UNIVERSIDAD DE LOS ANDES
III ESCUELA LATINOAMERICANA DE REDES
MÉRIDA - EDO MÉRIDA - VENEZUELA

TALLER UNIX

Elaborado por:
José Vicente Núñez Zuleta

TABLA DE CONTENIDO.

Agradecimientos 5

Introducción 6

¿Cómo puedo ser un buen usuario? 7

Breve historia del sistema UNIX 7

Como entrar y salir del sistema operativo 7

ADVERTENCIA: Cuando debe usar el login root *8*

Como cambiar de password 8

Documentación en línea del sistema 8

NOTA sobre la ayuda en línea: *10*

¿Quién está en el sistema? 10

Nociones de archivos y directorios 11

¿Qué es un archivo? *11*

Árbol de directorios en UNIX (Linux) y permisología de archivos 11

Moviéndose entre directorios *12*

Atajos para moverse entre directorios *13*

Creando y eliminando directorios *14*

Viendo los atributos de los archivos 15

Dos atajos útiles: * y ? 17

Uso adicional de los caractéres especiales: *17*

Viendo el contenido de un archivo de texto 18

Copiando, moviendo, cambiando el nombre a los archivos 20

Creando archivos 21

Permisología y propiedad de archivos 22

Permisología de los archivos *22*

NOTA acerca de la permisología de los archivos: *23*

Propiedad de los archivos *25*

NOTA sobre el cambio de grupos y usuarios: *26*

Enlaces 26

Edición de archivos con el editor vi 28

NOTA sobre los comandos de vi: *30*

Buscando caracteres *32*

¿Cómo busca caracteres el usuario avanzado? *32*

Reemplazando caracteres *33*

¿Cómo ser un usuario avanzado? 33

Ejecución de procesos en segundo plano 33

Estado del proceso de un usuario 33

Terminación de procesos en reserva 34

Ejecución de un proceso en un instante específico 35

Combinación de múltiples comandos 36

Entrada y salida de comandos 37

Completación de comandos 38

Uso de patrones 39

Reutilización, repetición y sustitución de comandos 39

Creando atajos para comandos 41

Usando sustitución de comandos 41

Usando control de trabajos 42

Trabajo en maquinas remotas 44

Copiando archivos de un sistema a otro 46

Comandos miscelaneos para el manejo de archivos 49

Comprimiendo archivos con gzip *49*

Empaquetando archivos con tar (tape archiver) *49*

Busquedas avanzadas de patrones y archivos 50

Uso de grep para buscar patrones *50*

Uso de find para buscar archivos *50*

Bibliografía 52

INDICE DE ILUSTRACIONES

Ilustración 1: Árbol de directorios bajo Linux.....11

Ilustración 2: Tipos de permisos presentes en un archivo bajo UNIX 22

Ilustración 3: Cadena de acceso para modificar la permisología de un archivo 23

Ilustración 4: Equivalencia en números binarios de la permisología de un archivo 24

Ilustración 5: Teclas substitutas de movimiento para Vi ..30

INDICE DE TABLAS

Tabla 1: Comandos usuales del editor Vi.....32

Tabla 2: Comandos usados para control de trabajos.....42

AGRADECIMIENTOS

Quisiera agradecer a las siguientes personas por sus correciones y sugerencias en la elaboración de este documento:

- Daniel Pradilla (daniel@ing.ula.ve)
- Nestor Angulo Reina (nestor@ing.ula.ve)
- Mónica Venegas (monicav@ing.ula.ve)
- Alpha Pernia (alpha@ing.ula.ve)
- Gustavo Camargo (tavo@ula.ve)
- Leonardo Contreras (leoc@ula.ve)
- Pauly Quintero (pauly@ula.ve)
- Carlos Navas (navac@ula.ve)
- Vanessa Hamar (vanessa@ula.ve)

INTRODUCCIÓN

Esta guía tiene como objeto ser una referencia rápida a aquellos comandos y órdenes que son usadas en UNIX con mayor frecuencia, ya sea como usuario (principiante o avanzado).

Para que el uso de esta guía sea más efectivo se recomienda que el lector vaya practicando los comandos aquí descritos al lado del computador, para que pueda comprobar los resultados por sí mismo. También se recomienda que el lector consulte la bibliografía al final de la guía para que así profundice sus conocimientos.

Todos los comandos fueron probados bajo Linux (distribución Slackware). Debido a la variedad de implementaciones de UNIX la sintaxis de estos comando puede variar.

Esta guía siempre estará bajo construcción y puede contener errores.

Esta guia viene sin garantías de ningún tipo, y se prohibe su uso para fines comerciales.

Este documento se publica bajo la Licencia General GNU, lo que permite su libre distribución gratuita siempre que se conserve en su integridad y se retenga el copyright original.

Este guía tiene un copyright de 1997 por José Vicente Núñez Zuleta. Puede ser reproducido en cualquier forma y distribuido libremente mientras el fichero se mantenga intacto , incluyendo esta línea.

Si tiene dudas, adiciones, comentarios acerca de esta guia puede contactarme en la siguiente dirección:

Jose@ing.ula.ve

La última versión de este documento puede ser conseguida en:
http://www.ing.ula.ve/~jose

Todas las marcas registradas pertenecen a sus respectivos autores.

¿CÓMO PUEDO SER UN BUEN USUARIO?

BREVE HISTORIA DEL SISTEMA UNIX

UNIX es uno de los sistemas operativos más populares en el mundo entero debido a su gran base de soporte y distribución. Fue desarrollado originalmente como un sistema operativo multitarea para minicomputadoras y mainframes a mediados de los años 70, pero desde entonces ha crecido hasta convertirse en uno de los sistemas operativos más usados en cualquier lado, pese a su interfaz confusa y su falta de estandarización.

UNIX es un sistema operativo multiusuario, multitarea. Esto significa que muchas personas pueden estar usando la computadora al mismo tiempo, corriendo diferentes aplicaciones.

En este manual trabajaremos con una implementación de UNIX llamada *Linux*. Linux es un clone de UNIX diseñado (en un principio) para las plataformas Intel 80386 y 80486.

Linux soporta un amplio rango de software como TeX, X window, el compilador GNU C/C++ y TCP/IP. Linux es versátil, es distribuido de manera gratuita en los términos de la Licencia Publica General de GNU (GNU General Public License), lo que permite que programadores de todo el mundo hagan herramientas para él a un costo muy bajo.

COMO ENTRAR Y SALIR DEL SISTEMA OPERATIVO

Debido a que UNIX es un sistema operativo multiusuario, para empezar a utilizar el sistema los usuarios deben realizar dos pasos:

1. Deben identificarse frente al sistema (login)
2. Deben demostrar que son los usuarios autorizados de la cuenta (password).

Para entrar a su maquina con Linux escriba lo siguiente:

En el aviso de entrada escriba su login (en el ejemplo es angel)

```
Linux 2.0.27 (leon.felinos.ve) (ttyp0)


leon login: angel
```

Luego introduzca su palabra clave (esta no se muestra por pantalla por razones de seguridad).

```
Password:
```

Si el intento es exitoso el sistema le permitirá entrar.

```
Last login: Fri Sep 26 22:37:46 from tigre.felinos.ve
Linux 2.0.27.
angel has logged on ttyp0 from pantera.
leon:~>
```

Y para salir escriba la palabra exit o logout

```
leon:~>exit
```

ADVERTENCIA: CUANDO DEBE USAR EL LOGIN *ROOT*

Usted tendrá que utilizar el login *root (superusuario)* de vez en cuando (si es adminisatrador del sistema, claro está). Algunas cosas no pueden ser hechas bajo UNIX sin entrar al sistema como root. Sin embargo debe evitar utilizar la cuenta de root como su cuenta regular ya que la cuenta de root no tiene limitaciones de ningún tipo en cuanto a permisos de modificación de archivos (para un nuevo usuario es muy fácil destruir un sistema UNIX con un solo comando).

Use esta cuenta sólo cuando sea necesario.

COMO CAMBIAR DE PASSWORD

Es conveniente cambiar su palabra clave de vez en cuando, ya que esta puede ser averiguada por personas sin escrúpulos que tratan de entrar al sistema (de hecho esto es muy fácil de hacer y hay muchas herramientas disponibles para ello).

Para cambiar su password debe entrar a su cuenta (como se vio anteriormente) y debe escribir:

```
tigre:~> passwd
```

La maquina le preguntará el password viejo

```
Changing password for angel
Enter old password:
```

Una vez introducido correctamente, el sistema le preguntará el password nuevo

```
Enter new password:
```

Se le pedirá otra vez el password (para verificar que lo haya escrito bien).

```
Re-type new password:
```

Una vez introducida la palabra clave, el sistema le avisará que el password fue cambiado con éxito.

```
Password changed.
tigre:~>
```

DOCUMENTACIÓN EN LÍNEA DEL SISTEMA

UNIX posee en línea información acerca de todos sus comandos. Para verlos sólo tiene que escribir:

```
man [comando]
```

Por ejemplo, suponga que usted quiere aprender más acerca del comando *passwd* que acaba de aprender. Para ello sólo tiene que escribir:

```
tigre:~> man passwd
```

Y verá una salida como la siguiente

```
PASSWD(1)           Linux Programmer's Manual
PASSWD(1)

NAME
       passwd - change password
```

SYNOPSIS

passwd [name [password]]

DESCRIPTION

Without arguments passwd will change the password for the current user. First the user is asked for the old password, then prompted twice for the new password in order to catch typing errors. The new password must be at least six characters long, and have both upper and lower case letters or non-letters. The new password must not be equal to the old password, and it must not match the username.

The one and two argument forms may only be used by the superuser. Using the one argument form, the superuser may change the password for that user. The superuser is not asked for the users old password, and the rules for proper legitimate reasons to choose a non-conformant password.

The two argument form gives the user the password stated as the second argument. This may be useful when giving many users an initial generated password.

Giving an empty string as the second argument erases the password for the user.

FILES

/etc/passwd
/etc/shells

SEE ALSO

chsh(1), chfn(1)

BUGS

A password consisting of all digits is allowed.
No warnings are printed if the superuser chooses a poor password.
The -f and -s options are not supported.

Peter Orbaek (poe@daimi.aau.dk)

Quizás se haya dado cuenta que los comandos parecen estar seguidos de números entre paréntesis, usualmente (1). Esto se refiere a una sección del manual. Antes, cuando los manuales de UNIX venían impresos en volúmenes, la información era organizada de la siguiente manera (nos referimos a la familia de UNIX BSD, no System V):

- Comandos del usuario en la sección 1
- Llamadas del sistema en la sección 2
- Rutinas en C y otras bibliotecas de rutinas en la sección 3
- Archivos especiales, manejadores de dispositivos, hardware en la sección 4
- Archivos de configuración en la sección 5
- Juegos en la sección 6
- Comandos misceláneos en la sección 7
- Comandos de administración en la sección 8
- Comandos de mantenimiento en la sección 9

Para consultar secciones distintas de un comando debe escribir *man sección comando*. Por ejemplo, las siguientes ordenes producen salidas de información diferentes:

tigre:~> ***man 2 mount*** (manual de programadores para mount).

tigre:~> ***man 8 mount*** (se muestra el uso de la herramienta mount).

A veces usted sabe que quiere hacer, pero no sabe con qué comando hacerlo. Suponga por ejemplo que usted quiere saber como cambiar el nombre a los archivos bajo UNIX:

```
tigre:~> man -k rename
mren (1)              - rename an existing MSDOS file
mv (1)                - rename files
rename (2)            - change the name or location of a file
rename (3tcl)         - Rename or delete a command
XSetWMName, XGetWMName, XStoreName, XFetchName (3x) - set or
read a window's WM_NAME property
tigre:~>
```

Con `man -k [comando]` puede hacer una buena búsqueda (como puede ver se retornan muchos resultados).

NOTA sobre la ayuda en línea:

Recuerde que de ahora en adelante puede obtener ayuda en línea sobre los comandos que utilizará en esta práctica.

Para que el comando *man –k* funcione en el sistema, el administrador del sistema deberá ejecutar el comando *catman –k* periodicamente.

¿QUIÉN ESTÁ EN EL SISTEMA?

El comando *who* le permite ver quien está en el sistema, de manera sencilla:

```
tigre:~> who
root     tty1     Sep 27 11:00
josev    ttyp0    Sep 27 10:38 (pantera.felinos.ve)
jose     ttyp1    Sep 27 10:59 (pantera.felinos.ve)
foto     ttyp2    Sep 27 10:59 (pantera.felinos.ve)
```

```
tigre:~>
```

De izquierda a derecha se muestra al login del usuario, el terminal en el que está conectado (en el ejemplo todos son terminales *virtuales* excepto el que dice tty1 que es por consola) la fecha y hora en que se inició la conexión y el lugar desde donde se lleva a cabo (todos se están conectando de manera remota desde una maquina llamada pantera.felinos.ve excepto root).

