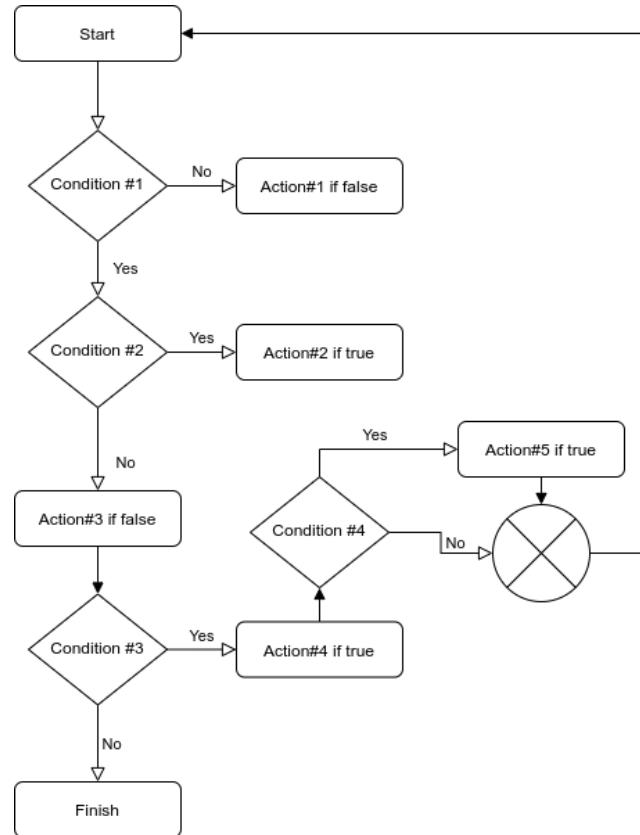


Estructuras condicionales

Estructuras condicionales e iteradores

Hasta ahora, solo hemos conocido una única manera de ejecutar nuestros *scripts*: tienen un punto de entrada y un único punto de salida, y de por medio la ejecución es siempre la misma (las únicas variaciones se producen por algún comando que se ejecuta internamente que produce distinto resultado según el momento de ejecución).

Si bien esto es muy útil, no ofrece en realidad algo “adaptable” que pueda realizar ciertas acciones según variables o valores. Por ejemplo, ¿qué sucede si queremos realizar cierta acción o mostrar un mensaje si el código de retorno (`$?`) es distinto de 0? ¿Cómo podemos notificar al usuario de que un fichero no existe y que el programa no se ejecuta?



Estructuras condicionales

cadena1 = cadena2	Verdadero si “cadena1” es igual a “cadena2”
cadena1 ≠ cadena2	Verdadero si “cadena1” no es igual a “cadena2” (!=).
cadena1 < cadena2	Verdadero si cadena1 es menor a cadena2
cadena1 > cadena2	Verdadero si cadena1 es mayor que cadena2
-n variable1	Verdadero si cadena1 no es nulo
-z variable1	Verdadero si cadena1 es nulo

Una primera aproximación muy eficaz son las estructuras de control, también conocidas como estructuras condicionales. Se comprenden de una secuencia de sentencias **if – elif – else** que permite evaluar condiciones, realizar una acción si son verdaderas o realizar una acción si son falsas.

Estas sentencias se pueden “anidar” tantas veces se necesite y siguen siempre la misma estructura (en bash):

```
if [ expresión ]; then  
    # Acción si verdadero  
fi
```

Para comparar cadenas de caracteres, se usa la tabla de la izquierda.

Estructuras condicionales

```
user@users-desktop:~$ if [ "a" = "b" ]; then echo "Son iguales"; fi
user@users-desktop:~$ if [ "a" != "b" ]; then echo "Son diferentes"; fi
Son diferentes
user@users-desktop:~$ if [ "variable1" > "variable2" ]; then echo "variable1 es mayor"; fi
variable1 es mayor
user@users-desktop:~$ cat dia.sh
#!/bin/env bash
DIA=$(date +%A)
if [ $DIA = "viernes" ]; then
    echo "Mufasa, IT'S FRIDAY THEN!"
elif [ $DIA = "miércoles" ]; then
    echo "What a week, huh?"
    echo "Captain, it's Wednesday..."
else
    echo "Todavía toca aguantar..."
fi
user@users-desktop:~$ ./dia.sh
Todavía toca aguantar...
```

Estructuras condicionales

Sin embargo, las estructuras de comparación anteriores solo sirven para trabajar con strings, es decir, cadenas de texto.

Si queremos comparar con números es necesario utilizar una tabla similar a la vista anteriormente.

