**Tutorial de Linux**

# Índice

## 1 Introducción

Los nuevos usuarios de UNIX y Linux pueden estar un poco intimidados por el tamaño y aparente complejidad del sistema que tienen ante si. Hay muchos buenos libros sobre el uso de UNIX para todos los niveles, desde novatos a expertos. Pero ninguno de estos libros cubre específicamente una introducción al uso de Linux. Mientras el 95% del uso de Linux es exactamente como cualquier otro UNIX, la forma mas clara de comenzar con su nuevo sistema es un tutorial a medida para Linux. He aqui ese tutorial.

Este tutorial no presentara gran cantidad de detalles o cubrirá temas muy avanzados. Sino que esta pensado para permitir al nuevo usuario de Linux comenzar a usar el sistema y situarlo en una posición en la que el o ella puedan leer libros mas generales sobre UNIX y entender las diferencias básicas entre otros sistemas UNIX y Linux.

Se va a presuponer muy poco, excepto quizá alguna familiaridad con los ordenadores personales y MS-DOS. Pero incluso si no es un usuario de MS-DOS, debería ser capaz de entender cualquier cosa de las que hablemos. A primera vista, UNIX parece como MS-DOS (después de todo, partes de MS-DOS fueron tomadas de CP/M, el cual fue a su vez inspirado en UNIX). Pero, solo las características superficiales de UNIX se parecen a MS-DOS. Incluso si es completamente nuevo en el mundo de los PC, este tutorial debería serle de ayuda.

Y, antes de comenzar: No tenga miedo de experimentar. El sistema no le morderá. No puede destruir nada trabajando con el sistema. UNIX tiene ciertos sistemas de seguridad para evitar que usuarios 'normales' (del tipo que suponemos que es usted) dañen ficheros esenciales para el sistema. Incluso si ocurre el peor de los casos que es que borre todos sus ficheros, tendrá que volver atrás y reinstalar el sistema, pero incluso en ese caso, no hay nada que perder.

## 2 Conceptos basicos de UNIX

UNIX es un sistema operativo multitarea y multiusuario. Esto significa que puede haber mas de una persona usando un ordenador a la vez, cada uno de ellos ejecutando a su vez diferentes aplicaciones. (Esto difiere de MS-DOS, donde solo una persona puede usar el sistema en un momento dado). Bajo UNIX, para que los usuarios puedan identificarse en el sistema, deben presentarse (login), proceso que consta de dos pasos: Introducir el nombre de usuario (login) (el nombre con que será identificado por el sistema), y una palabra de paso (password), la cual es su llave personal secreta para entrar en la cuenta. Como solo usted conoce su palabra de paso, nadie mas podrá presentarse en el sistema con su nombre de usuario.

En los sistemas UNIX tradicionales, el administrador del sistema asignara el nombre de usuario y una palabra de paso inicial en el momento de crear la cuenta de usuario. Como usted es el administrador del sistema, debe configurar su propia cuenta antes de poder presentarse ver Sección 2.1 mas adelante. Para el resto de las discusiones, usaremos el nombre de usuario "madhawk".

Además, cada sistema UNIX tiene un nombre del sistema (hostname) asignado. Este "hostname" le da nombre a la maquina, además de carácter y encanto. El nombre del sistema es usado para identificar maquinas en una red, pero incluso aunque la maquina no este en red, debería tener su nombre. En nuestros ejemplos, el nombre del sistema será "mousehouse"

### 2.1 Creación de una cuenta

Antes de poder usar el sistema, deberá configurarse una cuenta de usuario. Esto es necesario, porque no es buena idea usar la cuenta de root para los usos normales. La cuenta de root debería reservarse para el uso de comandos privilegiados y para el mantenimiento del sistema.

Para crear su propia cuenta, necesita entrar en la cuenta de root y usar las ordenes *useradd* o *adduser*.

### 2.2 Presentación en el sistema (loggin in)

En el momento de presentarse en el sistema, vera la siguiente linea de comandos en la pantalla:

```
mousehouse login:
```

Ahora, introduzca su nombre de usuario y pulse |_Return_.| Nuestro héroe madhawk, teclearía lo

siguiente:

```
mousehouse login:   madhawk
Password:
```

Ahora introduzca la palabra de paso. Esta no será mostrada en la pantalla conforme se va tecleando, por lo que debe teclear cuidadosamente. Si introduce una palabra de paso incorrecta, se mostrara el siguiente mensaje

```
Login incorrect
```

y deberá intentarlo de nuevo.
Una vez que ha introducido correctamente el nombre de usuario y la palabra de paso, esta oficialmente "presentado" en el sistema y libre para comenzar a trabajar.

**2.3 Consolas virtuales**

La consola del sistema es el monitor y teclado conectado directamente al sistema. (Como UNIX es un sistema operativo multiusuario, puede tener otros terminales conectados a puertos serie del sistema, pero estos no serán la consola). Linux, como otras versiones de UNIX, proporciona acceso a consolas virtuales (o VC's), las cuales le permitirán tener mas de una sesión de trabajo activa desde la consola a la vez.
Para demostrar esto, entre en su sistema (como hemos visto antes). Ahora pulse |_alt-F2_.| Debería ver la pregunta login: de nuevo. Esta viendo la segunda consola virtual ha entrado en el sistema por la primera. Para volver a la primera VC, pulse |_alt-F1_.| Voila! ha vuelto a la primera sesión.
Un sistema Linux recién instalado probablemente le permita acceder a las primeras cuatro VC's, usando |_alt-F1_ | a |_alt-F4_.| Pero es posible habilitar hasta 12 VC's_ una por cada tecla de función del teclado. Como puede ver, el uso de VC's es muy potente puede estar trabajando en diferentes VC's a la vez.
Mientras que el uso de VC's es algo limitado (después de todo, solo puede mirar un VC cada vez), esto debería darle una idea de las capacidades multiusuario del sistema. Mientras esta trabajando en el VC #1, puede conmutar al VC #2 y comenzar a trabajar en otra cosa.

**2.4 Interpretes de comandos y comandos**

En la mayoría de las exploraciones en el mundo de UNIX, estará hablando con el sistema a traves del uso de un interprete de comandos. Un interprete de comandos es simplemente un programa que toma la entrada del usuario (p.ej. las ordenes que teclea) y las traduce a instrucciones. Esto puede ser comparado con el COMMAND.COM de MS-DOS, el cual efectúa esencialmente las misma tarea. El interprete de comandos es solo uno de los interfaces con UNIX. Hay muchos interfaces posibles como el sistema X Windows, el cual le permite ejecutar comandos usando el raton y el teclado.
Tan pronto como entra en el sistema, el sistema arranca un interprete de comandos y Ud. ya puede teclear ordenes al sistema. Veamos un ejemplo rápido. Aqui, madhawk entra en el sistema y es situado en el interprete de comandos

```
mousehouse login:   madhawk
Password:   madhawk's password
Welcome to Mousehouse!

/home/madhawk#
```

"/home/madhawk#" es el "prompt" del interprete de comandos, indicando que esta listo para recibir ordenes. Tratemos de decirle al sistema que haga algo interesante:

```
/home/madhawk# make love
make:   *** No way to make target `love'.   Stop.
/home/madhawk#
```

Bien, como resulta que make es el nombre de un programa ya existente en el sistema, el interprete de comandos lo ejecuta. (Desafortunadamente, el sistema no esta siendo muy amigable).
Esto nos lleva a una cuestion importante: ¿Que son ordenes? ¿Que ocurre cuando tecleamos "make love"? La primera palabra de la orden, "make", es el nombre de la orden a ejecutar. El resto de la

orden es tomado como argumentos de la orden. Ejemplos:

```
/home/madhawk# cp foo bar
```

Aqui, el nombre de la orden es "cp", y los argumentos son "foo" y "bar".
Cuando teclea una orden, el interprete de comandos hace varias cosas. Primero de todo, busca el nombre de la orden y comprueba si es una orden interna. (Es decir, una orden que el propio interprete de comandos sabe ejecutar por si mismo. Hay bastantes ordenes de ese tipo que veremos mas adelante). El interprete de comandos también comprueba si la orden es un "alias" o nombre sustitutorio de otra orden. Si no se cumple ninguno de estos casos, el interprete de comandos busca el programa y lo ejecuta pasándole los argumentos especificados en la linea de comandos.
En nuestro ejemplo, el interprete de comandos busca el programa llamado make y lo ejecuta con el argumento love. make es un programa usado a menudo para compilar programas grandes, y toma como argumentos el nombre de un "objetivo" a compilar. En el caso de "make love", ordenamos a make que compile el objetivo love. Como make no puede encontrar un objetivo de ese nombre, falla enviando un mensaje de error y volviendo al interprete de comandos.
¿Que ocurre si tecleamos una orden y el interprete de comandos no puede encontrar el programa de ese nombre? Bien, probémoslo:

```
/home/madhawk# eat dirt
eat: command not found
/home/madhawk#
```

Bastante simple, si no se puede encontrar el programa con el nombre dado en la orden (aqui "eat"), se muestra un mensaje de error que debería de ser autoexplicativo. A menudo vera este mensaje de error si se equivoca al teclear una orden (por ejemplo, si hubiese tecleado "mkae love" en lugar de "make love".

**2.5 Salida del sistema**

Antes de ahondar mas, deberíamos ver como salir del sistema. Desde la linea de ordenes usaremos la orden para salir. Hay otras formas, pero esta es la mas facil.

```
/home/madhawk# exit
```

**2.6 Cambiando la palabra de paso**

También debe asegurarse de la forma de cambiar su palabra de paso. La orden *passwd* le pedirá su palabra de paso vieja y la nueva. Volverá a pedir una segunda vez la nueva para validarla. Tenga cuidado de no olvidar su palabra de paso si eso ocurre, deberá pedirle al administrador del sistema que la modifique por usted.

**2.7 Ficheros y Directorios**

Bajo la mayoría de los sistemas operativos (UNIX incluido), existe el concepto de fichero, el cual es un conjunto de información al que se le ha asignado un nombre (llamado nombre del fichero).
Ejemplos de fichero son un mensaje de correo, o un programa que puede ser ejecutado. Esencialmente, cualquier cosa salvada en el disco es guardada en un fichero individual.
Los ficheros son identificados por sus nombres. Por ejemplo, el fichero que contiene su historial podría ser salvado con el nombre history-paper. Estos nombres usualmente identifican el fichero y su contenido de alguna forma significativa para usted. No hay un formato estandar para los nombres de los ficheros como lo hay en MS-DOS y en otros sistemas operativos; en general estos pueden contener cualquier carácter (excepto / _ ver la discusión sobre "pathnames" (rutas de ficheros) mas adelante), y están limitados a 256 caracteres de longitud.
Con el concepto de fichero aparece el concepto de directorio. Un directorio es simplemente una colección de ficheros. Puede ser considerado como una "carpeta" que contiene muchos ficheros diferentes. Los directorios también tienen nombre con el que los podemos identificar. Además, los directorios mantienen una estructura de árbol; es decir, directorios pueden contener otros directorios.
Un fichero puede ser referenciado por su nombre con camino, el cual esta constituido por su nombre, antecedido por el nombre del directorio que lo contiene. Por ejemplo, supongamos que

madhawk tiene un directorio de nombre papers que contiene tres ficheros: history-final, english-lit y masters-thesis. (Cada uno de los tres ficheros contiene información sobre tres de los proyectos en los que madhawk esta trabajando). Para referirse al fichero english-lit, madhawk puede especificar su camino:

papers/english-lit

Como puede ver, el directorio y el nombre del fichero van separados por un carácter /. Poresta razón, los nombres de fichero no pueden contener este carácter. Los usuarios de MS-DOS encontraran esta convención familiar, aunque en el mundo MS-DOS se usa el carácter \).

Como hemos mencionado, los directorios pueden anidarse uno dentro de otro. Por ejemplo, supongamos que madhawk tiene otro directorio dentro de papers llamado cheat-sheet. El camino de este fichero seria

papers/notes/cheat-sheet

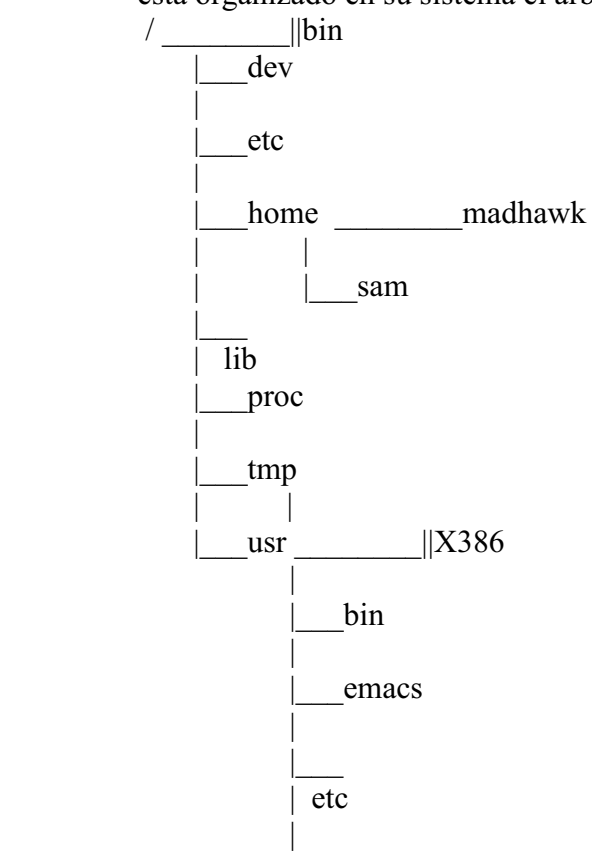
Por lo tanto, el camino realmente es la "ruta" que se debe tomar para localizar a un fichero. El directorio sobre un subdirectorio dado es conocido como el directorio padre. Aqui, el directorio papers es el padre del directorio notes.

**2.8 El árbol de directorios**

La mayoría de los sistemas UNIX tienen una distribucion de ficheros estandar, de forma que recursos y ficheros puedan ser fácilmente localizados. Esta distribucion forma el árbol de directorios, el cual comienza en el directorio "/", también conocido como "directorio raíz". Directamente por debajo de / hay algunos subdirectorios importantes: /bin, /etc, /dev y /usr, entre otros. Estos a su vez contienen otros directorios con ficheros de configuración del sistema, programas, etc.

En particular, cada usuario tiene un directorio "home". Este es el directorio en el que el usuario guardara sus ficheros. En los ejemplos anteriores, todos los ficheros de madhawk (como cheat-sheer y history-final) estaban contenidos en el directorio home de madhawk. Usualmente, los directorios home de los usuarios cuelgan de /home y son nombrados con el nombre del usuario al que pertenecen. Por lo tanto, el directorio "home" de madhawk es /home/madhawk.

En la figura 1 se muestra un árbol de directorio de ejemplo. Este debería darle una idea de como esta organizado en su sistema el árbol de directorios.