NOCIONES DE ARCHIVOS Y DIRECTORIOS

¿QUÉ ES UN ARCHIVO?

Un archivo es la unidad básica del sistema operativo UNIX. Casi todo es tratado como un archivo, incluyendo:

- *Documentos*: Esto incluye archivos de texto, como cartas, código fuente de computadora, páginas escritas en html, o cualquier cosa que usted escriba.
- *Comandos*: Muchos comandos son archivos ejecutables; Ellos son archivos que usted puede correr. Por ejemplo, el comando pwd (Print Working Directory) es un comando ejecutable.
- *Dispositivos*: UNIX trata su terminal, impresora y unidades de disco como archivos. Por ejemplo el dispositivo /dev/fd0 (unidad de disco flexible 1) es un archivo.
- *Absolutamente nada*: Inclusive hay un archivo especial llamado */dev/null* , el cual es un agujero negro (cualquier tipo de dato enviado allí es descartado).
- *Directorios*: Un directorio es un archivo que sirve para contener otros archivos.

ÁRBOL DE DIRECTORIOS EN UNIX (LINUX) Y PERMISOLOGÍA DE ARCHIVOS

Existe una jerarquía de directorios los cuales organizan las utilidades y herramientas del sistema operativo UNIX (todas sus ramas comienzan desde /). Usted puede ver el árbol de directorios escribiendo:

```
tigre:~> ls /
System.map   cdrom/       lib/         quota.group  tmp/
dev/         lost+found/  quota.user   usr/ bin/         etc/
mnt/         root/        var/
boot/        home/        proc/        sbin/        vmlinuz
tigre:~>
```

Veamos la jerarquía de directorios como un árbol invertido:

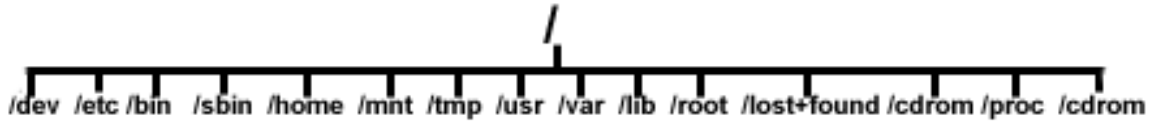

Ilustración 1: Árbol de directorios bajo Linux

Esto es más o menos lo que contiene cada directorio:

- /: El directorio raíz. Es la base la estructura del árbol del sistema de archivos. Todos los demás archivos y directorios, independientemente de su localización física, son contenidos físicamente en este directorio.

- /bin: Directorio de comandos binarios. Incluye los ejecutables públicos que son parte del sistema operativo UNIX.
- /sbin: Ejecutables de administración y funcionamiento del sistema
- /dev: Directorio de dispositivos, contiene archivos especiales como impresoras, discos, consola, etc.
- /etc: Archivos de configuración del sistema. Este directorio contiene muchos archivos de administración y archivos de configuración.
- /lib: Contiene los archivos de librerías para C y otros lenguajes.
- /Lost+found: Directorio de archivos perdidos. Errores de disco o un apagado incorrecto del sistema puede causar que los archivos se *pierdan* (archivos perdidos se refieren a sitios en disco que están marcados como usados en las estructuras de datos de los discos, pero que no están listados en ningún directorio). La herramienta *fsck* encuentra esos archivos y los coloca en este directorio.
- /mnt: Directorio de montaje. Este directorio está vacío y se usa para montar temporalmente particiones de discos.
- /home: Directorio hogar de los usuarios.
- /usr: Contiene directorios para las colas de impresión, correo, programas generados localmente, ejecutables para los comandos de usuarios.
- /tmp: Directorio temporal, disponible para todos los usuarios como directorio para archivos transitorios.
- /var: Guarda los directorios volátiles de las colas del sistema (impresión, correo, mensajes).

MOVIÉNDOSE ENTRE DIRECTORIOS

Supongamos que queremos movernos al directorio */etc* y queremos examinar los archivos que allí se encuentran. Para ello utilizaremos las herramientas *cd y pwd*:

Vemos en que directorio estamos parados actualmente

```
tigre:~> pwd
/home/angel
tigre:~>
```

Vemos que estamos en nuestro directorio hogar (/home/angel) . Ahora vamos a /etc utilizando una *ruta absoluta.*

```
tigre:~> cd /etc
tigre:/etc>
```

Y finalmente pedimos un listado de los archivos que están allí con *ls (Se muestra sólo un listado parcial)*

```
tigre:/etc> ls
DIR_COLORS          ftpconversions      makedev.cfg
sendmail.st
HOSTNAME            ftpgroups           motd
services
NETWORKING          ftpusers            msgs/               shells
NNTP_INEWS_DOMAIN   gateways            mtab                skel/
X11@                group               mtools.conf
slip.hosts
XF86Config          host.conf           named/
slip.login
```

```
XF86Config.bak     hosts               named.boot
snooptab
XF86Config.orig    hosts.allow         named.boot.orig
ssh_config
tigre:/etc>
```

Veamos de nuevo en qué directorio estamos trabajando

```
tigre:/etc> pwd
/etc
tigre:/etc>
```

Vimos que hay un directorio llamado *skel* dentro de /etc cuando usamos ls (la ruta completa del directorio es /etc/skel). Podemos ir allá (desde cualquier sitio) utilizando una ruta absoluta:

```
tigre:~> cd /etc/skel/
tigre:/etc/skel>
```

Pero *si aún estamos parados en /etc* podemos entrar al directorio *skel* utilizando una *ruta relativa (*en este caso a etc)

```
tigre:/etc> cd skel/
tigre:/etc/skel>
```

Estamos trabajando en /etc/skel. Supongamos que queremos retroceder a */etc sin utilizar una ruta absoluta*. Podemos hacerlo (para este y cualquier directorio) con la siguiente instrucción

```
leon:/etc/skel> cd ..
leon:/etc>
```

La notación ".." significa el nivel anterior, mientras que la notación "." significa el directorio actual.

Se pueden mezclar tantos niveles de ".." como se quiera (con esto regresamos desde /etc/skel a / utilizando una ruta relativa).

```
leon:/etc/skel> cd ../../
leon:/>
```

Ahora regresemos a nuestro directorio hogar

```
tigre:/etc> cd /home/angel
tigre:~>
```

ATAJOS PARA MOVERSE ENTRE DIRECTORIOS

¿Qué cree usted que pasaría si tuviera que moverse desde / a un directorio dentro de su directorio hogar, por ejemplo */home/usuarios/eslared97/angel/documentos*?. Sería muy tedioso hacerlo con una ruta absoluta. Podemos utilizar el siguiente atajo *(cd ~)*

```
leon:/> cd ~angel/documentos
leon:~> pwd
/home/usuarios/eslared97/angel/documentos
leon:~>
```

Pero ¿y qué pasa cuando yo debo moverme entre directorios con muchos niveles de profundidad, y ninguno de ellos mi directorio hogar?

Hay varias formas de "saltar" entre directorios

Con `cd -` podemos regresar al nivel directorio anterior sin mucho trabajo

```
leon:~> cd /etc
leon:/etc> cd /var/spool/mail
leon:/var/spool/mail> cd -
leon:/etc>
```

Los siguientes comandos son muy útiles cuando la cantidad de saltos es mayor que uno (para ello utilizamos la pila de directorios)

```
leon:~> cd /var/spool/mail
```

Guardamos el directorio actual en la pila de directorios

```
leon:/var/spool/mail> pushd .
/var/spool/mail /var/spool/mail
```

Nos vamos a otro directorio y los guardamos también en la pila

```
leon:/var/spool/mail> cd /etc
leon:/etc> pushd .
/etc /etc /var/spool/mail
leon:/etc> cd /var/adm/
leon:/var/adm> pushd .
/var/adm /var/adm /etc /var/spool/mail
leon:/var/adm> cd /mnt
```

Vamos a consultar la pila para ver que tiene (el tope de la pila se ve de izquierda a derecha)

```
leon:/mnt> dirs
/mnt /var/adm /etc /var/spool/mail
```

Ahora regresamos a cada uno de los directorios visitados con anterioridad usando la pila de directorios

```
leon:/mnt> popd
/var/adm /etc /var/spool/mail
leon:/var/adm> popd
/etc /var/spool/mail
leon:/etc> popd
/var/spool/mail
leon:/var/spool/mail>
```

CREANDO Y ELIMINANDO DIRECTORIOS

La forma de crear directorios es con *mkdir directorio*
Los siguientes ejemplos ilustran un poco mejor el uso de mkdir (usando una ruta relativa, una ruta absoluta y un atajo)

```
leon:~> mkdir documentos
leon:~> mkdir /home/angel/documentos2
leon:~> mkdir ~angel/documentos3
leon:~> mkdir prueba
leon:~>
```

Para eliminar un directorio utilizamos *rmdir directorio*. *El directorio debe estar vacio*.

```
leon:~> rmdir documentos
leon:~> rmdir /home/foto/documentos2
leon:~> rmdir ~foto/documentos3
leon:~>
```

Existe una forma de eliminar subdirectorios vacíos (incluyendo al directorio principal) con rmdir

```
leon:~> mkdir prueba1
leon:~> cd prueba1/
leon:~/prueba> mkdir prueba2
```

Con esto hacemos el directorio ~angel/prueba/prueba2. Ahora queremos eliminar a prueba2 junto con su directorio padre prueba

```
leon:~/prueba> cd
leon:~> rmdir -p prueba1/prueba2/
```

VIENDO LOS ATRIBUTOS DE LOS ARCHIVOS

El comando *ls* le dice al usuario qué archivos (y directorios) están en un directorio. Después del comando *cd*, l*s* es quizás el comando UNIX más utilizado.

Con anterioridad vimos que la sintaxis del comando *ls* es

ls archivo

Sin embargo ls puede hacer cosas más interesantes que sólo mostrar el contenido de un directorio. Por ejemplo, si queremos ver a todos los archivos (incluyendo a los archivos ocultos) escribiremos algo así

```
leon:~> ls -a
./           .ctwmrc      .history     .lessrc      foto/
prueba/
../          .fvwm2rc95   .less        .steprc      passwd.orig
leon:~>
```

En esta salida por pantalla tenemos que passwd.orig es un archivo normal mientras que .steprc es un archivo oculto (es decir, que no puede ser visto con la invocación de ls sin argumentosl). La utilidad de los archivos ocultos es que son un poco más difíciles de eliminar o modificar inadvertidamente.

Fíjese en la diferencia al hacer solamente ls en este directorio

```
leon:~> ls
foto/        passwd.orig  prueba/
leon:~>
```

También podríamos estar interesados en ver el tamaño de cada archivo en bloques de 1Kb (tamaño por omisión):

```
leon:~> ls -s
total 3
   1 foto/           1 passwd.orig     1 prueba/
leon:~>
```

Vemos que el archivo passwd.orig tiene un tamaño de 1Kb.

Sin embargo la opción más interesante de ls es la siguiente:

```
leon:~> ls -l
total 3
drwxr-xr-x  10 angel     users        1024 Sep 27 17:22 foto/
-rw-r--r--   1 angel     users         652 Sep 27 17:24
passwd.orig
drwx------   2 angel     users        1024 Jun 26 10:22 prueba/
```

¿Qué es toda esta información? La información (que se lee de izquierda a derecha) indica la permisología del archivo, número de enlaces, el propietario y grupo del archivo, su tamaño en bytes, su última fecha de modificación y finalmente en nombre del archivo (o directorio). Veamos lo que significa cada uno de estos puntos con más detalle:

- Permisos: es lo que puede hacer con el archivo.
- Enlaces: son la cantidad de puntos permitidos que pueden ocupar en el sistema.
- Propietario: es quien creó el archivo.
- Tamaño: Tamaño del archivo en bytes (más preciso que ls -s).
- Tiempo de modificación: cuando el archivo fue creado, editado o modificado.
- Nombre: el nombre del archivo o directorio.

