**SHELL SCRIPT**

1. **Palabras reservadas y scripts**

Todos los lenguajes de programación tienen palabras reservadas y este no iba a ser una excepción.

A continuación, se enumeran algunas de ellas para los scripts que vamos a realizar.

**CLS**: borra la pantalla

**TITLE**: cambia el título de la ventana de comandos

**REM**: comentarios

**:NOMBRE\_ETIQUETA**: establecemos etiquetas

**GOTO** NOMBRE\_ETIQUETA: ir a la etiqueta con ese nombre

**ECHO** *mensaje*: Muestra en pantalla un mensaje concreto

**ECHO.** : Inserta una línea en blanco

**@ECHO OFF**: cancela el eco de las órdenes

**PAUSE**: paramos la ejecución del programa

**COLOR** XY: Establecemos color para el fondo (X) y para el texto (Y); si se escribe color /? en una ventana de comandos se pueden ver los códigos de los colores

**CHCP 1252**: Respeta las tildes y las eñes (ñ)

1. **Variables**

Las variables las creamos usando el comando SET.

SET nombre = Laura

Si queremos preguntar algo y asignarlo a la variable, usaremos SET /P.

SET /P nombre = Escribe tu nombre:

También podemos realizar operaciones aritméticas sobre nuestras variables con SET /A.

SET /A incremento = %n% + 1

Como habrás observado, cuando estamos utilizando una variable, la tenemos que incluir entre los signos de tanto por ciento (%).

También podemos trabajar con argumentos de entrada en la línea de comandos. Estos se usarán dentro del script como %1, %2, %3, ... dependiendo del orden en que se han introducido en la línea de comandos.

1. **Operadores booleanos**

IF %nombre% == "Laura" GOTO HOMBRE

IF NOT %nombre% == "Laura" GOTO MUJER

También podemos utilizar como comparadores:

EQU (equal to = igual), NEQ (not equal to = distinto), LSS (less than = menor que), LEQ (less than r equal to = menor o igual que), GTR (greater than = mayor que) y GEQ (greater than or equal to = mayor o igual que)

1. **Bucles**

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

SET n = 1

:BUCLE

SET /A n = %n% + 1

ECHO %n%

IF %n% NEQ 11 GOTO BUCLE

:FIN

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

FOR %i IN (1 2 3 4 5) DO ECHO %i (en cmd)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

FOR /F "tokens=1-2 delims=," %%a IN (archivo.txt) DO orden (barrido de un archivo que contiene múltiples objetos, cuyas propiedades están separadas por comas)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Con el comando de contatenación (&) podemos ejecutar dos o más órdenes en un bucle FOR.

1. **EJECICIOS resueltos Shell Script**
2. **­ Básicos**
   1. Realizar un script llamado '**01­hola­mundo.sh**' que muestre por pantalla "Hola mundo!".

#! /bin/bash

echo "Hola mundo!"

* 1. Ídem pero que en vez de "mundo" muestre los parámetros introducidos ('**02­hola­ parametros.sh**').

#! /bin/bash echo "Hola $@!"

* 1. Ídem y que además verifique que al menos hayamos introducido un parámetro ('**03­hola­al­menos­1­parametro.sh**').

#! /bin/bash

echo "número de parámetros = $#"

# si número de parámetros menor o igual que 0 if [ $# ­le 0 ]; then

echo "Hay que introducir al menos un parámetro." exit 1

fi

echo "Hola $@!"

* 1. Ídem y que además separe cada argumento por ", " ('**04­hola­parametros­ separados.sh**').

#! /bin/bash

# si número de parámetros menor o igual que 0 if [ $# ­le 0 ]; then

echo "Hay que introducir al menos un parámetro." exit 1

fi

MENSAJE="Hola"

PRIMERO=1

# mientras haya parámetros while [ ­n "$1" ]; do

if [ $PRIMERO ­eq 1 ]; then MENSAJE="$MENSAJE $1"

PRIMERO=0

else

MENSAJE="$MENSAJE, $1"

fi

# pasamos al siguiente parámetro shift

done

# mostramos la salida por pantalla echo $MENSAJE"!"

* 1. Ídem y que además en caso de error muestra una ayuda ('**05­hola­con­ayuda.sh**').

#! /bin/bash

# función de ayuda function ayuda() {

cat << DESCRPCION\_AYUDA SYNOPSIS

$0 NOMBRE\_1 [NOMBRE\_2] ... [NOMBRE\_N]

DESCRIPCION

Muestra "Hola NOMBRE\_1, NOMBRE\_2, ... NOMBRE\_N!" por pantalla.

CÓDIGOS DE RETORNO

1 Si el número de parámetros es menor que 1 DESCRPCION\_AYUDA

}

# si número de parámetros <= 0 if test $# ­le 0 ; then

echo "Hay que introducir al menos un parámetro." ayuda

exit 1 fi

MENSAJE="Hola"

PRIMERO=1

# mientras haya parámetros while [ ­n "$1" ]; do

if [ $PRIMERO ­eq 1 ]; then MENSAJE="$MENSAJE $1"

PRIMERO=0

else

MENSAJE="$MENSAJE, $1"

fi

# pasamos al siguiente parámetro shift

done

# mostramos la salida por pantalla echo $MENSAJE"!"

exit 0

* 1. Ídem y que además verifique que sean usuarios conectados al sistema ('**06­hola­ usuario.sh**').

#! /bin/bash

# función de ayuda function ayuda() {

cat << DESCRPCION\_AYUDA SYNOPSIS

$0 NOMBRE\_1 [NOMBRE\_2] ... [NOMBRE\_N]

DESCRIPCION

Muestra "Hola NOMBRE\_1, NOMBRE\_2, ... NOMBRE\_N!" por pantalla.

CÓDIGOS DE RETORNO

* + 1. Si el número de parámetros es menor que 1
    2. Si el usuario no está conectado DESCRPCION\_AYUDA

}

# si número de parámetros <= 0 if [ $# ­le 0 ] ; then

echo "Hay que introducir al menos un parámetro." ayuda

exit 1 fi

MENSAJE="Hola"

PRIMERO=1

# mientras haya parámetros while [ ­n "$1" ]; do

ESTA\_CONECTADO=`who | grep $1`

if [ ­z "$ESTA\_CONECTADO" ]; then

echo "El usuario $1 no está conectado" ayuda

exit 2

fi

if [ $PRIMERO ­eq 1 ]; then MENSAJE="$MENSAJE $1" PRIMERO=0

else fi

MENSAJE="$MENSAJE, $1"

done

# pasamos al siguiente parámetro shift

# mostramos la salida por pantalla echo ${MENSAJE}"!"

* 1. Realizar un script llamado '**usuarioconectado**' que retorna un SI si el primer parámetro coincide con algún usuario conectado o NO en caso contrario.

#! /bin/bash function ayuda() {

cat << DESCRIPCION\_AYUDA SYNOPSIS

$0 NOMBRE\_USUARIO

DESCRIPCION

Devuelve:

SI si NOMBRE\_USUARIO coincide con algún usuario conectado o NO si NOMBRE\_USUARIO no coincide con ningún usuario conectado

CÓDIGOS DE RETORNO

1 Si el número de parámetros es distinto de 1 DESCRIPCION\_AYUDA

}

# si número de parámetros distinto 1 if [ $# ­ne 1 ]; then

echo "El número de parámetros debe de igual a 1" ayuda

exit 1

fi

ESTA\_CONECTADO=`who | grep $1`

if [ ­z "$ESTA\_CONECTADO" ]; then echo "NO"

else

fi

echo "SI"

* 1. Modificar el fichero '**.bashrc**' para modificar el PATH y añadir la carpeta de estos ejercicios. Para ello añade la siguiente linea: **export PATH=$PATH":~/ruta\_carpeta\_ejercicios"**

Con esto ponemos el comando en el PATH para que pueda ejecutarse desde cualquier sitio.

* 1. Modificar el script '**06­hola­usuario.sh**' para que llame a 'usuarioconectado' ('**09­hola­usuario.sh**').

#! /bin/bash

# función de ayuda function ayuda() {

cat << DESCRPCION\_AYUDA SYNOPSIS

$0 NOMBRE\_1 [NOMBRE\_2] ... [NOMBRE\_N]

DESCRIPCION

Muestra "Hola NOMBRE\_1, NOMBRE\_2, ... NOMBRE\_N!" por pantalla.

CÓDIGOS DE RETORNO

1. Si el número de parámetros es menor que 1
2. Si el usuario no está conectado DESCRPCION\_AYUDA

}

# si número de parámetros <= 0 if [ $# ­le 0 ] ; then

echo "Hay que introducir al menos un parámetro." ayuda

exit 1 fi

MENSAJE="Hola"

PRIMERO=1

# mientras haya parámetros while [ ­n "$1" ]; do

ESTA\_CONECTADO=`./usuarioconectado $1` if [ "$ESTA\_CONECTADO" == "NO" ]; then

echo "El usuario $1 no está conectado" ayuda

exit 2

fi

if [ $PRIMERO ­eq 1 ]; then MENSAJE="$MENSAJE $1" PRIMERO=0

else fi

MENSAJE="$MENSAJE, $1"

done

# pasamos al siguiente parámetro shift

# mostramos la salida por pantalla echo ${MENSAJE}"!"

* 1. Realizar un script llamado '**usuariosistema**' que retorna un SI si el primer parámetro coincide con algún usuario del sistema o NO en caso contrario.