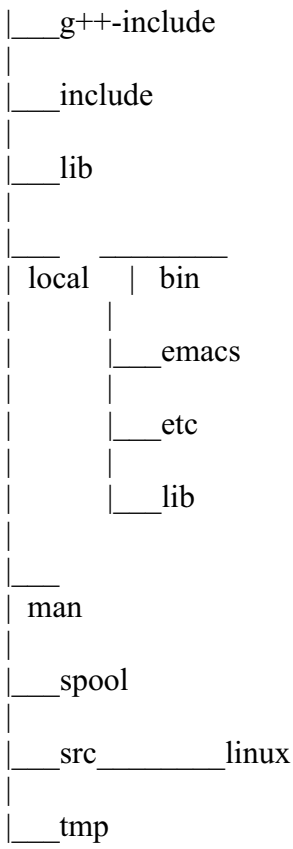



**Figura 1: Típico árbol de directorios UNIX (resumido).**

**3.2.9 Directorio de trabajo actual**

En cualquier momento, las ordenes que teclee al interprete de comandos son dadas en términos de su directorio de trabajo actual. Puede pensar en su directorio actual de trabajo como en el directorio en el que actualmente esta "situado". Cuando entra en el sistema, su directorio de trabajo se inicializa a su directorio home/home/madhawk en nuestro caso. En cualquier momento que referencie a un fichero, puede hacerlo en relación a su directorio de trabajo actual, en lugar de especificar el camino completo del fichero.

Vemos un ejemplo. madhawk tiene el directorio papers, y papers contiene el fichero history-final.

Si madhawk quiere echar un vistazo a ese fichero, puede usar la orden

/home/madhawk# more /home/madhawk/papers/history-final

La orden *more* simplemente muestra el fichero, pantalla a pantalla. Pero, como el directorio de trabajo actual de madhawk es /home/madhawk, podría haberse referido al fichero de forma relativa a su directorio de trabajo actual. La orden seria

/home/madhawk# more papers/history-final

Por lo tanto, si comienza el nombre de un fichero (como papers/final) con un carácter distinto a "/", el sistema supone que se esta refiriendo al fichero con su posición relativa a su directorio de trabajo. Esto es conocido como camino relativo.

Por otra parte, si comienza el nombre del fichero con "/", el sistema interpreta esto como un camino completo es decir, el camino al fichero completo desde el directorio raíz, /. Esto es conocido como camino absoluto.

**2.10 Refiriéndose al directorio home**

Bajo tcsh y bash, el directorio "home" puede ser referenciado usando el carácter de la tilde ( ~ ). Por ejemplo, la orden

/home/madhawk# more ~/papers/history-final

es equivalente a

/home/madhawk# more /home/madhawk/papers/history-final

El carácter "~" es simplemente sustituido por el interprete de comandos, con el nombre del

directorio home. Además, también puede especificar otros directorios home de usuarios con la tilde. El camino "~zipzero/letters" es traducido por el interprete de ordenes a "/home/zipzero/letters" (si /home/zipzero es el directorio home de ZipZero). El uso de la tilde es simplemente un atajo; no existe ningún directorio llamado "~" es simplemente una ayuda sintáctica proporcionada por el interprete de comandos.

## 3 Primeros pasos en UNIX

Antes de comenzar es importante destacar que todos los nombres de ficheros y comandos son "case-sensitive" (que hacen diferencia entre mayúsculas y minúsculas, a diferencia de sistemas operativos como MS-DOS). Por ejemplo, el comando make es diferente a Make o MAKE. Lo mismo ocurre en el caso de nombres de ficheros o directorios.

### 3.1 Moviéndonos por el entorno

Ahora que ya podemos presentarnos como usuarios, y sabemos como indicar ficheros con su camino completo, ¿como podemos cambiar nuestro directorio de trabajo?
La orden para movernos por la estructura de directorios es *cd*, abreviación de "cambio de directorio". Hay que destacar, que la mayoría de las ordenes UNIX mas usadas son de dos o tres letras. La forma de uso de la orden cd es:

cd <directorio>

donde <directorio> es el nombre del directorio al que queremos ir.
Como dijimos, al entrar al sistema comenzamos en el directorio "home". Si madhawk quiere ir al subdirectorio papers, debería usar la orden

```
/home/madhawk# cd papers
/home/madhawk/papers#
```

Como se puede ver, la linea de comandos de madhawk cambia para mostrar su directorio actual de trabajo. Ahora que ya esta en el directorio papers puede echarle un vistazo a su fichero history-final con el comando more

```
/home/madhawk/papers# more history-final
```

Ahora madhawk esta en el subdirectorio papers, para volver al directorio padre de este, usara la orden *cd* ..

```
/home/madhawk/papers# cd ..
/home/madhawk#
```

(Dese cuenta del espacio entre "cd" y ".."). Cada directorio tiene una entrada de nombre ".." la cual se refiere al directorio padre. De igual forma, existe en cada directorio la entrada "." la cual se refiere a si mismo. Así que el comando nos deja donde estamos.

```
/home/madhawk/papers# cd .
/home/madhawk#
```

También pueden usarse nombres con el camino absoluto en la orden cd. Para ir al directorio de ZipZero con cd, introduciremos la siguiente orden.

```
/home/madhawk/papers# cd /home/zipzero
/home/zipzero#
```

También, usando cd sin argumentos nos llevara a nuestro directorio de origen.

```
/home/zipzero# cd
/home/madhawk#
```

### 3.2 Mirando el contenido de los directorios

Ahora que ya sabe como moverse por los directorios probablemente pensara: ¿Y bien? El simple movimiento por el árbol de directorios es poco útil, necesitamos un nuevo comando, *ls*. ls muestra por el terminal la lista de ficheros y directorios, por defecto, los del directorio activo. Por ejemplo;

```
/home/madhawk# ls
Mail
letters
papers
```

```
/home/madhawk#
```

Aqui podemos ver que madhawk tiene tres entradas en su directorio actual: Mail, letters y papers. Esto no nos dice demasiado ¿son ficheros o directorios?. Podemos usar la opción *-F* de la orden ls para obtener mas información.

```
/home/madhawk# ls -F
Mail/
letters/
papers/
/home/madhawk#
```

Por el carácter / añadido a cada nombre sabemos que las tres entradas son subdirectorios.
La orden ls -F puede también añadir al final "*", esto indica que es un fichero ejecutable. Si ls -F no añade nada, entonces es un fichero normal, es decir no es ni un directorio ni un ejecutable.
Por lo general cada orden UNIX puede tomar una serie de opciones definidas en forma de argumentos. Estos usualmente comienzan con el carácter "-", como vimos antes con ls -F. La opción -F le dice a ls que de mas información sobre el tipo de ficheros en este caso añadiendo un / detrás de cada nombre de un directorio.
Si a ls le pasamos un nombre de directorio, mostrara el contenido de ese directorio.

```
/home/madhawk# ls -F papers
english-lit
history-final
masters-thesis
notes/
/home/madhawk#
```

Para ver un listado mas interesante, veamos el contenido de directorio del sistema /etc.

```
/home/madhawk# ls /etc
Images          ftpusers            lpc          rc.new          shells
adm             getty          magic          rc0.d          startcons
bcheckrc        gettydefs      motd           rc1.d          swapoff
brc             group          mount          rc2.d          swapon
brc~            inet           mtab           rc3.d          syslog.conf
csh.cshrc       init           mtools         rc4.d          syslog.pid
csh.login       init.d         pac            rc5.d          syslogd.reload
efault          initrunlvl     passwd         rmt            termcap
disktab         inittab        printcap       rpc            umount
fdprm           inittab.old    profile        rpcinfo        update
fstab           issue          psdatabase     securetty      utmp
ftpaccess       lilo           rc             services       wtmp
/home/madhawk#
```

(Para los usuarios de MS-DOS, nótese que los nombres de los ficheros pueden ser mayores de caracteres y pueden contener puntos en cualquier posición. Incluso es posible que un fichero contenga mas de un punto en su nombre.)
Vayamos al directorio raíz con "cd .." y desde allí vayamos al directorio /usr/bin.

```
/home/madhawk# cd ..
/home# cd ..
/# cd usr
/usr# cd bin
/usr/bin#
```

También podemos movernos dentro de directorios en múltiples pasos, como en cd /usr/bin.
Trate de moverse por varios directorios usando ls y cd. En algunos casos podrá encontrarse el desagradable mensaje de error "Permission denied". Esto simplemente es debido a cuestiones de seguridad del UNIX. Para poder moverse o listar un directorio debe de tener permisos para poder hacerlo. Hablaremos mas sobre ello en la Sección 9.

**3.3 Creando directorios nuevos**

Es el momento de aprender a crear directorios. Para ello se usa la orden *mkdir*. Pruebe lo siguiente:

```
/home/madhawk# mkdir foo
/home/madhawk# ls -F
Mail/
foo/
letters/
papers/
/home/madhawk# cd foo
/home/madhawk/foo# ls
/home/madhawk/foo#
```

¡Enhorabuena! Acaba de crear un directorio nuevo y moverse a el. Como no hay ningún fichero en el directorio nuevo, veamos como copiar ficheros desde un lugar a otro.

**3.4 Copia de ficheros**

La copia de ficheros es efectuada por la orden *cp*:

```
/home/madhawk/foo# cp /etc/termcap  .
/home/madhawk/foo# cp /etc/shells  .
/home/madhawk/foo# ls -F
shells      termcap
/home/madhawk/foo# cp shells bells
/home/madhawk/foo# ls -F
bells      shells      termcap
/home/madhawk/foo#
```

La orden cp copia los ficheros listados en la linea de comandos al fichero o directorio pasado como ultimo argumento. Nótese como se usa el directorio "." para referirnos al directorio actual.

**3.5 Moviendo ficheros**

La orden *mv* mueve ficheros en lugar de copiarlos. La sintaxis es muy sencilla.

```
/home/madhawk/foo# mv termcap sells
/home/madhawk/foo# ls -F
bells      sells      shells
/home/madhawk/foo#
```

Nótese como termcap ya no existe, en su lugar esta el fichero sells.  Esta orden puede usarse para renombrar ficheros, como acabamos de hacer, pero también para mover ficheros a directorios diferentes.

Nota: mv y cp sobreescribirán los ficheros destino (si ya existen) sin consultar. Sea cuidadoso cuando mueva un fichero a otro directorio: puede haber ya un fichero con el mismo nombre que será sobrescrito.

**3.6 Borrando ficheros y directorios**

Para borrar un fichero, use la orden *rm*. ("rm" viene de "remove").

```
/home/madhawk/foo# rm bells sells
/home/madhawk/foo# ls -F
shells
/home/madhawk/foo#
```

Nos hemos quedado solo con el fichero "shells", pero no nos quejaremos. Nótese que rm por defecto no preguntara antes de borrar un fichero luego, sea cuidadoso.

Una orden relacionada con rm es *rmdir*. Esta orden borra un directorio, pero solo si esta vacío. Si el directorio contiene ficheros o subdirectorios, rmdir se quejara.

**3.7 Mirando los ficheros**

Las ordenes *more* y *cat* son usadas para ver el contenido de ficheros. more muestra el fichero

pantalla a pantalla mientras que cat lo muestra entero de una vez.
Para ver el contenido del fichero shells podemos usar la orden

/home/madhawk/foo# more shells

Por si esta interesado en el contenido de shells, es una lista de interpretes de comandos validos disponibles en el sistema. En la mayoría de los sistemas incluye /bin/sh, /bin/bash y /bin/csh. Hablaremos sobre los diferentes interpretes de comandos mas adelante.
Durante la ejecución de more pulse |_Space_ | para avanzar a la pagina siguiente y |_b_| para volver a la pagina anterior. Hay otros comandos disponibles, los citados son solo los mas básicos. |_q_| finalizara la ejecución de more.
Salga de more y pruebe cat /etc/termcap. El texto probablemente pasara demasiado rápido como para poder leerlo. El nombre "cat" viene de "concaténate", que es para lo que realmente sirve el programa. La orden cat puede ser usada para concatenar el contenido de varios ficheros y guardar el resultado en otro fichero. Esto se discutirá mas adelante.

### 3.8 Obteniendo ayuda en línea

Prácticamente cada sistema UNIX, incluido Linux, proporciona una utilidad conocida como "paginas de manual". Estas paginas contienen documentación en línea para todas las ordenes del sistema, recursos, ficheros de configuración, etc.
La orden usada para acceder a las paginas de manual es *man*. Por ejemplo, si esta interesado en conocer otras opciones de la orden ls, puede escribir

/home/madhawk# man ls

y le será mostrada la pagina de manual para ls.
Desafortunadamente la mayoría de las paginas de manual han sido escritas por gente que ya conocía lo que la orden o recurso hacia, por esto, las paginas de manual usualmente solo contienen detalles técnicos de la orden sin ningún tipo de tutorial de uso. Pese a esto, estas paginas son una gran fuente de información que permiten refrescar la memoria si olvidamos la sintaxis de un comando. Igualmente, estas paginas le darán mucha información sobre ordenes que no trataremos en este tutorial.
Le sugiero que pruebe man con los comandos que ya hemos tratado y con los que vayamos introduciendo. Notara que alguno de los comandos no tiene pagina de manual. Esto puede ser debido a diferentes motivos. En primer lugar, las paginas no han sido escritas aun (el Proyecto de Documentación de Linux es también el responsable de las paginas de manual). En segundo lugar, la orden puede ser interna del interprete de comandos, o un alias (como los tratados en la Sección 2.4), en cuyo caso no tendrán una pagina propia. Un ejemplo es la orden cd la cual es interna del interprete de comandos. El propio interprete de comandos es quien procesa cd no hay un programa separado.

## 4 Sumario de Ordenes Básicas

Esta sección introduce algunos de las ordenes básicas mas útiles de un sistema UNIX, incluidas las ya cubiertas en las secciones anteriores.
Nótese que las opciones usualmente comienzan con "-" y en la mayoría de los casos se pueden añadir múltiples opciones de una letra con un único "-". Por ejemplo, en lugar de usar ls -l -F es posible usar ls -lF.
En lugar de listar todas las opciones disponibles para cada uno de los comandos solo hablaremos de aquellas mas útiles o importantes. De hecho, la mayoría de las ordenes tienen un gran numero de opciones (muchas de las cuales nunca usara). Puede usar man para ver las paginas de manual de cada orden, la cual mostrara la lista completa de opciones disponibles.
Nótese también, que la mayoría de las ordenes toman una lista de ficheros o directorios como argumentos, denotados como "<fichero1> . . . <ficheroN>". Por ejemplo, la orden cp toma como argumentos la lista de ficheros a copiar, seguidos del fichero o directorio destino. Cuando se copia mas de un fichero, el destino debe de ser un directorio.

*Cd* *Cambia el directorio de trabajo actual.*

Sintaxis: cd <directorio>

<directorio> es el directorio al que cambiamos. ("." se refiere al directorio actual, ".." al directorio padre.)

Ejemplo: cd ../foo pone ../foo como directorio actual.

*Ls* *Muestra información sobre los ficheros o directorios indicados.*

Sintaxis: ls <fichero1> <fichero2> ...<ficheroN>

Donde <fichero1> a <ficheroN> son los ficheros o directorios a listar.

Opciones: Hay mas opciones de las que podría suponer. Las mas usadas comúnmente son:
-F (usada para mostrar información sobre el tipo de fichero),
-l (da un listado "largo" incluyendo tamaño, propietario, permisos, etc.)

Ejemplo: ls -lF /home/madhawk mostrara el contenido del directorio /home/madhawk.

*Cp* *Copia fichero(s) en otro fichero o directorio.*

Sintaxis: cp <fichero1> <fichero2> ...<ficheroN> <destino>

Donde <fichero1> a <ficheroN> son los ficheros a copiar, y <destino> es el fichero o directorio destino.

Ejemplo: cp ../frog joe copia el fichero ../frog al fichero o directorio joe.

