Curso de sistemas GNU/Linux
Bash scripting
Argumentos, entrada estándar, funciones e includes.

Por Jorge Fuertes
http://jorgefuertes.com

Revisado a 15 de mayo de 2009

Índice

1. Introducción

En la programación de guiones muchas veces escribimos una simple secuencia de órdenes que hace el trabajo que necesitamos, pero que nos evitará tener que teclear dichas órdenes constantemente. En cuanto queremos refinar un poquito más nuestros guiones para que no sean una mera secuencia de órdenes necesitaremos hacer cosas como admitir argumentos en la llamada a nuestro programa, definir funciones que puedan ser llamadas desde distintas partes del código para poder romper con la estructura lineal del guión y además necesitaremos reutilizar nuestro código, definiendo colecciones de funciones que puedan ser integradas en diversos guiones. Quizá en algún momento necesitemos también admitir un fichero o un texto por la entrada estándar.

Todo esto lo podemos hacer en Bash. La entrada estándar podremos leerla con `/dev/stdin`, los argumentos con las variables predefinidas `$#` y `$numero_argumento`, las funciones definiéndolas con `function` y la inclusión de colecciones u otras partes de programa con `source`.

2. La entrada estándar

Leer la entrada estándar de un guión es muy sencillo. Basta con un simple `cat /dev/stdin`, por ejemplo:

```
#!/bin/bash

echo "La entrada estándar es:"
cat /dev/stdin
echo "<EOF>"
```

Podemos probarlo así:

```
#> echo "Hola"|sh prueba.sh
La entrada estándar es:
Hola
<EOF>
```

3. Argumentos de llamada al programa

3.1. ¿Qué son los argumentos?

Los argumentos son una serie de valores que pasamos a nuestro programa en la línea de comandos desde la que lo invocamos. Por ejemplo:

`#> mi-programa.sh 100 hola`

El nombre de nuestro programa es `mi-programa.sh`, el primer argumento es **"100"** y el segundo **"hola"**.

Obsérvese que los argumentos van separados por espacios. Si queremos que una frase sea un sólo argumento, deberemos entrecomillarla.

3.2. Número de argumentos

Podemos saber cuantos argumentos se han introducido en la línea de comandos gracias a la variable predefinida `$#`. Un ejemplo:

```
#!/bin/bash

echo "Has introducido $# argumentos."
```

Si lo ejecutamos:

```
#> programa.sh 100 200 50 hola

Has introducido 4 argumentos.
```

3.3. Los argumentos

Recuperaremos los argumentos desde nuestro programa haciendo uso de las variables predefinidas `$1`, `$2`, `$3`, etc. La variable `$0` no es un argumento sino que corresponde al nombre de nuestro script, y de la 1 en adelante son cada uno de los argumentos que se han pasado en la línea de comandos.

Por ejemplo:

```
#!/bin/bash

echo "Este programa se llama $0"
echo "Has introducido $# argumentos que son:"
echo "El 1º: $1"
echo "El 2º: $2"
echo "El 3º: $3"
echo "Y el resto ya los sabes."
```

Evidentemente el programa anterior es muy mejorable, por ejemplo, y lo haremos en los ejercicios, podríamos hacer un bucle que mirase cuantos argumentos se han introducido y mostrase cada uno de ellos.

Se podría hacer algo así:

```
#!/bin/bash

echo "Este programa se llama $0"
echo "Has introducido $# argumentos que son:"

for i in $(seq 1 $#)
do
        echo -n "Argumento $i: "
        eval echo \$$i
done
```

4. Funciones

4.1. Definiendo y utilizando funciones

Para definir una función utilizaremos la palabra reservada "*function*". En Bash resulta muy sencillo definir funciones ya que no hay que especificar ni el número ni el tipo de argumentos, ni siquiera el retorno.

Por ejemplo en el siguiente programa definimos una función que suma dos números y a continuación la utilizamos:

```
#!/bin/bash

# Función que suma:
function suma {
    echo "$1 + $2"|bc
}

suma 5 10
suma 10 20

echo "Fin de programa."
```

Es una forma muy sencilla ya que la propia función escribe en pantalla el resultado de la operación.

4.2. Argumentos de las funciones

Al igual que un programa, en el ámbito de una función, los argumentos están contenidos en las variables `$1`, `$2`, `$3`, etc...

