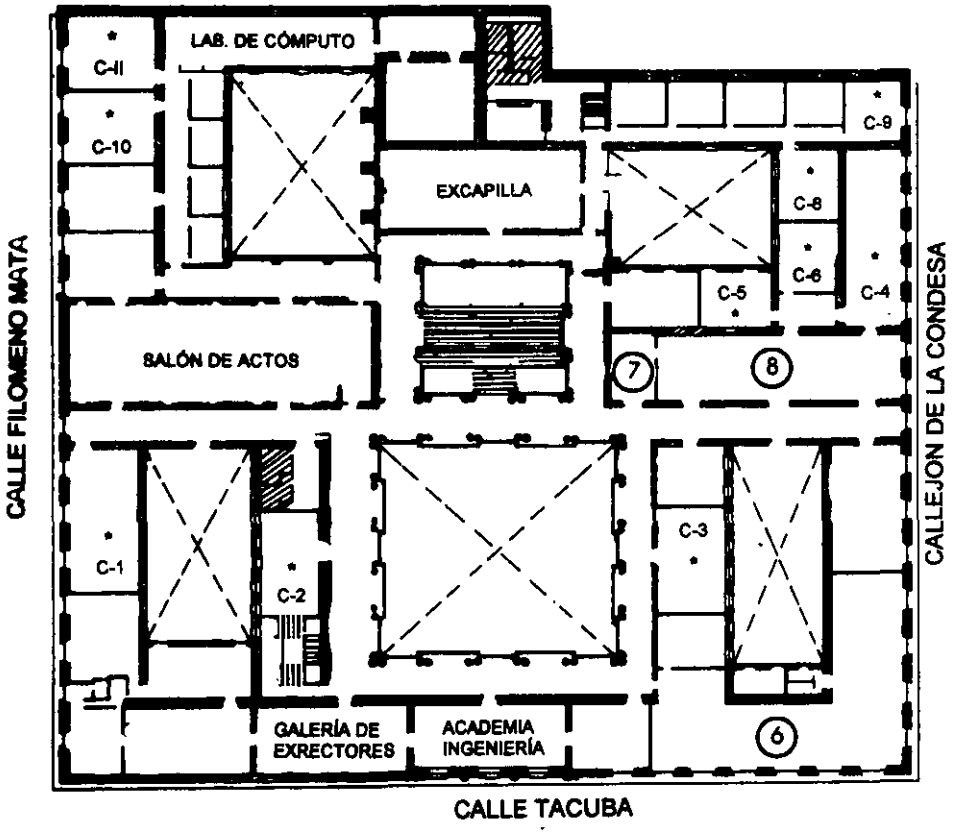


PLANTA BAJA



MEZZANINNE

PALACIO DE MINERIA



1er. PISO

GUÍA DE LOCALIZACIÓN

1. ACCESO
2. BIBLIOTECA HISTÓRICA
3. LIBRERÍA UNAM
4. CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN "ING. BRUNO MASCANZONI"
5. PROGRAMA DE APOYO A LA TITULACIÓN
6. OFICINAS GENERALES
7. ENTREGA DE MATERIAL Y CONTROL DE ASISTENCIA
8. SALA DE DESCANSO

SANITARIOS

* AULAS

DIVISIÓN DE EDUCACIÓN CONTINUA
FACULTAD DE INGENIERÍA U.N.A.M.
CURSOS ABIERTOS

DIVISIÓN DE EDUCACIÓN CONTINUA
DEC

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

FACULTAD DE INGENIERÍA

DIPLOMADO EN REDES WAN

UNIX PARA ADMINISTRACIÓN DE REDES

(CA091 Agosto de 1998)

PROFESORES:

ALEJANDRO ESPEJEL ROSALES
OLGA LIDIA TORRES RIVERA
CESAR VEGA CLADERON

ale@ds5000.super.unam.mx
lidy@ds5000.super.unam.mx
cvc@ds5000.super.unam.mx

UNIX PARA ADMINISTRACIÓN DE REDES

UNIX BÁSICO

INTRODUCCIÓN

Definición de sistema operativo

Un *sistema operativo* es un conjunto de programas que controlan el sistema de E/S de una computadora (como teclado y unidad de disco) y que carga y ejecuta otros programas. Es también un conjunto de mecanismos y políticas útiles para compartir de manera ordenada los recursos del sistema. Además, un sistema operativo actúa como una interfase entre una computadora y el usuario de la misma, ya que le permite interactuar con ella de una manera amigable y flexible.

Historia de UNIX

- UNIX es un sistema operativo multiusuario y multitarea. *Multiusuario* significa que permite que muchas personas usen la misma computadora al mismo tiempo; *multitarea* quiere decir que cada usuario puede ejecutar muchos programas diferentes simultáneamente.

- Las raíces de UNIX se remontan hasta mediados de los 60's, cuando la **AT&T** (American Telephone and Telegraph), **Honeywell**, **General Electric** y el **MIT** (Massachusetts Institute of Technology) iniciaron un proyecto denominado *Multics* (Multiplexed Information and Computing Service), financiado principalmente por la **DARPA** (Department of Defense Advanced Research Projects Agency). El sistema operativo Multics fue diseñado pensando en la seguridad militar, para resistir ataques externos y proteger entre sí a los usuarios del sistema. Multics soportaba el concepto de *seguridad multinivel*, esto es, poder tener en la misma computadora información altamente secreta, secreta, confidencial y no clasificada.

- En 1969, AT&T decide salir del proyecto Multics, debido a que éste ya estaba muy retrasado en cuanto a tiempo y sus creadores habían prometido más de lo que podían ofrecer. Ese mismo año, *Ken Thompson*, un investigador de AT&T que había participado en el proyecto Multics, prosiguió con las idea de este nuevo sistema en una máquina PDP-7. *Dennis Ritchie*, quien también había trabajado en Multics, se unió a Thompson. *Brian Kernighan* sugirió el nombre UNIX para el nuevo sistema, en contraposición a Multics, ya que mientras Multics intentaba hacer muchas cosas, UNIX intentaba hacer bien sólo una cosa: ejecutar programas.

- En 1971, Thompson y Ritchie reescribieron UNIX para la "nueva" computadora PDP-11 de Digital Equipment.

- Al paso del tiempo, UNIX vino evolucionando gracias a las mejoras añadidas por ambos científicos. UNIX estaba basado en pequeños programas o herramientas que ejecutaban una sola función. Sin embargo, juntando estas herramientas, los programadores podían hacer cosas más complicadas cada vez.

- En 1973, Thompson reescribe UNIX en el lenguaje de programación C, recién inventado por Ritchie. C era un lenguaje simple y portátil, es decir, los programas escritos en C podían llevarse fácilmente de una máquina a otra.

- En 1973 ya había unas 25 computadoras diferentes corriendo UNIX en los Laboratorios Bell.

- UNIX comenzó a hacerse popular en universidades y empresas. A principios de 1974, la Universidad de California, en Berkeley, compró una copia de UNIX para su nueva computadora PDP-11/45. Sin embargo, en lugar de sólo utilizarlo, dos estudiantes de Berkeley, *Bill Joy* y *Chuck Haley*, comenzaron a modificar el código fuente de UNIX. Así nació UNIX BSD (Berkeley Software Distribution), del cual se liberaron unas 30 copias en 1977. A parir de entonces, y durante los siguientes 6 años, UNIX BSD fue mejorándose (financiado por la DARPA) con respecto al UNIX de AT&T, por lo que se hizo muy popular entre las comunidades académica y de investigación.

- Actualmente, UNIX corre en millones de computadoras, desde PC's hasta supercomputadoras, y se utiliza en ambientes educativos, de investigación y hasta comerciales.

Características importantes de UNIX

- Es un sistema operativo multiusuario.
- Es un sistema operativo multitarea.
- Está escrito en un lenguaje de alto nivel (C).
- Dispone de un lenguaje de control programable (sh, csh, ksh).
- Permite la creación de programas de aplicación y de sistemas.
- Emplea manejo dinámico de memoria.
- Permite la comunicación entre procesos, brindando protección a archivos, cuentas y procesos.
- Permite la comunicación entre usuarios.
- Emplea un sistema jerárquico de archivos.
- Usa un manejo consistente de archivos de diversos tipos.
- Permite el redireccionamiento de E/S.
- Tiene un alto grado de portabilidad.
- Puede reconfigurarse en cada instalación para adaptarse a requerimientos de configuración de hardware particulares.

Estructura general de UNIX

UNIX es un sistema operativo estructurado por 3 capas: el *núcleo*, las *interfaces de usuario* y los *programas de aplicaciones* (ver figura 1):

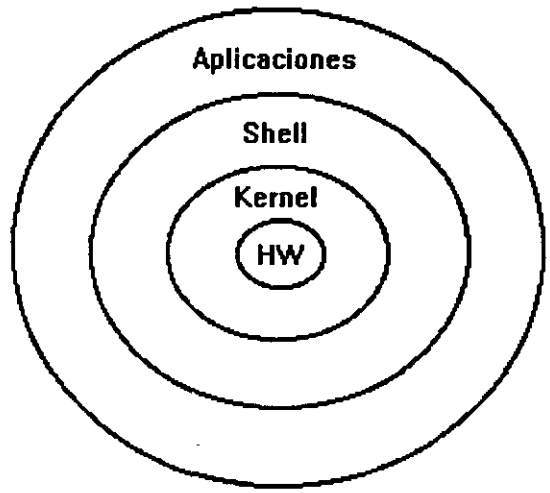


Figura 1. Estructura en capas del sistema operativo UNIX.

El núcleo o kernel

Es el componente central de UNIX. Es un programa de unas 10, 000 líneas escrito casi totalmente en C, excepto por una parte escrita en ensamblador, encargada de manejar algunas interrupciones propias del procesador. El kernel se comunica directamente con el hardware a través de interrupciones y excepciones.

La función principal del núcleo es atender a varios usuarios y múltiples tareas concurrentemente, lo cuál logra repartiendo el tiempo del procesador entre todos ellos, dando la ilusión tanto a usuarios como a tareas de que cada una de ellos esta siendo atendido de forma individual.

Otras funciones del núcleo son:

1. Crear procesos, asignarles tiempo de atención y sincronizarlos.
2. Asignar la atención del procesador a los procesos que lo requieran.
3. Administrar el espacio en el sistema de archivos, lo cual incluye:

--- Acceso, protección y administración de usuarios.
--- Comunicación entre usuarios y entre procesos.
--- Manipulación de E/S y administración de dispositivos periféricos.

4. Supervisar la transmisión de datos entre la memoria principal y los periféricos.

El núcleo reside siempre en la memoria principal (RAM) y controla la computadora, por lo que ningún otro proceso puede interrumpirlo, aunque si puede recibir llamadas para proporcionar algunos servicios. Un proceso llama al kernel mediante módulos especiales conocidos como *llamadas al sistema.*

El kernel consiste de dos partes principales:

1. *Sección de control de procesos.* Asigna recursos, programa procesos y atiende sus requerimientos de servicio.
2. *Sección de control de dispositivos.* Supervisa la transferencia de datos entre la memoria principal y los dispositivos periféricos.

Cuando se inicia la operación de la computadora se debe cargar en memoria una copia del núcleo, que generalmente reside en el disco duro (a esta operación se le conoce como **bootstrap**).

Interfaces de usuario

Existen diferentes tipos de interfaces, cada una de ellas con características especiales que permiten al usuario comunicarse con el sistema. Algunos tipos son:

- Bourne Shell
- C Shell
- Korn Shell

Aunque existen diversas interfaces, podemos hablar en general de un **shell**. Un shell es un lenguaje de control, un intérprete y un lenguaje de programación.

Como *lenguaje de control*, un shell nos permite:

- Modificar las características con que se ejecutan los programas en UNIX.
- Configurar diversos ambientes de ejecución.
- Automatización de tareas.

Como *lenguaje de programación*, un shell tiene la siguientes características:

- Sustitución de metacaracteres (caracteres con un significado especial para UNIX).
- Control del medio ambiente, a través de las variables de ambiente.
- Redireccionamiento de la E/S.
- Interconexión de procesos.
- Procesamiento de archivos.
- Uso de variables y de estructuras de control.

Finalmente, como *intérprete*, el shell nos permite comunicarnos con la máquina, proporcionarle datos e interpretar los resultados devueltos.

Programas de usuario y aplicaciones

UNIX incluye múltiples esquemas para crear, editar y procesar documentos, a través de una gran cantidad de programas de utilerías.

Un programador puede crear sus propias aplicaciones en lenguajes tales como C, FORTRAN, C++ o Perl (incluso en ensamblador). Existe además un gran número de bibliotecas de funciones que permiten extender dichas aplicaciones, para lo cual el programador sólo debe incluirlas en su programa.

Actualmente opera una gran variedad de bases de datos bajo ambiente UNIX, tales como Informix, Oracle y Sybase.

Comienzo de una sesión

Para que un usuario pueda acceder a una terminal de computadora, el administrador del equipo debe encender la máquina y después de que el sistema ha sido iniciado se pueden encender las terminales, las cuáles permitirán que los usuarios y la máquina se comuniquen.

El administrador del sistema debe crear las cuentas o claves de los usuarios, otorgándoles un nombre y una contraseña. El *nombre (login)* generalmente es un apellido del usuario o bien, sus iniciales, en tanto que la *contraseña (password)* debe ser una mezcla de caracteres (números, letras, símbolos especiales, signos de puntuación, etc.) sin significado propio, por cuestiones de seguridad.

Una vez que el sistema esta encendido, aparecerá en la terminal la palabra **"login: "**, y la máquina esperará hasta que se teclee el nombre asignado al usuario. Si éste nombre es correcto, el sistema solicita ahora el password del usuario por medio del mensaje **"password: "**, el cuál debe ser tecleado entonces (no se sorprenda si al teclearlo no aparece nada en pantalla o únicamente aparece una serie de asteriscos (*****), esto es por razones de seguridad). Si el login o el password son incorrectos, el sistema envía al usuario un mensaje de error haciéndoselo saber; de lo contrario, el sistema mostrará un mensaje de bienvenida (no necesariamente) y luego aparecerá cualquiera de los siguientes símbolos (quizá seguidos o antecedidos por otras palabras): **$, %, #**. Estos símbolos indican que el usuario se encuentra en una sesión interactiva del sistema (shell), que para este caso se encargará de interpretar los comandos ú órdenes que el usuario teclee.

Ejecución de comandos

Para que el sistema operativo UNIX pueda ejecutar alguna orden o comando, es necesario presentarlo en el formato o sintaxis correcta:

% comando [opciones] [argumentos]

En dónde:

comando	*Es el nombre de la orden que desea ejecutarse.*
opciones	*Determinan una manera especifica en que va a ejecutarse el comando.*
argumentos	*Especifican los datos con que va a operar el comando (generalmente son el nombre de un archivo o directorio).*

Para ejecutar un comando éste debe teclearse luego del símbolo $, % ó # (denominado *prompt*) y presionarse la tecla <ENTER>.

Para detener la ejecución de una orden o comando deben presionarse simultáneamente las teclas <CTRL> y <C> o la tecla <DEL> o la tecla <BREAK> solas; después de esto aparece nuevamente el símbolo prompt, lo cuál significa que UNIX se encuentra en espera de alguna nueva orden o comando.

Salida de sesión

Para salir del shell basta con teclear al mismo tiempo las teclas <CTRL> y <D> o teclear la palabra **"exit"** o la palabra **"logout"**. Luego de algunos segundos, UNIX despliega nuevamente la palabra **"login: "**, lo cuál implica que se encuentra en espera de que otro usuario ingrese al sistema.

EL SISTEMA DE ARCHIVOS

Introducción

El sistema de archivos de UNIX controla la forma en que se almacena la información guardada en archivos y directorios en el disco, además de que usuarios pueden acceder y la forma en que lo hacen.

El sistema de archivos de UNIX está basado en un modelo arborescente y recursivo, en donde los nodos pueden ser tanto archivos como directorios, y estos últimos pueden contener otros directorios (subdirectorios).

La raíz del sistema de archivos (*root*) se denota con el símbolo /, y de ahí se desprende un conjunto de directorios que contiene todos los archivos del sistema. Cada directorio, a su vez, funciona como la subraíz de un nuevo árbol que depende de él, y que también puede estar formado por directorios y archivos. Un archivo siempre ocupará el nivel más bajo dentro del árbol, porque de un archivo no pueden depender otros; si así fuera, sería un directorio, es decir, los archivos son hojas de árbol.

Los directorios de UNIX pueden contener archivos, nombres lógicos que representan dispositivos, ligas simbólicas y otros directorios. Desde un punto de vista simple, todo lo que un sistema de archivos almacena son archivos, aunque en realidad lo que se almacena son **i-nodos** (un i-nodo es la unidad básica de un sistema de archivos que almacena todo sobre un archivo, excepto su nombre).

UNIX almacena la siguiente información general sobre cada archivo:

- La ubicación del contenido del archivo en el disco.
- El tipo de archivo.
- El tamaño del archivo en bytes.
- La fecha y hora en que fue modificado por última vez el i-nodo del archivo (*ctime*).
- La fecha y hora en que fue modificado por última vez el contenido del archivo (*mtime*).
- La fecha y hora en que fue accedido por última vez el archivo (*atime*).
- El número de nombres que tiene el archivo (conteo de referencias).

UNIX también almacena la siguiente información de seguridad para cada archivo:

- El dueño del archivo (UID).
- El grupo del archivo (GID).
- Los bits de permisos o simplemente permisos de archivo.

Nombres de los archivos

Los nombres para archivos de UNIX pueden ser de varios caracteres, dependiendo del sistema operativo de que se trate (hasta 14 en UNIX System V). Son válidos todos los caracteres del código ASCII, con excepción de los caracteres especiales para UNIX, tales como:

/ ( ) ; * ! & < > ? y el espacio en blanco.

Rutas en los archivos

Se define en forma unívoca el nombre de todo archivo o directorio mediante lo que se conoce como **ruta** (*path*): el conjunto completo de directorios, iniciando desde la raíz (/), por los que hay que pasar para poder llegar al archivo o directorio en cuestión. Cada nombre se separa de los otros con el símbolo /, aunque sólo el primero de ellos se refiere a la raíz.

El sistema de archivos de UNIX mantiene varios directorios para su uso propio, como por ejemplo:

/bin *Contiene programas ejecutables y utilerías.*
/etc *Contiene programas y archivos de datos para el administrador del sistema.*
/dev *Contiene archivos especiales.*
/tmp *Contiene archivos temporales que pueden ser creados por cualquier usuario.*
/usr *Contiene los directorios de los usuarios, incluyendo mail, bin y news.*

Es importante mencionar que debido a que UNIX es un sistema multiusuario, cada uno de los usuarios del sistema deberá tener un espacio asignado dentro del árbol. Los administradores generalmente usan los sistemas de archivos /home, /export/home, /user/users, etc. para reservar espacio para sus usuarios, de tal forma que cada usuario tiene un directorio asignado para poder guardar ahí sus archivos. A este directorio se le conoce como **directorio home**.

