Unidad 8
Programación Shell-script en Linux

Versión 16,02,2022

Definición de Script

Un script es un archivo que incluye un conjunto de comandos. Son ejecutados desde la primera línea hasta la última (de forma secuencial).

¿Qué es el shell script?

Un Shell Script es un script para la shell de comandos (terminal). Para crear un script basta con un editar un fichero nuevo y en el nombre poner .sh Ejemplo: HolaMundo.sh

Definición de Script

En la primera línea del script se debe indicar que shell que vas a usar (/bin/bash/ , /usr/bin/perl , etc) Aunque da igual la que uses lo importante es el contenido:

#! /bin/bash

#! Se conoce con el nombre de **Sha Bang**.

Se denomina “sha-bang” a la secuencia **#!** con la que se inician los scripts. Su función es indicarle al sistema que se trata de un conjunto de comandos para que sean interpretados. En realidad, es un número mágico de dos bytes. El número mágico es un marcador especial para indicar el tipo de archivo, en este caso, indica que se trata de un script de shell ejecutable.

Para introducir comentarios se debe poner **#**. Por cada linea que deseéis poner un comentario, lo primero que debe tener es **#**. Es importante añadir comentarios comentando la utilidad del script o las variables que se crean.

¿Cómo ejecutar un script?

Antes de poder ejecutarlo, debemos darle permisos de ejecución. (**+x**) por ello, haremos uso del comando chmod y damos permisos de ejecución, si se desea, se pueden dar a todos los usuarios y grupos.

chmod 755 /ruta_del_script

Para el usuario propietario

chmod 777 /ruta_del_script

Para cualquier usuario

Una vez hecho todo lo anterior, usaremos:

./nombredelscript.sh

Pero también podemos usar, si es un **shell script bash**:

bash nombredelscript.sh

Código de un script

Vamos a empezar desarrollando lo esencial para ir desarrollando estructuras más complejas.

Lo primero es saber cómo dar valor a una variable. Es tan sencillo como poner:

nombre_variable=valor

Si deseas guardar la salida de un programa solo tienes que ponerlo entre tildes invertidas:

nombre_variable=`comando`

Código de un script

Tambien hay un comando que lee por teclado las variables (read). Para ponerlo es:

read [opciones] nombre_variable1 nombre_variable2 nombre_variableN

Ejemplo:

read -p “Introduce el nombre y los apellidos” nombre apellidos

Tiene un montón de opciones pero estas son las más importantes y usadas:

-n num_car : Número máximo de caracteres que puedes introducir por teclado

-p “frase” : Te muestra por pantalla una frase para tu saber que debes introducir

-d “delimitador” : Especificas cual va a ser el delimitador, es decir si dices que el delimitador sera “;” pues todo lo que venga antes de un “;” lo cogerá una variable y todo lo que venga después de ese delimitador hasta el próximo “;” lo cogerá otra variable.

-d delim continuar hasta que el primer carácter de DELIM sea leído, en vez de una nueva línea.

Cuando queremos utilizar el valor de una variable en el código, nos referiremos a éste como:

$nombre_variable

Variable IFS

IFS= delimitador (Input File Separator)

Sirve para cambiar el delimitador por defecto, que es “ “, así puedes evitar decir:

read -d “,”

Si **IFS=,** en todo el script se usará como separador un “,”.

Para comprobar el valor de **IFS**:

printf "%q" "$IFS"

Se puede usar también para ver el valor de cualquier variable en lugar de usar el comando **echo**.

Ejemplos

A lo largo de los ejemplos se introducen algunos comandos básicos de Linux.