Como lo habrá intuido ya, el usuario puede combinar varias opciones para obtener información más precisa acerca de los archivos que está examinando

```
leon:~> ls -las
total 75
   2 drwx------   4 foto     users        2048 Sep 27
17:24 ./
   1 drwxr-xr-x   5 root     root         1024 Jul  9
19:34 ../
  14 -rw-------   1 foto     users       13228 Feb 19
1997 .ctwmrc
  28 -rw-------   1 foto     users       26779 Mar  8
1997 .fvwm2rc95
   1 -rw-------   1 foto     users         494 Sep 27
13:43 .history
   1 -rw-------   1 foto     users          34 Nov 24
1993 .less
   1 -rw-------   1 foto     users         114 Nov 24
1993 .lessrc
  24 -rw-------   1 foto     users       23270 Jul 24
13:05 .steprc
   1 drwxr-xr-x  10 foto     users        1024 Sep 27
17:22 foto/
```

```
   1 -rw-r--r--   1 foto     users         652 Sep 27
17:24 passwd.orig
   1 drwx------   2 foto     users        1024 Jun 26
10:22 prueba/
leon:~>
```

Más adelante profundizaremos el concepto de permisología y propiedad de archivo.

DOS ATAJOS ÚTILES: * Y ?

UNIX le permite a usted trabajar con más de un archivo a la vez al proveer dos *metacaracteres*, llamados así porque ellos se refieren a cualquier carácter, o grupo de caracteres, en el nombre de un archivo. De esta manera usted puede operar en un número de archivos cuyos nombres coincidan el patrón que usted provee.
Por ejemplo, queremos listar todos los archivos que comiencen por h en /etc

```
leon:~> ls /etc/h*
/etc/host.conf    /etc/hosts.allow  /etc/hosts.equiv
/etc/hosts        /etc/hosts.deny   /etc/hosts.lpd
leon:~>
```

Quiero listar todos los archivos que terminen en .conf

```
leon:~> ls /etc/*.conf
/etc/host.conf    /etc/ld.so.conf   /etc/mtools.conf
/etc/syslog.conf
/etc/inetd.conf   /etc/lilo.conf    /etc/resolv.conf
/etc/yp.conf
leon:~>
```

Como puede ver, él "*" representa cualquier número de caracteres en el nombre de un archivo. Un asterisco por si mismo representa a todos los archivos en un directorio excepto los archivos ocultos. En cambio el carácter "?" representa un solo carácter en el nombre de un archivo.

Por ejemplo, si quiero listar todos los archivos en /etc que comiencen por h y tengan 5 letras

```
leon:~> ls /etc/h????
/etc/hosts
leon:~>
```

Puedo mezclar ambas opciones. La siguiente salida me muestra todos los archivos que tengan una extensión de 4 letras

```
leon:~> ls /etc/*.????
/etc/at.deny          /etc/ld.so.conf       /etc/named.boot.orig
/etc/host.conf        /etc/lilo.conf        /etc/resolv.conf
/etc/hosts.deny       /etc/mtools.conf      /etc/syslog.conf
/etc/inetd.conf       /etc/named.boot       /etc/yp.conf
leon:~>
```

Uso adicional de los caractéres especiales:

Los caractéres especiales que acaba de aprender pueden ser utilizados también para mover, copiar y borrar archivos (es decir, cualquier operación de manipulación de archivos puede utilizar estos caractéres).

VIENDO EL CONTENIDO DE UN ARCHIVO DE TEXTO

El comando *more* muestra el contenido de un archivo una pantalla a la vez; si el archivo ocupa más de una pantalla mostrará la palabra

```
-more-
```

seguido de el porcentaje mostrado del archivo hasta ahora, en el fondo de la pantalla.

Presione la barra espaciadora para ver la siguiente pantalla del archivo, presione *h* para obtener ayuda, presione *q* para salir.

Supongamos que queremos ver el contenido del archivo /etc/inetd.conf

```
leon:/etc> more inetd.conf
# See "man 8 inetd" for more information.
#
# If you make changes to this file, either reboot your machine
or send the
# inetd a HUP signal:
# Do a "ps x" as root and look up the pid of inetd. Then do a
# "kill -HUP <pid of inetd>".
# The inetd will re-read this file whenever it gets that signal.
#
# <service_name> <sock_type> <proto> <flags> <user>
<server_path> <args>
#
echo    stream  tcp     nowait  root    internal
echo    dgram   udp     wait    root    internal
discard stream  tcp     nowait  root    internal
discard dgram   udp     wait    root    internal
daytime stream  tcp     nowait  root    internal
daytime dgram   udp     wait    root    internal
chargen stream  tcp     nowait  root    internal
chargen dgram   udp     wait    root    internal
time    stream  tcp     nowait  root    internal
time    dgram   udp     wait    root    internal
#
# These are standard services.
--More--(17%)
```

También puede buscar cadenas de texto dentro del archivo. Escriba / seguido de la cadena de texto buscada y pulse *enter*. Busquemos la cadena *rlogin* en el archivo.

```
# These are standard services.
/rlogin
...skipping
#
shell   stream  tcp     nowait  root    /usr/sbin/tcpd  in.rshd
-L
login   stream  tcp     nowait  root    /usr/sbin/tcpd
in.rlogind
# exec  stream  tcp     nowait  root    /usr/sbin/tcpd
in.rexecd
```

```
# talk  dgram   udp     wait    root    /usr/sbin/tcpd  in.talkd
#ntalk  dgram   udp     wait    root    /usr/sbin/tcpd  in.talkd
#
# Kerberos authenticated services
#
# klogin        stream  tcp     nowait  root    /usr/sbin/tcpd
rlogind -k
# eklogin       stream  tcp     nowait  root    /usr/sbin/tcpd
rlogind -k -x
# kshell        stream  tcp     nowait  root    /usr/sbin/tcpd
rshd -k
#
# Services run ONLY on the Kerberos server
#
# krbupdate     stream  tcp     nowait  root    /usr/sbin/tcpd
registerd
# kpasswd       stream  tcp     nowait  root    /usr/sbin/tcpd
kpasswdd
#
# Pop et al
#
# pop2  stream  tcp     nowait  root    /usr/sbin/tcpd  in.pop2d
--More--(57%)
```

Como puede ver, more busca la primera ocurrencia y se detiene.

Otra forma de ver el contenido de todo el archivo de una sola vez es con el comando *cat*

```
leon:/etc> cat /etc/passwd
root:XkWExN0RX5KaM:0:0:root:/root:/bin/bash
bin:*:1:1:bin:/bin:
daemon:*:2:2:daemon:/sbin:
adm:*:3:4:adm:/var/adm:
lp:*:4:7:lp:/var/spool/lpd:
sync:*:5:0:sync:/sbin:/bin/sync
shutdown:*:6:0:shutdown:/sbin:/sbin/shutdown
halt:*:7:0:halt:/sbin:/sbin/halt
mail:*:8:12:mail:/var/spool/mail:
news:*:9:13:news:/usr/lib/news:
uucp:*:10:14:uucp:/var/spool/uucppublic:
operator:*:11:0:operator:/root:/bin/bash
games:*:12:100:games:/usr/games:
man:*:13:15:man:/usr/man:
postmaster:*:14:12:postmaster:/var/spool/mail:/bin/bash
nobody:*:65534:100:nobody:/dev/null:
ftp:*:404:1::/home/ftp:/bin/bash
guest:*:405:100:guest:/dev/null:/dev/null
+foto::::::
+jose::::::
leon:/etc>
```

Cat muestra el archivo de una sola vez, no se detiene si el archivo ocupa más de una pantalla.

Pero ¿y que pasa si estoy interesado en ver el final del archivo solamente?

```
leon:/etc> tail /etc/inetd.conf
```

```
netbios-ns      dgram   udp     wait    root    /usr/sbin/nmbd
nmbd
#
# Sun-RPC based services.
# <service
name/version><sock_type><rpc/prot><flags><user><server><args>
#
# rstatd/1-3    dgram   rpc/udp wait    root    /usr/sbin/tcpd
rpc.rstatd
# rusersd/2-3   dgram   rpc/udp wait    root    /usr/sbin/tcpd
rpc.rusersd
# walld/1               dgram   rpc/udp wait    root
/usr/sbin/tcpd  rpc.rwal
ld
#
# End of inetd.conf.
leon:/etc>
```

Tail por omisión me muestra solamente las 10 últimas líneas al menos que le indique lo contrario.

COPIANDO, MOVIENDO, CAMBIANDO EL NOMBRE A LOS ARCHIVOS

Para copiar archivos en la misma maquina, use el comando *cp*. La sintaxis más sencilla de *cp* es la siguiente

cp archivo-fuente archivo-destino

Saquemos una copia del /etc/passwd y dejémosla en nuestro directorio hogar

```
leon:~> cp /etc/passwd ~angel
leon:~>
```

Supongamos que ahora quiero una copia de *todo* el directorio /etc en mi directorio hogar

```
leon:~> cp -r /etc ~angel
cp: /etc/inet: cannot copy cyclic symbolic link
cp: /etc/syslog.conf: Permission denied
cp: /etc/rmt: Permission denied
cp: /etc/psdevtab: Permission denied
leon:~> ls
etc/ passwd prueba/
leon:~>
```

El comando `cp -r` hace una copia recursiva que copia un directorio con todos subdirectorios en otro directorio (en el ejemplo es /home/angel) No se preocupe por los mensajes que dicen permiso denegado (significan que usted no tiene permiso para copiar esos archivos. Se darán más detalles en la práctica).

Vamos a cambiarle el nombre de /home/angel/passwd a /home/angel/passwd.orig

```
leon:~> mv passwd passwd.orig
leon:~> ls
etc/ passwd.orig  prueba/
leon:~>
```

mv sirve para cambiarle el nombre un archivo, así como para mover uno o más archivos de un directorio a otro. Vamos a mover todo el directorio ~angel/etc dentro de ~angel/prueba

```
leon:~> mv etc/ prueba/
leon:~> cd prueba
leon:~/prueba> ls
etc/
leon:~/prueba>
```

Ahora que terminamos de mover los archivos, queremos eliminarlos para desocupar espacio dentro de nuestro disco duro. Eliminemos primero a ~angel/passwd.orig con *rm*

```
leon:~/prueba> cd
leon:~> ls
foto/        passwd.orig  prueba/
leon:~> rm passwd.orig
leon:~> ls
foto/    prueba/
leon:~>
```

¿Recuerda que *rmdir* sólo podía eliminar directorios vacios? Con *rm -rf directorio* usted puede eliminar el contenido de un directorio, incluyendo el directorio. Eliminemos ~angel/prueba

```
leon:~> rm -rf prueba/
leon:~> ls
```

NOTA SOBRE EL BORRADO DE ARCHIVOS:

Borrar archivos es una operación muy delicada. Los archivos eliminados bajo UNIX no pueden ser recuperados. Asegúrese que el archivo que va a eliminar es realmente el que va a eliminar. Se recomienda, mientras aprende a utilizar el comando rm, que lo ejecute con la opción -i (modo interactivo) para que este le pregunte si está seguro o no si desea eliminar el archivo (la sintaxis completa es rm -ir archivo).

CREANDO ARCHIVOS

Existen varias formas de crear archivos.

- Copiando o moviendo archivos (usted ya sabe como hacer eso)
- La salida de un programa. Por ejemplo el comando gcc produce programas ejecutables hechos en C.
- Usando el comando *touch*. Touch crea archivos vacios o actualiza la fecha de los archivos existentes. La sintaxis es *touch archivo*.
- Usando el comando cat. Cuando usted quiere crear un archivo y no quiere usar un editor puede hacer lo siguiente

  ```
  leon:~> cat > prueba
  ```

  Comience a escribir cualquier texto

  ```
  Esta es una prueba del uso de cat como un editor
  simple. Espero que sea util.
  ```

Ahora presione simultáneamente las teclas control + D (^D) para finalizar la edición
Veamos si en realidad creamos un archivo.

```
leon:~> ls
foto/   prueba
leon:~> cat prueba
Esta es una prueba del uso de cat como un editor
simple. Espero que sea util.
leon:~>
```

PERMISOLOGÍA Y PROPIEDAD DE ARCHIVOS

PERMISOLOGÍA DE LOS ARCHIVOS

Los permisos determinan qué usuarios puede hacer o no cosas con un archivo o directorio. Los siguientes son los permisos asociados a los archivos y directorios:

- Lectura: Un archivo debe tener permiso de lectura para ser examinado o copiado. Un directorio debe tener permiso de lectura para poder listar sus contenidos.
- Escritura: Un archivo debe tener permiso de escritura para poder ser modificado, eliminado o renombrado. Un directorio debe tener permiso de escritura para que puedan borrarse archivos dentro de él.
- Ejecución: Un archivo ejecutable es aquel que un usuario puede correr, como un programa o un *shell script*. Si es un directorio significa que se puede entrar dentro de él con cd.

