La mayoría de utilidades para usar y administrar Linux se ejecutan escribiendo comandos o líneas de comandos, a éstos interpretes se les llama shell. Los comandos se escriben proporcionando correctamente la sintaxis y pueden incluir parámetros, datos, variables, etc.

Claves de acceso

Originalmente el administrador del equipo asigna una contraseña o password al usuario. Es responsabilidad de usuario modificar esta contraseña para obtener mayor seguridad. La contraseña es intransferible y se debe cumplir ciertas condiciones para diseñarla:

- Las claves pueden contener caracteres no alfanuméricos así como también letras y números.
- Al menos 5 caracteres.
- Puede ser de más de 8 caracteres, pero solo se reconocen los primeros 8.
- Debe contener al menos una letra mayúscula o 1 dígito.
- Se recomienda cambiarse constantemente.

passwd: Se usa para cambiar la contraseña

Ejemplo:

```
passwd
```

Introduciendo sólo `passwd` te permite cambiar la cambiar la contraseña. Después de introducir `passwd` recibirás la siguiente respuesta:

```
Current Password:

New Password:

Confirm New Password:
```

Cada uno de estos datos debe ser introducido correctamente para que la contraseña se cambie satisfactoramente.

Descripción de los Comandos

Documentación

El comando man provee información sobre el uso correcto de los comandos y se encuentra en /usr/bin/man.

man: Ve las páginas del manual en línea

Para ver las páginas del manual, suponiendo que estén en línea.

Ejemplos:

```
man cp
```

Nos da la información en el manual sobre el comando `cp`. Para averiguar más sobre el comando man, pruebe

```
man man
```

apropos: Lista los comandos relacionados con un tema en particular.

Ejemplo:

```
apropos man
```

ls: Lista los archivos de un directorio

Probablemente el comando más usado en Linux, `ls` nos permite ver el contenido de un directorio y opcionalmente sus subdirectorios.

Quizás uno de los comandos más utilizados, sirve para listar archivos.
Su sintaxis es:

```
ls [opciones] [archivo o carpeta...]
```

Si se ejecuta `ls` sin argumentos, dará como resultado un listado de todos los archivos (incluyendo directorios) del directorio donde el usuario está posicionado. Sus opciones son:

-a Lista todos los archivos, incluyendo aquellos que comienzan con un «.».
-d Lista el nombre del directorio en vez de los archivos contenidos en él.

-l Lista los archivos con mucho más detalle, especificando para cada archivo sus permisos, el número de enlaces rígidos, el nombre del propietario, el grupo al que pertenece, el tamaño en bytes, y la fecha de modificación.
-r Invierte el orden de listado de los archivos.

cd: Cambio de directorio

Con `cd` cambiamos el directorio donde estamos trabajando.

Ejemplos:

Cambio absoluto de directorio:

```
cd /usr/local/bin
```

Cambia al directorio anterior

```
cd ..
```

Cambia al directorio home

```
cd ~
```

Se regresa al directorio raíz

```
cd /
```

pwd: Da el nombre del directorio actual

Nos recuerda, cuando estamos perdidos, del nombre del directorio actual. `pwd` nos da el camino completo.

```
pwd
```

mkdir: Crea un directorio

Crea un directorio vacio en el directorio actual, por ejemplo

```
mkdir xyz
```

Crea el directorio `xyz` en el directorio actual.

rm: Borra archivos

Se puede usar la opción `-i` para pedir confirmación de cada archivo a borrar:

```
$rm -i archivos(s)
```

La opción `-r` borra archivos y directorios al mismo tiempo:

```
$rm -r trayectoria/
```

rmdir: Borra directorios vacíos:

```
$rmdir directorio/
```

touch: Crea archivos

```
touch nombre del archivo
```

Redireccionando la entrada y la salida estándar

El redireccionamiento de la salida o entrada estándar significa que la entrada viene de otro lado que no sea el teclado de la terminal y que la salida va hacia otro lado que no sea la pantalla de la terminal.

