Guía básica de uso para el sistema operativo GNU/Linux

Noviembre 2015

Por Paulo Colomés F.

www.netlearning.cl

¿Qué es Linux?

Linux es un sistema operativo multitarea descendiente de la gama de sistemas UNIX. Su nombre es una combinación del nombre del autor, un finlandés llamado Linus Torvalds, y la X (casi obligatoria) de UNIX. Linux ha sido históricamente utilizado en ambientes de servidor, donde los entornos gráficos y escritorios no son en absoluto necesarios, sin embargo en distribuciones actualizadas es posible lograr un desempeño de entorno gráfico tanto o más superior que sistemas operativos como Microsoft Windows o Mac OSX, los cuales basan todo su potencial en el entorno gráfico y es precisamente esta una de las grandes diferencias entre Linux y los demás sistemas. Linux se puede administrar totalmente utilizando solamente un entorno de texto (shell).

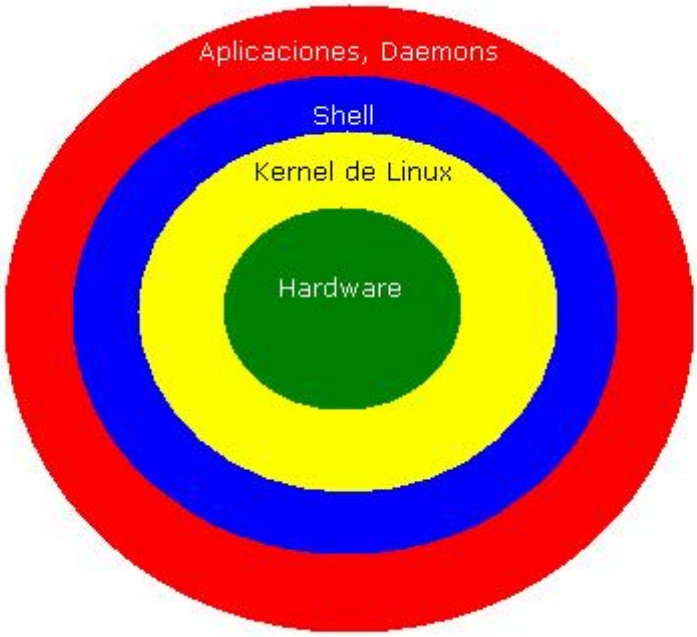
En el año 91 se creó la versión 1.0 de Linux, es decir, el Kernel 1.0. Este Kernel (núcleo) rápidamente comenzó a distribuirse por Internet (muy poco común en aquella época) y en círculos relacionados con el ámbito universitario donde fue utilizado por todos aquellos que requerían de un sistema abierto (código libre).

Ya a mediados de los 80 Richard Stallman había comenzado un proyecto enfocado en crear un sistema operativo totalmente nuevo, libre, de código abierto y estable. Esto en directa oposición al UNIX de entonces que era propietario y por lo mismo no se podía interactuar directamente con el código. Además era bastante caro (unos U$7000) y por lo mismo era casi imposible trabajar con un sistema operativo fuera de las universidades y grandes organizaciones. Para solucionar eso, la Free Software Foundation (dirigida por Stallman) inició el proyecto GNU (GNU's not Unix). En poco tiempo ya habían desarrollado múltiples herramientas concebidas como "libres" (código abierto y distribución gratuita) para distintos propósitos, pero sin embargo les faltaba un corazón, un núcleo que sea el principal motor del sistema. El núcleo desarrollado por Linus Torvalds encajaba a la perfección con el resto de herramientas y pronto se dio a conocer a Linux con el nombre de GNU/Linux, con el fin de demostrar y mantener vigente la unión de ambos sistemas.

No fue si no hasta mediados de los 90 que Linux vio su gran auge. Hasta ese momento no habían grandes razones para preferirlo frente a otras opciones de software propietario (DOS, UNIX, etc). El crecimiento explosivo de Internet dio lugar a que se desarrollaran nuevas soluciones para administrar servidores Web y fue entonces cuando se inventó Apache. Apache es una herramienta de administración para plataformas Web (servidor Web) diseñada para funcionar nativamente bajo Linux. La solución de Apache constituía una gran herramienta que el resto de los servidores Web no tenía, y era la posibilidad de almacenar en un solo servidor muchos sitios Web con nombres distintos. Esta modalidad, llamada VirtualHosts, hizo que rápidamente las empresas comenzaran a utilizar Linux con Apache para gestionar sitios. Cuando se tenían 800 clientes, cada uno requiriendo una página personal en Internet, el costo de mantención se hacía elevadísimo. Apache cambió eso e hizo de Linux uno de los sistemas operativos más utilizados.

Desde entonces ha pasado mucho tiempo y en la actualidad Linux ya va en su versión 2.6.22. La estabilidad y facilidad de manejo han hecho de este sistema operativo el preferido por aquellos administradores entusiastas quienes les gusta optimizar al máximo sus sistemas y sacarles el mejor rendimiento posible.

Podemos decir entonces que Linux es netamente un Kernel sobre el cual se cargan aplicaciones personalizadas. Es precisamente esta combinación de aplicaciones que se montan sobre el Kernel lo que hace la diferencia entre las distintas **distribuciones** de Linux.



Entorno de sistema bajo Linux

Existen muchísimas distribuciones, pero ha habido algunas que se han destacado sobre otras por distintas razones, que incluyen facilidad de instalación y administración, excelente soporte, orientación a algunas aplicaciones específicas (Ej: The Linux Router Project), etc. Entre las distribuciones más conocidas se puede destacar:

- **RedHat y derivados: Fedora, CentOS, Mandrake, WhiteBox, etc.**
- **Debian y derivados: Ubuntu, Knoppix**
- **Slackware**
- **SuSE**
- **Caldera**
- **Etc**

La diferencia entre estas distintas distribuciones radica básicamente en las interfaces de usuario (UI), las shells, el software que viene incorporado y los entornos gráficos. Más allá de estas diferencias, todos los Linux usan el mismo corazón (Kernel) en distintas versiones.

Introducción a Linux

Existen personas que llevan muchos años trabajando en sistemas y nunca han dejado el tradicional Windows. Quizá simplemente por comodidad o costumbre, pero en la mayoría de los casos es porque existe una suerte de miedo infundado respecto a la gran dificultad que presenta Linux para utilizarse y que quienes manejan este sistema son prácticamente seres superdotados. Nada más lejos de la realidad. Linux termina siendo muy fácil y cómodo de manejar cuando uno conoce la parte básica tan solo.