*Mv* *Mueve fichero(s) a otro fichero o directorio.*

Es equivalente a una copia seguida del borrado del original. Puede ser usado para renombrar ficheros, como el comando MS-DOS RENAME.

Sintaxis: mv <fichero1> <fichero2> ...<ficheroN> <destino>

Donde <fichero1> a <ficheroN> son los ficheros a "mover" y <destination> es el fichero o directorio destino.

Ejemplo: mv ../frog joe mueve el fichero ../frog al fichero o directorio joe.

*rm* *Borra ficheros.*

Nótese que cuando los ficheros son borrados en UNIX, son irrecuperables (a diferencia de MS-DOS, donde usualmente se puede recuperar un fichero borrado).

Sintaxis: rm <fichero1> <fichero2> ...<ficheroN>

Donde <fichero1> a <ficheroN> son los nombres de los ficheros a borrar.

Opciones:
i pedirá confirmación antes de borrar un fichero.

Ejemplo: rm -i /home/madhawk/joe /home/madhawk/frog borra los ficheros joe y frog en /home/madhawk.

*Mkdir* *Crea directorios nuevos.*

Sintaxis: mkdir <dir1> <dir2> ...<dirN>

Donde <dir1> a <dirN> son los directorios a crear.

Ejemplo: mkdir /home/madhawk/test crea el directorio test colgando de /home/madhawk.

*Rmdir* *Esta orden borra directorios vacíos.*

Al usar rmdir, el directorio de trabajo actual no debe de estar dentro del directorio a borrar.

Sintaxis: rmdir <dir1> <dir2> ...<dirN>

Donde <dir1> a <dirN> son los directorios a borrar.

Ejemplo: rmdir /home/madhawk/papers borra el directorio /home/madhawk/papers si esta vacío.

*Man* *Muestra la pagina de manual del comando o recurso (cualquier utilidad del sistema que no es un comando, como funciones de librería) dado*.

Sintaxis: man <command>

Donde <command> es el nombre del comando o recurso sobre el que queremos obtener la ayuda

Ejemplo: man ls muestra ayuda sobre la orden ls.

*More* *Muestra el contenido de los ficheros indicados, una pantalla cada vez.*

Sintaxis: more <fichero1> <fichero2> ...<ficheroN>

Donde <fichero1> a <ficheroN> son los ficheros a mostrar.

Ejemplo: more papers/history-final muestra por el terminal el contenido del fichero papers/history-final.

Cat Oficialmente usado para concatenar ficheros, cat también es usado para mostrar el contenido

completo de un fichero de una vez.
Sintaxis: cat <fichero1> <fichero2> ...<ficheroN>
Donde <fichero1> a <ficheroN> son los ficheros a mostrar.
Ejemplo: cat letters/from-mdw muestra por el terminal el contenido del fichero letters/from-mdw.

Echo Simplemente envía al terminal los argumentos pasados.
Sintaxis: echo <arg1> <arg2> ...<argN>
Donde <arg1> a <argN> son los argumentos a mostrar.
Ejemplo: echo ``Hola mundo" muestra la cadena "Hola mundo".

Grep Muestra todas las líneas de un fichero dado que coinciden con un cierto patrón.
Sintaxis: grep <patrón> <fichero1> <fichero2> ...<ficheroN>
Donde <patrón> es una expresión regular y <fichero1> a <ficheroN> son los ficheros donde buscar.
Ejemplo: grep loomer /etc/hosts mostrara todas las líneas en el fichero /etc/hosts que contienen la cadena "loomer".

**5 Explorando el Sistema de Ficheros**

El sistema de ficheros es la colección de ficheros y la jerarquía de directorios de su sistema. Le prometimos acompañarle por el sistema de ficheros, y ha llegado el momento.
Tiene el nivel y conocimientos para entender de lo que estamos hablando, además de una guía de carreteras. (Figura 3.1 en la pagina 3 y 4).
Primero cambie al directorio raíz (cd /) y ejecute ls -F. Probablemente vera estos directorios:
bin, dev, etc, home, install, lib, mnt, proc, root, tmp, user, usr, y var.
Echemos un vistazo a cada uno de estos directorios.

/bin /bin es la abreviación de "binaries", o ejecutables. Es donde residen la mayoría de los programas esenciales del sistema. Use la orden "ls -F /bin" para listar los ficheros. Podrá ver algunas ordenes que reconocerá, como cp, ls y mv. Estos son los programas para estas ordenes. Cuando usa la orden cp esta ejecutando el programa /bin/cp.

Usando ls -F vera que la mayoría (si no todos) los ficheros de /bin tienen un asterisco ("*") añadido al final de sus nombres. Esto indica que son ficheros ejecutables, como describe la Sección 3.2.

/dev El siguiente es /dev. Echémosle un vistazo de nuevo con ls -F. Los "ficheros" en /dev son conocidos como controladores de dispositivo (device drivers) son usados para acceder a los dispositivos del sistema y recursos, como discos duros, modems, memoria, etc. Por ejemplo, de la misma forma que puede leer datos de un fichero, puede leerla desde la entrada del raton leyendo / dev/mouse.

Los ficheros que comienzan su nombre con fd son controladores de disqueteras. fd0 es la primera disquetera, fd1 la segunda. Ahora, alguien astuto se dará cuenta de que hay mas controladores de dispositivo para disqueteras de los que hemos mencionado. Estos representan tipos específicos de discos. Por ejemplo, fd1H1440 accederá a discos de 3.5" de alta densidad en la disquetera 1. Aqui tenemos una lista de algunos de los controladores de dispositivo mas usados. Nótese que incluso aunque puede que no tenga alguno de los dispositivos listados, tendrá entradas en dev de cualquier forma.

/dev/console hace referencia a la consola del sistema es decir, al monitor conectado directamente a su sistema.
Los dispositivos /dev/ttyS y /dev/cua son usados para acceder a los puertos serie. Por ejemplo, / dev/ttyS0 hace referencia a "COM1" bajo MS-DOS. Los dispositivos /dev/cua son "callout", los cuales son usados en conjunción con un módem.
Los nombres de dispositivo que comienzan por hd acceden a discos duros.
/dev/hda hace referencia a la totalidad del primer disco duro, mientras que /dev/hda1 hace referencia a la primera partición en /dev/hda.
Los nombres de dispositivo que comienzan con sd son dispositivos SCSI. Si tiene un disco duro SCSI, en lugar de acceder a el mediante /dev/hda, deberá acceder a /dev/sda. Las cintas SCSI son

accedidas vía dispositivos st y los CD-ROM SCSI vía sr.
Los nombres que comienzan por lp acceden a los puertos paralelo. /dev/lp0 hace referencia a "LPT1" en el mundo MS-DOS.
/dev/null es usado como "agujero negro" cualquier dato enviado a este dispositivo desaparece. ¿Para que puede ser útil esto?. Bien, si desea suprimir la salida por pantalla de una orden, podría enviar la salida a /dev/null. Hablaremos mas sobre esto después.
Los nombres que comienzan por /dev/tty hacen referencia a "consolas virtuales" de su sistema (accesibles mediante las teclas |_alt-F1_,||_alt-F2_,|etc). /dev/tty1 hace referencia a la primera VC, /dev/tty2 a la segunda, etc.
Los nombres de dispositivo que comienzan con /dev/pty son "pseudo-terminales". Estos son usados para proporcionar un "terminal" a sesiones remotas. Por ejemplo, si su maquina esta en una red, telnet de entrada usara uno de los dispositivos /dev/pty.
/etc /etc contiene una serie de ficheros de configuración del sistema. Estos incluyen /etc/passwd (la base de datos de usuarios), /etc/rc (guiones de inicialización del sistema), etc.
/sbin sbin se usa para almacenar programas esenciales del sistema, que usara el administrador del sistema.
/home /home contiene los directorios "home" de los usuarios. Por ejemplo, /home/madhawk es el directorio del usuario "madhawk". En un sistema recién instalado, no habrá ningún usuario en este directorio.
/lib /lib contiene las imágenes de las librerías compartidas. Estos ficheros contienen código que compartirán muchos programas. En lugar de que cada programa contenga una copia propia de las rutinas compartidas, estas son guardadas en un lugar común, en /lib. Esto hace que los programas ejecutables sean menores y reduce el espacio usado en disco.
/proc /proc es un "sistema de ficheros virtual". Los ficheros que contiene realmente residen en memoria, no en un disco. Hacen referencia a varios procesos que corren en el sistema, y le permiten obtener información acerca de que programas y procesos están corriendo en un momento dado. Entraremos en mas detalles en la sección 11.1.
/tmp Muchos programas tienen la necesidad de generar cierta información temporal y guardarla en un fichero temporal. El lugar habitual para esos ficheros es en /tmp.
/usr /usr es un directorio muy importante. Contienen una serie de subdirectorios que contienen a su vez algunos de los mas importantes y útiles programas y ficheros de configuración usados en el sistema.
Los directorios descritos arriba son esenciales para que el sistema este operativo, pero la mayoría de las cosas que se encuentran en /usr son opcionales para el sistema. De cualquier forma, son estas cosas opcionales las que hacen que el sistema sea útil e interesante. Sin /usr, tendría un sistema aburrido, solo con programas como cp y ls. usr contiene la mayoría de los paquetes grandes de programas y sus ficheros de configuración.

- /usr/X386 — /usr/X386 contiene el sistema X Window si usted lo instala. El sistema X Window es un entorno gráfico grande y potente el cual proporciona un gran numero de utilidades y programas gráficos, mostrados en "ventanas" en su pantalla. Si esta familiarizado con los entornos Microsoft Windows o Macintosh, X Window le será muy familiar. El directorio /usr/X386 contiene todos los ejecutables de X Window, ficheros de configuración y de soporte.
- /usr/bin — /usr/bin es el almacén real de programas del sistema UNIX. Contiene la mayoría de los programas que no se encuentran en otras partes como /bin.
- /usr/etc — Como /etc contiene diferentes ficheros de configuración y programas del sistema, /usr/etc contiene incluso mas que el anterior. En general, los ficheros que se encuentran en /usr/etc/ no son esenciales para el sistema, a diferencia de los que se encuentran en /etc, que si lo son.
- /usr/include — /usr/include contiene los ficheros de cabecera para el compilador de C. Estos ficheros (la mayoría de los cuales terminan en .h, de "header") declaran estructuras de datos, subrutinas y constantes usados en la escritura de programas en C. Los ficheros que se encuentran en /usr/include/sys son

generalmente usados en la programación de UNIX a nivel de sistema. Si esta familiarizado con el lenguaje de programación C, aqui encontrara los ficheros de cabecera como stdio.h, el cual declara funciones como printf().

/usr/g++-include /usr/g++-include contiene ficheros de cabecera para el compilador de C++ (muy parecido a /usr/include).

/usr/lib /usr/lib contiene las librerías equivalentes "stub" y "static" a los ficheros encontrados en /lib. Al compilar un programa, este es "enlazado" con las librerias que se encuentran en /usr/lib, las cuales dirigen al programa a buscar en /lib cuando necesita el código de la librería. Además, varios programas guardan ficheros de configuración en /usr/lib.

/usr/local /usr/local es muy parecido a /usr contiene programas y ficheros no esenciales para el sistema, pero que hacen el sistema mas divertido y excitante. En general, los programas que se encuentran en /usr/local son específicos de su sistema esto es, el directorio /usr/local difiere bastante entre sistemas UNIX. Aqui encontrara programas grandes como TEX (sistema de formateo de documentos) y Emacs (gran y potente editor), si los instala.

/usr/man Este directorio contiene las paginas de manual. Hay dos subdirectorios para cada pagina "sección" de las paginas (use la orden man man para mas detalles). Por ejemplo, /usr/man/man1 contiene los fuentes (es decir, los originales por formatear) de las paginas de manual de la sección 1, y / usr/man/cat1 las paginas ya formateadas de la sección 1.

/usr/src /usr/src contiene el código fuente (programas por compilar) de varios programas de su sistema. El mas importante es /usr/src/Linux, el cual contiene el código fuente del Nucleo de Linux.

/var /var contiene directorios que a menudo cambian su tamaño o tienden a crecer. Muchos de estos directorios solían residir en /usr, pero desde que estamos tratando de dejarlo relativamente inalterable, los directorios que cambian a menudo han sido llevados a /var. Algunos de estos directorios son:

/var/adm /var/adm contiene varios ficheros de interés para el administrador del sistema, específicamente históricos del sistema, los cuales recogen errores o problemas con el sistema. Otros ficheros guardan las sesiones de presentación en el sistema, así como los intentos fallidos.

/var/spool /var/spool contiene ficheros que van a ser pasados a otro programa. Por ejemplo, si su maquina esta conectada a una red, el correo de llegada será almacenado en /var/spool/Mail hasta que lo lea o lo borre. Artículos nuevos de las news" tanto salientes como entrantes pueden encontrarse en / var/spool/news, etc.

**6 Tipos de interpretes de Comandos**

Como hemos mencionado anteriormente en numerosas ocasiones, UNIX es un sistema operativo multitarea y multiusuario. La multitarea es muy útil, y una vez la haya probado, la usara continuamente. En poco tiempo podrá ejecutar programas "de fondo", conmutar entre múltiples tareas y "entubar" programas unos entre otros para conseguir resultados complejos con un único comando.

Muchas de las características que trataremos en esta sección son proporcionadas por el interprete de comandos. Hay que tener cuidado en no confundir UNIX (el sistema operativo) con el interprete de comandos, este ultimo, es un interface con el sistema que hay debajo. El interprete de comandos proporciona la funcionalidad sobre el UNIX

El interprete de comandos no es solo un interprete interactivo de los comandos que tecleamos, es también un potente lenguaje de programación, el cual permite escribir guiones, que permiten juntar varias ordenes en un fichero. Los usuarios de MS-DOS reconocerán esto como los ficheros "batch". El uso de los guiones del interprete de comandos es una herramienta muy potente que le permitirá automatizar e incrementar el uso de UNIX. Ver la sección 13.1 para mas información.

Hay varios tipos de interpretes de comandos en el mundo UNIX. Los dos mas importantes son el "Bourne shell" y el "C shell". El interprete de comandos Bourne, usa una sintaxis de comandos como la usada en los primeros sistemas UNIX, como el System III. El nombre del interprete Bourne en la mayoría de los UNIX es /bin/sh (donde sh viene de "shell", interprete de comandos en ingles). El interprete C usa una sintaxis diferente, a veces parecida a la del lenguaje de programación C, y en la mayoría de los sistemas UNIX se encuentra como /bin/csh.
Bajo Linux hay algunas diferencias en los interpretes de comandos disponibles. Dos de los mas usados son el "Bourne Again Shell" o "Bash" (/bin/bash) y Tcsh (/bin/tcsh). Bash es un equivalente al Bourne con muchas características avanzadas de la C shell. Como Bash es un súper conjunto de la sintaxis del Bourne, cualquier guión escrito para el interprete de comandos Bourne standard funcionara en Bash. Para los que prefieren el uso del interprete de comandos C, Linux tiene el Tcsh, que es una versión extendida del C original.
El tipo de interprete de comandos que decida usar es puramente una cuestion de gustos. Algunas personas prefieren la sintaxis del Bourne con las características avanzadas que proporciona Bash, y otros prefieren el mas estructurado interprete de comandos C. En lo que respecta a los comandos usuales como cp, ls..etc, es indiferente el tipo de interprete de comandos usado, la sintaxis es la misma. Solo, cuando se escriben guiones para el interprete de comandos, o se usan características avanzadas aparecen las diferencias entre los diferentes interpretes de comandos.
Como estamos discutiendo sobre las diferencias entre los interpretes de comandos Bourne y C, abajo veremos esas diferencias. Para los propósitos de este tutorial, la mayoría de las diferencias son mínimas. (Si eres realmente curioso a este respecto, lee las paginas de manual para bash y tcsh).

