



Páginas

- [Página principal](#)
- [Índice de guías](#)
- [Foro](#)
- [Vídeo tutoriales](#)
- [Sobre mí](#)
- [Guía Oficial](#)



Como hacer un Script en Ubuntu

Este post es para usuarios avanzados y está sacado de <http://doc.ubuntu-es.org>



DEFINICION

Un **Script** (o guion) para **Bash** es un archivo de texto que contiene una sucesión de comandos de **Shell** que pueden ejecutar diversas tareas de acuerdo al contenido del texto del guión. De esta forma pueden automatizarse muchas acciones para alguna necesidad particular o para la administración de sistemas. El guión debe escribirse en un orden lógico pues Bash ejecutará el guión en el orden en que se escriben las líneas, de la misma forma que cuando se realiza una tarea cualquiera por una persona, por ejemplo; *primero hay que poner la escalera y luego subirse.*

Los Scripts de Bash deben tener siempre como primera línea del guión el texto, para que el **Sistema Operativo** ejecute la acción usando el programa Bash.

`#!/bin/bash`

Una vez escrito el Script y guardado en el disco en alguno de los directorios "bin" con el nombre y **permiso de ejecución** apropiados, se invoca, escribiendo en la consola el nombre del guión. Si el guión tiene una interfaz gráfica se invoca como otro programa cualquiera, uno o dos clic sobre el guión o su icono. Este puede escribirse en cualquiera de los editores de texto de Linux, por ejemplo **Kwrite** y será ya un guión funcional cuando se salve a alguno de los "bin".

Nota: Es buena práctica cuando se escribe un guión salvarlo apenas se hayan escrito las primeras línea para ir comprobando su funcionamiento e ir corrigiendo los problemas.

VARIABLES

Es impensable elaborar Scripts de Bash sin el uso de las **variables**. Una variable es una estructura de texto (una letra, un número o sucesiones de ellos) que representa alguno de los elementos que varían en valor y/o significado en el entorno de la Shell, sirviendo como elemento básico de entrada/salida de valores a y desde los comandos en su ejecución consecutiva. Para invocar una variable se utiliza el carácter especial **\$** precediendo al nombre de la variable.

Existen dos tipos de variables:

1. Variables intrínsecas de Bash.

Estas son elaboradas por defecto por el propio Bash y son:

- **\$0** -> Nombre del guión
- **\$1...\$n** -> Variables que almacenan los n argumentos (opciones) proporcionados al comando.
- **\$#** -> Variable que contiene el total de los argumentos proporcionados.

- **\$*** -> Conjunto de los argumentos.
- **\$?** -> Valor de ejecución del comando anterior, si es cero es que el comando anterior se ejecutó sin errores, de lo contrario hubo algún error.
- **\$\$** -> Identifica el proceso del guión.
- **#!** -> Identifica el último proceso arrancado en el trasfondo (background).

2. Variables creadas por el programador.

Las variables pueden ser creadas en cualquier momento, pero siempre antes de su utilización de manera muy simple, se escribe:

nombre_variable=valor_variable

En cualquier momento posterior a la creación si se coloca \$nombre_variable dentro del entorno de la Shell el sistema colocará allí valor_variable.

SALUDO=Bienvenido

En cualquier momento posterior si se pone \$SALUDO, Bash colocará ahí Bienvenido.

Una variable también puede ser la salida de un comando si ponemos al principio y final del mismo un acento invertido.

SALIDA=`comando`

Le indicará al sistema que donde se escriba \$SALIDA debe poner la salida de ese comando. Es práctica común utilizar mayúsculas para las variables a fin de identificarlas fácilmente dentro del guión.

Cuando se ejecutan Scripts que pueden ser "hijos" de otro guión en ocasiones es necesario exportar las variables, esto se hace escribiendo:

export nombre_variable

3. Caracteres especiales.

Existe un grupo de caracteres especiales (también llamados meta caracteres) que tienen significado propio para Bash. Algunos son:

- **** -> Le indica a Bash que ignore el carácter especial que viene después.
- **" "** -> Cuando se encierra entre comillas dobles un texto o una variables si esta es una frase (cadena de palabras) Bash lo interpretará como una cadena única.
- **\$** -> Identifica que lo que le sigue es una variable.
- **' '** -> Las comillas simples se usan para desactivar todos los caracteres especiales encerrados dentro de ellas, así tenemos que si escribe '\$VARIABLE' Bash interpreta literalmente lo escrito y no como variable.
- **#** -> Cuando se coloca este carácter dentro de una línea del guión, Bash ignora el resto de la línea. Muy útil para hacer comentarios y anotaciones o para inhabilitar una línea de comandos al hacer pruebas.
- **;** -> Este carácter se usa para separar la ejecución de distintos comandos en una misma línea de comandos.
- **` `** -> Se utiliza como se explicó en el punto anterior, para convertir la salida de un comando en una variable. El comando en cuestión se ejecuta en una sub shell.

También están **|, (), !, >, <**, cuyo significado se verá mas adelante. El espacio es otro carácter especial y se interpreta por bash como el separador del nombre del programa y las opciones dentro de la línea de comandos, por esta razón es importante encerrar entre comillas dobles el texto o las propias variables cuando son una frase de varias palabras.

Otro carácter que debe evitarse en lo posible su uso es el guión (-) ya que para la mayoría de los programas se usa para indicarle al propio programa que lo que sigue es una de sus opciones, de manera tal por ejemplo, si usted crea un archivo con nombre -archivo (en caso que pueda) después será difícil borrarlo ya que rm (programa que borra) tratará el archivo como una de sus opciones (al "ver" el Script) y dará de error algo así, "Opción -archivo no se reconoce".

4. Palabras especiales.

Hay un grupo de palabras que tienen significado especial para bash y que siempre que se pueda deben evitarse cuando se escriben líneas de comandos para no crearle "confusiones" algunas son: exit, break, continue, true, false, return etc... cuyo significado es mas o menos así:

- **exit** -> Se sale del guión
- **break** -> Se manda explícitamente a salir de un ciclo
- **continue** -> Se manda explícitamente a retornar en un ciclo
- **return** -> Como exit pero solo se sale del comando u operación sin cerrar el guión
- **true** -> Indica que una condición es verdadera
- **false** -> Indica que una condición es falsa

5. Argumentos propios de Bash.

Bash como programa tiene algunos argumentos útiles y propios que se usan con frecuencia en la elaboración de Scripts en los condicionales vinculados a la determinación de elementos sobre los archivos, variables, cadenas de palabras o cadenas de pruebas, los mas comunes son:

1. Argumentos de Archivos -----> Cierto si.... (salida 0)

-d -----> Archivo existe y es un directorio
 -c -----> Archivo existe y es de caracteres
 -e -----> Archivo existe
 -h -----> Archivo existe y es un vínculo simbólico
 -s -----> Archivo existe y no está vacío
 -f -----> Archivo existe y es normal
 -r -----> Tienes permiso de lectura del archivo
 -w -----> Tienes permiso de escritura en el archivo
 -x -----> Tienes permiso de ejecución del archivo
 -O -----> Eres propietario del archivo
 -G -----> Perteneces al grupo que tiene acceso al archivo
 -n -----> Variable existe y no es nula
 Archivo1 nt Archivo2 -----> Archivo1 es mas nuevo que Archivo2
 Archivo1 -ot Archivo2 -----> Archivo1 es mas viejo que Archivo2