Mejor veámoslo en el siguiente ejemplo:

```
#!/bin/bash

function prueba {
    echo "El primer argumento es: $1."
    echo "El segundo argumento es: $2."
}

# Llamada a la función:
prueba "Hola" "Mundo"

echo "Fin de programa."
```

4.3. Variables por referencia

Podemos pasarle a nuestras funciones un nombre de variable que podrá ser utilizado por la misma para guardar algún valor. Por ejemplo en el siguiente programa definimos una función que nos devuelve el año actual en la variable que le decimos:

```
#!/bin/bash

# Devuelve el año actual:
function year {
YEAR=$(date +"%Y")
eval "$1=\"$YEAR\""
}

# La probamos:

echo -c "El año actual es: "
year

echo
echo "Fin de programa."
```

5. Incluir ficheros de programa y funciones

5.1. Partes de programa, ficheros de configuración

A veces, por claridad y por reutilización de código necesitaremos separar nuestro programa en trozos más pequeños. Podemos escribir partes de nuestro programa en ficheros que después incluiremos en nuestro programa principal con la orden `source`.

Por ejemplo si creamos un fichero de configuración, que incluya una serie de variables que queremos definir al principio de nuestro programa, crearíamos un fichero `configuracion.inc.sh` así:

```
# configuración.inc.sh
# Configuración de mi programa:

CAMINO="/root/scripts"
LOG="/var/log/milog.log"
TMP="/tmp/mifichero.tmp"
NUM1=100
```

Y en nuestro programa principal haríamos lo siguiente:

```
#!/bin/bash

# Este es mi programa principal.

# En primer lugar cargamos la configuración:
souce /root/scripts/configuracion.inc.sh

# Desde aquí ya disponemos de las variables de configuración:
echo "Esta aplicación reside en $CAMINO"
echo "Y tiene un fichero temporal en $TMP"

echo $NUM1 > $TMP
```

```
echo "Ejecutado" >> $LOG
echo "Fin de programa."
```

5.2. Colecciones de funciones

Podemos crear ficheros con distintas colecciones o bibliotecas de funciones para reutilizarlas en nuestro programas. Por ejemplo podemos crear una la entrada de usuario y llamarle `input.inc.sh`:

```
# Mi biblioteca de funciones.

# pregunta <texto> <var_sin_dolar> [por defecto]
# Hace una pregunta al usuario metiendo la respuesta en
# la variable pasada por referencia. Admite un valor
# por defecto.

function pregunta {
    RESPUESTA=""
    echo -e "> ${1} (${3}): \c"
    read RESPUESTA
    if [ -z "$RESPUESTA" ]
    then
        RESPUESTA=$3
    fi
    eval "$2=\"$RESPUESTA\""
}
```

Ahora podemos llamar a funciones de esta biblioteca desde cualquiera de nuestros programas, tal y como hacemos en el siguiente ejemplo:

```
#!/bin/bash

# Incluir la biblioteca de funciones:
source input.inc.sh

echo "Hola, bienvenido."

pregunta "¿Cómo te llamas?" NOMBRE "Pepe"

echo "Tu nombre es $NOMBRE."
```

Como vemos, el programa incluye la biblioteca de funciones con la orden "*source*" y a partir de ahí puede utilizar cualquiera de las funciones de dicha biblioteca, aunque en este caso sólo hay una, que es una función para realizar preguntas al usuario admitiendo valores por defecto.

6. Ejercicios

6.1. Notas

- **Scripts o guiones**: Cree un script o guión de *Bash* para cada uno de los ejercicios, y llámelo `ej-func-num.sh`, siendo *num* el número de ejercicio[1]. Por ejemplo el script del ejercicio 1 deberá llamarse `ej-func-1.sh`.
- **Funciones**: También debe crear un fichero llamado `funciones.inc.sh` que contendrá todas las funciones que necesite para sus programas.
- **Modificaciones**: Si se le pide modificar un programa anterior, lo que debe hacer es copiar el programa anterior pero con el nombre correspondiente al ejercicio que esté haciendo, es decir, al acabar los ejercicios debe tener un programa para cada uno de ellos.
- **Ficheros de configuración**: Si necesita crear un fichero de configuración este deberá llamarse `ej-func-num-conf.inc.sh`, siendo *num* el número de ejercicio. Por ejemplo la configuración del ejercicio 1, de necesitarse, deberá llamarse `ej-func-1-conf.inc.sh`.