**7 Caracteres Comodín**
Una característica importante de la mayoría de los interpretes de comandos en UNIX es la capacidad para referirse a mas de un fichero usando caracteres especiales. Estos llamados comodines le permiten referirse a, por ejemplo, todos los ficheros que contienen el carácter “n".
El comodín "*" hace referencia cualquier carácter o cadena de caracteres en el fichero. Por ejemplo, cuando usa el carácter "*" en el nombre de un fichero, el interprete de comandos lo sustituye por todas las combinaciones posibles provenientes de los ficheros en el directorio al cual nos estamos refiriendo.
Veamos un ejemplo rápido. Supongamos que madhawk tiene los ficheros frog, joe y stuff en el directorio actual.

```
/home/madhawk# ls
frog     joe     stuff
/home/madhawk#
```

Para acceder a todos los ficheros con la letra "o" en su nombre, hemos de usar la orden

```
/home/madhawk# ls *o*
frog     joe
/home/madhawk#
```

Como puede ver, el comodín "*" ha sido sustituido con todas las combinaciones posibles que coincidían de entre los ficheros del directorio actual.
El uso de "*" solo, simplemente se refiere a todos los ficheros, puesto que todos los caracteres coinciden con el comodín.

```
/home/madhawk# ls *
frog     joe     stuff
/home/madhawk#
```

Veamos unos pocos ejemplos mas.

```
/home/madhawk# ls f*
frog
/home/madhawk# ls *ff
stuff
/home/madhawk# ls *f*
frog     stuff
```

```
/home/madhawk# ls s*f
stuff
/home/madhawk#
```

El proceso de la sustitución de "*" en nombres de ficheros es llamado expansión de comodines y es efectuado por el interprete de comandos. Esto es importante: las ordenes individuales, como ls, nunca ven el "*" en su lista de parámetros. Es el interprete quien expande los comodines para incluir todos los nombres de ficheros que se adaptan. Luego la orden

```
/home/madhawk# ls *o*
```

es expandida para obtener

```
/home/madhawk# ls frog joe
```

Una nota importante acerca del carácter comodín "*". El uso de este comodín NO cuadrara con nombres de ficheros que comiencen con un punto ("."). Estos ficheros son tratados como "ocultos" aunque no están realmente ocultos, simplemente no son mostrados en un listado normal de ls y no son afectados por el uso del comodín "*".

He aqui un ejemplo. Ya hemos mencionado que cada directorio tiene dos entradas especiales: "." que hace referencia al directorio actual y ".." que se refiere al directorio padre. De cualquier forma, cuando use ls esas dos entradas no se mostraran.

```
/home/madhawk# ls
frog    joe    stuff
/home/madhawk#
```

Si usa el parámetro -a con ls podrá ver nombres de ficheros que comienzan por ".". Observe:

```
/home/madhawk# ls -a
.    ..    .bash_profile    .bashrc    frog    joe    stuff
/home/madhawk#
```

Ahora podemos ver las dos entradas especiales, "." y "..", así como otros dos ficheros "ocultos" . bash_profile y .bashrc. Estos dos ficheros son usados en el arranque por bash cuando madhawk se presenta al sistema. Mas información sobre esto en la Sección 13.3.

Note que cuando usamos el comodín "*", no se muestra ninguno de los nombres de fichero que comienzan por ".".

```
/home/madhawk# ls *
frog    joe    stuff
/home/madhawk#
```

Esto es una característica de seguridad: si "*" coincidiera con ficheros que comienzan por "." actuaría sobre "." y "..". Esto puede ser peligroso con ciertas ordenes.

Otro carácter comodín es "?". Este carácter comodín solo expande un único carácter. Luego "ls ?" mostrara todos los nombres de ficheros con un carácter de longitud, y "ls termca?" mostrara "termcap" pero no "termcap.backup". Aqui tenemos otro ejemplo:

```
/home/madhawk# ls j?e
joe
/home/madhawk# ls f??g
frog
/home/madhawk# ls ????f
stuff
/home/madhawk#
```

Como puede ver, los caracteres comodín le permiten referirse a mas de un fichero a la vez. En el resumen de ordenes en la Sección 4 dijimos que cp y mv pueden copiar o mover múltiples ficheros de una vez. Por ejemplo,

```
/home/madhawk# cp /etc/s* /home/madhawk
```

copiara todos los ficheros de /etc que comiencen por "s" al directorio /home/madhawk. Por lo tanto, el formato de la orden cp es realmente

```
cp <fichero1> <fichero2> <fichero3> ...<ficheroN> <destino>
```

donde <fichero1> a <ficheroN> es la lista de los ficheros a copiar, y <destino> es el fichero o directorio destino donde copiarlos. mv tiene idéntica sintaxis.

Nótese que si esta copiando o moviendo mas de un fichero, <destino> debe ser un directorio. Solo puede copiar o mover un único fichero a otro fichero.

## 8 Fontanería UNIX

### 8.1 Entrada y salida estándar

Muchos comandos UNIX toman su entrada de algo conocido como entrada estándar y envían su salida a la salida estándar (a menudo abreviado como "stdin" y "stdout"). El interprete de comandos configura el sistema de forma que la entrada estándar es el teclado y la salida la pantalla.
Veamos un ejemplo con el comando cat. Normalmente cat lee datos de los ficheros cuyos nombres se pasan como argumentos en la linea de comandos y envía estos datos directamente a la salida estándar. Luego, usando el comando

```
/home/madhawk/papers# cat history-final masters-thesis
```

mostrara por pantalla el contenido del fichero history-final seguido por masters-thesis.
Si no se le pasan nombres de ficheros a cat como parámetros, leerá datos de stdin y los enviara a stdout. Veamos un ejemplo.

```
/home/madhawk/papers# cat
Hello there.
Hello there.
Bye.
Bye._____
|_ctrl-D_ |
/home/madhawk/papers#
```

Como se puede ver, cada línea que el usuario teclea (impresa en itálica) es inmediatamente reenviada al monitor por cat. Cuando se esta leyendo de la entrada estándar, los comandos reconocen el fin de la entrada de datos cuando reciben el carácter EOT (end-of-text, fin de texto). Normalmente es generado con la combinación |_ctrl-D_.|
Veamos otro ejemplo. El comando sort toma como entrada líneas de texto (de nuevo leerá desde stdin si no se le proporcionan nombres de ficheros en la línea de comandos), y devuelve la salida ordenada a stdout. Pruebe lo siguiente:

```
/home/madhawk/papers# sort
bananas
carrots
apples
|_ctrl-D_ |
apples
bananas
carrots
/home/madhawk/papers#
```

Podemos ordenar alfabéticamente la lista de la compra... ¿no es útil UNIX?

### 8.2 Redireccionando la entrada y salida

Ahora, supongamos que queremos que la salida de sort vaya a un fichero para poder salvar la lista ordenada de salida. El interprete de comandos nos permite redireccionar la salida estándar a un fichero usando el símbolo ">". Veamos como funciona.

```
/home/madhawk/papers# sort > shopping-list
bananas
carrots
apples___
|_ctrl-D_ |
/home/madhawk/papers#
```

Como puede ver, el resultado de sort no se muestra por pantalla, en su lugar es salvado en el fichero shopping-list. Echemos un vistazo al fichero.

```
/home/madhawk/papers# cat shopping-list
```

```
apples
bananas
carrots
/home/madhawk/papers#
```

Ya podemos ordenar la lista de la compra y además guardarla.

Supongamos ahora que teníamos guardada nuestra lista de compra desordenada original en el fichero ítems. Una forma de ordenar la información y salvarla en un fichero podría ser darle a sort el nombre del fichero a leer en lugar de la entrada estándar y redireccionar la salida estándar como hicimos arriba.

```
/home/madhawk/papers# sort ítems > shopping-list
/home/madhawk/papers# cat shopping-list
apples
bananas
carrots
/home/madhawk/papers#
```

Hay otra forma de hacer esto. No solo puede ser redireccionada la salida estándar, también puede ser redireccionada la entrada estándar usando el símbolo "<".

```
/home/madhawk/papers# sort < items
apples
bananas
carrots
/home/madhawk/papers#
```

Técnicamente, sort < ítems es equivalente a sort ítems, pero nos permite demostrar que sort < ítems se comporta como si los datos del fichero fueran tecleados por la entrada estándar. El interprete de comandos es quien maneja las redirecciones. sort no recibe el nombre del fichero (ítems) a leer, desde el punto de vista de sort, esta leyendo datos de la entrada estándar como si fueran tecleados desde el teclado.

Esto introduce el concepto de filtro. Un filtro es un programa que lee datos de la entrada estándar, los procesa de alguna forma, y devuelve los datos procesados por la salida estándar. Usando la redirección la entrada estándar y/o salida estándar pueden ser referenciadas desde ficheros. sort es un filtro simple: ordena los datos de entrada y envía el resultado a la salida estándar. cat es incluso mas simple, no hace nada con los datos de entrada, simplemente envía a la salida cualquier cosa que le llega.

**8.3 Uso de tuberias (pipes)**

Ya hemos visto como usar sort como un filtro. Pero estos ejemplos suponen que tenemos los datos en un fichero en alguna parte o vamos a introducir los datos manualmente por la entrada estándar. ¿Que pasa si los datos que queremos ordenar provienen de la salida de otro comando, como ls?. Por ejemplo, usando la opción -r con sort ordenaremos los datos en orden inverso. Si queremos listar los ficheros en el directorio actual en orden inverso, una forma podría ser.

```
/home/madhawk/papers# ls
english-list
history-final
masters-thesis
notes
/home/madhawk/papers# ls > file-list
/home/madhawk/papers# sort -r file-list
notes
masters-thesis
history-final
english-list
/home/madhawk/papers#
```

Aqui, salvamos la salida de ls en un fichero, y entonces ejecutamos sort -r sobre ese fichero. Pero

esta forma necesita crear un fichero temporal en el que salvar los datos generados por ls.
La solución es usar las pipes. El uso de pipes es otra característica del interprete de comandos, que nos permite conectar una cadena de comandos en un "pipe", donde la stdout del primero es enviada directamente a la stdin del segundo y así sucesivamente. Queremos conectar la salida de ls con la entrada de sort. Para crear un pipe se usa el símbolo "|":

```
/home/madhawk/papers# ls | sort –r
notes
masters-thesis
history-final
english-list
/home/madhawk/papers#
```

Esta forma es mas corta y obviamente mas facil de escribir.
Otro ejemplo útil usando el comando

```
/home/madhawk/papers# ls /usr/bin
```

mostrara una lista larga de los ficheros, la mayoría de los cuales pasara rápidamente ante nuestros ojos sin que podamos leerla. En lugar de esto, usemos more para mostrar la lista de ficheros en /usr/bin.

```
/home/madhawk/papers# ls /usr/bin | more
```

Ahora podemos ir avanzando pagina a pagina cómodamente.
¡Pero la diversión no termina aqui!. Podemos "entubar" mas de dos comandos a la vez. El comando head es un filtro que muestra la primeras líneas del canal de entrada (aqui la entrada desde una pipe). Si queremos ver el ultimo fichero del directorio actual en orden alfabético, usaremos:

```
/home/madhawk/papers# ls | sort -r | head –1
notes
/home/madhawk/papers#
```

Donde head -1 simplemente muestra la primera línea de la entrada que recibe en este caso, el flujo de datos ordenados inversamente provenientes de ls).

### 8.4 Redirección no destructiva

El uso de ">" para redireccionar la salida a un fichero es destructivo: en otras palabras, el comando

```
/home/madhawk/papers# ls > file-list
```

sobreescribe el contenido del fichero file-list. Si en su lugar, usamos el símbolo ">>", la salida será añadida al final del fichero nombrado, en lugar de ser sobrescrito.

```
/home/madhawk/papers# ls >> file-list
```

añadirá la salida de ls al final de file-list.
Es conveniente tener en cuenta que la redirección y el uso de pipes son características proporcionadas por el interprete de comandos este, proporciona estos servicios mediante el uso de la sintaxis ">", ">>" y "|".

## 9 Permisos de Ficheros

### 9.1 Conceptos de permisos de ficheros

Al ser UNIX un sistema multiusuario, para proteger ficheros de usuarios particulares de la manipulación por parte de otros, UNIX proporciona un mecanismo conocido como permisos de ficheros. Este mecanismo permite que ficheros y directorios "pertenezcan" a un usuario en particular. Por ejemplo, como madhawk creo ficheros en su directorio "home", madhawk es el propietario de esos ficheros y tiene acceso a ellos.
UNIX también permite que los ficheros sean compartidos entre usuarios y grupos de usuarios.
Si madhawk lo desea, podría restringir el acceso a sus ficheros de forma que ningún otro usuario tenga acceso. De cualquier modo, en la mayoría de los sistemas por defecto se permite que otros usuarios puedan leer tus ficheros pero no modificarlos o borrarlos.
Como hemos explicado arriba, cada fichero pertenece a un usuario en particular. Por otra parte, los ficheros también pertenecen a un grupo en particular, que es un conjunto de usuarios definido por el sistema. Cada usuario pertenece al menos a un grupo cuando es creado. El administrador del

sistema puede hacer que un usuario tenga acceso a mas de un grupo.
Los grupos usualmente son definidos por el tipo de usuarios que acceden a la maquina. Por ejemplo, en un sistema UNIX de una universidad, los usuarios pueden ser divididos en los grupos estudiantes, dirección, profesores e invitados. Hay también unos pocos grupos definidos por el sistema (como bin y admin) los cuales son usados por el propio sistema para controlar el acceso a los recursos muy raramente los usuarios normales pertenecen a estos grupos.
Los permisos están divididos en tres tipos: lectura, escritura y ejecución. Estos permisos pueden ser fijados para tres clases de usuarios: el propietario del fichero, el grupo al que pertenece el fichero y para todos los usuarios independientemente del grupo.
El permiso de lectura permite a un usuario leer el contenido del fichero o en el caso de un directorio, listar el contenido del mismo (usando ls). El permiso de escritura permite a un usuario escribir y modificar el fichero. Para directorios, el permiso de escritura permite crear nuevos ficheros o borrar ficheros ya existentes en dicho directorio. Por ultimo, el permiso de ejecución permite a un usuario ejecutar el fichero si es un programa o guión del interprete de comandos. Para directorios, el permiso de ejecución permite al usuario cambiar al directorio en cuestion con cd.

