Programación en BASH - COMO de introducción

Mike G (mikkey) disponible en dynamo.com.ar
Traducido por Gabriel Rodríguez Alberich *chewie@asef.us.es*
jueves, 27 de julio de 2000, a las 09:36:18 ART

Este artículo pretende ayudarle a comenzar a programar shell scripts a un nivel básico/intermedio. No pretende ser un documento avanzado (vea el título). NO soy un experto ni un gurú de la programación en shell. Decidí escribir esto porque aprenderé mucho con ello y puede serle útil a otras personas. Cualquier aportación será apreciada, especialmente en forma de parche :)

Contents

1 Introducción

1.1 Obteniendo la última versión

http://www.linuxdoc.org/HOWTO/Bash-Prog-Intro-HOWTO.html

1.2 Requisitos

Le será útil tener una cierta familiaridad con la línea de comandos de GNU/Linux y con los conceptos básicos de la programación. Aunque esto no es una introducción a la programación, explica (o al menos lo intenta) muchos conceptos básicos.

1.3 Usos de este documento

Este documento intenta ser útil en las siguientes situaciones

- Si tiene alguna idea de programación y quiere empezar a programar algunos shell scripts.
- Si tiene una idea vaga de programar en shell y quiere algún tipo de referencia.
- Si quiere ver algunos scripts y comentarios para empezar a escribir los suyos propios.
- Si está migrando desde DOS/Windows (o ya lo ha hecho) y quiere hacer procesos "por lotes".
- Si es un completo novato y lee todo COMO disponible.

2 Scripts muy sencillos

Este COMO tratará de darle algunos consejos sobre la programación de shell scripts, basándose profundamente en ejemplos.

En esta sección encontrará varios scripts pequeños que esperanzadamente le ayudarán a entender algunas técnicas.

2.1 Típico script ‘hola mundo’

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

Este script tiene sólo dos líneas. La primera le indica al sistema qué programa usar para ejecutar el fichero.

La segunda línea es la única acción realizada por este script, que imprime ’Hola Mundo’ en la terminal.

Si le sale algo como *./hello.sh: Comando desconocido.*, probablemente la primera línea, ’#!/bin/bash’, está mal. Ejecute `whereis bash`, o vea ’encontrando el bash’ para saber cómo debe escribir esta línea.

2.2 Un script de copia de seguridad muy simple

```
#!/bin/bash
tar -cZf /var/my-backup.tgz /home/yo/
```

En este script, en vez de imprimir un mensaje en la terminal, creamos un tar-ball del directorio home de un usuario. Esto NO pretende ser un script útil; más tarde se ofrece un script de copia de seguridad más útil.

3 Todo sobre redirección

3.1 Teoría y referencia rápida

Existen 3 descriptores de ficheros: stdin, stdout y stderr (std=estándar).

Básicamente, usted puede:

1. redirigir stdout a un fichero
2. redirigir stderr a un fichero
3. redirigir stdout a stderr
4. redirigir stderr a stdout
5. redirigir stderr y stdout a un fichero
6. redirigir stderr y stdout a stdout
7. redirigir stderr y stdout a stderr

El número 1 ’representa’ a stdout, y 2 a stderr.

Una pequeña nota para ver todo esto: con el comando less puede visualizar stdout (que permanecerá en el búfer) y stderr, que se imprimirá en la pantalla, pero será borrado si intenta leer el búfer.

3.2 Ejemplo: stdout a un fichero

Esto hará que la salida de un programa se escriba en un fichero.

```
ls -l > ls-l.txt
```

En este caso, se creará un fichero llamado 'ls-l.txt' que contendrá lo que se vería en la pantalla si escribiese el comando 'ls -l' y lo ejecutase.

3.3 Ejemplo: stderr a un fichero

Esto hará que la salida stderr de un programa se escriba en un fichero.

```
grep da * 2> errores-de-grep.txt
```

En este caso, se creará un fichero llamado 'errores-de-grep.txt' que contendrá la parte stderr de la salida que daría el comando 'grep da *'.

3.4 Ejemplo: stdout a stderr

Esto hará que la salida stdout de un programa se escriba en el mismo descriptor de fichero que stderr.

```
grep da * 1>&2
```

En este caso, la parte stdout del comando se envía a stderr; puede observar eso de varias maneras.