2. Argumentos de cadenas -----> Cierto si

-z -----> La cadena está vacía
 -n -----> La cadena no está vacía
 cadena1 = cadena2 -----> Si las cadenas son iguales
 cadena1 != cadena2 -----> Si las cadenas son diferentes
 cadena1 <> Si la cadena 1 va antes en el orden lexicográfico
 cadena1 >cadena2 -----> Si la cadena 1 va despues en el orden lexicográfico

6. Entrada / Salida.

En algunas ocasiones será necesario leer ciertas variables desde el teclado o imprimirlas a la pantalla, para imprimir a la pantalla se pueden invocar dos programas en la linea de comandos:

- **echo**
- **printf** (que es un echo mejorado)

Y para leer desde el teclado se usa:

- **read**

Si hacemos un read sin asignar variable, el dato de almacena en **\$REPLY** una variable del sistema. Tanto el comando **echo** como **read** tienen sus propias opciones.

Ejemplos:

1. Si creamos en una linea del Script una variable como un comando y queremos imprimir la variable a la pantalla podemos hacer algo así:

```
VARIABLE=`comando`  
echo "$VARIABLE"
```

La palabra \$VARIABLE está puesta entre comillas dobles para que se imprima todo el texto ignorando los espacios entre palabras.

2. Si escribimos en una linea del guión

```
read PREGUNTA
```

habremos creado una variable de nombre PREGUNTA así es que si luego ponemos

```
echo "$PREGUNTA"
```

Se imprimirá a la pantalla lo que se escribió en el teclado al presionar la tecla Enter.

Con los elementos tratados hasta aquí ya podemos escribir nuestros primeros Scripts

Script 1

```
#!/bin/bash  
echo Hola mundo
```

Cuando se corre este guión se imprimirá a la pantalla Hola mundo

Script 2 -> Lo mismo usando una variable

```
#!/bin/bash  
VARIABLE=Hola mundo  
echo "$VARIABLE"
```

Nótese la variable entre comillas dobles para que imprima todo el texto.

Script 3 -> Cuando se usan mas de una variable

```
#!/bin/bash  
VARIABLE=Hola  
SALUDO=mundo  
echo "$VARIABLE""$SALUDO"
```

En los tres casos se imprimirá a la pantalla Hola mundo

Script 4 -> Si se usan caracteres especiales la cosa puede cambiar

```
#!/bin/bash  
VAR=auto  
echo "Me compré un $VAR" Imprimirá Me compré un auto  
echo 'Me compré un $VAR' Imprimirá Me compré un $VAR  
echo "Me compré un \$VAR" Imprimirá Me compré un $VAR
```

Note como las comillas simples y el carácter \ hacen que Bash ignore la función del carácter especial \$. Siempre las comillas simples harán que se ignore todos los meta caracteres encerrados entre ellas y \ solo el que sigue después.

7. Condicionales.

Los condicionales son claves para "explicarle" a la máquina como debe proceder en una tarea cualquiera, esto se hace casi como si se estuviera explicando una tarea a ejecutar a otra persona.

- **if then fi**

El condicional por excelencia tiene seis palabras claves que son **if**, **elif**, **else**, **then** y **fi**. Donde las palabras tienen un significado comunicativo (en Inglés) casi literal, tal y cual se tratara con otra persona y que Bash por defecto las entienda con ese significado.

- **if -> si** condicional (de si esto o lo otro)
- **elif -> también si** (contracción de else if)
- **else -> De cualquier otra manera**
- **then -> Entonces**
- **fi -> if invertido**, indica que se acabó la condicional abierta con if

Solo son imprescindibles en la estructura del Script **if**, **then** y **fi**. Supongamos ahora que es usted el jefe de una oficina y tiene una secretaria y que por alguna razón le han pedido que envíe una copia de cualquier documento que lo identifique; normalmente le diría a la secretaria algo así:

```
"Maria, por favor, busca en el archivo alguna identificación" (condición a evaluar)
if "si es una copia del pasaporte" (primer resultado de la condición); then (entonces)
"envíala por fax a...." (equivalente al comando a ejecutar)
elif "si es de la licencia de conducción" (segundo resultado de la condición); then
"envíala por correo" (otro comando a ejecutar)
elif "si es del carnet de identidad" (tercer resultado de la condición); then
"envíala con un mensajero " (otro comando diferente)
else "de cualquier otra manera"
"pasa un fax diciendo que la enviaré mañana" (otro comando)
fi
```

Observe que la acción a ejecutar (equivalente al comando) se hace si la condición se evalúa como verdadera de lo contrario se ignora y se pasa a la próxima, si ninguna es verdadera se ejecuta finalmente la acción después del else. La sintaxis de bash se debe tener en cuenta a la hora de escribir el Script o de lo contrario Bash no entenderá lo que usted quiso decirle, Pongamos ejemplos de guiones reales

Script 5

```
#!/bin/bash
VAR1=Pablo
VAR2=Pedro
if [ "$VAR1" = "$VAR2" ]; then
echo Son iguales
else
echo Son diferentes
fi
```

Los corchetes son parte de la sintaxis de Bash y en realidad son un atajo (shortcut) al programa test que es el que ejecuta la acción de comparación. Observe siempre los espacios vacíos entre los elementos que conforman la línea de comandos (excepto entre el último corchete y el ;), recuerde que ese espacio vacío por defecto Bash lo interpreta como final de un elemento y comienzo de otro. Si corre este guión siempre se imprimirá a pantalla Son diferentes, ya que la condición es falsa. Pero si cambia el valor de VAR2=Pablo entonces se imprime Son iguales. Guión 6 Un guión que verifica si existe un directorio y si no existe lo crea e imprime mensajes a pantalla comunicando la acción ejecutada.

```
#!/bin/bash
DIR=~/.fotos (crea como variable el directorio /home/fotos)
if [ ! -d "$DIR" ]; then (verifica si no existe el directorio)
mkdir "$DIR" (si la condición es cierta, no existe el directorio, lo crea)
if [ $? -eq 0 ]; then (verifica si la acción se ejecutó sin errores, de serlo imprime lo que sigue)
echo "$DIR" ha sido creado..."
else (de lo contrario imprime)
echo "Se produce un error al crear "$DIR"
fi (Se cierra la condición abierta en la realización del directorio segundo if)
else ( de lo contrario, relativo al primer if)
echo "Se usará "$DIR" existente"
fi
```

En este guión pueden verse varias cosas nuevas:

1. El carácter **!** niega la acción, si se hubiera escrito `if [-d "$DIR"]` lo que se estaba evaluando era la condición ¿existe el directorio "\$DIR"? pero al colocar **!** se evalúa lo contrario.
2. El carácter **~** significa el /home del usuario.
3. La expresión **-eq** se utiliza cuando quieren compararse valores numéricos, y significa =
4. Se usa una de las variables del sistema **"\$?"** explicada mas arriba.
5. Pueden utilizarse unos condicionales dentro de otros siempre que se cierren apropiadamente.

```
#!/bin/bash
echo "Diga si o no:"
read VAR
if [ "$VAR" = si ]; then
echo "Escribiste -si-"
elif [ "$VAR" = no ]; then
```

```

echo "Escribiste -no-"
elif [ "$VAR" = "" ]; then
echo "No puede dejarlo en blanco"
else
echo "Lo que escribió no se acepta"
fi

```

Observe que se está evaluando varias opciones de la misma condición por lo que lo apropiado es incorporar los respectivos elif dentro de la misma condicional. Un elemento nuevo que se incorpora aquí es la condición " " que quiere decir "la variable está vacía", en este caso, cuando no se escribió nada.

