

Ejemplos de script de shell: Un complemento al capítulo sobre programación en shell.

Administración de linux

Una guía básica

Pedro Pablo Fábrega Martínez

http://dns.bdat.net/documentos/ejemplos_scripts/

[Aviso Legal](#)

Distribución bajo licencia [Creative Commons](#)

En resumen, se permite la reproducción parcial o total de estos textos en cualquier medio, impreso o electrónico, siempre que no se impongan condiciones adicionales a la reproducción y distribución de las copias o de los trabajos derivados que incorporen este documento. En resumen, cualquier trabajo derivado de ese texto debe mantener esta nota de copyright.

Tabla de contenidos

Introducción

1. Parámetros posicionales y variables internas

1.1 Script que muestra los tres primeros parámetros pasado en la línea de órdenes

1.2 Script que muestra el número de parámetros pasados

1.3 Script que escribe todos los parámetros pasados en una línea

1.4 Script que escribe todos los parámetros pasados en una línea y separados por el signo +

2. condiciones (test)

2.1 Script que comprueba que hay dos parámetros posicionales

2.2 Script que comprueba si los dos parámetros son iguales

2.3 Script que escribe «IGUALES» si los dos parámetros posicionales pasado son iguales y «DISTINTOS» en otro caso

2.3.1 Analizamos el programa:

3. Ficheros y directorios (1)

3.1 Script que dice «Es un fichero» si el argumento pasado es un fichero regular. En otro caso debe escribir «No es un fichero».

3.2 Script que dice «Es un directorio» si el argumento pasado es un directorio. En otro caso debe escribir «No es un directorio».

3.3 Script que dice «Es un enlace» si el argumento pasado es un enlace simbólico. En otro caso debe escribir «No es un enlace simbólico».

3.4 Script que escribe las características del fichero o directorio, pasado como argumento: tipo y permisos.

3.5 Script que escribe el tipo de fichero pasado como argumento.

3.6 Script que muestre el contenido de su argumento, si es un directorio muestre

los ficheros que contiene y si es un fichero su contenido.

4. Bucles

4.1 Script que escribe los cinco primeros números

4.2 Script que suma todos los números pasados como argumentos.

4.3 Script que suma todos los números pasados como argumentos y escriba la operación y el resultado: p.e. $1 + 2 + 3 = 6$

4.4 Script que escribe el tipo de todos los ficheros pasados como argumento.

4.5 Script que renombra todos los ficheros cuyo nombre sea de la forma texto.html y en otro que sea de la forma texto.xhtml.

5. Lectura de ficheros

5.1 Script que muestra las línea del fichero /etc/passwd numeradas.

5.2 Script que saca una lista de todos los usuarios que no tienen un directorio con

su nombre en /home.

5.2.1 Caso general

5.2.2 Corolario

5.3 Script que suma las cifras de un fichero

6. Edición de ficheros y expresiones regulares

6.1 Hacer un guion que separe las frases del fichero pasado como argumento y las guarde en otro fichero nuevo.

6.2 Guion que sustituye la palabra "´" por "á" en los ficheros pasados como argumento.

6.3 Guion que transforma un fichero cuyas línea tengan el formato nombre:apellidos:edad en el formato INSERT INTO TABLA VALUES ("nombre",apellidos",edad);

7. Prácticas con comandos Unix

7.1 Guion que diga si el usuario cuyo nombre pasamos como argumento está o no conectado.

7.2 Guion que diga qué usuarios de la lista que pasamos como argumento están o

no conectados.

8. Ejercicios propuestos

Introducción

Todos los scripts están diseñados para ejecutarse en el shell bash, pero tampoco debe suponer demasiada dificultad utilizarlos en otra shell de tipo Bourne.

En estos ejemplos además, de una forma práctica, vamos a recordar, repasar e introducir ciertos conceptos teóricos. De todas formas, para aprovechar estos ejemplos y ganar tiempo y conocimientos, deberías leer el capítulo de programación en shell.

Ni que decir tiene que es necesario conocer todas las órdenes de Unix que vamos a utilizar en los ejemplos propuestos.

He intentado secuenciar los script por temas y por dificultad aunque esto no es siempre posible.