La respuesta para tener éxito administrando sistemas Linux es una sola, la constancia y el gusto por aprender. Hay quienes usan Linux dos veces y cuando se encuentran con los primeros problemas vuelven a sus máquinas Windows. Y también hay quienes ante un problema siguen adelante hasta que logran solucionarlo y terminan aprendiendo mucho más rápido que si solamente se dedicaran a leer documentación y poco y nada ponen las manos en Linux.

Antes de lanzarse a ejecutar comandos hay que conocer un par de cosas. Existen dos modos de trabajo: **Modo Texto (o Consola)** y **Modo Gráfico.** Ya se había dicho que Linux es completamente administrable bajo el modo texto. Está optimizado para no requerir de un ambiente de escritorio para realizar las funciones de servidor. Es necesario saber que:

- Los entornos de escritorio más conocido son GNOME y KDesktop (KDE)
- Los entornos de texto se conocen con el nombre de **Shell**
- Existen varias Shells, pero las más conocidas son: Bash (Bourne Again Shell), ksh (Korn Shell), csh (Shell C), tcsh, sh y ssh (Secure Shell).
- En Linux (y UNIX) existen 3 tipos de usuarios, cada uno con distintos privilegios. Primero están los usuarios de sistema que son aquellos usuarios que son creados por distintas aplicaciones y solamente sirven para que esas aplicaciones funcionen correctamente. Ejemplo: Cuando se instala MySQL en el sistema, se crea un usuario llamado **mysql**. Los usuarios de sistema no pueden loguearse en la máquina, a menos que expresamente se les indique.

  En segundo lugar tenemos a los usuarios "reales". Estos son aquellos que son creados por el administrador para poder trabajar en el sistema con privilegios restringidos. Sí pueden loguearse.

  El más importante de todos los usuarios es el súper administrador denominado **root.** Es root quien tiene todos los permisos para ejecutar cualquier cosa y por lo mismo hay que tener especial cuidado con él.

  Existe incluso un nivel superior al usuario root que es el mismo Kernel. Hay cosas que root no puede hacer y el Kernel si, pero eso es otro asunto.

- Los sistemas basados en UNIX (como Linux) son sensibles a mayúsculas y minúsculas (case sensitive). Por norma se escribe siempre todo con minúscula. Ejemplo, el comando "cat" no funcionará si se escribe "Cat".

Conocidas estas cosas, ya podemos trabajar directamente con Linux.

Estructura de directorios

El directorio principal (padre) se denomina **raíz** y se identifica con el símbolo "/" (slash). De / se desprende el siguiente esquema con alguno de los directorios más importantes:

Directorio	Significado	Uso	Ejemplo
/	Raíz	Directorio principal	-
/bin/	Binaries (Binarios)	Almacena los binarios principales, los cuales pueden ser ejecutados por cualquier usuario	/bin/sh, /bin/cat, /bin/ls
/boot/	Boot (Arranque)	Contiene la información de arranque.	Kernel, initird, boot loaders, etc.
/dev/	Devices (dispositivos)	Almacena los dispositivos esenciales	/dev/null, /dev/hda1, /dev/cdrom0, /dev/usb0
/etc/	Et cetera (Y el resto)	Aquí se guardan los archivos de configuración, tanto del sistema mismo como de aplicaciones externas	/etc/passwd, /etc/mysql/my.cnf, /etc/apache2/apache2.conf /etc/X11
/home/	Hogar	Directorio personal de cada usuario.	/home/usuario/ /home/alberto/
/lib/	Librerías	Contiene las librerías esenciales para los binarios	/lib/libalias.so.4
/mnt/	Mount (montar)	Directorio para montar sistemas de archivo temporales	/mnt/cdrom/ /mnt/Windows/ /mnt/usb/
/opt/	Optional (Opcion)	Paquetes de aplicaciones opcionales	/opt/Google-Earth/
/proc/	Procesos	Sistema de archivos virtual que almacena información para el Kernel y los procesos como archivos de texto.	/proc/version /proc/cpuinfo
/root/	Raíz	Directorio personal para el usuario **root**. Es el único que tiene directorio personal fuera de /home/	/root/archivos/personales
/tmp/	Temporal	Directorio de archivos temporales	/tmp/prueba.c
/usr/	User	Datos de Usuario	/usr/share/
/var/	Variable	Almacena los datos variables del sistema. Normalmente es una partición extra.	/var/log/ /var/www/ /var/run/ /var/mail/