Linux reconoce tres canales de datos estándar:

- entrada estándar: por ella lee las instrucciones el programa ejecutado actualmente, la mayoría de las veces es el teclado,
- salida estándar: hacia allí, normalmente la pantalla, manda el programa sus salidas,
- salida estándar de errores: a través de este canal se escriben los mensajes de error.

Cada uno de estos canales puede ser redireccionado por la shell.

Cada uno de estos canales de entrada-salida está coordinado por un número:
entrada estándar ("stdin"): 0
salida estándar ("stdout"): 1
salida estándar de errores ("stderr"): 2

"`stdin`", "`stdout`" y "`stderr`" normalmente aparecen siempre en la pantalla pero pueden ser redireccionados.

Con < se redirecciona "`stdin`" y con > "`stdout`" (es =a 1>), donde > sin número delante es sólo una abreviatura de 1 > . Sin embargo, el número 2 delante de > indica a la "shell" que debe de ser redireccionada la salida estándar de errores.

Carácter	**Resultado**
comando < archivo	Toma la entrada de archivo
comando > archivo	Envía la salida de comando al archivo; sobreescribe cualquier cosa del archivo
comando 2> archivo	Envía la salida de error de comando al archivo
comando >> archivo	Añade la salida de comando al final del archivo
comando 2>&1	Envía la salida de error a la salida estándar
comando &> archivo	Envía la salida estándar y de error a archivo; equivale a comando > archivo 2>&1

Ejemplos

```
>       $date > fecha.txt
        $cat fecha.txt

>>      $time >> fecha.txt
        $cat fecha.txt

2>      $ls /root 2> error.txt
        $cat error.txt

2>>     $ifconfig 2>> error.txt
        $cat error.txt

&>      $find /home/ –name “*.txt” &> salida.txt

<       $tac < archivo.txt
```

Ejemplos Salida estándar

```
ls
ls -F /usr/bin > listado
ls
listado
cat listado > fichero
```

La salida del comando "`cat listado`" es el fichero listado. Así hemos inventado un nuevo (y no tan eficiente) método de copiar ficheros.

```
/home/larry$ cat > zorro
```

El rápido zorro marrón salta sobre el descuidado perro.
pulsa Ctrl+d

Ahora se ha creado el fichero zorro que contiene la frase "El rapido zorro marron salta sobre el descuidado perro". Un último uso del versátil comando cat es concatenar ficheros. cat imprimirá cada fichero dado como parámetro, uno despues de otro. El comando "**cat listado zorro**" imprimirá el listado del directorio /usr/bin, y luego la tonta frase. Así, el comando "**cat listado zorro > listyzorro**" creará un nuevo fichero conteniendo los contenidos de listado y zorro.

cat: Concatena o ve el contenido de un archivos

La función "oficial" de `cat` es de pegar o encadenar archivos. El archivo resultado va a stdout. Cuando hay un solo archivo este aparece por pantalla. Por eso, `cat` se usa mucho para ver el contenido de un archivo, aunque para eso es mejor `more`.

Ejemplos:

```
cat > filename.txt
```

Presione [Intro] para ir a una línea vacía y utilice las teclas **[Ctrl]+[d]** para salir de `cat`.

El comando `cat` visualizará también los contenidos de un archivo entero en la pantalla (por ejemplo, teclee `cat filename.txt`). Si un archivo es bastante largo, se desplazará rápidamente y por completo por la pantalla. Para evitar esto, use el comando `cat filename.txt | less`.

head y tail: Muestra porciones de un archivo

Por omisión el comando `head` despliega las primeras 10 líneas de un archivos y el comando `tail` despliega las últimas 10 líneas de un archivo. El formato de los dos comandos es:

```
head [-número] archivo

tail [-numero] archivo
```

Ejemplos:

```
head - 2 nuevo2

tail -2 nuevo2
```

Dónde: número es el número de líneas a desplegar ya sea al inicio o al final.