- **case-in esac**

Cuando una variable puede adquirir varios valores o significados diferentes, ya hemos visto como puede usarse la palabra elif para hacer diferentes ejecuciones de comandos dentro de una misma condicional if-then-fi de acuerdo al valor de la variable. Una forma de realizar la misma acción sin escribir tantas líneas de condicionales elif y con ello disminuir el tamaño del guión es la utilización de la sentencia **case-in-esac**. Esta sentencia permite vincular patrones de texto con conjuntos de comandos; cuando la variable de la sentencia coincide con alguno de los patrones, se ejecuta el conjunto de comandos asociados. La sintaxis de la sentencia case-in esac es como sigue

```

case "nombre_variable" in
posibilidad 1) "uno o mas comandos" ;;
posibilidad 2) "uno o mas comandos" ;;
posibilidad n) "uno o mas comandos" ;;
esac

```

Script 8

```

#!/bin/bash
echo "Diga si o no:"
read VAR
case "$VAR" in
si) echo "Escribiste -si-" ;;
no) echo "Escribiste -no-" ;;
*) echo "Lo que escribió no se acepta" ;;
esac

```

Este Script es el mismo que el Script 7 pero utilizando la sentencia case-in-esac Observe que el carácter (*) utilizado en la última opción significa "patrón no contemplado" en este caso.

8. Funciones

Como mecanismo de estructuración en la codificación de Scripts, existe la posibilidad de crear funciones. Su definición exige la definición de un nombre y un cuerpo. El nombre que debe ser representativo , es seguido de apertura y cierre de paréntesis, mientras que el cuerpo se delimita con llaves. La sintaxis es la siguiente.

```

nombre_función ()
{
uno o mas comandos
}

```

Una vez definida la función se utiliza como si de un comando se tratase, invocándolo con el nombre de la función. Hay que hacer una invocación de la función ya definida para que se ejecute el código en su interior y se convierta en operativa. Las funciones son muy útiles cuando segmentos del código del Script son repetitivos, de tal forma solo se escriben una vez y se invocan todas las veces que haga falta, practicando el divino arte de la recursión.

Creando una función simple:

```

ayuda () (se define la función ayuda)
{
echo "Las opciones son si o no, luego apriete Enter"
}

```

Después de creada y activada la función, cada vez que necesitemos la "ayuda" dentro del guión solo colocamos la palabra ayuda como si se tratase de un comando mas y Bash ejecutará el código incluido dentro de la función, es decir imprimirá el texto "Las opciones son si o no, luego apriete Enter". Las funciones pueden ser definidas en cualquier orden, pueden ser tantas como haga falta y pueden contener un paquete relativamente complejo de comandos. Un programador que ha pensado la estructura del Script antes de empezarlo puede y de hecho se hace, crear todas las funciones que necesitará al empezar el guión. Pruebe lo siguiente

Script 9

```

#!/bin/bash
salir () #(Se crea la función salir)
{
exit #(comando)
}
hola() #(Se crea la función Hola)
{
echo Hola #(comando)
}

```

```
hola # (Se invoca la función Hola)
salir # ( Se invoca la función salir)
echo "Esto no se imprime nunca"
```

Verá que el último echo no se imprime ya que primero se invoca la función hola y luego la función salir que cierra el guión (exit). Trate ahora poniendo un comentario (#) a la línea que invoca la función salir (línea 11) y note la diferencia, verá como se imprime el último echo. Observe también como se han comentado aquellas cosas que no son parte integrante del guión pero que se pueden escribir para hacer aclaraciones o anotaciones de interés.

9. Ciclos, lazos o bucles

- **While-do done**

La sentencia **while-do done** se utiliza para ejecutar un grupo de comandos en forma repetida mientras una condición sea verdadera. Su sintaxis es:

```
while
lista de comandos 1
do
lista de comandos 2
```

Mientras la condición de control (lista de comandos1) sea verdadera, se ejecutaran los comandos comprendidos entre **do** y **done** en forma repetida, si la condición da falsa (o encuentra una interrupción explícita dentro del código) el programa sale del bucle (se para) y continua la ejecución por debajo del while. Un ejemplo de la utilidad de este lazo es la posibilidad de poder escoger varias opciones de un menú sin tener que correr el guión para cada opción, es decir se escoge y evalúa una opción y el programa no se cierra, vuelve al menú principal y se puede escoger otra opción, tantas veces como sea necesario. Veamos un ejemplo de como elaborar un menú de opciones.

```
#!/bin/bash
while [ "$OPCION" != 5 ]
do
echo "[1] Listar archivos"
echo "[2] Ver directorio de trabajo"
echo "[3] Crear directorio"
echo "[4] Crear usuario"
echo "[5] Salir"
read -p "Ingrese una opción: " OPCION
case $OPCION in
1) ls;;
2) pwd;;
3) read -p "Nombre del directorio: " DIRECTORIO
mkdir $DIRECTORIO;;
4) if id | grep uid=0
then
read -p "Nombre del usuario: " NOMBREUSUARIO
useradd $NOMBREUSUARIO
else
echo "Se necesitan permisos de root"
fi;;
5) ;;
*) echo "Opción ingresada invalida, intente de nuevo";;
esac
done
exit 0
```

Descripción del Script:

1. En la primera línea condicionamos el lazo a que la opción escogida sea diferente de 5.
2. Luego se hace una lista de echos de las opciones desde 1 hasta 5 con su descripción para que sean imprimidas a la pantalla y así poder escoger alguna.
3. Le sigue el comando read para que lea del teclado la opción escogida (variable OPCION), a read se le ha agregado -p que hace que imprima un mensaje, en este caso imprime Ingrese una opción.
4. Para ahorrar líneas del guión se elabora un case con los comandos que deben ejecutarse en cada caso ls para listar los archivos [1], pwd (present work directory) para ver directorio de trabajo [2], otro read para escribir el nombre del directorio que quiere crear [3] y hacer la variable DIRECTORIO seguido por mkdir que crea el directorio, luego se crea una condicional if-fi para chequear si el usuario tiene permisos de root, necesario para la opción [4] de crear un usuario rechazándolo de lo contrario, después viene la opción [5] vacía que ejecuta el comando exit 0, finalmente se incluye "cualquier otra cosa" con el carácter *

Este guión resulta interesante porque se usan las dos formas de compactar el guión vistas hasta ahora, la sentencia case-in esac y la while-do done. Además empiezan a aparecer incluidos en los comandos algunos de los programas muy usados de Linux al escribir guiones.