Dos conceptos importantes en la terminología de sistemas de archivos son los de directorio actual y directorio padre (denotados por . y .., respectivamente). La idea básica es que un punto (.) siempre se refiere al *directorio actual de trabajo*, en tanto doble punto (..) se refiere al directorio inmediatamente superior al directorio actual o *directorio padre*. Por ejemplo, supongamos que entramos a sesión y tecleamos la siguiente secuencia de comandos (ver figura 2):

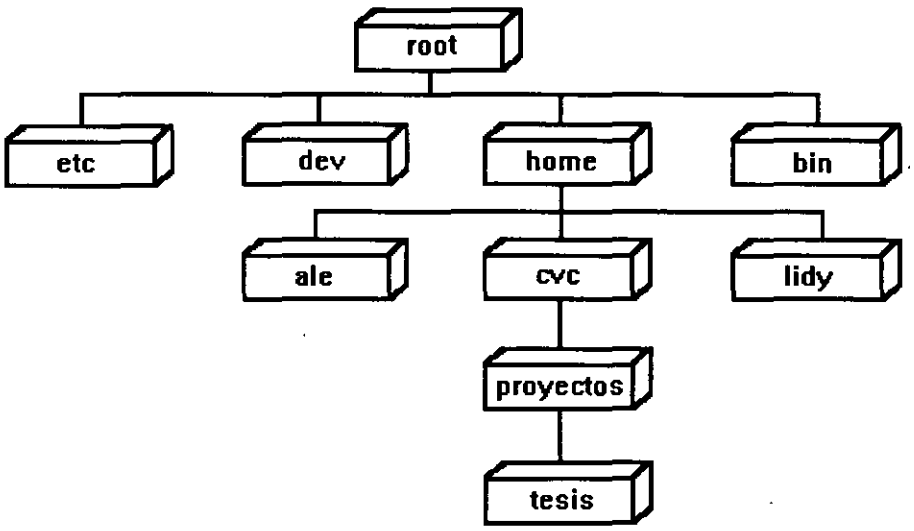


Figura 2. La estructura jerárquica del sistema de archivos de UNIX.

% **pwd** *(el comando pwd muestra el directorio actual de trabajo)*
/home/cvc

% cd proyectos — *(el comando cd se usa para cambiar de directorio de trabajo)*

% pwd
/home/cvc/proyectos

% cd tesis

% pwd
/home/cvc/proyectos/tesis

Si estando en esta ruta tecleamos lo siguiente:

% cp *.doc .. *(el comando cp copia archivos de un directorio a otro)*

Copiaremos todos los archivos del directorio actual de trabajo con extensión .doc al directorio /home/cvc/proyectos.

De igual forma, si estamos en /home/cvc/proyectos:

% tar cvf tesis.tar ./tesis *(el comando tar sirve para respaldar un conjunto de archivos)*

Entonces guardaremos en el archivo tesis.tar un respaldo de todos los archivos contenidos en el directorio /home/cvc/proyectos/tesis (incluyendo los subdirectorios contenidos en él).

Rutas absolutas y relativas

Puede hacerse referencia a un archivo mediante su ruta, la cuál puede ser absoluta o relativa con respecto al directorio donde uno se encuentre actualmente.

La *ruta absoluta* se define como el nombre que identifica a un archivo o directorio del sistema de archivos a partir del directorio raíz. Por ejemplo, /home/cvc/proyectos/tesis.

La *ruta relativa* permite al usuario referirse a archivos o directorios sin necesidad de mencionar la ruta absoluta, sino simplemente haciendo referencia a dicho archivo desde el directorio actual de trabajo. Por ejemplo, si nos encontramos en el directorio /home/cvc/proyectos y deseamos ver el contenido del archivo bookmark.html, ubicado en el directorio home, basta con teclear el siguiente comando:

% cat ../bookmark.html *(el comando cat muestra el contenido de un archivo)*

en lugar de

% cat /home/cvc/bookmark.html

Permisos de los archivos

Cuando se da de alta un nuevo usuario en un sistema, se le asigna un número único de identificación (UID) y el número del grupo al que pertenece (GID). Cuando dicho usuario crea un archivo, éste se marca con el UID y el GID. De igual forma, se le asignan 10 bits de protección, 9 de los cuáles especifican permisos de lectura, escritura y ejecución para el propietario, para miembros del grupo y para el resto de los usuarios del sistema. El primer bit especifica el tipo de archivo de que se trata.

Los permisos de los archivos indican el tipo de archivo y el tipo de acceso que se concede a los usuarios del sistema.

Los siguientes son ejemplos de permisos de archivos:

-rw-------
drwxr-xr-x

El primer caracter indica el tipo de archivo. Los tipos de archivos en UNIX son:

Contenido	***Significado***
-	*Archivo plano*
d	*Directorio*
c	*Dispositivo de caracteres (terminal o impresora)*
b	*Dispositivo de bloques (disco o cinta)*
l	*Liga simbólica (sólo en UNIX BSD)*
s	*Socket (sólo en UNIX BSD)*
p	*FIFO (sólo en UNIX System V)*

Los siguientes 9 caracteres acomodados en grupos de 3 indican quien puede hacer que en el sistema. Hay 3 tipos de permisos:

r *Permiso de lectura*
w *Permiso de escritura*
x *Permiso de ejecución*

También hay 3 tipos de grupos:

dueño *El dueño del archivo*
grupo *Los usuarios que están en el mismo grupo del dueño del archivo*
otros *Los usuarios restantes (excepto el superusuario)*

Ejemplo:

-rwxr-x---

Aquí se conceden permisos de lectura, escritura y ejecución al dueño del archivo, permisos de lectura y ejecución para los miembros del grupo del dueño del archivo y ningún permiso para el resto del mundo.

Los permisos de acceso a un archivo difieren si se trata de un archivo ordinario o de un directorio. La siguiente tabla muestra esta diferencia:

Permiso	*Archivo*	*Directorio*
r	Permiso para ver el contenido del archivo	Permiso para listar el contenido del directorio
w	Permiso para modificar el contenido del archivo	Permiso para cambiar el contenido del directorio
x	Permiso para ejecutar un archivo	Permiso para ingresar a un directorio

OPERACIONES DE ENTRADA Y SALIDA

Introducción

El sistema de E/S de UNIX se divide en 2 sistemas complementarios: el *sistema de E/S por bloques* y el *sistema de E/S por caracteres*. El primero se emplea para el manejo de discos y cintas magnéticas, mientras que el segundo se usa para atender terminales e impresoras.

En general, UNIX emplea programas especiales (escritos en C) conocidos como **controladores** o manejadores (*drivers*) para atender a cada familia de dispositivos de E/S. Los procesos se comunican con los dispositivos mediante *llamadas* a su controlador. Además, desde el punto de vista de los procesos, los controladores aparecen como si fueran archivos en los que se lee o escribe, lográndose así una homogeneidad y elegancia en el diseño.

Las llamadas del sistema de más bajo nivel hacen referencia a un flujo de entrada y salida a través de un número de identificación llamado **descriptor de archivo** (*file descriptor* o simplemente *fd*).

Por convención, cada proceso UNIX tiene asignados 3 canales de E/S (ver figura 3):

1. **Entrada estándar** (fd = 0), generalmente asociada con el teclado
2. **Salida estándar** (fd = 1), generalmente asociada con la pantalla
3. **Salida de errores estándar** (fd = 2), generalmente asociada con la pantalla

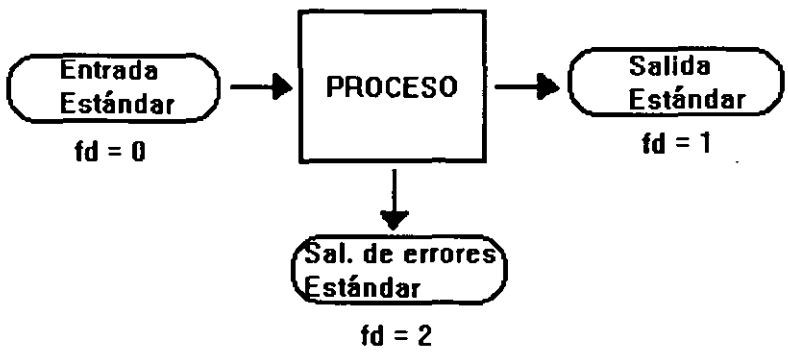


Figura 3. Canales de E/S asociados a un proceso en UNIX.

Redireccionamiento

La **redirección** es una de las características más útiles de UNIX. Permite tomar la información que generalmente se envía a la pantalla y canalizarla a un lugar distinto (tal como un archivo); o bien, tomar la información que generalmente proviene del teclado de otro lugar (otro archivo).

Un argumento del siguiente tipo

< *archivo_entrada*

provoca que el archivo llamado *archivo_entrada* se lea como la entrada estándar.

Un argumento de la forma

> *archivo_salida*

provoca que el archivo llamado *archivo_salida* sea utilizado como la salida estándar. Si este archivo ya existía, y tiene permisos de escritura, su contenido se pierde.

Un argumento de la forma

>> *archivo_salida*

redirecciona la salida estándar hacia el archivo llamado archivo_salida, pero respetando su contenido anterior, esto es, añadiendo la información.

2> *archivo_temporal* ó 2>> *archivo_temporal* (sh)
>& *archivo_temporal* ó >>& *archivo_temporal* (csh)

redirecciona la salida de errores al archivo denominado archivo_temporal, reemplazando o añadiendo la información existente.

Ejemplos:

% **cat < /etc/passwd**
(Toma el archivo /etc/passwd como entrada y lee su contenido, mostrándolo en pantalla)

% **who >** *usuarios*
(Envía al archivo *usuarios* un listado de los usuarios que actualmente se encuentran en sesión)

% **w >> usuarios**
(Añade al archivo *usuarios* un listado más detallado de los usuarios que actualmente se encuentran en sesión)

$ **ls** *archivo_inexistente* **2>** *mensaje_error*
(Lista las características del archivo *archivo_inexistente*, enviando los mensajes de error generados al archivo *mensaje_error*)

Comunicaciones entre procesos

Pueden ejecutarse dos o más comandos en la misma línea, separándolos por el caracter ;, sin necesidad de teclear cada comando en una línea. Por ejemplo:

% **clear ; date** > *fecha* ; **cat** *fecha*; **rm** *fecha*

La línea anterior borra la pantalla, guarda la fecha actual del sistema en el archivo llamado fecha, muestra el contenido de dicho archivo y, finalmente, borra este archivo.

El operador **pipeline**, representado por el símbolo |, conduce la salida de un comando hasta otro comando. Por ejemplo, supongamos que alguien desea saber cuantos usuarios se encuentran actualmente en sesión; basta con usar la línea siguiente:

% who | wc -l *(el comando who muestra un listado de los usuarios que actualmente se encuentran utilizando el sistema; el comando wc, con la opción -l, cuenta el número de líneas de un archivo)*

Esto es, la salida del comando who se convierte en la entrada del comando wc.

Supongamos ahora que un usuario quiere saber cuantos archivos tiene en su directorio actual de trabajo. Si no existiese la comunicación entre procesos en UNIX, el usuario debería teclear los siguientes comandos:

% **ls** > *numero_archivos* *(el comando ls lista todos los archivos del directorio actual)*

% **wc -w** *numero_archivos* *(el comando wc, con la opción -w, cuenta el número de palabras que contiene un archivo)*

% **rm** *numero_archivos* *(el comando rm borra el archivo especificado)*

Sin embargo, gracias al operador pipeline, sólo es necesario teclear la siguiente línea:

% **ls | wc -w**

EDICIÓN DE TEXTO EN UNIX

Introducción

Los editores de UNIX permiten crear y modificar archivos de texto que pueden contener cartas, documentos, programas de computadora, etc. UNIX cuenta con varios editores de texto, entre los que destacan por su uso:

- ed
- vi
- emacs

Existen básicamente dos tipos de editores en UNIX: *editores de línea* y *editores de pantalla*. En los primeros el contenido de un archivo se muestra y edita línea por línea, en tanto en los segundos es posible editar texto en una pantalla completa, por lo que uno puede desplazarse libremente por todo el archivo (hacer *scroll*). El editor ed es un ejemplo de un editor de línea, mientras que vi ejemplifica los editores de pantalla.

El editor de pantalla **vi** (*visual editor*) es una herramienta usada para la creación y modificación de archivos de texto bajo ambiente UNIX.

Cualquier modificación hecha al texto se logra colocando el cursor en el lugar adecuado. Los cambios realizados se reflejan inmediatamente en el cuerpo del texto, por lo que es fácil apreciar el formato de la información.

Invocación del vi

El editor vi puede invocarse mediante cualquiera de los siguientes comandos:

% **vi** *nombre_archivo*

Si el archivo existe, se muestra la primera parte del mismo y el cursor se coloca en la primera línea. Sí no existe, se creará automáticamente.

% **vi** *nombre_archivo_1 nombre_archivo_2 nombre_archivo_3 ...*

Cuando se invoca al editor con más de un nombre de archivo como argumentos, se despliega el primero en la pantalla y se puede conmutar entre ellos posteriormente.

% **vi** + *nombre_archivo*
Se despliega en la pantalla la última parte del archivo y el cursor se coloca al principio de la última línea.

% **vi** +*n nombre_archivo*
El cursor se coloca al principio de la línea *n* del archivo, dónde *n* es un número entero no mayor al número total de líneas del archivo.

Modos de trabajo del editor vi

El editor vi tiene dos modos de trabajo: el *modo de comandos* y el *modo de inserción*. Al comenzar una sesión, vi se encuentra por omisión en el modo de comandos, esto es, en espera de la ejecución de algún comando. Para conmutar entre ambos modos se usa la tecla <ESC>.

Órdenes básicas

Inserción de texto

a *Insertar texto a la derecha del cursor*
i *Insertar texto a la izquierda del cursor*
I *Insertar texto al principio de la línea*
A *Insertar texto al final de la línea*
o *Inserta una línea abajo de la línea actual*
O *Inserta una línea arriba de la línea actual*

Movimiento en el texto

j	*Mueve el cursor una línea abajo*
k	*Mueve el cursor una línea arriba*
h	*Mueve el cursor un caracter a la izquierda*
l	*Mueve el cursor un caracter a la derecha*
<CTRL><F>	*Mueve el texto una página hacia adelante*
<CTRL><B>	*Mueve el texto una página hacia atrás*
1G	*Mueve el cursor al inicio de la primera línea del archivo*
G ó **:$<ENTER>**	*Mueve el cursor al inicio de la última línea del archivo*
:n<ENTER>	*Mueve el cursor al inicio de la línea* ***n***
w	*Va a la siguiente palabra*
e	*Va al final de la siguiente palabra*
b	*Va a l inicio de la siguiente palabra*
M	*Va a la mitad de la pantalla*
<SHIFT>j	*Une la línea actual con la línea de abajo*

Búsqueda y reemplazo de caracteres y cadenas

*/cadena_buscada***<ENTER>**	*Busca la cadena hacia adelante*
:/*cadena_buscada***<ENTER>**	*Busca la cadena hacia adelante*
?*cadena_buscada***<ENTER>**	*Busca la cadena hacia atrás*
:?*cadena_buscada***<ENTER>**	*Busca la cadena hacia atrás*
:s/*cadena_actual*/*nueva_cadena*/**g**	*Reemplaza* ***cadena_actual*** *por* ***nueva_cadena*** *sólo en la línea actual*
:%s/*cadena_actual*/*nueva_cadena*/**g**	*Reemplaza* ***cadena_actual*** *por* ***nueva_cadena*** *en todo el archivo*
r*nuevo_caracter*	*Reemplaza el caracter donde esta posicionado el cursor por* ***nuevo_caracter***
:?	*Repite la última búsqueda hacia atrás*
:??	*Repite la última búsqueda hacia adelante*

Comandos Undo y Repeat

u *Deshace el último cambio realizado*
U *Deshace el los últimos cambios realizados a la línea actual*
. *Repite el último comando*

Borrado y copiado de caracteres y líneas

x *Borra el caracter donde se encuentra el cursor*
*n***x** *Borra **n** caracteres, a partir de la posición actual del cursor*
dw *Borra la palabra actual*
*n***dw** *Borra **n** palabras, incluyendo la actual*
dd *Borra la línea actual*
*n***dd** *Borra **n** líneas, a partir de la actual*
*n***yy** *Copia **n** líneas, a partir de la actual*
p *Coloca las líneas borradas o copiadas abajo de la línea actual*
P *Coloca las líneas borradas o copiadas arriba de la línea actual*
D ó **d$** *Borra desde la posición del cursor hasta el final de la línea*
d) *Borra el párrafo siguiente*
d(*Borra el párrafo anterior*

Conmutando entre archivos

:n *Va al siguiente archivo abierto*
:e *nombre_archivo* *Conmuta al archivo **nombre_archivo***
:r *nombre_archivo* *Añade el contenido del archivo **nombre_archivo** al archivo actual*

Comandos varios

:! *comando_shell* Ejecutar un comando de UNIX sin salir del editor
:set number Enumera las líneas del texto durante la edición
:set nonumber Deshabilita la numeración de las líneas del texto
:set all Muestra las variables de ambiente del editor

Saliendo de vi

:w *Grabar el archivo sin salir del editor*
:wq ó **:x** *Grabar el archivo y salir del editor*
:q! *Salir del editor sin grabar los cambios*
:w! *Nombre_archivo* *Renombrar el archivo actual como **nombre_archivo***

COMANDOS BASICOS

pwd Este comando nos muestra la ruta del lugar en donde nos encontramos en el sistema.

Ejemplo:

```
lidy@aketzali:~> pwd
/home/lidy/Cursos/Unix_Admon
```

ls lista el contenido de directorios, ls tiene muchas opciones entre las más importantes se encuentran:

-l Muestra en formato largo
-a muestra los archivos ocultos
-F lista los directorios y archivos, a los directorios los marca con una / y a los ejecutables con *, normalmente.

Ejemplo:

```
lidy@aketzali:~/Cursos> ls -laF
total 7
drwxr-xr-x  7 lidy   users     1024 Aug 13 13:38 ./
drwx------ 19 lidy   users     1024 Aug 21 12:56 ../
drwx------  3 lidy   users     1024 Aug 13 17:17 Administracion_unix/
drwx------  6 lidy   users     1024 Aug 13 18:52 Introduccion_unix/
drwx------  2 lidy   users     1024 Aug 13 13:38 Programacion_en_Shell/
drwxr-xr-x  3 lidy   users     1024 Aug 21 15:57 Unix_Admon/
drwx------  6 lidy   users     1024 Aug 13 13:38 Utilerias_de_Unix/
-rwx------  6 lidy   users     1024 Aug 13 14:30 script1*
```

```
lidy@aketzali:~/Cursos> ls
Administracion_unix   Programacion_en_Shell  Utilerias_de_Unix
Introduccion_unix     Unix_Admon
```

```
lidy@aketzali:~/Cursos> ls -la
total 7
drwxr-xr-x  7 lidy   users     1024 Aug 13 13:38 .
drwx------ 19 lidy   users     1024 Aug 21 12:56 ..
drwx------  3 lidy   users     1024 Aug 13 17:17 Administracion_unix
drwx------  6 lidy   users     1024 Aug 13 18:52 Introduccion_unix
drwx------  2 lidy   users     1024 Aug 13 13:38 Programacion_en_Shell
drwxr-xr-x  2 lidy   users     1024 Aug 21 15:23 Unix_Admon
drwx------  6 lidy   users     1024 Aug 13 13:38 Utilerias_de_Unix
```

cd Nos cambia de directorio de trabajo

Sintaxis:

% cd Nombre_directorio

Ejemplo:

Si me encuentro en /home/lidy y me quiero cambiar al directorio Cursos:

% cd Cursos

Si ahora doy :

% pwd
/home/lidy/Cursos

Si me quiero regresar un nivel hacia arriba (donde estaba):

% cd ..

y ahora estoy en /home/lidy

Tambien me puedo cambiar a un directorio del cual conozco su ruta:

% cd Cursos/Unix_Admon

o

% cd ./Cursos/Unix_Admon

o

% cd $HOME/Cursos/Unix_Admon Si se que su ruta empieza con /home/lidy

Ejemplo 2:

Si me encuentro en el directorio /home/lidy/Cursos/Unix_Admon/temario y quiero cambiarme al directorio /home/lidy, puedo hacerlo asi:

% cd ../../../

Como a la vez /home/lidy es mi home, tambien puedo hacer:

% cd cd sin ningun parametro me lleva a mi home.