more: Muestra el contenido de un archivo

Despliega el contenido del archivo una pantalla a la vez se utiliza el comando

```
$more archivo
```

cp: Copiar archivos

El comando `cp` se usa para realizar copias de archivos, el formato es:

```
$cp archivo nuevo_archivo
```

Para que el comando pida confirmación para copiar si ya existe un archivo con el nombre del nuevo_archivo se usa la opción `-i`:

```
$cp -i texto texto1
```

Para copiar todo el contenido de un directorio y todos los subdirectorios a un destino específico se usa la opción `-R`:

```
$cp -R /usr/juan /usr/sofia
```

scp: Copiar archivos entre servidores

El comando `scp` utiliza por defecto el puerto 22, y se conecta mediante un enlace encriptado ssh

Se puede utilizar `scp` para copiar archivos de un servidor local a otro remoto, también se puede copiar del remoto al local y también se puede copiar entre dos remotos, mientras estas conectado a un tercer servidor, y el tráfico no pasará por el servidor en que estás.

Se puede usar `scp` en Linux, Mac y Windows.

Copiar un archivo local a un destino remoto

```
scp /ruta/al/archivo-origen usuario@servidor:/ruta/al/directorio-destino/
```

Copiar un directorio y todo su contenido a un directorio en el servidor remoto

```
scp -r /ruta/al/directorio-origen usuario@servidor:/ruta/al/directorio-destino/
```

mv: Mover archivos entre directorio (o renombrar)

El comando `mv` es similar a `cp`, excepto que borra el origen. En otras palabras, mueve archivos de un directorio a otro, o de un archivo a otro. En este último caso, como el original desaparece, `mv` puede a veces tener efectos inesperados. El último argumento de `mv` indica el destino del movimiento; los primeros son los orígenes.

Un uso muy frecuente de `mv` es de cambiar el nombre a un archivo. Supongamos, por ejemplo, que `viejo` existe y le queremos cambiar al nombre `nuevo`; nos aseguramos primero con `ls` que el nombre `nuevo` no existe; luego hacemos

```
mv viejo nuevo
```

con lo cual `viejo` queda rebautizado a `nuevo`.

Ahora suponemos que `subdir` es un directorio. Para mover archivos a este directorio, pudiéramos usar

```
mv xyz uvw subdir
```

En este caso, `xyz` y `uvw` se mueven al subdirectorio `subdir`. Pero si `subdir` fuese un archivo, o no existiese, este comando mueve, esencialmente, renombra `xyz` a `uvw` un comportamiento probablemente inesperado. Cuidado se pierde el archivo `uvw` original. Se pueden evitar accidentes con la opción interactiva:

```
mv -i xyz xxx
```

En este caso, `mv` pregunta antes de mover (renombrar); esto es bueno en general pero latoso si hay que mover grandes cantidades de archivos.

find: Encontrar archivos

El comando `find` busca archivos que cumplan las condiciones que especifique el usuario, comenzando por el directorio que nombre. Por ejemplo, si quiere buscar nombres de archivos que concuerden con determinado patrón o que hayan sido modificados durante un periodo de tiempo determinado.

```
$ find directorio opciones
```

Donde directorio es el nombre del directorio inicial y opciones representa las opciones del comando `find`.

Por ejemplo, para ver que archivos del directorio en uso y sus subdirectorios terminan en s, escriba:

```
find . -name '*s'
```

Búsqueda por nombre

```
-name nombre_de_archivo

find ~/ -name nombre del archivo
```

Selecciona archivos cuyo elemento situado más a la derecha concuerda con `nombre_de_archivo`. Escriba `nombre_de_archivo` entre comillas si éste incluye patrones de sustitución de nombre de archivo.