- **until-do done**

La sentencia **until-do done** es lo contrario de while-do done es decir el lazo se cierra o para, cuando la condición sea falsa. Si le parece que ambas son muy parecidas está en lo cierto. En ambos casos se pueden elaborar bucles o ciclos infinitos si la condición de control es siempre verdadera o falsa según el caso, veamos Lazos infinitos Bucles infinitos son aquellos donde la

ejecución continua dentro del bucle indefinidamente, veamos como hacer un bucle infinito mediante **while**:

```
while true
do
comando 1
comando 2
comando n
done
```

La condición siempre es verdadera y se ejecutara el bucle indefinidamente, mediante **until** sería así:

```
until false
do
comando 1
comando 2
comando n
done
```

Existe la posibilidad de salir de un bucle, independientemente del estado de la condición, el comando **break** produce el abandono del bucle inmediatamente. Veamos el guión anterior sobre la creación de un menú utilizando un lazo infinito y el comando **break**

```
while true
do
echo "[1] Listar archivos"
echo "[2] Ver directorio de trabajo"
echo "[3] Crear directorio"
echo "[4] Crear usuario"
echo "[5] Salir"
read -p "Ingrese una opción: " OPCION
case $OPCION in
1) ls;;
2) pwd;;
3) read -p "Nombre del directorio: " DIRECTORIO
mkdir $DIRECTORIO;;
4) if id | grep uid=0
then
read -p "Nombre del usuario: " NOMBREUSUARIO
useradd $NOMBREUSUARIO
else
echo "Se necesitan permisos de root"
fi;;
5)
echo "Abandonando el programa..."
break;;
*)
echo "Opción ingresada invalida, intente de nuevo";;
esac
done
Guión 11ne
exit 0
```

- **for-in-done**

Es otro tipo de ciclo o lazo disponible, la diferencia con los anteriores es que no se basa en una condición, sino que ejecuta el bucle una cantidad determinada de veces, su sintaxis es la siguiente:

```
for variable in arg 1 arg 2 .....arg n
do
comando 1
comando 2
comando n
done
```

Ejemplos

```
for LETRA in a b c d e f
do
echo $LETRA
done
```

En este guión el comando echo se ejecutara tantas veces como argumentos se hayan puesto después del in, por lo tanto imprimirá seis líneas cada una con una letra de la a a la f.

```
for ARCHIVO in *
if [-d $ARCHIVO]; then
cd $ARCHIVO
rm *.tmp
cd ..
fi
done
```

Este es un guión entra en todos los subdirectorios del directorio actual de trabajo y borrará todos los archivos .tmp (temporales). En este caso el carácter * se usa en la primera línea con el significado "tantas veces como sea necesario" y en la penúltima línea como "cualquier cosa".

Es frecuente la necesidad de redirigir resultados de la ejecución de un comando a diferentes lugares, que pueden ser los descriptores de ficheros **stdin**, **stdout** y **stderr**, a la entrada de otro comando o a un archivo en el disco duro, esto se llama redirección y es muy útil en la escritura de guiones.

1. Los descriptores de archivos

En Bash al igual que en cualquier otro programa de consola de Linux tenemos tres flujos o descriptores de archivos abiertos por defecto:

- La entrada estándar (**STDIN**)
- La salida estándar (**STDOUT**)
- El error estándar (**STDERR**)

El primero puede ser utilizado para leer de él, y los otros dos para enviar datos hacia ellos. Normalmente **STDIN** viene del teclado de la terminal en uso, y tanto **STDOUT** como **STDERR** van hacia la pantalla. **STDOUT** muestra los datos normales o esperados durante la ejecución, y **STDERR** se utiliza para enviar datos de depuración o errores. Cualquier programa iniciado desde el shell, a menos que se le indique explícitamente, hereda estos tres descriptores de archivo permitiéndole interactuar con el usuario.

- Enviar STDOUT a un archivo

En ocasiones necesitamos enviar la salida estándar a un archivo y no a la pantalla, ya sea porque es muy grande para "manejar a ojo" o porque nos interesa guardarla a disco duro. Para enviar la salida estándar a un archivo usamos `>` con lo que se sobrescribe el archivo si ya existe, o `>>` que solo agrega los datos de salida al final del archivo ya existente.

```
#!/bin/bash
```

```
ls -R /home/mis_fotos > /tmp/indice
```

Creará un archivo llamado `/tmp/indice` donde estará el listado de los archivos bajo `/home/mis_fotos`.

- Tomar STDIN de un archivo

Si queremos que un proceso tome su entrada estándar de un archivo existente usamos `<`

- Enviar STDERR a un archivo

Si queremos enviar la salida de errores a un archivo se procede igual que lo que se mencionaba con respecto a la salida estándar pero se usa `&>` o `&>>` según el caso.

- Enviar STDERR a STDOUT

Para esto se escribe al final de la línea de comandos `2>&1`.

- Enviar STDOUT a STDERR

En este caso se escribe al final de la línea de comandos `1>&2`

2. Entubado

Las tuberías se utilizan para enviar la salida de un comando o proceso a la entrada de otro, esto es con frecuencia necesario para completar una acción iniciada con un comando que debe ser completada con otro. Es simple el modo de operar, solo se coloca el carácter `|` en la línea de comandos entre un programa y otro. Este carácter (`|`) se conoce como tubo (pipe)

```
#!/bin/bash
```

```
file -b "$1" | grep -i "vorbis" >/dev/null 2>&1
```

```
if [ $? -eq 0 ]; then
```

```
oggdec "$1"
```

```
echo "Hecho"
```

```
else
```

```
echo "Archivo no soportado"
```

```
exit
```

```
fi
```

Este guión convierte a wav cualquier archivo de audio ogg. Primero se invoca a `file` para que analice el tipo de archivo correspondiente a la variable `$1` que como ya se sabe es el primer argumento introducido en la línea de comandos (por ejemplo la ruta hasta un archivo). Luego la salida de `file` se entuba al programa `grep` que determina si dentro del archivo aparece la palabra `vorbis` (caso de los archivos de audio ogg). El condicional `if-then-fi` chequea que sea cierto (es decir la palabra `vorbis` si existía, por lo que es un archivo ogg de audio), entonces se decodifica a wav con el comando `oggdec`, de lo contrario se imprime que es un archivo no soportado. Tanto la salida estándar como la de errores se envía a `/dev/null`, un dispositivo que "desaparece" la información suprimiendo la salida por pantalla. Esto es conveniente y saludable en muchas líneas de comandos cuando la salida puede generar gran cantidad de información tanto de salida estándar como de errores y estos no nos interesan. Solo se escribe `>/dev/null 2>&1`.