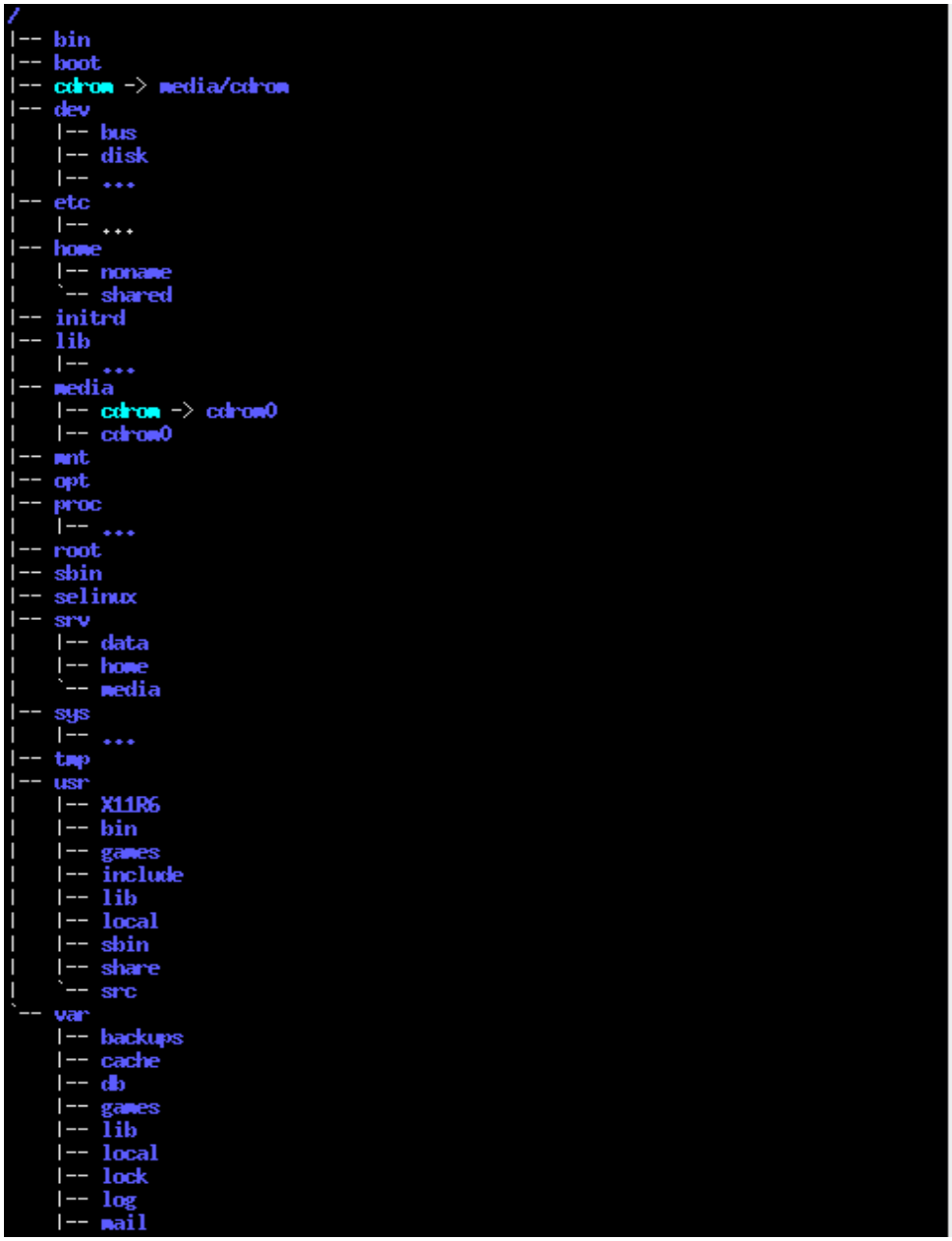


Imagen de la estructura de directorios obtenida desde una Shell Linux

Comandos

La mayoría de los comandos UNIX son aplicables a Linux y se componen de la siguiente forma:

1.- Comando espacio sintaxis espacio objetivo
2.- Comando espacio sintaxis espacio origen espacio objetivo

Ejemplo:
1) ls –lh /var/

2) mount –t ntfs /dev/hda1 /mnt/windows

Para conseguir ayuda en línea sobre un comando cualquiera se puede escribir el nombre del comando solo (sin sintaxis y objetivo), con la opción ––help (o –h) y consultando el manual. Ejemplos:

Comando solo:

```
fzas# ping
usage: ping [-AaDdfnoQqRrv] [-c count] [-i wait] [-l preload] [-M mask | time]
          [-m ttl] [-P policy] [-p pattern] [-S src_addr] [-s packetsize]
          [-t timeout] [-z tos] [-G sweepmaxsize ] [-g sweepminsize ]
          [-h sweepincrsize ] host
       ping [-AaDdfLnoQqRrv] [-c count] [-I iface] [-i wait] [-l preload]
          [-M mask | time] [-m ttl] [-P policy] [-p pattern] [-S src_addr]
          [-s packetsize] [-T ttl] [-t timeout] [-z tos] mcast-group
fzas#
```

Con la opción ––help (o –h):

```
fzas# date --help
date: illegal option -- -
usage: date [-jnu] [-d dst] [-r seconds] [-t west] [-v[+|-]val[ymwdHMS]] ...
          [-f fmt date | [[[[[cc]yy]mm]dd]HH]MM[.ss]] [+format]
fzas#
```

O consultando el manual:

```
fzas# man date

NAME
     date -- display or set date and time

SYNOPSIS
     date [-ju] [-r seconds] [-v [+|-]val[ymwdHMS]] ... [+output_fmt]
     date [-jnu] [[[[[cc]yy]mm]dd]HH]MM[.ss]
     date [-jnu] -f input_fmt new_date [+output_fmt]
     date [-d dst] [-t minutes_west]

DESCRIPTION
     When invoked without arguments, the date utility displays the current
     date and time.  Otherwise, depending on the options specified, date will
     set the date and time or print it in a user-defined way …
```

La siguiente es una lista de los comandos más utilizados junto con una descripción. Los suficientes como para movernos sin problemas e interactuar a buen nivel con un sistema Unix/Linux. (Copy/Paste de http://blackshell.usebox.net/pub/shell/taller_sh/x137.html). Siguiendo las instrucciones inmediatamente anteriores pueden consultar los detalles de cada comando

---

ls
muestra el contenido de un directorio

echo
hace eco en pantalla

Ejemplo:

$ echo hola mundo!

cat
muestra el contenido de un fichero

more
muestra el contenido de un fichero haciendo pausas entre pantallas si el fichero es largo

man
muestra la página del manual de un comando

Ejemplo:

$ man ls

clear
borra la pantalla

cp
copia ficheros y directorios

Ejemplo:

$ cp fichero_original fichero_copia

mv
mueve ficheros

Ejemplo:

$ mv fichero fichero2

rm
borra ficheros

Ejemplo:

$ rm fichero

ln
enlazar (referenciar) ficheros

Ejemplo de enlace "duro" (hardlink):

$ ln fichero enlace

Ejemplo de enlace "suave" (softlink):

$ ln -s fichero enlace_simbólico

cd
cambia de directorio de trabajo si no se indica directorio,
nos traslada a $HOME

Ejemplo:

$ cd directorio

pwd
muestra el directorio de trabajo actual

mkdir
crea directorios

Ejemplo:

$ mkdir directorio

rmdir
borra directorios (vacíos)

Ejemplo:

$ rmdir directorio

env
muestra las variables de entorno del programa

head
muestra las n primeras lineas de un fichero (10 por defecto)

Ejemplo:

$ head fichero

tail
muestra las n últimas lineas de un fichero (10 por defecto)

Ejemplo:

$ tail fichero

grep
busca ocurrencias de una cadena en un fichero

Ejemplo:

$ grep cadena fichero

ps
muestra los procesos en el sistema

kill
Envía una señal a un proceso indicando su PID (Process IDentifier, o número único que identifica a cada proceso)

Ejemplo:

$ kill 1002

export
Exporta una variable al entorno del programa

Ejemplo:

$ export VARIABLE=valor

read
Lee una linea de la entrada estándar y la almacena en una variable

Ejemplo:

$ read linea

$
Delante de una variable permite acceder a su contenido

Ejemplo:

$ echo $SHELL

;
Separa dos comandos en una misma linea

Ejemplo:

$ read linea ; echo se ha leído: $linea

file
indica de qué tipo es un fichero

cal

muestra el calendario del mes actual

wc
cuenta lineas, palabras o bytes en ficheros

Ejemplo:

$ echo hola que tal | wc

date
muestra hora y fecha actuales

Ejemplo:

$ date

Ejemplo de fecha en formato yyyy-mm-dd:

$ date "+%Y-%m-%d"

passwd
cambia la contraseña de un usuario

chmod
cambia los permisos de un fichero

chown
cambia el propietario de un fichero

chgrp
cambia el grupo propietario de un fichero

reset
restaura la terminal de texto

whereis
indica donde se puede encontrar un fuente, binario o manual

Ejemplo:

$ whereis ls

which
indica donde está un comando

Ejemplo:

$ which ls

locate
busca ficheros

find
búsqueda avanzada de ficheros

who
quién tiene sesión abierta en la máquina

tac
concatena ficheros y los muestra a la inversa

touch
actualiza la fecha y hora de un fichero, si no existe lo crea

Ejemplo:

$ touch fichero_inexistente

less
una versión más elaborada de **more** que permite desplazarnos por el texto, hacer búsquedas, etc.

df
muestra el espacio libre y ocupados de los discos

du
calcula el espacio de disco usado

mail
programa simple para enviar y leer correo

tar
empaquetar ficheros

Ejemplo empaquetar:

$ tar cvf fichero.tar directorio

Ejemplo desempaquetar:

$ tar xvf fichero.tar

gzip
comprimir un fichero

gunzip
descomprimir un fichero comprimido con **gzip**

zcat
muestra el contenido de un fichero comprimido con **gzip**

ldd
muestra las librerías que usa un programa

halt
apaga la máquina

reboot

reinicia la máquina

shutdown
apaga o reinicia la máquina

true
cierto, o uno

false
falso, o cero

exit
termina la sesión y muestra el *login* del sistema

logout
termina la sesión y muestra el *login* del sistema

seq
genera una secuencia de números

Ejemplo:

```
$ seq 1 10
```

cut
elimina partes de ficheros

Ejemplo:

```
$ echo hola que tal | cut -d " " -f 2
```

awk
escáner de patrones y lenguaje de programación para procesar textos

Ejemplo:

```
$ echo hola que tal | awk '{ print $1 "!",  $2, $3 "?" }'
```

tr
elimina o traduce caracteres

Ejemplo:

```
$ echo hola que tal | tr a A
```

sed
realiza transformaciones en flujos de bytes

Ejemplo:

```
$ echo hola que tal | sed 's/a/A/g'
```

(substituye las 'a' por 'A' en todo el flujo)

fmt
da formato a cada párrafo de un fichero

sort
ordena ficheros de texto

sleep
detiene el proceso durante n segundos

Ejemplo:

$ sleep 5 ; echo Han pasado 5 segundos

uniq
lee de **stdin** y compara lineas adyacentes escribiendo las lineas únicas a **stdout**

Comandos comúnmente utilizados y sintaxis ampliada

La gran mayoría de los comandos UNIX cuentan con una sintaxis que permite ampliar el trabajo o acceder a información extra. Es de mucha ayuda consultar el manual de cada comando para obtener detalles completos acerca de la función y opciones extras que se pueden incorporar como sintaxis.

Para acceder a los manuales se utiliza el comando **man** seguido del comando que se quiere obtener ayuda y opcionalmente el número de la versión del manual.

Ejemplo: Para ver el manual del comando **ls**:

$man ls

Y si se quisiera ver la versión 5 del manual de **ls** (si estuviese disponible) se escribe:
$man ls 5

Otra forma muy práctica de obtener ayuda es escribiendo la opción **--help**, la cual muestra un resumen de la ayuda que trae el manual. Se trata de una ayuda en pantalla muy útil y resumida.

Ejemplo:

$cp -–help

o
$cp -h

Comando **ls**:
ls sirve para listar el contenido de un directorio de la misma manera que lo hace "dir" para DOS. Sin embargo, la sintaxis de **ls** permite obtener mayor información en pantalla:

ls –l: Lista y muestra detalles del archivo o directorio, tales como atributos de permisos, tamaño, fecha de la última modificación, propietario y grupo
ls –lh: Lo mismo que lo anterior pero el tamaño es mostrado en lenguaje HUMANO.

ls –lha: La opción "a" (de ALL) lista todos los directorios y archivos incluyendo los ocultos.

Comando **su:**

su (Switch User) sirve para cambiar de usuario. Si se escribe sin un nombre de usuario, entonces se entenderá que se quiere cambiar al usuario **root**.

Ejemplo:
$su rriquelme (Cambia al usuario rriquelme)
$su (cambia al usuario **root**)

Al cambiar de usuarios se van creando dependencias recursivas, las cuales pueden cerrarse con el comando **exit.** Si se cambia usuarios sin utilizar "exit" entonces estos quedarán ejecutándose en memoria.

Recursividad de usuarios:

$su jose
$su
$su esteban
$su jose
$su

Al ejecutar esos comandos en ese orden ocurrirá que el usuario actual cambiará al usuario **jose**, luego cambiará a **root** pero jose quedará igualmente activo. Si en este punto se escribe **exit**, la sesión de root se cerrará y se volverá al usuario **jose**. Si en vez de escribir **exit** se cambia al usuario **esteban,** seguirán abiertas las sesiones root y jose y así sucesivamente.

Comando **cd:**

cd permite Cambiar Directorio del mismo modo que "dir" en DOS.

Ejemplo:
$cd /etc
Esto sirve para ingresar al directorio "etc" que está en la raíz.

$cd /var/www/html
Con esto se puede ir al directorio donde se alojan los sitios Web en caso de que la máquina cuente con un servidor http como Apache.

Es importante notar que se debe escribir la ruta completa del directorio. Esto significa que DEBE comenzar con un slash /. Para el ejemplo anterior, si no se escribiera el slash al comienzo entonces la shell entendería que se está tratando de ir al directorio "var" dentro del cual se está trabajando en ese momento.

Ejemplo:

$pwd (este comando sirve para ver la ruta de trabajo actual)
/home/usuarios
$ cd var/www/html
Error: no existe el directorio solicitado.