1. Parámetros posicionales y variables internas

1.1 Script que muestra los tres primeros parámetros pasado en la línea de órdenes

Los parámetros que se pasan a un script los podemos encontrar en las variables \$1, \$2, ... También resulta útil \$0 que contiene el nombre del script. En este caso bastaría:

```
echo $1 $2 $3
```

1.2 Script que muestra el número de parámetros pasados

Con este script quiero que observes que hay ciertas variables que proporcionan información en un script. Por ejemplo, una variable que resulta muy útil es \$#, que contiene el número de argumentos. En este caso nuestro script quedaría como:

```
echo $#
```

1.3 Script que escribe todos los parámetros pasados en una línea

Con este script quiero repasar las variables \$(EN) y \$* que contienen la lista de argumentos, con una pequeña diferencia: \$(EN) separa los distintos argumentos por un espacio mientras que \$* los separa según el contenido de la variable IFS (Internal Field Separator), que en principio también es un espacio.

Entonces nuestro script quedaría como:

```
echo $*
```

ó

```
echo $(EN)
```

Estas variables también serán de gran utilidad.

1.4 Script que escribe todos los parámetros pasados en una línea y separados por el signo +

Si vemos el ejemplo anterior podemos deducir que si ponemos IFS=+ ahora el separador de campos será el que nos interesa:

```
IFS=+  
echo $*
```

y escribirá todos los parámetros separados por +

2. condiciones (test)

2.1 Script que comprueba que hay dos parámetros posicionales

En este script tenemos que tomar una decisión dependiendo del contenido de la variable \$# como vimos en un ejemplo anterior.

```
if [ $# -eq 2 ]  
then  
    echo "Hay dos parámetros"  
fi
```

if [\$# -eq 2] Comprobamos si el número de argumentos es dos con una comparación numérica.

Todo lo que pongamos tras "then" y hasta el (else) "fi" se ejecutará cuando la condición sea cierta.

Este ejemplo resulta particularmente útil para construir scripts que deban tener un número exacto de parámetros.

2.2 Script que comprueba si los dos parámetros son iguales

```
if [ $1 == $2 ]  
then  
    echo "Los parámetros son iguales"  
fi
```

Destacamos aquí:

if [\$1 == \$2] Realizamos una comparación de cadenas de caracteres.

2.3 Script que escribe «IGUALES» si los dos parámetros posicionales pasado son iguales y «DISTINTOS» en otro caso

Ahora empezamos con las comparaciones para poder tomar una decisión dentro del script. Según el enunciado vamos a suponer que el script tiene sólo dos argumentos; después vamos a comparar los argumentos y mostrar lo que se pide:

```
if [ $# -eq 2 ]
then
    echo "Uso: $0 arg1 arg2"
    exit
fi
if [ $1 == $2 ]
then
    echo "IGUALES"
else
    echo "DISTINTOS"
fi
```

2.3.1 Analizamos el programa:

En primer lugar, en general vemos que un if puede tener o no un "else" asociado, como en cualquier lenguaje.

Y con más detalle, vemos las línea más significativas:

echo "Uso: \$0 arg1 arg2" Mostramos un mensaje sobre como se usa el script. Observamos que \$0 es el nombre del script.

exit Da por terminado el script, puesto que al no tener los argumentos necesarios no debería continuar su ejecución.

3. Ficheros y directorios (1)

Nota: Todos los script que requieran un número exacto de parámetros deben realizar la comprobación del número de parámetros

3.1 Script que dice «Es un fichero» si el argumento pasado es un fichero regular. En otro caso debe escribir «No es un fichero».

El objetivo de este script es hacer prácticas con las condiciones que proporciona bash para realizar comprobaciones del sistema de ficheros, comprobar ficheros, directorios, permisos, etc.

En primer lugar deberemos verificar el número de argumentos como veíamos en el ejemplo anterior y posteriormente comprobamos si el argumento se corresponde con un fichero:

```
if [ $# -eq 1 ]
then
    echo "Uso: $0 fichero"
    exit
```

```

fi
if [ -f $1 ]
then
    echo "$1 es un fichero"
else
    echo "$1 NO es un fichero"
fi

```

La opción "-f" de la orden test o "[]" devuelve "Verdadero" si el valor que indicamos se corresponde con un fichero regular. Aconsejo un repaso a las distintas condiciones para comprobar otros tipos de ficheros como pipes, dispositivos o enlaces.

Si tenemos ficheros con nombres que contengan espacios y otro metacaracter, el argumento debería estar entre comillas dobles:

```

if [ $# -eq 1 ]
then
    echo "Uso: $0 fichero"
    exit
fi
if [ -f "$1" ]
then
    echo "$1 es un fichero"
else
    echo "$1 NO es un fichero"
fi

```

3.2 Script que dice «Es un directorio» si el argumento pasado es un directorio. En otro caso debe escribir «No es un directorio».