11. Globales y expansiones.

1. Globales

Estos son aliados cuando uno quiere ahorrarse teclazos y funcionan como "generalizadores" de cosas, los globales mas comunes son:

1. `~` Le dice a Bash que es el directorio home del usuario.
2. `*` Significa "todo lo que puedas incluir ahí" de forma tal que si ponemos el comando `ls ~/*.wav` listará todos los archivos `.wav` que están en el directorio home del usuario. Ahora si escribimos `ls ~/m*` nos listará todos los archivos de home que empiecen con `m`.

3. . Un punto en el entorno de la shell significa "el directorio donde estamos trabajando"

Ejemplo:

```
~$ #!/bin/bash
DIR=.
mkdir "$DIR"
echo "$?"
```

Si escribimos este guión y lo corremos dará un error. Por supuesto, le estamos mandando a hacer el directorio donde estamos. Habrá notado usted que es muy común a la hora de compilar programas desde el binario utilizar ./configure, con esto le estamos diciendo a Bash "corre el archivo configure que está en este mismo directorio".

2. Expansiones.

Las expansiones son mas configurables y trabajan con argumentos mucho mas definidos, está claramente hecha para hacer mas inteligente la shell. Cuando especificamos una lista de valores o argumentos separados por comas entre llaves, Bash la expande convirtiéndola en la cadena expandida con cada uno de los argumentos, por ejemplo:
el comando

```
~$ echo este/directorio/{algo,muy,demasiado}/largo
```

dará como resultado la impresión a pantalla de:

```
~$ este/directorio/algo/largo este/directorio/muy/largo
este/directorio/demasiado/largo
```

Hay que tener en cuenta que:

a) La expansión funciona sobre una sola palabra sin espacios si escribimos:

```
~$ echo esto {es,parece} difícil
escribirá:
esto es parece difícil
```

b) La expansión no se realiza entre comillas simples ni dobles por lo que no sirve para corregir el ejemplo anterior:

```
~$ echo "esto {es,parece} difícil"
dará:
esto {es,parece} difícil
```

c) Lo que debe hacerse es ignorar o escapar los espacios y escribir

```
~$ echo esto\ {es,parece}\ confuso
así obtendremos lo que queríamos:
esto es difícil esto parece confuso.
```

Pueden ponerse múltiples expansiones en una sola línea y se obtendrán todas las combinaciones posibles.

```
~$ echo {una,otra}\ combinación\ { bastante,muy}\ difícil.
Responde
una combinación bastante difícil. otra combinación bastante difícil.
una combinación muy difícil. otra combinación muy difícil
```

12. Aritmética de Bash.

Se pueden ejecutar en Bash las principales acciones aritméticas entre las variables utilizando los signos:

- + suma
- - resta
- * multiplicación
- / división

Las operaciones tienen su sintaxis que debe ser respetada para que Bash lo haga adecuadamente.

- Pruebe esto en la shell o la línea de comandos (consola).

```
~$ echo 1+1
```

La respuesta será 1+1 porque bash lo interpreta como caracteres simples, para que realice la operación de suma hay que escribir:

```
~$ echo $((1+1)) o
```

```
echo ${1+1}
```

Bash no maneja números fraccionarios solo números enteros por lo tanto si usted escribe:

```
~$ echo ${3/4} la respuesta será cero, sin embargo si escribe:
echo ${4/2} la respuesta será correcta 2
```

- También podrá utilizar a expr para las operaciones de la forma siguiente:

```
~$ expr argumento1 signo argumento2
```

pruebe en la consola

```
~$ expr 2+2 la respuesta será 2+2 porque no hemos dejado espacios en
blanco entre signos y argumentos o
```

```
~$ expr 2 + 2 (con espacios en blanco) la respuesta será 4 o
expr 4 / 2 la respuesta será 2
```

Cuando se use es signo * para la multiplicación debe anteponerle una barra invertida para que Bash no lo interprete como un global, sería:

```
~$ expr 10 \* 10 la respuesta será 100
```

El programa expr da sus resultados directamente a la salida estándar pero tampoco maneja números fraccionarios. Hay que observar siempre un espacio entre los argumentos.

- Para operar con fraccionarios debe entubar la expresión al programa bc de la forma siguiente:

```
~$ echo operación | bc -l por ejemplo;
echo 3/4 | bc -l
```

El resultado será 0.75 o

```
~$ echo 2+2.5 | bc -l
```

Devolverá 4.5 En algunas distribuciones el programa bc no se instala por defecto. Hay otras expresiones que Bash interpreta aritméticamente;

-lt Menor que
-le Menor o igual que
-eq Igual que
-ge Mayor o igual que
-gt Mayor que
-ne Distinto que

12. Lógica de Bash.

Para la shell los caracteres que tienen un significado lógico en la comparación o evaluación de archivos son:

> Mayor que
< Menor que
>= Mayor o igual que
<= Menor o igual que
! Diferente que
|| OR (ó)
&& AND (y)

```
~$ #!/bin/bash
ARCHIVO=$1
file -b "$1" | grep -i 'JPEG' || file -b "$1" | grep -i 'GIF' || file
-b "$1" | grep -i 'PNG' || file -b "$1" | grep -i 'BITMAP' >/dev/null
2>&1
if [ $? -eq 0 ]; then
echo "Es una imagen"
else "No es una imagen"
fi
```

En este guión hemos supuesto que un archivo cualquiera se convierte en la variable \$1 y queremos averiguar si el archivo es una imagen en alguno de los formatos mas comunes, primero acudimos a file para que "lea" el texto que contiene el archivo y lo entubamos a grep que buscará patrones de texto de lo que le entrega file. Como necesitamos averiguar si alguno de los patrones JPEG, GIF, PNG o BITMAP aparece dentro del archivo utilizamos varias instancias de file

y grep separadas con OR (|), de esta forma le estamos diciendo en el comando "busca si aparece JPEG o GIF o PNG o BITMAP, si lo encuentras entonces imprime"

"Es una imagen" de cualquier otra forma imprime "No es una imagen"

El tema está sacado de: <http://doc.ubuntu-es.org>

[Tweet](#)

Publicado por [juanetebitel](#) 

Etiquetas: [TERMINAL](#)

30 comentarios :

1. 1
[hector](#)16 oct. 2009 4:04:00

Te felicito, grandes tutoriales que estas realizando.

[Responder](#)[Eliminar](#)
2. 2
[gobi](#)14 dic. 2009 6:40:00

Excelente articulo, lo añadiré a mi colección.

[Responder](#)[Eliminar](#)
3. 3
[Maxi](#)6 nov. 2010 3:31:00

Que buen tuto!! No sabes cuanto me ayudo!!! ;))
thankssssssssss!!!

[Responder](#)[Eliminar](#)
4. 4
[bibalgr](#)29 nov. 2010 15:15:00

Muy buen post Juanetebitel, gracias

[Responder](#)[Eliminar](#)
5. 5
[Anónimo](#)30 nov. 2010 0:38:00

Hola Juanetebitel mi nombre es Kikilovem:
Encantado de saludar a todo el mundo. Debe de ser un gran tutorial pero un newbie como yo no ha comprendido nada, aunque me hubiera gustado. Toda esta guía para actualizar mi ubuntu 10.10, me ha servido extraordinariamente bien, pues solo he tenido que "pegar" y "copiar" comandos y acto seguido pulsar "Enter". Pero lo verdaderamente interesante, y es mi gran frustración, es no saber el uso de la consola. Si das pescado a un hambriento saciarás su apetito, pero lo realmente interesante es enseñarle a pescar.