Para buscar por tipo de extensión

```
find ~/ -name " *.extension_del_tipo_de_archivo"

find ~/ -name  "*.txt"
```

locate: Busca archivos

`Locate` es un comando de búsqueda de archivos, bastante parecido al comando `find`. La diferencia de `locate` es que la búsqueda la hace en una base de datos indexada para aumentar significativamente la velocidad de respuesta. Esto quiere decir, que `locate` realmente no busca en el disco del sistema, sino que en un archivo con la lista de todos los archivos que existen en el GNU/Linux. Generalmente todas las distribuciones de GNU/Linux ejecutan a una hora determinada (generalmente cerca de las 4:00am, ya que

tarda algún tiempo realizar esta tarea) un comando para actualizar la base de datos que utiliza `locate`, dicho comando se llama **updatedb**. Su sintaxis es:

```
locate nombre_del_archivo
```

vi: Editor de texto

Para abrir el editor damos la orden `vi nombre_del_archivo` si queremos abrir o editar algún archivo, o simplemente `vi`, y entramos al editor en blanco y crear un archivo desde cero.

Órdenes básicas.

Moverse a la izquierda	**h**
Moverse a la derecha	**l**
Moverse arriba	**k**
Moverse abajo	**j**
Insertar texto	**i**
Borrar caracter (como Supr)	**x**

En `vi` las instrucciones se realizan con una orden del teclado `Esc`, precedida de dos puntos '`:`'. Algunas órdenes importantes son:

Salir sin grabar los cambios	**q**
Salir grabando los cambios	**x**
Salir grabando los cambios	**wq**
Salvar los cambios actuales	**w**
Salvar como archivo	**w archivo**
Insertar desde el cursor archivo	**r archivo**
Editar archivo	**e archivo**
Guarda el documento con el nombre especificado	**:file *nombre***

Para forzar la salida sin salvar , debemos escribir **:q!**.

Hay una orden con la que se tiene que tener mucho cuidado, esa es la orden **:x**, porque si por accidente ejecutas la orden **:X** `Vi` lo guardará, pero después de encriptarlo. Primero te pedirá la clave para la encriptación. Si eso no es lo que quieres, se recomienda darle un **kill** al `Vi`.

Comandos Administración

df: Muestra el espacio libre en discos

Ejemplos:

```
df –h
```

-h Mostrar los tamaños en formato legible por humanos (1K 234M 2G)

du: Informa de cuánto espacio en disco ocupa un archivo o directorio.

```
du [opciones] directorios
```

Opciones:

-a	Muestra el uso de espacio de cada archivo.
-k	Escribe el tamaño de los archivos en unidades de 1024 octetos, en vez de las unidades de 512 octetos por defecto.
-s	En vez de la salida por defecto, informa sólo de la suma total de cada uno de los archivos especificados.
-h	Muestra la capacidad de la carpeta actual

```
du –h
84K
```

clear: Limpia la pantalla

Comandos informativos

Comando	Descripción
logname	Muestra el login actual
hostname	Muestra o establece el nombre de la máquina
w	Informa sobre los usuarios conectados y sus procesos
whatis *comando*	Breve descripción del comando
which *comando*	Busca la ubicación del comando en los directorios del PATH
whereis *comando*	Directorio, página de manual y fuente del comando
who	Muestra los usuarios conectados al sistema
whoami	Muestra el nombre del usuario -user id

chmod: Cambia los permisos de un archivo o un directorio

Para usarlo debe ser el propietario del archivo o del directorio

```
$chmod permisos nombre_archivo
```

Los permisos pueden especificarse de varias formas. A continuación se cita uno de los métodos más sencillos:

Usar una o más letras para indicar los usuarios implicados:

u (para el usuario)
g (para el grupo)
o (para otros)
a (all; para todas las categorías anteriores)

Indicar si los permisos se van a agregar (+) o eliminar (-).