Este script es idéntico al anterior salvo que tenemos que modificar la condición para comprobar directorios:

```

if [ $# -eq 1 ]
then
    echo "Uso: $0 fichero"
    exit
fi
if [ -d $1 ]
then
    echo "$1 es un directorio"
else
    echo "$1 NO es un directorio"
fi

```

3.3 Script que dice «Es un enlace» si el argumento pasado es un enlace simbólico. En otro caso debe escribir «No es un enlace simbólico».

Este ejemplo también es idéntico a los anteriores cambiando la condición:

```

if [ $# -eq 1 ]
then
    echo "Uso: $0 fichero"
    exit
fi
if [ -h $1 ]
then
    echo "$1 es un enlace simbólico"
else
    echo "$1 NO es un enlace simbólico"
fi

```

3.4 Script que escribe las características del fichero o directorio, pasado como argumento: tipo y permisos.

En este ejemplo vamos a ver un resumen de distintas condiciones que podemos consultar al sistema de ficheros. Además, vamos a ver como podemos concatenar cadenas de caracteres.

```

if [ $# -eq 1 ]
then
    echo "Uso: $0 fichero"
    exit
fi
FICHERO="$0: "
if [ -f $1 ]
then
    FICHERO="$FICHERO fichero"
else
    if [ -d $1 ]
    then
        FICHERO="$FICHERO directorio"
    else
        echo "$FICHERO otro tipo"
    fi
fi
if [ -r $1 ]
then
    FICHERO="$FICHERO lectura"
fi
if [ -w $1 ]
then
    FICHERO="$FICHERO escritura"
fi
if [ -x $1 ]
then
    FICHERO="$FICHERO ejecución"
fi

```

Como novedad de este script destacamos un "if" anidado, un "if" dentro de otro.

3.5 Script que escribe el tipo de fichero pasado como argumento.

En este ejemplo vemos otras condiciones posibles que podemos comprobar:

```
if [ $# -eq 1 ]
then
    echo "Uso: $0 fichero"
    exit
fi
if [ -f $1 ]
then
    echo "$1 Fichero regular"
fi
```

```
if [ -d $1 ]
then
    echo "$1 directorio"
fi

if [ -c $1 ]
then
    echo "$1 dispositivo carácter"
fi

if [ -b $1 ]
then
    echo "$1 dispositivo bloque"
fi

if [ -p $1 ]
then
    echo "$1 tubería con nombre"
fi
```

3.6 Script que muestre el contenido de su argumento, si es un directorio muestre los ficheros que contiene y si es un fichero su contenido.

```
if [ -f $1 ]
then
    cat $1
fi
if [ -d $1 ]
then
    ls $1
fi
```

4 Bucles

Hasta ahora todos los script se ejecutaban de una sola pasada, ahora vamos a ver como podemos repetir un bloque de instrucciones un determinado número de veces.

4.1 Script que escribe los cinco primeros números

Para hacer este script necesitamos un bucle que se repita cinco veces. Bueno, podíamos haber puesto:

```
echo "1 2 3 4 5"
```

pero no se trata de eso, sino de comprender los bucles en shell.

En primer lugar vamos a hacer este ejemplo con un bucle "for" que va tomando sucesivamente los valores que indicamos:

```
for N in 1 2 3 4 5
do
    echo $N
done
```

Para hacer este script con un bucle while tendremos que utilizar un contador para saber cuantas veces lo hemos repetido.

```
CONTADOR=0
while [ CONTADOR -le 5 ]
do
    let CONTADOR=CONTADOR+1
    echo $CONTADOR
done
```

Cuando estemos diseñando un script el sentido común nos debe orientar sobre qué tipo de bucle debemos elegir.

4.2 Script que suma todos los números pasados como argumentos.

Ahora vamos a utilizar la variable \$(EN) que contiene todos los parámetros posicionales para ir recorriéndolos de uno en uno e ir sumándolos sobre un acumulador. Con la orden "for" podemos ir recorriendo uno a uno todos los argumentos:

```
ACUMULADOR=0
for N in $(EN)
do
    let ACUMULADOR=ACUMULADOR+N
done
echo $ACUMULADOR
```

4.3 Script que suma todos los números pasados como argumentos y escriba la operación y el resultado: p.e. 1 + 2 + 3 = 6

Ante todo hay que tener claro el ejemplo anterior. Para este ejemplo también vamos a usar la variable \$* y el separador de campos, que en nuestro caso nos interesa que sea "+" para que lo incluya entre cada uno de los argumentos simulando la operación:

```

ACUMULADOR=0
for N in $(EN)
do
    let ACUMULADOR=ACUMULADOR+N
done
IFS=+
echo "$*=$ACUMULADOR"