#! /bin/bash function ayuda() {

cat << DESCRIPCION\_AYUDA SYNOPSIS

$0 NOMBRE\_USUARIO

DESCRIPCION

Devuelve:

SI si NOMBRE\_USUARIO coincide con algún usuario del sistema o NO si NOMBRE\_USUARIO no coincide con ningún usuario del sistema

CÓDIGOS DE RETORNO

1 Si el número de parámetros es distinto de 1 DESCRIPCION\_AYUDA

}

# si número de parámetros distinto 1 if [ $# ­ne 1 ]; then

echo "El número de parámetros debe de igual a 1" ayuda

exit 1

fi

ESTA\_EN\_SISTEMA=`grep ­E ^$1: /etc/passwd` if [ ­z "$ESTA\_EN\_SISTEMA" ]; then

echo "NO"

else fi

echo "SI"

* 1. Modificar el script '**09­hola­usuario.sh**' para que llame a 'usuariosistema' ('**11­hola­usuario.sh**').

#! /bin/bash

# función de ayuda function ayuda() {

cat << DESCRPCION\_AYUDA SYNOPSIS

$0 NOMBRE\_1 [NOMBRE\_2] ... [NOMBRE\_N]

DESCRIPCION

Muestra "Hola NOMBRE\_1, NOMBRE\_2, ... NOMBRE\_N!" por pantalla.

CÓDIGOS DE RETORNO

1. Si el número de parámetros es menor que 1
2. Si el usuario no está en el sistema DESCRPCION\_AYUDA

}

# si número de parámetros <= 0 if [ $# ­le 0 ] ; then

echo "Hay que introducir al menos un parámetro." ayuda

exit 1 fi

MENSAJE="Hola"

PRIMERO=1

# mientras haya parámetros while [ ­n "$1" ]; do

ESTA\_USUARIO=`./usuariosistema $1` if [ "$ESTA\_USUARIO" == "NO" ]; then

echo "El usuario $1 no está en el sistema" ayuda

exit 2

fi

if [ $PRIMERO ­eq 1 ]; then MENSAJE="$MENSAJE $1" PRIMERO=0

else fi

MENSAJE="$MENSAJE, $1"

done

# pasamos al siguiente parámetro shift

# mostramos la salida por pantalla echo ${MENSAJE}"!"

1. **­ Calculadora**
2. Realizar un script llamado '**suma**' que realice la suma de 2 parámetros introducidos (tendrá que poder sumar números decimales, como 2.2 + 3).

#! /bin/bash

# función de ayuda function ayuda() {

cat << DESCRPCION\_AYUDA SYNOPSIS

$0 NUMERO\_1 NUMERO\_2

DESCRIPCIóN

Retorna la suma de NUMERO\_1 y NUMERO\_2

CÓDIGOS DE RETORNO

1. Si el número de parámetros es distinto de 2
2. Si algún parámetro no es un número DESCRPCION\_AYUDA

}

function comprobarQueNoEsNumero() { if [ ­n "$1" \

­a "$1" != "0" \

­a "`echo $1 | awk '{ print $1\*1 }'`" != "$1" ]; then

echo "El parámetro '$1' no es un número" ayuda

exit 2

fi

}

if [ $# ­ne 2 ]; then

echo "El número de parámetros debe de ser igual a 2" ayuda

exit 1

fi

comprobarQueNoEsNumero $1 comprobarQueNoEsNumero $2

echo $1 $2 | awk '{ print $1 + $2 }'

1. Realizar un script llamado '**resta**' que realice la resta de 2 parámetros introducidos (tendrá que poder sumar números decimales, como 2.2 – 3).

#! /bin/bash

# función de ayuda function ayuda() {

cat << DESCRPCION\_AYUDA SYNOPSIS

$0 NUMERO\_1 NUMERO\_2

DESCRIPCIÓN

Retorna la resta de NUMERO\_1 y NUMERO\_2

CÓDIGOS DE RETORNO

* 1. Si el número de parámetros es distinto de 2
  2. Si algún parámetro no es un número DESCRPCION\_AYUDA

}

function comprobarQueNoEsNumero() { if [ ­n "$1" \

­a "$1" != "0" \

­a "`echo $1 | awk '{ print $1\*1 }'`" != "$1" ]; then

echo "El parámetro '$1' no es un número" ayuda

exit 2

fi

}

if [ $# ­ne 2 ]; then

echo "El número de parámetros debe de ser igual a 2" ayuda

exit 1

fi

comprobarQueNoEsNumero $1 comprobarQueNoEsNumero $2

echo $1 $2 | awk '{ print $1 ­ $2 }'

1. Realizar un script llamado '**multiplica**' que multiplique los 2 parámetros introducidos (tendrá que poder multiplicar números decimales, como 2.2 \* 3).

#! /bin/bash

# función de ayuda function ayuda() {

cat << DESCRPCION\_AYUDA SYNOPSIS

$0 NUMERO\_1 NUMERO\_2

DESCRIPCIÓN

Retorna la multiplicación de NUMERO\_1 y NUMERO\_2

CÓDIGOS DE RETORNO

1. Si el número de parámetros es distinto de 2
2. Si algún parámetro no es un número DESCRPCION\_AYUDA

}

function comprobarQueNoEsNumero() { if [ ­n "$1" \

­a "$1" != "0" \

­a "`echo $1 | awk '{ print $1\*1 }'`" != "$1" ]; then

echo "El parámetro '$1' no es un número" ayuda

exit 2

fi

}

if [ $# ­ne 2 ]; then

echo "El número de parámetros debe de ser igual a 2" ayuda

exit 1

fi

comprobarQueNoEsNumero $1 comprobarQueNoEsNumero $2

echo $1 $2 | awk '{ print $1 \* $2 }'

1. Realizar un script llamado '**division**' que realice la división de 2 parámetros introducidos (tendrá que poder sumar números decimales, como 2.2 / 3).

#! /bin/bash

# función de ayuda function ayuda() {

cat << DESCRPCION\_AYUDA SYNOPSIS

$0 NUMERO\_1 NUMERO\_2

DESCRIPCION

Retorna la división de NUMERO\_1 y NUMERO\_2

CÓDIGOS DE RETORNO

1. Si el número de parámetros es distinto de 2
2. Si algún parámetro no es un número DESCRPCION\_AYUDA

}

function comprobarQueNoEsNumero() { if [ ­n "$1" \

­a "$1" != "0" \

­a "`echo $1 | awk '{ print $1\*1 }'`" != "$1" ]; then

echo "El parámetro '$1' no es un número" ayuda

exit 2

fi

}

if [ $# ­ne 2 ]; then

echo "El número de parámetros debe de ser igual a 2" ayuda

exit 1

fi

comprobarQueNoEsNumero $1 comprobarQueNoEsNumero $2

echo $1 $2 | awk '{ print $1 / $2 }'

1. Realizar un script llamado '**calc01.sh**' que realice operaciones básicas entre

2 números llamando a cada uno de los scripts anteriormente creados (suma, resta, multiplicación y división).

#! /bin/bash

# función de ayuda function ayuda() {

cat << DESCRIPCION\_AYUDA SYNOPSIS

$0 NUMERO\_1 OPERACIÓN NUMERO\_2

DESCRIPCIÓN

Retorna el resultado de la OPERACIÓN entre NUMERO\_1 y NUMERO\_2

OPERACIÓN puede tener estos valores:

+ sum mas

­ res menos x mul por

/ div entre

CÓDIGOS DE RETORNO

1. Si el número de parámetros es distinto de 2.
2. Si algún parámetro no es un número.
3. Si la operación introducida es inválida. DESCRIPCION\_AYUDA

}

function comprobarQueNoEsNumero() { if [ ­n "$1" \

­a "$1" != "0" \

­a "`echo $1 | awk '{ print $1\*1 }'`" != "$1" ]; then

echo "El parámetro '$1' no es un número" ayuda

exit 2

fi

}

# si número de parámetros distinto 3 if [ $# ­ne 3 ]; then

echo "El número de parámetros debe de ser igual a 3" ayuda

exit 1

fi

comprobarQueNoEsNumero $1 comprobarQueNoEsNumero $3

case $2 in

+|sum|mas) ./suma $1 $3 ;;

­|res|menos) ./resta $1 $3 ;; x|mul|por) ./multiplica $1 $3 ;;

/|div|entre) ./division $1 $3 ;;

\*) echo "La operación '$2' es inválida." ; ayuda ; exit 3 ;; esac

1. Ídem pero sin llamar a los scripts ('**calc02.sh**'). #! /bin/bash

# función de ayuda function ayuda() {

cat << DESCRIPCION\_AYUDA SYNOPSIS

$0 NUMERO\_1 OPERACIÓN NUMERO\_2

DESCRIPCIÓN

Retorna el resultado de la OPERACIÓN entre NUMERO\_1 y NUMERO\_2

OPERACIÓN puede tener estos valores:

+ sum mas

­ res menos x mul por

/ div entre

CÓDIGOS DE RETORNO

1. Si el número de parámetros es distinto de 2.
2. Si algún parámetro no es un número.
3. Si la operación introducida es inválida. DESCRIPCION\_AYUDA

}

function comprobarQueNoEsNumero() { if [ ­n "$1" \

­a "$1" != "0" \

­a "`echo $1 | awk '{ print $1\*1 }'`" != "$1" ]; then

echo "El parámetro '$1' no es un número" ayuda

exit 2

fi

}

if [ $# ­ne 3 ]; then

echo "El número de parámetros debe de ser igual a 3" ayuda

exit 1

fi

comprobarQueNoEsNumero $1 comprobarQueNoEsNumero $3

case $2 in

+|sum|mas) echo $1 $3 | awk '{ print $1 + $2 }' ;;

­|res|menos) echo $1 $3 | awk '{ print $1 ­ $2 }' ;; x|mul|por) echo $1 $3 | awk '{ print $1 \* $2 }' ;;

/|div|entre) echo $1 $3 | awk '{ print $1 / $2 }' ;;

\*) echo "La operación '$2' es inválida." ; ayuda ; exit 3 ;; esac

1. Realizar un script llamado '**calc03.sh**' que calcule el valor una expresión numérica pasada por parámetro.

#! /bin/bash

# función de ayuda function ayuda() {

cat << DESCRIPCION\_AYUDA SYNOPIS

$0 EXPRESIÓN\_NUMÉRICA DESCRIPCIÓN

Muestra por pantalla el valor de EXPRESIÓN\_NUMÉRICA. CODIGOS DE RETORNO

1. Si no hay ningún error.
2. Si el número de parámetros es distinto de 1.
3. Si hay un error de formato en la expresión introducida.
4. Si hay un error de entra y salida.
5. Si hay un error al ejecutar la expresión introducida. DESCRIPCION\_AYUDA

}

# función de error function error() {

echo "$0: línea $1: Error $3: $2" exit $3

}

# si primer parámetro == '­h' o == '­­help' if [ "$1" == "­h" ­o "$1" == "­­help" ]; then

ayuda exit 0

fi

# si número de parámetros distinto 1 if [ $# ­ne 1 ] ; then

error $LINENO "Hay que introducir 1 y solamente 1 parámetro." 1

fi

# si el parámetro no concuerda con la expresión regular if [ ­z "`echo $1 | grep ­E ^[\\*\/0­9\(\)\+\­]+$`" ]; then

error $LINENO "Error de formato en la expresión introducida." 2

fi

# guardamos la expresión ($1) en el fichero oculto .expresion.awk # dentro de la HOME del usuario

echo "{ print $1 }" > ~/.expresion.awk

# si hay un error en el último comando ejecutado if [ "$?" != "0" ]; then

error $LINENO "Error de entrada y salida." 3

fi

# ejecutamos awk con el fichero oculto .expresion.awk echo "" | awk ­f ~/.expresion.awk 2> ~/.log.awk

# si hay un error en el último comando ejecutado if [ "$?" != "0" ]; then

error $LINENO "Error al ejecutar la expresión introducida." 4

fi

1. Realizar a mano un fichero '**notas.csv**' con los siguientes datos:

|  |  |  |  |
| --- | --- | --- | --- |
| Pepito | 3.1 | 4.4 | 5.7 |
| Fulanito | 4.2 | 6.5 | 8.8 |
| Menganito | 5.3 | 5.6 | 5.0 |

1. Realizar un fichero '**notas.awk**' y su correspondiente interfaz '**notas.sh**' para que al final obtengamos algo parecido a esto:

+­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­+­­­­­­­+­­­­­­+

| NOMBRE EX1 EX2 EX3 | MED | APTO |

+­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­+­­­­­­­+­­­­­­+

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| | Pepito | 3.1 | 4.4 | 5.7 | | 4.4 | | | NO | |
| | Fulanito | 4.2 | 6.5 | 8.8 | | 6.5 | | | SI | |
| | Menganito | 5.3 | 5.6 | 5.0 | | 5.3 | | | SI | |
| +­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­+­­­­­­­+­­­­­­+ | | | | | | |
| | TOTAL | 4.2 | 5.5 | 6.5 | | 5.4 | | | 2 | |
| +­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­+­­­­­­­+­­­­­­+ | | | | | | |

**notas.awk:**

# esto se ejecutará solo una vez al principio BEGIN {

print "+­­­­­­­­­­­­­­­­­­­­­­­­­­­+­­­­­+­­­­­­+" print "| NOMBRE EX1 EX2 EX3 | MED | APTO |" print "+­­­­­­­­­­­­­­­­­­­­­­­­­­­+­­­­­+­­­­­­+"

}

# esto se ejecutará para cada una de las líneas del fichero

{

suma2+=$2 suma3+=$3 suma4+=$4

mediaFila=($2+$3+$4)/3

apto="NO"

if ( mediaFila >= 5 ) { apto="SI"

aptos++

}

print "| "$0" | "mediaFila" | "apto" |"

}

# esto se ejecutará solo una vez al final END {

media2=suma2/3 media3=suma3/3 media4=suma4/3

media=(media2+media3+media4)/3

print "+­­­­­­­­­­­­­­­­­­­­­­­­­­­+­­­­­+­­­­­­+"

print "| MEDIAS "media2" "media3" "media4" | "media" | "aptos" |" print "+­­­­­­­­­­­­­­­­­­­­­­­­­­­+­­­­­+­­­­­­+"

}

**notas.sh:**

awk ­f notas.awk notas.csv

1. **­ Banco**
2. Realizar un script llamado '**banco**' para añadir, buscar y listar movimientos bancarios, y calcular el saldo de la cuenta.

#! /bin/bash BANCO\_FILE=~/.banco.txt function help() {

cat << DESCRIPCION\_AYUDA SYNOPSIS

$0 OPCION [PARAMETRO\_2] ... [PARAMETRO\_N]

DESCRIPCIÓN

Añade, busca, lista y opera con movimientos bancarios.

OPCIONES

­h ­­help Muestra una ayuda.

­a ­­add FECHA CONCEPTO CANTIDAD Añade un movimiento bancario.

­s ­­search PATRÓN Busca un movimiento bancario.

­l ­­list Lista los movimientos bancarios ordenados por fecha.

­t ­­total Calcula el saldo total de la cuenta.

CÓDIGOS DE RETORNO

* 1. Si no hay ningún error.
  2. SI la opción introducida no es válida.
  3. si un argumento númerico no es un número.
  4. Si el número de parámetros es erróneo.
  5. si un argumento de tipo fecha no es una fecha.
  6. Si hay un error de entrada/salida en $BANCO\_FILE. DESCRIPCION\_AYUDA

}

function exitWithError() {

LINEA\_ERROR=$1 MENSAJE\_ERROR=$2 CODIGO\_ERROR=$3

echo "$0: línea $LINEA\_ERRO: Error $CODIGO\_ERROR: $MENSAJE\_ERROR" exit $CODIGO\_ERROR

}

function testDateExists() {

DATE=$1

# si ya existe un movimiento bancario para la misma fecha if [ ­n "`grep ­E ^$DATE $BANCO\_FILE`" ]; then

exitWithError $LINENO "Ya existe un movimiento bancario para la fecha '$DATE'." 12

fi

}

function testIsDate() { DATE=$1

EXPRESION\_CONCUERDA=`echo $DATE | grep ­E \

^20[0­9]{2}­[01][0­9]­[0­3][0­9]$`

if [ ­z "$EXPRESION\_CONCUERDA" ]; then

exitWithError $LINENO "'$DATE' no es una fecha" 4

fi

}

function testIsNubmer() {

NUMBER=$1

if [ ­n "$NUMBER" \

­a $NUMBER != "0" \

­a "`echo $NUMBER | awk '{ print $1\*1 }'`" != "$NUMBER" ]; then

exitWithError $LINENO "'$NUMBER' no es un número" 2

fi

}

function testParameterNumer() {

PARAMETER\_NUMBER=$1

shift

if [ $# ­ne $PARAMETER\_NUMBER ]; then exitWithError $LINENO \

"Número de parámetros " \ "obligatorio: $PARAMETER\_NUMBER" 3

fi

}

function add() {

testParameterNumer 3 $@ FECHA=$1

CONCEPTO=$2 CANTIDAD=$3

testIsDate $FECHA testDateExists $FECHA testIsNubmer $CANTIDAD

echo "$FECHA $CONCEPTO $CANTIDAD" >> $BANCO\_FILE

}

function search() { testParameterNumer 1 $@ PATRON=$1

grep $PATRON $BANCO\_FILE

}

# lista movimientos ordenados por fecha function list() {

sort ­nk 1 $BANCO\_FILE

}

function total() {

awk '{s+=$3} END {print "Total="s}' $BANCO\_FILE

}

function createFileIfNotExists() {

touch $BANCO\_FILE

if [ "$?" != "0" ]; then exitWithError $LINENO \ "Error de entrada/salida en" \ "el fichero $BANCO\_FILE" 5

fi

}

function menu() {

case $1 in

­h|­­help) help;;

­a|­­add) shift; add $@;;

­s|­­search) shift; search $@;;

­l|­­list) list;;

­t|­­total) total;;

\*) exitWithError $LINENO "Opción '$1' inválida." 1

esac

}

function init() {

createFileIfNotExists menu $@

}

init $@

1. Realizar un script llamado '**banco­menu.sh**' que sirva de interfaz del anterior.

#! /bin/bash

# script que añade, busca y opera con movimientos bancarios. # variables globales

BANCO\_SCRIPT=./banco

# función de ayuda function ayuda() {

cat << DESCRIPCION\_AYUDA SYNOPIS

$0 [OPCIONES] DESCRIPCIÓN

Añade y busca y opera con movimientos bancarios.