En este caso, se estaría tratando de acceder a /home/usuarios/var/www/html y no a /var/www/html.

Si se escribe el comando **cd** sin ningún parámetro, entonces la shell nos devolverá a nuestro directorio personal. Ejemplo:

```
$pwd
/etc/X11
$cd
$pwd
/home/usuarios
```

Comando **history:**

Este comando sirve para ver un historial de todos los comandos ejecutados por el usuario actual. Cada comando se puede repetir con la flecha "arriba" del teclado. Pero si **history** nos muestra lo que se escribió hace 250 comandos atrás, no es necesario presionar 250 veces la flecha arriba del teclado. En este caso se escribe en la shell el número del comando precedido por el signo de exclamación.
Ejemplo:

```
$history
3 cd
4 ls /
...
..
$!4 (con esto se estaría ejecutando nuevamente ls /)
```

Manejo de Pantalla

Es muy común que al trabajar en un sistema Linux donde no existe entorno gráfico se muestre contenido en la pantalla el cual sobrepasa en tamaño el límite de líneas y aparezca la última parte de la salida. Esto se puede manejar de varias formas, pero las más utilizadas son:

a) Utilizar **more**
b) Subir pantallas
c) Utilizar **grep**

Utilizando more:

"more" (más) es una utilidad de manejo de pantalla que sirve para mostrar el contenido de un archivo sin necesidad de entrar a él, de la misma manera que **cat** pero con opciones más avanzadas.

Si se lista el contenido de un directorio que contiene muchos recursos (como /bin), se puede utilizar **more** para listar por pantallas, del mismo modo que "dir /p" de DOS.

Ejemplo:

```
$ ls /bin |more
```

En este caso se ha utilizado el signo PIPER | el cual se ocupa para "anexar" dos comandos. Aquí, la salida de **ls /bin** se anexa a **more** provocando que se vaya listando por pantallas el contenido del directorio /bin.

Al presionar la barra de espacio se avanza de pantalla en pantalla. También se puede avanzar línea a línea con las flechas del teclado. Y si se quiere cerrar la salida se utiliza CTRL+C.

Subir pantallas:

Otra forma de ver el contenido es utilizar la combinación de teclas SHIFT + RePag o AvPag. Eso permite subir o bajar en el contenido de la pantalla hasta una cierta cantidad de líneas.

Utilizando grep:

Grep es una muy útil herramienta de filtrado de texto. Sirve para buscar una palabra dentro de un directorio e indicar en que línea de que archivos existe esa palabra. Si se utiliza con | sirve para mostrar en pantalla solamente las líneas que contienen la palabra buscada.

Ejemplo:

$lspci
Este comando muestra los dispositivos de hardware de la máquina

$lspci |grep VGA
Mostrará solamente la línea que contenga la palabra VGA, es decir, el adaptador de video.

Para buscar una palabra precisa dentro de todos los archivos de un directorio se utiliza la siguiente sintaxis:

$grep -d recurse "Admin" /etc/

Esto haría que grep busque la palabra Admin recursivamente dentro del directorio /etc/. Las búsquedas recursivas significan que se consultará el directorio indicado y todos los subdirectorios y archivos dentro de él.

Sistema de permisos UNIX

En los sistemas UNIX/Linux se puede lograr una gestión muy eficiente basada en restricciones-permisos para cada directorio, archivo o ejecutable de la máquina. Los permisos o atributos se basan en la numeración octal y se organizan en base a tres conceptos:

- Escritura o **w** (Write)
- Lectura o **r** (Read)
- Ejecución o **x** (Execute)

-

Para ver los permisos de cualquier directorio o archivo se utiliza el comando "**ls –lh /directorio/objetivo"** (sin comillas). Si no se especifica un objetivo, "ls –lh" devolverá el detalle de todos los archivos y subdirectorios del directorio de trabajo actual. La opción "l" (ele) deriva de la palabra "large" y quiere decir que muestre la salida en forma extensiva (larga) y la "h" sirve para mostrar los tamaños en formato más "H"umanamente comprensible y no en bytes. Adicionalmente se puede agregar la opción **a** con lo cual el comando ls mostraría en pantalla todos (ALL) los archivos y directorios, incluyendo los ocultos (cuyos nombres comienzan con un punto, ejemplo: .htaccess).

Ejemplo:

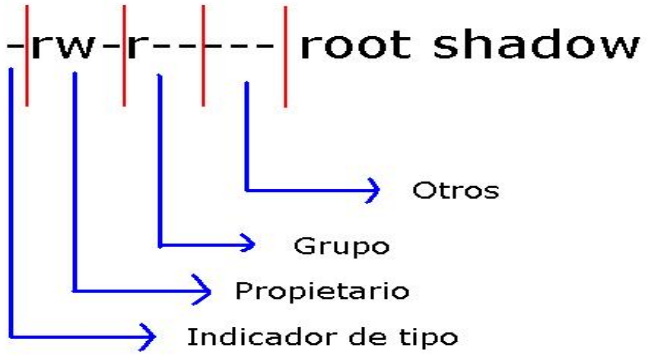
$ls –lh /etc/shadow

-rw-r----- root shadow 4,0 KB Mar 18 de Sept de 2007 shadow

La primera parte (-rw-r-----) muestra los atributos de lectura, escritura y ejecución.
La segunda parte (root) muestra el propietario (owner) del archivo
La tercera parte (shadow) indica el grupo al cual pertenece el archivo
La quinta parte (4,0 KB) indica el tamaño. Muestra en KB porque se agregó la opción "h" de "human" al comando ls –l, de lo contrario mostraría el tamaño en bytes.
La siguiente sección (Mar 18 de Sept de 2007) indica la fecha de la última modificación del archivo.
Y la última (shadow) muestra el nombre del archivo consultado.

Para comprender de mejor manera los atributos se debe considerar solamente la primera, segunda y tercera parte.

La primera parte consta de 10 indicadores los cuales se descomponen de la siguiente forma:



El **indicador de tipo** es una referencia utilizada para indicar que clase de objeto es lo que se está consultando. El símbolo **–** indica que es un archivo, una letra **d** indica que es un directorio, una **l** significa que es un enlace (link), una **s** indica que es un socket.

Los tres indicadores siguientes (2,3 y 4) muestran los permisos para el dueño del recurso, en este caso **root** puede leer (r) y escribir (w), pero no tiene los permisos para ejecutar (x) si se tratara de un archivo de tipo ejecutable.