**9.2 Interpretando los permisos de ficheros**
Veamos un ejemplo del uso de permisos de ficheros. Usando el comando ls con la opcion -l se mostrara un listado "largo" de los ficheros, el cual incluye los permisos de ficheros.

```
/home/madhawk/foo# ls -l stuff
-rw-r--r--  1 madhawk  users  505 Mar 13 19:05 stuff
/home/madhawk/foo#
```

El primer campo impreso en el listado representa los permisos de ficheros. El tercer campo es el propietario del fichero (madhawk), y el cuarto es el grupo al cual pertenece el fichero (users). Obviamente, el ultimo campo es el nombre del fichero (stuff), y los demás campos los trataremos mas adelante.
Este fichero pertenece a madhawk y al grupo users. Echemos un vistazo a los permisos. La cadena -rw-r--r-- nos informa, por orden, de los permisos para el propietario, el grupo del fichero y cualquier otro usuario.
El primer carácter de la cadena de permisos ("-") representa el tipo de fichero. El "-" significa que es un fichero regular. Las siguientes tres letras ("rw-") representan los permisos para el propietario del fichero, madhawk. El "r" para "lectura" y "w" para escritura. Luego madhawk tiene permisos de lectura y escritura para el fichero stuff.
Como ya mencionamos, aparte de los permisos de lectura y escritura esta el permiso de "ejecución", representado por una "x". Como hay un "-" en lugar del "x", significa que madhawk no tiene permiso para ejecutar ese fichero. Esto es correcto, puesto que stuff no es un programa de ningún tipo. Por supuesto, como el fichero es de madhawk, puede darse a si mismo permiso de ejecución si lo desea. Esto será cubierto en breve.
Los siguientes tres caracteres, r-- representan los permisos para los miembros del grupo. El grupo al que pertenece el fichero es users. Como solo aparece un "r" cualquier usuario que pertenezca al grupo users puede leer este fichero.
Los últimos tres caracteres, también r--, representan los permisos para cualquier otro usuario del sistema (diferentes del propietario o de los pertenecientes al grupo users). De nuevo, como solo esta presente el "r", los demás usuarios pueden leer el fichero, pero no escribir en el o ejecutarlo.
Aqui tenemos otros ejemplos de permisos de grupo.
-rwxr-xr-x El propietario del fichero puede leer, escribir y ejecutar el fichero. Los usuarios pertenecientes al grupo del fichero, y todos los demás usuarios pueden leer y ejecutar el fichero.
-rw------- El propietario del fichero puede leer y escribir. Nadie mas puede acceder al fichero.
-rwxrwxrwx Todos los usuarios pueden leer, escribir y ejecutar el fichero.

**9.3 Dependencias**
Es importante darse cuenta de que los permisos de un fichero también dependen de los permisos del directorio en el que residen. Por ejemplo, aunque un fichero tenga los permisos -rwxrwxrwx, otros

usuarios no podrán acceder a el a menos que también tengan permiso de lectura y ejecución para el directorio en el cual se encuentra el fichero. Si madhawk quiere restringir el acceso a todos sus ficheros, podría simplemente poner los permisos de su directorio "home" /home/madhawk a -rwx------. De esta forma ningún usuario podrá acceder a su directorio ni a ninguno de sus ficheros o subdirectorios. madhawk no necesita preocuparse de los permisos individuales de cada uno de sus ficheros.
En otras palabras, para acceder a un fichero, debes de tener permiso de ejecución de todos los directorios a lo largo del camino de acceso al fichero, además de permiso de lectura (o ejecución) del fichero en particular.
Habitualmente, los usuarios de un sistema UNIX son muy abiertos con sus ficheros. Los permisos que se dan a los ficheros usualmente son -rw-r--r--, lo que permite a todos los demás usuarios leer los ficheros, pero no modificarlos de ninguna forma. Los directorios, usualmente tienen los permisos -rwxr-xr-x, lo que permite que los demás usuarios puedan moverse y ver los directorios, pero sin poder crear o borrar nuevos ficheros en ellos.
Muchos usuarios pueden querer limitar el acceso de otros usuarios a sus ficheros. Poniendo los permisos de un fichero a -rw------- no se permitirá a ningún otro usuario acceder al fichero. Igualmente, poniendo los permisos del directorio a -rwx------ no se permitirá a los demás usuarios acceder al directorio en cuestion.

### 9.4 Cambiando permisos

El comando chmod se usa para establecer los permisos de un fichero. Solo el propietario puede cambiar los permisos del fichero. La sintaxis de chmod es:

```
chmod {a,u,g,o}{+,-}{r,w,x} <filenames>
```

Brevemente, indicamos a que usuarios afecta all, user, group o other. Entonces se especifica si se están añadiendo permisos (+) o quitándolos (-). Finalmente se especifica que tipo de permiso Read, write o execute. Algunos ejemplos:

| | |
|---|---|
| chmod a+r stuff | Da a todos los usuarios acceso al fichero. |
| chmod +r stuff | Como arriba si no se indica a, u, g, o por defecto se toma a. |
| chmod og-x stuff | Quita permisos de ejecución a todos los usuarios excepto al propietario. |
| chmod u+rwx stuff | Permite al propietario leer, escribir y ejecutar el fichero. |
| chmod o-rwx stuff | Quita permisos de lectura, escritura y ejecución a todos los usuarios menos al propietario y a los usuarios del grupo del fichero. |

## 10 Manejando enlaces de ficheros

Los enlaces le permiten dar a un único fichero múltiples nombres. Los ficheros son identificados por el sistema por su numero de inodo, el cual es el único identificador del fichero para el sistema de ficheros. Un directorio es una lista de números de inodo con sus correspondientes nombres de fichero. Cada nombre de fichero en un directorio es un enlace a un inodo particular.

### 10.1 Enlaces duros (Hard links)

La orden ln es usada para crear múltiples enlaces para un fichero. Por ejemplo, supongamos que tiene un fichero foo en un directorio. Usando ls -i, veremos el numero de inodo para el fichero.

```
# ls -i foo
22192 foo
#
```

Aqui, el fichero foo tiene el numero de inodo 22192 en el sistema de ficheros. Podemos crear otro enlace a foo, llamado bar:

```
# ln foo bar
```

Con ls -i veremos que los dos ficheros tienen el mismo inodo.

```
# ls -i foo bar
22192 bar    22192 foo
#
```

Ahora, accediendo a foo o a bar accederemos al mismo fichero. Si hace cambios en foo, estos

cambios también serán efectuados en bar. Para todos los efectos, foo y bar son el mismo fichero. Estos enlaces son conocidos como enlaces duros (Hard links) porque directamente crean el enlace al inodo. Nótese que solo podemos crear enlaces duros entre ficheros del mismo sistema de ficheros; enlaces simbólicos (ver mas adelante) no tienen esta restricción.
Cuando borra un fichero con rm, esta solamente borrando un enlace a un fichero. Si usa el comando

```
# rm foo
```

solo el enlace de nombre foo es borrado; bar todavía existirá. Un fichero es solo definitivamente borrado del sistema cuando no quedan enlaces a el. Usualmente, los ficheros tienen un único enlace, por lo que el uso de rm los borra. Pero si el fichero tiene múltiples enlaces, el uso de rm solo borrara un único enlace; para borrar el fichero, deberá borrar todos los enlaces del fichero.
La orden ls -l muestra el numero de enlaces a un fichero (entre otra información).

```
# ls -l foo bar
-rw-r--r--   2 root     root          12 Aug   5 16:51 bar
-rw-r--r--   2 root     root          12 Aug   5 16:50 foo
#
```

La segunda columna en el listado, "2", especifica el numero de enlaces al fichero.
Así resulta que un directorio no es mas que un fichero que contiene información sobre la translación enlace a inodo. También, cada directorio tiene al menos dos enlaces duros en el: "." (un enlace apuntando a si mismo) y ".." (un enlace apuntando al directorio padre). En el directorio raíz (/), el enlace ".." simplemente apunta a /.

### 10.2 Enlaces simbólicos

Los enlaces simbólicos son otro tipo de enlace, que es diferente al enlace duro. Un enlace simbólico permite dar a un fichero el nombre de otro, pero no enlaza el fichero con un inodo.
La orden ln -s crea un enlace simbólico a un fichero. Por ejemplo, si usamos la orden

```
# ln -s foo bar
```

crearemos un enlace simbólico bar apuntando al fichero foo. Si usamos ls -i, veremos que los dos ficheros tienen inodos diferentes, en efecto.

```
# ls -i foo bar
22195 bar    22192 foo
#
```

De cualquier modo, usando ls -l vemos que el fichero bar es un enlace simbólico apuntando a foo.

```
# ls -l foo bar
lrwxrwxrwx   1 root     root           3 Aug   5 16:51 bar -> foo
-rw-r--r--   1 root     root          12 Aug   5 16:50 foo
#
```

Los bits de permisos en un enlace simbólico no se usan (siempre aparecen como rwxrwxrwx). En su lugar, los permisos del enlace simbólico son determinados por los permisos del fichero "apuntado" por el enlace (en nuestro ejemplo, el fichero foo).
Funcionalmente, los enlaces duros y simbólicos son similares, pero hay algunas diferencias. Por una parte, puede crear un enlace simbólico a un fichero que no existe; lo mismo no es cierto para enlaces duros. Los enlaces simbólicos son procesados por el núcleo de forma diferente a los duros, lo cual es solo una diferencia técnica, pero a veces importante. Los enlaces simbólicos son de ayuda puesto que identifican al fichero al que apuntan; con enlaces duros no hay forma fácil de saber que fichero esta enlazado al mismo inodo.
Los enlaces se usan en muchas partes del sistema Linux. Los enlaces simbólicos son especialmente importantes para las imágenes de las librerías compartidas en /lib.

## 11 Control de Tareas

### 11.1 Tareas y procesos

Control de Tareas es una utilidad incluida en muchos shells (incluidas Bash y Tcsh), que permite el control de multitud de comandos o tareas al momento. Antes de seguir, deberemos hablar un poco sobre los procesos.

Cada vez que usted ejecuta un programa, usted lanza lo que se conoce como proceso, que es simplemente el nombre que se le da a un programa cuando se esta ejecutando. El comando ps visualiza la lista de procesos que se están ejecutando actualmente, por ejemplo:

```
/home/madhawk# ps
PID TT STAT   TIME COMMAND
24   3 S      0:03 (bash)
161   3 R      0:00 ps
/home/madhawk#
```

La columna PID representa el identificador de proceso. La ultima columna COMMAND, es el nombre del proceso que se esta ejecutando. Ahora solo estamos viendo los procesos que esta ejecutando madhawk. Vemos que hay dos procesos, bash (Que es el shell o interprete de comandos que usa madhawk), y el propio comando ps. Como puede observar, la bash se ejecuta concurrentemente con el comando ps. La bash ejecuto ps cuando madhawk tecleo el comando. Cuando ps termina de ejecutarse (después de mostrar la tabla de procesos), el control retorna al proceso bash, que muestra el prompt, indicando que esta listo para recibir otro comando.

Un proceso que esta corriendo se denomina tarea para el shell. Los términos proceso y tarea, son intercambiables. Sin embargo, se suele denominar "tarea" a un proceso, cuando es usado en conjunción con control de tareas, que es un rasgo del shell que permite cambiar entre distintas tareas.

En muchos casos, los usuarios solo ejecutan un trabajo cada vez, que es el ultimo comando que ellos teclearon desde el shell. Sin embargo, usando el control de tareas, usted podrá ejecutar diferentes tareas al mismo tiempo, cambiando entre cada uno de ellos conforme lo necesite. ¿Cuan beneficioso puede llegar a ser esto?. Supongamos que esta usted con su procesador de textos, y de repente necesita parar y realizar otra tarea, con el control de tareas, usted podrá suspender temporalmente el editor, y volver al shell para realizar cualquier otra tarea, y luego regresar al editor como si no lo hubiese dejado nunca. Lo siguiente solo es un ejemplo, hay montones de usos prácticos del control de tareas.

**11.2 Primer plano y Segundo plano**

Un proceso puede estar en Primer plano o en Segundo plano. Solo puede haber un proceso en primer plano al mismo tiempo, el proceso que esta en primer plano, es el que interactúa con usted recibe entradas de teclado, y envía las salidas al monitor. (Salvo, por supuesto, que haya re-dirigido la entrada o la salida, como se describe en la Sección 8). El proceso en segundo plano, no recibe ninguna señal desde el teclado por lo general, se ejecutan en silencio sin necesidad de interacción.

Algunos programas necesitan mucho tiempo para terminar, y no hacen nada interesante mientras tanto. Compilar programas es una de estas tareas, así como comprimir un fichero grande. No tiene sentido que se siente y se aburra mientras estos procesos terminan. En estos casos es mejor lanzarlos en segundo plano, para dejar el ordenador en condiciones de ejecutar otro programa.

Los procesos pueden ser suspendidos. Un proceso suspendido es aquel que no se esta ejecutando actualmente, sino que esta temporalmente parado. Después de suspender una tarea, puede indicar a la misma que continúe, en primer plano o en segundo, según necesite. Retomar una tarea suspendida no cambia en nada el estado de la misma la tarea continuara ejecutándose justo donde se dejo.

Tenga en cuenta que suspender un trabajo no es lo mismo que interrumpirlo. Cuando usted interrumpe un proceso (generalmente con la pulsación de |_ctrl-C_|), el proceso muere, y deja de estar en memoria y utilizar recursos del ordenador. Una vez eliminado, el proceso no puede continuar ejecutándose, y deberá ser lanzado otra vez para volver a realizar sus tareas. También se puede dar el caso de que algunos programas capturan la interrupción, de modo que pulsando |_ctrl-C_| no se para inmediatamente. Esto se hace para permitir al programa realizar operaciones necesarias de limpieza antes de terminar. De hecho, algunos programas simplemente no se dejan matar por ninguna interrupción.

**11.3 Envío a segundo plano y eliminación de procesos**

Empecemos con un ejemplo sencillo. El comando yes es un comando aparentemente inútil que envía una serie interminable de y-es a la salida estándar. (Realmente es muy útil. Si se utiliza una tubería (o "pipe") para unir la salida de yes con otro comando que haga preguntas del tipo si/no, la serie de y-es confirmara todas las preguntas.)

Pruebe con esto.

```
/home/madhawk# yes
y
y
y
y
y
```

La serie de y-es continuara hasta el infinito, a no ser que usted la elimine, pulsando la tecla de interrupción, generalmente |_ctrl-C_.| También puede deshacerse de esta serie de y-es redigiriendo la salida estándar de yes hacia /dev/null, que como recordara es una especie de "agujero negro" o papelera para los datos. Todo lo que usted envié allí, desaparecerá.

```
/home/madhawk# yes > /dev/null
```

Ahora va mucho mejor, el terminal no se ensucia, pero el prompt de la shell no retorna. Esto es porque yes sigue ejecutándose y enviando esos inútiles y-es a /dev/null. Para recuperarlo, pulse la tecla de interrupción.

Supongamos ahora que queremos dejar que el comando yes siga ejecutándose, y volver al mismo tiempo a la shell para trabajar en otras cosas. Para ello nos enviaremos a yes a segundo plano, lo que nos permitirá ejecutarlo, pero sin necesidad de interacción.

Una forma de mandar procesos a segundo plano es añadiendo un carácter "&" al final de cada comando.

```
/home/madhawk# yes > /dev/null &
[1] 164
/home/madhawk#
```

Como podrá ver, ha regresado a la shell. ¿Pero que es eso de "[1] 164"?, ¿se esta ejecutando realmente el comando yes?

"[1]" representa el numero de tarea del proceso yes. La shell asigna un numero a cada tarea que se este ejecutando. Como yes es el único comando que se esta ejecutando, se le asigna el numero de tarea 1. El numero "164" es el numero de identificación del proceso, o PID, que es el numero que el sistema le asigna al proceso. Ambos números pueden usarse para referirse a la tarea como veremos después.