OPCIONES

­h ­­help Muesta esta ayuda.

CODIGOS DE RETORNO

0 Si no hay ningún error. DESCRIPCION\_AYUDA

}

# función menu function menu() {

cat << DESCRIPCION\_MENU

+­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­+

| MENU DEL BANCO |

+­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­+

| a ­ Añadir un movimiento bancario. |

| s ­ Buscar un movimiento bancario. |

| l ­ Listar todos los movimientos bancarios ordenados por fecha. |

| t ­ Calcular el saldo total de la cuenta. |

| e ­ Salir del programa. |

+­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­+

DESCRIPCION\_MENU

}

# función de error

# $1 línea de error # $2 mensaje de error function error() {

echo "$0: línea $1: $2"

}

# función para añadir un movimiento bancario function add() {

echo "AÑADIR UN MOVIMIENTO BANCARIO"

read ­p "Introduce el fecha: " FECHA

read ­p "Introduce el concepto: " CONCEPTO read ­p "Introduce la cantidad: " CANTIDAD

$BANCO\_SCRIPT ­­add $FECHA $CONCEPTO $CANTIDAD

elegir\_menu

}

# función para buscar un movimiento bancario function search() {

echo "BUSCAR MOVIMIENTO BANCARIO"

read ­p "Introduce un patrón de búsqueda: " PATRON

$BANCO\_SCRIPT ­­search $PATRON elegir\_menu

}

# función para listar movimientos bancarios ordenados por mes y día function list() {

echo "LISTAR ORDENADO POR FECHA"

$BANCO\_SCRIPT ­­list elegir\_menu

}

# función para mostrar el saldo total de la cuenta function total() {

echo "SALDO TOTAL DE LA CUENTA"

$BANCO\_SCRIPT ­­total elegir\_menu

}

# función para salir del programa function salir() {

exit 0

}

# función opción invalida function opcion\_invalida() {

echo "Opción '$1' inválida." elegir\_menu

}

# función elegir\_menu function elegir\_menu() {

menu

read ­p "Elige una opción: " OPCION clear

case $OPCION in

1. add ;;
2. search ;;

l) list ;;

c) total ;;

s) salir ;;

\*) opcion\_invalida $OPCION;; esac

}

# si primer parámetro == '­h' o == '­­help' if [ "$1" == "­h" ­o "$1" == "­­help" ]; then

ayuda exit 0

fi clear

elegir\_menu

1. Realizar un script llamado '**banco­flags.sh**' para poder usar el script '**banco**' mediante CLI.

#! /bin/bash

# script que añade, busca y opera con movimientos bancarios. # variables globales

BANCO\_SCRIPT=./banco

# función de ayuda function ayuda() {

cat << DESCRIPCION\_AYUDA SYNOPIS

$0 [OPCIÓN] [PARÁMETROS]

DESCRIPCIÓN

Añade, busca, lista y opera con movimientos bancarios.

OPCIONES

­h ­­help Muestra una ayuda.

­a ­­add FECHA CONCEPTO CANTIDAD Añade un movimiento bancario.

­s ­­search PATRÓN Busca un movimiento bancario.

­l ­­list Lista los movimientos bancarios ordenados por fecha.

­t ­­total Calcula el saldo total de la cuenta.

CÓDIGOS DE RETORNO

1. Si no hay ningún error.
2. SI la opción introducida no es válida.
3. si un argumento númerico no es un número.
4. Si el número de parámetros es erróneo.
5. si un argumento de tipo fecha no es una fecha.
6. Si hay un error de entrada/salida en $BANCO\_FILE. DESCRIPCION\_AYUDA

}

# función para añadir un movimiento bancario function add() {

echo "AÑADIR UN MOVIMIENTO BANCARIO"

$BANCO\_SCRIPT ­­add $@

echo "­­­­­­­­­­­­­­­­­­­­­­­­­­­­­"

}

# función para búscar un movimiento bancario function search() {

echo "BUSCAR MOVIMIENTO BANCARIO ($1)"

$BANCO\_SCRIPT ­­search $1

echo "­­­­­­­­­­­­­­­­­­­­­­­­­­­­­"

}

# función para listar movimientos bancarios ordenados por mes y día function list() {

echo "LISTAR ORDENADO POR FECHA"

$BANCO\_SCRIPT ­­list

echo "­­­­­­­­­­­­­­­­­­­­­­­­­­­­­"

}

# función para mostrar el saldo total de la cuenta function total() {

echo "SALDO TOTAL DE LA CUENTA"

$BANCO\_SCRIPT ­­total

echo "­­­­­­­­­­­­­­­­­­­­­­­­­­­­­"

}

# función opción invalida function opcion\_invalida() {

echo "Opción '$1' inválida." exit 6

}

while getopts "ha:s:lt" option ; do case "$option" in

h) ayuda ;;

a) add $OPTARG ;;

s) search $OPTARG ;;

l) list ;;

t) total ;;

\*) opcion\_invalida $option ;; esac

done

1. **­ Demonios**
2. Realizar un demonio llamado '**alerta**' que escriba la fecha cada X segundos en un log llamado '**~/alerta.log**'.

#! /bin/bash

# función de ayuda function ayuda() {

cat << DESCRIPCION\_AYUDA SYNOPIS

$0 [SEGUNDOS] DESCRIPCION

Escribe la fecha cada X segundos en el log '~/alerta.log' CODIGOS DE RETORNO

0 Si no hay ningún error DESCRIPCION\_AYUDA

}

# si primer parámetro == '­h' o == '­­help' if [ "$1" == "­h" ­o "$1" == "­­help" ]; then

ayuda exit 0

fi

function main() {

DEFAULT=2

# comprobar que SEGUNDOS es un número

if [ "$SEGUNDOS" != "0" ­a "`echo $SEGUNDOS | awk '{ print $1 \* 1 }'`" != "$SEGUNDOS" ]; then

echo "El parámetro '$1' no es un número. Se cogerá el valor por defecto ($DEFAULT)"

SEGUNDOS=$DEFAULT

fi

# reinicio alerta.log echo "" > ~/alerta.log

while [ true ]; do

date +%d/%m/%Y" "%H:%M:%S >> ~/alerta.log sleep $SEGUNDOS

done

}

echo $$ main $1

1. Realizar las interfaces del demonio 'alerta' con las opciones básicas: start, stop, restart y status ('**servicio­alerta.sh**').

#! /bin/bash

# función de ayuda function ayuda() {

cat << DESCRIPCION\_AYUDA SYNOPIS

$0 start|stop|restart|status

DESCRIPCIÓN

Muestra que arraca, para, relanza y nos muestra el estado de 'alerta'.

CÓDIGOS DE RETORNO

0 Si no hay ningún error. DESCRIPCION\_AYUDA

}

DAEMON=alerta PIDFILE=/tmp/$DAEMON.pid

# función que arranca 'alerta' function do\_start() {

# si exite el fichero

if [ ­e $PIDFILE ]; then

echo "El proceso ya se está ejecutando." exit 0;

fi

./$DAEMON &

echo $! > $PIDFILE echo "Ejecutandose..."

}

# función que para 'alerta' function do\_stop() {

# si exite el fichero

if [ ­e $PIDFILE ]; then kill ­9 `cat $PIDFILE` rm $PIDFILE

fi

echo "Parado."

}

# función que para y arrance 'alerta' function do\_restart() {

do\_stop do\_start

}

# función que muestra el estado de 'alerta' function do\_status() {

# si exite el fichero

if [ ­e $PIDFILE ]; then echo "Ejecutandose..."

else

echo "Parado."

fi

}

# si primer parámetro == '­h' o == '­­help' if [ "$1" == "­h" ­o "$1" == "­­help" ]; then

ayuda exit 0

fi

case $1 in

start)

do\_start ;; stop)

do\_stop ;; restart)

do\_restart ;; status)

do\_status ;;

\*)

esac

1. **­ Copias**

echo "Parámetro '$1' incorrecto." ;;

1. Realizar un script llamado '**copia­total**' que empaquete y comprima el contenido de la carpeta '**~/carpeta\_a\_copiar**' en un fichero llamado '**total­ aaaa.mm.dd­HH.MM.SS.tar.zip**' en la carpeta '**~/copia\_seguridad**'.