3.5 Ejemplo: stderr a stdout

Esto hará que la salida stderr de un programa se escriba en el mismo descriptor de fichero que stdout.

```
grep * 2>&1
```

En este caso, la parte stderr del comando se envía a stdout. Si hace una tubería con less, verá que las líneas que normalmente 'desaparecen' (al ser escritas en stderr), ahora permanecen (porque están en el stdout).

3.6 Ejemplo: stderr y stdout a un fichero

Esto colocará toda la salida de un programa en un fichero. A veces, esto es conveniente en las entradas del cron, si quiere que un comando se ejecute en absoluto silencio.

```
rm -f $(find / -name core) &> /dev/null
```

Esto (pensando en la entrada del cron) eliminará todo archivo llamado 'core' en cualquier directorio. Tenga en cuenta que tiene que estar muy seguro de lo que hace un comando si le va a eliminar la salida.

4 Tuberías

Esta sección explica de una manera muy sencilla y práctica cómo utilizar tuberías, y por qué querría utilizarlas.

4.1 Qué son y por qué querrá utilizarlas

Las tuberías le permiten utilizar (muy sencillo, insisto) la salida de un programa como la entrada de otro.

4.2 Ejemplo: una tubería sencilla con sed

Ésta es una manera muy sencilla de utilizar tuberías.

```
ls -l | sed -e "s/[aeio]/u/g"
```

En este caso, ocurre lo siguiente: primero se ejecuta el comando ls -l, y luego su salida, en vez de imprimirse en la pantalla, se envía (entuba) al programa sed, que imprime su salida correspondiente.

4.3 Ejemplo: una alternativa a ls -l *.txt

Probablemente ésta es una manera más difícil de hacer un ls -l *.txt, pero se muestra para ilustrar el funcionamiento de las tuberías, no para resolver ese dilema.

```
ls -l | grep "\.txt$"
```

En este caso, la salida del programa ls -l se envía al programa grep, que imprimirá las líneas que concuerden con la regex (expresión regular) "\.txt$".

5 Variables

Puede usar variables como en cualquier otro lenguaje de programación. No existen tipos de datos. Una variable de bash puede contener un número, un caracter o una cadena de caracteres.

No necesita declarar una variable. Se creará sólo con asignarle un valor a su referencia.

5.1 Ejemplo: ¡Hola Mundo! utilizando variables

```
#!/bin/bash
CAD="¡Hola Mundo!"
echo $CAD
```

La segunda línea crea una variable llamada STR y le asigna la cadena "¡Hola Mundo!". Luego se recupera el VALOR de esta variable poniéndole un '$' al principio. Por favor, tenga en cuenta (¡inténtelo!) que si no usa el signo '$', la salida del programa será diferente, y probablemente no sea lo que usted quería.

5.2 Ejemplo: Un script de copia de seguridad muy simple (algo mejor)

```
#!/bin/bash
OF=/var/mi-backup-$(date +%Y%m%d).tgz
tar -cZf $OF /home/yo/
```

Este script introduce algo nuevo. Antes que nada, debería familiarizarse con la creación y asignación de variable de la línea 2. Fíjese en la expresión '$(date +%Y%m%d)'. Si ejecuta el script se dará cuenta de que ejecuta el comando que hay dentro de los paréntesis, capturando su salida.

Tenga en cuenta que en este script, el fichero de salida será distinto cada día, debido al formato pasado al comando date (+%Y%m%d). Puede cambiar esto especificando un formato diferente.

Algunos ejemplos más:

echo ls

echo $(ls)

5.3 Variables locales

Las variables locales pueden crearse utilizando la palabra clave *local*.

```
#!/bin/bash
HOLA=Hola
function hola {
        local HOLA=Mundo
        echo $HOLA
}
echo $HOLA
hola
echo $HOLA
```

Este ejemplo debería bastar para mostrarle el uso de una variable local.

6 Estructuras Condicionales

Las estructuras condicionales le permiten decidir si se realiza una acción o no; esta decisión se toma evaluando una expresión.

6.1 Pura teoría

Los condicionales tienen muchas formas. La más básica es: **if** *expresión* **then** *sentencia* donde 'sentencia' sólo se ejecuta si 'expresión' se evalúa como verdadera. '2<1' es una expresión que se evalúa falsa, mientras que '2>1' se evalúa verdadera.