Ahora usted tiene el proceso yes corriendo en segundo plano, y enviando constantemente la señal y hacia el dispositivo /dev/null. Para chequear el estado del proceso, utilice el comando interno de la shell jobs:

```
/home/madhawk# jobs
[1]+  Running                 yes >/dev/null   &
/home/madhawk#
```

¡Ahí esta!. También puede usar el comando ps, como mostramos antes, para comprobar el estado de la tarea.

Para eliminar una tarea, utilice el comando kill. Este comando toma como argumento un numero de tarea o un numero de ID de un proceso. Esta era la tarea 1, así que usando el comando

```
/home/madhawk# kill %1
```

matara la tarea. Cuando se identifica la tarea con el numero de tarea, se debe preceder el numero con el carácter de porcentaje (`%").

Ahora que ya hemos matado la tarea, podemos usar el comando jobs de nuevo para comprobarlo:

```
/home/madhawk# jobs
[1]+  Terminated              yes >/dev/null
/home/madhawk#
```

La tarea esta, en efecto, muerta, y si usa el comando jobs de nuevo, no mostrara nada.

También podrá matar la tarea usando el numero de ID de proceso (PID), el cual se muestra conjuntamente con el ID de tarea cuando arranca la misma. En nuestro ejemplo el ID de proceso es 164, así que el comando

```
/home/madhawk# kill 164
```

es equivalente a

```
/home/madhawk# kill %1
```

No es necesario usar el "%" cuando nos referimos a una tarea a través de su ID de proceso.

**11.4 Parada y relanzamiento de tareas**

Hay otra manera de poner una tarea en segundo plano. Usted puede lanzarlo como un proceso normal (en primer plano), pararlo, y después relanzarlo en segundo plano.

Primero, lance el proceso yes en primer plano como lo haría normalmente:

```
/home/madhawk# yes > /dev/null
```

De nuevo, dado que yes corre en primer plano, no debe retornar el prompt de la shell.

Ahora, en vez de interrumpir la tarea con |_ctrl-C_,| suspenderemos la tarea. El suspender una tarea no la mata: solamente la detiene temporalmente hasta que Ud. la retoma. Para hacer esto usted debe pulsar la tecla de suspender, que suele ser |_ctrl-Z_.|

```
/home/madhawk#_yes > /dev/null
|_ctrl-Z_ |
[1]+   Stopped                 yes >/dev/null
/home/madhawk#
```

Mientras el proceso esta suspendido, simplemente no se esta ejecutando. No gasta tiempo de CPU en la tarea. Sin embargo, usted puede retomar el proceso de nuevo como si nada hubiera pasado. Continuara ejecutándose donde se dejo.

Para relanzar la tarea en primer plano, use el comando *fg* (del ingles "foreground").

```
/home/madhawk# fg
yes >/dev/null
```

La shell muestra el nombre del comando de nuevo, de forma que tenga conocimiento de que tarea es la que ha puesto en primer plano. Pare la tarea de nuevo, con |_ctrl-Z_| Esta vez utilice el comando *bg* para poner la tarea en segundo plano. Esto hará que el comando siga ejecutándose igual que si lo hubiese hecho desde el principio con "&" como en la sección anterior.

```
/home/madhawk# bg
[1]+ yes >/dev/null &
/home/madhawk#
```

Y tenemos de nuevo el prompt. El comando jobs debería decirnos que yes se esta ejecutando, y podemos matar la tarea con kill tal y como lo hicimos antes.

¿Como podemos parar la tarea de nuevo? Si pulsa |_ctrl-Z_| no funcionara, ya que el proceso esta en segundo plano. La respuesta es poner el proceso en primer plano de nuevo, con el comando fg, y entonces pararlo. Como puede observar podrá usar fg tanto con tareas detenidas, como con las que estén segundo plano.

Hay una gran diferencia entre una tarea que se encuentra en segundo plano, y una que se encuentra detenida. Una tarea detenida es una tarea que no se esta ejecutando, es decir, que no usa tiempo de CPU, y que no esta haciendo ningún trabajo (la tarea aun ocupa un lugar en memoria, aunque puede ser volcada a disco). Una tarea en segundo plano, se esta ejecutando, y usando memoria, a la vez que completando alguna acción mientras usted hace otro trabajo. Sin embargo, una tarea en segundo plano puede intentar mostrar texto en su terminal, lo que puede resultar molesto si esta intentando hacer otra cosa. Por ejemplo, si usted uso el comando

```
/home/madhawk# yes &
```

sin redirigir stdout a /dev/null, una cadena de y-es se mostraran en su monitor, sin modo alguno de interrumpirlo (no puede hacer uso de |_ctrl-C_| para interrumpir tareas en segundo plano). Para poder parar esas interminables y-es, tendría que usar el comando fg para pasar la tarea a primer plano, y entonces usar |_ctrl-C_| para matarla.

Otra observación. Normalmente, los comandos "fg" y "bg" actúan sobre el ultimo proceso parado

(indicado por un "+" junto al numero de tarea cuando usa el comando jobs). Si usted tiene varios procesos corriendo a la vez, podrá mandar a primer o segundo plano una tarea especifica indicando el ID de tarea como argumento de fg o bg, como en

```
/home/madhawk# fg %2
```

(para la tarea de primer plano numero 2), o

```
/home/madhawk# bg %3
```

(para la tarea de segundo plano numero 3). No se pueden usar los ID de proceso con fg o bg. Además de esto, si usa el numero de tarea por si solo, como

```
/home/madhawk# %2
```

es equivalente a

```
/home/madhawk# fg %2
```

Solo recordarle que el uso de control de tareas es una utilidad de la shell. Los comandos fg, bg y jobs son internos de la shell. Si por algún motivo usted utiliza una shell que no soporta control de tareas, no espere disponer de estos comandos.
Y además, hay algunos aspectos del control de tareas que difieren entre Bash y Tcsh. De hecho, algunas shells no proporcionan ningún control de tareas sin embargo, la mayoría de las shells disponibles para Linux soportan control de tareas.

## 12 Usando el editor vi

Un editor de texto es simplemente un programa usado para la edición de ficheros que contienen texto, como una carta, un programa en C, o un fichero de configuración del sistema. Mientras que hay muchos editores de texto disponibles en Linux, el único editor que esta garantizado encontrar en cualquier sistema UNIX es vi el "visual editor". vi no es el editor mas fácil de usar, ni es muy auto explicativo. De cualquier forma, como es tan común en el mundo UNIX y es posible que alguna vez necesite usarlo, aqui encontrara algo de documentación.

La elección de un editor es principalmente una cuestión de gusto personal y estilo. Muchos usuarios prefieren el barroco, autoexplicativo y potente Emacs un editor con mas características que cualquier otro programa único en el mundo UNIX. Por ejemplo, Emacs tiene integrado su propio dialecto del lenguaje de programación LISP y tiene muchas extensiones (una de ellas es el programa "Eliza"- como programa de IA).
Pero como Emacs y todos sus ficheros de soporte es relativamente grande, puede que no tenga acceso a el en muchos sistemas. vi , por otra parte, es pequeño y potente, pero mas difícil de usar. De cualquier modo, una vez que conozca la forma de funcionamiento de vi, es muy fácil usarlo. Simplemente la curva de aprendizaje es bastante pronunciada al comienzo.

Esta sección es una Introducción coherente a vi no discutiremos todas sus características, solo aquellas necesarias para que sepa como comenzar. Puede dirigirse a la pagina de manual de vi si esta interesado en aprender mas acerca de las características de este editor, o puede leer el libro Learning the vi Editor de O'Reilly and Associates.

### 12.1 Conceptos

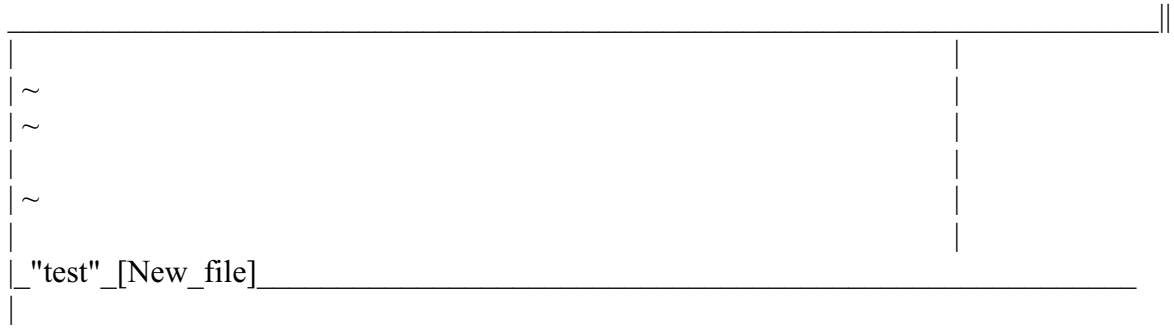
Mientras se usa vi, en cualquier momento estará en uno de tres posibles modos de operación. Estos modos son conocidos como modo ordenes, modo inserción y modo ultima linea.

Cuando inicia vi, esta en el modo ordenes. Este modo le permite usar ciertas ordenes para editar ficheros o cambiar a otros modos. Por ejemplo, tecleando "x" mientras esta en el modo ordenes, borra el carácter que hay debajo del cursor. Las teclas del cursor mueven este por el fichero que estamos editando. Generalmente, las ordenes usadas en este modo son solo de uno o dos caracteres de longitud.
Habitualmente insertara o editara texto desde el modo inserción. Usando vi, probablemente dedicara la mayor parte del tiempo en este modo. Inicia el modo de inserción al usar una orden como "i" (para "insertar") desde el modo de ordenes. Una vez en el modo de inserción, ira insertando texto en el documento desde la posición actual del cursor. Para salir del modo de inserción y volver al de

ordenes, pulse |_esc_|.
Modo ultima linea es un modo especial usado para proporcionar ciertas ordenes extendidas a vi. Al usar esos comandos, aparecen en la ultima linea de la pantalla (de ahí el nombre). Por ejemplo, cuando teclea ":" desde el modo de ordenes, entrara en el modo ultima linea, y podrá usar ordenes como "wq" (para escribir el fichero a disco y salir de vi), o "q!" (para salir de vi sin guardar los cambios). El modo de ultima linea es habitualmente usado por ordenes vi mayores de un carácter.
En el modo de ultima linea, introduce una orden de una sola linea y pulsa |_enter_| para ejecutarla.

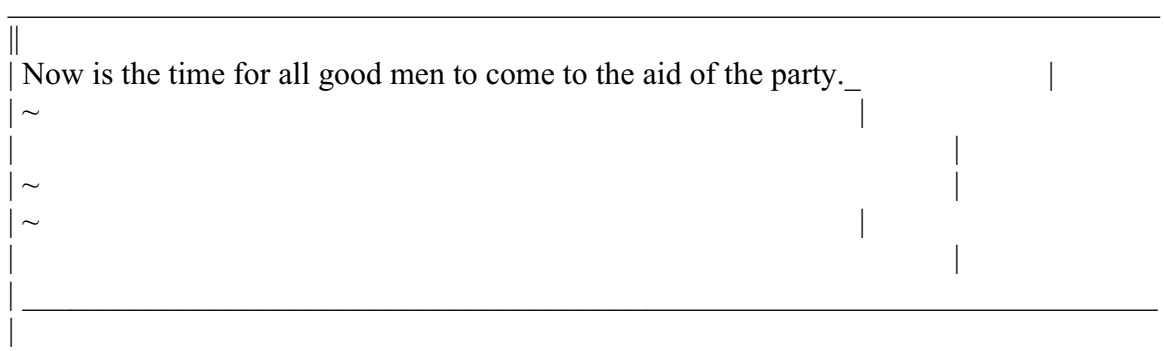
**12.2 Comenzando con vi**
La mejor forma de entender estos conceptos es arrancar vi y editar un fichero. En el ejemplo "screens" que veremos, vamos a mostrar solo unas pocas líneas de texto, como si la pantalla tuviese solo seis líneas de altura (en lugar de veinticuatro).

La sintaxis de vi es
vi <filename>
donde <filename> es el nombre del fichero que desea editar.
Arranque vi tecleando
/home/madhawk# vi test
lo que editara el fichero test. Debería ver algo como




La columna de caracteres "~" indica que esta al final del fichero.

**Insertando texto**
Esta ahora en modo ordenes; para poder insertar texto en el fichero, pulse |_i_| (lo que le hará entrar en modo inserción), y comience a escribir.

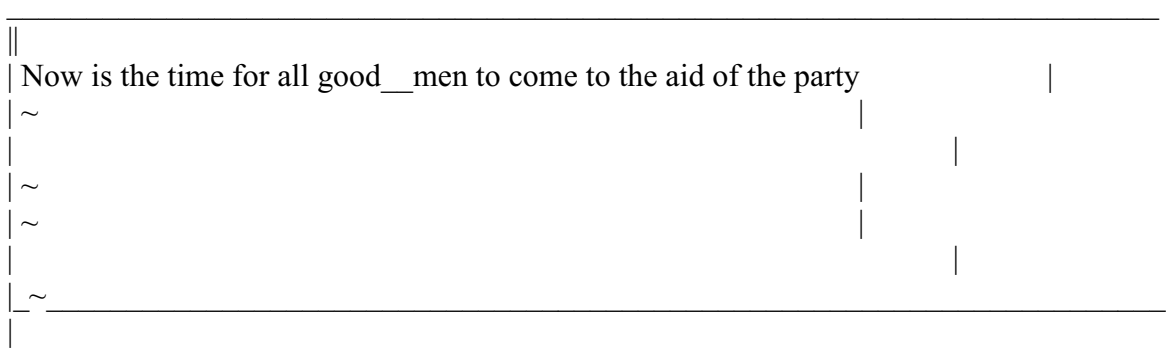



Mientras inserta texto, puede escribir tantas líneas como desee (pulsando |_return_ |después de cada una, por supuesto), y puede corregir los errores con la tecla de borrado de carácter.

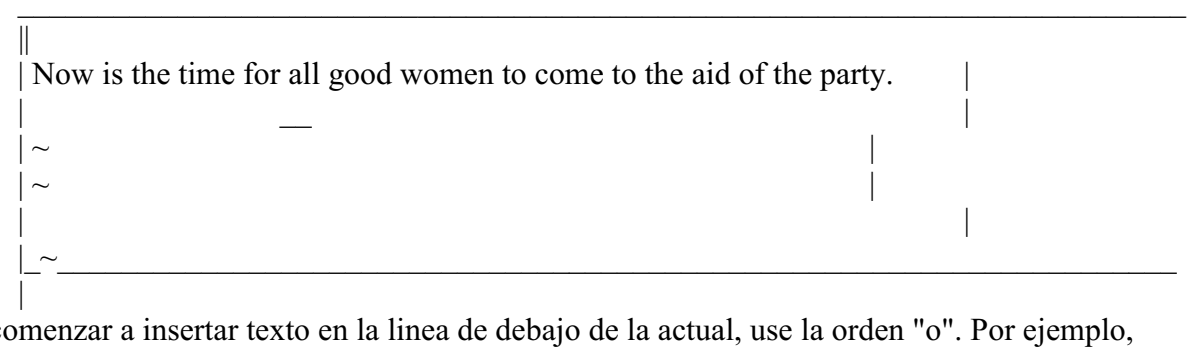
Para salir del modo de inserción y volver al modo de ordenes, pulse |_esc_|.
Mientras este en modo ordenes, puede usar las teclas del cursor para moverse por el fichero. En nuestro ejemplo, como solo tenemos una linea, el tratar de usar las teclas de linea arriba o abajo, probablemente hará que vi emita un pitido.