#! /bin/bash

############################################################ # INICIO VARIABLES ############################################################

# variable con la fecha en el formato indicado (aaaa.mm.dd­HH.MM.SS)

FECHA=`date +%Y.%m.%d­%H.%M.%S`

# variable con la ruta de los ficheros RUTA\_FICHEROS=~/copia\_seguridad

# variable con el fichero con la fecha de la última copia total FICHERO\_ULTIMA\_COPIA\_TOTAL=$RUTA\_FICHEROS/fecha­ultima­copia­total.txt

# variable con el fichero comprimido FICHERO\_COMPRIMIDO=$RUTA\_FICHEROS/total­$FECHA.tar.zip

# variable con el directorio que queremos copiar y comprimir DIRECTORIO\_A\_COPIAR=~/directorio\_a\_copiar

############################################################ # FIN VARIABLES ############################################################

# si no exixte el directorio a copiar mostramos un error y paramos la ejecución

if [ ! ­d $DIRECTORIO\_A\_COPIAR ]; then

echo "No exixte el directorio a copiar." exit 1

fi

# si no exixte el directorio de los ficheros lo creamos if [ ! ­d $RUTA\_FICHEROS ]; then

mkdirs $RUTA\_FICHEROS

fi

# guardar la fecha de la última copia total en FICHERO\_ULTIMA\_COPIA\_TOTAL echo $FECHA > $FICHERO\_ULTIMA\_COPIA\_TOTAL

# empaquetamos y comprimimos el DIRECTORIO\_A\_COPIAR en FICHERO\_COMPRIMIDO (mediante zip)

zip ­r $FICHERO\_COMPRIMIDO $DIRECTORIO\_A\_COPIAR

1. Realizar un script llamado '**copia­diferencial**' que empaquete y comprima los

ficheros de la carpeta '**~/carpeta\_a\_copiar**' modificados desde la última copia total (si no existe copia total no hacer nada) en un fichero llamado '**diferencial­aaaa.mm.dd­HH.MM.SS.tar.zip**' en la carpeta '**~/copia\_seguridad**'.

#! /bin/bash

# script que empaqueta y comprime los ficheros modificados

# desde la última copia­incremental si existe y es la copia más reciente

# sino desde la última copia­diferencial si existe y es la copia más reciente # sino desde la última copia­total si existe

# en un fichero llamado diferencial­aaaa.mm.dd­HH.MM.SS.tar.zip # en la carpeta /root/copia\_seguridad

############################################################ # INICIO VARIABLES ############################################################

# variable con la fecha en el formato indicado (aaaa.mm.dd­HH.MM.SS)

FECHA=`date +%Y.%m.%d­%H.%M.%S`

# variable con la ruta de los ficheros RUTA\_FICHEROS=~/copia\_seguridad

# variable con el fichero con la fecha de la última copia total FICHERO\_ULTIMA\_COPIA\_TOTAL=$RUTA\_FICHEROS/fecha­ultima­copia­total.txt

# variable con el fichero con la fecha de la última copia diferencial FICHERO\_ULTIMA\_COPIA\_DIFERENCIAL=$RUTA\_FICHEROS/fecha­ultima­copia­ diferencial.txt

# variable con el fichero comprimido FICHERO\_COMPRIMIDO=$RUTA\_FICHEROS/diferencial­$FECHA.tar.zip

# variable con eldirectorio que queremos copiar y comprimir DIRECTORIO\_A\_COPIAR=~/directorio\_a\_copiar

############################################################ # FIN VARIABLES ############################################################

# si no exixte el directorio a copiar mostramos un error y paramos la ejecución

if [ ! ­d $DIRECTORIO\_A\_COPIAR ]; then

echo "No exixte el directorio a copiar." exit 1

fi

# si no existe el FICHERO\_ULTIMA\_COPIA\_TOTAL mostramos un error y paramos la ejecución

if [ ! ­e $FICHERO\_ULTIMA\_COPIA\_TOTAL ]; then

echo "No hay última copia total." exit 1

fi

# si no exixte el directorio de los ficheros lo creamos if [ ! ­d $RUTA\_FICHEROS ]; then

mkdirs $RUTA\_FICHEROS

fi

# guardar la fecha de la última copia diferencial en FICHERO\_ULTIMA\_COPIA\_DIFERENCIAL

echo $FECHA > $FICHERO\_ULTIMA\_COPIA\_DIFERENCIAL

# empaquetamos y comprimimos los ficheros modificados desde la última copia total

find $DIRECTORIO\_A\_COPIAR/\* ­newer $FICHERO\_ULTIMA\_COPIA\_TOTAL | zip ­@

$FICHERO\_COMPRIMIDO

1. Realizar un script llamado '**copia­incremental**' que empaquete y comprima los ficheros de la carpeta '**~/carpeta\_a\_copiar**' modificados desde la última **copia incremental** (si no existe copia incremental, desde la última copia total, y si no existe copia total no hacer nada) en un fichero llamado '**incremental­ aaaa.mm.dd­HH.MM.SS.tar.zip**' en la carpeta '**~/copia\_seguridad**'.

#! /bin/bash

# script que empaqueta y comprime los ficheros modificados desde la última # copia ya sea incremental, diferencial o total de la carpeta /root/logs/ # en un fichero llamado diferencial­aaaa.mm.dd­HH.MM.SS.tar.zip

# en la carpeta /root/copia\_seguridad

############################################################ # INICIO VARIABLES ############################################################

# variable con la fecha en el formato indicado (aaaa.mm.dd­HH.MM.SS)

FECHA=`date +%Y.%m.%d­%H.%M.%S`

# variable con la ruta de los ficheros RUTA\_FICHEROS=~/copia\_seguridad

# variable con el fichero con la fecha de la última copia total FICHERO\_ULTIMA\_COPIA\_TOTAL=$RUTA\_FICHEROS/fecha­ultima­copia­total.txt

# variable con el fichero con la fecha de la última copia diferencial FICHERO\_ULTIMA\_COPIA\_DIFERENCIAL=$RUTA\_FICHEROS/fecha­ultima­copia­ diferencial.txt

# variable con el fichero con la fecha de la última copia incremental FICHERO\_ULTIMA\_COPIA\_INCREMENTAL=$RUTA\_FICHEROS/fecha­ultima­copia­ incremental.txt

# variable con el fichero comprimido FICHERO\_COMPRIMIDO=$RUTA\_FICHEROS/diferencial­$FECHA.tar.zip

# variable con eldirectorio que queremos copiar y comprimir DIRECTORIO\_A\_COPIAR=~/directorio\_a\_copiar

############################################################ # FIN VARIABLES ############################################################

############################################################ # INICIO FUNCIONES ############################################################

# función que empaqueta y comprime los ficheros modificados # desde la última copía incremental

function copia\_desde\_ultima\_incremental() {

# empaqueta y comprime los ficheros modificados desde la última copia incremental

find $DIRECTORIO\_A\_COPIAR/\*.txt ­newer $FICHERO\_ULTIMA\_INCREMENTAL | zip

­@ $FICHERO\_COMPRIMIDO

# guardar la fecha de la última copia incremental en FICHERO\_ULTIMA\_COPIA\_INCREMENTAL

echo $FECHA > $FICHERO\_ULTIMA\_COPIA\_INCREMENTAL

# salimos exit 0

}

# función que empaqueta y comprime los ficheros modificados # desde la última copía diferencial

function copia\_desde\_ultima\_diferencial() {

# guardar la fecha de la última copia incremental en FICHERO\_ULTIMA\_COPIA\_INCREMENTAL

echo $FECHA > $FICHERO\_ULTIMA\_COPIA\_INCREMENTAL

# empaqueta y comprime los ficheros modificados desde la última copia diferencial

find $DIRECTORIO\_A\_COPIAR/\*.txt ­newer $FICHERO\_ULTIMA\_DIFERENCIAL | zip

­@ $FICHERO\_COMPRIMIDO

# salimos exit 0

}

# función que empaqueta y comprime los ficheros modificados # desde la última copía total

function copia\_desde\_ultima\_total() {

# guardar la fecha de la última copia incremental en FICHERO\_ULTIMA\_COPIA\_INCREMENTAL

echo $FECHA > $FICHERO\_ULTIMA\_COPIA\_INCREMENTAL

# empaqueta y comprime los ficheros modificados desde la última copia

total

find $DIRECTORIO\_A\_COPIAR/\*.txt ­newer $FICHERO\_ULTIMA\_COPIA | zip ­@

$FICHERO\_COMPRIMIDO

# salimos exit 0

}

############################################################ # FIN FUNCIONES ############################################################

# si no exixte el directorio a copiar mostramos un error y paramos la ejecución

if [ ! ­d $DIRECTORIO\_A\_COPIAR ]; then

echo "No exixte el directorio a copiar." exit 1

fi

# si no existe el FICHERO\_ULTIMA\_COPIA\_TOTAL mostramos un error y paramos la ejecución

if [ ! ­e $FICHERO\_ULTIMA\_COPIA\_TOTAL ]; then

echo "No hay última copia total." exit 1

fi

# si no existe el FICHERO\_ULTIMA\_COPIA\_INCREMENTAL if [ ! ­e $FICHERO\_ULTIMA\_COPIA\_INCREMENTAL ]; then

copia\_desde\_ultima\_total

fi

1. Modificar el fichero '**miCrontab**' para que imprima la fecha en el fichero '**~/ultimo­crontab.txt**' cada minuto, y ejecutarlo con crontab.