HolaMundo.sh

```
#!/bin/bash
clear
echo “Hola mundo ,este es mi primer script”
```

ScriptUno.sh

```
#!/bin/bash
clear
nombre=“Perico”
apellidos=“Palotes”
echo “Te llamas $nombre $apellidos”
```

Fecha.sh

```
#!/bin/bash
clear
fecha=`date | cut -d “ ” -f 1,2,3`
hora=`date | cut -d “ ” -f 4`
echo “Hoy es $fecha y son las $hora”
```

OtroScript.sh

```
#!/bin/bash
IFS=,
clear
read -p “Introduce el nombre,apellidos (separados por coma): ” nombre apellidos
echo “El nombre es $nombre y los apellidos son $apellidos”
```

Comando test

Este comando sirve para expresar condiciones y evaluarlas, si son correctas origina códigos de salida = 0 y si son falsas = 1

El comando pretende abreviar un poco en algunos casos, por eso se suele utilizar su forma corta:

test expresiones es igual a **[expresión]**

Hay que tener en cuenta que la forma es:

[(espacio)**expresión**(espacio)**]**

Si no se ponen los espacios en blanco daría lugar a error.

Comando test

Un ejemplo de su uso:

test -f /home/alumno && echo “Existe directorio”

La salida sera:

Existe Directorio

En la forma resumida se puede escribir:

[-f /home/alumno] && echo “fichero existe…”

La salida sera:

fichero existe…

Expresiones test

Estas son algunas de las más comunes.

Comprobación directorios:

test -f /ruta/nombre –> Comprueba si es un fichero normal

test -l /ruta/nombre –> Comprueba si es un enlace suave

test -d /ruta/ –> Comprueba que existe el directorio

test -x /ruta/nombre –> Comprueba si es un ejecutable

test -s /ruta/nombre –> comprueba que su tamaño es mayor a 0

Comprobación de cadenas de texto:

test “$cadena1” = “$cadena2” –> Comprueba si son iguales

test -z $cadena –> Comprueba si está vacía

test “$cadena1” != “$cadena2” –> Comprueba que son diferentes

Expresiones test

Comprobación de expresiones numéricas:

test exp -eq exp2 –> Comprueba si son iguales

test exp -ge exp2 –> Comprueba si exp >= exp2

test exp -ne exp2 –> Comprueba si exp distinto de exp2

test exp -gt exp2 –> Comprueba si exp > exp2

test exp -le exp2 –> Comprueba si exp <= exp2

test exp -lt exp2 –> Comprueba si exp < exp2

Para concatenar expresiones a evaluar:

-o -> OR

-a -> AND

! -> NOT

Ejemplos

numeros.sh

```
#!/bin/bash
clear
read -p “Introduce dos números ” num1 num2
if [ -z $num1 -o -z $num2 ]
then
echo “Debes introducir dos números, por favor”
elif [ $num1 -eq $num2 ]
then
echo “Los números son iguales”
elif [ $num1 -gt $num2 ]
then
echo “El $num1 > que $num2″
fi
```

Ejemplos

BuscaFich.sh

```
#!/bin/bash
clear
read -p “Introduce directorio a buscar… ” direct
read -p “Nombre de fichero a buscar… ” nombre
if [ ! -d $direct ]
then
echo “$direct no existe”
else
find $direct -name “*.$nombre” -exec ls -l ‘{}’ \;
fi
```

Estructuras condicionales

La estructura básica de una condición sería:

if condición
then
    comandos
    ...
else
    comandos
    ...
fi

Si la condición se cumple entraría por el **then**, en caso de que no, por el **else**. Para cerrar la estructura se usa **fi**.

Estructuras condicionales

Uno más complejo con if anidados, sería:

```
if condición1
then
    comandos
elif condición2
then
    comandos
elif condición3
then
    comandos
else
    comandos
fi
```

La condición es cualquier cosa que devuelva algo que sea 0 o verdadero.