Hay muchas formas de insertar texto a parte de la orden i. Por ejemplo, la orden a inserta texto comenzando detrás de la posición actual del cursor, en lugar de la posición actual del cursor. Por ejemplo, use la tecla de cursor a la izquierda para desplazar el cursor entre las palabras "good" y

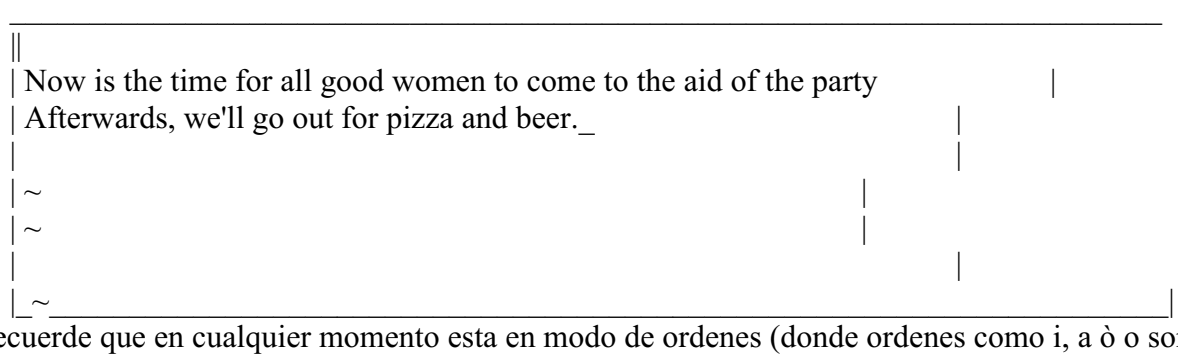
"men".




Pulse |_a_| para iniciar el modo inserción, teclee "wo" y pulse |_esc_| para volver al modo de ordenes.



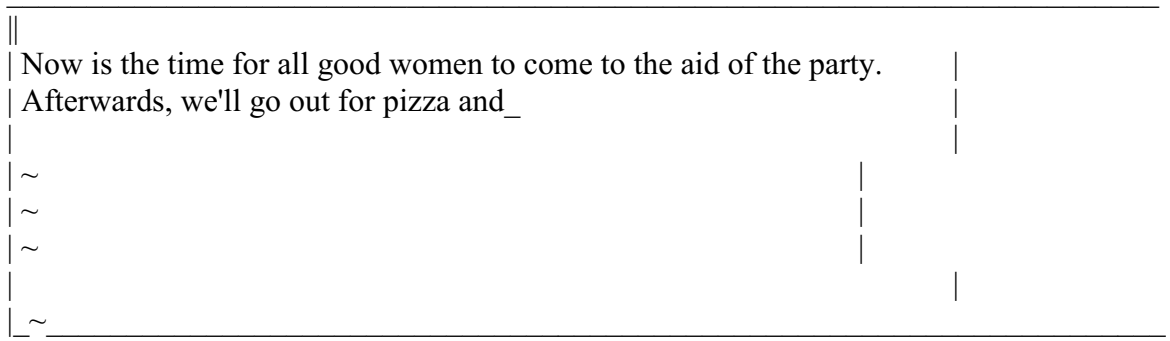

Para comenzar a insertar texto en la linea de debajo de la actual, use la orden "o". Por ejemplo, pulse |_o_| y teclee otra linea o dos:



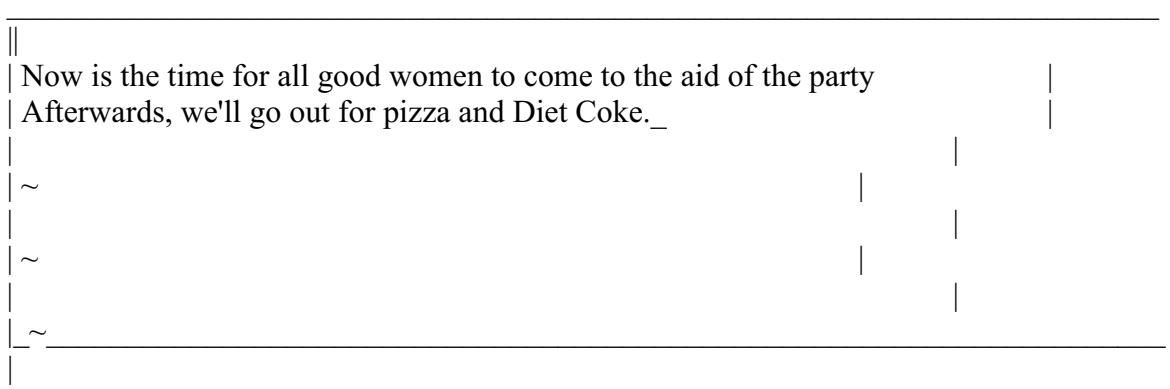

Solo recuerde que en cualquier momento esta en modo de ordenes (donde ordenes como i, a ò o son validas, o en modo de inserción (cuando este insertando texto, pulse |_esc_|para volver al modo de ordenes), o en modo de ultima linea (donde puede introducir comandos extendidos, como veremos mas adelante).

**12.4 Borrando texto**

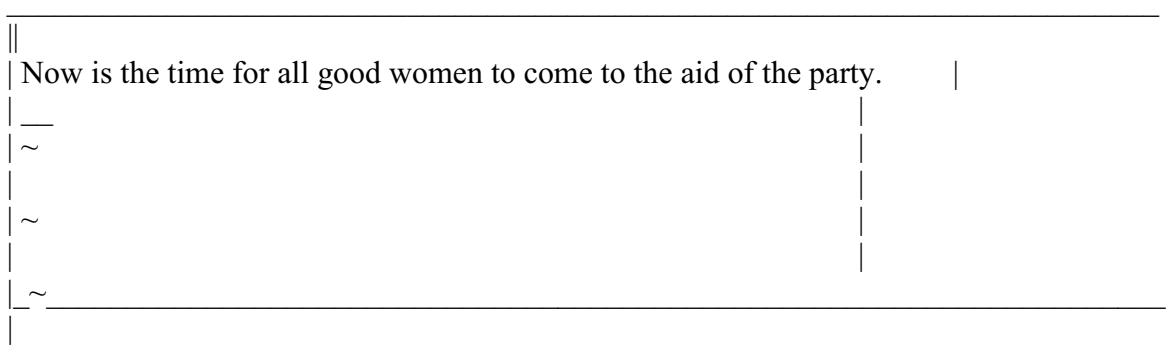
Desde el modo de ordenes, la orden x borra el carácter debajo del cursor.  Si pulsa |_x_| cinco veces, terminara con:



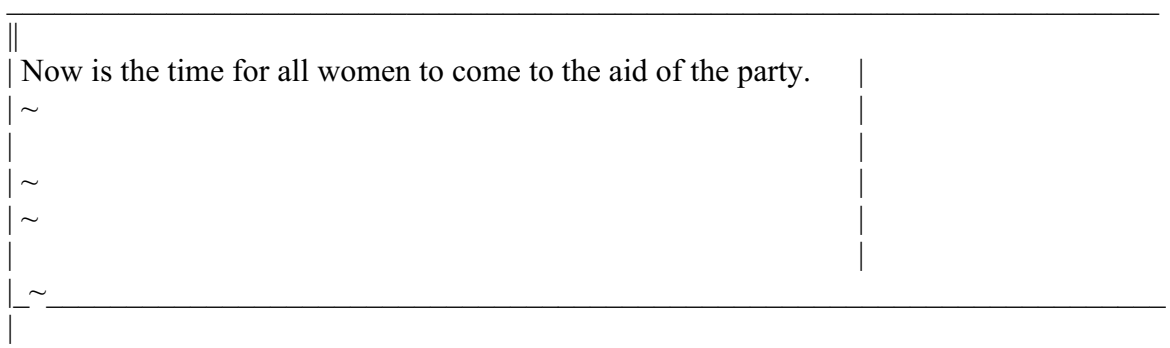
Ahora pulse |_a_|, inserte algún texto, seguido de |_esc_|:




Puede borrar líneas enteras usando la orden dd (es decir, pulse |_d_| dos veces en una fila). Si el cursor esta en la segunda linea y teclea dd,

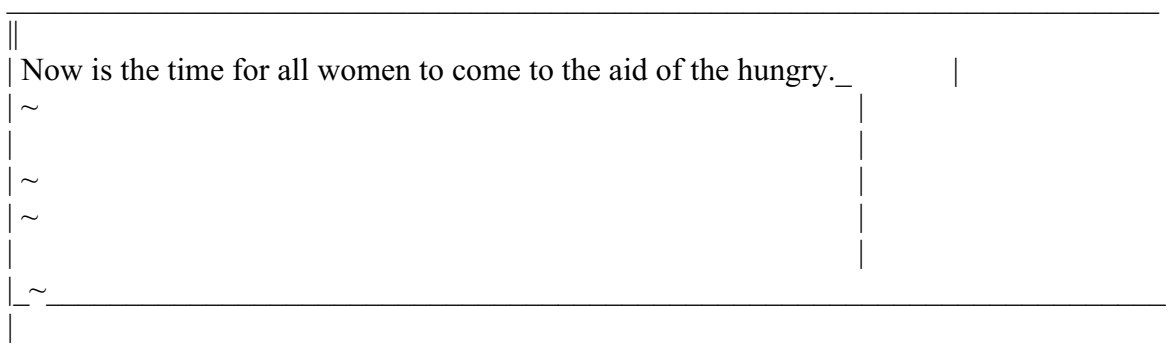



Para borrar la palabra sobre la que se encuentra el cursor, use la orden dw. Situé el cursor sobre la palabra "good" y pulse dw.




**12.5 Modificando texto**

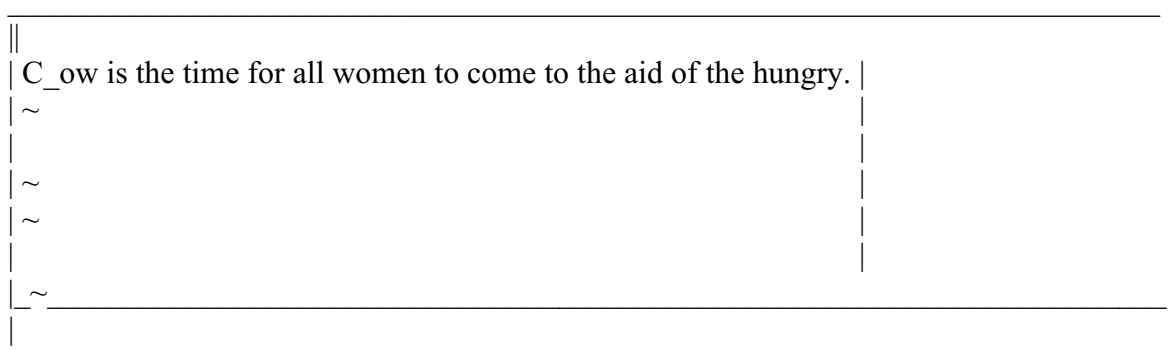
Puede sustituir secciones de texto usando la orden R. Situé el cursor en la primera letra de "party" y pulse |_R_|, y escriba la palabra "hungry".



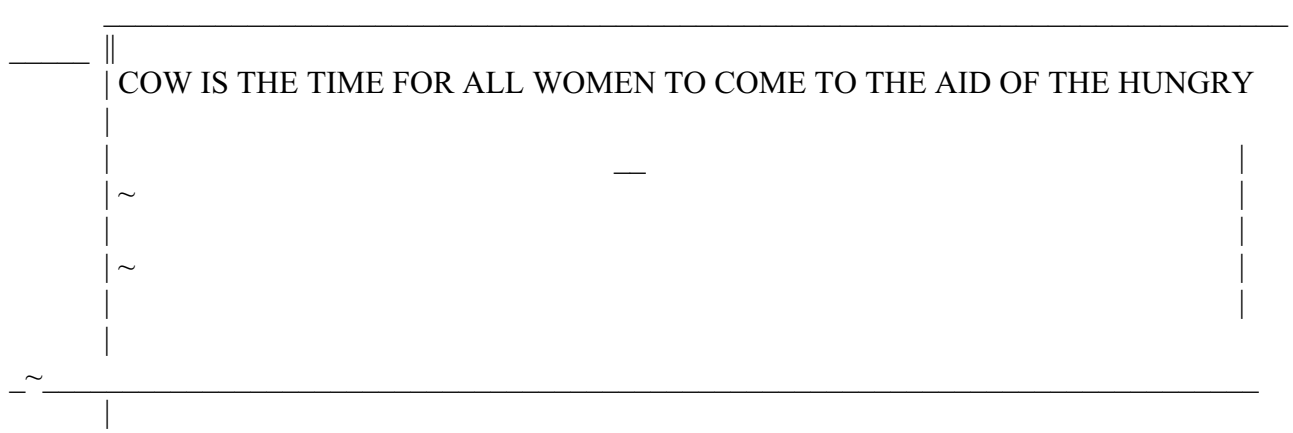

El uso de R para editar texto es bastante parecido al uso de las ordenes i y a, pero R sobreescribe

texto en lugar de insertarlo.

La orden r sustituye un único carácter situado debajo del cursor. Por ejemplo, sitúe el cursor al comienzo de la palabra “now" y escriba r seguido de C. Obtendrá:




La orden "~" cambia de mayúsculas a minúsculas o viceversa la letra sobre la que se encuentra el cursor. Por ejemplo, si sitúa el cursor sobre la "o" de "Cow", y repetidamente pulsa |_~_|, obtendrá:




**12.6 Ordenes de movimiento**

Ya conoce como usar las teclas del cursor para moverse por el documento. Además, puede usar las ordenes h, j, k y l para mover el cursor a la izquierda, abajo, arriba y derecha respectivamente. Esto es muy cómodo cuando (por alguna razón) sus teclas de cursor no funcionen correctamente.

La orden w mueve el cursor al comienzo de la siguiente palabra;  b lo lleva al comienzo de la palabra anterior.
La orden 0 (cero) mueve el cursor al comienzo de la linea actual, y la orden $ lo lleva al final de la linea.
Al editar ficheros grandes, querrá moverse hacia adelante y atrás a lo largo del fichero mostrando una pantalla cada vez. Pulsando |_ctrl-F_| avanza el cursor una pantalla hacia adelante y |_ctrl-B_| lo lleva una pantalla atrás.

Para llevar el cursor al final del fichero, pulse G. Puede también desplazarse a una linea arbitraria; por ejemplo, pulsando la orden 10G llevara el cursor a la linea 10 del fichero. Para desplazarse al comienzo, use 1G. Puede asociar ordenes de desplazamiento con otras ordenes como es el borrado. Por ejemplo,

La orden d$ borrara todo desde la posición del cursor al final de la linea; dG borrara todo desde la posición del cursor al final del fichero.

**12.7 Guardando ficheros y saliendo de vi**

Para salir de vi sin modificar el fichero use la orden :q!. Al pulsar ":", el cursor se desplazara a la ultima linea de la pantalla; esta en modo ultima linea.

```
                                                                                
____||
      | COW IS THE TIME FOR ALL WOMEN TO COME TO THE AID OF THE HUNGRY
      |
      | ~                                                                     |
      | ~                                                                     |
      |                                                                       |
      | ~                                                                     |
      | :                                                                     |
      |
_____________________________________________________________________________|
```

En el modo de ultima linea hay disponibles una serie de ordenes extendidas. Una de ellas es q!, la cual permite salir de vi sin guardar los cambios. La orden :wq salva el fichero y sale de vi. La orden ZZ (desde el modo de ordenes, sin ":") es equivalente a :wq. Recuerde que debe pulsar |_enter_| después de introducir la orden para que esta se ejecute en el modo ultima linea.