Los siguientes tres indicadores muestran los atributos para el grupo al cual pertenece el archivo. En este caso, solamente los usuarios que pertenezcan al grupo **shadow** pueden leer su contenido**.** Y finalmente los 3 indicadores restantes muestran los permisos para todos los otros usuarios que no son ni el propietario ni pertenecen al grupo. El ejemplo mostrado indica que ningún otro usuario podría leer, escribir o ejecutar el archivo.

Para modificar los permisos se utilizan los comandos **chmod, chown y chgrp.**

CHMOD: Deriva de los términos CHange MODe (cambiar modo) y se utiliza para cambiar los atributos (indicadores del 2 al 10) para el propietario, grupo y otros usuarios. La sintaxis de **chmod** se basa en el sistema octal, el cual representa los permisos mediante números del 0 al 7 y se atribuyen realizando una simple interpretación binaria de 3 digitos.

La siguiente tabla ayuda bastante a comprender y saber utilizar los permisos adecuados en cualquier situación, por lo que se aconseja que se aprenda a dibujarla.

Nro	Valor Binario			Atributos	
0	0	0	0	-	-
1	0	0	1	-	-
2	0	1	0	-	W
3	0	1	1	-	W
4	1	0	0	R	-
5	1	0	1	R	-
6	1	1	0	R	W
7	1	1	1	R	W

Para utilizar **chmod** se utilizan los números como valor para representar los permisos. Así, si se quisiese entregar permisos de Lectura, Escritura y Ejecución entonces el valor que se utiliza es 7.

La figura antes mostrada tiene los valores 640 (rw-r-----).

Ejemplo:
a) Para permitir que todos los usuarios de la máquina puedan leer /etc/shadow

$chmod 644 /etc/shadow

Y el resultado de ls –lh sería:

$ls –lh /etc/shadow

-rw-r--r-- root shadow 4,0 KB Mar 18 de Sept de 2007 shadow

Hay que recordar que el 6 corresponde a los permisos para el propietario, el primer 4 corresponde a los permisos para el grupo y el segundo 4 son los permisos para otros usuarios.

b) Para permitir que todos los usuarios puedan leer, ejecutar y escribir el archivo:
$chmod 777 /etc/shadow

Y el resultado de ls –lh sería:
$ls –lh /etc/shadow

-rwxrwxrwx root shadow 4,0 KB Mar 18 de Sept de 2007 shadow

IMPORTANTE: Hay que tener dos cosas en consideración con respecto a **chmod** y el ejemplo mostrado. En primer lugar modificar los permisos para /etc/shadow supondría un riesgo tremendo para la seguridad de la máquina. Es por lo mismo que aquí se ha utilizado de ejemplo. No deben modificar los permisos de ese archivo por seguridad. Por otro lado, solamente el usuario propietario y aquellos con los permisos apropiados (ejemplo root) pueden cambiar los atributos de un directorio o archivo específico.

CHOWN. Deriva de los términos CHAnge OWNer (cambiar propietario) y sirve para modificar el dueño y el grupo de un archivo o directorio. La sintaxis es bastante simple:

$chown usuario:grupo objetivo

Ejemplo:
$chown rodrigo:administradores /var/www/sitioweb/config

CHGRP: Muy similar al anterior, pero sirve solamente para cambiar el grupo. Deriva de CHange GRouP.

En definitiva el sistema de atributos UNIX es muy eficaz en cuanto a permisos-restricciones y por lo mismo su correcto uso puede aumentar de manera muy importante la seguridad de la máquina.

Archivos y directorios importantes

Existen algunos archivos que manejan información importante del funcionamiento del sistema y es necesario conocerlos.

/etc/passwd

El archivo /etc/passwd almacena información de los usuarios del sistema. Para verlo se utiliza el comando **cat**, el cual sirve para mostrar en pantalla el contenido de cualquier archivo sin entrar a él. /etc/passwd tiene una estructura muy fácil de interpretar. Cada línea es la información de un (1) usuario válido de la máquina. Para identificar a los usuarios primero es necesario conocer los tipos que existen en Linux.

Los usuarios de Linux (UNIX) se pueden separar en categorías según su nivel de privilegios. De menores a mayores privilegios están:

- Usuarios de sistema
- Usuarios "reales"
- root
- Kernel

Las diferencias entre ellos ya se explicaron anteriormente en esta guía.
Volviendo a /etc/passwd se puede ver un ejemplo de la primera línea:

root:x:0:Super Administrador:/root/:/bin/bash

Los parámetros están separados por dos puntos (:). El primero, **root**, muestra el nombre de usuario. Luego, la **x** significa que la contraseña de este usuario estará almacenada bajo cifrado en otro archivo llamado /etc/shadow. El tercer parámetro indica el UID (User Identificator, Identificador de Usuario), el cual es un número que se utiliza para identificar usuarios dentro del sistema. Root siempre tendrá el UID 0, mientras que otros usuarios reales tendrán por lo general un UID que empiece en 1000. En cuarto lugar se encuentra la descripción para el usuario y es solo un texto informativo. A continuación está el directorio personal de ese usuario donde aparecerá cada vez que inicie una sesión. Y por último se indica la ruta completa a la **shell** que usará el usuario para iniciar sesión.

El archivo /etc/passwd puede ser leído por cualquier usuario de la máquina, a diferencia de /etc/shadow que sólo puede ser leído y escrito por **root** y leído por los usuarios pertenecientes al grupo **shadow**. De esta manera se evita que cualquier usuario sin privilegios del sistema pueda acceder al contenido de /etc/shadow (donde se almacenan las contraseñas cifradas) para intentar romper las contraseñas, por ejemplo utilizando **John The Ripper.**