6.2. Enunciados

1. Cree un programa que diga su propio nombre de programa, aunque este sea renombrado (mv) por el usuario, que diga cuantos argumentos se le han pasado y que ponga cada uno de ellos en una línea, con el número de argumento delante.

2. Cree un programa que cuente cuantas palabras hay en un fichero que se le pasará como argumento.

3. Modifique el programa anterior para que admita también el contenido del fichero por la entrada estándar.

[1] En Español, la construcción "siendo num tal cosa" quiere decir que hay que sustituir *num* por lo que se dice a continuación, no que haya que poner literalmente *num*.

4. Haga un programa que ordene una lista de palabras que se le pasen por la entrada estándar pero que admita como argumento la palabra "inverso" que, de pasarse, invertirá la ordenación de ascendente a descendente.

5. Debe escribir un programa que acepte operaciones numéricas por la entrada estándar, por ejemplo `2 + 4 * 3` y que además de mostrar el resultado por pantalla almacene la operación y el resultado en un fichero llamado `cinta.txt`.

6. Escriba un programa que admita un nombre como argumento (ej. Juan). Dicho nombre deberá ser almacenada en el fichero `edades.txt` si no existe ya él, y además se deberá preguntar el teléfono de dicho nombre y almacenarlo junto a él. Si el nombre ya está registrado se mostrará por pantalla el teléfono y se preguntará al usuario si desea modificarlo, preguntando un nuevo teléfono y cambiándolo en el fichero.

7. Cree un fichero que sirva como configuración y que contenga las variables `NOMBRE` y `EDAD` definidas con los valores "*pepe*" y "*33*". Después cree un programa aparte que diga el nombre y la edad configuradas.

8. Escriba una biblioteca de funciones y programe una función *"decir"* que admita una cadena de texto. Dicha cadena de texto deberá salir tanto por pantalla como a un fichero de log en `/var/log/ej-func.log`. Escriba además el programa que utilice esa función en dos ocasiones.

9. Agregue a su biblioteca de funciones una que diga la fecha actual en formato *dia-mes-año* y otra que diga la hora en formato *horas:minutos:segundos*. Escriba un programa que utilice ambas funciones.

10. Añada una función que obtenga la **mac-address** del interfaz que le pasen como primer argumento y que la meta en la variable que le pasen como segundo argumento. Añada también otra función que haga lo mismo pero con la **dirección IP**. Para terminar escriba un programa que diga tanto por pantalla como por un fichero log la dirección mac y la IP que tiene su interfaz **eth0**.

6.3. Soluciones

1. Cree un programa que diga su propio nombre de programa, aunque este sea renombrado (mv) por el usuario, que diga cuantos argumentos se le han pasado y que ponga cada uno de ellos en una línea, con el número de argumento delante.

```
#!/bin/bash

clear

echo "este programa se llama $(echo $0|cut -f2 -d'/')"
echo
echo "has pasado $# argumentos"
echo

if [ $# -gt 0 ]
then
        for i in $(seq 1 $#)
        do
                echo "$i $(eval echo \$$i)"
        done
fi
echo
```

2. Cree un programa que cuente cuantas palabras hay en un fichero que se le pasará como argumento.

```
#!/bin/bash

resul=$(cat $1|wc -w)
echo -e "El fichero ${1} tiene ${resul} palabras.\n"
```

3. Modifique el programa anterior para que admita también el contenido del fichero por la entrada estándar.

```
#!/bin/bash

if [ $# -eq 0 ]
then
        NPALABRAS=$(cat /dev/stdin | wc -w)
        echo "Registradas $NPALABRAS palabras por la entrada estandar."
else
        NPALABRAS=$(cat $1|wc -w)
        echo "El fichero $1 tiene $NPALABRAS palabras."
fi
```

4. Haga un programa que ordene una lista de palabras que se le pasen por la entrada estándar pero que admita como argumento la palabra "inverso" que, de pasarse, invertirá la ordenación de ascendente a descendente.

```
#!/bin/bash

if [ "$1" == "inverso" ]
then
        echo "Ordenación inversa:"
        cat /dev/stdin | grep -Ev "^$" | sort -r
else
        echo "Ordenación normal:"
        cat /dev/stdin | grep -Ev "^$" | sort
fi

echo "Fin."
```