######################################################

# minuto (0­59),

# |

# |

# |

# |

# |

# |

hora (0­23),

| día del mes (1­31),

| |

| |

| |

| |

mes (1­12),

# # # #

| día de la semana (0­6 donde 0=Domingo) #

| |

| |

comandos

|

# #

######################################################

\* \* \* \* \* date > ~/ultimo­crontab.txt

1. **­ Varios**
2. Crear un script llamado '**array.sh**' que declare un array, lo rellene con datos y luego itere sobre el mismo para mostrar los datos.

declare ­a ARRAY;

ARRAY=("cero" "uno" [3]="tres") ARRAY[2]="dos"

LENGTH=${#ARRAY[\*]}

for (( i=0; i<LENGTH; i++ )); do echo $i=${ARRAY[i]}

done

1. Realizar a mano un fichero '**roles.csv**' con los siguientes datos: Pepito:Jefe,Sistemas

Fulanito:Jefe,Desarrollo Menganito:Operario,Sistemas,Desarrollo

1. Realizar un script '**roles­sin­awk.sh**', que, sin utilizar awk, al final obtengamos algo parecido a esto:

Desarrollo

­> Fulanito Menganito Operario

­> Menganito Sistemas

­> Pepito Menganito Jefe

­> Pepito Fulanito ROLES\_FILE=./roles.csv

ROLES=`cut ­d : ­f 2 $ROLES\_FILE | sed 's/,/\n/g' | sort | uniq`

for ROL in $ROLES; do echo $ROL

NAMES=`grep ­E $ROL $ROLES\_FILE | cut ­d : ­f 1`

echo " ­> "$NAMES done

1. Realizar un fichero '**roles.awk**' y su correspondiente interfaz '**roles­con­ awk.sh**' para que al final obtengamos lo mismo que el ejercicio anterior.

**roles.awk:**

# esto se ejecutará solo una vez al principio BEGIN {

FS = ",|:"

}

# esto se ejecutará para cada una de las líneas del fichero

{

nombre=$1

for (N=2; N<=NF; N++) {

rol=$N

roles[rol]=""roles[rol]" "nombre

}

}

# esto se ejecutará solo una vez al final END {

for ( rol in roles) { print rol

print " ­>" roles[rol]

}

}

**roles­con­awk.sh:**

awk ­f roles.awk roles.csv

1. Realizar un script llamado '**ordena'** que liste el contenido del directorio actual ordenado por tamaño del archivo de menor a mayor. El listado sólo mostrará el nombre de los archivos y el número de línea correspondiente. En el caso de que se introduzca algún parámetro se mostrará el siguiente mensaje de error: “No se permiten parámetros.” y retornará un código de retorno igual a 1.

#! /bin/bash

# si el número de parámetros es distinto de 0 if [ "$#" != "0" ]; then

# muestra un mensaje de error y sale echo "No se permiten parámetros." exit 1

fi

# muestra el listado ordenado de menor a mayor por el tamaño # sacando sólo el nombre del archivo y el número de línea ls ­l | sort ­nk 5 | awk '{ print $8 }' | nl

1. Realizar un script llamado '**jaula'** que cree, sólo si no existe, el directorio

**. jaula** en la $HOME del usuario y mueva los ficheros pasados por parámetro a dicho directorio. En el caso de que no se le pase ningún parámetro se mostrará el siguiente mensaje de error: “Hay que introducir al menos un parámetro.” y retornará un código de retorno igual a 1. En el caso de que algún fichero introducido por parámetro no exista se mostrará el siguiente

mensaje de error: “El fichero '$FICHERO' no existe.” y retornará un código de retorno igual a 2. Si el fichero **. jaula** existe en la $HOME del usuario pero no es un directorio mostrará el siguiente mensaje de error: “El fichero

'$HOME/.jaula' no es un directorio.” y retornará un código de retorno igual a 3.

#! /bin/bash

# función que verifica que el número de parámetros es correcto function verificarNumeroParametros() {

# si el número de parámetros es igual a 0 if [ "$#" == "0" ]; then

# muestra un mensaje de error y sale

echo "Hay que introducir al menos un parámetro." exit 1

fi

}

# funcion que crea la jaula si no existe function crearJaula() {

JAULA=~/.jaula

if [ ­e $JAULA ]; then

# si la jaula existe y no es un directorio if [ ! ­d $JAULA ]; then

# muestra un mensaje de error y sale

echo "El fichero '~/.jaula' no es un directorio." exit 3

fi

else

# creamos la jaula mkdir $JAULA

fi

}

# función que mueve a la jaula los ficheros pasados por parámetro function moverFicheros() {

# ejecutar mientras haya parámetros while [ "x$1" != "x" ]; do

FICHERO=$1

# si el fichero pasado por parámeto no existe if [ ! ­e $FICHERO ]; then

# muestra un mensaje de error y sale echo "El fichero '$FICHERO' no existe." exit 2

fi

# mueve el fichero a la jaula mv $FICHERO $JAULA

shift # pasamos al siguiente parámetro

done

}

verificarNumeroParametros $@ crearJaula

moverFicheros $@

1. Realizar un script llamado '**calendario'** al que si pasamos el parámetro **­c** o el parámetro **­­corta** mostrará la fecha de hoy con el formato “$DIA/$MES/$AÑO” y si le pasamos el parámetro **­l** o **­­larga** mostrará la fecha de hoy con el formato “Hoy es el día '$DIA' del mes '$MES' del año '$AÑO'.”. En el caso de

que no se introduzca ningún parámetro se mostrará el calendario del mes actual. En el caso de que el número de parámetros introducidos sea distinto de

1 se mostrará el siguiente mensaje de error: “Sólo se admite un parámetro.” y retornará un código de retorno igual a 1. Si pasamos otra cosa que no sea **­c**,

**­­corta, ­l** o **­­larga** mostrará el siguiente mensaje de error: “Opción incorrecta.” y retornará un código de retorno igual a 2.

#! /bin/bash

# si el número de parámetros es igual a 0 if [ "$#" == "0" ]; then

# muestra el calendario del mes actual y sale cal

exit 0

fi

# si el número de parámetros es distinto de 1 if [ "$#" != "1" ]; then

# muestra un mensaje de error y sale echo "Sólo se admite un parámetro." exit 1

fi

# dependiendo del parámetro introducido case $1 in

­c|­­corta) date +"%d/%m/%Y" ;;

­l|­­larga) date +"Hoy es el día '%d' del mes '%m' del año '%Y'." ;;

\*) echo "Opción incorrecta." ; exit 2 ;; esac

# si todo ha ido bien sale exit 0

1. Realizar un script llamado '**elevado'** que calcule “a^b”, osea “a elevado a b”, donde “a” será el primer parámetro y “b” el segundo parámetro. En el caso de que el número de parámetros introducidos sea menor que 2 se mostrará el siguiente mensaje de error: “Para ejecutar este script se necesitan 2 números.” y retornará un código de retorno igual a 2.

#! /bin/bash

# si el número de parámetros es distinto de 2 if [ $# ­ne 2 ]; then

echo "Para ejecutar este script se necesitan 2 números." exit 2

fi

#inicializamos variables ELEVADO=1

# para cada parámetro introducido

for ((CONTADOR=0; CONTADOR<$2; CONTADOR++)); do

ELEVADO=`echo $ELEVADO $1 | awk '{ print $1\*$2 }'` done

echo $ELEVADO

1. Realizar un script llamado '**citas**' en el que se puedan utilizar las siguientes opciones:

*­h ­­help Para mostrar un texto de ayuda.*

*­a ­­add Para añadir una cita con HORA\_INI, HORA\_FIN, y NOMBRE\_PACIENTE.*

*­s ­­search Para buscar los pacientes que contengan PATRÓN.*

*­i ­­init Para buscar las citas que empiecen a HORA\_INICIO.*

*­e ­­end Para buscar las citas que terminen a HORA\_FINAL.*

*­n ­­name Para listar todas las citas ordenadas por NOMBRE\_PACIENTE.*

*­o ­­hour Para listar todas las citas ordenadas por HORA\_INICIO.*

* *Para cada una de las opciones se comprobará que se introducen el número de parámetros correctos y con el formato correcto.*
* *HORA\_INCIO y HORA\_FINAL serán números enteros comprendidos entre 00 y 23.*
* *Al introducir una cita nueva se comprobará que no se solape con otra ya introducida.*
* *Se comprobará también que no se repita ningún nombre de paciente.*

#! /bin/bash

# script que gestiona las citas de una consulta.

# Para cada una de las opciones se comprobará que se introducen el número de parámetros correctos con el formato correcto.