Examinemos en detalle el siguiente archivo

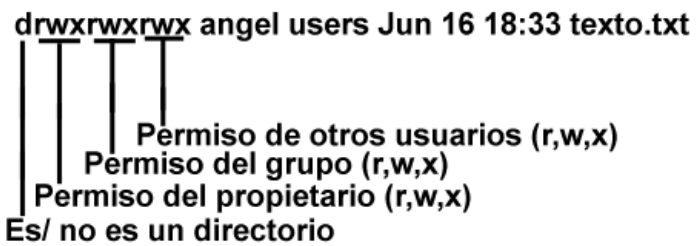

Ilustración 2: Tipos de permisos presentes en un archivo bajo UNIX

Aqui cada letra significa lo siguiente:

- **d** (**d**irectory). El archivo es o no un directorio
- **r** (**r**ead). El archivo puede leerse
- **w** (**w**rite). El archivo puede modificarse, escribirse.
- **x** (e**x**ecutable). El archivo puede ejecutarse

Veamos que significa cada uno de los renglones de la permisología

- Permiso del propietario: Estas 3 letras dicen qué permisos tiene el propietario con respecto a un archivo o directorio de su propiedad. Esto es, que puede leerlo, escribirlo, ejecutarlo (archivo) o buscar en él (directorio).
- Permisos de grupo: Permiten compartir un archivo con un grupo de usuarios. Estas 3 letras dicen que permisos tienen los archivos para los usuarios que pertenecen al mismo grupo del archivo.
- Permisos para otros usuarios: Estas 3 letras dicen qué permisos tienen otros usuarios con respecto al archivo o directorio. Útil para resguardar la privacidad del archivo.

NOTA acerca de la permisología de los archivos:

Nunca deje un archivo con permisología para ser escrito por todo el mundo. Eso significa que cualquier persona dentro del sistema puede alterar el contenido del archivo o destruir el archivo mismo.

```
leon:~> ls -l
total 2
drwxr-xr-x  10 angel     users        1024 Sep 27 17:22 foto/
-rw-r--r--   1 angel     users          78 Sep 29 09:13 prueba
leon:~>
```

De la salida por pantalla vemos que foto es un directorio, que tiene permiso de lectura,escritura y ejecución por angel, lectura y ejecución para el grupo users, y lectura y ejecución para otros. Vemos también como prueba es un archivo que tiene permiso de lectura y escritura y sólo lectura para el grupo y otros.

Con la herramienta *chmod* podemos cambiar la permisología de un archivo. La sintaxis es la siguiente:

```
chmod [cadena(s) de acceso archivo(s)]
```

Donde la cadena de acceso puede ser construida de esta manera

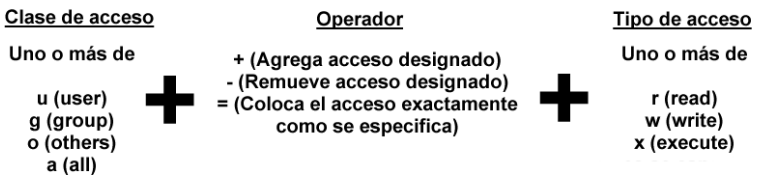

Ilustración 3: Cadena de acceso para modificar la permisología de un archivo

Ilustremos mejor el uso de *chmod* con unos ejemplos.

```
leon:~> ls -l
total 2
drwxr-xr-x  10 angel     users        1024 Sep 27 17:22 foto/
-rw-r--r--   1 angel     users          78 Sep 29 09:13 prueba
leon:~>
```

Vemos que el archivo prueba tiene permiso de lectura para todo el mundo. Vamos a colocarle permiso de lectura sólo para el usuario, ningún permiso para grupo y otros

```
leon:~> chmod u=r,og=- prueba
leon:~> ls -l
total 2
drwxr-xr-x  10 angel     users        1024 Sep 27 17:22 foto/
-r--------   1 angel     users          78 Sep 29 09:13 prueba
leon:~>
```

Supongamos que nos arrepentimos y decidimos darle permiso de lectura y escritura al grupo

```
leon:~> chmod g+wr prueba
leon:~> ls -l
total 2
drwxr-xr-x  10 angel     users        1024 Sep 27 17:22 foto/
-r--rw----   1 angel     users          78 Sep 29 09:13 prueba
leon:~>
```

Finalmente queremos quitarle el permiso de lectura y ejecución a el directorio foto y a todos los archivos contenidos dentro de el (incluyendo otros directorios)

```
leon:~> chmod -R go-xr foto/
leon:~> ls -l
total 2
drwx------  10 angel     users        1024 Sep 27 17:22 foto/
-r--rw----   1 angel     users          78 Sep 29 09:13 prueba
leon:~>
```

También podemos especificar los permisos de un archivo utilizando una forma numérica.
El siguiente archivo tiene la siguiente conversión:

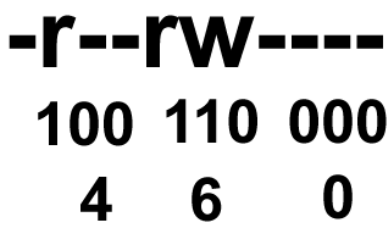

Ilustración 4: Equivalencia en números binarios de la permisología de un archivo

Es decir, cada letra correspondiente a una parte del permiso (r,w,x) puede estar activada o no (en 1 ó 0) para formar un número binario, el cual es convertido luego a número decimal.

Por ejemplo para ponerle permiso de lectura para el usuario, lectura y escritura para el grupo y ningún permiso a otros debemos escribir

```
leon:~> chmod 460 prueba
leon:~> ls -l
total 2
drwx------  10 angel     users        1024 Sep 27 17:22 foto/
```

```
-r--rw----   1 angel     users          78 Sep 29 09:13 prueba
leon:~>
```

El concepto de propiedad de un archivo bajo UNIX es más complejo. Un archivo puede ser propiedad de un usuario y de un grupo (y ambos tipos de propiedad sobre el archivo son independientes). Un usuario normal no puede cambiar la propiedad de un archivo bajo UNIX (sólo root puede hacerlo), pero si puede cambiar el grupo de un archivo si el usuario pertenece al grupo al cual va a cambiar el archivo.

Toda esta compleja estructura (propiedad de archivos y permisología) tiene su motivación en permitir que las protecciones de archivos y permisos sean organizadas de acuerdo a las necesidades locales.

Veamos cómo cambiar el grupo a un archivo suponiendo que el administrador nos ha incluido en otro grupo adicional

Vamos a crear un archivo y veamos su grupo (en este caso es users)

```
leon:~> touch miarchivo
leon:~> ls -l
total 1
drwx------  10 angel     users        1024 Sep 27 17:22 foto/
-rw-r--r--   1 angel     users           0 Sep 30 07:29
miarchivo
```

Nosotros sabemos que pertenecemos a el grupo prueba (además del grupo users), por lo que cambiamos nuestro grupo activo (un usuario solo puede tener un grupo activo a la vez)

```
leon:~> newgrp prueba
```

Una vez cambiado el grupo activo, cambiamos el grupo del archivo y vemos los resultados.

```
> chgrp prueba miarchivo
> exit
leon:~> ls -l
total 1
drwx------  10 foto     users        1024 Sep 27 17:22 foto/
-rw-r--r--   1 foto     prueba          0 Sep 30 07:29 miarchivo
leon:~>
```

Como el comando chmod, chgrp soporta la opción -R (recursiva).

PROPIEDAD DE LOS ARCHIVOS

Los archivos bajo UNIX son propiedad de un usuario (comúnmente del que lo creó) y de un grupo de usuarios. El manejo de esos dos usuarios es independiente, lo que permite controlar y compartir recursos de manera efectiva.

Para cambiar el dueño de un archivo se utiliza el comando *chown* (en Linux este comando sólo puede ser ejecutado por el root, siguiendo la tradición de la familia de UNIX BSD).

La sintaxis es la siguiente:

```
chown [propietario archivo (s)]
```

Por ejemplo, si usted fuera root en nuestra maquina ficticia *león* podría hacer esto:

```
leon:/home/angel# ls -l
total 1
drwx------  10 angel     users        1024 Sep 27 17:22 foto/
-rw-r--r--   1 angel     users           0 Oct  2 22:35 prueba
leon:/home/foto# chown jose prueba
leon:/home/foto# ls -l
total 1
drwx------  10 angel     users        1024 Sep 27 17:22 foto/
-rw-r--r--   1 jose      users           0 Oct  2 22:35 prueba
leon:/home/foto#
```

De la misma manera podemos cambiar el grupo con *chgrp*. A diferencia de chown, chgrp sí puede ser ejecutado por un usuario normal pero para que tenga efecto el usuario debe pertenecer al grupo del archivo que intenta cambiar y debe poder cambiarse de grupo activo (es decir, el usuario debe pertenecer a por lo menos dos grupos). Para demostrar esto (suponiendo que angel pertenece a los grupos users y prueba), creemos un archivo llamado prueba y veamos a qué grupo pertenece:

```
leon:~> touch archivo
leon:~> ls -l
total 1
-rw-r--r--   1 angel     users           0 Oct  2 22:40 archivo
drwx------  10 angel     users        1024 Sep 27 17:22 foto/
```

Cambiamos ahora nuestro grupo activo (es decir cambiamos de *users* a *prueba*)

```
leon:~> newgrp prueba
```

Luego le cambiamos el grupo al archivo

```
> chgrp prueba archivo
```

Finalmente nos salimos y vemos los resultados

```
> exit
exit
leon:~> ls -lg
total 1
-rw-r--r--   1 angel     prueba          0 Oct  2 22:40 archivo
drwx------  10 angel     users        1024 Sep 27 17:22 foto/
leon:~>
```

NOTA sobre el cambio de grupos y usuarios:

Bajo Linux y las variantes de UNIX bajo BSD se puede cambiar el grupo y el usuario al mismo tiempo con un solo comando. La sintaxis es chmod propietario.grupo archivo(s).

ENLACES

Un enlace es un mecanismo que permite que varios nombres archivos se refieran a un solo archivo en disco (es decir, es un nombre asociado a un archivo). Cuando usted crea un archivo, el sistema hace el primer enlace por usted.

Hay dos tipos de enlaces:

- Enlaces duros: Asocian uno o más nombres a una misma localidad física (inodo).
- Enlaces simbólicos: Es un archivo que apunta a otra ruta en el sistema de archivos. Los enlaces simbólicos pueden apuntar a otros dispositivos físicos debido a que apuntan aun nombre de archivo, no a una localidad física de disco (inodo).

Veamos esto con el siguiente ejemplo

```
leon:~> cat > prueba.txt
Esta es una prueba de lo que puede pasar
al enlazar un archivo utilizando ln -s y ln -h
(^D)
leon:~> ls -s
total 2
   1 foto/          1 prueba.txt
leon:~>
```

Ahora vamos a hacer un enlace simbólico y un enlace duro hacia prueba.txt

Primero hagamos el enlace duro

```
leon:~> ln prueba.txt duro
```

Y luego el enlace suave

```
leon:~> ln -s prueba.txt suave
```

Veamos los resultados

```
leon:~> ls -l
total 3
-rw-r--r--   2 angel users          89 Oct  3 11:38
duro
drwx------  10 angel users        1024 Sep 27 17:22
foto/
-rw-r--r--   2 angel users          89 Oct  3 11:38
prueba.txt
lrwxrwxrwx   1 angel users          10 Oct  3 21:46
suave -> prueba.txt
leon:~>
```

Como puede ver en el ejemplo, *suave* es un enlace simbólico (su permisología comienza con la letra l), mientras que el enlace llamado *duro* se ve como un archivo (note como el número de enlaces es igual a 2).

Podemos hacer una prueba adicional para ver cómo los enlaces duros comparten los mismos inodos, mientras que los enlaces suaves no:

```
leon:~> ls -i
  10718 duro           8524 foto/         10718
prueba.txt    10719 suave@
leon:~>
```

Como puede ver duro y prueba.txt comparten el inodo 10718.

Si borramos a prueba.txt, el enlace llamado "duro" todavía funcionará (comparten el mismo inodo), pero el enlace llamado "suave" estará vacio porque esta apuntando al nombre prueba.txt, no a un inodo.

EDICIÓN DE ARCHIVOS CON EL EDITOR VI

Vi es el editor de texto más difundido en el ambiente UNIX. Vi significa "Visual Display Editor" (editor visual de pantalla). Debido a que vi no está basado en ventanas, puede ser utilizado en cualquier tipo de terminal.

Usted puede editar e introducir texto con vi, pero éste no es un procesador de palabras: usted le da formato al texto con vi utilizando códigos que entonces son interpretados por otro programa (como *nroff* o *troff*).