5. Debe escribir un programa que acepte operaciones numéricas por la entrada estándar, por ejemplo `2 + 4 * 3` y que además de mostrar el resultado por pantalla almacene la operación y el resultado en un fichero llamado `cinta.txt`.

```
#!/bin/bash

OP=$(cat /dev/stdin)
RES=$(echo "scale=2; $OP"|bc)

echo "La operación $OP es igual a $RES"

echo "$OP = $RES" >> cinta.txt

echo "Resultado guardado en cinta.txt."
```

6. Escriba un programa que admita un nombre como argumento (ej. Juan). Dicho nombre deberá ser almacenada en el fichero `edades.txt` si no existe ya él, y además se deberá preguntar el teléfono de dicho nombre y almacenarlo junto a él. Si el nombre ya está registrado se mostrará por pantalla

el teléfono y se preguntará al usuario si desea modificarlo, preguntando un nuevo teléfono y cambiándolo en el fichero.

```
#!/bin/bash

function guardar {
        echo "Guardando nuevo usuario $1..."
        read -p "introduce el teléfono " tel
        echo "$1 $tel" >> edades.txt
        echo "Guardado."
}

function buscar {
        echo "Buscando $1..."
        cat edades.txt | grep $1 &> /dev/null
        return $?
}

clear

if [ -f edades.txt ]
then
        buscar $1
        if [ $? -eq 0 ]
        then
                # Encontrado:
                echo "el usuario $1 existe"
                echo
                tel=$(cat edades.txt|grep $1|cut -d" " -f2)
                echo "y su teléfono es $tel"
                echo
                read -p "¿quieres modificar el teléfono? (y/n): " cambio
                if [ "$cambio" == "y" ]
                then
                        read -p "introduce el número teléfono: " telnuevo
                        sed "s/${tel}/${telnuevo}/" edades.txt > edades2.txt
                        mv edades2.txt edades.txt
                        echo "Guardado."
                fi
        else
                echo "el usuario $1 no existe"
                guardar $1
        fi
else
        echo "No hay ningún nombre registrado todavía."
        guardar $1
fi
```

7. Cree un fichero que sirva como configuración y que contenga las variables

`NOMBRE` y `EDAD` definidas con los valores "*pepe*" y "*33*". Después cree un programa aparte que diga el nombre y la edad configuradas.

Configuración:

```
# Fichero de configuración:

nombre=pepe
edad=33
```

Programa:

```
#!/bin/bash

source ej-func-7-conf.inc.sh
echo "el nombre es $nombre y la edad es $edad"
```

8. Escriba una biblioteca de funciones y programe una función *"decir"* que admita una cadena de texto. Dicha cadena de texto deberá salir tanto por pantalla como a un fichero de log en `/var/log/ej-func.log`. Escriba además el programa que utilice esa función en dos ocasiones.

```
#!/bin/bash

source funciones.inc.sh

decir "Hola Mundo."
decir "Hello World."
```

9. Agregue a su biblioteca de funciones una que diga la fecha actual en formato *dia-mes-año* y otra que diga la hora en formato *horas:minutos:segundos*. Escriba un programa que utilice ambas funciones.

```
#!/bin/bash

source funciones.inc.sh

fecha
hora
```

10. Añada una función que obtenga la **mac-address** del interfaz que le pasen como primer argumento y que la meta en la variable que le pasen como segundo argumento. Añada también otra función que haga lo mismo pero con la **dirección IP**. Para terminar escriba un programa que diga tanto por pantalla como por un fichero log la dirección mac y la IP que tiene su interfaz **eth0**.

```
#!/bin/bash

# Configuración:
LOG="/var/log/ej-func-10.log"
INTERFAZ="eth0"

clear

source funciones.inc.sh

function decir {
        echo "$1"
        echo "$1" >> $LOG
}

mac $INTERFAZ MAC
dirip $INTERFAZ IP

decir "Interfaz $INTERFAZ:"
decir "mac-address....: $MAC"
decir "ip ............: $IP"
```

