

Solución Ejercicios de Shell Script

01 - Básicos

1. Realizar un script llamado **'01-hola-mundo.sh'** que muestre por pantalla "Hola mundo!".

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

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

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

3. Ídem y que además verifique que al menos hayamos introducido un parámetro (**'03-hola-al-menos-1-parametro.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 $@"
```

4. Ídem y que además separe cada argumento por ", " (**'04-hola-parametros-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!"
```

5. Ídem y que además en caso de error muestra una ayuda ('05-hola-con-ayuda.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
```

6. Ídem y que además verifique que sean usuarios conectados al sistema ('06-hola-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 $1"
        PRIMERO=0
    else
        MENSAJE="$MENSAJE, $1"
    fi

    # pasamos al siguiente parámetro
    shift
done

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

7. 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
    echo "SI"
fi
```

8. Modificar el fichero '**.bashrc**' para modificar el PATH y añadir la carpeta de estos ejercicios. Para ello añade la siguiente línea: **export PATH=\$PATH:~/ruta_carpeta_ejercicios**

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

9. Modificar el script '**06-hola-usuario.sh**' para que llame a 'usuarioconectado' ('**09-hola-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=`./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 $1"
        PRIMERO=0
    else
        MENSAJE="$MENSAJE, $1"
    fi

    # pasamos al siguiente parámetro
    shift
done

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

10. 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
    echo "SI"
fi
```

11. Modificar el script '**09-hola-usuario.sh**' para que llame a 'usuariosistema' ('**11-hola-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 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 $1"
        PRIMERO=0
    else
        MENSAJE="$MENSAJE, $1"
    fi

    # pasamos al siguiente parámetro
    shift
done

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

02 - Calculadora

12. 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 }'
```


13. 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 }'
```

14. 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 }'
```

15. 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 }'
```

16. 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
```

17. 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
```

18. 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
SYNOPSIS
    $0 EXPRESIÓN_NUMÉRICA
DESCRIPCIÓN
    Muestra por pantalla el valor de EXPRESIÓN_NUMÉRICA.
CODIGOS DE RETORNO
    0 Si no hay ningún error.
    1 Si el número de parámetros es distinto de 1.
    2 Si hay un error de formato en la expresión introducida.
    3 Si hay un error de entrada y salida.
    4 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 ^[\*\0-9\(\)\+\-]+\`" ]; 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
```

19. 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

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

	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
```

03 - Banco

21. 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
    0 Si no hay ningún error.
    1 SI la opción introducida no es válida.
    2 si un argumento numérico no es un número.
    3 Si el número de parámetros es erróneo.
    4 si un argumento de tipo fecha no es una fecha.
    5 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 $DATE | grep -E \
        ^20[0-9]{2}-[01][0-9]-[0-3][0-9]$`

    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 $@

```

22. Realizar un script llamado '**banco-menu.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
SYNOPSIS
    $0 [OPCIONES]
DESCRIPCIÓN
    Añade y busca y opera con movimientos bancarios.
OPCIONES
    -h --help    Muestra 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
        a) add ;;
        b) 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

```

23. Realizar un script llamado '**banco-flags.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
SYNOPSIS
    $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
    0 Si no hay ningún error.
    1 SI la opción introducida no es válida.
    2 si un argumento numérico no es un número.
    3 Si el número de parámetros es erróneo.
    4 si un argumento de tipo fecha no es una fecha.
    5 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 buscar 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
```

04 - Demonios

24. 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
SYNOPSIS
    $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
    SEGUNDOS=$1

    # 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
```

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

```
#!/bin/bash

# función de ayuda
function ayuda() {
cat << DESCRIPCION_AYUDA
SYNOPSIS
    $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 existe 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 existe el fichero
    if [ -e $PIDFILE ]; then
        kill -9 `cat $PIDFILE`
        rm $PIDFILE
    fi
    echo "Parado."
}

# función que para y arranca 'alerta'
function do_restart() {

    do_stop
    do_start
}
```



```
# función que muestra el estado de 'alerta'
function do_status() {

    # si existe 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 ;;
    *)
        echo "Parámetro '$1' incorrecto." ;;
esac
```

05 - Copias

26. Realizar un script llamado '**copia-total**' que empaquete y comprima el contenido de la carpeta '**~/carpeta_a_copiar**' en un fichero llamado '**total-aaaa.mm.dd-HH.MM.SS.tar.zip**' en la carpeta '**~/copia_seguridad**'.