```

4.4 Script que escribe el tipo de todos los ficheros pasados como argumento.

En este ejercicio lo que pretendemos es ver como podemos procesar múltiples argumentos de idéntica forma, es decir, nosotros al ejecutar el script lo que hacemos es poner una lista de ficheros y para cada uno de ellos realizar las mismas comprobaciones. Por lo demás las comprobaciones serían idénticas a las que vimos en un ejercicio anterior por lo que, para no extendernos innecesariamente, vamos a poner sólo si es un fichero regular o directorio:

```

for F IN $(EN)
do
    if [ -f $F ]
    then
        echo "es un fichero"
    fi
    if [ -d $F ]
    then
        echo "es un directorio"
    fi
done

```

4.5 Script que renombra todos los ficheros cuyo nombre sea de la forma texto.html y en otro que sea de la forma texto.xhtml.

Este script sólo cambia el nombre a los ficheros del directorio activo

```

for fichero in *.html
do
    nombre=$(basename $fichero .html)
    mv $fichero $nombre.xhtml
done

```

Para cambiarlo en toda la rama de directorios:

```

for fichero in $(find . -name "*.html" -type f)
do
    dir=$(dirname)
    nombre=$(basename $fichero .html)
    mv $fichero $dir/$nombre.xhtml
done

```

Una forma de procesar todos los ficheros de una rama del árbol de directorios es

'for fichero in \$(find . -name "*.html" -type f)', donde "fichero" va tomando la ruta relativa de cada uno de los ficheros encontrados.

Este ejemplo puede servir como base para un script que realice ciertas copias de seguridad.

5 Lectura de ficheros

En muchos casos vamos a necesitar leer un fichero línea a línea desde un script, y como el código siempre va a ser muy parecido lo recordamos para los siguientes ejemplos.

Este código lee el fichero /ruta/datos línea a línea y la muestra en pantalla:

```
while read LINEA
do
    echo $LINEA
done < /ruta/datos
```

5.1 Script que muestra las línea del fichero /etc/passwd numeradas.

Realizamos las modificaciones pertinentes al anterior prototipo para añadir un contador que nos muestra la línea numerada:

```
NUM=1
while read LINEA
do
    echo "$NUM $LINEA"
let NUM=NUM+1
done < /etc/passwd
```

5.2 Script que saca una lista de todos los usuarios que no tienen un directorio con su nombre en /home.

```
while read LINEA
do
    $usuario=$(echo $LINEA|cut -f1 -d:)
    if ! [ -f /home/$usuario ]
    then
        echo $usuario
    fi
done </etc/passwd
```

Comentarios y recordatorio:

el operador \$() de bash ejecuta una orden y devuelve su salida estándar. También se pueden usar comillas invertidas (`) para ejecutar una orden y obtener el resultado.

Cada línea la leemos del fichero /etc/passwd

cut -f1 -d: cortaría el campo uno (-f 1) obtenido por un separador : de los valores

de la entrada estándar, que es el resultado. Como el primer campo del fichero /etc/passwd es el nombre de usuario, entonces estaríamos obteniendo este nombre.

5.2.1 Caso general

La base de datos de usuarios no tiene por qué estar completa en el fichero /etc/passwd, puede estar sobre otros soportes como NIS, LDAP, Winbind, db, ... En consecuencia la forma de obtener la lista real de usuarios es ejecutar la orden "getent passwd", así como para obtener la lista de grupos ejecutamos "getent group".

En este caso, nuestro anterior script podría quedar como:

```
for LINEA in $(getent passwd)
do
    $usuario=$(echo $LINEA|cut -f1 -d:)
    if ! [ -f /home/usuario ]
    then
        echo $usuario
    fi
done
```

5.2.2 Corolario

Esta forma de obtener una parte de una cadena para almacenarla en una variable va a ser de uso frecuente en el desarrollo de scripts.

5.3 Script que suma las cifras de un fichero

Suponemos que tenemos un fichero llamado cantidades en el que cada línea está formada por cantidades separadas por un espacio y tenemos que hacer un guion que sume cada línea.

```
while read LINEA
do
    TL=0
    for CIFRA in $LINEA
    do
        let TL=TL+CIFRA
    done
    echo $TL
done < fichero_cifras.txt
```

Comentarios

Cada línea será de la forma "11 22 33 44" lo que permite utilizar un for para ir procesando individualmente cada uno de los componentes. Muy parecido a ejercicios anteriores donde utilizábamos \$(EN).