Vi tiene 3 modos de trabajo:

- Modo de comandos
- Modo de última línea (o modo de dos puntos)
- Modo de entrada

Vamos a editar un archivo que ya esta creado y con ésto ejercitaremos un poco. Copiemos el archivo que muestra el mensaje del dia (/etc/motd) a nuestro directorio hogar.

```
leon:~> cp /etc/motd ~angel
```

Ahora vamos a abrirlo con vi

```
leon:~> vi motd
```

Lo que veremos por pantalla será algo parecido a esto

```
Linux 2.0.27.
```

En este momento estamos en el *modo de comandos*. Vamos a agregar texto debajo de la primera línea escribiendo "o" y luego el texto (la o no se ve por pantalla porque usted ha pasado *al modo de entrada*). Usted puede escribir normalmente, pulsando enter para continuar a la siguiente línea.

Una vez concluida la edición, presione escape (ESC) para volver al modo de comandos

```
Linux 2.0.27.
Bienvenidos a mi maquina. Este mensaje aparecera cada vez
que un usuario entre a ella, una vez que lo vuelva a colocar en
/etc/motd. Un superusuario (root) podria ayudarme con eso.

Kawuabunga!!!!
```

Ahora es tiempo de guardar nuestro trabajo. Escriba **:w** (write)

```
Linux 2.0.27.
Bienvenidos a mi maquina. Este mensaje aparecera cada vez
que un usuario entre a ella, una vez que lo vuelva a colocar en
/etc/motd. Un superusuario (root) podria ayudarme con eso.

Kawuabunga!!!!

~
```

```
:w
```

Luego presione **enter** y verá algo parecido al fondo de la pantalla:

```
Linux 2.0.27.
Bienvenidos a mi maquina. Este mensaje aparecera cada vez
que un usuario entre a ella, una vez que lo vuelva a colocar en
/etc/motd. Un superusuario (root) podria ayudarme con eso.

Kawuabunga!!!!
~
~
~
 Wrote "motd"  6 lines, 211 characters
```

El modo anterior fue un *modo de última línea.*

Una vez guardado nuestro trabajo inicial, podremos hacer más modificaciones. Para ello será muy útil saber el número de línea en el que estamos trabajando, por lo que debe escribir **:set number** (el número de línea no es agregado al archivo, sólo se ve por pantalla. Para quitar el número escriba **:set no number**)

```
Linux 2.0.27.
Bienvenidos a mi maquina. Este mensaje aparecera cada vez
que un usuario entre a ella, una vez que lo vuelva a colocar en
/etc/motd. Un superusuario (root) podria ayudarme con eso.

Kawuabunga!!!!
~
:set number
```

Luego presione **enter**:

```
1  Linux 2.0.27.
2  Bienvenidos a mi maquina. Este mensaje aparecera cada vez
3  que un usuario entre a ella, una vez que lo vuelva a colocar
en
4  /etc/motd. Un superusuario (root) podria ayudarme con eso.
5
6  Kawuabunga!!!!
~
```

Podemos viajar hacia cualquier línea del documento con sólo escribir :#*número de línea.* Supongamos que queremos introducir texto delante de Kawuabunga en la línea 6:

```
1  Linux 2.0.27.
2  Bienvenidos a mi maquina. Este mensaje aparecera cada vez
3  que un usuario entre a ella, una vez que lo vuelva a colocar
en
4  /etc/motd. Un superusuario (root) podria ayudarme con eso.
5
6  Kawuabunga!!!!
~
:6 <ENTER>
```

Y luego presione la letra *i* (de insert). Cuando termine de escribir pulse *ESC.*

```
1  Linux 2.0.27.
2  Bienvenidos a mi maquina. Este mensaje aparecera cada vez
3  que un usuario entre a ella, una vez que lo vuelva a colocar
en
4  /etc/motd. Un superusuario (root) podria ayudarme con eso.
5
        6  Esto es Kawuabunga!!!!
```

NOTA sobre los comandos de vi:

Los comandos en vi diferencian entre mayúsculas y minúsculas. El mismo comando escrito en mayúsculas tiene un comportamiento distinto a si es escrito en minúsculas. Por ejemplo el comando i (insert) lo hace después de donde esta ubicado el cursor, mientras que el comando I lo hace al principio de la línea.

Para salirse de vi escriba **:q**

Usted puede moverse por las líneas del texto utilizando las teclas del cursor. Sin embargo, si esta utilizando un terminal remoto las teclas pudiesen no trabajar correctamente (dependiendo de la emulación del terminal) por lo que es bueno que aprenda las teclas *substitutas* usadas en vi

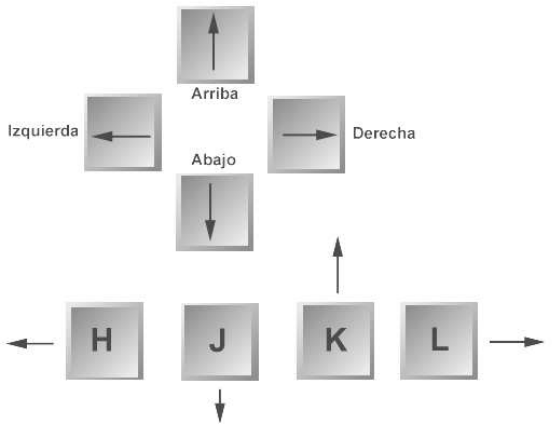

Ilustración 5: Teclas substitutas de movimiento para Vi

La siguiente tabla contiene una serie de comandos útiles, así como su explicación:

Moverse una palabra: w, b	w (word) mueve el cursor a la derecha de una palabra una palabra a la vez, b (back) mueve el cursor a la izquierda una palabra.
Moverse a los extremos de la pantalla: H, M ,L	H (HIGH) mueve el cursor al tope de la pantalla, M (MIDDLE)lo coloca en la mitad, L (Bottom) lo pone al final de la pantalla.
Recorrer texto: Ctrl - {f, d, b, u}	Puede recorrer el archivo por pantalla una pantalla a la vez, media pantalla, adelante o hacia atrás. Es mucho mejor que tratar de recorrer un archivo largo utilizando sólo las teclas de cursor.
Agregar texto: a, A	Inserta texto a la derecha del cursor, inserta texto al final de la línea. Presione ESC

	cuando termine.
Insertar texto: i, I	Inserta texto a la izquierda del cursor, inserta texto al comienzo de la línea. Presione ESC cuando termine.
Abrir una línea: o, O	Abre una línea debajo de la posición actual, abre una línea arriba de la posición actual. Presione ESC cuando termine.
Cambiar palabra: cw	Para reemplazar una palabra, posicione el cursor al principio de la palabra y escriba cw seguida de la nueva palabra. Presione ESC cuando termine.
Cambiar línea: cc	Para reemplazar una línea, posicione el cursor en cualquier parte de la línea. Esto la hará desaparecer. Presione ESC cuando termine.
Cambiar parte de la línea: C	Permite reemplazar parte de la línea que está a la derecha del cursor. Presione ESC cuando termine.
Substituir caracteres: s	Para sustituir uno o más caracteres en donde está ubicado el cursor, escriba s seguido del nuevo texto.
Reemplazar caracteres: r	Este comando permite reemplazar un solo carácter bajo el cursor por otro carácter. Coloque el cursor sobre el carácter que desea reemplazar y escriba r.
Deshacer comandos: u, U	Con u deshace el último cambio hecho, con U deshace todos los cambios hechos
Romper o unir líneas: r+enter, J	Para romper una línea vaya al punto de quiebre y escriba r + enter. Para unir una línea con la línea que está debajo escriba J en el punto de unión.
Borrar un carácter, una palabra, una línea y parte de una línea: x, dw, dd, D	Para borrar un carácter posicione el cursor sobre el carácter que va a ser borrado y escriba x. Para borrar una palabra posicione el cursor al principio de la palabra y escriba dw (se borrará la parte derecha de la palabra). Para borrar una línea, coloque el cursor en cualquier parte de la linea y escriba dd. Finalmente, para borrar parte de una línea coloque el cursor a la izquierda de la parte de la línea que desea eliminar y escriba la letra D.
Factores de repetición	Muchos comandos en vi pueden ser precedidos por un factor de repetición, el cual es un número que le dice a vi cuantas veces debe repetir un comando dado. Por ejemplo el comando 8dd borrará 8 líneas mientras que el comando 5x borrará 5 caracteres.
Repetir el comando anterior: .	Escribiendo punto (.) repite el comando de cambio de texto anterior.
Copiando, moviendo líneas: yy, dd	Para copiar (mover) una línea de texto coloque el cursor en cualquier parte de la línea a copiar (mover) y escriba yy (dd). Luego coloque el cursor en el sitio en donde quiere pegar la línea y escriba p (pegar

Copiar, mover, borrar líneas con comandos de última línea	abajo) o P (pegar arriba) para colocarla. Para copiar una línea escriba : #linea-inicio, #linea-final co #linea. Los primeros dos números son el rango desde donde se va a hacer la copia, el tercer número es desde donde van a ser pegadas las nuevas líneas. Por ejemplo si queremos copiar desde la línea 10 hasta la línea 25 en la línea 40 escribiríamos : 10,25 co 40 La sintaxis para mover líneas es : #linea-inicio,#linea-final m #linea Y finalmente la sintaxis para eliminar un grupo de líneas es : #linea-inicio, #linea-final

Tabla 1: Comandos usuales del editor Vi

BUSCANDO CARACTERES

Una de las características más notables de vi es su habilidad para encontrar y reemplazar cadenas de caracteres con increíble facilidad. Para encontrar una cadena de caracteres escriba / seguido de la *cadena de caracteres a buscar* y pulse *enter*. Usted puede escribir n (next) para ir a la próxima ocurrencia del carácter, N para ir a la anterior.

Si lo que desea es buscar desde el final hasta el principio entonces escriba ? en vez de /.

Por ejemplo, para buscar la cadena super en nuestro archivo de ejemplo (la cual llevará el cursor a la línea 4) escriba

```
/super <ENTER>
```

¿CÓMO BUSCA CARACTERES EL USUARIO AVANZADO?

Usted puede hacer que sus búsquedas en vi sean más precisas, indicando por ejemplo si quiere buscar al principio o al final de la línea. Veamos algunas búsquedas refinadas.

Para encontrar una cadena de caracteres al comienzo de una línea debe escribir lo siguiente:

```
/^cadena
```

En cambio para encontrar la cadena al final de la línea:

```
/cadena$
```

Para encontrar el comienzo de una palabra escriba \< al comienzo de la cadena; para encontrar el final de una palabra escriba \> al final de la palabra. Es decir, para encontrar una palabra debe escribir:

```
/\<cadena\>
```

Para encontrar cualquier carácter (desconocido o variable) dentro de una cadena de caracteres escriba un punto (.) en la cadena. Por ejemplo, el siguiente comando busca cualquier cosa que tenga la cadena *aquina*:

```
/.aquina
```

Usted puede utilizar rangos de caracteres y caracteres especiales para hacer búsquedas. Por ejemplo, para buscar cualquier cadena que comience desde la "a " hasta la "z", que termine con la cadena "lla" y que encuentre todas las ocurrencias de "lla" usamos:

```
/[a-z]*lla
```

Esto encuentra la cadena *ella* en la línea 3 de nuestro ejemplo.

REEMPLAZANDO CARACTERES

El comando básico para reemplazar caracteres es el siguiente:

```
:g/cadena-de-busqueda/s//cadena a reemplazar/g
```

Supongamos que queremos cambiar la palabra Kawuabunga por caramba en la línea 6 de nuestro texto de ejemplo. Para hacerlo escribiríamos:

```
:g/Kawuabunga/s//caramba/g
```

¿CÓMO SER UN USUARIO AVANZADO?

EJECUCIÓN DE PROCESOS EN SEGUNDO PLANO

Normalmente la concha (shell) espera que un comando se complete antes de pedirle otro comando. Si usted piensa que un comando puede tardar mucho tiempo en ejecutarse puede invocarlo agregándole un & al final para que este corra en segundo plano y usted pueda seguir introduciendo comandos en la concha. Por ejemplo, supongamos que queremos ordenar alfabéticamente un archivo llamado *muy-grande* y sabemos que esta operación puede tardar mucho tiempo:

```
sort muy-grande > ordenado &
```

Usted puede correr varios comandos en segundo plano (sin embargo sea considerado y no coloque muchos procesos en un sistema multiusuario. Recuerde que hay otros que quieren aprovechar también los recursos del sistema):

```
sort muy-grande1 > ordenado1 &
sort muy-grande2 > ordenado2 &
sort muy-grande3 > ordenado3 &
```

ESTADO DEL PROCESO DE UN USUARIO

Unix permite a un usuario ejecutar más de un proceso a la vez. Cada uno de esos procesos tiene un identificador llamado número del proceso. Para ver información acerca de los procesos utilizamos el comando ps:

```
leon:~> ps
  PID TTY STAT  TIME COMMAND
 6366  p0 S     0:01 -tcsh
 6828  p0 R     0:01 ps
```

```
leon:~>
```

El comando ps me está diciendo que el comando tcsh (la concha) tiene el identificador de proceso 6366, está corriendo en el terminal p0, está detenido (**S**leeping) y lleva un minuto ejecutándose.