Los condicionales tienen otras formas, como: **if** *expresión* **then** *sentencia1* **else** *sentencia2*. Aquí 'sentencia1' se ejecuta si 'expresión' es verdadera. De otra manera se ejecuta 'sentencia2'.

Otra forma más de condicional es: **if** *expresión1* **then** *sentencia1* **else if** *expresión2* **then** *sentencia2* **else** *sentencia3*. En esta forma sólo se añade "ELSE IF 'expresión2' THEN 'sentencia2'", que hace que sentencia2 se ejecute si expresión2 se evalúa verdadera. El resto es como puede imaginarse (véanse las formas anteriores).

Unas palabras sobre la sintaxis:

La base de las construcciones 'if' es ésta:

if [expresión];

then

código si 'expresión' es verdadera.

fi

6.2 Ejemplo: Ejemplo básico de condicional if .. then

```
#!/bin/bash
if [ "petete" = "petete" ]; then
   echo expresión evaluada como verdadera
fi
```

El código que se ejecutará si la expresión entre corchetes es verdadera se encuentra entre la palabra 'then' y la palabra 'fi', que indica el final del código ejecutado condicionalmente.

6.3 Ejemplo: Ejemplo básico de condicional if .. then ... else

```
#!/bin/bash     if [ "petete" = "petete" ]; then
   echo expresión evaluada como verdadera
else
   echo expresión evaluada como falsa
fi
```

6.4 Ejemplo: Condicionales con variables

```
#!/bin/bash
T1="petete"
T2="peteto"
if [ "$T1" = "$T2" ]; then
    echo expresión evaluada como verdadera
else
    echo expresión evaluada como falsa
fi
```

6.5 Ejemplo: comprobando si existe un fichero

un agradecimiento más a mike

```
#!/bin/bash
FILE=~/.basrc
if [ -f $FILE ]; then
    echo el fichero $FILE existe
else
    echo fichero no encontrado
fi
if [ 'test -f $FILE']
```

7 Los bucles for, while y until

En esta sección se encontrará con los bucles for, while y until.

El bucle **for** es distinto a los de otros lenguajes de programación. Básicamente, le permite iterar sobre una serie de 'palabras' contenidas dentro de una cadena.

El bucle **while** ejecuta un trozo de códico si la expresión de control es verdadera, y sólo se para cuando es falsa (o se encuentra una interrupción explícita dentro del código en ejecución).

El bucle **until** es casi idéntico al bucle loop, excepto en que el código se ejecuta mientras la expresión de control se evalúe como falsa.

Si sospecha que while y until son demasiado parecidos, está en lo cierto.

7.1 Por ejemplo

```
#!/bin/bash
for i in $( ls ); do
    echo item: $i
done
```

En la segunda línea declaramos i como la variable que recibirá los diferentes valores contenidos en $(ls).

La tercera línea podría ser más larga o podría haber más líneas antes del done (4).

'done' (4) indica que el código que ha utilizado el valor de $i ha acabado e $i puede tomar el nuevo valor.

Este script no tiene mucho sentido, pero una manera más útil de usar el bucle for sería hacer que concordasen sólo ciertos ficheros en el ejemplo anterior.

7.2 for tipo-C

Fiesh sugirió añadir esta forma de bucle. Es un bucle for más parecido al for de C/perl...

```
#!/bin/bash
for i in `seq 1 10`;
do
        echo $i
done
```

7.3 Ejemplo de while

```
#!/bin/bash
CONTADOR=0
while [  $CONTADOR -lt 10 ]; do
    echo El contador es $CONTADOR
    let CONTADOR=CONTADOR+1
done
```

Este script 'emula' la conocida (C, Pascal, perl, etc) estructura 'for'.

7.4 Ejemplo de until

```
#!/bin/bash
CONTADOR=20
until [  $CONTADOR -lt 10 ]; do
    echo CONTADOR $CONTADOR
    let CONTADOR-=1
done
```

8 Funciones

Como en casi todo lenguaje de programación, puede utilizar funciones para agrupar trozos de código de una manera más lógica, o practicar el divino arte de la recursión.

Declarar una función es sólo cuestión de escribir function mi_func { mi_código }.

Llamar a la función es como llamar a otro programa, sólo hay que escribir su nombre.