[Responder](#)[Eliminar](#)
6. 6
[juanetebitel](#)30 nov. 2010 8:04:00

Hola Kikilovem, realmente este tuto no es para un recién llegado a linux y no es necesario para un usuario normal que sólo quiere utilizar el programa y sus aplicaciones. Pero Linux tiene la virtud de que lo podemos modificar a nuestras necesidades y aquí es donde entran los scripts. Se puede decir que la cúspide de la utilización de Linux es aprender a hacer Scripts y compilar el Kernel a nuestro antojo y según nuestras necesidades para así tener un sistema mejor si cabe.

Para entender mejor, deberías primero de leer otros:

<http://ubuntu-guia.blogspot.com/2009/07/navegador-de-archivos-nautilus.html>

<http://ubuntu-guia.blogspot.com/2010/09/activar-desactivar-root-ubuntu.html>

<http://ubuntu-guia.blogspot.com/2009/06/como-instalar-paquetes-y-programas-en.html>

<http://ubuntu-guia.blogspot.com/2009/07/comandos-basicos-de-linux.html>

[Responder](#)[Eliminar](#)
7. 7


bilalgr5 dic. 2010 16:28:00

Hola Juanetebitel que tal estas?? me puedes explicar por favor el uso de set, unset, export, env y printenv y su relacion con PATH, y esto de variables de entorno y variables globales, es que me he liado un montón.

[Responder](#)[Eliminar](#)



8.

8

bilalgr5 dic. 2010 20:19:00

por favor tambien me puedes dar algunos ejemplos del uso del wait, disown, trap, sleep. Te lo agradeceré mucho si me los explicas.

Un saludo

[Responder](#)[Eliminar](#)

9.

9

*Makova*17 abr. 2011 19:21:00

Hola Juan.
Este lo tienes que cambiar;

pruebe en la consola

expr 2+2 la respuesta será 4 o
expr 4 / 2 la respuesta será 2

por este;

expr 2 + 2 la respuesta será 4

Ya se que lo has hecho a drede para que uno mismo lo corrija y se de cuenta y comience a entender "bash".

Eres un máquina, gracias.

Un cordial saludo...

[Responder](#)[Eliminar](#)



10.

10

*juanetebitel*18 abr. 2011 7:40:00

Gracias Makova, pues la verdad es que ha sido un error mío y no ha sido a drede. Ahora mismo lo modifico.

[Responder](#)[Eliminar](#)

11.

11

*Willem Ponce Ramal*29 abr. 2012 17:11:00

Gracias

[Responder](#)[Eliminar](#)

12.

12

*Anónimo*19 ago. 2012 5:39:00

saludos seria bueno que crearas tutoriales de programacion en gambas2 y qt

como usar qt designer y kdevelop, lazarus, gambas , eclipse, etc

jors desde venezuela

[Responder](#)[Eliminar](#)

[Respuestas](#)



1.

12.a

*juanetebitel*20 ago. 2012 9:20:00

Hola Jors, el compañero del Foro "jsbsan" tiene un blog muy completo con foro incluido sobre Gambas en:
<http://jsbsan.blogspot.com.es/>
saludos

[Eliminar](#)

[Responder](#)

13.

13

*Christian*23 oct. 2012 19:55:00

HOLA PERDON BUEN DIA ALGUIEN SABRA UN SCRIPT DE CREACION DE USUARIOS??

MUCHA SGRACIAS



[Responder](#)[Eliminar](#)

14.

14

Anónimo25 nov. 2013 0:59:00

hola sabe alguien hacer control de errores en bash??

[Responder](#)[Eliminar](#)[Respuestas](#)

14.a

1.



juanetebitel25 nov. 2013 19:28:00

Hola, Si comentas como anónimo, por favor, escribe al menos tu Nombre o Nick, para así no tener una conversación tan impersonal, gracias

El más simple y eficaz (para mí):

set -e

Para Scripts más complejos, un derivado del anterior que da una mejor salida:

set -o errexit

Pero hay otras muchas funciones que puedes probar

En [esta página](#) ponen algunas muy interesantes.

saludos

[Eliminar](#)[Responder](#)

15.

15

Anónimo2 dic. 2013 18:22:00

Hola Juan,

Mi nickname es Anakin. Es la primera vez que te doy mi feedback a pesar que tus posts son una continua referencia en mis búsquedas. Te felicito por tu enorme trabajo y la claridad con la que lo logras presentar. Aquí te dejo unos comentarios que pueden mejorar en un 0.000...9% tu post puesto que en algunos casos faltan algunos detalles o son simplemente errores "typewriting":

ejemplo en el bucle "until-do done"

penúltima línea del ejemplo

"doGuión 11ne"

por

"done"

segundo ejemplo del bucle "for-in-done"

insertar "do" después de la primera línea

for ARCHIVO in *

if [-d \$ARCHIVO]; then

por

for ARCHIVO in *

do

if [-d \$ARCHIVO]; then

ejemplo c de "2.expansiones"

echo esto\ {es,parece}\ confuso

por

echo esto\ {es\ diffil,parece\ confuso}

en ejemplo sobre la "lógica de Bash" es innecesaria la asignación de la variable ARCHIVO porque no se utiliza posteriormente, adicionalmente falta un echo para mostrar la respuesta "No es una imagen". En consecuencia se debe reemplazar:

#!/bin/bash

ARCHIVO=\$1

file -b "\$1" | grep -i 'JPEG' || file -b "\$1" | grep -i 'GIF' || file -b "\$1" | grep -i 'PNG' || file -b "\$1" | grep -i 'BMAP' >/dev/null 2>&1

if [\$? -eq 0]; then

echo "Es una imagen"

else "No es una imagen"

fi

por

#!/bin/bash

```
file -b "$1" | grep -i 'JPEG' || file -b "$1" | grep -i 'GIF' || file -b "$1" | grep -i 'PNG' || file -b "$1" | grep -i 'BITMAP' >/dev/null 2>&1
if [ $? -eq 0 ]; then
echo "Es una imagen"
else
echo "No es una imagen"
fi
```

Finalmente, dos errores muy evidentes de typewriting:

```
#
"Exixten dos tipos de variables: "
por
"Existen dos tipos de variables: "

#
"Si hacemos un read sin asignar variable, el dato de almacena en $REPLY una variable
del"
por
"Si hacemos un read sin asignar variable, el dato se almacena en $REPLY una variable
del"

Mil gracias por tu trabajo
```

[Responder](#)[Eliminar](#)

16. 

[Diego Valencia](#)26 feb. 2014 1:52:00

16

Ayuda yo tengo un script de nombre aaa.sh así

```
~/diego/aaa.f
3
4
```

el programa aaa.f me pide dos variables

¿Como puedo ingresar esas dos variables en el programa?