Utilizar una o más letras para indicar los permisos:

r (read; de lectura)
w (write; de escritura)
x (execute; de ejecución)

Modo octal
Como resultado de la combinación de los tres tipos de permisos (lectura, escritura y ejecución), con las tres clases de usuarios (dueño, grupo y otros), se obtiene 23 = 8 permisos en total que pueden ser asignados o denegados de forma independiente.
La base 8 se utiliza habitualmente para que exista un dígito por cada combinación de permisos (un bit a modo de bandera por cada permiso, con valor 1 ó 0 según el permiso esté concedido o denegado).
Así, las posibles combinaciones se resumen en números octales de tres dígitos del 000 al 777, cada uno de los cuales permite establecer un tipo de permiso distinto a cada clase de usuario:
El primer dígito establece el tipo de permiso deseado al dueño; el segundo al grupo; y el tercero al resto de los usuarios.

Número	Binario	Lectura (r)	Escritura (w)	Ejecución (x)
0	000	No	No	No
1	001	No	No	Si
2	010	No	Si	No
3	011	No	Si	Si
4	100	Si	No	No
5	101	Si	No	Si
6	110	Si	Si	No
7	111	Si	Si	Si

Ejemplo: Permisos para garlic

Permiso	Usuario	Grupo	Otros
Lectura	4	4	0
Escritura	2	2	0
Ejecución	1	1	1

Total	7	7	1

Por ejemplo:

```
chmod 766 file.txt  # brinda acceso total al dueño y lectura y escritura a los demás
chmod 770 file.txt  # brinda acceso total al dueño y al grupo y elimina todos los permisos a los demás usuarios
chmod 635 file.txt  # Permite lectura y escritura al dueño,  escritura y ejecución al grupo  y lectura y ejecución al resto
```

Modo carácter

Posee 3 modificadores que permiten realizar la tarea:

+ – añade un modo
– – elimina un modo
= – específica un modo (sobrescribiendo el modo anterior)

Por ejemplo:

```
chmod +r arch.txt      # agrega permisos de lectura a todos los usuarios
chmod u+w arch.txt  # agrega permisos de escritura al dueño
chmod –x arch.txt     # elimina el permiso de ejecución a todos los usuarios
chmod u=rw,go= arch.txt  # establece los permisos de lectura y escritura al dueño y elimina todos los permisos a  los demás usuarios
```

En el ejemplo siguiente, se agrega un permiso de lectura al directorio carrots para los usuarios que pertenezcan al mismo grupo (de esta forma, permisos es g+w y nombre es carrots):

```
$ ls -l carrots
drwxr-xr-x 3 user2 1024 Feb 10 11:15 carrots
$ chmod g+w carrots
$ ls -l carrots
drwxrwxr-x 3 user2 1024 Feb 10 11:15 carrots

$
```

zip: Comprimir archivos

```
zip -r documento.zip carpeta_o_archivo_a_comprimir
```

El parámetro `r` indica al comando `zip` que debe hacer una compresión de manera recursiva, en el caso de comprimir un directorio, comprime todo lo que hay en él.

unzip: Descomprimir archivos zip

```
unzip documento.zip
```

Esta línea descomprime el fichero `zip` , y deja todos los archivos contenidos en la carpeta donde te encuentres en ese momento.

tar: empaquetar archivos

El comando `tar` es utilizado normalmente para empaquetar archivos. El comando `tar` no comprime automáticamente los archivos mientras los empaqueta. El formato del comando `tar` es:

```
tar <opciones> <archivo1> <archivo2> ...<archivoN>
```

`<opciones>` es la lista de comandos y opciones para `tar`, `<archivo1>` hasta `<archivoN>` es la lista de archivos a añadir o extraer del archivo empaquetado.

Opción	Descripción
c	Crea un nuevo archivo tar.
v	Modo verbose, quiere decir que mostrará por pantalla las operaciones que va realizando archivo por archivo, si no se pone esta opción ejecutará la acción pero en pantalla no veremos el proceso.
x	Extrae los archivos (Descomprime los ficheros que se encuentran dentro del archivo tar).
t	Nos muestra el contenido del archivo tar. Esto es cuando tu deseas saber que es lo que contiene ese archivo sin necesidad de desempaquetarlo.

p	Mantiene los permisos originales de los archivos.
f	Cuando se usa con la opción -c, usa el nombre del archivo especificado para la creación del archivo tar; cuando se usa con la opción -x, retira del archivo el archivo específico.
Z	Comprime el archivo tar con gzip.
J	Comprime el archivo tar con bzip2.