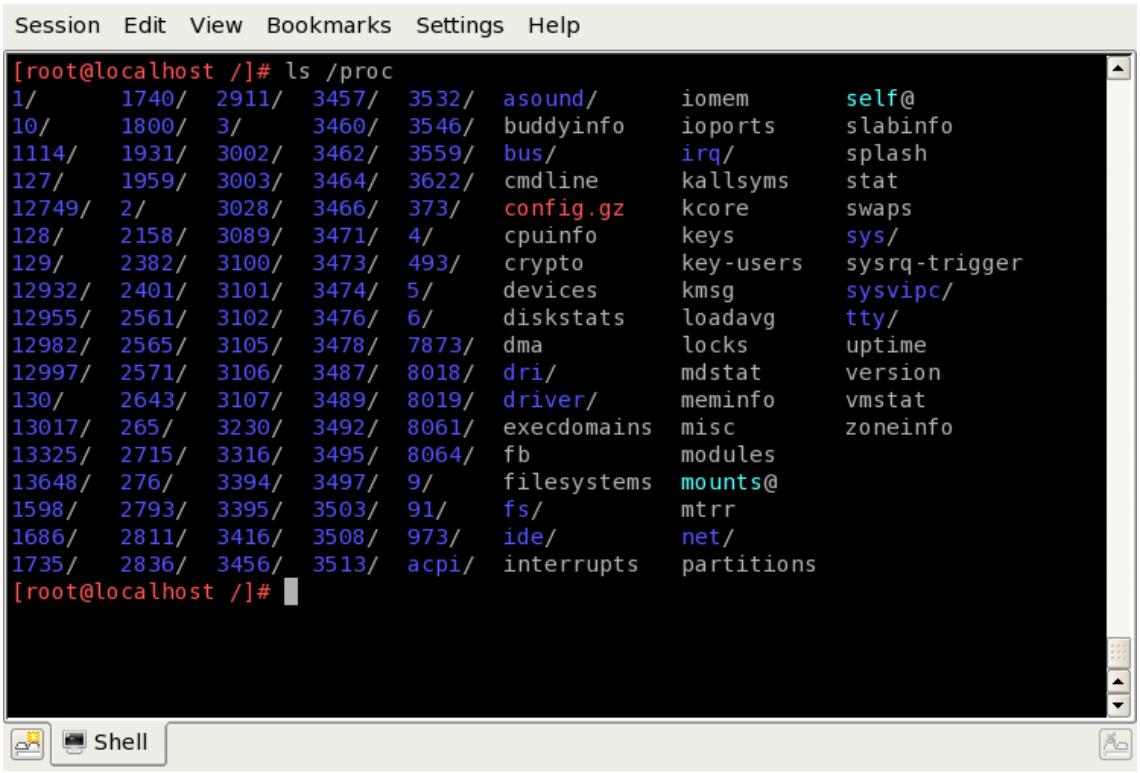
/proc

/proc es un directorio especial que se utiliza para mostrar información del kernel en tiempo real. Se monta como un sistema de archivos independiente de sólo lectura, el cual puede ser modificado solamente por el mismo kernel y ni root puede escribir en él.

Todos los procesos de la máquina crean subdirectorios dentro de /proc (de ahí el nombre) con su identificador de proceso (PID), además de tener información relevante de los dispositivos de sistema y datos del mismo kernel.

Ejemplo:

/proc/version contiene información respecto a la versión del Kernel de Linux que se está utilizando. Contiene la salida del comando **uname –a**.
/proc/cpuinfo contiene información respecto al tipo, modelo, marca, velocidad y demás características del procesador utilizado.



Directorio /proc

/dev

Los dispositivos en Linux se identifican mediante un archivo dentro del directorio /dev. Algunos ejemplos son /dev/cdrom0, /dev/tty1, /dev/audio, etc.

En el caso de los discos duros, la identificación es bastante particular. No se denominan con letras como en Windows, si no que se llaman mediante un archivo dentro de /dev/. El comando para ver los discos y el espacio en uso de ellos se llama **df**.

Por ejemplo, si existe una máquina con dos discos duros tipo IDE y ambos con dos particiones, entonces Linux mostraría algo similar a esto:

/dev/hda1
/dev/hda2
/dev/hdb1
/dev/hdb2

Las letras "hd" vienen de "Hard Disk" (Disco Duro), la tercera letra identifica al disco físico y el número identifica a la partición. Cuando los discos son de tipo SATA y SCSI, Linux carga el módulo SCSI en el Kernel y por lo tanto los discos se mostrarían con el nombre de **/dev/sda1, /dev/sda2, /dev/sdb1,**etc. Las letras "sd" derivan de SCSCI Disk.

Todas las particiones de Linux deben ir "montadas" en un directorio único. Esto se conoce como "punto de montaje" y se utiliza para separar directorios en particiones distintas. Por ejemplo, todo el directorio /var/ se puede manejar como una partición independiente. Lo normal es que cuando se trabaja con el disco utilizando una partición completa se monte en la raíz /.

Vale indicar que en cualquier instalación Linux siempre se pedirán definir como mínimo dos particiones. Una de ellas es denominada partición **swap** o "partición de intercambio" y se utiliza como memoria virtual. Esta partición especial debe ser del doble de la memoria RAM de la máquina, aunque se recomienda que no sea mayor a 1,5 GB. La partición SWAP no se monta en ningún directorio.

Sin embargo las demás particiones sí deben ir montadas en directorios y ademaś se les debe definir un File System (Sistema de Archivos). Linux soporta una gran gama de FS distintos, pero el más utilizado en las versiones 2.6 del Kernel es el tipo **ext3**.

También es destacable el hecho de que desde un sistema Linux se puede leer y escribir cualquier tipo de partición de Windows, tanto FAT, FAT32 o NTFS, mientras que desde Windows no se puede trabajar con particiones Linux.

Ejemplos de particiones en Linux:

[root@localhost /]# fdisk /dev/hdc

The number of cylinders for this disk is set to 79656.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
(e.g., DOS FDISK, OS/2 FDISK)

Command (m for help): p

Disk /dev/hdc: 41.1 GB, 41110142976 bytes
16 heads, 63 sectors/track, 79656 cylinders
*Units = cylinders of 1008 * 512 = 516096 bytes*

Device Boot	Start	End	Blocks	Id	System
/dev/hdc1 *	16	40641	20474842+	f	W95 Ext'd (LBA)
/dev/hdc2	40642	79656	19663560	83	Linux
/dev/hdc5	17	39175	19735821	7	HPFS/NTFS
/dev/hdc6	39175	40641	738958+	82	Linux swap / Solaris

Distribución de Software en LINUX

En Linux existen al menos tres formas de obtener software y varían en su distribución, instalación, configuración, complejidad y actualización.

1.- Distribución del código fuente
2.- Distribución de paquetes precompilados
3.- Distribución de paquetes precompilados por repositorios

Distribución del código fuente

Los desarrolladores de aplicaciones normalmente dejan a disposición los códigos fuentes para que sean descargados, configurados, modificados si es que se desease, compilados y se instalen los ejecutables. Los códigos vienen siempre comprimidos e incrustados en tarballs.