`num1 -eq num2` Verdadero si `num1` igual a `num2`

`num1 -ne num2` Verdadero si `num1` distinto de `num2`

`num1 -lt num2` Verdadero si `num1` menor que `num2` (*less than*)

`num1 -gt num2` Verdadero si `num1` mayor que `num2` (*greater than*)

`num1 -le num2` Verdadero si `num1` menor o igual que `num2` (*lower or equals*)

`num1 -ge num2` Verdadero si `num1` mayor o igual que `num2` (*greater or equals*)

Estructuras condicionales

```
user@users-desktop:~$ cat numeros.sh
#!/bin/env bash
let NUM=RANDOM

if [ $NUM -gt 5 ]; then
    echo "$NUM es mayor que 5"
fi
if [ $NUM -lt 5 ]; then
    echo "$NUM es menor que 5"
fi

exit 0
user@users-desktop:~$ ./numeros.sh
15757 es mayor que 5
user@users-desktop:~$ ./numeros.sh
user@users-desktop:~$ ./numeros.sh
3 es menor que 5
```

Estructuras condicionales

Es fundamental utilizar las sentencias anteriores con los tipos de dato correctos. En otro caso, se producirán errores o fallarán:

```
if [3 -eq 5 ]; then
```

bash: [3: command not found
Se ha usado [3 en lugar de [3

```
if [ "a" -eq "b" ]; then
```

bash: [: jose: integer expression expected
Se debería haber usado '='

```
if [ 3 = 4 ]; then
```

Esto no produce error y en apariencia funciona. Sin embargo,
no es así. Se ha de usar -eq

```
if [ 3 > 4 ]; then
```

Esta expresión devuelve verdadero, lo cual es falso. Por ello
no se han de usar operadores *strings* con números.

```
if [ a=b ]; then
```

No se ha dejado espacio en la condición y produce
“Verdadero”. Cuidado con este tipo de errores

```
#!/bin/bash
PROFESOR="Javier"
if [ $PROFSOR = "Javier" ]; then
    echo "Hola Javier"
fi
```

bash: [= unary operator expected
Al haber un error en el nombre de la
variable, es como haber escrito "[=
"Juana"]"

Estructuras condicionales

Al inicio se ha mencionado que una estructura condicional no cuenta únicamente con **if – else** sino que se puede anidar y complicar “a placer” (aunque es mejor intentar simplificar siempre que sea posible).

Todas las estructuras condicionales se inician siempre con un “**if**”. Sin embargo, se pueden cerrar directamente, añadir una rama “**else**” o añadir el condicionante “**elif**”.

Es fundamental que una estructura “**if**” siempre acabe por “**fi**”, indicativo de que ya se ha terminado de procesar cualquier orden que haya en su interior. Finalmente, en “**if**” y “**elif**” es necesario añadir “**then**” después de

```
if [ expresión 1 ]; then
    acción si verdadero
else
    acción si falso
fi

if [ expresión 1 ]; then
    acción si verdadero
elif [ expresión 2 ]; then
    acción si #1 falso y #2 verdadero
elif [ expresión 3 ]; then
    acción si #1 y #2 falsos y #3
    verdadero
else
    acción si todas las expresiones falsas
fi
```

Estructuras condicionales

-a fichero	Verdadero si existe el fichero
-d fichero	Verdadero si existe y es un directorio
-f fichero	Verdadero si existe y es un fichero regular
-r fichero	Verdadero si existe y se puede leer
-w fichero	Verdadero si existe y se puede escribir
-x fichero	Verdadero si existe y se puede ejecutar
f1 -nt f2	Verdadero si “f1” es más nuevo que “f2” (<i>newer than</i>)
f1 -ot f2	Verdadero si “f1” es más viejo que “f2” (<i>older than</i>)

Hasta ahora hemos visto operadores tanto aritméticos como para cadenas de texto, pero en realidad en las expresiones de los bloques **if** – **elif** – **else** se puede usar cualquier operación que devuelva un valor lógico: 0 ó 1 (Verdadero y Falso).

Por ejemplo, se pueden usar las operaciones del comando “**test**” de bash para realizar comprobaciones sobre ficheros.

Este comando devuelve valor ‘0’ si la prueba es verdadera y un valor distinto de ‘0’ para indicar que ha sido fallida y el porqué.

A la izquierda se muestran algunos usos de la función **test** sobre ficheros.

Estructuras condicionales

Con lo que se ha visto, se pueden evaluar condiciones y realizar acciones si la primera condición no se cumple. Sin embargo, ¿cómo podemos evaluar múltiples condiciones en una única sentencia? ¿Cómo se puede realizar una acción única y exclusivamente si el comando falla (o se ejecuta correctamente)?