% cd $HOME

cat despliega el contenido de un archivo.

Sintaxis:

% **cat** [opciones] <archivo>

Nota: NO lo hace por paginas.

Ejemplo:

crear 2 archivos uno llamado prueba1 y otro prueba2
conteniendo soy prueba 1, soy prueba 2, respectivamente.

Ejecutar las siguientes lineas:

% **cat** prueba1>> prueba2

Verificar lo que ahora contiene prueba con:

% **cat** prueba2
soy prueba 2
soy prueba 1

Y lo que hizo fue sustituir el contenido de prueba2 por el de prueba1.

more Despliega el contenido de un archivo(s) por paginas.

Sintaxis:

more [opciones] <nombre_archivo>

Ejemplo:

% **more** /etc/passwd

cp Copia archivos

Sintaxis:

cp [opciones] fuente destino
cp [opciones] fuente... directorio

Entre su opciones mas importantes se encuentra la :

-r Que nos sirve para copiar recursivamente del directorio donde nos encontramos, hacia abajo, al destino.

Ejemplo:

% cp archivo_existente nuevo_archivo

nuevo_archivo contiene exactamente lo mismo
que archivo_existente.

% cp -r /etc $HOME Copia recursivamente todo
el directorio /etc a mi home
con todos sus archivos, directorios
y contenido de esos directorios.

% cp arch1_existente arch2_existente

Sobreescribe el arch2 con el contenido del arch1.
El contenido de arch1 se pierde.

mkdir

Crea directorios

Sintaxis:

mkdir [-p] [-m mode] [--parents] [--mode=mode] [--help][--version] dir...

Ejemplo:

% **mkdir** Cursos

rmdir

Borra directorios que no contengan archivos, si se
desea borrar recursivamente, ver rm -r.

Sintaxis:

rmdir [-p] [--parents] [--help] [--version] dir...

Ejemplo:

% ***rmdir*** Cursos

rm

Este comando te permite borrar archivos.

Sintaxis:

rm [-f] [-i] file ...
rm -r [-f] [-i] dirname . . . [file . . .]

Opciones:

-f Forza a que sea borrado un archivo.

-r Indica que es recursivo. Para borrar los archivos de un directorio.

Ejemplos:

```
pulque 3% ls
anibal                  dumpster
archivo1                mbox
com                     nodependencias-berenice32
dependencias            nodependencias-berenice8
dependencias-berenice32 nodependencias-sirio
dependencias-berenice8  pos
dependencias-sirio      w
```

Se quiere eliminar el archivo con nombre archivo1 entonces :

pulque 4% **rm** archivo1

Para borrar un directorio con todos sus archivos es:

```
pulque 19% ls -la
total 4
drwxr-xr-x   3 beca    user       512 Aug 21 15:59 .
drwxr-xr-x   9 beca    user      1024 Aug 21 15:57 ..
drwxr-xr-x   2 beca    user       512 Aug 21 15:59 Directorio

pulque 20% cd Directorio

pulque 21% ls -la

drwxr-xr-x   2 beca    user       512 Aug 21 15:59 .
drwxr-xr-x   3 beca    user       512 Aug 21 15:59 ..
-rw-r--r--   1 beca    user         0 Aug 21 15:57 Arch1
-rw-r--r--   1 beca    user         0 Aug 21 15:57 Arch2


pulque 22% cd ..
pulque 23% ls
Directorio
pulque 24% rm -fr Directorio
```

EXPRESIONES REGULARES.

La sintaxis de las expresiones regulares utilizan un conjunto de caracteres los cuales tienen un significado especial para realizar búsquedas. A estos caracteres se les llama metacaracteres o wild card en ingles. Los principales metacaracteres utilizados en expresiones regulares son:

El circunflejo (^): Hace referecia al principio de la línea, esto es que coincidirá con todas las líneas que inicien con el patrón dado.

El signo de pesos ($): Hace referencia al final de la línea, es similar al circunflejo solo que ahora coincide con todas las líneas que acaben con el patrón dado.

El punto (.): Sustituye únicamente un carácter, es decir coincidirá con las cadenas a las cuales solo cambie el carácter que ocupe el lugar del punto en el patrón buscado

El asterisco (*): Sustituye cero o más ocurrencias del carácter anterior, coincidirá una cadena con el patrón buscado si existe una repetición del carácter anterior al asterisco. El asterisco se puede combinar con el punto para buscar cualquier secuencia de caracteres, incluyendo ninguno.

Los corchetes ([ ]): Sustituyen una serie de caracteres especificados dentro de los corchetes en el patrón de búsqueda, también se pueden especificar rangos por medio de un guión (-).

BUSQUEDAS DE CADENAS.

grep

Es un comando el cual se utiliza para la localización de cadenas en archivos de texto. Lo que hace grep es tomar las líneas que llegan de la entrada estándar, busca en ellas el patrón o patrones especificados e imprime las líneas que lo(s) contienen en la salida estándar.

***Su sintaxis es*:**

% **grep** [*opciones*] *patrón* [*archivo ...*]

El patrón de búsqueda debe de ponerse entre comillas en caso de que contenga espacios o caracteres con dignificado especial para el shell, como &,(,),*,etc. Es importante aclarar que grep trabaja por líneas, es decir, no encontrara un patrón dado si ese patrón comienza en una línea y termina en la siguiente.

Opciones:

-i Ignora la distinción entre mayúsculas y minúsculas.
-v Imprime las líneas que NO contienen el patrón dado.
-c Regresa el numero de líneas que contienen el patrón.
-n Imprime la línea donde se encuentra el patrón y el numero que ocupa en el archivo
-l Cuando se especifican múltiples archivos, regresa los nombres de los archivos que contienen el patrón.

Ejemplos:

% **grep** Casa lista // Muestra todas las líneas en el archivo lista que contengan la cadena "Casa".

% **grep** -v -i "^Hola" * // Muestra las líneas en todos los archivos del directorio actual donde no se encuentre una línea la cual empieza con la cadena Hola sin importar sí esta en mayúsculas o minúsculas.

% **grep** "ch.*se" *.dat // Imprime las línea de los archivos que terminen con .dat y que contendrán los caracteres ch, seguidos de cualquier numero de caracteres y terminando con se.

fgrep

Es un comando similar a grep con la diferencia de que fgrep permite buscar múltiples objetivos que se pueden especificar en la línea de comandos o en un archivo. A cambio de esto fgrep no permite expresiones regulares.

Sintaxis:

% **fgrep** [*opciones*] "*cadena1*
> *cadena2*
> ...
> *cadenaN*" [*archivo* ...]

Opciones:

Acepta todas las opciones mencionadas antes en el grep, incluyendo...

-f archivo // Especifica el archivo de donde se tomaran las cadenas de búsqueda.

egrep

Es un comando que incluye lo mejor de grep y de fgrep, y además incluye otras mejoras para la búsqueda, ya que añade algunos caracteres adicionales para realizar búsquedas extendidas e implementa la utilización de la barra vertical para separar patrones de búsqueda además de también utilizar la separación en líneas como en fgrep.

El signo más (+): Es similar al asterisco, con la diferencia de que sustituye una o más repeticiones del carácter anterior.

El signo de interrogación (?): Funciona también de manera similar al asterisco pero solo sustituye cero o una repetición del carácter anterior.

ORDENAMIENTO

sort

Este comando se utiliza cuando se quiere ordenar el contenido de un archivo de acuerdo a un criterio especifico (alfabético, numérico, ascendente, descendente y de acuerdo a un campo especifico) El resultado de este comando es desplegada en la salida estándar.

Sintaxis:

% **sort** [*opciones*] [*+pos1* [*pos2*]] [*archivo...*]

Si se especifican varios archivos el ordenamiento se hace en general y no archivo por archivo.

Opciones:

-b	// Ignora espacios y tabuladores antes de la cadena a ordenar.
-d	// Hace ordenamiento de "diccionario" solo considera letras números y espacios en blanco.
-f	// Toma como iguales minúsculas y mayúsculas.
-n	// Realiza un ordenamiento numérico.
-r	// Ordena en forma descendente.
-tx	// Especifica x como el separador de campos.
-o archivo	// Direcciona la salida al archivo especificado.
+pos1 [-pos2]	// Especifican el campo a utilizar para realizar el ordenamiento. Especifican el primer (pos1 toma valores de 0 hasta n) y ultimo carácter (pos2 valor del campo - 1) en el cual se va a basar el ordenamiento. ******** INVESTIGAR MAS PROFUNDAMENTE *********

Ejemplos:

% **sort** /etc/passwd // Ordena el archivo passwd con los parámetros default, es decir comenzando desde el primer carácter y haciendo un ordenamiento ascendente.

% **grep** "/bin/sh$" /etc/passwd | **sort** -nr -t: +2 -3 // Primero se extraen los usuarios que tienen el Bourne Shell como shell por default y se pasan al sort por medio de una tubería, el cual hace un ordenamiento en base al UID, esto es, el segundo campo. El ordenamiento es descendente.

cut

Nos permite seleccionar partes (campos) de un archivo en el sentido horizontal.

Sintaxis:

% **cut** -*flista* [-d*char*] [-s] [*archivo...*]

Opciones:

-lista // lista de enteros separados por comas (,), que indican campos, pudiendo también especificar rangos.
-flista // La lista indica los números de campos separados por cierto delimitador.
-dchar // Char es un carácter que se especifica como delimitador de campos.
-s // Indica que las líneas que no contengan el delimitador serán ignoradas.

Ejemplos:

% **cut** -d: -f1,5 /etc/passwd // Despliega las cuentas existentes en el sistema junto con el nombre real del usuario.

% **who am i** | **cut** -f1 -d" " // Imprime el nombre de la cuenta actual de trabajo.

SUSTITUCION DE CARACTERES

tr

La utilidad de este comando radica en que convierte ciertos caracteres en otros, esto es, toma la información que se le pasa de la entrada estándar (es la única forma de que este comando funcione ya que no puede leer de un archivo) y al arrojarla a la salida estándar, ejecuta las sustituciones indicadas.

Sintaxis:

% **tr** [*opciones*] [*cadena1* [*cadena2*]]

La sustitución se hace de la siguiente manera: los caracteres incluidos en la cadena1 serán sustituidos por sus caracteres correspondientes en cadena2.

Opciones:

-c // Reemplaza los caracteres NO contenidos en cadena1 por los que contiene cadena1

-d // Borra los caracteres que se encuentren en cadena1 (no requiere cadena2)

-s // Todos los caracteres que se encuentren repetidos y en forma contigua que estén en cadena2 se eliminaran desplegando a la salida un solo carácter.

Ejemplos:

% **tr** "[a-z]" "[A-Z]" < arch1 > arch2 // Cambia todos los caracteres en minúsculas del arch1 por su correspondiente en mayúsculas y los guarda en arch2

% **tr** -cs "[A-Z][a-z]" "[\012*]" < arch1 > arch2 // Cambia todos los caracteres que no sean letras en cambios de línea (código 12 octal), dejando en el arch2 el texto sin repetición de cambios de línea.

uniq

Este comando nos muestra las líneas repetidas en un archivo de texto. Para que el comando realice su labor se tiene que haber ordenado el archivo previamente.

Sintaxis:

% **uniq** [*opciones*] [*arch_entrada* [*arch_salida*]]

Opciones:

-u // Imprime las líneas no repetidas.
-d // Imprime una sola copia de las líneas repetidas (El modo normal de operación de uniq es la combinación de las opciones -u y -d)
-c // Imprime antes de cada línea las veces que se repite.
-n // Ignora los primeros n campos
+n // Ignora los primeros n caracteres

Ejemplo:

% **cut** -d: -f1-6 /etc/passwd |**sort** -t: +5 | **tr** " :" ".\011" |**uniq** -5 // Se elimina él ultimo campo del archivo passwd y se ordena en base al home de cada usuario, después se reemplazan los espacios por puntos y los dos puntos por tabuladores (código 11 octal), y por ultimo se muestran los usuarios que tengan un mismo home.

head

Este comando sirve para mostrar las primeras líneas de un archivo, las líneas que mostrara por default son 10, la única opción que tiene es -n , donde n son él numero de líneas que se desean desplegar. Por ejemplo:

% **head** -20 /etc/host // Nos muestra las primeras 20 líneas del archivo host.

tail

Funcione de manera similar al comando head, solo que nos muestra las ultimas líneas que se encuentran en un archivo.

Sintaxis:

% **tail** [+-[n] [*unidad*] [[-]f] [*archivo* ...]

Opciones:

-n	// Imprime las ultimas n líneas del archivo.
+n	// Imprime el archivo a partir de la línea n.
unidad	// Es la unidad en la que se va a realizar el conteo, por defaul son línea. Puede ser l para representar líneas, b para bloques o c para caracteres.
[-]f	// Realiza un monitoreo del crecimiento del archivo por otro proceso.

CALCULOS Y CONTEOS.

expr

Este comando nos sirve para evaluar expresiones aritméticas o de cadena desde un shell.

Sintaxis:

% **expr** *argumentos*

Donde los argumentos son evaluados y el resultado es escrito a la salida estándar. Los caracteres que tengan un significado especial para el shell tienen que estar precedidos del carácter de escape (\). Solo se pueden manejar números enteros o cadenas.

La lista de operadores validos es la siguiente, en orden de precedencia. los operadores que tienen la misma precedencia se listan entre llaves {}.

expr1 | expr2	// Regresa expr1 si su valor es diferente que cero o cadena nula, de lo contrario regresa expr2.
expr1 \& expr2	// Regresa expr1 si expr2 NO tiene un valor de cero o cadena nula, de lo contrario regresa 0.

expr1 {=,\>,\>=,\<,\<=,!=} expr2 // Regresa 1 si es verdadero y cero de lo contrario.

expr1 {+,-} expr2	// Regresa el resultado de la ecuación indicada.
expr1 {*,/,%} expr2	// Regresa el resultado de la ecuación indicada.

Ejemplos:

% a=`**expr** 3 + 4` // El valor de a es modificado por la ecuación, para ver dicho valor se escribe % echo $a y se pensiona ENTER.

% c=`**expr** $c + 1` // Incrementa el valor de la variable c.

wc

A pesar de que parece el significado de un lugar "privado", el nombre de este comando significa "word count", y nos sirve para contar palabras, líneas y caracteres en el flujo de entrada estándar o de un archivo.

Sintaxis:

% **wc** [*opciones*] [*archivo...*]

Si se especifican múltiples archivos, el comando mostrara la información de cada archivo por separado.

Opciones:

-c // Cuenta el numero de caracteres.
-l // Cuenta las líneas.
-w // Cuenta las palabras (conjunto de caracteres delimitados por espacios en blanco, tabuladores o saltos de línea).

find

Permite localizar archivos que cumplan con ciertas condiciones y actuar sobre ellos de diversas formas.

Sintaxis:

% **find** *rutas operadores*

Operadores de localización de Archivos.

-name nombre // Encuentra los archivos que coincidan con el nombre especificado. Si se utiliza metacaracteres es necesario utilizar las comillas

-perm permisos // Encuentra los archivos que tengan los permisos numéricos especificados.

-type x // Encuentra los archivos del tipo especificado por x: f = archivos ordinarios, d = directorios, l = para ligas simbólicas, etc.

-user usuario // Encuentra los archivos que pertenezcan al usuario especificado.

-group grupo // Encuentra los archivos del grupo especificado.

-size n // Encuentra los archivos del tamaño especificado. (n exactamente, +n mayor que n, -n menor que n).

-atime n // Archivos accedidos hace n días (también se utiliza +n y -n).

-mtime n // Archivos modificados hace n días (también se utiliza +n y -n).

-ctime n // Archivos creados hace n días (también se utiliza +n y -n).

-newer archivo // Encuentra los archivos modificados mas recientemente que el especificado.

OPERADORES DE ACCION.

-print // Imprime en la salida estándar la ruta de cada archivo encontrado.

-exec comando // Ejecuta el comando especificado cada vez que se encuentra un archivo. La cadena {} en comando será reemplazada por el nombre del archivo, siempre y cuando este rodeada por espacio en blanco. Comando debe terminar con \;.

-ok comando // Igual que -exec, pero preguntando antes de ejecutar cada comando.

OPERADORES RELACIONALES

op1 -a op2 // Encuentra los archivos que cumplan con los dos operadores. Este operador no es necesario, basta poner los operadores uno tras otro.

op11 -o op2 // Encuentra los archivos que cumplan con cualquiera de los dos operadores.

!operador // Encuentra los archivos que no cumplan con el operador.

\(expr \) // Permite agrupar operadores para que sean evaluados antes que el resto.

Ejemplos:

% **find** . -name "*.o" -exec **rm** -f {}\; // Borra todos los archivos con terminación .o que estén debajo del directorio actual.

% **find** /tmp -atime +15 -print -exec **rm** -f {} \; // Encuentra todos los archivos dentro de /tmp que hayan sido accedidos hace mas de 15 días (o sea, que no hayan sido accedidos en los últimos 15 días), imprime sus nombres y los borra.

% **find** .\! -type d -print //Imprime los nombres de todos los archivos que no sean directorios.

pulque% **find** $HOME \(-name a.out -o -name '*.o' \ -atime +7 \ -exec rm {} \;

Remueve todos los archivos del tu directorio home con el nombre a.out o con terminacion .o, que no han sido accesados en 7 dias.

pulque% **find** . -size +40 -xdev -print

Imprime todos los archivos de 40 bloques, en la particion del disco -xdev

pulque% **find** . ! perm -0100 -size +1 -type -print | xargs gzip -v

Imprime todos los archivos que no tengan los permisos 0100, que tenga un tama~n o en bloques de 1 (Esto depende de la londitud del bloque de sistema de archivos), lque no sean directorios y ejecuta el comando gzip -v.

Si el sistema no tiene xargs, la sintaxis seria asi:

pulque% **find** . ! -perm -0100 -size +1 -type f -exec gzip -v {}\;

pulque% **find** . -type f -print |xargs ls -l

Busca todos los archivos desde raiz y los lista en la pantalla.

pulque% **find** . -user root -perm -4000 -print

Imprime todos los archivos de user rootque contenga permisos -4000.

pulque% **find** . -group 10 -perm -2000 -print

Imprime los archivos con permisos -2000 del grupo 10.

SEGURIDAD DE ARCHIVOS.

TIPOS DE PERMISOS

Símbolo	Permiso	Significado
R	Lectura	Se puede abrir el archivo para poder ver su contenido, permite ver archivos y subdirectorios dentro del subdirectorio con este permiso.
W	Escritura	Se puede sobreescribir el archivo o modificar su contenido. Permite crear nuevos archivos o subdirectorios bajo el directorio
X	Ejecución	Permite la ejecución de un archivo, permite el acceso a un archivo o subdirectorio dentro del directorio solo si se sabe su nombre

CASOS ESPECIALES.

SUID y SGID.