El comando ps tiene muchas opciones. Una de las más comunes es averiguar todos los procesos que se están ejecutando en la maquina con información del usuario que los corre (esto incluye a todos los usuarios):

```
leon:~> ps -aux
USER       PID %CPU %MEM  SIZE   RSS TTY STAT START   TIME
COMMAND
angel     6366  0.0  9.8  1132   692  p0 S    07:29   0:01 -tcsh
angel     6954  0.0  5.1   824   364  p0 R    14:36   0:01 ps -
aux
angel     6955  0.0  3.8   792   268  p0 S    14:36   0:00 more
bin         52  0.0  4.3   796   304  ?  S   Sep 30   0:15
/usr/sbin/rpc.portmap
root         1  0.0  4.3   780   304  ?  S   Sep 30   0:07 init
[3]
root         2  0.0  0.0     0     0  ?  SW  Sep 30   0:00
(kflushd)
root         3  0.0  0.0     0     0  ?  SW< Sep 30   0:00
(kswapd)
root         4  0.0  0.0     0     0  ?  SW  Sep 30   0:00
(nfsiod)
root         5  0.0  0.0     0     0  ?  SW  Sep 30   0:00
(nfsiod)
root         6  0.0  0.0     0     0  ?  SW  Sep 30
```

Podemos ver que se están ejecutando procesos del usuario angel y del usuario root.

TERMINACIÓN DE PROCESOS EN RESERVA

Para matar (detener) procesos en reserva utilizamos la orden **kill # proceso**.

Veamos su uso con el siguiente ejemplo.

```
leon:~> ftp localhost &
[1] 6957
leon:~> Connected to localhost.
220 leon FTP server (Version wu-2.4(1) Tue Dec 5 20:51:15 CST
1995) ready.
Name (localhost:angel):
[1]  + Suspended (tty input)          ftp localhost
```

Corremos el programa ftp en segundo plano. Veamos ahora cual su número de proceso.

```
leon:~> ps -x
  PID TTY STAT  TIME COMMAND
 6366  p0 S     0:02 -tcsh
 6957  p0 T     0:00 ftp localhost
 6960  p0 R     0:01 ps -x
```

Averiguamos que es el proceso 6957. Vamos a terminarlo.

```
leon:~> kill 6957

[1]    Terminated                    ftp localhost
```

Veamos si todavía esta ejecutándose

```
leon:~> ps -x
  PID TTY STAT  TIME COMMAND
 6366  p0 S     0:02 -tcsh
 6961  p0 R     0:01 ps -x
```

Eliminar un proceso puede ser útil cuando queremos terminar una tarea que no debimos haber iniciado o cuando queremos eliminar una tarea que no está funcionando correctamente.

NOTA: PARA DETENER UN PROCESO EN PRIMER PLANO BASTA SÓLO CON ESCRIBIR CTRL - C (^C) EN EL TERMINAL DESDE DONDE FUE INVOCADO.

EJECUCIÓN DE UN PROCESO EN UN INSTANTE ESPECÍFICO

Para ejecutar un programa o script en una fecha y hora especificos puede utilizar el programa *at* con la siguiente sintaxis:

```
at [hora] [dia] [archivo de comandos]
```

Archivo contiene una serie de instrucciones que usted desea ejecutar. Supongamos que usted desea escribir un mensaje por pantalla y lo almaceno en un archivo (editado con vi por ejemplo):

```
leon:~> vi mensaje
echo "Este es un mensaje sin utilidad por pantalla"
```

Ahora hagamos una prueba para ver como funciona el comando:

```
leon:~> date
Sun Oct  5 20:22:54 GMT-4 1997
leon:~> at 2024 -f mensaje
Job 1 will be executed using /bin/sh
```

Con atq podemos ver el trabajo dentro de la cola de ejecución (y lo podemos remover con atrm):

```
leon:~> atq
Date                    Owner   Queue   Job#
20:35:00 10/05/97       angel   c       4
```

Pasado un rato, nos llega un correo mostrando la salida por pantalla del comando

```
New mail for angel@leon has arrived:
----
From: Angel Negro <angel@leon.felinos.ve>
Subject: Output from your job c0000400decbc3

Este es un mensaje sin utilidad por pantalla
```

Supongamos que ahora queremos eliminar un trabajo de la cola. Primero averiguamos la fecha y hora (at supone que el día es actual si la hora no ha transcurrido, supone que es del día siguiente) y luego introducimos el comando con at.

```
leon:~> date
Sun Oct  5 20:37:13 GMT-4 1997
leon:~> at 21:00 -f mensaje
Job 2 will be executed using /bin/sh
leon:~> atq
Date                    Owner   Queue   Job#
21:00:00 10/05/97       angel   c       5
```

Luego vemos la cola, averiguamos su número y lo eliminamos con atrm:

```
leon:~> atrm 5
leon:~> atq
leon:~>
```

Como puede ver, el comando atq no muestra nada en la cola.

El comando at puede ser invocado de varias maneras:

at 8am (Hoy a las 8 de la mañana).
at 2130 (Hoy a las 9:30 PM).
at 12N fri week (Mediodía, una semana después del próximo viernes)
at 2PM apr 3 (El próximo 3 de abril a las 2 PM).

NOTA SOBRE EL COMANDO AT:

Otra forma de ejecutar tareas automáticas es por medio del crontab. Para averiguar más acerca de esta herramienta escriba man crontab.

Combinación de múltiples comandos

Usted puede conectar comandos juntos con el carácter | (barra vertical o pipe) para hacer que la salida de un primer comando se convierta en la entrada del segundo:

```
leon:~> ps -aux | grep angel
angel     8033  3.5  9.4  1124   664  p0 S    05:58   0:01 -tcsh
angel     8039  0.0  5.1   824   364  p0 R    05:58   0:01 ps -
aux
angel     8040  0.0  4.3   868   304  p0 S    05:58   0:00 grep
angel
leon:~>
```

Con este comando estamos pidiendo un listado de todos los procesos y estamos filtrando (con ayuda de la utilidad grep) a todos aquellos que tengan la cadena de caracteres angel. Los comandos interconectados con pipe se llaman tuberías. El tamaño de la tubería puede ser tan largo como se quiera:

```
leon:~> ps -aux | grep angel | grep -v "grep"|grep -v "ps"
angel     8033  0.4  9.5  1124   672  p0 S    05:58   0:01 -tcsh
leon:~>
```

En el ejemplo anterior utilizamos un segundo grep para suprimir la salida de grep como un proceso en nuestro listado y un tercer grep en la tubería para suprimir la salida de ps.

Otra forma de combinar comandos secuenciales es hacerlo utilizando paréntesis con o sin puntos y comas. La forma cómo se agrupan los comandos es muy importante:

```
leon:~> date ; ls /etc| wc -l
Mon Oct  6 06:10:34 GMT-4 1997
     82
leon:~>
```

El comando anterior me muestra la fecha por pantalla y luego me cuenta cuantos archivos existen en el directorio /etc, mientras que:

```
leon:~> (date ; ls /etc) | wc -l
     83
leon:~>
```

Me cuenta a la fecha producida por date como una línea más, aparte del listado producido por ls.

¿ producirán las siguientes líneas resultados distintos?:

```
leon:~> ( echo "En el directorio raiz hay" ; ls / | wc -l ; echo
"archivos y ocupan el siguiente espacio:" ; df )

leon:~> echo "En el directorio raiz hay" ; (ls / | wc -l) ; echo
"archivos y ocupan el siguiente espacio:" ; df
```

ENTRADA Y SALIDA DE COMANDOS

Por omisión, muchos comandos que producen salida la escriben al terminal. Si usted utiliza el carácter >, la concha le permite enviar la salida a un archivo:

```
leon:~> ls /etc|wc -l >~/listado-archivos-etc
leon:~> cat ~/listado-archivos-etc
     82
leon:~>
```

Como podemos ver, el contenido de ese listado fue enviado a un archivo llamado listado-archivos-etc, en vez de ser mostrado por pantalla. El carácter > sobreescribe cualquier archivo que haya sido creado previamente, mientras que el carácter >> agrega al final del archivo:

```
leon:~> cat /etc/passwd|wc -l > ~/listado-archivos-etc
leon:~> cat ~/listado-archivos-etc
     20
```

Como puede ver, el contenido del archivo fue sobreescrito (ahora no es 82 sino 20). Vamos a agregar un comentario al final del archivo:

```
leon:~> echo "Hay 20 lineas en mi archivo ahora" >> ~/listado-
archivos-etc
leon:~> cat ~/listado-archivos-etc
     20
Hay 20 lineas en mi archivo ahora
leon:~>
```

Así como podemos crear un archivo con la redirección, podemos leer también desde un archivo con el carácter <. Supongamos que queremos ver cuantas líneas tiene el archivo que acabamos de crear:

```
leon:~> wc -l <~/listado-archivos-etc
      2
leon:~>
```

Finalmente, podemos utilizar la redirección en ambos sentidos sin ningún problema. Vamos a convertir de minusculas a mayusculas el texto del archivo que acabamos de crear:

```
leon:~> cat ~/listado-archivos-etc
     20
Hay 20 lineas en mi archivo ahora
leon:~> tr a-z A-z < ~/listado-archivos-etc >mayusculas
leon:~> cat mayusculas
     20
HAY 20 LINEAS EN MI ARCHIVO AHORA
leon:~>
```

COMPLETACIÓN DE COMANDOS

Cuando usted escribe la primera parte de un nombre, la concha a veces puede saber el resto del nombre, de manera que usted no tenga que escribirlo. Esta característica se llama completación de comandos. Para utilizarla, usted escribe la primera parte del nombre de un archivo, presiona una tecla especial y la concha hace el resto.

Para activar la completación de comandos en tcsh (para averiguar si esa es su concha escriba echo $SHELL), escriba el prefijo del nombre y presione la tecla TAB (tabulación).

En el ejemplo anterior, construimos un archivo llamado ~/mayusculas. Supongamos que queremos ver su contenido otra vez:

```
leon:~> cat may <TAB>
leon:~> cat mayusculas <ENTER>
20
HAY 20 LINEAS EN MI ARCHIVO AHORA
leon:~>
```

En csh, la completación de comandos se usa de manera un poco diferente. Usted presiona ESC en vez de TAB, y además, debe colocar el siguiente comando en su archivo de configuración ~/.cshrc:

```
set filec
```

Si un prefijo es ambiguo (se encuentra más de un nombre de archivo) la concha emitirá un sonido. Cuando esto ocurra usted puede ver un listado de las posibilidades que encontró la concha con Ctrl-D.
Finalmente, usted puede ayudar a la concha excluyendo ciertos sufijos de la completación de comandos, agregando la siguiente línea a su ~/.cshrc:

```
Set fignore = (.o *.bak)
```

NOTA: EL ARCHIVO .CSHRC ES UTILIZADO TANTO POR TCSH COMO POR CSH. TCSH ES COMPATIBLE CON LA SINTAXIS DE CSH PERO NO AL REVES.

USO DE PATRONES

Los patrones de archivos usando *, ?. o [...] son útiles para especificar grupos de archivos relacionados, pero sólo si estos ya existen. Si usted está tratando de especificar nombres para nuevos archivos existe una técnica que lo puede ayudar.

Suponga que usted quiere crear los directorios angel1, angel2, angel3 y angel4 para guardar allí fotos de sus ángeles favoritos ;-). Un patron como el siguiente fallaría porque los directorios no existen:

```
leon:~> mkdir angel[1-4]
mkdir: No match.
leon:~>
```

Pero en cambio, si usted utiliza {} para crear los directorios, los resultados serán diferentes:

```
leon:~> mkdir angel{1,2,3,4}
leon:~> ls
angel1/               angel4/               mensaje
angel2/               listado-archivos-etc
angel3/               mayusculas
leon:~>
```

REUTILIZACIÓN, REPETICIÓN Y SUSTITUCIÓN DE COMANDOS

La concha tiene un mecanismo para mantener una historia de los comandos. Eso significa que recuerda los comandos que usted ejecuta y permite que los vuelva a correr con unos pocos golpes de tecla. Supongamos que usted esta ejecutando los siguientes comandos (que tienen mucho en común):

```
leon:~> ls
leon:~> rmdir angel*
leon:~> ls
leon:~> mkdir angel{1,2,3,4}
leon:~> ls
leon:~> ls
leon:~> cat listado-archivos-etc
leon:~> cat mayusculas
leon:~> cat mensaje
```

Podemos ver si están en nuestra historia de comandos con la sentencia history (se muestra sólo una salida abreviada):

```
leon:~> history
32  6:58    cat mayusculas
33  7:11    mkdir angel[1-4]
34  7:12    mkdir angel{1-4}
35  7:12    ls
36  7:13    mkdir angel{1,2,3,4}
37  7:13    ls
38  7:13    rmdir angel*
39  7:13    ls
40  7:13    mkdir angel{1,2,3,4}
41  7:13    ls
42  7:55    ls
43  7:55    cat listado-archivos-etc
44  7:55    cat mayusculas
45  7:55    cat mensaje
```

```
46  7:58    history
```

Vemos que aparecen todos los comandos que hemos escrito en nuestra sesión (podemos limitar el tamaño de la historia colocando la línea `set history=20` en el ~/.cshrc).