Si quisiéramos calcular también el total del fichero haríamos;

```
TT=0
while read LINEA
do
```

```
TL=0
for CIFRA in $LINEA
do
    let TL=TL+CIFRA
done
let TT=TT+TL
echo $TL
done < fichero_cifras.txt
echo $TT
```

6 Edición de ficheros y expresiones regulares

6.1 Hacer un guion que separe las frases del fichero pasado como argumento y las guarde en otro fichero nuevo.

Este script es tan simple que se puede hacer en una sola línea.

En realidad separar las frases consiste en añadir un retorno de carro tras cada fin de frase, que es un punto. Evidentemente esto lo vamos a hacer utilizando expresiones regulares.

```
perl -p -e "s/\./\n/g" $1 >$1.separado.txt
```

Comentarios:

El primer punto corresponde la a expresión regular y por tanto necesitamos protegerlo para que no se interprete como un carácter cualquiera, que es su significado como expresión regular. El segundo punto no es necesario protegerlo porque no forma parte de la expresión regular, forma parte del texto sustitutivo, un punto y un retorno de carro.

6.2 Guion que sustituye la palabra "´" por "á" en los ficheros pasados como argumento.

```
for F in $(EN)
do
perl -p -e "s/&acute;/á/g" $F >$F.mod
mv $F.mod $F
done
```

Comentarios

No podemos redirigir al mismo fichero que estamos editando, tenemos que utilizar un fichero intermedio.

6.3 Guion que transforma un fichero cuyas línea tengan el formato nombre:apellidos:edad en el formato INSERT INTO TABLA VALUES ("nombre",apellidos",edad);

```
while read LINEA
do
  NOMBRE=$(echo $LINEA|cut -f1 -d:)
  APELLIDOS=$(echo $LINEA|cut -f2 -d:)
  EDAD=$(echo $LINEA|cut -f3 -d:)
  echo "INSERT INTO tabla VALUES ('$NOMBRE','$APELLIDOS','$EDAD');">fichero.sql
done<fichero.dat
```

7 Prácticas con comandos Unix

7.1 Guion que diga si el usuario cuyo nombre pasamos como argumento está o no conectado.

```
if who|grep $1 >/dev/null 2>&1
then
  echo $1 conectado
else
  echo $1 NO conectado
fi
```

Recordatorio:

Podemos usar la ejecución de una orden como una condición para el if, de forma que si la orden termina correctamente toma el valor verdadero y falso en otro caso. Cuando un orden termina fija la variable \$? con el código de terminación que es la misma que utiliza if para tomar la decisión.

7.2 Guion que diga qué usuarios de la lista que pasamos como argumento están o no conectados.

```
for usuario in $(EN)
do
  if who|grep $usuario >/dev/null 2>&1
  then
    echo $usuario conectado
  else
    echo $usuario NO conectado
  fi
done
```

Este ejercicio es una generalización del anterior. Aquí podemos observar una forma de ampliar un script usando la variable \$(EN) que contiene la lista de argumentos.

Ahora queremos comprobar que el usuario realmente existe:

```
for usuario in $(EN)
do
if getent passwd| grep $usuario >/dev/null 2>&1
if who|grep $usuario >/dev/null 2>&1
then
echo $usuario conectado
else
echo $usuario NO conectado
fi
done
```

Sólo queda por mencionar que hemos redirigido las salida estándar y de errores a /dev/null, primero redirigimos la salida estándar (>/dev/null) y luego redirigimos la salida de errores a la salida estándar (2>&1).

Si la búsqueda sólo nos interesa sobre /etc/passwd la condición quedaría como:

```
if grep ^$usuario /etc/passwd >/dev/null 2>&1
```

8 Ejercicios propuestos

11. Hacer un guion que muestre cuantas veces está conectado un usuario.
14. Hacer un script que muestre sólo los directorios que hay en el directorio actual.
15. Hacer un guion que muestre los nombres de ficheros que contengan algún carácter que no sea ni una letra.
16. Hacer un guion que una el contenido de los dos ficheros pasados como argumento en un fichero nuevo.
18. Modificar el anterior guion para que muestre en pantalla cada frase y el número de caracteres que tiene.
20. Hacer un programa que verifique si los directorios /sbin y /usr/sbin están en la variable PATH
21. Hacer un guion que muestre las línea del fichero pasado como primer argumento las palabras pasadas como segundo y tercer argumentos. El guion debe verificar el número de argumentos.
23. Hacer un script que sustituya la frase «Compañía de las Indias Orientales» por «Sociedad de Pérdida Ilimitadas» en todos los ficheros que haya en /var/www/html y cualquiera de su subdirectorios.
24. Hacer un script que mate todos los procesos cuyo nombre empiece por el texto pasado como argumento.
27. Hacer un script que muestre numerados en una línea distinta todos los argumentos pasados al ejecutarlo