6.3.1. Ficheros auxiliares

Funciones (*funciones.inc.sh*):

```
#!/bin/bash

# Biblioteca de funciones

function decir {
        echo "> $1"
        echo "$1" >> /var/log/ej-func-8.log
}

function fecha {
        echo "Fecha: " $(date +"%d-%m-%Y")
}

function hora {
        echo "Hora: " $(date +"%H:%M:%S")
}

function mac {
        MAC=$(ip l show $1|grep "ether "|tr -s " " "|"|cut -f3 -d"|")
        eval "$2=$MAC"
}

function dirip {
        IP=$(ip a show eth1|grep "inet "|tr -s " " "|"|cut -f3 -d"|"|cut -f1 -d"/")
        eval "$2=$IP"
}
```

7. Sobre esta unidad didáctica

7.1. Notas y advertencias

Debian: Esta guía está basada en el sistema *Debian GNU/Linux*, podría haber pequeños cambios si se aplica a otras distribuciones de *GNU*, pero en su mayor parte funcionará bien con la excepción de lo referido al sistema de paquetería de programas, los comandos que empiezan por *apt*, ya que otras *distros* no basadas en *Debian* podrían incorporar sistemas diferentes para el manejo de sus paquetes.

7.2. Derechos

Esta guía se cede bajo contrato Coloriuris. Sólo puede ser utilizada previa aceptación del contrato de cesión sito en:

- http://www.coloriuris.net/contratos/ef5af6aaa441ab9c213273fade56dca1

Dicho contrato garantiza que estoy cediendo los derechos de uso y modificación sin ánimo de lucro.

7.3. Agradecimientos

El autor quiere reflejar su agradecimiento a todas las páginas de Internet que ponen a disposición de todo el mundo sus contenidos, así como a todo aquél que publica artículos, manuales y experiencias en Internet, ya que eso favorece a la difusión del conocimiento y al desarrollo humano. *La información quiere ser libre.*

Un agradecimiento muy especial a toda la comunidad del Software Libre. Sin ellos el autor viviría en la oscuridad: Programadores, traductores, asociaciones, hacktivistas, webmasters, etc...

También quiero agradecer muy especialmente su ayuda a mis alumnos y lectores, por tomarse la molestia de comunicarme las erratas y por darme ideas para mejorar los ejercicios.

7.4. Revisiones

El autor irá eventualmente publicando revisiones de esta unidad en su página personal, y estará encantado de recibir sugerencias y dudas en la misma o en su email:

- http://jorgefuertes.com.
- cursos@jorgefuertes.com.

Por supuesto se puede contactar con el autor para contratarle para hacer nuevas unidades, adaptaciones, modificaciones, cursos, etc...

8. Anexo: Una biblioteca de funciones de ejemplo

8.1. Introducción

La siguiente biblioteca es una colección de funciones que el autor utiliza habitualmente para sus propios programa de Bash. Su utilidad para el lector puede ser evidente o no, ya que dependiendo de la naturaleza de nuestro trabajo necesitaremos unos u otros apoyos en nuestro desarrollo, sin embargo, las funciones aquí presentadas son bastante generales y contemplan algunos *trucos* para mejorar y facilitar la interacción con el usuario y la entrada/salida.

El alumno debería construir un programa que utilice todas las funciones al menos una vez, con la salvedad de las relacionadas con base de datos, que sólo funcionarán de estar instalado **MySQL**.

En cuanto a su uso fuera del propio de estos apuntes que ya tienen su propia licencia, podrá considerarse que la biblioteca es GPL2 a todos los efectos. El autor agradecerá correcciones y mejoras a la misma.