En programación (y en bash en particular) existen operadores binarios. Sin entrar en mucho detalle, los que necesitamos son:

- **&&** - “and”: se evalúa como “y”. Requiere de ambas partes ser verdaderas.
- **||** - “or”: se evalúa como “o”. Requiere de al menos una parte ser verdadera.
- **!** - “not”: se evalúa como “no”. Niega el resultado de la operación.

```
#!/bin/env bash

if [ EXPR1 ] && [ EXPR2 ]; then
    # Acción a ejecutar si EXPR1 y EXPR2
    # son verdaderos
fi

if [ EXPR1 ] || [ EXPR2 ]; then
    # Acción a ejecutar si EXPR1 o EXPR2
    # son verdaderos (alguno)
fi

if ! [ EXPR1 ]; then
    # Acción a ejecutar si EXPR1 no es
    # verdadero
fi

if [[ EXPR1 && EXPR2 ]]; then
    # Como el primero, más seguro y eficaz
fi

if [[ -z $VAR || $VAR = 'b' ]]; then
    # ¿Fallará?
fi

echo "Salida por pantalla" && echo "Hola"

cat /nonexistentfile || echo "No existe"
```

Estructuras condicionales

```
● ● ●

# Declaramos una variable vacía
$ var=''
# y comparamos su contenido con ''
$ [ $var = '' ] && echo True
-bash: [: =: unary operator expected
$ [ "$var" = '' ] && echo True
True
$ [[ $var = '' ]] && echo True
True
#####
# ¿Qué sucede aquí?
#####

$ var=
$ [ "$var" < a ] && echo True
-bash: a: No such file or directory
$ [ "$var" \< a ] && echo True
True
$ [[ $var < a ]] && echo True
True
#####
$ var=a
$ [ "$var" > b ] && echo True || echo False
True
$ [[ "$var" > b ]] && echo True || echo False
False
#####
$ [[ abc = a* ]] && echo True || echo False
True
$ [[ abb =~ ab+ ]] && echo True || echo False
True
```

[] o [[]], ¿cuál usar y cuál es mejor?

Hasta ahora, siempre hemos trabajado con los condicionales y las expresiones “if” usando un único par de corchetes. Sin embargo, en el ejemplo anterior hemos usado dos pares de corchetes.

En el estándar POSIX, no se recoge la funcionalidad de los corchetes dobles, por lo que en principio no tiene compatibilidad. Sin embargo, los encontramos directamente en las consolas bash, zsh, yash, etc. ¿Qué ventajas ofrece?

Por una parte, es mucho más seguro y funciona “como esperas que funcione”. Por otro lado, soluciona muchos de los problemas que hemos detectado anteriormente.

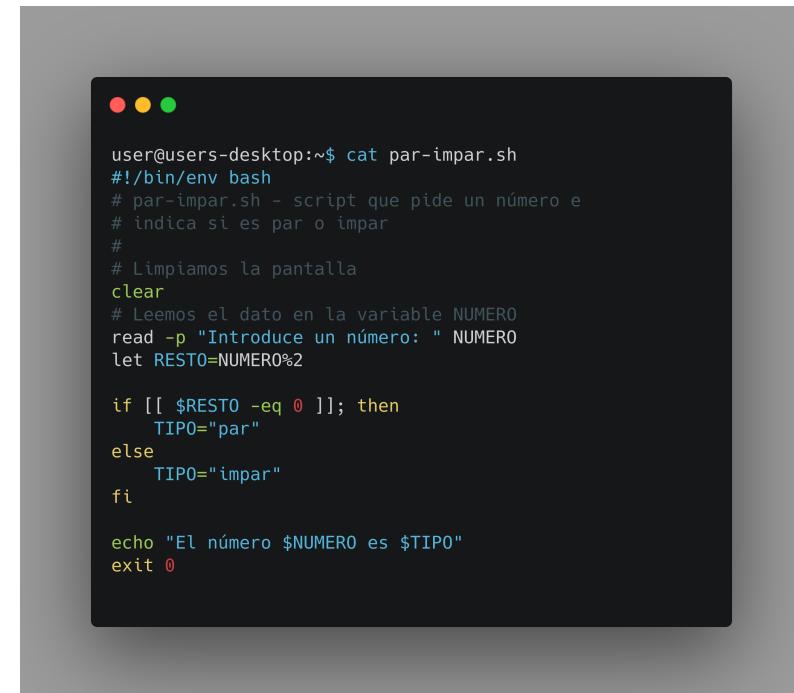
Estructuras condicionales

A lo largo del temario, hemos visto que una forma de pedirle datos al usuario es mediante el comando `read` o bien mediante parámetros. Vamos a explorar un poco más cómo funciona este último.