Juguemos un poco con nuestra historia de comandos:

```
leon:~> rmdir angel*
```

El siguiente comando busca a cualquier comando que contenga el carácter *3* en la historia y lo ejecuta de nuevo:

```
leon:~> !?3
mkdir angel{1,2,3,4}
```

Repetimos el comando más reciente que comience por r:

```
leon:~> !r
rmdir angel*
```

Pudimos haberlo hecho invocando de nuevo al último comando escrito:

```
leon:~> !!
rmdir angel *
```

O bien pudimos haberlo buscado de la historia de comandos directamente:

```
leon:~> !38
rmdir angel *
```

Repitamos el la última sentencia que utilizaba el comando cat y cambiemosla por rm:

```
leon:~> !45:s/cat/rm
rm mensaje
```

Aprovechemos este último comando y eliminemos el archivo *mayusculas* cambiando el argumento al comando que acabamos de ejecutar:

```
leon:~> ^mensaje^mayusculas
rm mayusculas
```

Podemos crear de nuevo al archivo mayusculas usando touch y utilizando el primer parámetro del comando anterior:

```
leon:~> touch !^
touch mayusculas
```

NOTA SOBRE LA COMPLETACIÓN DE COMANDOS:

Para que un comando utilice todos los argumentos del comando anterior escriba comando !. Para que un comando utilice el último argumento del comando anterior escriba comando !$.*

Podemos repetir una serie de comandos utilizando un lazo (loop). Supongamos que queremos crear los archivos archivo2, experimento, prueba1:

```
leon:~> foreach i ( prueba1 archivo2 experimento )
foreach? touch $i
foreach? end
leon:~> ls
archivo2     experimento  prueba1
leon:~>
```

Primero declaramos el lazo y una variable de control, luego le decimos qué comando queremos usar con cada una de los elementos en el lazo y finalmente terminamos de definir las instrucciones escribiendo la palabra end (con lo cual se inicia la ejecución del lazo).

CREANDO ATAJOS PARA COMANDOS

A veces es conveniente crear atajos para los comandos mas utilizados (esto se puede hacer utilizando el archivo .cshrc). Por ejemplo, supongamos que cada vez que queramos limpiar la pantalla escribimos cls (en vez de clear) y h para ver la historia de comandos (en vez de history):

```
leon:~> alias cls clear
leon:~> alias h history
leon:~> h
     1  23:02   ls
     2  23:02   rm listado-archivos-etc
     3  23:02   rm mayusculas
     4  23:02   ls
     5  23:02   clear
     6  23:03   foreach i ( prueba1 archivo2 experimento )
     7  23:03   foreach i ( prueba1 archivo2 experimento )
     8  23:03   ls
     9  23:14   alias cls clear
    10  23:14   alias h history
    11  23:14   h
leon:~>
```

USANDO SUSTITUCIÓN DE COMANDOS

Supongamos que queremos obtener información del demonio inetd (demonio encargado de controlar otros demonios del sistema). Podríamos hacerlo así:

```
leon:~> ps -aux | grep in.inetd | grep -v grep
root        54  0.0  4.7   780   336  ?  S   Sep 30   0:02
/usr/sbin/inetd
leon:~>
```

Sin embargo, sabemos que algunos demonios dejan dicho con que identificador de proceso están corriendo en un archivo de texto en el directorio /var/run (este es el caso del inetd). Si pudiéramos darle esa información a ps todo sería más sencillo:

```
leon:~> ps -aux `cat /var/run/inetd.pid`
USER       PID %CPU %MEM  SIZE   RSS TTY STAT START   TIME
COMMAND
root        54  0.0  4.7   780   336  ?  S   Sep 30   0:02
/usr/sbin/inetd
leon:~>
```

De esta manera nos evitamos utilizar dos veces al programa grep (y solamente utilizamos una vez al programa cat).

USANDO CONTROL DE TRABAJOS

Un trabajo es cualquier comando que usted ha iniciado y aún no se ha completado. La concha provee una facilidad de control que detener (suspender) comandos de manera temporal y los resuma más tarde, o mueva los comandos atrás y adelante entre el primer y segundo plano.

El control de trabajos es útil en las siguientes situaciones:

- Usted puede mover un trabajo lento al segundo plano, y trabajar en otra cosa mientras el trabajo lento se ejecuta. También el usuario puede detener un trabajo largo o un trabajo que se está tardando más de los esperado y puede colocarlo en segundo plano sin necesidad de matarlo.
- Usted puede poner a dormir un trabajo, resumiéndolo después para que continúe.
- Usted puede matar un comando si decide no dejarlo terminar por alguna razón.

La siguiente tabla muestra la lista de comandos básicos para control de trabajos:

Comando	Efecto del comando
Ctrl - z	Para un trabajo en primer plano
Ctrl - c	Interrumpe un trabajo en primer plano
Fg	Envía un trabajo que estaba detenido o en segundo plano a primer plano
Bg	Mueve un trabajo detenido a segundo plano
Kill %	Mata un trabajo detenido o en segundo plano
Stop %	Detiene un trabajo en segundo plano
Jobs	Muestra la lista de trabajo actuales

Tabla 2: Comandos usados para control de trabajos

Vamos a probar ahora los comandos que acabamos de aprender. Detengamos un trabajo que está en primer plano:

```
leon:~> sleep 2000 <Ctrl-Z>
Suspended
leon:~>
```

Ahora vamos a dejarlo corriendo en segundo plano:

```
leon:~> bg
[1]    sleep 2000 &
leon:~>
```

Veamos como podemos traer un proceso a primer plano con fg:

```
leon:~> sleep 5000 <Ctrl-Z>
Suspended
leon:~> fg
sleep 5000
```

Veamos ahora que trabajos están en segundo plano:

```
leon:~> jobs
[1]  + Running                       sleep 2000
[2]    Running                       sleep 5000
leon:~>
```

El signo + significa que el trabajo 1 es el trabajo actual. Para detener el trabajo actual escriba:

```
leon:~> stop %

[1]  + Suspended (signal)            sleep 2000
leon:~>
```

Podemos referirnos a los trabajos por número. Eliminemos el trabajo 2:

```
leon:~> kill %2

[2]    Terminated                    sleep 5000
leon:~>
leon:~> jobs
[1]  + Suspended (signal)            sleep 2000
leon:~>
```

Aún tenemos un trabajo detenido. Podemos matarlo si queremos, refiriéndonos al proceso por su nombre:

```
leon:~> kill %sleep

[1]    Terminated                    sleep 2000
leon:~>
```

Otra forma de referirnos a un trabajo es decirle a la concha que busque cierta cadena (y no el inicio de la cadena). Suponga que tenemos el siguiente caso:

```
leon:~> jobs
[1]  + Running                       sleep 5000
[2]  - Running                       sleep 5001
leon:~>
```

Aquí podemos eliminar al trabajo 2 sin necesidad de utilizar su número de trabajo (fíjese que ambos comienzan con la cadena sleep y el método de referencia anterior no nos sirve):

```
leon:~> kill %?01
leon:~>
[2]    Terminated                    sleep 5001
leon:~>
```

Con esta orden le decimos a la concha que busque un grupo de caracteres dentro del identificador del trabajo.

TRABAJO EN MAQUINAS REMOTAS

Una característica muy poderosa de UNIX es que permite trabajar a distancia con otras máquinas. Tanto telnet como rlogin permiten conectarse a computadoras remotas, permitiendo trabajar en ellas como si se estuviera sentado frente a ellas introduciendo ordenes por teclado.

El uso de telnet y rlogin depende de la disponibilidad de estas herramientas (por ejemplo, bajo Windows95 usted conseguirá solamente telnet, mientras que bajo cualquier UNIX encontrará tanto telnet como rlogin). Telnet y rlogin poseen funciones similares y veremos cómo usar ambas herramientas.

Supongamos que queremos conectarnos a una máquina llamada tigre.felinos.ve usando telnet:

```
leon:~>telnet tigre.felinos.ve
tigre login: angel
Password:
Last login: Wed Oct  8 08:52:42 from leon.felinos.ve
Linux 2.0.27.
tigre:~>
```

Otra forma de invocar telnet es escribiendo telnet <ENTER> y escribir los comandos desde allí:

```
leon:~> telnet
telnet>
```

Una vez dentro de telnet podemos pedir ayuda para ver sus comandos:

```
telnet> ?
Commands may be abbreviated.  Commands are:

close           close current connection
logout          forcibly logout remote user and close the connection
display         display operating parameters
mode            try to enter line or character mode ('mode ?' for more)
open            connect to a site
quit            exit telnet
send            transmit special characters ('send ?' for more)
set             set operating parameters ('set ?' for more)
unset           unset operating parameters ('unset ?' for more)
status          print status information
toggle          toggle operating parameters ('toggle ?' for more)
slc             change state of special charaters ('slc ?' for more)
z               suspend telnet
environ         change environment variables ('environ ?' for more)
?               print help information
telnet>
```

Y desde allí podemos conectarnos al sistema remoto:

```
telnet> open tigre
```

```
Trying 192.168.1.2...
Connected to tigre.felinos.ve.
Escape character is '^]'.

Linux 2.0.27 (tigre.felinos.ve) (ttyp0)


tigre login:
```

NOTA SOBRE EL USO DE TELNET:

Con telnet -l usuario máquina puede especificar con qué cuenta quiere trabajar en la máquina remota.

En cambio con rlogin no hace falta especificar el usuario (el programa asume que la cuenta que están en uso es la misma que está en el sistema remoto):

```
leon:~> rlogin tigre
Password:
Last login: Wed Oct  8 08:53:12 from leon.felinos.ve
Linux 2.0.27.
tigre:~>
```

Sin embargo eso no significa que usted no pueda especificar el usuario que usted quiere ser en el sistema remoto (suponiendo que usted sabe el password de esa cuenta):

```
leon:~> rlogin tigre -l mendez
Password:
```

NOTA SOBRE SISTEMAS EQUIVALENTES:

Existe una opción que permite conectarse a otros sistemas (esto se llama equivalencia de sistemas) sin que estos pidan password (esta opción es necesaria con programas como rsh o rcp). Para habilitar esta opción cree un archivo llamado .rhosts en su directorio hogar y coloque allí unas líneas como la siguientes (supongamos que el usuario que hace esto se llama angel y tiene su cuenta en una maquina llamada leon):

leon jose angel josev
tigre josev
puma

Significa que angel, josev y jose pueden conectarse desde leon, que josev puede desde tigre y que angel puede desde puma (todos sin entrar password).

Bajo Linux esta característica está deshabilitada y sólo funciona si estas mismas lineas son agregadas por root en /etc/hosts.equiv:

localhost
pantera.felinos.ve
leon.felinos.ve

puma.felinos.ve

No se recomienda esta opción porque puede ser una brecha de seguridad.

COPIANDO ARCHIVOS DE UN SISTEMA A OTRO

Puede copiar archivos entre sistemas con el comando ftp, escribiendo lo siguiente y presionando enter:

ftp maquina

O escribiendo ftp, presionando luego enter:

ftp

Y después escriba la maquina a la cual se va a conectar:

open maquina

Se debe ser cuidadoso con el tipo o modo de transferencia del archivo. Por lo general se utiliza el modo de transferencia binario (útil para traer programas compilados, archivos comprimidos, imágenes, cualquier cosa que no sea un archivo de texto plano), y el modo de texto (el cual sirve sólo para traer archivos de texto plano).

NOTA SOBRE LOS MODOS DE TRANSFERENCIA:

El modo binario también sirve para traer archivos de texto. El modo binario se asegura que la secuencia de bits del archivo fuente sea la misma que la del archivo de destino, aunque estos bits no tengan un significado útil en la maquina destino (por ejemplo, secuencias de escape de un archivo de texto hecho bajo msdos, distintas a las de UNIX).