8.2. La biblioteca

```
# funciones.inc.sh
#
# Biblioteca de funciones de uso general.
#
# Copyright (C) 2007 Jorge Fuertes (queru@queru.org)
#
# Este programa es software libre: usted puede redistribuirlo
# y/o modificarlo bajo los términos de la Licencia Pública
# General GNU publicada por la Fundación para el Software
# Libre, ya sea la versión 3 de la Licencia, o (a su elección)
# cualquier versión posterior.
#
# Este programa seA ALGUNA; ni siquiera la garantía implícita ,
# MERCANTIL o de APTITUD PARA UN PROP?SITO DETERMINADO.
# Consulte los detalles de la Licencia Pública General GNU para
# obtener una información más detallada.
#
# Debería haber recibido una copia de la Licencia Pública General
# GNU junto a este programa.
# En caso contrario, consulte http://www.gnu.org/licenses/
#

# Configuración de colores:
NORMAL="\e[0m"
BOLD="\e[1m"
INVERSO="$BOLD\e[30;47m"
ROJO="$BOLD\e[31m"
VERDE="$BOLD\e[32m"
MARRON="$BOLD\e[33m"
AZUL="$BOLD\e[34m"
```

```
MAGENTA="$BOLD\e[35m"
CYAN="$BOLD\e[36m"
BLANCO="$BOLD\e[37m"
FORTUNECOLOR="$MARRON"
AMARILLO="$MARRON"


# +---------------+
# | Las funciones |
# +---------------+

# titulo <texto>
# Escribe un título en pantalla, para que todos los programas
# tengan un aspecto común.
function titulo {
    echo -e "\n${BLANCO}---=[${CYAN}${1}${BLANCO}]=---${NORMAL}"
}

# ok <errorlevel>
# Para usar después de un 'haciendo', cierra la línea con OK o FALLO
# dependiendo del errorlevel pasado. Normalmente 'ok $?'.
function ok {
    if [ $1 -eq 0 ]
    then
        echo -e "${VERDE}OK${NORMAL}"
    else
        echo -e "${ROJO}FALLO${NORMAL} (Cod.${1})"
    fi
}

# pregunta <texto> <var_sin_dolar> [por defecto]
# Hace una pregunta al usuario, poniéndo el resultado en la variable
# del segundo argumento y poniéndo el tercer argumento como respuesta
# si el usuario responde en blanco.
function pregunta {
    RESPUESTA=""
    echo -e "${VERDE}>${BLANCO}${1}${NORMAL} (${3}): \c"
    read RESPUESTA
    if [ -z "$RESPUESTA" ]
    then
        RESPUESTA=$3
    fi
    eval "$2=\"$RESPUESTA\""
}

# preguntaobg <texto> <var_sin_dolar> [por defecto]
# Igual que la anterior, pero una respuesta es obligatoria
# si no se pasa valor por defecto.
function preguntaobg {
    RESPUESTA=""
```

```
    while [ -z "$RESPUESTA" ]
    do
        echo -e "${VERDE}>${BLANCO}${1}${NORMAL} (${3})(*): \c"
        read RESPUESTA
        if [ -z "$RESPUESTA" ]
        then
            RESPUESTA=$3
        fi
    done
    eval "$2=\"$RESPUESTA\""
}

# haciendo <texto>
# Para iniciar una acción informando al usuario.
# Al terminar dicha acción se deberá usar 'ok $?'.
function haciendo {
    echo -e "  ${AMARILLO}- ${BLANCO}${1}${NORMAL}...\c"
}

# informa <texto>
# Da una información al usuario.
function informa {
    echo -e "${AMARILLO}+${NORMAL} ${1}${NORMAL}"
}

# finalizado <errorlevel>
# Finaliza el programa saliendo con el errorlevel que se le diga.
function finalizado {
    echo -e "\n*** ${BLANCO}Finalizado${NORMAL} ***"
    exit $1
}

# query <texto> <sql>
# Lanza una consulta a MySQL y muestra el resultado.
function query {
    haciendo $1
    RES=$(echo $2 | mysql|tr "\n" "|")
    ok $?
    if [ -z "$RES" ]
    then
        informa "Sin resultado."
        return 1
    else
        informa "Resultado:"
        echo $RES|tr "|" "\n"
    fi
}

# sql <texto> <sql>
# Envía SQL sin esperar respuesta:
```

```
function sql {
    haciendo $1
    echo $2 | mysql
    ok $?
}

# sino <texto>
# Hace una pregunta al usuario pero sólo le permite
# responder 's' o 'n'. Devuelve el estado 0 o 1.
function sino {
    echo -e "${VERDE}>${BLANCO}${1}${NORMAL} (s/N): \c"
    read -n1 RESPUESTA
    echo
    if [[ "$RESPUESTA" == "s" || "$RESPUESTA" == "S" ]]
    then
        return 0
    else
        return 1
    fi
}

# errorgrave <texto>
# Muestra un error grave y sale del programa.
function errorgrave {
    echo -e "\n${ROJO}> ERROR${NORMAL}: ${1}\n"
    exit 1
}

# aviso <texto>
# Muestra un aviso por pantalla.
function aviso {
    echo -e "\n${AMARILLO}> ${ROJO}AVISO${NORMAL}: ${1}\n"
}

# <EOF>
```