En ocasiones es necesario que un usuario no privilegiado sea capaz de realizar tareas que requieren de privilegios, por ejemplo:

El programa passwd necesita modificar el archivo /etc/passwd para cambiar el passwd del usuario. Sin embargo los permisos de dicho archivo no permiten que un usuario normal pueda escribir o modificar la información existente, y solamente el dueño (root) tiene el permiso necesario para dicha tarea.

Para resolver esta clase de problemas se crearon los mecanismos de SUID (set-UID) y SGID (set-GID) los cuales permiten a un programa asumir la identidad de otro usuario o grupo al momento de ejecutarse.

Cuando se ejecuta un programa con SUID, su UID efectivo se convierte en el dueño del programa en vez que el usuario que lo ejecuta. El SGID funciona de manera similar, pero el GID efectivo del proceso será el del grupo del programa y no el del usuario que lo ejecuta. Un programa puede ser tanto SUID como SGID al mismo tiempo.

STICKY

Un programa que tengá el sticky bit no será borrado del área de swap del sistema cuando finalice su ejecución.

Cuando se utilizan estos casos especiales se identifican gracias a que el campo de ejecución en los permisos del archivo es modificado.

CAMBIO DE PERMISOS

El cambio en los permisos de un archivo o de un subdirectorio se realiza por medio del comando del sistema chmod(1). Solo se pueden cambiar los permisos de un archivo si se es el dueño o root.

% **chmod** *opciones modo archivo*

Opciones:

R // Cambia los permisos en forma recursiva, esto es, cambia los permisos del directorio o archivo especificado hacia abajo en un nivel jerárquico.
f // Elimina los mensajes de error que se presenten durante la ejecución del comando.

Existen dos formas de cambiar los permisos de los archivos y directorios, esto es cuando el modo es simbólico y cuando es absoluto.

MODO SIMBOLICO.

El modo simbólico de los permisos tiene ls siguiente forma:

% **chmod** [ugoa] [-+] *permisos*

u Pone los permisos del usuario.
g Pone los permisos del grupo.
o Pone los permisos para los otros o para los demás.
a Se refiere todas las anteriores, este es el valor por omisión.

\- Desactiva permisos.
\+ Activa permisos.
= Activa los permisos especificados y elimina los demás.

Permisos:

r Permiso de lectura.
w Permiso de escritura.
x Permiso de ejecución.
s Permiso de setuid o setgid.
t Permiso de sticky.

MODO ABSOLUTO.

El modo absoluto también es conocido como modo octal, esto es porque los permisos son representados por medio de números octales.

___	000	0	No se tiene ninguna clase de permisos.
__x	001	1	Solo se tiene el permiso de ejecución.
_w_	010	2	Solo se tiene el permiso de escritura.
_wx	011	3	Se tienen permisos de escritura y de ejecución.
R__	100	4	Solo se tiene el permiso de lectura.
R_x	101	5	Se tienen permisos de lectura y de ejecución.
Rw_	110	6	Se tienen permisos de lectura y de escritura.
Rwx	111	7	Se cuenta con todos los permisos.

4000	Permiso de setuid.
2000	Permiso de setgid.
1000	Permiso de sticky.

Ejemplos:

% **chmod** g-w arch1 // Quita el permiso de escritura del grupo en el archivo arch1
% **chmod** a+rwx arch1 // Otorga los permisos de lectura, escritura y ejecución para todos.
% **chmod** a=r arch1 // Concede permiso de lectura a todos y elimina los demás.
% **chmod** 752 arch1 // Coloca los siguientes permisos rwxr_x_w_ al archivo arch1
% **chmod** 4000 arch1 // Se crea una versión de SUID del archivo ejecutable arch1

EL FILTRO UMASK

El umask (user file-creation mode mask) es un numero octal de tres dígitos que UNIX usa para determinar los permisos que tendrá un archivo recién creado. Para asignarle un valor al comando umask se requiere obtener el complemento octal de los permisos en modo numérico deseados. Por ejemplo, para obtener el modo 754 por default, reste 777 - 754 = 023, y este valor será el que se utilice con el comando umask.

% **umask** 023

Cuando esta línea se ejecuta, todos los archivos que se creen a continuación tendrán estos permisos automáticamente, Usualmente se pone al comando umask en el archivo de de inicialización del login.

LIGAS

El comando ln (Link)

Descripción

El comando ln permite agregar uno o más nombres a un archivo existente, proporcionando un sistema de referencia cruzada. Cada nombre localizado en tu directorio tiene la misma prioridad para leer o escribir del/en el archivo. Sin embargo, si el archivo se liga a otro usuario, generalmente solo el creador del archivo puede escribir en él.

El comando ln puede usarse para crear tanto ligas duras como simbólicas. Una liga dura es un apuntador a un archivo y es indistinguible de la entrada de directorio (liga) original. Los cambios a un archivo son independientes del nombre utilizado para referirse al archivo. Las ligas duras no pueden extenderse a sistemas de archivos ni tampoco pueden referirse a directorios. El comando ln crea ligas duras por omisión (por default).

Una liga simbólica es un apuntador indirecto a un archivo; su entrada de directorio contiene el nombre del archivo al cual esta enlazado. Las ligas simbólicas si pueden extenderse a sistemas de archivos y también pueden referirse a directorios.

En el sistema Unix, el nombre que eliges para un archivo actúa como un enlace entre ese archivo y tu sistema de directorios. Estas ligas enlazan los archivos a los directorios.

El comando ln establece un nuevo enlace entre el archivo y un directorio. Todos los enlaces o nombres anteriores siguen existiendo, pero no se crean nuevos archivos. Ligar un archivo a diferentes directorios hace que parezca como si existieran varios archivos diferentes, pero de hecho solo hay un archivo con múltiples nombres. Esto es muy útil cuando deseas que un archivo en particular pueda ser accedido desde varios directorios sin teclear toda su ruta.

Sintaxis:

La forma:

% **ln** nombre_archivo1 nombre_archivo2

Convierte a nombre_archivo2 en una liga (o alias) para nombre_archivo1

La forma:

% **ln** nombre_archivo(s) nombre_directorio

Te permite colocar nuevos nombres de archivo o ligas en otros directorios.

Opciones:

-n si nombre_archivo2 es el nombre de un archivo existente, no se sobreescribe el contenido del archivo.

-s Crea una liga simbolica que te permite enlazar a traves de sistemas de archivo.

Aplicación:

Una aplicación para este comando seria cuando se desea establecer varios subdirectorios y que todos accedan a la misma lista de correo. Por supuesto, puede crearse una copia de la lista de correo en cada subdirectorio, pero eso utilizaría mucho espacio en disco. Usando el comando ln es posible ligar el archivo que contiene la lista a un nombre en cada subdirectorio , y así el archivo existirá en cada subdirectorio. Una ventaja adicional es que si se actualiza el archivo desde uno de los subdirectorios, se actualizara también en los demás. Una limitante del comando ln es que no es posible usarlo para
asignar dos nombres al mismo directorio.

Ejemplo 1 :

Supongamos que el usuario Cesar se encuentra en su directorio home (/opt/home/cvc) y teclea el siguiente comando:

% ln carta letter

Unix checa si ya existe el nombre letter. De no ser asi, entonces da el nuevo nombre (letter) al archivo, manteniendo el nombre anterior (carta). El comando % ls -l mostraría que ambos archivos tienen dos ligas o enlaces. Cesar entonces llamaría al archivo por cualquiera de los dos nombres.

Si ya existe un archivo llamado letter en el directorio home, entonces ln borra primero el nombre anterior antes de hacer la liga, a menos que se especifique lo contrario con la opción -n.

Ejemplo 2 :

Siguiendo con el ejemplo anterior, si letter ya existe como subdirectorio, supongamos ahora que Cesar tiene un subdirectorio llamado correo. El comando

% ln carta correo

agrega el nombre carta al subdirectorio correo, pero mantiene intacto el nombre original, es decir, el mismo archivo esta ahora en dos directorios, y tiene las rutas:

/opt/home/cvc/carta
y
/opt/home/cvc/correo/carta

No se trata de copias, sino del mismo archivo. Cuando Cesar se encuentra en su directorio home y quiere ver el contenido del archivo carta, teclea:

% cat carta

Que como ya vimos es una abreviatura para % cat /opt/home/cvc/carta

Si cesar entra al directorio correo, puede ver el contenido del archivo tecleando:

% cat carta

Que es una abreviatura para % cat /opt/home/cvc/correo/carta

Sin el comando ln, Cesar tendria que teclear:

% cat % cat /opt/home/carta

para ver el contenido del archivo carta desde el subdirectorio correo.

Supongamos que Cesar, estando en su directorio home, ahora teclea el comando:

% rm carta

El archivo carta desaparece de su directorio home, pero el archivo todavía existe en el subdirectorio correo. Solamente cuando todos los nombres o ligas al archivo se han borrado sera eliminado el archivo mismo.

COMUNICACION ENTRE USUARIOS

write Manda mensajes a otros usuarios de la misma maquina.

Sintaxis:

% **write** usuario
.....
.....
Ctrl-D

Ejemplo:

lidy@aketzali:~/Cursos/Unix_Admon> **write humcgr**
hola esta es una prueba
regresame el write, ok?
contestame
:)
bye!
lidy@aketzali:~/Cursos/Unix_Admon>
Message from humcgr@aketzali.super.unam.mx on ttyp8 at 14:48 ...
Hola!!!
EOF

talk Conversar con otro usuario

Sintaxis:

% **talk** user [tty_name]

Ejemplo:

Cuando te llaman:

Message from Talk_Daemon@aketzali.super.unam.mx at 14:55 ...
talk: connection requested by humcgr@aketzali.super.unam.mx.
talk: respond with: talk humcgr@aketzali.super.unam.mx

Para responder o para tu llamar tambien:
ds5000.super.unam.mx> **talk humcgr@aketzali.super.unam.mx**

Cuando se establece la coneccion y ya "hablaron"
para terminarla es con Ctrl-C.

MAIL — Mandar correo electronico.

ds5000.super.unam.mx> **Mail -vs** lidy@conga.super.unam.mx

Subject: prue

prueba... .
.
EOT
Warning: alias database out of date
lidy@conga.super.unam.mx... Connecting to conga.super.unam.mx (smtp)...
220 conga.dgsca.unam.mx Sendmail NX5.67e/NX3.0M ready at Thu, 20 Aug 98 15:18:12 -0600
>>> HELO ds5000.super.unam.mx
250 conga.dgsca.unam.mx Hello ds5000.super.unam.mx, pleased to meet you
>>> MAIL From:<lidy@asc.unam.mx>
250 <lidy@asc.unam.mx>... Sender ok
>>> RCPT To:<lidy@conga.super.unam.mx>
250 <lidy@conga.super.unam.mx>... Recipient ok
>>> DATA
354 Enter mail, end with "." on a line by itself
>>> .
250 Ok
>>> QUIT
221 conga.dgsca.unam.mx closing connection
lidy@conga.super.unam.mx... Sent
ds5000.super.unam.mx>

Otra forma:

ds5000.super.unam.mx> mail lidy@conga.super.unam.mx
holas!
ds5000.super.unam.mx>

Otra forma:

ds5000.super.unam.mx> mail
& help

Otra forma:

Para mandar un archivo por mail:

ds5000.super.unam.mx> mail lidy@conga.super.unam.mx < archivo

PROCESOS

Un **proceso** es un programa en ejecución, por lo que el shell de UNIX es considerado como un proceso (el prompt que aparece en pantalla indica que el shell está ejecutándose en ese momento, lo cual puede comprobarse con el comando *ps*).

Procesos en background y en foreground

Un proceso puede ejecutarse en una de dos formas:

1. **Primer plano** (*foreground*). Se dice que un proceso está corriendo en primer plano cuando es totalmente visible para el usuario y cuando el usuario interactúa con él. Cuando se ejecuta un proceso en primer plano dentro de un shell, el usuario debe esperar a que dicho proceso termine antes de ejecutar otro.

2. **Segundo plano** (*background*). Los procesos que se ejecutan en segundo plano no se despliegan en pantalla, por lo que es común que el usuario no observe qué se está ejecutando; mientras se desarrolla un proceso en segundo plano el usuario puede ejecutar otro proceso, el cuál se ejecuta al mismo tiempo que el proceso que está en background.

Para ejecutar un proceso en segundo plano basta con añadir el símbolo **&** al final de la línea de comandos y pulsar <ENTER>. El sistema devuelve entonces el **PID** (*Process IDentifier*) del proceso que va a ejecutarse en background, que es un número que distingue de forma unívoca a un proceso de otro.

Ejemplo:

% find / -name "core" -exec rm –f { } \; &

Este comando busca a partir del directorio raíz todos los archivos cuyo nombre es core (estos archivos se generan cuando se produce un error de asignación de memoria o algo por el estilo y generalmente tienen un enorme tamaño, del orden de varios MB) y los borra, sin que el usuario haga nada más que ejecutar el comando

Eliminación de procesos

Para eliminar un proceso se utiliza la combinación de comandos *ps* y *kill*. El primero despliega un listado de los procesos en ejecución (útil para ver el PID del proceso que deseamos eliminar), en tanto el segundo elimina el proceso indicado.

Por ejemplo, supongamos que no recordamos cual era el PID devuelto al iniciar el proceso generado por el siguiente comando:

% tar cvf /tmp/respaldo_etc_980820.tar ./etc &
Con el siguiente comando es posible ver el PID del proceso que deseamos eliminar:

% ps –fea | grep cvc (asumiendo que el dueño del proceso es cvc)

Una vez que conocemos el PID del proceso que deseamos "matar", basta con teclear el siguiente comando:

% kill –9 1527 (asumiendo que el PID del proceso era 1527)

La opción *–9* del comando kill mata un proceso incondicionalmente, para lo cuál es necesario ser el dueño del proceso o el superusuario (root).

INTRODUCCIÓN A LA PROGRAMACIÓN EN SHELL

El shell de UNIX es al mismo tiempo un lenguaje de control, un lenguaje de programación y un intérprete de comandos.

El shell actúa como intermediario entre el kernel del sistema operativo y el usuario, invocandolo e interactuando con él para permitir que los usuarios ejecuten comandos y otras aplicaciones.

A cada usuario se le asigna un shell por omisión, el cual se inicializa cada vez que el usuario entra a sesión o bien, cuando abre una nueva ventana dentro de un ambiente gráfico. El shell interpreta los comandos dados por el usuario, los cuales pueden leídos a su vez desde el prompt del sistema o desde un archivo.

Un **script** es un archivo que contiene comandos del shell. Los scripts son *interpretados*, no compilados, esto es, son leídos y ejecutados secuencialmente una línea a la vez (un programa compilado es leído completamente y convertido a código objeto, el cuál a su vez es enlazado para generar un programa ejecutable).

A grandes rasgos, podemos decir que el shell tiene las siguientes funciones:

- **Uso interactivo** (el sistema ejecuta comandos tecleados desde el prompt).
- **Personalización de una sesión de trabajo** (el ambiente de trabajo, como la definición del directorio de trabajo del usuario, la configuración de la terminal, etc., es controlado por variables definidas por el shell).
- **Programación** (permite la creación y ejecución de scripts).

Algunas de las **aplicaciones** de los scripts en shell son:

- *Instalación de software*
- *Procedimientos de respaldo*
- *Mejora de algunos comandos*
- *Personalización del ambiente de trabajo*

Tradicionalmente UNIX maneja 3 shells: *Bourne shell*, *Korn shell* y *C shell*. Cada shell tiene su propio lenguaje de programación de alto nivel utilizado para ejecutar secuencias de comandos, seleccionar entre operaciones alternas, realizar operaciones lógicas y acciones repetitivas. El Bourne shell y el Korn shell tienen sintaxis similares, mientras que el C Shell tiene una sintaxis similar a la del lenguaje C y ofrece otras capacidades.

Tabla comparativa de los shells de UNIX

Shell	Autor	Características	Comando	Prompt
Bourn e Shell	Steve Bourne	• Shell original del UNIX de AT&T • Es el shell estándar de UNIX • No permite historia de comandos • Permite la redirección de Entrada/Salida • Permite el uso de metacaracteres (comodines) para la abreviación de los nombres de archivos • Permite al usuario definir sus propias variables • Posee un conjunto de comandos propios para la programación • Usado por todos los scripts de administración del sistema	/bin/sh	$
Korn Shell	David Korn	• Compatible con Bourne Shell • Incluye operaciones aritméticas, arreglos y manipulación de cadenas • Permite historia de comandos (acceso a comandos ya tecleados) • Usa un mayor número de metacaracteres • Permite utilizar arreglos y expresiones aritméticas • Permite usar alias (abreviación de comandos)	/bin/ksh	$
C Shell	Bill Joy	• Diseñado en la Universidad de California, en Berkeley • Redireccionamiento de E/S • Incorpora alias e historia de comandos • Sintaxis similar a la del lenguaje C	/bin/csh	%

El proceso de login

Cuando un usuario entra a sesión se ejecutan los siguientes pasos:

1. Validación del login y entonces del password
2. Inicialización del UID y el GID del usuario
3. Establecimiento del directorio de trabajo (directorio home)
4. Inicio del shell asignado por default (en el archivo /etc/passwd)
5. Ejecución del archivo /etc/profile
6. Ejecución del programa /$HOME/.profile (Bourne shell ó Korn shell) ó /$HOME/.cshrc (C shell) ($HOME = directorio home del usuario)
7. Ejecución del programa /$HOME/.login
8. Aparición del prompt en pantalla

Los archivos .profile y .cshrc contienen comandos del sistema y variables especiales del shell, cuyos valores determinan el ambiente de trabajo del usuario.

Un ejemplo del contenido del archivo **/etc/.profile** sería:

```
PATH=.:/bin:/usr/sbin:/etc
TERM=vt100
PS1="tolsa% "
export TERM PATH
clear
```

Un ejemplo del contenido del archivo **/etc/.cshrc** sería:

```
set path=(/bin /usr/sbin /etc)
setenv TERM vt100
set prompt="tolsa % "
set history=40
clear
```

Variables

Una **variable** es un nombre que se refiere a un área de almacenamiento temporal en la memoria del sistema. La programación en shell utiliza dos tipos de variables:

- *Variables del shell*
- *Variables de ambiente*

La diferencia es que las variables del shell son como variables locales, es decir sólo son conocidas por el shell que las genera, en tanto las variables de ambiente son como variables globales, que pueden ser heredadas a procesos hijos, es decir, son reconocidas por cualquier programa o script.

Variables en Bourne y Korn shell

Variables del shell

Las siguientes variables son declaradas automáticamente por el shell cada vez que se inicia una nueva sesión. Generalmente estas variables son utilizadas por programas del usuario y scripts.

Variable	Significado
$#	Número de parámetros pasados desde la línea de comandos
$!	PID del último proceso ejecutado en background
$0	Primer parámetro pasado desde la línea de comandos
$*n*	*n-ésimo* parámetro pasado desde la línea de comandos
$*	Contiene todos los parámetros pasados desde la línea de comandos
$$	PID del proceso actual

Otras variables del shell

Las variables que se muestran a continuación no son declaradas automáticamente por el shell. Generalmente éstas son iniciadas en el archivo .profile y pueden usarse cuando así se requiera.