```
#!/bin/bash

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

# variable con la fecha en el formato indicado (aaaa.mm.dd-HH.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/fecha-ultima-copia-total.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 existe el directorio a copiar mostramos un error
# y paramos la ejecución
if [ ! -d $DIRECTORIO_A_COPIAR ]; then
    echo "No existe el directorio a copiar."
    exit 1
fi

# si no existe el directorio de los ficheros lo creamos
if [ ! -d $RUTA_FICHEROS ]; then
    mkdirs $RUTA_FICHEROS
fi

# empaquetamos y comprimimos el DIRECTORIO_A_COPIAR
# en FICHERO_COMPRIMIDO (mediante zip)
zip -r $FICHERO_COMPRIMIDO $DIRECTORIO_A_COPIAR

# guardar la fecha de la última copia total en FICHERO_ULTIMA_COPIA_TOTAL
echo $FECHA > $FICHERO_ULTIMA_COPIA_TOTAL

```

27. Realizar un script llamado '**copia-diferencial**' 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 '**diferencial-aaaa.mm.dd-HH.MM.SS.tar.zip**' en la carpeta '**~/copia_seguridad**'.

```

#!/bin/bash

# script que empaqueta y comprime los ficheros modificados
# desde la última copia-total
# en un fichero llamado diferencial-aaaa.mm.dd-HH.MM.SS.tar.zip

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

# variable con la fecha en el formato indicado (aaaa.mm.dd-HH.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/fecha-ultima-copia-total.txt

# variable con el fichero con la fecha de la última copia diferencial
FICHERO_ULTIMA_COPIA_DIFERENCIAL=$RUTA_FICHEROS/fecha-ultima-copia-
diferencial.txt

# variable con el fichero comprimido
FICHERO_COMPRIMIDO=$RUTA_FICHEROS/diferencial-$FECHA.tar.zip

# variable con el directorio que queremos copiar y comprimir
DIRECTORIO_A_COPIAR=~/directorio_a_copiar

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

```

```

# si no existe el directorio a copiar mostramos un error y paramos la
ejecución
if [ ! -d $DIRECTORIO_A_COPIAR ]; then
    echo "No existe 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 directorio de los ficheros lo creamos
if [ ! -d $RUTA_FICHEROS ]; then
    mkdirs $RUTA_FICHEROS
fi

# empaquetamos y comprimimos los ficheros modificados
# desde la última copia total
find $DIRECTORIO_A_COPIAR/* -newer $FICHERO_ULTIMA_COPIA_TOTAL | zip -@
$FICHERO_COMPRIMIDO

# guardar la fecha de la última copia diferencial
# en FICHERO_ULTIMA_COPIA_DIFERENCIAL
echo $FECHA > $FICHERO_ULTIMA_COPIA_DIFERENCIAL

```

28. Realizar un script llamado '**copia-incremental**' que empaquete y comprima los ficheros de la carpeta '**~/carpeta_a_copiar**' modificados desde la última **copia incremental** en un fichero llamado '**incremental-aaaa.mm.dd-HH.MM.SS.tar.zip**' en la carpeta '**~/copia_seguridad**'.

```

#!/bin/bash

# script que empaqueta y comprime los ficheros modificados
# desde la última copia incremental
# en un fichero llamado incremental-aaaa.mm.dd-HH.MM.SS.tar.zip

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

# variable con la fecha en el formato indicado (aaaa.mm.dd-HH.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 incremental
FICHERO_ULTIMA_COPIA_INCREMENTAL=$RUTA_FICHEROS/fecha-ultima-copia-
incremental.txt

# variable con el fichero comprimido
FICHERO_COMPRIMIDO=$RUTA_FICHEROS/incremental-$FECHA.tar.zip

# variable con el directorio que queremos copiar y comprimir
DIRECTORIO_A_COPIAR=~/directorio_a_copiar

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

```

```

# si no existe el directorio a copiar mostramos un error
# y paramos la ejecución
if [ ! -d $DIRECTORIO_A_COPIAR ]; then
    echo "No existe el directorio a copiar."
    exit 1
fi

# si no existe el FICHERO_ULTIMA_COPIA_INCREMENTAL mostramos un error
# y paramos la ejecución
if [ ! -e $FICHERO_ULTIMA_COPIA_INCREMENTAL ]; then
    echo "No hay última copia incremental."
    exit 1
fi