8.1 Ejemplo de funciones

```
#!/bin/bash
function salir {
    exit
}
function hola {
    echo ¡Hola!
}
hola
salir
echo petete
```

Las líneas 2-4 contienen la función 'salir'. Las líneas 5-7 contienen la función 'hola'. Si no está completamente seguro de lo que hace este script, por favor, ¡pruébelo!.

Tenga en cuenta que una función no necesita que sea declarada en un orden específico.

Cuando ejecute el script se dará cuenta de que: primero se llama a la función 'hola', luego a la función 'quit', y el programa nunca llega a la línea 10.

8.2 Ejemplo de funciones con parámetros

```
#!/bin/bash
function salir {
   exit
}
function e {
    echo $1
}
e Hola
e Mundo
salir
echo petete
```

Este script es casi idéntico al anterior. La diferencia principal es la función 'e'. Esta función imprime el primer argumento que recibe. Los argumentos, dentro de las funciones, son tratados de la misma manera que los argumentos suministrados al script.

9 Interfaces de usuario

9.1 Utilizando select para hacer menús sencillos

```
#!/bin/bash
OPCIONES="Hola Salir"
select opt in $OPCIONES; do
    if [ "$opt" = "Salir" ]; then
     echo done
     exit
    elif [ "$opt" = "Hola" ]; then
     echo Hola Mundo
    else
     clear
     echo opción errónea
    fi
done
```

Si ejecuta este script verá que es el sueño de un programador para hacer menús basados en texto. Probablemente se dará cuenta de que es muy similar a la construcción 'for', sólo que en vez de iterar para cada 'palabra' en $OPCIONES, se lo pide al usuario.

9.2 Utilizando la línea de comandos

```
#!/bin/bash
if [ -z "$1" ]; then
    echo uso: $0 directorio
    exit
fi
SRCD=$1
TGTD="/var/backups/"
OF=home-$(date +%Y%m%d).tgz
tar -cZf $TGTD$OF $SRCD
```

Lo que hace este script debería estar claro para usted. La expresión del primer condicional comprueba si el programa ha recibido algún argumento ($1) y sale si no lo ha recibido, mostrándole al usuario un pequeño mensaje de uso. El resto del script debería estar claro.

10 Miscelánea

10.1 Leyendo información del usuario

En muchas ocasiones, puede querer solicitar al usuario alguna información, y existen varias maneras para hacer esto. Ésta es una de ellas:

```
#!/bin/bash
echo Por favor, introduzca su nombre
read NOMBRE
echo "¡Hola $NOMBRE!"
```

Como variante, se pueden obtener múltiples valores con read. Este ejemplo debería clarificarlo.

```
#!/bin/bash
echo Por favor, introduzca su nombre y primer apellido
read NO AP
echo "¡Hola $AP, $NO!"
```

10.2 Evaluación aritmética

Pruebe esto en la línea de comandos (o en una shell):

echo 1 + 1

Si esperaba ver '2', quedará desilusionado. ¿Qué hacer si quiere que BASH evalúe unos números? La solución es ésta:

echo $((1+1))

Esto producirá una salida más 'lógica'. Esto se hace para evaluar una expresión aritmética. También puede hacerlo de esta manera:

echo $[1+1]

Si necesita usar fracciones, u otras matemáticas, puede utilizar bc para evaluar expresiones aritméticas.

Si ejecuta "echo $[3/4]" en la línea de comandos, devolverá 0, porque bash sólo utiliza enteros en sus respuestas. Si ejecuta "echo 3/4|bc -l", devolverá 0.75.

10.3 Encontrando el bash

De un mensaje de mike (vea los agradecimientos):

siempre usas #!/bin/bash .. a lo mejor quieres dar un ejemplo

de cómo saber dónde encontrar el bash.

'locate bash' es preferible, pero no todas las máquinas

tienen locate.

'find ./ -name bash' desde el directorio raíz funcionará,

normalmente.

Sitios donde poder buscar:

ls -l /bin/bash

ls -l /sbin/bash

ls -l /usr/local/bin/bash

ls -l /usr/bin/bash

ls -l /usr/sbin/bash

ls -l /usr/local/sbin/bash

(no se me ocurre ningún otro directorio... lo he encontrado

la mayoría de estos sitios en sistemas diferentes).

También puedes probar 'which bash'.

10.4 Obteniendo el valor devuelto por un programa

En bash, el valor de retorno de un programa se guarda en una variable especial llamada $?.