Variable	Significado
HOME=*dir*	Guarda el directorio home del usuario
IFS=*caracteres*	Separadores de campos internos (espacio, tabuladores, salto de línea, etc.)
MAIL=*archivo*	Archivo en dónde el usuario recibe su correo
MAILCHECK=*n*	Frecuencia (en segundos) con que el shell verifica si hay mensajes de correo para el usuario
PATH=*dirs*	Almacena uno o más directorios, separados por :, en donde se buscará un programa cuando este se ejecute
PS1=*cadena*	Almacena el prompt primario (generalmente, $)
PS2=*cadena*	Almacena el prompt secundario (generalmente, <)
SHELL=*archivo*	Nombre del shell utilizado
TERM=*cadena*	Tipo de terminal que se está utilizando

Variables definidas por el usuario

UNIX permite al usuario definir sus propias variables, lo cuál es útil cuando se crean programas o scripts, ya que es posible definirlas, modificar su valor, borrarlas, etc. El nombre de las variables no puede comenzar con un número, pero sí con valores alfabéticos o *underscore* (_).

Para crear una variable o asignarle un nuevo valor a una ya creada se usa la siguiente sintaxis:

nombre_variable=valor ; **export** *nombre_variable* (Bourne shell)
export *nombre_variable=valor* (Korn shell)

Para desplegar el valor de una variable se usa el comando echo:

% echo *nombre_variable*

Ejemplo (Bourne shell):

```
$ TERM=vt100
$ export TERM
$ echo TERM
TERM
$ echo $TERM
vt100
$
```

Variables en C shell

Archivos especiales

El C shell cuenta con los siguientes archivos especiales:

.login	Ejecutado antes del .cshrc
.cshrc	Ejecutado cada vez que el usuario ejecuta el comando /bin/csh
.history	Historia de comandos del login anterior
.logout	Ejecutado cuando el usuario sale de sesión

Variables del shell

Para asignar un valor a una variable del shell se usa la siguiente sintaxis:

% set *nombre_variable=valor*

Ejemplo:

% set prompt = "Tus deseos son ordenes: "

Para ver las variables del shell y su valor se usa el comando set:

% set
prompt Tus deseos son ordenes:

Variables de ambiente

Para asignar un valor a una variable de ambiente se usa la siguiente sintaxis:

% **setenv** *nombre_variable valor*

Ejemplo:

% setenv *DISPLAY* 132.248.120.21:0.0

Para ver las variables de ambiente y su valor actual se utiliza el comando setenv:

% setenv
TERM = DISPLAY 132.248.120.21:0.0

Por convención, los nombres de las variables de ambiente se declaran con mayúsculas. Algunas variables de ambiente de C shell son:

Variable	Significado
USER	Nombre del usuario
DISPLAY	Host y puerto en el que van a desplegarse aplicaciones desde sesiones remotas
PWD	Directorio actual de trabajo

Programación en shell

El Korn shell es una extensión totalmente compatible de Bourne, es decir, tiene algunas mejoras y características diferentes, lo cual implica que todo script escrito en Bourne shell correrá en Korn shell, pero no necesariamente al revés. Es por eso que el siguiente script en Bourne shell puede también ser ejecutado en Korn shell, simplemente cambiando la primera línea por la línea **#!/bin/ksh.**

```
#!/bin/sh

clear
echo "Nombre del programa: $0"
echo ""
echo "Argumentos mostrados directamente: $*"
echo ""
echo "Argumentos mostrados mediante un for:"
echo ""
for arg in $*
do
        echo $arg
done
echo ""
echo  "Total de argumentos: $#"
echo ""
echo "Como te llamas? "
read nombre
n_arch=`ls –a | wc –w`
dir=`pwd`
echo ""
echo "$nombre, si no uso variables tienes `ls –a | wc –w` archivos en el directorio `pwd`"
echo ""
echo "$nombre, si uso variables tienes $n_arch archivos en el directorio $dir"
echo ""
if [ $n_arch –lt 5 ]
then
        echo "De hecho, tienes menos de 5 archivos"
else
        if [ $n_arch -eq 10 ]
then
                echo "Te das cuenta que tienes exactamente 10 archivos?"
else
                echo "Sabes, tienes entre 5 y 9 archivos o mas de 10"
        fi
fi
echo""
```

La versión de este script en C shell sería:

```
#!/bin/csh

clear
echo "Nombre del programa: $0"
echo ""
echo "Argumentos mostrados directamente: $*"
echo ""
echo "Argumentos mostrados mediante un for:"
echo ""
foreach arg ($*)
        echo $arg
end
echo ""
echo  "Total de argumentos: $argv"
echo ""
echo "Como te llamas? "
set nombre=$<
set n_arch=`ls –a | wc –w`
set dir=`pwd`
echo ""
echo "$nombre, si no uso variables tienes `ls –a | wc –w` archivos en el directorio `pwd`"
echo ""
echo "$nombre, si uso variables tienes $n_arch archivos en el directorio $dir"
echo ""
if ( $n_arch < 5 ) then
        echo "De hecho, tienes menos de 5 archivos"
else if ( $n_arch == 10 ) then
                echo "Te das cuenta que tienes exactamente 10 archivos?"
else
                echo "Sabes, tienes más de 10 archivos"
endif
echo""
```

Lo que este script hace es lo siguiente:

1. Muestra en la salida estándar (la terminal) el nombre del programa.
2. Imprime en la pantalla los argumentos recibidos desde la línea de comandos (primero usando variables y luego usando un ciclo).
3. Muestra el número total de argumentos.
4. Lee un dato (nombre) desde la entrada estándar (el teclado).
5. Asigna valores a dos variables, una que contiene el número de archivos del directorio actual (n_arch) y otra que contiene el nombre del directorio actual (dir).
6. Muestra el número total de archivos (incluyendo los ocultos) que contiene el directorio actual de trabajo (primero sin usar las variables creadas y luego usándolas).
7. Verifica si dicho número es menor a 5, en cuyo caso envía un mensaje haciéndoselo saber al usuario. Si no es ese el caso, entonces verifica si el número de archivos es igual a 10; si es así,

entonces envía otro mensaje; si no es ni el primer ni el segundo caso, entonces se envía un mensaje diferente.

Como puede verse, existen ciertas diferencias en cuanto a la sintaxis de Bourne y C shell, las cuales deben considerarse cuando se escriben scripts, para elegir el lenguaje adecuado.

ADMINISTRACIÓN UNIX

INTRODUCCIÓN

REQUISITOS GENERALES DEL ADMINISTRADOR

- Planear las actividades

  "No reiniciar la maquina ni hacerle mantenimiento sin avisarle a los usuarios"

- "Siempre" tener copias de seguridad

  Respaldar archivos tales como /etc/group antes de modificarlos. Una practica común es copiar como /etc/group.240898 para referencia.

- Conocer las utilerias básicas

Se recomienda saber programar en Bourne Shell, de preferencia así como también saber leer makefiles. Además de las utilerias indispensables:

man
find
at
cron (Automatiza tareas)

- Conocer la documentación

man (comando)
manuales impresos

SGI : Owner´s Guide (HW)
System Administration
Software Installation, etc.

Manuales gráficos

SGI: insight

libros
README
INSTALL

- Conocer el hardware de la maquina
- Establecer políticas de uso y administración

motd
issue
/tmp
horarios de mantenimiento

man

Sintaxis:

%**man** comando

ORGANIZACIÓN BASICA DE LOS MANUALES

AT & T	BSD	
1	1	Comandos a nivel de usuario y aplicaciones.
2	2	Llamadas al sistema y código de error del kernel.
3	3	Llamadas a las librerías
4	5	Formatos de los archivos estándar
5	7	Archivos y documentos
6	6	Juegos y demos
7	4	Dispositivos, drivers y protocolos de red
-	8	Comandos para la administración del sistema

CONOCER EL HARDWARE DE LA MAQUINA

* **Marca** (Sun, SGI, PC, etc.)
* **Características Físicas**

  CPU
  Memoria
  HD

* **Dispositivos Periféricos** (Unidad de cinta, CD-ROM, floppie, etc)
* **Capacidad de despliegue**
* **Encendido/Apagado.** Primero se encienden todos los dispositivos, luego el CPU.
  Para apagar primero el CPU y luego los dispositivos.

* **P ara cambiar un dispositivo, primero se apaga la maquina**
* **Conocer la ubicación física del equipo a administrar** (y tener acceso a él)

ESTABLECER LAS POLITICAS DE USO DEL SISTEMA

/etc/issue Este archivo contiene las políticas.

/etc/motd **motd** significa message of the day. Mensajes de bienvenida (procurar que no sea muy largo.

POLITICAS A ESTABLECER

- Horario de mantenimiento. Disponer de la maquina sin tener problemas con los usuarios.
- Responsabilidad de los respaldos
- Cuotas de disco
- Seguridad
- Mantener canales de comunicación con los usuarios a través de:

/etc/issue
/etc/motd
mail
news /usr/news/____

____
mensajes (políticas, nuevas instalaciones)

TAREAS DEL ADMINISTRADOR (root)

La cuenta de root es la cuenta de superusuario, root es dueño de todo el sistema de archivos y tiene acceso a todos los archivos. Con esta cuenta se llevan a cabo las tareas de administración del sistema.

- **Administrar usuarios.** Dar de alta y dar de baja a los usuarios, así como la modificación de las capacidades y privilegios de los mismos.
- **Configuración de dispositivos**. Hacer disponibles y compartir dispositivos, como impresoras, terminales, módem, unidades de cinta, etc.
- **Hacer respaldos**. Esto implica hacer copias de los archivos y guardar los respaldos para una posible restauración si es que los archivos del sistema se pierden o se dañan. Esto también incluye la determinación de los momentos de ejecución de los respaldos.
- **Detección del sistema.** La detención del sistema debe hacerse en una forma ordenada para evitar
  inconsistencias en el sistema operativo.
- **Capacitar usuarios**. Proporcionar u obtener buena capacitación para los usuarios, a fin de que puedan utilizar el sistema de manera efectiva y eficiente.
- **Asegurar el sistema**. Proporcionar un ambiente seguro. Los usuarios necesitan ser protegidos de interferencias entre ellos causadas por acciones accidentales.
- **Registrar los cambios del sistema**. Deberá mantener un libro de registro para anotar cualquier actividad significativa con relación al sistema.
- **Asesorar a los usuarios**. Actuar como "experto local" para ayudar a los usuarios ordinarios del sistema.

EL NÚCLEO (KERNEL) Y EL INTERPRETE DE COMANDOS

KERNEL

Propiamente es el sistema operativo y permanece siempre en memoria.

FUNCIONES GENERALES

* Control de procesos (asigna recursos, programa procesos y atiende requerimientos de servicios)

* Control de dispositivos (supervisa la transferencia de datos entre la memoria principal y dispositivos periféricos)

KERNEL

- Permanece en memoria (/vmunix, /unix)
- Crear y manejar procesos, swap, scheduler, comunicación entre procesos.
- Asignación de tiempos de atención.
- Manipulación de dispositivos de entrada-salida [I/O] (drivers)
- Crear y manejar los sistemas de archivos (permisos, ids, i-nodos, superbloque)
- Manejo de memoria

El kernel puede sèr configurado y adaptarse a los requerimientos particulares del hardware.

INTERPRETE DE COMANDOS (shell)

Es un lenguaje de control, interprete y lenguaje de programación, tiene las características que lo hacen sumamente flexibles para las tareas de un centro de computo e incluye:

* Características de control normales (secuenciación, iteración condicional, selección y otras mas).
* Paso de parámetros.
* Comunicación entre ordenes de Shell.
* Shell permite modificar en forma dinámica las características con que se ejecutan los programas en Unix: la salida estándar puede direccionarse a un archivo, a un proceso o a un dispositivo, asimismo, es posible interconectar entre sí, y a la vez usuarios pueden ver versiones "distintas" de Sistema Operativo" debido a la capacidad del Shell para configurar diversos ambientes de trabajo.

El shell es el programa que toma las peticiones del usuario (como comandos) y son interpretados para que el sistema operativo actúe y realice la función que el usuario ha pedido. Este programa es la llave para coordinar y realizar algunas de las tareas de Unix.

El shell es el intermediario entre el Kernel del sistema operativo y el usuario, esta labor la hace de forma transparente. Generalmente a este programa se le conoce como interprete de comandos, ya que precisamente es esa una de sus funciones.

Además el shell puede considerarse como un lenguaje de alto nivel el cual es estructurado, capaz de obtener información de tareas, procesos y usuarios, además de interactuar con ellos.

LA ESTRUCTURA DEL SISTEMA DE ARCHIVOS

Su estructura básica es bastante convencional, contiene:

SISTEMA DE ARCHIVOS

Jerárquico, estructura de árbol. (los archivos no están en un nivel, sino en varios)
Rutas absolutas y relativas. (Cualquier archivo puede ser accedido por su nombre o por su nombre-trayectoria [path-name], que especifica su posición absoluta en la jerarquía y donde los usuarios pueden cambiar su directorio actual a cualquier posición.)
Concepto de dueño de archivo, permisos, grupos.
Los dispositivos se manejan como archivos.
Diferentes tipos de archivos (ordinarios, directorios, especiales)

A pesar que el usuario tiene una imagen sencilla del sistema de archivos, las tablas que se manejan son muy complejas. Desde el punto de vista de Unix, divide cada partición del disco en cuatro regiones autoidentificadas:

Bootstrapping Dirección 0 no es usado por el sistema. Se deja para el procedimiento de bootstrap.

Area de Superbloque Almacena toda la información de las direcciones del disco.

Tabla de I-nodos Esta área es una lista de definiciones de archivo llamada tabla de I-nodos. Cada nodo esta numerado de tal forma que la combinación del nombre del dispositivo y su numero en esta lista sirven para identificar en forma única, un archivo en particular.

Area de datos. Esta área se usa para almacenar el contenido de los archivos.

boot block	Super bloque	Lista de inodos	bloques de datos

Antes de que cualquier partición del disco pueda ser utilizada, un filesystem se debe construir en ella.
Cada uno de los Filesystems ocupa un solo dispositivo, accedido con un nombre de archivo específico, sin importar si los datos están salvados realmente como una parte o todo en el disco físico, como partes de varios discos físicos, o como la agregación de múltiples discos físicos.

Cuando se hace un filesystem, ciertas estructuras de datos se escriben al disco que será utilizado para tener acceso y para organizar el espacio físico del disco con los archivos.

La más importante de estas estructuras es el Superbloque. El Superbloque es una tabla que contiene información importante acerca de los Filesystems.

Superbloque:

- Tamaño del FS
- Numero de Bloques libres
- Lista de bloques libres disponibles
- Indice del siguiente bloque libre
- Tamaño de la lista de nodos
- Numero de nodos libres
- Lista de nodos libres
- Indice del siguiente inodo libre
- Bandera de modificación (se prende si el Superbloque ha sido Modificado)

Hay un Superbloque para cada sistema de archivos. Hay respaldos del Superbloque a lo largo de un sistema de archivos, al fsck se le puede indicar que lea el Superbloque desde otra parte del disco.

Inodo: (es una estructura de datos)

- Identificador de dueño y grupo
- Tipo de archivo
- Permisos de acceso
- Fechas de acceso, modificación y modificación del inodo
- Numero de ligas del archivo.
- Direcciones del disco
- Tamaño del archivo

Ejemplos:

ls -lu	Acceso
ls -l	Modificación
ls -lc	Fecha de modificación del inodo

TIPOS DE SISTEMAS DE ARCHIVOS

Uso	AIX	Digital UNIX	HP-UX 10	SCO UNIX	SunOS	Solaris	IRIX	Linux
local	ifs	ufs	hfs	EAFS	4.2	ufs	efs	ext2
NFS	nfs	nfs	nfs	nfs	NFS	nfs	nfs	nfs
CD-ROM	cdrfs	cdfs	cdfs	HS	hsfs	hsfs	iso9660	iso9660
swap		ufs	swap[fs]		swap	swap	swap	swap
DOS				DOS	pcfs		dos	msdos
/proc		procfs				proc	proc	proc
other		advfs	uxfs	S51K		s5	hfs(Mac)	sysv
			lofs				xfs	minix

PROCEDIMIENTOS DE OPERACIÓN DE BOOT Y BAJA DEL SISTEMA

ALTA DEL SISTEMA

- Cargar e inicializar el Kernel
- Detectar dispositivos y su configuración
- Creación de procesos espontáneos
- Intervención del operador (solo modo single-user)
- Ejecución de scripts y demonios
- Operación multiusuario (si no hay intervención del operador)

BOOT DEL PROM

Bootstrapping: Se le llama así al proceso de dar de alta una estación de trabajo.
Booting: inicialización.

Encendiendo la maquina:

Se tienen que encender primero los dispositivos periféricos y al final el CPU.

Que es lo que pasa en ese momento?

Cuando se enciende la maquina, toma de la memoria ROM (memoria no volátil) los pasos para poder inicializarse.

Pasos:

1. Busca el sistema de booteo (bloque cero)
2. Despierta el Kernel el cual interactua con la ROM para formar sus tablas.

El kernel se puede encontrar en:

/umunix	En el caso de BSD
/unix	En el caso de SysV
/kernel/unix	

3. Corre el proceso init (tiene el Pid 1), se considera el padre de todos los procesos, de aquí se empiezan a correr todos los procesos necesarios para levantar la maquina.

Init se auxilia de ciertos archivos de configuración, que están bajo /etc o /sbin

En el caso de BSD estos archivos son:

rclocal, rc

En el caso de SysV:

rcn	donde n es un numero, que varia dependiendo el nivel.

FUNCIONES DE LOS SCRIPTS DE INICIALIZACION

- Verificar la integridad de los sistemas de archivos en el sistema
- Montar los sistemas de archivos
- Labores de limpieza
  - . Cuotas
  - . borrar archivos temporales
- Despertar a los demonios (Impresión, mail, cron, etc)

CAMBIANDO EL DISPOSITIVO DE BOOT

Se puede inicializar desde:

- disco duro
- red
- cinta
- cdrom

NIVELES DE INICIALIZACIÓN

Existen 3 niveles en BSD:

- Monitor
- Single-user
- Multiusuario

MONITOR

boot -s	Inicialización de modo monousuario
set ENV boot = cdrom	Variable de booteo
test <memory> <disk>	prueba cierto hardware

SINGLE USER

- Solo esta root
- Solo se monta / y /usr

MULTIUSUARIO

- todos los sistemas de archivos se montan
- se trabaja en red

En SysV se manejan mas niveles, estos son:

0	Shutdown
1	Estado administrativo (monousuario o Single-user)
2	Multiusuario sin red
3	Multiusuario con red
4	Definido por el administrador
5	Firware
6	Reboot

Niveles:

0 Shutdown

Se puede apagar el sistema.

4 Definido por el Administrador

Se define un single-user con red (por ejemplo).

5 Firware

Mantenimiento y diagnostico del sistema.

6 Reboot

Se cambia de nivel

EL ARCHIVO /etc/inittab

Este archivo describe como el proceso de INIT debe instalar el sistema en cierto nivel de inicialización.

Ejemplo de una parte del contenido de un archivo /etc/inittab

```
id:3:initdefault:

# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit

l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6

# Run gettys in standard runlevels
1:12345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
```

En general:

id:estado:accion:proceso

id:	es una etiqueta de 2 caracteres.
estado:	estados en los que se ejecuta el proceso.
proceso:	comando a ejecutar
accion:	es la accion que va a realizar, cualquiera de las siguientes

wait: Ejecuta y espera a que termine el proceso
respawn: Ejecuta el proceso y cada vez que este se "muera" reinicializa el proceso.
Ejemplo: getty se utiliza en este proceso.
boot: Cuando la maquina bootee.
bootwait: Espera a que termine el proceso y es cuando bootea la maquina.
initdefault: Ejecutado en estado de default.
off: Cuando encuentra un proceso con esta opción lo "mata".

BAJA DEL SISTEMA

Razones para dar de baja el sistema:

- Labores administrativas
- Instalación de patches.