Veamos como se comporta una sesión de ejemplo:

```
leon:~> ftp tigre
Connected to tigre.felinos.ve.
220 tigre FTP server (Version wu-2.4(1) Tue Dec 5 20:51:15 CST
1995) ready.
Name (tigre:angel): angel
331 Password required for angel.
Password:
230 User angel logged in.
ftp>
```

NOTA SOBRE FTP ANONIMO:

No es necesario tener un login y password en una maquina remota para utilizar ftp. Se puede utilizar un servidor ftp anonimo, el cual permite dejar y traer archivos de dominio público. Los servidores anónimos piden como login anonymous o ftp y como password la dirección de correo electronica de quien usa el servicio (jose@ing.ula.ve por ejemplo) .

Una vez dentro del directorio remoto podemos examinar el contenido del directorio con el comando *dir* o con el comando *ls*:

```
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
total 339
drwxr-xr-x   2 angel    users        1024 Oct  8 21:18 .
drwxr-xr-x  10 root     root         1024 Oct  8 08:52 ..
-rw-------   1 angel    users         328 Oct  8 21:14 .history
-rw-r--r--   1 angel    users          34 Nov 24  1993 .less
-rw-r--r--   1 angel    users         114 Nov 24  1993 .lessrc
-rw-r--r--   1 angel    users          17 Oct  8 09:53 .rhosts
-rw-rw-r--   1 angel    users      328520 Oct  8 21:17 alltags.html
-rw-rw-r--   1 angel    users        8945 Oct  8 21:18 angel.jpg
226 Transfer complete.
ftp>
```

Vemos que tenemos un archivo posiblemente escrito en html (alltags.html) y una foto llamada angel.jpg. Suponga que en esta sesión ficticia usted quiere traerse ambos archivos:

```
ftp> get alltags.html
200 PORT command successful.
150 Opening ASCII mode data connection for alltags.html
(328520 bytes).
226 Transfer complete.
331704 bytes received in 1.62 secs (2e+02 Kbytes/sec)
```

Ftp está por omisión en modo ascii (por lo que podemos traernos el archivo hecho en html sin ningún problema). Lo traemos usando el comando get.

Para traer la foto, primero especificamos que se trata de un archivo binario, después lo traemos:

```
ftp> bin
200 Type set to I.
ftp> get angel.jpg
200 PORT command successful.
150 Opening BINARY mode data connection for angel.jpg
(8945 bytes).
226 Transfer complete.
8945 bytes received in 0.293 secs (30 Kbytes/sec)
ftp>
```

¿No sería más cómodo traernos los dos archivos de una sola vez con un solo comando?. La sentencia mget se encarga de esto (y prompt off nos da la comodidad de que ftp no nos esté preguntando si queremos traernos un archivo por cada archivo en el directorio):

```
ftp> bin
200 Type set to I.
ftp> prompt
Interactive mode off.
ftp> mget *
local: alltags.html remote: alltags.html
200 PORT command successful.
150 Opening BINARY mode data connection for alltags.html (328520
bytes).
226 Transfer complete.
328520 bytes received in 0.717 secs (4.5e+02 Kbytes/sec)
```

```
local: angel.jpg remote: angel.jpg
200 PORT command successful.
150 Opening BINARY mode data connection for angel.jpg (8945
bytes).
226 Transfer complete.
8945 bytes received in 0.184 secs (48 Kbytes/sec)
ftp>
```

Note como trajimos un archivo de texto (html) utilizando un modo binario (lo contrario no puede hacerse).

Ftp permite un manejo de directorios similar a una concha UNIX como tcsh. El comando pwd está disponible (y nos muestra el directorio de trabajo), ls -R me permite listar de manera recursiva el contenido de subdirectorios. Con ftp también podemos especificar el directorio local en donde queremos colocar nuestros archivos (si estamos trayendolos de algun lado) o en donde están los archivos que queremos dejar. Supongamos que queremos dejar una copia de una imagen llamada angel2.jpg en el servidor remoto, y que la imagen se encuentra en el directorio ~angel/fotos/angel2.jpg:

```
ftp> lcd ~angel/fotos
Local directory now /home/angel/fotos
ftp> bin
200 Type set to I.
ftp> put angel2.jpg
200 PORT command successful.
150 Opening BINARY mode data connection for angel2.jpg.
226 Transfer complete.
8945 bytes sent in 0.0116 secs (7.5e+02 Kbytes/sec)
ftp>
```

Veamos las operaciones que se realizaron aquí:

1. Cambiamos el directorio local de ~angel a ~angel/fotos
2. Colocamos el modo de transferencia en binario
3. Enviamos el archivo con el comando put

Tambíen podemos enviar varios archivos con el comando **mput** (el comando prompt off también se aplica aquí):

```
ftp> hash
Hash mark printing on (1024 bytes/hash mark).
```

Con el comando hash podemos monitorear de manera un poco más grafica como va la transferencia del archivo:

```
ftp> mput *
local: ./angel2.jpg remote: ./angel2.jpg
200 PORT command successful.
150 Opening BINARY mode data connection for ./angel2.jpg.
########
226 Transfer complete.
8945 bytes sent in 0.0108 secs (8.1e+02 Kbytes/sec)
ftp>
```

En este caso sólo había un archivo dentro del directorio ~angel/fotos.

Un último truco con ftp: usted puede darle ordenes a la concha desde ftp usando el carácter !:

```
ftp> !
> who
angel    ttyp0    Oct  8 20:48 (pantera.felinos.)
> exit
exit
ftp>
```

Fíjese como ejecutamos el comando who en la maquina local y luego volvimos a ftp para continuar trabajando en la máquina remota.

COMANDOS MISCELANEOS PARA EL MANEJO DE ARCHIVOS

COMPRIMIENDO ARCHIVOS CON GZIP

Los archivos abundan en un sistema UNIX y en vez de eliminarlos, una mejor opción es comprimirlos (hasta que estos vuelvan a ser necesitados) para que ocupen menos espacio.

Para comprimir un archivo con gzip escriba lo siguiente:

```
gzip -9 [archivo]
```

La opción -9 le dice a gzip que aplique una compresión máxima al archivo.

Un archivo comprimido con gzip tiene la extensión .gz agregada al final del archivo.

EMPAQUETANDO ARCHIVOS CON TAR (TAPE ARCHIVER)

La utilidad tar ha sido utilizada bajo UNIX por varios años. La utilidad tar fue diseñada para crear un solo archivo a partir de varios archivos (similar a pkzip, sólo que tar no comprime los archivos).

La sintaxis general de tar es

```
tar [opciones] [archivo]
```

Tar tiene muchas opciones numerosas y algunas de ellas hasta confusas.

Un ejemplo simple para crear un archivo es el siguiente:

```
tar cvf respaldo1.tar /home/angel
```

Tar me crea el archivo respaldo1.tar con todo el contenido del directorio /home/angel. La opcion c (create) me crea un archivo desde cero (y si el archivo existe lo sobreescribe), la opción v (verbose) me muestra por pantalla todos los mensajes a medida que se empaqueta el archivo y la opción f (file) me dice cual es el nombre del archivo de salida (tar no le agrega la extensión .tar a sus archivos, aunque se hace por convención para saber que fueron empaquetados con tar).

Para desempaquetar un archivo hecho con tar escribimos

```
tar xvf [archivo.tar]
```

Tar se encarga de volver a crear los subdirectorios y directorios en donde estaban almacenados cada uno de los archivos (lo que lo hace muy útil para hacer respaldos).

Si usted lo que quiere es ver el contenido del archivo tar antes de descomprimirlo escriba

```
tar tvf [archivo.tar]
```

BUSQUEDAS AVANZADAS DE PATRONES Y ARCHIVOS

USO DE GREP PARA BUSCAR PATRONES

El comando grep busca en su entrada lineas que contienen un patrón dado. La mayoría de los usuarios utilizan grep para buscar cadenas de caracteres dentro de archivos.

Por ejemplo, supongamos que estamos interesados en saber cuántas veces aparece la palabra ftp dentro del archivo /etc/services:

```
leon:~> grep -c ftp /etc/services
3
leon:~>
```

Otro uso muy común de grep es encontrar información solamente de ciertos procesos de un usuario. Veamos cómo podemos saber qué procesos está ejecutando el usuario angel:

```
leon:~> ps -aux | grep angel
angel    13264  0.3 10.0  1132   704  p0 S    09:17   0:01 -tcsh
angel    13297  2.8  6.2   844   436  p0 T    09:24   0:00 ftp tigre
angel    13299  0.0  5.1   824   364  p0 R    09:24   0:01 ps -aux
angel    13300  0.0  4.3   868   304  p0 S    09:24   0:00 grep angel
leon:~>
```

¡Escriba solamente ps -aux y pruebe qué ocurre!

Otro sitio útil para usar grep es con el comando man -k. En el ejemplo utilizado para mostrar ayuda sobre el cambio de nombre de archivos pudimos haber utilizado:

```
leon:~> man -k rename|grep files
mv (1)               - rename files
mv (1)               - rename files
leon:~>
```

USO DE FIND PARA BUSCAR ARCHIVOS

Find es uno de esos comandos que le hacen pensar como pudo haber estado trabajando con UNIX sin haberlo utilizado (una vez aprendido, claro esta ;-)) .

Find localiza archivos que cumplen con ciertas características comunes, las cuales usted especifica, en cualquier parte del sistema en la cual usted le dice que busque. Find tiene la siguiente sintaxis:

```
find directorios-de-busqueda-de-inicio criterios-de-busqueda-y-
acciones
```

El uso más sencillo de find es encontrar archivos y mostrarnos su ubicación. Por ejemplo, queremos saber en donde están todos nuestros archivos jpg en el directorio ~angel:

```
leon:~> find ~angel -name "*.jpg" -print
/home/angel/angel.jpg
/home/angel/fotos/angel2.jpg
leon:~>
```

Como puede ver, find hace una búsqueda recursiva por todo el árbol de directorio.

Supongamos que hace tres dias nosotros copiamos ciertos archivos y queremos saber cuáles son (este es un ejemplo de una busqueda refinada):

```
leon:~> find ~angel -name "*" -atime -2 -print
/home/angel
/home/angel/X_COURSE.TGZ
/home/angel/alltags.html
/home/angel/angel.jpg
/home/angel/fotos
/home/angel/fotos/angel2.jpg
leon:~>
```

El parámetro atime -2 significa archivos cuyo tiempo de acceso sea menor a dos días.

Pero la utilidad de find va más allá de solamente mostrar la ubicación de archivos. Podemos decirle a find que ejecute ciertas acciones sobre los archivos que cumplan cierto criterio. Pidámosle que nos muestre el tamaño y permisos de los archivos que tienen más de 50Kb en nuestro directorio:

```
leon:~> find ~angel -size +100 -exec ls -ls {} \;
 139 -rw-rw-r--   1 angel    users      140580 Oct  8 20:59
/home/angel/X_COURSE.TGZ
324 -rw-rw-r--   1 angel    users      328520 Oct  8 21:06
/home/angel/alltags.html
leon:~>
```

Un uso muy útil de find es el de eliminar basura. Podemos pedirle que elimine los archivos que tengan más de 50Kb y que nos los muestre por pantalla:

```
leon:~> find ~angel -size +100 -print -exec rm {} \;
/home/angel/X_COURSE.TGZ
/home/angel/alltags.html
leon:~>
```

NOTA SOBRE EL USO DE FIND:
Tenga cuidado con el comando anterior. Un archivo de más de 50Kb podría ser un documento importante. Lea las páginas de find para saber más acerca de su uso.

BIBLIOGRAFÍA

La siguiente bibliografía está relacionada con el tema y puede resultar de mucha utilidad en su aprendizaje:

- Sun Microsystems. SunOS User's Guide, Getting Started. 9-61. 170p. 1990.
- Husain Kamran, Parker Tim. Linux Unleashed. Sams Publishing. 1015p. 1992.
- Thomas Rebecca, Yates Jean. Sistema operativo UNIX, guía del usuario. OSBORNE, McGraw-Hill. 493p.1985.
- Fisher Sharon. Riding the Internet Highway. New Riders Publishing. 246p. 1993.
- Krol De. The Whole Intenet. User's Guide & Catalog. O'Reilly & Associates, INC. 365p. 1992.
- Dunbois Paul. Using Csh an Tcsh. O'Reilly & Associates, INC. 211p. 1995.
- Frisch A. Essential System Administration. O'Reilly & Associates, INC. 417p. 1993.
- Blanco Lino, Galan Susana. Linux, al día en una hora. Anaya Multimedia. 127p. 1997.
- Página oficial de Linux en Internet. http://www.linux.org
- Linux Documentation Proyect en castellano. http://www.infor.es/LuCAS/