Por ejemplo, el comando

```
# tar cvf backup.tar /etc
```

empaquetará todos los archivos de `/etc` en el archivo backup.tar.

En el primer argumento de `tar ("cvf")`, la opción `"c"` le dice a tar que cree un nuevo archivo (create). La opción `"v"` fuerza a tar en el modo verbose, imprimiendo los nombres de los archivos según se empaquetan. La opción `"f"` le dice a tar que el siguiente argumento (backup.tar) es el nombre del archivo a crear. **El resto de los argumentos de tar son los nombres de archivos y directorios a añadir al archivo empaquetado.**

Descomprimir archivos con tar

```
# tar xvf backup.tar
```

Esto, extraerá el archivo backup.tar en el directorio actual. Esto puede ser peligroso, porque si el archivo ya existía se sobrescribirá. Por otra parte, antes de extraer archivos tar es importante conocer donde se deben desempaquetar.

Por ejemplo, digamos que se empaquetaron los siguientes archivos: /etc/hosts, /etc/group, y /etc/passwd. Si se usó:

```
# tar cvf backup.tar /etc/hosts /etc/group /etc/passwd
```

el nombre de directorio /etc se añadió al principio de cada nombre de archivo. Para poder extraer los archivos en el directorio correcto, se necesitará utilizar los siguientes comandos:

```
# cd /
# tar xvf backup.tar
```

Ya que los archivos se extraen con el nombre de camino almacenado. Sin embargo, si se empaquetaron los archivos con los comandos:

```
# cd /etc
# tar cvf hosts group passwd
```

Los nombres de directorio no se salvaron en el archivo empaquetado. Por esto se necesitara hacer "cd /etc" antes de extraer los archivos. Como se puede ver, el cómo haya sido creado un archivo tar marca una gran diferencia en como se extrae. Se puede usar el comando:

```
# tar tvf backup.tar
```

para mostrar un "índice" del archivo tar antes de desempaquetarlo. De esta forma se puede ver que directorio se utilizó como origen de los nombres de los archivos, y se puede extraer el archivo desde la localización correcta.

Además agregando el switch 'z' utiliza el gzip y el gunzip para comprimir / descomprimir.

```
$ tar czvf archivo.tar.gz /etc
```

Si queremos descomprimir igual que en la línea que desempaquetamos, solo que ahora cambiamos la c por la x de extracción:

```
$tar xzvf archivo.tar.gz
```

Hay otro modo de compresión que es el **formato bz2.** Para comprimir y descomprimir es el mismo procedimiento que con gz, la única diferencia es que ya no va la letra **z**, sino la **j.**

Ejemplo - Para poder empaquetar y comprimir se usa el siguiente comando:

```
$ tar cjvf archivo.tar.bz2 /etc
```

Si queremos descomprimir , las opciones son muy similares a la orden anterior, solo que ahora le cambiamos la c por la x de extracción:

```
$tar xjvf archivo.tar.gz
```

Archivos .bashrc y .bash_profile

Bashrc y Bash_Profile son dos archivos ocultos ubicados en el directorio home de cada usuario, pero en realidad los dos tienen la misma función, ejecutar comandos al inicio de cada sección, por ejemplo para definir variables para nuestras rutas, crear alias, y cualquier otro comando que se nos ocurra.
De acuerdo con la página de manual de `bash` , `.bash_profile` se ejecuta en los depósitos de inicio de sesión, mientras que `.bashrc` se ejecuta para interactivo.
En `.bash_profile` los comandos se ejecutan en non-login shell, esto es via SSH, por consola, cada vez que nos logueamos por xdm, etc... y los de `.bashrc` se ejecutan en cada instancia de bash, por ejemplo cada vez que abrimos una ventana de xterm.
`.bash_profile` se puede usar cuando se quiere que tal comando se ejecute solamente cuando inicie la maquina por primera vez o en una sección remota, un ejemplo es que un comando genere el estado del sistema cada vez que se inicie la maquina con nuestro usuario, porque si ese comando se pone en `.bashrc`, cada vez que se ejecute la terminal se correrá ese proceso.