# HORA\_INCIO y HORA\_FINAL serán numeros enteros comprendidos entre 00 y 23. # Al introducir una cita nueva se comprobará que no se solape con otra ya introducida.

# Se comprobará también que no se repita ningún nombre de paciente.

# variables globales CITAS\_FILE=~/citas.txt

# función de ayuda function ayuda() {

cat << DESCRIPCION\_AYUDA SYNOPIS

$0 [OPCIONES] [HORA\_INICIAL] [HORA\_FINAL] [NOMBRE\_PACIENTE]

DESCRIPCIÓN

Añade y busca citas de una consulta.

OPCIONES

­h ­­help Para mostrar un texto de ayuda.

­a ­­add Para añadir una cita con HORA\_INICIAL, HORA\_FINAL, y NOMBRE\_PACIENTE.

­s ­­search Para buscar los pacientes que contengan PATRÓN.

­i ­­init Para buscar las citas que empiecen a HORA\_INICIAL.

­e ­­end Para buscar las citas que terminen a HORA\_FINAL.

­n ­­name Para listar todas las citas ordenadas por NOMBRE\_PACIENTE.

­o ­­hour Para listar todas las citas ordenadas por HORA\_INICIAL.

CÓDIGOS DE RETORNO

1. Si no hay ningún error.
2. Si el número de parámetros es incorrecto.
3. Si el formato de los parámetros es incorrecto.
4. Si al añadir una cita se solapa con otra ya introducida.
5. Si al añadir una cita ya existe NOMBRE\_PACIENTE.
6. Si se introduce una opción inválida.
7. Si ocurre otro error no mencionado. DESCRIPCION\_AYUDA

}

# función de error

# $1 línea de error # $2 mensaje de error

# $3 código de retorno function error() {

# muetra un mensaje de error

echo "$0: Línea $1: Error $3: $2"

# termina la ejecución del script con el codigo de retorno indicado exit $3

}

# función que comprueba el número de parámetros introducido # $1 linea error

# $2 número de parámetros esperados # $@ parámetros

function numero\_parametros() {

# líenea error LINEA\_ERROR=$1

# eliminamos el primer parámetro shift

# número de parámetros esperados NUMERO\_PARAMETROS\_ESPERADOS=$1

# volvemos a eliminar el primer parámetro shift

# número de parámetros que nos queda NUMERO\_PARAMETROS\_REALES=$#

then

# si el número de parámetros introducido es distinto del esperado

if [ "$NUMERO\_PARAMETROS\_REALES" != "$NUMERO\_PARAMETROS\_ESPERADOS" ];

error $LINEA\_ERROR "Número de parámetros '$NUMERO\_PARAMETROS\_REALES' distinto de '$NUMERO\_PARAMETROS\_ESPERADOS'." 1

fi

}

# función que comprueba que el formato de la hora es correcto # $1 línea de error

# $2 hora

function formato\_hora() {

# líenea error LINEA\_ERROR=$1

# recogemos la hora HORA=$2

# si la hora introducida no es un número de 2 cifras if [ ­z "`echo $HORA | grep ­E ^[0­9]{2}$`" ]; then

error $LINEA\_ERROR "'$HORA' no es un número de 2 cifras." 2

fi

# si la hora no está comprendida entre 00 y 23 if [ $HORA ­gt 23 ]; then

error $LINEA\_ERROR "'$HORA' no está comprendida entre 00 y 23" 2

fi

}

# función que comprueba que la hora inicial no se solapa con alguna cita ya introducida

# $1 línea de error

# $2 hora inicial

function solape\_hora\_inicial() {

# líenea error LINEA\_ERROR=$1

# recogemos la hora inicial HORA\_INICIAL=$2

# fichero con el programa awk FICHERO\_PROGRAMA\_AWK=~/awk­program.txt

# sentencia awk para buscar si ya existe una cita que se solape

echo "{ if ( $HORA\_INICIAL >= \$1 && $HORA\_INICIAL < \$2 ) print \$0 }"

> $FICHERO\_PROGRAMA\_AWK

# ejecutamos el comando y guardamos el resultado en una variable CITA\_SOLAPADA=`more $CITAS\_FILE | awk ­f $FICHERO\_PROGRAMA\_AWK`

# si existe alguna cita que se solape if [ ­n "$CITA\_SOLAPADA" ]; then

error $LINEA\_ERROR "La hora inicial '$HORA\_INICIAL' se solapa con la cita '$CITA\_SOLAPADA'." 3

fi

}

# función que comprueba que la hora final no se solapa con alguna cita ya introducida

# $1 línea de error

# $2 hora final

function solape\_hora\_final() {

# líenea error LINEA\_ERROR=$1

# recogemos la hora final HORA\_FINAL=$2

# fichero con el programa awk FICHERO\_PROGRAMA\_AWK=~/awk­program.txt

# sentencia awk para buscar si ya existe una cita que se solape

echo "{ if ( $HORA\_FINAL > \$1 && $HORA\_FINAL <= \$2 ) print \$0 }" >

$FICHERO\_PROGRAMA\_AWK

# ejecutamos el comando y guardamos el resultado en una variable CITA\_SOLAPADA=`more $CITAS\_FILE | awk ­f $FICHERO\_PROGRAMA\_AWK`

# si existe alguna cita que se solape if [ ­n "$CITA\_SOLAPADA" ]; then

error $LINEA\_ERROR "La hora final '$HORA\_FINAL' se solapa con la cita '$CITA\_SOLAPADA'." 3

fi

}

# función que comprueba que no exista ya un nombre de paciente # $1 línea de error

# $2 nombre del paciente a comprobar function nombre\_paciente() {

# líenea error LINEA\_ERROR=$1

# recogemos el nombre del paciente NOMBRE\_PACIENTE=$2

# si el nombre ya está en el fichero de citas

if [ ­n "`grep $NOMBRE\_PACIENTE $CITAS\_FILE`" ]; then

error $LINEA\_ERROR "Nombre '$NOMBRE\_PACIENTE' repetido." 4

fi

}

# función que comprueba si ha habido algún error inexperado # $1 línea de error

function error\_inexperado() {

# líenea error LINEA\_ERROR=$1

# si ocurre algún error inexperado if [ "$?" != "0" ]; then

error $LINEA\_ERROR "Error inexperado." 6

fi

}

# función para añadir una cita # $1 hora incio cita

# $2 hora final cita

# $3 nombre del paciente function add() {

# comprobamos que el número de parámetros sea igual a 3 numero\_parametros $LINENO 3 $@

# inicializamos las variables con los parámetros introducidos HORA\_INICIAL=$1

HORA\_FINAL=$2 NOMBRE\_PACIENTE=$3

# comprobamos que el formato de la hora inicial sea el correcto formato\_hora $LINENO $HORA\_INICIAL

# comprobamos que la hora inicial no se solape con alguna cita solape\_hora\_inicial $LINENO $HORA\_INICIAL

# comprobamos que el formato de la hora final sea el correcto formato\_hora $LINENO $HORA\_FINAL

# comprobamos que la hora final no se solape con alguna cita solape\_hora\_final $LINENO $HORA\_FINAL

# comprobamos que no exista ya el nombre del paciente nombre\_paciente $LINENO $NOMBRE\_PACIENTE

# grabamos al final del fichero

echo $HORA\_INICIAL $HORA\_FINAL $NOMBRE\_PACIENTE >> $CITAS\_FILE

# comprobamos que no haya habido ningún error inexperado error\_inexperado $LINENO

}

# función de búsqueda por NOMBRE\_PACIENTE # $1 patrón de búsqueda

function search() {

# comprobamos que el número de parámetros sea igual a 1 numero\_parametros $LINENO 1 $@

PATRON=$1

# buscamos el patrón introducido en el fichero de las citas grep $PATRON $CITAS\_FILE

# comprobamos que no haya habido ningún error inexperado error\_inexperado $LINENO

}

# función de búsqueda por HORA\_INICIAL # $1 hora inicial

function init() {

# comprobamos que el número de parámetros sea igual a 1 numero\_parametros $LINENO 1 $@

HORA\_INICIAL=$1

# comprobamos que el formato de la hora inicial sea el correcto formato\_hora $LINENO $HORA\_INICIAL

# buscamos la hora inicial en el fichero de las citas grep ­E ^$HORA\_INICIAL.\*$ $CITAS\_FILE

# comprobamos que no haya habido ningún error inexperado error\_inexperado $LINENO

}

# función de búsqueda por HORA\_FINAL # $1 hora final

function end() {

# comprobamos que el número de parámetros sea igual a 1 numero\_parametros $LINENO 1 $@

HORA\_FINAL=$1

# comprobamos que el formato de la hora final sea el correcto formato\_hora $LINENO $HORA\_FINAL

# buscamos la hora final en el fichero de las citas grep ­E ^...$HORA\_FINAL.\*$ $CITAS\_FILE

# comprobamos que no haya habido ningún error inexperado error\_inexperado $LINENO

}

# función de listado ordenados por NOMBRE\_PACIENTE function name() {

# comprobamos que el número de parámetros sea igual a 0 numero\_parametros $LINENO 0 $@

# ordenamos el fichero de las citas por la tercera columna sort ­k 3 $CITAS\_FILE

# comprobamos que no haya habido ningún error inexperado error\_inexperado $LINENO

}

# función de listado ordenados por HORA\_INICIAL function hour() {

# comprobamos que el número de parámetros sea igual a 0 numero\_parametros $LINENO 0 $@

# ordenamos el fichero de las citas por la primera columna en formato numérico

sort ­nk 1 $CITAS\_FILE

# comprobamos que no haya habido ningún error inexperado error\_inexperado $LINENO

}

# vemos si tenemos acceso al fichero de las citas touch $CITAS\_FILE

# comprobamos que no haya habido ningún error inexperado error\_inexperado

# recogemos la opción selecionada OPCION=$1

# eliminamos el primer parámetro shift

# dependiendo de la opción seleccionada case $OPCION in

­h|­­help) ayuda;;