Antes de dar de baja el sistema:

1. Notificar a los usuarios (mediante: wall, mail, rwall)
2. Mandarles una señal a los procesos
3. Sacar a los usuários de sesión
4. Terminar todas las operaciones de disco

El Comando: shutdown

Este comando ejecuta todos los pasos para dar de baja el sistema, por default manda a modo monousuario en BSD y SystemV, excepto en AIX en donde manda a modo monitor, su sintaxis cambia de BSD a SystemV, veamos su sintaxis en cada caso:

Para SystemV:

shutdown -g n [-i level] -y

donde:

n numero de segundos antes del shutdown
level nivel de inicialización (0,1, S[s], 5, 6)
0 Apagado total (off modo monitor)
1 Nivel Administración
Ss Single-user
5 firmware (ROM)
6 reboot
-y sin confirmación

Para BSD:

shutdown -y [grace time] "message"

donde:

-y	Sin confirmación
message	Es el mensaje avisando a los usuarios
+m	minutos
h:m	horas:minutos en formato 24 horas
now	

Ejemplo:

shutdown +60 "El sistema será dado de baja"

-f fasthalt No correr el fsck (No crear el archivo fastboot)
Para ahorrar tiempo en el "reboot".

-h halt Parar el sistema en ese momento.

-k Simula una baja del sistema, pero no lo da de baja.

-r Reboot

Sistema	Pathname	TIME	REBOOT	HALT	SINGLE-USER	NO FSCK
Solaris	/usr/sbin/shutdown	-g seg	-i6	-i0	-i5	
HPUX	/etc/shutdown	seg	-r	-h		
IRIX	/etc/shutdown	-g seg	-i6	-i0	-i5	
SunOS	/usr/etc/shutdown	+mins	-r	-h		-f
BSDI	/sbin/shutdown	+mins	-r	-h		-f

Ejemplos:

shutdown -g0 -i0
shutdown -h

Otra forma:

sync
sync
halt

y luego apagar la maquina.

sync Checa que todas las operaciones a disco terminen, es mejor dar 2 veces este comando (o hasta 3) para reducir la probabilidad de que este usando el disco y después ya darle el halt.

sync sincroniza las cabezas de los discos, con eso checamos que no se estén moviendo estas cabezas.

halt para totalmente el sistema

APAGADO DE LA MÁQUINA

OK>> power-off (después del shutdown)

Combinación de teclas stop-a

Se usa en las maquinas Sun para dar un "reset", haciendo una analogía con las Pc's. Cuando el sistema no responde y no hay forma de entrar remotamente a matar los procesos que causan el problema. Se utiliza stop-a, sin embargo no es recomendable, debido a que no se siguen todos los pasos para la baja del sistema ya mencionados. Seguramente después de stop-a habrá problemas por lo que se tendrá que correr el comando fsck.

CUENTAS DE USUARIO

Registro del usuario

Usuarios pueden ser para el sistema:

- otros usuarios (NFS)
- funciones particulares del sistema (contabilidad), por ejemplo adm es el encargado de la contabilidad.
- grupos de personas en función similar (una cuenta en la que todos tengan el password)
- un individuo

Usuario.- Ente que:

- es propietario de archivos y procesos
- edita archivos
- corre programas
- usa el sistema

Usuarios estándar en Unix

root (UID 0)	
daemon (UID 1)	demonios
bin (UID 2)	binarios del sistema
sys (UID 3)	archivos especiales del sistema muy particulares
adm (UID 4)	contabilidad
uucp	unix to unix copy protocol.
operator	en Cray, respaldos, monitoreo
nobody	nfs
guest, demo	Generalmente se deben cancelar o borrar.

Clasificación de Claves:

- Claves de usuario
- Claves de sistema

Clasificación por su uso:

- convencionales
- privilegiadas
- restringidas (Shell restringido, accesar a comandos que no estén en su path ejecutando solo un proceso, por ejemplo netscape, no permite modificar su .profile) En este tipo de cuentas podemos también restringir el uso de los comandos, prohibiendo el telnet, ftp, cd, etc. los que consideremos necesarios.

EL ARCHIVO /etc/passwd

Formato:

login:passwd-encryp:UID:GID:GECOS:DIR HOME:shell

donde:

login Es el nombre del usuario en el sistema y es único.

UID Es el numero de identificación de usuario y también es único.
Se recomienda que sea consecutivo.

UID = 0 root
UID < 100 claves del sistema

GID Es el numero de grupo al que pertenece el usuario.

GID < 10 grupos del sistema

GECOS Campo en donde se ponen comentarios como Departamento al que pertenece, teléfono, fecha de cumpleaños, etc donde se le pueda localizar, separados por coma.

DIR HOME En este campo se encuentra la ruta de su directorio HOME.
Home directory.

shell Programa a ejecutar una vez que se accede al sistema

/bin/sh
/bin/bash
/bin/ksh
/bin/rsh
/bin/csh
/bin/tcsh
/bin/netscape

passwd-encryp puede aparecer de dos formas:

- Con el password encriptado

ale:f0x2A1rU/O.Mc:577:100:Espejel Rosales Alejandro,6228169:/home/ale:/bin/tcsh

Con una "x" en el campo de password, lo cual significa que su password encriptado se encuentra en el archivo /etc/shadow.

ale:x:577:100:Espejel Rosales Alejandro,6228169:/home/ale:/bin/tcsh

Por cuestiones de seguridad del sistema es mejor que aparezca de esta ultima forma, es decir hay que utilizar el archivo /etc/shadow.

Permisos de /etc/passwd

% ls -l /etc/passwd

-rw-r--r-- 1 root root 9058 Aug 17 08:35 /etc/passwd

Ejemplo del archivo /etc/passwd

root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:
daemon:x:2:2:daemon:/sbin:
adm:x:3:4:adm:/var/adm:
lp:x:4:7:lp:/var/spool/lpd:
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:
lidy:x:576:100:Olga Lidia Torres Rivera,25/04/75,6228169,:/home/lidy:/bin/tcsh
ale:x:577:100:Espejel Rosales Alejandro,07/05/75,6228169:/home/ale:/bin/tcsh
cvc:x:599:100:Cesar Vega Calderon,02/02/74,6228169:/home/cvc:/bin/tcsh

EL ARCHIVO /etc/group

Formato:

grupo::GID:
o bien
grupo::GID:usuarios_adicionales (separados por comas)

Este archivo se define para dar ciertos privilegios a grupos de claves.
Se puede definir de dos formas:

-implicitamente Con que aparezca ese numero de gpo. en el archivo de /etc/passwd.

Ejemplo:

lidy:x:576:100:Olga Lidia Torres Rivera,25/04/75,6228169,:/home/lidy:/bin/tcsh

- explícitamente En el /etc/group darle un nombre

Ejemplo:

sys::3:root,bin,adm
adm::4:root,adm,daemon
users::100:

El segundo campo aunque no se use se mantiene, por compatibilidad con las aplicaciones. Antes se usaba como campo para password de grupo si se sabia el password, permitía cambiarse de grupo.

Grupos estándar en Unix (dependiendo del sistema)

wheel, system, root	
daemon	Demonios y servicios
kmem (BSD),sys(SV)	Todos los archivos, dispositivos y comandos que accedan a memoria
tty	Acceso a terminales

Permisos de /etc/group

```
% ls -l /etc/group
-rw-r--r--  1 root   root       389 Jul 17 12:14 /etc/group
```

Ejemplo del archivo /etc/group

```
root::0:root
bin::1:root,bin,daemon
daemon::2:root,bin,daemon
sys::3:root,bin,adm
adm::4:root,adm,daemon
tty::5:
lp::7:daemon,lp
users::100:
http:x:597:
slist:x:102:
```

Comandos:

groups	Lista los grupos a quien pertenece el usuario que ejecuta el comando.

Ejemplo:

```
lidy@aketzali:~/Cursos/Unix_Admon> groups
users

[root@aketzali /root]# groups
root bin daemon sys adm disk wheel
```

groupadd	Crea un nuevo grupo
groupmod	Modifica un grupo

EL ARCHIVO /etc/shadow

Para evitar que el resto de los usuarios vean los passwords encriptados, ya que se pueden obtener passwords encriptados generales y así adivinar un password, tomando solo los 11 caracteres ya que los de la salt son un numero entre 0 y 4096.

Formato:

login:password-encryp:lastchg:min:max:warn:inactive:expire:flag

donde:

lastchg	Numero de ultimo cambio al password, se calcula por el numero de días desde 1970 a la fecha
min	Numero mínimo de días de vigencia
max	Numero máximo de días de para cambiar el password
warn	Le avisa al usuario que tiene que cambiar su password
inactive	Numero de días inactivo.
expire	Fecha de expiración desde alta.
flag	Utilización futura no definida.

permisos de /etc/shadow

-r-------- 1 root root 5245 Aug 17 08:35 /etc/shadow

SOLO ROOT puede leerlo.

Ejemplo del archivo /etc/shadow

raizar:5rg7/1n50F.eW:10445:-1:-1:-1:-1:-1:138360140
elyna:Kq.8/zs91rkB8:10456:-1:-1:-1:-1:-1:134104286

Comando

pwconv	Cambia de una tabla de password a una tabla de shadow
pwunconv	Cambia de la tabla de shadow a la tabla de password anterior

IDENTIFICÁNDOSE EN EL SISTEMA

Login: root
passwd: ------

CREANDO CUENTAS DE USUARIO MANUALMENTE

1. Asignarle un login (verificar que ese login no exista en /etc/passwd)
2. Asignarle un UID único al usuario
3. Determinar el grupo al que va a pertenecer el usuario
4. Editar /etc/passwd y /etc/group y /etc/shadow. Modificar las tablas
5. Asignarle un password
6. Crear su directorio de trabajo HOME (mkdir)
7. Poner archivos de inicialización en su HOME

    .login
    .cshrc
    .profile
    .bash_profile (linux) < born again shell >

8. Cambiar el dueño y el grupo a su directorio HOME y a sus archivos de inicialización.

    # chown -R usuario directorio (Cambio recursivo de dueño)

    # chgrp -R grupo directorio (Cambio recursivo de grupo)

    Ejemplos:

    # chown -R ale /home/ale
    # chgrp -R users /home/ale

9. Dar de alta al usuario en otros subsistemas

    - quotas
    - tablas de impresión

10. Asignarlo a grupos secundarios (/etc/group)

11. Probar la nueva cuenta

Aunque el proceso para llevar a cabo la creación de cuentas puede seguirse paso a paso, existen utilerias (Menús gráficos por ejemplo) que permiten hacerlo, ya que en ocasiones podemos omitir un paso, a continuación se mencionaran algunas:

Mecanismos automáticos para la creación de claves.

Xenix:	mkuser
SysV:	passmgmt (Modificar la tabla de password) useradd
AIX:	mkuser, SMIT (System Manager Integrator Tool) -- IBM --
Ultrix:	adduser
HP-UX:	sam
Irix:	Sysadm (antes de la versión 5.2) Cpeople (después de la versión 5.2)
Solaris:	admintool, passmgmt,useradd

Asignando contraseñas

Se recomienda que en el momento en que se cree una cuenta se asigne su contraseña (password). Cuando se están creando varias cuentas en un servidor, se toma una convención para los passwords de las cuentas nuevas en base a su login, por ejemplo si la cuenta que voy a crear es reyna, su password seria 10NEar20 por decir algo, cumpliendo una convención que yo ponga, en este caso seria: "La longitud del login por 2, penúltima y segunda letra mayúsculas, ultima y primer letra mayúscula, y longitud del login por 4". Esta convención la impone cada administrador.

La creación de cuentas se puede automatizar, creando un script y generando una formula para la creación de passwords.

Hay que recomendarle a los usuarios que cambien inmediatamente su password en cuanto reciban su cuenta.

Si posteriormente algún usuario olvido su password, este se puede cambiar entrando a la cuenta de root y ejecutando:

```
# passwd  usuario
```

ADMINISTRACION DE LOS ARCHIVOS DE INICIO

ARCHIVOS DE INICIO DEL SISTEMA DE USUARIOS

EL ARCHIVO /etc/profile

En este archivo se puede configurar la ruta, el prompt, los permisos por default.

EL ARCHIVO /etc/skel

Normalmente aquí se encuentran los esqueletos de login. Estos archivos son los que se copian al directorio HOME de la nueva cuenta.

Contenido de /etc/skel

```
drwxr-xr-x    2 root     root      1024 May  1 14:03 .
drwxr-xr-x   27 root     root      3072 Aug 21 08:25 ..
-rw-r--r--    1 root     root      3768 Nov  7  1997 .Xdefaults
-rw-r--r--    1 root     root        24 Jul 13  1994 .bash_logout
-rw-r--r--    1 root     root       220 Aug 23 1995 .bash_profile
-rw-r--r--    1 root     root       124 Aug 23 1995 .bashrc
-rw-rw-r--    1 root     root      3336 Oct 24  1997 .screenrc
```

EL COMANDO touch

- touch crea archivos de texto pero sin contenido.

Sintaxis:

touch [opciones] <nombre_archivo>

RESPALDOS DE ARCHIVOS.

dump

Permite realizar respaldos de sistemas de archivos, ya sea completos o increméntales. Maneja nueve niveles de respaldo incremental, y lleva un registro de cuando se hizo el ultimo respaldo de cada sistema de archivos. Hace respaldos en cinta o en disco duro.

NOTA: En solaris se llama ufsdump
NOTA: Procurar que no existan usuarios en el sistema en el momento que se haga el respaldo.

Sintaxis:

% dump opciones [argumentos] sistemas_de_archivos

Opciones:

0-9	Indica el nivel del respaldo, 0 indica que es un respaldo completo.
f archivo	Permite que el respaldo se realice en un dispositivo que no sea el dispositivo estándar de cinta. Se puede usar - para especificar la salida estándar.
u	Actualiza el archivo /etc/dumpdates en caso de que el respaldo termine exitosamente.

Ejemplos:

% dump 0 /dev/rz2h	// Realiza un respaldo completo del sistema de archivos /dev/rz2h
% dump 3f /dev/rmt8 /usr/users	// Realiza un respaldo nivel 3 en la unidad de cinta rmt8 del sistema de archivos /usr/users.

restore

Permite recuperar los archivos que hayan sido respaldados por dump

Sintaxis:

% restore opciones argumentos [archivo|directorio]

Opciones:

r	Recuperar el respaldo completo.
x	Extrae los archivos del respaldo debajo del directorio actual.
t	Lista los nombres de los archivos que se encuentran dentro del respaldo.
f	Recupera los archivos de la copia de respaldo especificada en vez de la unidad de cinta por default.
i	Entra en modo interactivo, en el cual se cuenta con comandos para especificar las opciones y realizar la recuperación de los respaldos.
s	indica cual archivo será recuperado del respaldo.

Ejemplos:

% restore xf /dev/rmt1 home/curso1	// Recupera el directorio desde el dispositivo de cinta /dev/rmt1, El directorio /export/home/curso1 es buscado sobre el directorio actual (y creado si es necesario) para ser recuperado sobre el directorio actual.

```
% restore if /dev/rmt1                    // Inicializa el modo interactivo.
restore > help
  ls      // Lista el directorio.
  cd      // Cambia de directorio.
  ...
```

tar

Nos permite hacer respaldos de maneras diferentes. Su nombre significa Tape Archiver, uno de sus usos mas comunes es el manejo de archivos de disco ya que se utiliza para la distribución de programas de dominio publico.

Sintaxis:

% tar [opciones] [archivo|directorio]

Opciones:

-c	// Crea un respaldo.
-x	// Extrae los archivos de un respaldo.
-v	// Proporciona información de lo que se esta haciendo.
-t	// Lista los archivos de un respaldo.
-r	// Añade los archivos al final del respaldo.
-f archivo	// Especifica el archivo o dispositivo para hacer el respaldo en lugar de la unidad de cinta por default para realizar el respaldo.

NOTA: Todas las opciones se agrupan en un solo bloque y después se coloca la correspondiente información de cada opción en el orden correcto.

Ejemplos:

% tar -c . // Crea un respaldo en la cinta por default desde el directorio actual hacia abajo.

% tar -cvf backup.tar ./usr ./etc ./bin // Crea un archivo llamado backup.tar que contiene los directorios usr, etc y bin.

% tar -cvf - . | gzip > ../respaldo.tar.gz // Crea un respaldo desde el directorio actual y lo manda hacia la salida estándar, es recibido por el comando gzip para que lo comprima y el archivo es guardado en el archivo respaldo.tar.gz sobre el directorio padre actual.

NOTA: Nunca utilizar rutas absolutas con el comando tar.

IDENTIFICACIÓN DE USUARIOS Y GRUPOS.

Toda persona que utiliza un sistema UNIX debe contar con una cuenta, la cual esta dividida en dos partes: un nombre de usuario (o login) y una contraseña (passwd).

El nombre del usuario o login es un identificador que le indica a la computadora quien es el usuario, este es publico u tanto la computadora como otras personas pueden utilizarlo para hacer referencia al usuario en cuestión, o para comunicarse con el.

La contraseña o passwd es un autentificador, el cual se usa para comprobarle al sistema operativo que el usuario es quien dice ser, y el cual es privado y secreto.

Cada usuario en el sistema tiene asignado un identificador numérico, el cual esta directamente relacionado con el nombre del usuario, y que es la representación interna que utiliza la computadora. El nombre del usuario es únicamente una conveniencia para los seres humanos. A este identificador se le llama User ID (UID).

Cada usuario también cuenta con un identificador de grupo o Goup ID (GID) el cual representa el grupo donde se encuentra declarado el usuario. Los grupos de usuarios sirven para agrupar a los usuarios de acuerdo a sus funciones o privilegios. Cada usuario pertenece a un grupo primario, cuyo GID es almacenado en el archivo /etc/passwd. Además, puede pertenecer a varios grupos secundarios.

El archivo /etc/passwd contiene toda la información acerca de cada usuario en el siguiente formato:

- Un usuario por línea.
- Siete campos por usuario separados por ":".

```
bin:*:1:1:bin:/bin:
daemon:*:2:2:daemon:/sbin:
adm:*:3:4:adm:/var/adm:
ale:sVSxP9ASuzYa2:501:100:Alejandro Espejel:/home/ale:/bin/sh
lidy:sTq1mNchtA007:502:100:Olga Lidia Torres Rivera:/home/lidy:/bin/sh
```

CAMPO	CONTENIDO
ale	Nombre del usuario (login).
sVSxP9ASuzYa2	Password cifrado.
501	Identificador de usuario (UID).
100	Identificador de grupo (GID).
Alejandro Espejel	Nombre real del Usuario.
/home/ale	Directorio home del usuario.
/bin/sh	Shell del usuario.

El archivo **/etc/group** almacena información sobre los grupos existentes en el sistema, en el siguiente formato:

- Un grupo por línea.
- Cuatro campos por grupo, separados por ":".

root::0:root
bin::1:root,bin,daemon
daemon::2:root,bin,daemon
sys::3:root,bin,adm
adm::4:root,adm,daemon

CAMPO	CONTENIDO
Adm	Nombre del grupo.
	Password del grupo cifrado.
4	Identificador numérico de grupo (GID).
roòt,adm,daemon	Lista de usuarios que están en el grupo.

EL COMANDO id.

Es un comando con el cual se puede obtener información sobre el usuario, desde UID, el GID, y los grupos secundarios a los que pertenezca cada usuario, tanto el numero como el nombre de cada uno de ellos.