`read` es un comando el cual detendrá la ejecución del *script* hasta que el usuario introduzca un valor por teclado.

Los parámetros que recibe son variados, pero destacan:

- `-p`: muestra un mensaje por pantalla.
- `-s`: no muestra por pantalla lo que escribe el usuario.
- `-n`: lee ‘n’ caracteres y continúa la ejecución, sin necesidad de pulsar Enter. Solo funciona en bash.



```
user@users-desktop:~$ cat par-impar.sh
#!/bin/env bash
# par-impar.sh - script que pide un número e
# indica si es par o impar
#
# Limpiamos la pantalla
clear
# Leemos el dato en la variable NUMERO
read -p "Introduce un número: " NUMERO
let RESTO=NUMERO%2

if [[ $RESTO -eq 0 ]]; then
    TIPO="par"
else
    TIPO="impar"
fi

echo "El número $NUMERO es $TIPO"
exit 0
```

Estructuras condicionales

```
user@users-desktop:~$ cat capicua.sh
#!/bin/env bash
# capicua.sh - indica si un número de 3 cifas
# es capicúa o no
# Limpiamos la pantalla
clear
read -n3 -p "Indica un número entre 100 y 999: " NUMERO
echo # Línea vacía
if [[ $NUMERO -lt 100 ]]; then
    echo "El número es menor que 100..."
    exit 1
else
    C1=$(echo $NUMERO | cut -c1)
    C3=$(echo $NUMERO | cut -c3)
    if [[ $C1 = $C3 ]]; then
        echo "El número $NUMERO es capicúa"
    else
        echo "El número $NUMERO no es capicúa"
    fi
fi
exit 0
```

Estructuras condicionales

```
$ cat case-example.sh
#!/bin/env bash

# Suponemos que existe una variable VAR que
# contiene un valor que queremos filtrar.
# La estructura de un "case" es la siguiente:
case VAR in
    valor1)
        <acciones si VAR = valor1>
        ;;
    valor2)
        <acciones si VAR = valor2>
        ;;
    *)
        <acciones si VAR != valor1 && VAR != valor2>
        ;;
esac
$ cat cp.sh
#!/bin/env bash
read -n3 -p "Primeros tres dígitos del C.P.: " CP
echo

case $CP in
    280)
        echo "Madrid capital";;
    110)
        echo "Cádiz capital";;
    114)
        echo "Jerez de la Frontera";;
    *)
        echo "Ese código postal no está contemplado";;
esac
```

El `if` - `elif` - `else` otorga una flexibilidad muy alta a los *scripts* que estamos realizando, pero ya hemos descubierto qué sucede si queremos contemplar múltiples condiciones y solo realizar alguna si no se cumplen las anteriores: el tamaño del *script* empieza a crecer así como su manejabilidad.

Es por eso por lo que existe la sentencia “`case`” en bash. El funcionamiento es el siguiente: *dada una variable, se selecciona la opción que es igual que el valor de la variable*.

Esto se puede apreciar con más detalle en el ejemplo de la izquierda.

Estructuras condicionales

El comando “case” no se limita únicamente a comprobar si un valor equivale a otro que hayamos definido por cada uno de los casos, sino que va “más allá”.

case se puede mezclar con los comodines que hemos visto anteriormente, permitiendo que para un mismo caso haya varios puntos de entrada.

Esto se consigue usando expresiones regulares, las cuales son interpretadas directamente por bash cuando se está ejecutando la estructura case.

En la derecha se muestran varios ejemplos de cómo se pueden filtrar múltiples casos a la vez:

```
$ cat cp-2.0.sh
#!/bin/env bash
read -n5 -p "Introduzca el código postal: " CP
echo

case $CP in
  110*)
    echo "Cádiz capital";;
  280*)
    echo "Madrid capital";;
  114*)
    echo "Jerez de la Frontera";;
  [1110-1111]*)
    echo "San Fernando";;
  *)
    echo "Ese código no está contemplado";;
esac

$ cat vowels.sh
#!/bin/env bash
read -n1 -p "Introduce una letra: " LETTER
echo

case $LETTER in
  a|e|i|o|u|A|E|I|O|U)
    echo "$LETTER es una vocal";;
  *)
    echo "$LETTER es una consonante";;
esac
```

Estructuras condicionales

0	El mono	4	La rata	8	El dragón
1	El gallo	5	El buey	9	La serpiente
2	El perro	6	El tigre	10	El caballo
3	El cerdo	7	El conejo	11	La cabra



```
$ ./horoscopo.sh
¿En qué año naciste?: 1998
Si naciste en 1998 te corresponde "El tigre" según el horóscopo chino
```

Estructuras condicionales