Esto ilustra cómo capturar el valor de retorno de un programa. Supongo que el directorio *dada* no existe. (Esto también es sugerencia de Mike).

```
#!/bin/bash
cd /dada &> /dev/null
echo rv: $?
cd $(pwd) &> /dev/null
echo rv: $?
```

10.5 Capurando la salida de un comando

Este pequeño script muestra todas las tablas de todas las bases de datos (suponiendo que tenga MySQL instalado). Considere también cambiar el comando 'mysql' para que use un nombre de usuario y clave válidos.

```
#!/bin/bash
DBS=`mysql -uroot  -e"show databases"`
for b in $DBS ;
do
        mysql -uroot -e"show tables from $b"
done
```

11 Tablas

11.1 Operadores de comparación de cadenas

s1 = s2

s1 coincide con s2

s1 != s2

s1 no coincide con s2

s1 < s2

s1 es alfabéticamente anterior a s2, con el *locale* actual

s1 > s2

s1 es alfabéticamente posterior a s2, con el *locale* actual

-n s1

s1 no es nulo (contiene uno o más caracteres)

-z s1

s1 es nulo

11.2 Ejemplo de comparación de cadenas

Comparando dos cadenas

```
#!/bin/bash
S1='cadena'
S2='Cadena'
if [ $S1!=$S2 ];
then
        echo "S1('$S1') no es igual a S2('$S2')"
fi
if [ $S1=$S1 ];
then
        echo "S1('$S1') es igual a S1('$S1')"
fi
```

Cito aquí el consejo de un correo enviado por Andreas Beck, referido al uso de *if [$1 = $2]*.

Esto no es buena idea, porque si $S1 o $S2 son vacíos, aparecerá un *parse error*. Es mejor: x$1=x$2 or "$1"="$2"

11.3 Operadores aritméticos

+ (adición)

- (sustracción)

* (producto)

/ (división)

% (módulo)

11.4 Operadores relacionales aritméticos

-lt (<)

-gt (>)

-le (<=)

-ge (>=)

-eq (==)

-ne (!=)

Los programadores de C tan sólo tienen que corresponder el operador con su paréntesis.

11.5 Comandos útiles

Esta sección ha sido reescrita por Kees (véanse agradecimientos)

Algunos de estos comandos contienen lenguajes de programación completos. Sólo se explicarán las bases de estos comandos. Para una descripción más detallada, eche un vistazo a las páginas man de cada uno.

sed (editor de flujo)

Sed es un editor no interactivo. En vez de alterar un fichero moviendo el cursor por la pantalla, se utiliza una serie de instrucciones de edición de sed, y el nombre del fichero a editar. También se puede describir a sed como un filtro. Miremos algunos ejemplos:

```
$sed 's/a_sustituir/sustituto/g' /tmp/petete
```

Sed sustituye la cadena 'a_sustituir' por la cadena 'sustituto', leyendo del fichero /tmp/petete. El resultado se envía a stdout (normalmente la consola), pero se puede añadir '> captura' al final de la línea de arriba para que sed envíe la salida al fichero 'capture'.

```
$sed 12, 18d /tmp/petete
```

Sed muestra todas las líneas de /tmp/petete excepto la 12 y la 18. El fichero original no queda alterado por este comando.

awk (manipulación de bases de datos, extracción y proceso de texto)

Existen muchas implementaciones del lenguaje de programacin AWK (los intérpretes más conocidos son gawk de GNU, y el 'nuevo awk' mawk). El principio es sencillo: AWK busca un patrón, y por cada patrón de búsqueda que coincida, se realiza una acción.

Si tenemos un fichero /tmp/petete con las siguientes líneas:

"prueba123

prueba

pprruueebbaa"

y ejecutamos:

```
$awk '/prueba/ {print}' /tmp/petete
```

test123

test

El patrón que busca AWK es 'prueba' y la acción que realiza cuando encuentra una línea en /tmp/petete con la cadena 'prueba' es 'print'.

```
$awk '/prueba/ {i=i+1} END {print i}' /tmp/petete
```

3

Cuando se utilizan muchos patrones, se puede reemplazar el texto entre comillas por '-f fichero.awk', y poner todos los patrones y acciones en 'fichero.awk'.

grep (impresión de líneas que coinciden con un patrón de búsqueda)

Ya hemos visto ejemplos del comando grep en los capítulos anteriores, que muestra las líneas que concuerdan con un patrón. Pero grep puede hacer más que eso.

```
$grep "busca esto" /var/log/messages -c
```

12

Se ha encontrado 12 veces la cadena "busca esto" en el fichero /var/log/messages.