Ejemplo

ExisteGrupoUsuario.sh

```
#!/bin/bash
clear
read -p “Introduce usuario… ” user
read -p “Introduce grupo… ” group
if `grep -e “^$user:.*” /etc/passwd >/dev/null`
then
    if `grep -e “^$group:.*” /etc/group >/dev/null`
    then
    echo “Usuario y grupo ya existen en el sistema”
    fi
elif `grep -e “^$group:.*” /etc/group >/dev/null`
then
    echo “usuario no existe, grupo si!!”
else
    echo “Ni grupo ni usuario existen”
fi
```

Estructuras condicionales

La estructura **case** es de la siguiente forma:

```
case expresion in
caso1)
    comandos ;;
caso2)
    comandos ;;
*)
    comandos ;;
esac
```

Según sea el valor de esa expresión se hará un caso u otro.

Ejemplo

tecla.sh

```
#!/bin/bash
clear
read -n 1 -p “Pulsa una tecla ” tecla
case $tecla in
[a-z,A-Z])
    echo “Ha introducido una letra” ;;
[0-9])
    echo “Ha introducido un numero” ;;
*)
    echo “Ha introducido un caracter especial” ;;
esac
```

Ejemplo

ejmplMenu.sh

```
#!/bin/bash
clear
echo “1.Ejemplo de menu uno”
echo “2.Ejemplo de menu dos”
read -n 1 -p “Introduce una opcion” opcion
case $opcion in
1) exit 1 ;;
2) exit 2 ;;
*) echo “No has introducido ni un dos ni un uno” ;;
esac
```

Estructuras condicionales

Tener que poner tantos **echo** es bastante molesto, por eso hay un comando que te ahorra hacer ese esfuerzo (**select**):

```
select variable in “caso 1″ “caso 2″ ... “caso N”
do
   break
done
```

```
case $variable in
"caso 1") comandos ;;
"caso 2") comandos ;;
…
"caso N") comandos ;;
*) comandos;;
esac
```

ó

```
case $REPLY in
 1) comandos ;;
 2) comandos ;;
 ...
 N) comandos ;;
*) comandos;;
 esac
```

El **break** sirve para que solo te muestre una vez el menú. Cuando usas **select** no hace falta pedir que introduzcas nada, ya que eso lo hace automáticamente. **$REPLY** guarda el valor pulsado en el menú del **select**. El **prompt** que te muestra **select** es **#?** pero se puede cambiar, poniendo otro valor a la variable **PS3**.

Ejemplo

ejmplMenu.sh (version 1)

```
#!/bin/bash
clear
echo “Elije una opcion”
select opcion in “Ejemplo de menu uno” “Ejemplo de menu dos”
do
    break
done
case $opcion in
“Ejemplo de menu uno”) echo “Has pulsado $REPLY” ;;
“Ejemplo de menu dos”) echo “Has pulsado $REPLY” ;;
*) echo “No has introducido ni un dos ni un uno” ;;
esac
```

Ejemplo

ejmplMenu.sh (version 2)

```
#!/bin/bash
clear
echo “Elije una opcion”
select opcion in “1.Ejemplo de menu uno” “2.Ejemplo de menu dos”
do
     break
done
case $REPLY in
1) echo $opcion ;;
2) echo $opcion ;;
*) echo “No has introducido ni un dos ni un uno” ;;
esac
```

Ejemplo

ExprEjemplo.sh

```
#!/bin/bash
clear
PS3=“Introduce Opción: ”
select opcion in “suma” “resta”
do
    break
done
read -p “Introduce dos números… ” num1 num2
case $opcion in
suma)
    echo “La suma de $num1 y $num2 es … `expr $num1 + $num2`
” ;;
resta)
    echo “La resta de $num1 y $num2 es … `expr $num1 - $num2` ” ;;
esac
```