script: Programación de Shells

En la administración de sistemas muchas veces significa realizar de forma reiterada las mismas actividades: controlar por qué ya no hay espacio en el disco duro, comprobar si el "spooler" de impresión sigue funcionando, etc.
Muchas de estas tareas requieren a veces más de un comando. El orden en el que todo esto se lleva a cabo es importante.
Una vez reconocido un proceso así, se ofrece la oportunidad de incluir todo en un programa que realiza el trabajo con una sola llamada. Programa por suerte no significa escribir un programa en C, si no que la "shell" de Linux permite incluir todos los comandos en un archivo y hacerlos ejecutar como si se estuviese en la consola tecleándolos.

Creación de script

ejemplo del clasico "Hola Mundo" en Bash.

```
#!/bin/bash
#
# Esto es un ejemplo en Bash del clásico "Hola Mundo"

echo "Hola Mundo"
```

`#!/bin/bash`: Esta línea indica donde se encuentra el interprete de comandos en nuestro sistema. Por defecto todos los sistemas que tengan Bash instalado, lo tendrán en el directorio `/bin`. Al utilizar esta línea, se puede ejecutar el script como un programa normal, ya que el sistema sabrá que es un script en Bash y que tiene que hacer con él.
Si el script de ejemplo se hubiera grabado como ejemplo.sh, se podría ejecutar de la siguiente manera:

```
# chmod ugo+x ejemplo.sh

# ./ejemplo.sh

Hola Mundo
```

Esto es un ejemplo en Bash del clásico "Hola Mundo". Todas las líneas que empiecen con el símbolo '#' serán tratadas como comentarios y no se ejecutaran.
`echo "Hola Mundo"`: Esto es el comando que imprime la cadena de texto en pantalla.

Ejecución de Scripts

Si los permisos de ejecución están establecidos, entonces se puede llamar a un "script" sencillamente escribiendo su nombre, se recomienda no utilizar nombres que el sistema ya ha utilizado para comandos.
Si el directorio actual no está en el "`path`", se puede llamar al "script" con un "`./`" delante. Para probar un "script" lo más sencillo es ampliar el "path" con un ".", es decir el directorio actual, ya que de esta forma se pueden hacer pruebas con el "script" en el directorio fácilmente. La inclusión del punto en el "path" debería de realizarse al final, ya que si no es así, es posible la visita de usuarios "malintencionados" y la ejecución de comandos no esperados.
Los "scripts de shell" así lanzados siempre se procesan en una "shell" nueva y por eso se les transmite la parte exportada del entorno. Si los "scripts" cambian el entorno, estos cambios no tendrán efecto en la "shell" desde la que se les llama.

```
tierra:~ # . /dia
```

ldd: Averiguar las dependencias de librerías de un programa
ldd es un comando muy interesante que nos muestra las librerías que necesita un programa o librería compartida.

Por ejemplo: Si quiero ver qué librerías necesita bash, ejecuto:

ldd /bin/bash

Y obtendré una salida como la siguiente:

linux-gate.so.1 => (0xb80de000)
libncurses.so.5 => /lib/libncurses.so.5 (0xb8095000)
libdl.so.2 => /lib/i686/cmov/libdl.so.2 (0xb8091000)
libc.so.6 => /lib/i686/cmov/libc.so.6 (0xb7f35000)
/lib/ld-linux.so.2 (0xb80df000)

Ojo, al comando ldd tenemos que proporcionarle la ruta completa al archivo (Ej: ldd /bin/bash). No vale con hacer: ldd bash. Si no sabemos dónde se encuentra localizado el programa o librería, siempre podemos buscarlo con el comando whitch.