[vale, este ejemplo es falso, el fichero /var/log/messages está alterado :-)]

wc (cuenta líneas, palabras y bytes)

En el siguiente ejemplo, vemos que la salida no es lo que esperábamos. El fichero petete utilizado en este ejemplo contiene el texto siguiente:

"programación en bash
como de introducción"

```
$wc --words --lines --bytes /tmp/petete
```

2 5 41 /tmp/petete

Wc no tiene en cuenta el orden de los parámetros. Wc siempre los imprime en un orden estándar, que es, como se puede ver: líneas, palabras, bytes y fichero.

sort (ordena líneas de ficheros de texto)

Esta vez, el fichero petete contiene el texto siguiente:

"b
c
a"

```
$sort /tmp/petete
```

Esto es lo que muestra la salida:

a
b
c

Los comandos no deberían ser tan fáciles :-)

bc (un lenguaje de programación de cálculos matemáticos)

Bc acepta cálculos desde la línea de comandos (entrada desde un fichero, pero no desde una redirección o una tubería), y también desde una interfaz de usuario. La siguiente demostración expone algunos de los comandos. Note que ejecuto bc con el parámetro -q para evitar el mensaje de bienvenida.

```
$bc -q
```

1 == 5

0

0.05 == 0.05

1

5 != 5

0

2 ^ 8

256

sqrt(9)

3

while (i != 9) {

i = i + 1;

print i

}

123456789

quit

tput (inicializa una terminal o consulta la base de datos de terminfo)

Una pequeña demostración de las capacidades de tput:

```
$tput cup 10 4
```

La línea de comandos aparece en (y10,x4).

```
$tput reset
```

Limpia la pantalla y la línea de comandos aparece en (y1,x1). Observe que (y0,x0) es la esquina superior izquierda.

```
$tput cols
```

80

Muestra el número de caracteres que caben en la dirección x.

Es muy recomendable familiarizarse con estos programas (al menos). Hay montones de programillas que le permitirán hacer virguerías en la línea de comandos.

[algunos ejemplos están copiados de las páginas man o los PUFs]

12 Más scripts

12.1 Aplicando un comando a todos los ficheros de un directorio.

12.2 Ejemplo: Un script de copia de seguridad muy simple (algo mejor)

```
#!/bin/bash
ORIG="/home/"
DEST="/var/copias_de_seguridad/"
FICH=home-$(date +%Y%m%d).tgz
tar -cZf $DEST$FICH $ORIG
```

12.3 Re-nombrador de ficheros

```
#!/bin/sh
# renom: renombra múltiples ficheros de acuerdo con ciertas
# reglas
# escrito por Felix Hudson  Enero - 2000

# primero comprueba los distintos 'modos' que tiene este
# programa
# si la primera ($1) condición coincide, se ejecuta esa parte
# del programa y acaba

# comprueba la condición de prefijo
if [ $1 = p ]; then

# ahora nos libramos de la variable de modo ($1) y ponemos $2
# de prefijo
  prefijo=$2 ; shift ; shift

# una rápida comprobación para ver si se especificó algún
# fichero
# si no, hay cosas mejores que hacer que renombrar ficheros
# inexistentes!!
  if [$1 = ]; then
     echo "no se especificaron ficheros"
     exit 0
  fi

# este bucle for itera a lo largo de todos los ficheros que
# le hemos especificado al programa
# renombra cada uno de ellos
  for fichero in $*
    do
    mv ${fichero} $prefijo$fichero
  done

# ahora salimos del programa
  exit 0
fi
```

```
# comprueba si es un renombramiento con sufijo
# el resto es casi idéntico a la parte anterior
# lea los comentarios anteriores
if [ $1 = s ]; then
  sufijo=$2 ; shift ; shift

   if [$1 = ]; then
    echo "no se especificaron ficheros"
   exit 0
   fi

 for fichero in $*
  do
   mv ${fichero} $fichero$sufijo
 done

 exit 0
fi

# comprueba si es una sustitución
if [ $1 = r ]; then

  shift

# he incluído esto para no dañar ningún fichero si el
# usuario no especifica que se haga nada
# tan sólo una medida de seguridad
  if [ $# -lt 3 ] ; then
    echo "uso: renom r [expresión] [sustituto] ficheros... "
    exit 0
  fi

# elimina el resto de información
  VIEJO=$1 ; NUEVO=$2 ; shift ; shift

# este bucle for itera a lo largo de todos los ficheros que
# le hemos especificado al programa
# renombra cada fichero utilizando el programa 'sed'
# es un sencillo programa desde la línea de comandos que
# analiza la entrada estándar y sustituye una expresión por
# una cadena dada
# aquí le pasamos el nombre del fichero (como entrada
# estándar)
  for fichero in $*
  do
    nuevo=`echo ${fichero} | sed s/${VIEJO}/${NUEVO}/g`
    mv ${fichero} $nuevo
  done
exit 0
fi

# si se llega a esta parte es que no se le pasó nada
# apropiado al programa, por lo que le decimos al usuario
# cómo hacerlo
echo "uso:"
```

```
        echo " renom p [prefijo] ficheros.."
        echo " renom s [sufijo] ficheros.."
        echo " renom r [expresión] [sustituto] ficheros.."
        exit 0

        # hecho!
```