Para salvar el fichero sin salir de vi, simplemente use :w.

**12.8 Editando otro fichero**

Para editar otro fichero use la orden :e. Por ejemplo, para dejar de editar el fichero test y en su lugar editar el fichero foo, use la orden

```
                                                                                
_______||
      | COW IS THE TIME FOR ALL WOMEN TO COME TO THE AID OF THE HUNGRY
      |
      | ~                                                                     |
      |                                                                       |
      | ~                                                   |
      | :e foo
      |
      |
_____________________________________________________________________________|
```

Si usa :e sin salvar primero el fichero, obtendrá el mensaje de error

```
                                                                                
______ ||
      _No_write_since_last_change_(":edit!"___overrides)
      |
_____________________________________________________________________________
      |
```

lo cual significa que vi no quiere editar otro fichero hasta que salve el primero. En este punto, puede usar :w para guardar el fichero original, y entonces usar :e, o puede usar la orden

```
                                                                                
______ ||
      | COW IS THE TIME FOR ALL WOMEN TO COME TO THE AID OF THE HUNGRY
      |
      | ~                                                                     |
      |                                                                       |
      | ~                                                                     |
      |                                                                       |
      | ~                                                                     |
      | :e!   foo
      |
      |
```

El signo "!" le dice a vi lo que realmente desea usted editar el nuevo fichero sin salvar los cambios del primero.

**12.9 Incluyendo otros ficheros**

Si usas la orden :r Puede incluir el contenido de otro fichero en el fichero que esta editando. Por ejemplo

```
r foo.txt
```

insertaría el contenido del fichero foo.txt en el texto en la posición actual de cursor.

**12.10 Ejecutando comandos del interprete**

Puede también ejecutar comandos del interprete desde el interior de vi. La orden :r! funciona como : r, pero en lugar de leer un fichero, inserta la salida de un comando dado en el fichero en la posición actual del cursor. Por ejemplo, si usa la orden

```
:r!   ls -F
```

obtendrá

```
COW IS THE TIME FOR ALL WOMEN TO COME TO THE AID OF THE HUNGRY
letters/
misc/
papers/
```

También puede salir a un interprete de comandos desde vi, es decir, ejecutar una orden desde dentro de vi y volver al editor una vez esta finalice. Por ejemplo, si usa la orden

```
:!   ls –F
```

la orden ls -F será ejecutada, y los resultados mostrados en la pantalla, pero no insertados en el fichero en edición. Si usa la orden

```
:shell
```

vi iniciara una instancia del interprete de comandos, permitiéndole temporalmente dejar a vi "parado" mientras ejecuta otras ordenes. Simplemente salga del interprete de comandos (usando la orden exit) para regresar a vi

**12.11 Obteniendo ayuda**

vi no proporciona demasiada ayuda de forma interactiva (la mayoría de los programas UNIX no lo hacen), pero siempre puede leer la pagina de manual para vi. vi es un "front-end" visual para el editor ex: es decir, es ex quien maneja la mayoría de las ordenes en el modo ultima linea. Luego además de leer la pagina de vi, consulte la de ex también.

## 13 Personalizando su entorno

El interprete de comandos proporciona muchos mecanismos para personalizar su entorno de trabajo.

Como hemos mencionado antes, el interprete de comandos es mas que un mero interprete es también un poderoso lenguaje de programación. Aunque escribir guiones del interprete de comandos es una tarea extensa, nos gustaría introducirle algunas formas en las que puede simplificar su trabajo en un sistema UNIX mediante el uso de características avanzadas del interprete.

Como mencionamos antes, diferentes interpretes usan diferentes sintaxis para la ejecución de guiones. Por ejemplo, Tcsh usa una notación al estilo C, mientras que Bourne usa otro tipo de sintaxis. En esta sección no nos fijaremos en las diferencias entre los dos y supondremos que los guiones son escritos con la sintaxis del interprete de comandos Bourne.

**13.1 Guiones del interprete de comandos**

Supongamos que usa una serie de comandos a menudo, y le gustaría acortar el tiempo requerido para teclear agrupándolos en una única "orden". Por ejemplo, las ordenes

```
/home/madhawk# cat chapter1 chapter2 chapter3 > book
/home/madhawk# wc -l book
/home/madhawk# lp book
```

concatenaran los ficheros chapter1, chapter2 y chapter3 y guardara el resultado en el fichero book. Entonces, se mostrara el recuento del numero de líneas del fichero book y finalmente se imprimira con el comando lp.

En lugar de teclear todos esos comandos, podría agruparlos en un guión del interprete de comandos. Describimos los guiones brevemente en la Sección 13.1. El guión usado para ejecutar todas las ordenes seria

```
#!/bin/sh
# A shell script to create and print the book
cat chapter1 chapter2 chapter3 > book
wc -l book
lp book
```

Si el guión se salva en el fichero makebook, podría simplemente usar la orden

```
/home/madhawk# makebook
```

para ejecutar todas las ordenes del guión. Los guiones son simples ficheros de texto; puede crearlos con un editor como Emacs o vi.

Veamos este guión. La primera linea "#!/bin/sh", identifica el fichero como un guión y le dice al interprete de comandos como ejecutarlo. Instruye al interprete a pasarle el guión a /bin/sh para la ejecución, donde /bin/sh es el programa del interprete. ¿Por que es esto importante? En la mayoría de los sistemas UNIX /bin/sh es un interprete de comandos Bourne, como Bash. Forzando al guión a ejecutarse usando /bin/sh nos estamos asegurando de que será interpretado según la sintaxis de Bourne. Esto hará que el guión se ejecute usando la sintaxis Bourne aunque este usando Tcsh como interprete de comandos.

La segunda linea es un comentario. Estos comienzan con el carácter "#" y continúan hasta el final de la linea. Los comentarios son ignorados por el interprete de comandos son habitualmente usados para identificar el guión con el programador.

El resto de las líneas del guión son simplemente ordenes como las que podría teclear directamente.

En efecto, el interprete de comandos lee cada linea del guión y ejecuta la linea como si hubiese sido tecleada en la linea de comandos.

Los permisos son importantes para los guiones. Si crea un guión, debe asegurarse de que tiene permisos de ejecución para poder ejecutarlo. La orden

```
/home/madhawk# chmod u+x makebook
```

puede ser usada para dar permisos de ejecución al guión makebook.

**13.2 Variables del interprete de comandos y el entorno**
El interprete de comandos le permite definir variables como la mayoría de los lenguajes de programación. Una variable es simplemente un trozo de datos al que se le da un nombre.

Nótese que Tcsh, así como otros interpretes del estilo C, usan un mecanismo diferente para inicializar variables del descrito aqui. Esta discusión supondrá el uso del interprete Bourne, como es Bash (el cual probablemente esta usando). Vea la pagina de manual de Tcsh para mas detalles.

Cuando asigna un valor a una variable (usando el operador "="), puede acceder a la variable añadiendo a su nombre "$", como se ve a continuación.

```
/home/madhawk# foo="hello there"
```

A la variable foo se le da el valor "hello there". Podemos ahora hacer referencia a ese valor a través del nombre de la variable con el prefijo "$". La orden

```
/home/madhawk# echo $foo
hello there
/home/madhawk#
```

produce el mismo resultado que

```
/home/madhawk# echo "hello there"
hello there
/home/madhawk#
```

Estas variables son internas al interprete. Esto significa que solo este podrá acceder a las variables. Esto puede ser útil en los guiones; si necesita mantener, por ejemplo, el nombre de un fichero, puede almacenarlo en una variable. Usando la orden set mostrara una lista de todas las variables definidas en el interprete de comandos.

De cualquier modo, el interprete de comandos permite exportar variables al entorno. El entorno es el conjunto de variables a las cuales tienen acceso todas las ordenes que ejecute. Una vez que se define una variable en el interprete, exportarla hace que se convierta también en parte del entorno. La orden export es usada para exportar variables al entorno.
De nuevo, hemos de diferenciar entre Bash y Tcsh. Si esta usando Tcsh, deberá usar una sintaxis diferente para las variables de entorno (se usa la orden setenv). Diríjase a la pagina de manual de Tcsh para mas información.

El entorno es muy importante en un sistema UNIX. Le permite configurar ciertas ordenes simplemente inicializando variables con las ordenes ya conocidas.

Veamos un ejemplo rápido. La variable de entorno PAGER es usada por la orden man. Especifica la orden que se usara para mostrar las paginas del manual una a una. Si inicializa PAGER con el nombre del programa, se usara este para mostrar las paginas de manual en lugar de more (el cual es usado por defecto).

Inicialice PAGER a "cat". Esto hará que la salida de man sea mostrada de una, sin pausas entre paginas.

```
/home/madhawk# PAGER="cat"
```

Ahora exportamos PAGER al entorno.

```
/home/madhawk# export PAGER
```

Pruebe la orden man ls. La pagina debería volar por su pantalla sin detenerse entre paginas.
Ahora, si inicializa PAGER a "more", se usara la orden more para mostrar las paginas del manual.

```
/home/madhawk# PAGER="more"
```

Nótese que no hemos de usar la orden export después del cambio de la variable PAGER. Solo hemos de exportar las variables una vez; cualquier cambio efectuado con posterioridad será automáticamente propagado al entorno.

Las paginas de manual para una orden en particular, le informaran acerca del uso de alguna variable de entorno por parte de esa orden; por ejemplo, la pagina de manual de man explica que PAGER es usado para especificar la orden de paginado.

Algunas ordenes comparten variables de entorno; por ejemplo, muchas ordenes usan la variable EDITOR para especificar el editor por defecto para usar si es necesario.
El entorno es también usado para guardar información importante acerca de la sesión en curso.
Un ejemplo es la variable de entorno HOME, que contiene el nombre del directorio de origen del usuario.

```
/home/madhawk/papers# echo $HOME
/home/madhawk
```

Otra variable de entorno interesante es PS1, la cual define el "prompt" principal que usara el interprete. Por ejemplo,

```
/home/madhawk# PS1="Your command, please: "
Your command, please:
```

Para volver a inicializar el "prompt" a su valor habitual (el cual contiene el directorio actual seguido por el símbolo "#"),

```
Your command, please:   PS1="\w# "
/home/madhawk#
```

La pagina de manual de bash describe la sintaxis usada para inicializar el "prompt".

#### 13.2.1 La variable de entorno PATH

Cuando usa la orden ls ¿como encuentra el interprete el programa ejecutable ls?. De hecho, ls se encuentra en /bin/ls en la mayoría de los sistemas. El interprete usa la variable de entorno PATH para localizar los ficheros ejecutables u ordenes que tecleamos.
Por ejemplo, su variable PATH puede inicializarse a:

```
/bin:/usr/bin:/usr/local/bin:.
```

Esto es una lista de directorios en los que el interprete debe buscar. Cada directorio esta separado por un ":". Cuando usa la orden ls, el interprete primero busca /bin/ls, luego /usr/bin/ls y así hasta que lo localice o acabe la lista.

Nótese que PATH no interviene en la localización de ficheros regulares. Por ejemplo, si usa la orden

```
/home/madhawk# cp foo bar
```

El interprete no usara PATH para localizar los ficheros foo y bar esos nombres se suponen completos. Solo se usara PATH para localizar el programa ejecutable cp.

Esto le permitirá ahorrar mucho tiempo; significa que no deberá recordar donde son guardadas las ordenes. En muchos sistemas los ficheros ejecutables se dispersan por muchos sitios, como /usr/bin, /bin o /usr/local/bin. En lugar de dar el nombre completo con el camino (como /usr/bin/cp), solo hemos de inicializar PATH con la lista de los directorios donde queremos que se busquen automáticamente.

Nótese que PATH contiene ".", el cual es el directorio actual de trabajo. Esto le permite crear guiones o programas y ejecutarlos desde su directorio de trabajo actual sin tener que especificarlo directamente (como en ./makebool). Si un directorio no esta en su PATH, entonces el interprete no buscara en el ordenes para ejecutar esto incluye al directorio de trabajo.

### 13.3 Guiones de inicialización del interprete

A parte de los guiones que puede crear, hay un numero de estos que usa el interprete de comandos para ciertos propósitos. Los mas importantes son sus guiones de inicialización, guiones automáticamente ejecutados por el interprete al abrir una sesión.

Los guiones de inicialización son eso, simples guiones como los descritos arriba. De cualquier modo, son muy útiles para la inicialización de su entorno al ejecutarse automáticamente. Por ejemplo, si siempre usa la orden Mail para comprobar si tiene correo al iniciar una sesión, incluya en su guión de inicialización dicha orden y será ejecutada automáticamente.
Tanto Bash como Tcsh distinguen entre un interprete de presentación y otras invocaciones del

interprete. Un interprete de presentación es el que se ejecuta en el momento de la presentación al sistema (login). Es el único que usara. De cualquier modo, si ejecuta una opción de salir a un interprete desde algún programa, como vi, inicializa otra instancia del interprete de comandos, el cual no es su interprete de presentación. Además, en cualquier momento que ejecuta un guión, automáticamente esta arrancando otro interprete que va a ser el encargado de ejecutar el guión.
Los ficheros de inicialización usados por Bash son: /etc/profile (configurado por el administrador del sistema, y ejecutado por todos los usuarios de Bash en el momento de la presentación al sistema), $HOME/.bash_profile (ejecutado por una sesión de presentación Bash) y $HOME/.bashrc (ejecutadas por todas las sesiones Bash que no son de presentación). Si .bash_profile no esta presente, se usa en su lugar .profile Tcsh usa los siguientes guiones de inicialización: /etc/csh.login (ejecutado por todos los usuarios de Tcsh en el momento de la presentación al sistema), $HOME/.tcshrc (ejecutado en la presentación al sistema por todas las instancias nuevas de Tcsh) y $HOME/.login (ejecutado en la presentación al sistema, seguido .tcshrc). Si .tcshrc no esta presente, .cshrc se usa en su lugar.
Para entender completamente la función de estos ficheros, necesitara aprender mas acerca del interprete de comandos. La programación de guiones es una materia complicada, mas allá del alcance de este libro. Lea las paginas de manual de bash y/o tcsh para aprender mas sobre la configuración de su entorno.

### 3.14 ¿Quieres seguir por tu cuenta?

Esperamos haberle proporcionado suficiente información para darle una idea básica de como usar el sistema, teniendo en cuenta que la mayoría de los aspectos mas importantes e interesantes de Linux no están cubiertos aqui esto es muy básico. Con esta base, en poco tiempo estará ejecutando complicadas aplicaciones y aprovechando todo el potencial de tu sistema. Si la cosa no es muy excitante al comienzo, no desespere hay mucho que aprender.

Una herramienta indispensable para aprender acerca del sistema son las paginas del manual.

Aunque muchas de las paginas pueden parecer confusas al principio, si se profundiza hay gran cantidad de información en ellas.

También es interesante leer un libro sobre la utilización de un sistema UNIX. Hay mucho mas en UNIX de lo que pueda parecer a simple vista- desafortunadamente, la mayoría de ello queda fuera del alcance de este tutorial.