Operaciones algebraicas

expr operación_a_evaluar

Operaciones numéricas:

expr $num1 + $num2 –> Devuelve la suma de num1 + num2

expr $num1 - $num2 –> Devuelve la resta de num1 - num2

expr $num1 * $num2 –> Devuelve el producto de num1 * num2

(se pone * para que no interprete el asterisco como comodín)

expr $num1 / $num2 –> Devuelve la división de num1 / num2

Operaciones lógicas:

expr $num1 >= $num2 –> Devuelve 1 si num1 >= num2

expr $num1 > $num2 –> Devuelve 1 si num1 > num2

expr $num1 <= $num2 –> Devuelve 1 si num1 < num2

expr $num1 < $num2 –> Devuelve 1 si num1 < num2

expr $num1 != $num2 –> Devuelve 1 si num1 es distinto de num2

Mucho cuidado porque expr no se puede usar directamente como una condición en instrucciones condicionales si no lo comparamos con 0 o 1 dentro en test.

Operaciones algebraicas

Operaciones con cadenas:
expr length $cadena –> Nº de caracteres de esa cadena
expr index $cadena_donde_busca $cadena_a_buscar –> Devuelve la posición donde encuentra los caracteres a buscar dentro de la cadena, si no, devuelve un 0.

ExprEjemplo2.sh

```
#!/bin/bash
clear
frase="Buenos días, estamos aprendiendo a programar"
long=`expr length "$frase"`
echo "La longitud de la cadena es... " $long
read -p "Introduce alguna cadena que buscar " buscar
if [ ! -z $buscar ]
then
 pos=`expr index "$frase" "$buscar"`
 if [ $pos -eq 0 ]
 then
 echo "No existe ningún caracter de $buscar en la cadena $frase"
 else
 echo "El primer carácter encontrado de $buscar se encuentra en la posición
$pos de la cadena de texto $frase"
 fi
fi
```

Bucles: FOR

for variable **in** valor1 valor2 … valorN

do

comando1

…

[**break** | **continue**]

done

El bucle se ejecuta para cada uno de los valores que toma la variable en esa lista.

Break: Rompe el bucle y no da más opción a que la variable se ejecute.

Continue: Salta al siguiente valor de la lista.

Ejemplo

Planetas.sh

```
#!/bin/bash
clear
for planeta in “Jupiter 10″ “Venus 30″ “Saturno 15″ “Mercurio 1″ Luna Tierra
do
    if [ “$planeta” = Tierra ]
    then
    break
    elif [ “$planeta” = Luna ]
    then
    continue
    else
    echo “El planeta $planeta 0.000.000 Km del Sol”
    fi
done
echo “fin del script”
```

Bucles: FOR

Otra de las particularidades de este bucle es que puede ejecutarse a la forma de un bucle en java.

((valores))

for((variable=valor; condición; incremento))

do

comando

...

[**break** | **continue**]

...

done

Es igual que en java pero con doble paréntesis.

Ejemplo

Contador.sh

```
#!/bin/bash
clear
read -p “Introduce un numero” numero
for(( a=0; a <= $numero; a++ ))
do
    echo -e “$a \n”
done
```

Bucles: While

El while se estructura de la siguiente forma:

while condicion

do

    break

done

While se usa para repetir un conjunto de comandos/instrucciones dependiendo de si se cumple o no la condición. La condiciones que se pueden poner en el **while** son con el comando **test** , poniendo un true (poniendo un **true** en el **while** se crea un bucle infinito) o poner un comando con comillas invertidas. El **break** se pone solo si quieres salir bruscamente del bucle.

Ejemplo

Calculadora.sh

```
#!/bin/bash
clear
opcion=2
while [ $opcion -ne 5 ]
do
	echo "1.suma"
	echo "2.resta"
	echo "3.multiplicación"
	echo "4.división“
	echo "5.salir"
	read -n 1 -p "Introduce una opcion " opcion
	case $opcion in
	1) read -p "Introduce el 1 numero " numero1
	read -p "Introduce el 2 numero " numero2
	echo "El resultado es `expr $numero1 + $numero2`" ;;
	2) read -p "Introduce el 1 numero " numero1
	read -p "Introduce el 2 numero " numero2
	echo "El resultado es `expr $numero1 - $numero2`" ;;
	3) read -p "Introduce el 1 numero " numero1
	read -p "Introduce el 2 numero " numero2
	echo "El resultado es `expr $numero1 '*' $numero2`" ;;
	4) read -p "Introduce el 1 numero " numero1
	read -p "Introduce el 2 numero " numero2
	echo "El resultado es `expr $numero1 '/' $numero2`" ;;
	esac
done
```