12.4 Re-nombrador de ficheros (sencillo)

```
#!/bin/bash
# renombra.sh
# renombrador de ficheros básico

criterio=$1
expresion=$2
sustituto=$3

for i in $( ls *$criterio* );
do
    orig=$i
    dest=$(echo $i | sed -e "s/$expresion/$sustituto/")
    mv $orig $dest
done
```

13 Cuando algo va mal (depuración)

13.1 Maneras de llamar a BASH

Una buena idea es poner esto en la primera línea:

```
#!/bin/bash -x
```

Esto producirá información interesante.

14 Sobre el documento

Siéntase libre para hacer sugerencias/correcciones, o lo que crea que sea interesante que aparezca en este documento. Intentaré actualizarlo tan pronto como me sea posible.

14.1 (sin) Garantía

Este documento no lleva garantía de ningún tipo.

14.2 Traducciones

Italiano: por William Ghelfi (wizzy está en tiscalinet.it). *http://web.tiscalinet.it/penguin_rules*

Francés: por Laurent Martelli *¿?*

Coreano: Minseok Park *http://kldp.org*

Corean: Chun Hye Jin *Desconocido*

Spanish: Gabriel Rodríguez Alberich *http://www.insflug.org*

Supongo que habrá más traducciones, pero no tengo información sobre ellas. Si las tiene, por favor, envíemelas para que actualice esta sección.

14.3 Agradecimientos

- A la gente que ha traducido este documento a otras lenguas (sección anterior).
- A Nathan Hurst por enviar montones de correcciones.
- A Jon Abbott por enviar comentarios sobre la evaluación de expresiones aritméticas.
- A Felix Hudson por escribir el script *renom*
- A Kees van den Broek (por enviar tantas correcciones y reescribir la sección de comandos útiles)
- Mike (pink) hizo algunas sugerencias sobre la localización del bash y la comprobación de los ficheros
- Fiesh hizo una buena sugerencia sobre la sección de bucles.
- Lion sugirió mencionar un error común (./hello.sh: Comando no encontrado.)
- Andreas Beck hizo varias correcciones y comentarios.

14.4 Historia

Añadidas nuevas traducciones y correcciones menores.

Añadida la sección de comandos útiles reescrita por Kess.

Incorporadas más correcciones y sugerencias.

Añadidos ejemplos sobre la comparación de cadenas.

v0.8 abandono del versionamiento. Supongo que con la fecha es suficiente.

v0.7 Más correcciones y algunas secciones TO-DO escritas.

v0.6 Correcciones menores.

v0.5 Añadida la sección de redireccionamiento.

v0.4 desaparición de su sitio debido a mi ex-jefe. Este documento tiene un nuevo sitio en: http://www.linuxdoc.org.

Anteriores: no me acuerdo y no he usado rcs ni cvs :(

14.5 Más recursos

Introducción a bash (bajo BE) *http://org.laol.net/lamug/beforever/bashtut.htm*

Programación en Bourne Shell http://207.213.123.70/book/