Por favor quien me puede ayudar??????????

[Responder](#)[Eliminar](#)

[Respuestas](#)

1. 

[Darwin Brochero](#)26 feb. 2014 16:49:00

16.a

con el simbolo \$. Mira un corto ejemplo desde una terminal

1. creas un archivo llamado "hi" en tu home
touch ~/hi

2. en él escribes lo siguiente y lo guardas
#!/bin/bash
echo "Hola \$1 \$2"

3. asumiendo que lo tienes en tu home, le das los permisos de ejecución
chmod u+x ~/hi

4. lo ejecutas desde tu home
cd ~
./hi diego valencia

Espero te sirva.

[Eliminar](#)
[Responder](#)

17. 

[Diego Valencia](#)26 feb. 2014 17:47:00

17

Lo que buscaba era solo esto:

```
~/diego/programa <<EOF
dato1
dato2
.
.
.
```

[Responder](#)
[Eliminar](#)


18

18.

[Juan Salome](#) 7 may. 2014 19:16:00

Hola Juan serías tan amable de indicarme como realizar un script para que me devuelva el nombre de usuario, ubicación actual en el sistema de archivos, particiones del disco y version del Kernel?

[Responder](#)
[Eliminar](#)
[Respuestas](#)

1.

18.a

[Anónimo](#) 7 may. 2014 21:16:00

Para saber la versión del Kernel se escribe en la terminal "uname -r" (sin comillas)

Para saber el nombre de usuario: "whoami" (sin comillas)

Las particiones del disco se ven en "Utilidad de discos" (Ubuntu 12.04) o "Discos" (Ubuntu 14.04)

[Eliminar](#)
[Responder](#)

19.



19

[romely herrera](#) 14 may. 2014 4:44:00

Buenas amigos quisiera saber como desfragmentar un juego para saberle solamente la logica al juego de domino espero su respuesta muchas gracias espero que lo tomen en cuenta !!

[Responder](#)
[Eliminar](#)

20.

20

[Anónimo](#) 14 ago. 2014 13:57:00

Problema con scripts en ubuntu 14.04

Hola Juan.

Excelente tu trabajo en esta web.

Acabo de actualizar a Ubuntu 14.04. Algunos programas que uso tienen scripts para su ingreso, como el CMapTools o el GvSIG. Estos crean su propia carpeta en "home" (y por tanto no instalan el script en las carpetas "bin"). En los Ubuntu anteriores se reconocía el archivo y se ejecutaban, pero ahora el 14.04 insiste en abrir como texto (con gedit). Al final estoy abriendo desde el terminal, con "./", pero me parece extraño. ¿Alguna sugerencia?. Gracias de antemano.

[Responder](#)
[Eliminar](#)
[Respuestas](#)

1.

20.a


[juanetebitel](#) 15 ago. 2014 20:22:00

Eso se configura en el menú de Nautilus: "Editar > Preferencias", pestaña "Comportamiento" >, apartado "Archivos de texto ejecutables", seleccionar "Preguntar cada vez". De esta forma, cuando hagamos doble clic sobre cualquier Script o archivo ejecutable nos mostrará tres opciones: Ejecutar en una terminal, Mostrar (abrirlo con el editor de texto) y Ejecutar (similar a Alt+F2)

O lo puedes configurar para que se ejecute directamente, como quieras.

Más información en:

<http://www.ubuntu-guia.com/2014/04/administrador-de-archivos-en-ubuntu.html>

[Eliminar](#)
[Responder](#)

21.

21

[Òscar](#) 19 ago. 2014 17:13:00

Hola;

Hace poco me instalaron el Xubuntu 14.04, antes tenía el windows xp. El cambio es notable, para bien, aunque me pierdo un poco. Necesito hacerte una consulta de la que no encuentro respuesta, cuando quiero reproducir un video me va lento, como si pasara fotograma a fotograma, en cambio el audio va bien. No tengo ni idea de como resolverlo, me puedes ayudar?

Gracias

[Responder](#)
[Eliminar](#)


22

[Luis Guillermo Quevedo Velez](#) 21 nov. 2014 14:46:00

Hola Juan.

Muy buen tutorial. Me ha resultado de gran utilidad.

Tengo una pregunta:

¿ Hay alguna manera de leer sólo la primera tecla pulsada, sin necesidad de esperar hasta que el usuario pulse ?

[Responder](#)[Eliminar](#)



23.

Luis Guillermo Quevedo Velez 21 nov. 2014 14:48:00

23

Flató la tecla de retorno en mi pregunta, la palabra E n t e r no fue tomada por el editor

[Responder](#)[Eliminar](#)



24.

manuel salas 27 jun. 2016 18:13:00

24

señores:

pagina : <http://doc.ubuntu-es.org> , ya no esta linea.

[Responder](#)[Eliminar](#)



25.

★*Rayne*★ *Goth Kestrell* ✨ † 12 ago. 2016 5:06:00

25

dayum son, me a venido perfecto para automatizar muchas cosas en ubuntu 14.04
ggwp

[Responder](#)[Eliminar](#)



Si comentas como anónimo, por favor, escribe al menos tu Nombre o Nick, para así no tener una conversación tan impersonal, gracias

[Entrada más reciente](#) [Entrada antigua](#) [Página principal](#)

Suscribirse a: [Enviar comentarios \(Atom \)](#)

[Buscar en el Blog](#)

[Descarga de Ubuntu](#)





Etiquetas, Apartados, Temas generales

ADMINISTRACION (17)

APARIENCIA (12)

COMPIZ efectos 3D (12)

GRUB arranque (12)

HARDWARE (5)

HERRAMIENTAS (6)

IMAGEN Y FOTOGRAFIA (4)

INSTALACION (23)

INTERNET (21)

SEGURIDAD (5)

SONIDO y VIDEO (15)

TARJETAS GRAFICAS (6)

TARJETAS SONIDO (6)

TERMINAL (7)

TRUCOS (10)

WINDOWS relación (3)



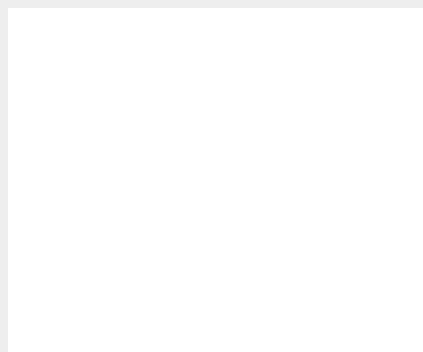
Suscripciones



Feed RSS entradas
Feed RSS comentarios
Feed Email
Facebook
YouTube



Seguidores



Las 10 guías más vistas este mes



Instalar Flash Player en Ubuntu (32 y 64 bits)

Actualizado a 25 de Abril de 2014 Hola a todos, en el Foro y en el blog se han realizado varias preguntas que me hacen ver que hay cierta c...



Instalar paquetes, programas o aplicaciones en Ubuntu

Actualizado el 20 de Abril de 2014. Hola a todos, soy Juanetebitel y vamos a ver cómo Instalar paquetes, programas, aplicaciones en Ubuntu...