Sintaxis:

% id opciones usuario

Opciones:

-u // Imprime solamente el identificador del usuario.

-g // Imprime solamente el identificador del grupo del usuario.

-G // Imprime el/los identificador(es) de el/los grupo(s) suplementario(s) del usuario.

EL SUPER USUARIO

En todos los sistemas UNIX existe una cuenta llamada root, a la que se le conoce como el superusuario. Esta es la cuenta que se utiliza para la administración del sistema. Las características principales de root son:

- Es la cuenta que tiene el UID=0.
- Puede hacer prácticamente cualquier cosa en el sistema.
- NO es para uso personal del administrador del sistema, solamente para realizar tareas de administración.

El super usuario es la mayor falla de seguridad en UNIX. Es la cuenta que concentra todo el poder, de manera que si cae en malas manos, es muy difícil evitar que se cause daño en el sistema.

Muchos de los huecos de seguridad en UNIX son del tipo que permite a un usuario común adquirir privilegios de root. Una vez que de descubre una falla de este tipo, todos los demás mecanismos de seguridad se vuelven inútiles.

EL COMANDO su.

Este comando sirve para cambiar de identidad dentro del sistema sin necesidad de salir de sesión. Significa substitute user. Si se usa sin argumentos, significa que se quiere entrar a la cuenta de root.

Sintaxis:

% su [-] usuario

Lo que hace su es preguntar el passwd del nuevo usuario y abrir un subshell con los privilegios del nuevo usuario. Si lo ejecuta root non pregunta passwd. El carácter guión (-) hace referencia a que se deseas substituir al usuario con todos los privilegios y con las variables de ambiente y archivos de inicialización, si se omite solo se obtendrán los privilegios del nuevo usuario.

PROPIETARIO DEL ARCHIVO

Los archivos en general pertenecen a quien los crea, aunque existe una manera de cambiar el dueño y el grupo de un archivo o directorio, esto es muy utilizado para la creación de cuentas por medio del administrador.

chown

Este comando es utilizado para realizar el cambio de dueño de un archivo o directorio.

Sintaxis:

% chown opción nuevo_dueño archivo

Opción:

-R // Modo recursivo, cambia de dueño a todos los archivos y subdirectorio hacia abajo.

chgrp

Este comando es utilizado para realizar el cambio de grupo de un archivo o directorio, es similar al comando chown.

Sintaxis:

% chgrp opción nuevo_dueño archivo

Opción:

-R // Modo recursivo, cambia de grupo a todos los archivos y subdirectorio hacia abajo.

NOTA: Tanto el comando chown como el chgrp solo pueden ser utilizados por el superusuario o por el dueño del archivo o directorio.

EL DIRECTORIO /etc/default

En este directorio se encuentran diferentes archivos, los cuales especifican los valores por default para ciertos comandos

fsck

Este comando se utiliza cuando se esta levantando el sistema y no existe el archivo fastboot, esto quiere decir que por alguna causa el sistema no se dio de baja correctamente, y por lo tanto se requieren corregir los problemas que pudo haber causado la caída del sistema. Entre los problemas que existen pueden estar, la inconsistencia entre la tabla de inodos, la partición del superbloque y las entradas a los directorios del sistema de archivo, lo que es llamado la fragmentación de los sistemas de archivos.

En unas implementaciones de Unix se permite que se ejecute este comando directamente desde el disco duro y en otras se requiere reiniciar desde cdrom para ejecutar el comando. El comando fsck se debe de ejecutar en unidades que no estén montadas.

Cuando durante el proceso de levantamiento el sistema operativo no encuentra el archivo fastboot se inicia la ejecución del comando fsck para la reparación y limpieza de los sistemas de archivos automáticamente, aunque en algunos casos el fsck encuentra serios problemas y requiere la intervención del administrador.

Durante la ejecución del comando fsck se buscan archivos perdidos referentes a una locación en el disco la cual esta marcada como en uso en ls estructura de datos del disco, pero que no están listados en ningún directorio. Estos archivos perdidos son colocados en el directorio /lost+found.

La utilerias fsck revisa los siguientes puntos de la estructura de los sistemas de archivos:

Superbloque: Es un área vulnerable porque cualquier cambio en la tabla de i-nodos o en el área de datos, en el sistema de archivos se ve reflejado en esta área. En esta primera fase el fsck revisa:

- tamaño del sistema de archivos
- tamaño de la tabla de i-nodos
- lista de bitmap libre
- lista de bloques libres
- lista de i-nodos libres

COMANDOS DE ESPACIO EN DISCO.

df

Este comando genera un reporte de todos los sistemas de archivos, su capacidad total, la capacidad disponible (libre), el espacio utilizado, el porcentaje usado, además del punto de montaje del sistema de archivos y el nombre del dispositivo asociado.

Sintaxis:

```
% df -k
```

La opción k es la mas usual ya que significa que la salida la imprima en Kilobytes y no en bloques, que seria la salida sin esta opción.

du

Este comando sirve para saber cuanto espacio en el disco ha sido utilizado, por los archivos y directorios, la información al igual que en el comando df es desplegada en unidades de bloks, a menos que se utilice la opción -k, que cambia las unidades de la salida del comando a kilobytes.

Sintaxis:

```
% du -k [ruta]
```

El comando du actúa sobre en directorio actual y todos los archivos o subdirectorios hacia abajo en el sistema de archivos.

last

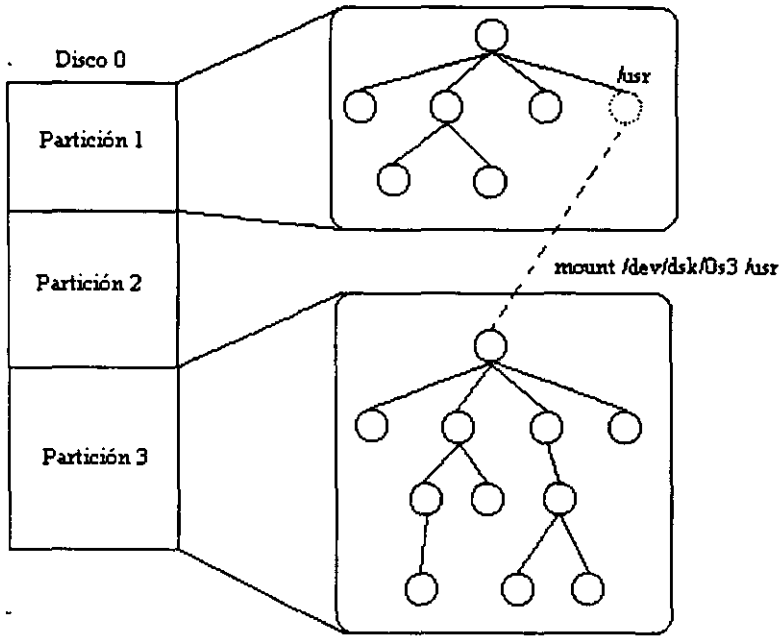
Este comando despliega información de cada vez que un usuario accedió al sistema. El comando last puede ser acompañado de una lista de nombres de usuarios o de nombres de terminales para hacer una búsqueda mas especifica.

Sintaxis:

```
% last [username | nom_terminal]
```

MONTAJE DE DISPOSITIVOS.

Aunque la raíz del sistema de archivos esta siempre almacenada en el mismo dispositivo, no es necesario que el sistema jerárquico de archivos completo resida en este dispositivo. Existe un sistema montable que solicita con dos argumentos: el nombre de directorio existente y el nombre de un archivo especial cuyo volumen de almacenamiento asociado puede tener la estructura de un sistema de archivos independiente, conteniendo su propio directorio jerárquico.



El efecto de la directiva mount es crear referencias en el directorio del primer sistema de archivos al lugar en donde se encuentra la otra raíz del sistema de archivos a montar. Después de mount no hay distinción entre archivos en el volumen removible y aquellos en el sistema permanente de archivos.

En cada sistema existe un archivo en donde se especifican los filesystems que se van a montar al arrancar el sistema, algunos de estos archivos son:

/etc/vfstab	**En solaris**
/etc/mnttab	**HP-UX**
/etc/Filesystems	**AIX**
/etc/fstab	**(estándar) Linux, IRIX**

Ejemplo de un archivo /etc/fstab:

```
# Dispositivo                tipo                    dump
#       Punto de montaje             opciones              fsck
#

/dev/hda1       /              ext2    defaults              1 1
/dev/hda6       /bt            ext2    defaults              1 2
/dev/hda5       /home          ext2    defaults,usrquota     1 2
/dev/hda7       /tmp           ext2    defaults              1 2
/dev/hda3       /var           ext2    defaults,usrquota     1 2
```

```
# fsck          checa la integridad del sistema
# mount -a  (monta todos los FS definidos en fstab) [ para BSD ]
# mount -a      [ igual pero para System V ]
          -all
```

```
# mount -F
```

Montar un CD:

Solaris:

```
# mount -f hsfs -o rc     /dev/dsk/c0t6d0s0  /cdrom
```

HP-UX

```
# mount -f cdrs -o ro /dev/dsh/c1d1s0 /cdrom
```

SunOS:

```
# mount -t hsfs -o ro /dev/sr0  /cdrom
```

IRIX: sólo se introduce el disco

Linux:

```
# mount /dev/cdrom /cdrom
```

Al ejecutar el comando df, nos muestra los archivos montados actualmente, reporta el numero de bloques de disco y archivos libres.

Ejemplo:

```
        % df
```

Salida:

```
Filesystem      1024-blocks  Used Available Capacity Mounted on
/dev/hda1         1290167  730841  492658    60%  /
/dev/hda6           69965      13   66339     0%  /bt
/dev/hda5         1492311  440216  974985    31%  /home
/dev/hda7          761835   60645  661833     8%  /tmp
/dev/hda3          349896   77300  254523    23%  /var
/dev/fd0             1423     152    1271    11%  /mnt/floppy
```

DESMONTAR

umount Desmonta sistemas de archivos

Para desmontar un disco flexible en este caso /dev/fd0, puede utilizarse este nombre (nombre de Dispositivo) o el punto de montaje /mnt/floppy.

Ejemplo:

```
% umount /mnt/floppy
```

o también

```
% umount /dev/fd0
```

ps

El comando ps lista características de los procesos del sistema. Esto es de utilidad ya que el comando produce un reporte general para los procesos que están en ejecución actualmente. Las opciones del comando controla cuales procesos son desplegados, así como la información que se va a desplegar de cada uno de ellos.

Sintaxis:

```
% ps [opciones]
```

Opción	Efecto
BSD	
ax	Despliega todos los procesos del sistema
c	Despliega el nombre del comando actual.
e	Despliega el ambiente en el cual esta ejecutando el comando
w	Produce una salida en formato amplio
System V	
-e	Despliega todos los procesos del sistema
-f	Produce una salida en formato amplio

kill

Muchas veces es necesario eliminar un proceso completo; para esto es utilizado el comando kill, donde se especifica la señal para "matar" cada proceso y el PID (identificador de proceso).

Sintaxis:

% kill [-signal] pid(s)

-9 // Es la señal para matar un proceso no importando si es un shell (incluso el shell actual).

Para saber características avanzadas sobre señales (nombre y numero) para matar procesos, revisar los manuales en línea.

at

Es un comando con el cual se puede programar la ejecución de un programa o comando a futuro, ejecutándolo solo una vez en el día y hora indicado.

Sintaxis:

% at [opciones] hora [día] [guión]

Opciones:

-r	// Borra los trabajos especificados de la cola.
-l	// Lista los trabajos especificados.
-f archivo	// Lee los comandos a ejecutar del archivo.
-hora	// Se especifica regularmente por medio de cuatro dígitos y en formato de 24 horas. Por ejemplo: 0100 es igual a 1:00 a.m., 1330 es lo mismo que 1:30 p.m.
día	// Se puede especificar el día de la semana o el mes y el día. Por ejemplo: Friday, Sep 27, etc.
guión	// Contiene los comandos que se van a ejecutar, si no se especifican, son leídos de la entrada estándar.

Ejemplos:

% at 0123
> mail ale@ds5000.super.unam.mx < reporte
> echo "El reporte se acaba de mandar" |mail lidy@ds5000.super.unam.mx
> <Ctrl-D>

Lo anterior mandara el archivo reporte a la cuneta ale y lo notificara después en la cuenta lidy a la 1:30 de la mañana.

```
% at 1700 Jan 24 ---
lp /usr/ventas/reportes/*
<Ctrl-D>
```

Se imprimirán a las 5:00 de la tarde del día 24 de Enero todos los archivos existentes bajo la ruta: /usr/ventas/reportes.

crontab

Es un comando que permite la ejecución periódica de tareas. Es atendido por el demonio cron, que revisa continuamente las tareas que se ejecutaran en un futuro y ejecuta las que sean necesarias.

Cualquier resultado a la salida estándar o a la salida de errores que produzca cron será enviado por correo electrónico al dueño de la tarea correspondiente.

Sintaxis:

% **crontab** [opciones] archivo

Opciones:

-l	// Lista el contenido del archivo de crontab.
-r	// Borra el contenido del archivo de crontab.
archivo	// Lugar donde se especifican las tareas a realizar por el comando crontab.

System V					
Minuto	**Hora**	**Día**	**Mes**	**Día_Semana**	**Comando**
0-59	0-23	1-31	1-12	0-6 (0=Domingo)	Comando

BSD						
Minuto	**Hora**	**Día**	**Mes**	**Día_Semana**	**Usuario**	**Comando**
0-59	0-23	1-31	1-12	1-7 (1=Lunes)	Usuario	Comando

Para hacer modificaciones:

% crontab -l > crontab.src	// Almacena el archivo de crontab existente.
% vi crontab.src	// Edición del archivo de crontab.
% crontab crontab.src	// Instala la nueva versión de la tabla de crontab.

Ejemplos de archivos de crontab:

0,15,30,45 * * * * /usr/lib/atrun

Ejecuta el comando atrun cada 15 minutos.

5 4 * * * /usr/adm/newsyslog

Ejecuta el comando newsyslog a las 4:05 a.m. todos los dias.

15 1 * * 1 /usr/security/cops

Ejecuta cops a la 1:15 a.m. de todos los lunes.

CONFIGURACIÓN DE LA RED

Introducción

Actualmente, las redes de computadoras permiten el intercambio de mensajes electrónicos en cuestión de segundos, entre personas que se encuentran en una misma universidad, en un mismo país o en polos opuestos del globo terráqueo. Permiten compartir tanto información como recursos (tales como impresoras y unidades de disco). Sin embargo, las redes también traen su propia dotación de problemas de seguridad, precisamente debido a su facilidad para permitir el intercambio de información y de recursos.

Internet

Internet es un conjunto de redes de computadoras interconectadas entre sí para la comunicación de datos a nivel mundial. Actualmente está presente en más de 150 países con más de 50 000 redes con millones de usuarios de todos los ámbitos. Usando una PC o una terminal en el hogar, en la escuela o en el trabajo, es posible acceder a cientos de miles de computadoras alrededor de todo el mundo. Con el programa adecuado, uno puede transferir archivos, contactarse en forma remota a una computadora que se encuentra a miles de kilómetros de distancia y usar el correo electrónico para enviar y recibir mensajes.

Cada computadora conectada a Internet tiene asignada una dirección numérica única de 32 bits. Generalmente, tales direcciones se expresan como un conjunto de 4 números de 8 bits cada uno, llamados *octetos*. Una dirección típica se ve como *132.248.120.27*. En teoría, pueden conectarse a Internet hasta 4 294 967 296 computadoras (2^{32}). En la práctica, se asignan a varias instituciones grandes bloques de números llamados *subredes*. Por ejemplo, el MIT tiene asignada la subred 18, es decir, todas las direcciones de sus *hosts* (un *host* es simplemente una computadora conectada a una red) comienzan con el número 18.

Para hacer la vida más fácil a los usuarios, se "bautiza" a las computadoras. De esta forma, una dirección alfabética consiste del *nombre de la máquina* seguido por el *nombre del dominio* en el cuál reside ésta. Por ejemplo, la computadora *132.248.159.13* se llama *tequila.super.unam.mx*. Existe software de sistemas que se encarga de traducir de manera automática nombres a números y viceversa (un ejemplo de dicho software se denomina **DNS** (Domain Name Service)) .

Modelo Cliente-Servidor

El protocolo de Internet está basado en el modelo cliente-servidor. Los programas llamados *clientes* contactan a través de la red a otros programas llamados *servidores*, los cuáles están a la espera de conexiones. En UNIX, los servidores se conocen como *demonios*.

Normalmente, los clientes y los servidores son dos programas distintos; por ejemplo, cuando uno teclea *telnet* , el programa cliente de telnet en nuestra computadora (llamado generalmente */usr/ucb/telnet*) contacta al servidor de telnet en la máquina remota (generalmente llamado */usr/etc/telnetd*). Una excepción a esto es el programa *sendmail*, que contiene tanto al cliente como al servidor en una misma aplicación.

TCP/IP

TCP/IP (*Transmission Control Protocol / Internet Protocol*), el protocolo de Internet, proporciona una transmisión bidireccional, confiable y ordenada entre dos programas corriendo en la misma computadora o en diferentes máquinas. *Confiable* significa que TCP/IP garantiza que cada byte transmitido llegue a su destino, o en su defecto, en caso de error, éste nos sea notificado. Cada conexión TCP/IP es distinta; cualquier cantidad de conexiones pueden llevarse a cabo simultáneamente mediante del envío de *streams* a través de la

misma red. TCP/IP e usa principalmente para el servicio de terminal virtual remota (telnet), transferencia de archivos (FTP) y correo electrónico (e-mail).

Puertos de red

La conexión de TCP/IP se lleva a cabo a través de *puertos* lógicos. Los puertos se identifican por números de 16 bits. De esta forma, cada conexión en Internet puede identificarse por un conjunto de 2 números de 32 bits y 2 números de 16 bits. Dichos números representan:

1. La dirección IP del emisor
2. El número de puerto del emisor
3. La dirección IP del receptor
4. El número de puerto del receptor

Cada servicio de red tiene un número de puerto "bien conocido". Por ejemplo, el número de puerto para el servicio telnet es el 23.

Puertos confiables

Los "*puertos confiables*" son los que se encuentran en el rango de 0 a 1023. En UNIX, sólo los programas que ejecuta el superusuario pueden "escuchar" u originar conexiones en estos puertos, lo cual evita que un usuario ordinario obtenga información privilegiada a través de estos puertos. Sin embargo, los puertos confiables son una *convención*, no un estándar.

UDP/IP

UDP/IP *(User Datagram Protocol / Internet Protocol)* proporciona un sistema simple y no confiable de envío de paquetes de datos entre dos o más programas corriendo en la misma computadora o en diferentes máquinas. "No confiable" significa que el sistema operativo no garantiza que cada paquete enviado llegue a su destino, o que los paquetes se envíen en orden. UDP utiliza datagramas en lugar de streams para el envío/recepción de información. La ventaja de UDP es que es un protocolo más rápido que TCP. Se usa principalmente en NFS, NIS y DNS, y servicios tales como talk y time.

Archivos y programas de red en UNIX

En UNIX, casi todos los servicios de red son proporcionados por servidores individuales. Los archivos */etc/services* y */etc/inetd.conf* controlan la ejecución de servidores cuando nuestra computadora es contactada a través de sus puertos de red.
Existen 2 tipos de servidores:

- **Servidores que siempre están corriendo** (servidores standalone). Estos servidores son iniciados automáticamente por los archivos /etc/rc* cuando inicia el sistema operativo. Un ejemplo de este tipo de servidores es el servidor de NFS (nfsd).