# si no existe el directorio de los ficheros lo creamos
if [ ! -d $RUTA_FICHEROS ]; then
    mkdirs $RUTA_FICHEROS
fi

# empaqueta y comprime los ficheros modificados
# desde la última copia incremental
find $DIRECTORIO_A_COPIAR/*.txt -newer $FICHERO_ULTIMA_COPIA_INCREMENTAL | zip -@
$FICHERO_COMPRIMIDO

# guardar la fecha de la última copia incremental
# en FICHERO_ULTIMA_COPIA_INCREMENTAL
echo $FECHA > $FICHERO_ULTIMA_COPIA_INCREMENTAL

```

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

```

#####
# minuto (0-59), #
# | hora (0-23), #
# | | día del mes (1-31), #
# | | | mes (1-12), #
# | | | | día de la semana (0-6 donde 0=Domingo) #
# | | | | | comandos #
# | | | | | #
#####
* * * * * date > ~/ultimo-crontab.txt

```

06 - Varios

30. 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
```

31. Realizar a mano un fichero **'roles.csv'** con los siguientes datos:

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

32. Realizar un script **'roles-sin-awk.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
```

33. Realizar un fichero '**roles.awk**' y su correspondiente interfaz '**roles-con-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]
    }
}
```

roles-con-awk.sh:

```
awk -f roles.awk roles.csv
```

34. 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
```

35. 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 parámetro 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 $@

```

36. 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

```


37. 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. Nota: se deberá realizar con una iteración.

```
#!/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
BASE=$1
EXPONENTE=$2

# para cada parámetro introducido
for ((CONTADOR=0; CONTADOR<$EXPONENTE; CONTADOR++)); do

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

done

echo $ELEVADO
```

38. 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_INICIO 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_INICIO 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
SYNOPSIS
    $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
    0 Si no hay ningún error.
    1 Si el número de parámetros es incorrecto.
    2 Si el formato de los parámetros es incorrecto.
    3 Si al añadir una cita se solapa con otra ya introducida.
    4 Si al añadir una cita ya existe NOMBRE_PACIENTE.
    5 Si se introduce una opción inválida.
    6 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() {

    # muestra un mensaje de error
    echo "$0: Línea $1: Error $3: $2"

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

```

```

# función que comprueba el número de parámetros introducido
# $1 línea error
# $2 número de parámetros esperados
# $@ parámetros
function numero_parametros() {

    # línea 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=$#

    # si el número de parámetros introducido es distinto del esperado
    if [ "$NUMERO_PARAMETROS_REALES" != "$NUMERO_PARAMETROS_ESPERADOS" ];
then
        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() {

    # línea error
    LINEA_ERROR=$1

    # recogemos la hora
    HORA=$2

    # si la hora introducida no es un número de 2 cifras
    if [ -z "`echo $HORA | grep -E ^[0-9]{2}$`" ]; 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=~/.awk-program.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=~/.awk-program.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 inesperado
    error_inesperado $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 inesperado
    error_inesperado $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 inesperado
    error_inesperado $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 inesperado
    error_inesperado $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 inesperado
    error_inesperado $LINENO
}

# vemos si tenemos acceso al fichero de las citas
touch $CITAS_FILE

# comprobamos que no haya habido ningún error inesperado
error_inesperado

# recogemos la opción seleccionada
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

```


39. Realizar un script llamado '**citas-menu.sh**' que sea una interfaz del script 'citas' mostrando un menú con las siguientes opciones:

1. *Añadir cita nueva.*
2. *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.*
7. *Salir del programa.*

```
#!/bin/bash

# script interfaz del script 'citas'

# variables globales
CITAS_SCRIPT=./citas

# función de ayuda
function ayuda() {
cat << DESCRIPCION_AYUDA
SYNOPSIS
    $0 [OPCIONES]
DESCRIPCIÓN
    Añade y busca citas HORA_INICIAL, HORA_FINAL y NOMBRE_PACIENTE.
OPCIONES
    -h --help    Muestra 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
        a) add ;;
        s) search ;;
        i) init ;;
        e) end ;;
        n) name ;;
        o) 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
```

40. Realizar un script llamado '**citas-flags.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
SYNOPSIS
    $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
    0 Si no hay ningún error.
    1 Si el número de parámetros es incorrecto.
    2 Si el formato de los parámetros es incorrecto.
    3 Si al añadir una cita se solapa con otra ya introducida.
    4 Si al añadir una cita ya existe NOMBRE_PACIENTE.
    5 Si se introduce una opción inválida.
    6 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 ;;
        n) name ;;
        o) hour ;;
        *) opcion_invalida $option ;;
    esac
done

```