­a|­­add) add $@;;

­s|­­search) search $@;;

­i|­­init) init $@;;

­e|­­end) end $@;;

­n|­­name) name $@;;

­o|­­hour) hour $@;;

\*) error $LINENO "Opción '$OPCION' inválida." 5;;

esac

1. Realizar un script llamado '**citas­menu.sh**' que sea una interfaz del script 'citas' motrando un menú con las siguientes opciones:
2. *Añadir cita nueva.*
3. *Buscar por nombre del paciente. 3.Buscar citas por hora inicial. 4.Buscar citas por hora final.*

*5.Listar las citas ordenadas por nombre del paciente. 6.Listar las citas ordenadas por hora inicial.*

1. *Salir del programa.*

*#! /bin/bash*

*# script interfaz del script 'citas'*

*# variables globales CITAS\_SCRIPT=./citas*

*# función de ayuda function ayuda() {*

*cat << DESCRIPCION\_AYUDA SYNOPIS*

*$0 [OPCIONES] DESCRIPCIÓN*

*Añade y busca citas HORA\_INICIAL, HORA\_FINAL y NOMBRE\_PACIENTE. OPCIONES*

*­h ­­help Muesta esta ayuda.*

*CODIGOS DE RETORNO*

*0 Si no hay ningún error. DESCRIPCION\_AYUDA*

*}*

*# función menu function menu() {*

*cat << DESCRIPCION\_MENU*

*+­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­+*

*| MENU DE CITAS |*

*+­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­+*

*| a. Añadir una cita con HORA\_INICIAL, HORA\_FINAL, y NOMBRE\_PACIENTE. |*

*| s. Buscar los pacientes que contengan PATRÓN. |*

*| i. Buscar las citas que empiecen a HORA\_INICIAL. |*

*| e. Buscar las citas que terminen a HORA\_FINAL. |*

*| n. Listar todas las citas ordenadas por NOMBRE\_PACIENTE. |*

*| o. Listar todas las citas ordenadas por HORA\_INICIAL. |*

*| s. Salir del programa. |*

*+­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­+*

*DESCRIPCION\_MENU*

*}*

*# función de error*

*# $1 línea de error # $2 mensaje de error function error() {*

*echo "$0: línea $1: $2"*

*}*

*# función para añadir una cita function add() {*

*echo "AÑADIR UNA CITA NUEVA"*

*read ­p "Introduce la hora inicial (de 00 a 23): " HORA\_INICIAL read ­p "Introduce la hora final (de 00 a 23): " HORA\_FINAL read ­p "Introduce el nombre del paciente: " NOMBRE\_PACIENTE*

*$CITAS\_SCRIPT ­­add $HORA\_INICIAL $HORA\_FINAL $NOMBRE\_PACIENTE*

*}*

*# función de búsqueda function search() {*

*echo "BUSCAR POR HORA INICIAL, HORA FINAL o NOMBRE DEL PACIENTE"*

*read ­p "Introduce un patrón de búsqueda: " PATRON*

*$CITAS\_SCRIPT ­­search $PATRON*

*}*

*# función de búsqueda por hora inicial function init() {*

*echo "BUSCAR POR HORA INICIAL"*

*read ­p "Introduce la hora inicial (de 00 a 23): " HORA\_INICIAL*

*$CITAS\_SCRIPT ­­init $HORA\_INICIAL*

*}*

*# función de búsqueda por hora final function end() {*

*echo "BUSCAR POR HORA FINAL"*

*read ­p "Introduce la hora final (de 00 a 23): " HORA\_FINAL*

*$CITAS\_SCRIPT ­­end $HORA\_FINAL*

*}*

*# función de listado ordenados por nombres function name() {*

*echo "LISTADO ORDENADO POR NOMBRE DEL PACIENTE"*

*$CITAS\_SCRIPT ­­name*

*}*

*# función de listado ordenados por hora de inicial function hour() {*

*echo "LISTADO ORDENADO POR HORA INICIAL"*

*$CITAS\_SCRIPT ­­hour*

*}*

*# función para salir del programa function salir() {*

*exit 0*

*}*

*# función elegir\_menu function elegir\_menu() {*

*menu*

*read ­p "Elige una opción: " OPCION clear*

*case $OPCION in*

* 1. *add ;;*

*s) search ;;*

*i) init ;;*

*e) end ;;*

1. *name ;;*
2. *hour ;;*

*s) salir ;;*

*\*) error $LINENO "Opción $1 inválida." ;;*

*esac*

*elegir\_menu*

*}*

*# si primer parámetro == '­h' o == '­­help' if [ "$1" == "­h" ­o "$1" == "­­help" ]; then*

*ayuda exit 0*

*fi clear*

*elegir\_menu*

1. Realizar un script llamado '**citas­flags.sh**' para poder usar el script '**citas**' mediante CLI.

#! /bin/bash

# script interfaz del script 'citas'

# variables globales CITAS\_SCRIPT=./citas

# función de ayuda function ayuda() {

cat << DESCRIPCION\_AYUDA SYNOPIS

$0 [OPCIONES] [HORA\_INICIAL] [HORA\_FINAL] [NOMBRE\_PACIENTE]

DESCRIPCIÓN

Añade y busca citas de una consulta.

OPCIONES

­h ­­help Para mostrar un texto de ayuda.

­a ­­add Para añadir una cita con HORA\_INICIAL, HORA\_FINAL, y NOMBRE\_PACIENTE.

­s ­­search Para buscar los pacientes que contengan PATRÓN.

­i ­­init Para buscar las citas que empiecen a HORA\_INICIAL.

­e ­­end Para buscar las citas que terminen a HORA\_FINAL.

­n ­­name Para listar todas las citas ordenadas por NOMBRE\_PACIENTE.

­o ­­hour Para listar todas las citas ordenadas por HORA\_INICIAL.

CÓDIGOS DE RETORNO

1. Si no hay ningún error.
2. Si el número de parámetros es incorrecto.
3. Si el formato de los parámetros es incorrecto.
4. Si al añadir una cita se solapa con otra ya introducida.
5. Si al añadir una cita ya existe NOMBRE\_PACIENTE.
6. Si se introduce una opción inválida.
7. Si ocurre otro error no mencionado. DESCRIPCION\_AYUDA

}

# función de error

# $1 línea de error # $2 mensaje de error function error() {

echo "$0: línea $1: $2"

}

# función para añadir una cita function add() {

echo "AÑADIR UNA CITA NUEVA"

$CITAS\_SCRIPT ­­add $@

echo "­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­"

}

# función de búsqueda function search() {

echo "BUSCAR POR PATRÓN ($@)"

$CITAS\_SCRIPT ­­search $@

echo "­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­"

}

# función de búsqueda por hora inicial function init() {

echo "BUSCAR POR HORA INICIAL"

$CITAS\_SCRIPT ­­init $@

echo "­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­"

}

# función de búsqueda por hora final function end() {

echo "BUSCAR POR HORA FINAL"

$CITAS\_SCRIPT ­­end $@

echo "­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­"

}

# función de listado ordenados por nombres function name() {

echo "LISTADO ORDENADO POR NOMBRE DEL PACIENTE"

$CITAS\_SCRIPT ­­name

echo "­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­"

}

# función de listado ordenados por hora de inicial function hour() {

echo "LISTADO ORDENADO POR HORA INICIAL"

$CITAS\_SCRIPT ­­hour

echo "­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­"

}

# función opción invalida function opcion\_invalida() {

echo "Opción '$1' inválida." exit 5

}

while getopts "ha:s:lt" option ; do case "$option" in

h) ayuda ;;

a) add $OPTARG ;;

s) search $OPTARG ;;

i) init $OPTARG ;;

e) end $OPTARG ;;

1. name ;;
2. hour ;;

\*) opcion\_invalida $option ;; esac

done