- **Servidores que sólo corren cuando es necesario** (servidores por petición). Un ejemplo de este tipo de servidores es el servidor de finger (fingerd).

A continuación se describen los archivos más importantes en UNIX para la configuración de la red.

El archivo /etc/hosts

Este archivo contiene una lista de las direcciones de red de cada host en una red local. Un ejemplo de este archivo se ve más o menos así:

 # /etc/hosts

 # Configuración de la máquina piaget
127.0.0.1 localhost
132.248.120.21 piaget.dgsca.unam.mx piaget

 # Nombres de hosts y direcciones IP para el manejo de sesiones remotas
132.248.120.22 carroll.dgsca.unam.mx carroll
132.248.120.25 rogers.dgsca.unam.mx rogers
132.248.120.27 entren.dgsca.unam.mx entren

En este ejemplo, la máquina *piaget.dgsca.unam.mx* tiene la dirección de red *132.248.120.21* y *piaget* puede usarse como un **alias** o segundo nombre para dicha computadora.

Cuando comenzaron a usarse las redes, existía un único archivo */etc/hosts* que contenía la dirección y el nombre de cada computadora en Internet., pero como este archivo fue creciendo a miles de líneas su mantenimiento se hizo imposible. Actualmente, la base de datos de los hosts en Internet está distribuida a través de *la Red*, de manera que cada organización actualiza sólo sus propias tablas cuando agrega una nueva computadora o cambia la dirección de una ya existente.

En Solaris, este archivo se llama **/etc/inet/hosts.**

El archivo /etc/hosts.equiv

Host confiable es un término inventado por las personas que desarrollaron el software de red de UNIX BSD. Si un host confía en otro, entonces cualquier usuario que tenga el mismo nombre de usuario en ambos hosts puede entrar al host confiable sin teclear ni login ni password.

El archivo */etc/hosts.equiv* contiene una lista de hosts confiables para nuestra computadora. Cada línea contiene un host diferente:

% cat /etc/hosts.equiv
tequila.super.unam.mx
clamato.super.unam.mx
ds5000.super.unam.mx

Este archivo permite que nuestra computadora confíe en las computadoras *tequila.super.unam.mx*, *clamato.super.unam.mx* y *ds5000.super.unam.mx*.

Algunos sistemas Sun han sido distribuidos con un archivo */etc/hosts.equiv* que contiene únicamente un signo +, lo cuál convierte a cada host conectado a Internet en hosts confiable para nuestra máquina, por lo que es muy importante borrar este símbolo de este archivo.

El archivo /etc/hosts.lpd

Si se coloca el nombre de un host en este archivo, entonces se permite que ese host utilice la impresora conectada a nuestra máquina, sin necesidad de declarar ese host como host confiable.

Por ejemplo, si se desea que el host *rogers.dgsca.unam.mx* haga uso de la impresora conectada a nuestra máquina, entonces basta con incluir la siguiente línea en dicho archivo:

% cat /etc/hosts.lpd
rogers.dgsca.unam.mx

Los archivos /$HOME/.rhosts

El término *usuarios confiables* es similar al de hosts confiables, sólo que se refiere a usuarios, no a máquinas. Si designamos a un usuario de otra computadora como usuario confiable de nuestra cuenta, entonces dicho usuario puede entrar a nuestra cuenta sin teclear un password.

El archivo */$HOME/.rhosts* permite que cada usuario construya su propio conjunto de hosts confiables, aplicables sólo a su cuenta.

Un archivo */export/home/cvc/.rhosts* que contenga lo siguiente:

% cat /export/home/cvc/.rhosts
tequila.super.unam.mx
clamato.super.unam.mx
piaget.dgsca.unam.mx cesar

permitirá que el usuario *cvc* pueda conectarse a nuestra computadora desde *tequila.super.unam.mx* o desde *clamato.super.unam.mx* sin teclear login ni password. Además, este archivo permite que un usuario llamado *cesar* se conecte a esta máquina desde el host *piaget.dgsca.unam.mx*.

Desde el punto de vista del usuario, los conceptos de hosts y usuarios confiables son muy cómodos, pero desde el punto de vista de la seguridad, son muy peligrosos, por lo que se recomienda no hacer uso del archivo */etc/hosts.equiv* y borrar todos los archivos *.rhosts* de los directorios home de los usuarios de nuestro sistema.

El archivo /etc/services

Este archivo contiene todos los servicios de red que UNIX implementa y también los que no. Cada línea de este archivo contiene a su vez el nombre del servicio, el número de puerto, el nombre del protocolo y una lista de alias. Un extracto de este archivo se vería como sigue:

% cat /etc/services
# /etc/services
#
telnet 23/tcp
smtp 25/tcp mail
time 39/udp timeserver

El programa /etc/inetd

Inicialmente, en UNIX se tenía corriendo un programa servidor diferente para cada servicio de red, pero como el número de servidores vino en aumento, comenzó a tenerse más y más servidores esperando conexiones de red, "durmiendo" en background, consumiendo recursos. Fue entonces que se desarrolló un programa servidor llamado **/etc/inetd**, el cual escucha muchos puertos de red al mismo tiempo y ejecuta el servidor por petición adecuado (basado en TCP o UDP) cuando recibe una conexión.

inetd es ejecutado por */etc/rc* cuando arranca el sistema. Entonces examina el contenido del archivo **/etc/inetd.conf** para saber que servicio de red va a manejar.

Un ejemplo de un archivo /etc/inetd.conf se muestra a continuación:

% cat /etc/inetd.conf

```
# Base de datos de configuración del servidor de internet.
#
ftp     stream  tcp     nowait  root    /usr/etc/ftpd ftpd
telnet  stream  tcp     nowait  root    /usr/etc/telnetd telnetd
uucp    stream  tcp     nowait  uucp    /usr/etc/uucpd uucpd
finger  stream  tcp     nowait  nobody  /usr/etc/fingerd fingerd
tftp    dgram   udp     wait    nobody  /usr/etc/tftpd tftpd
talk    dgram   udp     wait    root    /usr/etc/talkd talkd
time    stream  tcp     nowait  root    internal
echo    dgram   udp     wait    root    internal
```

Cada línea contiene al menos 6 campos, separados por espacios o tabuladores:

Nombre del servicio
El nombre del servicio que aparece en el archivo */etc/services*. **inetd** usa este nombre para determinar el número de puerto donde debe "escuchar" este servicio.

Tipo de socket
Si el servicio espera comunicarse mediante streams (stream) o a través de datagramas (dgram).

Tipo de protocolo
Si el servicio espera usar comunicaciones basadas en TCP o en UDP. TCP se usa con sockets stream, en tanto UDP utiliza datagramas.

Esperar/no esperar
Si el campo es *wait*, el servidor espera a que se procesen todos los datagramas subsecuentes recibidos en el socket. Si se especifica *nowait*, **inetd** crea y ejecuta un nuevo proceso servidor para cada datagrama adicional o solicitud de conexión recibida.

Usuario
Especifica el UID con el que se ejecuta el proceso servidor. Puede ser root (UID 0), daemon (UID 1), nobody (generalmente, UID -2) o un usuario actual del sistema. Este campo permite que los procesos servidores se ejecuten con menos permisos que root.

Nombre del comando y argumentos
Especifican el nombre del comando que va a ejecutarse y los argumentos pasados al comando, iniciando con argv[0] (el nombre del programa).

Algunos servicios, como echo, time y discard, listados como *internal*, son funciones tan triviales que son manejadas internamente por **inetd**, en lugar de ser ejecutadas por un programa especial.

El archivo /etc/sys_id (IRIX)

Este archivo contiene el nombre de la máquina. En IRIX, el nombre por default es IRIS.
% cat /etc/sys_id
vodka

En Solaris, este archivo se llama **/etc/nodename.**

El archivo /etc/resolv.conf (Solaris)

En este archivo se configura el DNS.

% cat /etc/resolv.conf
domain super.unam.mx

 # DNS's primarios y secundarios
nameserver 132.248.10.2
nameserver 132.248.204.1
nameserver 132.248.1.3

En IRIX no existe este archivo. El DNS se configura en /etc/hosts.

El archivo /etc/init.d/network.local (IRIX)

Este archivo contienen el ruteador por default.

% cat /etc/init.d/network.local
 # agregar el ruteador por default
route add default 132.248.159.1

Es IRIX, es necesario hacer las siguientes ligas simbólicas a este archivo para que se llamen los siguientes scripts al iniciarse y darse de baja el sistema:

% ln -s /etc/init.d/network.local /etc/rc0.d/K39network
% ln -s /etc/init.d/network.local /etc/rc2.d/S31network

Este script se llama con los argumentos *stop* (para "tirar" los servicios de red) o *start* (para "levantar" los servicios de red).

En Solaris, este archivo se llama **/etc/defaultrouter.** Sólo es necesario incluir la dirección IP del ruteador.

Servicios de red en UNIX

La siguiente tabla muestra los principales servicio de red que proporciona UNIX.

Nombre del servicio	Cliente	Servidor	Descripción	Sintaxis
telnet	telnet	telnetd	Proporciona el servicio de "terminal virtual remota", esto es, permite a un usuario ingresar a un sistema y usarlo como si estuviera sentado en una terminal directamente conectada a él. Se requiere tener una cuenta en dicha máquina.	**% telnet máquina_remota**
rlogin	rlogin	rlogind	Similar a telnet, sólo que no se requiere teclear el login, pues este se transmite automáticamente; si se usa el concepto de hosts confiables o de usuarios confiables, tampoco es necesario teclear un password. Sólo funciona en sistemas UNIX BSD.	**% rlogin máquina_remota**
rsh	rsh	rshd	Permite que un usuario ejecute un sólo comando en una máquina remota. Sólo funciona con el concepto de hosts y usuarios confiables. Funciona únicamente en sistemas UNIX BSD.	**% rsh máquina_remota**
finger	finger	fingerd	Si se usa sin argumentos, imprime información sobre los actuales usuarios de nuestro sistema; si se utiliza con el nombre de una máquina remota como argumento, entonces muestra información sobre quiénes están trabajando en ese momento en dicha máquina; si se usa como argumento el nombre de usuario y de la máquina remota, entonces imprime información sobre dicho usuario, aún cuando no esté en sesión en ese momento.	**% finger** **% finger @máquina_remota** **% finger usuario@máquina_remota**
e-mail	sendmail	sendmail	Permite el envío y recepción de mensajes electrónicos. Se requiere tener una cuenta de correo electrónico en algún host, la cuál consiste de un login y el nombre de la máquina: **login@host.**	**% mail e-mail_destino < mensaje,** dónde *mensaje* es un archivo que contiene el mensaje a ser enviado.
ftp	ftp	ftpd	Permite la transferencia de archivos entre sistemas con diferentes sistemas operativos. Normalmente se requiere tener una cuenta en la máquina remota, pero si el servidor ofrece el servicio de FTP anónimo, entonces basta con teclear la palabra **anonymous** ó **ftp** como login y la dirección de correo electrónico como password para acceder al sistema. Una excepción a esto es si nuestra cuenta está especificada en el archivo **/etc/ftpusers** del servidor, ya que de ser así, no podremos hacer uso de este servicio, pues este archivo lista a los usuarios que no pueden hacer uso de FTP.	**% finger máquina_remota**

Expresiones regulares

Las **expresiones regulares** son patrones que tienen ciertas semejanzas o contienen partes comunes, aunque no sean exactamente iguales.

La sintaxis de las expresiones regulares utiliza caracteres y **metacaracteres** (los metacaracteres son un conjunto de caracteres con significado especial para las búsquedas). Los principales metacaracteres utilizados en expresiones regulares son:

Metacaracter	¿A qué se refiere?	Ejemplos
^ (circunflejo)	Al principio de la línea.	^I, busca las líneas que inician con I.
$	Al final de la línea.	f$, busca las líneas que terminan con f.
. (punto)	Es un comodín, puede sustituir a cualquier caracter.	co.a, coincide con cosa, cota, coma, cola, etc.
*	Es un comodín; significa cero o más ocurrencias del caracter anterior. Si se combina con el punto, entonces .* significa cualquier secuencia de caracteres, incluyendo ninguno.	car*o, coincide con cao, caro, carro, carrro, etc. h.*a, coincide con ha, hola, hilda, honda, harta, etc.
[ ]	Indica los caracteres específicos que se quiere que sean aceptados en una posición específica dentro de la cadena de búsqueda	[aeiou], coincide con cualquiera de las vocales. [a-zA-Z], coincide con cualquiera de las letras del abecedario [a-z0-9], coincide con cualquiera de las letras del abecedario en minúsculas o con cualquier dígito del 0 al 9.
[^]	Indica los caracteres específicos que se quiere que **no** sean aceptados en una posición específica dentro de la cadena de búsqueda	[^XYZ], coincide con cualquier caracter que no sea X ni Y ni Z. [^0-9], coincide con cualquier caracter que no sea un dígito decimal.

Algunas veces se necesita localizar dentro de un archivo alguna cadena que contenga un metacaracter; en este caso se desea que dicho metacaracter **no** sea interpretado por el comando utilizado como expresión regular, sino como literalmente un caracter. Para lograr esto, basta con anteponer una diagonal invertida (\ ) al metacaracter, esto es, "escapar" al metacaracter.

Cuando se utilizan expresiones regulares, es recomendable encerrar entre comillas dobles (" ") el patrón de búsqueda, ya que muchos metacaracteres de expresiones regulares también tienen un significado especial para el shell, lo cual puede causar confusiones.

El comando grep

El comando **grep** (*global regular expression print*) sirve para localizar una cadena en un archivo de texto.

Sintaxis:

grep [*opciones*] *patrón* [*archivos(s)*]

Opciones:

-i *Considera las minúsculas y las mayúsculas como iguales.*
-v *Imprime las líneas que NO contienen el patrón especificado.*
-c *Indica el número de líneas que contienen el patrón, sin imprimir las líneas en sí.*
-n *Imprime el número de la(s) línea(s) que contiene(n) el patrón con respecto a su posición relativa dentro del archivo.*
-l *Imprime los nombres de los archivos que contienen en alguna(s) de sus líneas el patrón especificado.*

Ejemplos:

% grep "^s" /etc/group
Muestra todas las líneas del archivo /etc/group que comienzan con la letra s.

% grep "daemon$" /etc/group
Muestra todas las líneas del archivo /etc/group que terminan con la palabra daemon.

% grep -v root /etc/passwd
Muestra todas las líneas del archivo /etc/passwd que *no* contienen la cadena root.

% grep -i "ch.*se" diccionario
Muestra todas las líneas del archivo diccionario que contienen palabras tales como chse, chocaSE, cheese, CHINESE, Choose, etc.

% grep -l señor carta1 carta2 carta3
Muestra los nombres de los archivos (ya sea carta1, carta2 o carta3) que contienen la palabra señor en cualquiera de sus líneas.

% grep -c user /etc/group
Muestra el número de líneas en que aparece la palabra user dentro del archivo /etc/group.

% grep "/bin/sh$" /etc/passwd
Muestra todas las líneas del archivo /etc/passwd que corresponden a usuarios que tienen asignado el Bourne shell.

El comando fgrep

Este comando permite buscar múltiples patrones en uno o más archivos, pero no permite la utilización de expresiones regulares, lo cuál lo hace más rápido que grep.

Sintaxis:

fgrep *[opciones] "patrón1*
patrón2
patrón3
...
patrónN" [archivo(s)]

Otra forma de especificar los patrones a buscar es guardándolos en un archivo, poniendo cada patrón en una línea separada. Además de aceptar todas las opciones de grep, fgrep acepta una opción más:

-f *archivo* *toma del archivo especificado los patrones de búsqueda*

Ejemplos:

% fgrep "exito
> logro
> triunfo" curriculum.txt
Muestra las líneas del archivo curriculum.txt que contienen cualquiera de las palabras exito, logro o triunfo.

% cat patrones
^c
:$
% fgrep patrones /etc/passwd
Muestra las líneas del archivo /etc/passwd que inician con *c* o terminan con *:*.

El comando egrep

Este comando incluye lo mejor de grep y fgrep, y añade aún más cosas. Como grep, permite la búsqueda de expresiones regulares y añade algunos metacaracteres adicionales para realizar búsquedas extendidas.

egrep dispone de los siguientes metacaracteres:

+ *Indica una o más repeticiones del caracter anterior, en vez de cero o más, como el asterisco.*
? *Indica cero o una repeticiones del caracter anterior*

Como fgrep, permite buscar múltiples patrones y extraerlos de un archivo. Para especificar varios patrones de búsqueda, se pueden colocar en líneas separadas, como en fgrep, pero también se pueden separar por una barra vertical (|), bien, puede usarse la opción -f para leer los patrones desde un archivo.

Ejemplos:

% egrep -vf nombres alumnos
Imprime las líneas del archivo alumnos que *no* contengan ninguno de los nombres especificados en el archivo nombres.

% egrep "file[01]?[0-9]" lista
Imprime las líneas del archivo lista que contengan una cadena entre file0 y file19.

El comando find

El comando **find** permite localizar un grupo específico de archivos en todo el sistema de archivos (siempre y cuando se cuente con los permisos necesarios). Para realizar las búsquedas utiliza expresiones lógicas, las cuales pueden ser utilizadas solas o combinadas, usando los *operadores* AND (-a), OR (-o) y NOT (\!). Cuando los archivos son encontrados, pueden ejecutarse ciertas *acciones* sobre ellos.

Sintaxis:

find *ruta operadores y acciones*

Operadores:

-atime +n | n | -n Encuentra los archivos accedidos hace:
+n para más de *n* días
n para exactamente *n* días
-n para menos de *n* días
-mtime Encuentra los archivos modificados en los días especificados.
-size Encuentra los archivos con el tamaño especificado en bloques
-type Encuentra los archivos del tipo indicado, donde el tipo puede ser:
b para archivos de bloque
c para archivos de caracter
f para archivos ordinarios
d para directorios
l para ligas
-name Encuentra los archivos cuyo nombre coincida con la expresión regular indicada.
-perm Encuentra los archivos con los permisos indicados en modo octal.
-user Encuentra los archivos pertenecientes al usuario indicado
-group Encuentra los archivos pertenecientes al grupo indicado.

Acciones:

-print Imprime la ruta absoluta de los archivos encontrados.
-ls Lista con el formato indicado los archivos encontrados.
-ok Ejecuta *con* confirmación el comando indicado.
-exec comando {} \, Ejecuta *sin* confirmación el comando indicado (el argumento {} se sustituye con el archivo actual que ha encontrado find)

Ejemplos:

% find . -type d -exec rm -ri {} \;
Encuentra y borra (solicitando confirmación) todos los directorios a partir del directorio actual.

% find / -name "*.sh" -exec chmod u+x {} \;
Encuentra a partir del directorio raíz y hace ejecutables para el dueño todos los archivos que tengan la extensión .sh.

% find /usr/bin -perm -4000 -type f -exec ls -l {} \;
Encuentra a partir del directorio /usr/bin todos los archivos de tipo ordinario que tengan el SUID encendido y los lista con un formato largo.

% find /dev \(-type c -o -type b \) -ok rm {} \;
Encuentra a partir del directorio /dev todos los archivos de tipo caracter o de tipo bloque (archivos de dispositivos) e intenta borrarlos, solicitando confirmación.

% find /tmp -atime -2 -exec ls -l {} \;
Encuentra a partir del directorio /tmp todos los archivos que hayan sido accedidos en los últimos 2 días y los lista con un formato largo.