Ejemplo

Si en el while vas a poner una variable, debe declararse antes ya que sino no
entra, darle cualquier valor. Si no se le da valor antes, la variable no valdrá nada (en el ejemplo anterior hubiera fallado ya que estás diciendo que se hace el bucle mientras ” ” no sea igual a 5, sin embargo al dar valor a la variable opción antes del while, entra).

BucleInfinito.sh

```
#!/bin/bash
clear
while true
do
	read -p "Introduce la palabra fin para salir del bucle " fin
	if [ "$fin" = "fin" ]
	then
		exit;
	fi
done
```

Bucles: Until

La estructura repetitiva **until** es de la siguiente forma:

until condición

do

    break

done

La estructura **until** se usa para repetir un conjunto de comandos hasta que se cumpla la condición, cuando se cumple el script sale del **until**. Las condiciones y el **break** es lo mismo que en el **while**, si se usa una variable en el **until** se debe declarar antes.

Ejemplo

BorrarFicheros.sh

```
#!/bin/bash
clear
directorio=malo
until `cd $directorio 2> /dev/null`
do
    clear
    read -p "Introduce un directorio " directorio
done
echo "Borrado de ficheros"
rm -i $directorio/*
```

Este script comprueba si el directorio existe, si el directorio que introduces no existe te volverá a pedir la ruta, una vez introduzcas una ruta que existe, entonces saldrá del until y borrará todos los ficheros de su interior.

Parámetros posicionales

Son valores que se le pasan al script desde la línea de comandos cuando se ejecuta, se numeran en orden del 1 al 12. A partir del 10 hay que encerrarlo entre llaves, ej. **${11}**

$ ./Script valor1 valor2 … valorN

./Script = valor0

Para ver el contenido de las variables se utiliza el **$NumParámetro** (**$1 $2 $3 $4** ...)
El conjunto de todos los parámetros se puede recuperar de golpe con **$***
El número de parámetros que se le pasan al script esta definido como **$#**

Ejemplo

BorrarFicheroParametros.sh

```
#!/bin/bash
# Al ejecutar como 1º parámetro un directorio, 2º como fichero a borrar
clear
if [ $# -ne 2 ]
then
        echo “Debes ejecutarlo así: $0 directorio nombreFichero”; exit 65
elif [ ! -d $1 ]
then
        echo “El parámetro 1 no es un directorio!! “; exit 65
elif [ ! -f $1/$2 ]
then
        echo “El parámetro 2 no es un fichero!; exit 65
else
        echo “Borrando el fichero… “
        rm -fi $1/$2
fi
```

Vamos comprobando poco a poco si es un directorio correcto, si lo es, pasamos a comprobar el fichero, y si lo es lo borramos.

65 /* data format error */

Parámetros posicionales

Shift: Este comando desplaza elementos a la izquierda machacando el primero y se pierde, un ejemplo:

ParametrosBucleShift.sh

```
#! /bin/bash
clear
while [ “$1″ != “” ]
do
      echo “$1 $2 $3 $4 $5 $6 $7 $8 $9 ${10} ${11} ${12}”
      shift
done
```

El resultado sería el siguiente:

./ParametrosBucleShift.sh 1 dos tres 4 5 seis siete 8 nueve 10 11 doce

```
1 dos tres 4 5 seis siete 8 nueve 10 11 doce
dos tres 4 5 seis siete 8 nueve 10 11 doce
tres 4 5 seis siete 8 nueve 10 11 doce
4 5 seis siete 8 nueve 10 11 doce
5 seis siete 8 nueve 10 11 doce
seis siete 8 nueve 10 11 doce
siete 8 nueve 10 11 doce
8 nueve 10 11 doce
nueve 10 11 doce
10 11 doce
11 doce
doce
```

Parámetros especiales

Son parámetros identificados por un carácter especial creados por el shell y cuyo valor no puede ser modificado directamente. A continuación se muestran los parámetros especiales definidos en el estándar:

$* → Se expande a todos los parámetros posicionales desde el 1. Si se usa dentro de comillas dobles, se expande como una única palabra formada por los parámetros posicionales separados por el primer carácter de la variable IFS (si la variable IFS no está definida, se usa el espacio como separador y si está definida a la cadena nula, los campos se concatenan).

$@ → Se expande a todos los parámetros posicionales desde el 1, como campos separados, incluso aunque se use dentro de comillas dobles.

$0 → Nombre del shell o shell-script que se está ejecutando.

Parámetros especiales

$# → Nº de argumentos pasados al script (no incluye el nombre del script).

$? → Valor devuelto por el último comando, script, función o sentencia de control invocado. Recuerde que, en general, cualquier comando devuelve un valor. Usualmente, cuando un comando encuentra un error devuelve un valor distinto de cero.

$$ → PID del proceso shell que está interpretando el script.

$! → PID del último proceso puesto en segundo plano.

Parámetros especiales

Ejemplo1

```
#!/bin/bash
cont=0
clear
for a in $*
do
        cont=`expr $cont + 1`
        echo “El valor de la posición $cont es $a”
done
```

Ejemplo2

```
#!/bin/bash
cont=0
clear
max=$#
until [ $cont -eq $max ]
do
        cont=`expr $cont + 1`
        echo "EL valor de la posición $cont es $1"
        shift
done
```

Arrays

Los arrays de los script funcionan de la misma forma que los arrays de cualquier lenguaje de programación. Un array es un conjunto o agrupación de valores cuyo acceso se realiza por índices, en un script se puede almacenar en un mismo array todo tipo de cosas, números, cadenas, caracteres, etc.

En los arrays el primer elemento que se almacena lo hace en la posición 0 (en el ejemplo seria Paco). En los script no hace falta declarar el tamaño del array, puedes insertar tantos valores como desees. Para declarar un array:

declare -a nombre_array

La opción -a sirve para decir que lo que vas a declarar es un array. Para darle valores se puede hacer de dos formas:

1. Darle valores posición por posición.

nombre_array[posicion]=valor

2. Darle todos los valores de golpe (aunque también se puede decir la posición deseada en la que quieres guardar uno de los valores).

nombre_array=(valor1 valor2 valor3 [posicion]=valor4 ..... valorN)

Arrays

Para ver el contenido del array en una posición: **${nombre_array[posición]}**

Para saber cuantos elementos contiene un array: **${#nombre_array[*]}**

Para recuperar todos los elementos de un array: **${nombre_array[*]}**

Para borrar una posición del array: **unset nombre_array[posición]**

Para borrar todo el array : **unset nombre_array**

Ejemplo:
arrays.sh

```
#!/bin/bash
clear
contador=0
declare -a usuario=( Alberto John Roberto Laura Sergio Cristian Dani )
for valor in ${usuario[*]}
do
	echo "El usuario $contador vale $valor"
	contador=`expr $contador + 1`
done
```

Funciones

En el ámbito de la programación, una función es un tipo subalgoritmo, es el término para describir una secuencia de órdenes que hacen una tarea específica de una aplicación más grande.

function nombreFuncion **(){**
comando1
comando2
...
[**return** codigoSalida]
}

También se especifica sin poner **function**, pero puede llegar a dar problemas así que se recomienda ponerlo.