Comandos básicos para la terminal de Ubuntu

Hola a todos, soy Juanetebitel y vamos a ver los comandos básicos de Gnu-Linux Ubuntu para su utilización en una Terminal o Consola (Aplica...



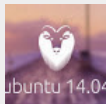
Instalar Oracle Java 7, 8 en Ubuntu 14.04

Actualizado a 28 de Abril de 2014 OpenJDK y el plugin IcedTea son mantenidos por la comunidad y reciben actualizaciones periódicas. Funci...



Cómo recuperar una contraseña olvidada en ubuntu

Podemos encontrar con la situación de que hemos perdido u olvidado la contraseña de superusuario o root que creamos cuando instalamos Ub...



Como instalar ubuntu 14.04

Actualizado el 18 de Abril de 2014 Vamos a ver como instalar ubuntu (versiones 12.04 a 14.04) pasito a pasito y con todo lujo de detalles, i...



Conexión de red en Ubuntu 14.04

Conectarse a internet en Ubuntu es realmente fácil, ya que el sistema detecta el hardware, instala el controlador y realiza la conexión aut...



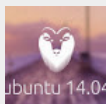
La cuenta del superusuario o root en Ubuntu

Actualizado a 20 de Abril de 2014 En esta guía vamos a hablar del root o superusuario, el usuario administrador, como dar privilegios de ro...



Error de repositorios al actualizar Ubuntu

Me habéis preguntado muchos sobre que al actualizar os da error en los repositorios. Aquí unos ej: "Imposible obtener http://...repos...



Cosas que hacer despues de instalar ubuntu 14.04

Ubuntu 14.04 Trusty LTS es una versión de larga duración, que tendrá soporte durante 5 años.

Ubuntu trae por defecto todo lo necesario par...



Archivo de Guías por Fecha

- ▶ 2016 (1)
 - ▶ febrero (1)
- ▶ 2014 (10)
 - ▶ mayo (1)
 - ▶ abril (7)
 - ▶ marzo (2)
- ▶ 2013 (9)
 - ▶ octubre (2)
 - ▶ septiembre (1)
 - ▶ agosto (1)
 - ▶ julio (1)
 - ▶ abril (1)
 - ▶ marzo (3)
- ▶ 2012 (24)
 - ▶ octubre (2)
 - ▶ agosto (1)
 - ▶ julio (3)
 - ▶ junio (3)
 - ▶ mayo (6)
 - ▶ abril (7)
 - ▶ marzo (1)
 - ▶ febrero (1)
- ▶ 2011 (26)
 - ▶ noviembre (1)
 - ▶ octubre (7)
 - ▶ agosto (1)
 - ▶ julio (2)
 - ▶ mayo (3)
 - ▶ abril (5)
 - ▶ marzo (2)
 - ▶ febrero (1)
 - ▶ enero (4)
- ▶ 2010 (74)
 - ▶ diciembre (4)
 - ▶ noviembre (5)
 - ▶ octubre (10)
 - ▶ septiembre (6)
 - ▶ agosto (8)
 - ▶ julio (4)
 - ▶ junio (12)
 - ▶ mayo (13)
 - ▶ abril (3)
 - ▶ marzo (2)
 - ▶ febrero (2)
 - ▶ enero (5)
- ▼ 2009 (46)
 - ▶ diciembre (2)
 - ▶ noviembre (3)
 - ▶ octubre (4)
 - ▼ septiembre (4)

Gestionar usuarios y grupos en Ubuntu

Como hacer un Script en Ubuntu

Instalar Jdownloader en Ubuntu

Los efectos más comunes de Compiz en Ubuntu


- ▶ agosto (11)
- ▶ julio (19)
- ▶ junio (3)




Comentarios recientes

 **Lucho**


En ubuntu 16.04 ya no es necesario compilar nada porque el paquete "opensc" (versión 0.15) del repositorio por defecto ya viene con el driver del DNI-e. Buena parte de la información la he en...

 **curiosidades**


hola soy nuevo en linux, quiero instalar genymotion me da error no abre. ya eh observado varios video en youtube, pero ninguna me funcione. si alguien me ayuda se lo agrade seria muchísimo por favo...

 **Pedro Gregorio Barrientos Chinchay**

Hola Juan eres un referente, para todos quienes empezamos en el software libre y me alegra saber q te enamoraste de UBUNTU, al igual q yo, fue un amor a primera vista, apesar que ya habia probado fedora, L...

 **Maurice Antonio Avila V.**

muchisimas gracias que seria de la comunidad sin personas como ustedes que comparten su conocimiento funciona perfecto! recupere ubuntu 14.4 solo debi cambiar el nombre de la version por trusty! graci...

 **techosverdesmt**

yo tambien tengo ese problema pero a pesar de que estoy en la carpeta que descomprimi me sale ese error no hay alguna otra solucion?



Jose H Garcia

tengo este problema que puedo hacer: Not using locking for read only lock file

/var/lib/dpkg/lock E: Unable to write to /var/cache/apt/E The package lists or status file could not be parsed or opened...



Wilkerman Lopez

hola jeaneteitel me puedes alluadar yo tengo ubuntu 14.04 y hace dos días la prendi y no me salía el nada solo el fondo de pantalla es decir no me salen ningún icono ni la barra de arriba solo logro...



★Rayne★ Goth Kestrell

dayum son, me a venido perfecto para automatizar muchas cosas en ubuntu 14.04

ggwp



Unknown

consulta como puede dejar la pantalla activa en ubuntu? gracias



Malagueños Originales y Libres

Hola: En Ubuntu 16.04 no se puede aumentar el número de áreas de trabajo, ¿te ha pasado con esta versión? ¿saber si se puede forzar desde consola para que haya más de una? Las áreas de trabajo están act...



Webs amigas

<http://www.makova.org/>

<http://linuxveredas.blogspot.com/>

<http://trastetes.blogspot.com.es/>

<http://xfceblog.blogspot.com.es/>

Transformación, customización de motos

<http://doc.ubuntu-es.org/>

<http://www.ubuntu-es.org/>

<http://www.youtube.com/user/pedrote2222>

<http://jsbsan.blogspot.com/>

<http://linuxdragon.wordpress.com/>

<http://www.hatteras.wordpress.com/>

<http://pcteknic.es/>

<http://diversistemas.com/>



pros:



Las 4 libertades del Software libre:

- > La libertad de ejecutar el programa, para cualquier propósito.
- > La libertad de estudiar como trabaja el programa, y cambiarlo para que haga lo que usted quiera.
- > La libertad de redistribuir copias para que pueda ayudar al prójimo.
- > La libertad de distribuir copias de sus versiones modificadas a terceros.



Condiciones de uso, Política de privacidad y Licencia de contenido

El contenido de esta obra está bajo una [licencia de Creative Commons Reconocimiento-NoComercial-CompartirIgual 3.0 Unported](#):

Desarrollo de aplicaciones Web: [Juaneteitel-Web](#). Con la tecnología de [Blogger](#).

#cookie