Este método tiene algunas ventajas y desventajas. Las principales ventajas son que siempre es la última versión de los software y permiten mantener sistemas muy actualizados. La desventaja más grande es que no son fáciles de compilar. Es común encontrarse con problemas en la primera etapa (configurar), donde se requieren las librerías necesarias para trabajar y se debe comenzar la tarea de instalar estas dependencias. Pero puede ocurrir que se instale una librería que para su correcto funcionamiento dependa de una segunda librería. En este caso se genera lo que se conoce como "árbol de dependencias".

El formato de compresión más utilizado es **.tar.gz** seguido de **.tar.bz2**. A diferencia de .zip y .rar, los primeros son mucho más efectivos y pueden llegar a comprimir hasta un 50% más.

Para descomprimir un archivo en formato **.tar.gz** se utiliza el siguiente comando:

```
$tar -xvzf archivo.tar.gz
```

El formato .tar indica que el contenido viene empaquetado en un objeto llamado Tarball. El Tarball sirve para unificar muchos archivos dentro de uno solo. El .gz indica que el paquete fue comprimido utilizando **gzip**.

La sintaxis de **tar** significa lo siguiente:
x = eXtraer
v = Verbose (salida detallada)
z = Descomprimir un .gz.
f = File (archivo)

Para los paquetes en formato **.tar.bz2** se utiliza practicamente la misma sintaxis, a diferencia del formato:

```
$tar -xvjf archivo.tar.bz2
```

Solamente se ha reemplazado la "z" por la "j", la cual indica que debe descomprimir un archivo bzip2. Una vez descomprimido los paquetes, se creará un directorio con el mismo nombre de los archivos comprimidos menos la extensión. En el caso anterior se crearía un directorio llamado "archivo".

Dentro de ese directorio hay varios archivos importantes que hay que considerar. El primero lleva por nombre **README** o **INSTALL** (pueden estar ambos juntos) y se escriben con mayúsculas para que les pongamos atención y se lean antes de hacer cualquier cosa.
Otro archivo importante es uno denominado "configure" el cual es un script ejecutable que tiene como misión recopilar información de nuestro sistema para ver si cumple con los requisitos para instalar el software que queremos compilar. Por ejemplo aquí pediría las librerías, compiladores, lenguajes de programación, etc.

El script "configure" se puede ejecutar de dos formas:

```
$sh configure
```
o bien
```
$./configure
```

Y por último, una vez que se haya completado la fase de configuración es necesario compilar. Para esto se utiliza generalmente la orden **make** (crear). Una vez que se haya compilado correctamente el código fuente, el comando **make install** envía los archivos creados a sus directorios pertinentes (como /bin, /etc, etc.).

Distribución de paquetes precompilados

Para hacer la tarea de instalación más fácil que instalar desde el código fuente se crearon los paquetes o binarios precompilados. Estos son paquetes instalables que vienen practicamente listos y en tan solo una instrucción ellos se configuran, compilan completamente, se distribuyen en los directorios y se instalan.

Los formatos más conocidos son los **.rpm** y **.deb**. Los paquetes de tipo .rpm sirven actualmente para plataformas basadas en RedHat (RPM = Redhat Package Manager) tales como Fedora, CentOS, etc.

Los .deb se instalan en sistemas derivados de Debian, como Knoppix y Ubuntu.

Cuando se descarga una aplicación en formato precompilado se puede instalar de la siguiente manera:

RPMs:
```
$rpm -Uvh nombredelpaquete1-0.1.rpm
```
o
```
$rpm -i nombredelpaquete1-0.1.rpm
```

DEBs:
```
$dpkg -i nombredelpaquete1-0.1.deb
```

Existe una herramienta llamada **alien** la cual puede transformar paquetes .rpm a .deb y viceversa.

Distribución de paquetes precompilados por repositorios

Los repositorios son sitios web, generalmente espejos de los oficiales, que contienen una lista actualizada de paquetes precompilados, librerías y todo lo necesario para instalar software de manera casi automática. Las herramientas de gestión de repositorios se encargan de:
- Actualizar sus índices contra el repositorio

- Actualizar el sistema
- Descargar el paquete precompilado
- Descargar las librerías y dependencias necesarias
- Configurar los paquetes
- Recompilarlos
- Instalarlos

Son sin duda una forma muy útil y práctica para conseguir software rápidamente. Las herramientas más utilizadas son:

- **yum** para sistemas basados en RedHat
- **apt-get** o **aptitude** para sistemas basados en Debian
- **urpmi** para Mandrake
- **YaST** para SuSE

Por ejemplo, si se quisiera instalar el sistema CUPS de impresión (Common Unix Printing System) con **apt-get** se haría lo siguiente:

```
[root@localhost /]# apt-get install cups
Reading Package Lists... Done
Building Dependency Tree... Done
The following extra packages will be installed:
  cups-common libcups2    <--------- Paquetes extras necesarios
The following packages will be upgraded
  cups cups-common libcups2
3 upgraded, 0 newly installed, 0 removed and 252 not upgraded.
Need to get 3021kB of archives.
After unpacking 528B of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://spout.ussg.indiana.edu pclinuxos/2007/main libcups2 1.2.4-
2pclos2007 [138kB]
Get:2 http://spout.ussg.indiana.edu pclinuxos/2007/main cups-common 1.2.4-
2pclos2007 [398kB]
Get:3 http://spout.ussg.indiana.edu pclinuxos/2007/main cups 1.2.4-2pclos2007
[2485kB]
Fetched 3021kB in 5m54s (8527B/s)
Committing changes...
    Preparing...
########################################### [100%]
   1:libcups2
########################################### [ 33%]
   2:cups-common
########################################### [ 67%]
   3:cups
########################################### [100%]
Done.
[root@localhost /]#
```

En este caso **apt-get** realizó lo siguiente:
- Se conectó al repositorio configurado
- Consultó si existía algún software llamado "cups"
- Encontró que sí, pero que depende de dos paquetes más: cups-common y libcups2.
- Los agregó a su lista de descarga
- Consultó si se le daba la aprobación para instalar esos paquetes también

- Descargó los paquetes
- Los desempaquetó y configuró
- Instaló los binarios, archivos de configuración, documentación y otros en los directorios que correspondían.