El código de salida especificado por un **return** es el código de salida del resultado de la ejecución de todos los comandos en la función. Si no se especifica un **return** devolverá el de la última salida de esa función.

Dentro de una función se pueden definir variables locales (solo reconocidas por esa función) y se especifican así:

local nombreVariable

Importante, las funciones se declaran al principio de los scripts.

Ejemplo

sumaFuncion.sh

```
#! /bin/bash

function suma() {
      local resultado
      read -p “Introduce el primer numero: ” num1
      read -p “Introduce el segundo numero: ” num2
      resultado=`expr $num1 + $num2`
      return $resultado
}

#———Cuerpo del script———
clear
# Llamo a la funcion suma
suma
echo “El resultado es $?”
```

Funciones

En el paso de parámetros en una función no se pueden definir como en otros lenguajes las variables que le pasas dentro de los paréntesis, sino que se pasan los valores poniéndolos a continuación del nombre de la función:
nombreFuncion valor1 valor2 valor 3 ...

Dentro de una función esos valores se recogen como:
valor1=$1 valor2=$2 ...

Ejemplo
sumaFuncionParametros.sh

```
#! /bin/bash

function suma() {
        local resultado
        resultado=`expr $1 + $2`
        return $resultado
}

#—–Cuerpo del script——
clear
read -p “Introduce el primer numero: ” num1
read -p “Introduce el segundo numero: ” num2
echo “Llamo a la funcion suma”
suma $num1 $num2
echo “El resultado es $?”
```

Funciones

Muy importante entender que si un shellscript recibe un argumento o parámetro, y a su vez tenemos funciones que reciben parámetros al ser llamadas, la variable $1 donde la función recibe el primer parámetro es diferente a la variable $1 que recibe el shellscript .

Ejemplo: Supongamos que llamamos a este programa, le tenemos que pasar dos argumentos, y debe calcular la mitad del segundo argumento.

./shellscriptConFunciones.sh 2 100

```
#! /bin/bash

function mitad() {
        local resultado
        # $1=100 y $2=2
        resultado=`expr $1 / $2`
        return $resultado
}

#—–Cuerpo del script——
clear
echo “Llamo a la funcion mitad”
# $1=2 y $2=100
suma $2 $1
echo “El resultado es $?”
```

Códigos de **exit**

```
$ cat /usr/include/sysexits.h
.....
#define EX_OK    0      /* successful termination */
#define EX__BASE      64    /* base value for error messages */
#define EX_USAGE      64    /* command line usage error */
#define EX_DATAERR  65    /* data format error */
#define EX_NOINPUT  66    /* cannot open input */
#define EX_NOUSER   67    /* addressee unknown */
#define EX_NOHOST   68    /* host name unknown */
#define EX_UNAVAILABLE   69    /* service unavailable */
#define EX_SOFTWARE   70    /* internal software error */
#define EX_OSERR      71    /* system error (e.g., can't fork) */
#define EX_OSFILE     72    /* critical OS file missing */
#define EX_CANTCREAT  73   /* can't create (user) output file */
#define EX_IOERR      74   /* input/output error */
#define EX_TEMPFAIL  75   /* temp failure; user is invited to retry */
#define EX_PROTOCOL   76   /* remote error in protocol */
#define EX_NOPERM   77   /* permission denied */
#define EX_CONFIG     78    /* configuration error */
#define EX__MAX78    /* maximum listed value */

#endif /* sysexits.h */
```

Según la documentación de Bash:

1: Catchall para errores generales
2: Uso indebido de los componentes de Shell
126: El comando invocado no se puede ejecutar
127: "comando no encontrado"
128: Argumento no válido para salir
128 + n: Señal de error fatal "n"
255: Salir del estado fuera de rango (exit solo toma argumentos enteros en el rango de 0 - 255)

Sirven para indicar al programador cual es el motivo que ha ocurrido para interrumpir la ejecución de un script.