Scripting en Bash

Introducción a los sistemas operativos

Shell interactiva

- El shell Bash se usa comúnmente de forma **interactiva**
- Permite ingresar y editar comandos y luego ejecutarlos cuando presiona la tecla Retorno .
- Muchos sistemas operativos basados en Unix y similares a Unix usan Bash como su shell predeterminado (en particular, Linux y macOS).
- Podemos obtener la salida "Hola mundo" ejecutando este comando en una terminal Shell

```
echo "Hola mundo"
```

Shell no interactiva

- Es posible crear un fichero ejecutable que contenga una secuencia de comandos, este fichero se denomina script.
- Cuando creamos un script bash usaremos la extensión .sh
- La primera línea del script indicará que se trata de un script bash de la siguiente manera

```
#! /bin/bash
```

- El script se irá ejecutando cada comando de arriba abajo hasta llegar al final del script

hola_mundo.sh

```
#! /bin/bash

echo "Hola Mundo"
```

Ejecutar hola_mundo.sh

Para poder ejecutar un script en UNIX debemos darle permisos de ejecución mediante el comando chmod

```
chmod +x hola_mundo.sh
```

Una vez dados los permisos podemos ejecutar el script de la siguiente manera

```
./hola_mundo.sh
```

Ejercicio

1. Escribe un script llamado primero.sh que imprima la frase "Mi primer script de bash" en una primera línea y después que imprima en otra línea tu nombre y apellidos.
2. Haz que el script sea ejecutable
3. Ejecuta el script y comprueba que los dos mensajes aparecen en pantalla

Variables

- Las variables en BASH no tienen tipo por lo que para declararlas simplemente hay que asignarles un valor

```
variable=23
```

- Cuando queramos acceder al valor de la variable deberemos poner el símbolo $ delante del nombre:

```
echo $variable # Esto sacará por pantalla el valor 23
echo variable # Esto sacará por pantalla el texto "variable"
```

- Si lo que queremos es asignar o reasignar un valor a una variable no hay que usar el signo $

```
variable="Hola mundo" # No necesito el $ porque estoy
                      # asignando un valor a la variable
echo $variable        # Aquí sí hace falta el $ porque estoy accediendo al
                      # valor que hay guardado en la variable
```

Ejercicio

1. Escribe un script llamado variables.sh que declare una variable que contenga el texto "Mi primer script de bash con variables"
2. Declara otra variable que contenga tu nombre y apellidos
3. Imprime en pantalla el contenido de la primera variable.
4. Imprime en pantalla el contenido de la otra variable.
5. Haz que el script sea ejecutable.
6. Ejecuta el script y comprueba que los dos mensajes aparecen en pantalla.

Argumentos

- Es posible pasar argumentos cuando se ejecuta un script bash
- Para acceder al valor de cada argumento utilizaremos el símbolo $ y el número del argumento, empezando por 1.
- Por ejemplo, si quisiéramos imprimir en pantalla el primer argumento haríamos:

```
echo $1
```

- Podemos pasar argumentos a nuestro script poniéndolos después de la llamada separados por un espacio

```
./mi_script.sh argumento1 argumento2
```

Ejercicio

1. Escribe un script llamado argumento.sh que declare una variable que contenga el texto "Mi primer script de bash con argumentos"
2. Imprime en pantalla el contenido de la variable.
3. Imprime en pantalla el contenido del primer argumento.
4. Haz que el script sea ejecutable.
5. Ejecuta el script pasándole tu nombre y apellidos como argumento, ten en cuenta que deberás usar comillas para que no considere cada palabra como un argumento distinto.

Ejercicio

1. Escribe un script llamado argumento_variables.sh que declare una variable que contenga el texto "Mi primer script de bash con argumentos"
2. Crea otra variable que guarda el valor del primer argumento
3. Imprime en pantalla el contenido de la primera variable.
4. Imprime en pantalla el contenido de la segunda variable.
5. Haz que el script sea ejecutable.
6. Ejecuta el script pasándole tu nombre y apellidos como argumento, ten en cuenta que deberás usar comillas para que no considere cada palabra como un argumento distinto.

Ejercicio

1. Escribe un script llamado argumentos.sh
2. Crea una variable que guarda el valor del primer argumento
3. Crea otra variable que guarde el valor del segundo argumento
4. Imprime en pantalla el contenido de la primera variable.
5. Imprime en pantalla el contenido de la segunda variable.
6. Haz que el script sea ejecutable.
7. Ejecuta el script pasándole tu nombre y apellidos como primer argumento y el texto "Script de argumentos" como segundo.

Exit status

- Por convención en los sistemas Unix cuando un proceso ha ido bien debe devolverse el valor 0
- Esto quiere decir que 0 es el equivalente a true en bash y para el false tenemos el 1

Estructura if

- Para comprobar una condición debemos ponerla entre dobles corchetes [[condición]]
- Es importante que se respeten los espacios entre los corchetes y la condición
- Después de la condición debe haber un salto de línea y abrir el bloque condicional con un then
- Para terminar la estructura pondremos un fi

If … then

```
if [[ condicion ]]
then
    # si es verdad (0)
fi
```

Operadores de comparación numéricos

- -eq: comprueba si lo que hay a izquierda y derecha es igual
- -ne: comprueba si lo que hay a izquierda y derecha es distinto
- -gt: comprueba si lo que hay a la izquierda es mayor que lo de la derecha
- -ge: comprueba si lo que hay a la izquierda es mayor o igual que lo de la derecha
- -lt: comprueba si lo que hay a la izquierda es menor que lo de la derecha
- -le: comprueba si lo que hay a la izquierda es menor o igual que lo de la derecha

Operadores de comparación de cadenas

- '==' (o simplemente '='): Devuelve true cuando ambas cadenas son iguales y false en caso contrario.

```
[[ $VAR == "value" ]]
```

- '!=': Devuelve true cuando ambas las cadenas son distintas y false en caso de ser iguales.

```
[[ $VAR == "value" ]]
```

Uso de los operadores de comparación

```
NUM1=10
NUM2=20
if [[ $NUM1 -gte $NUM2 ]]
then
    # si es verdad (0)
fi
```

Ejercicio

1. Crea un script llamado mayor.sh
2. Declara una variable que guarde el valor del primer argumento
3. Declara otra variable que guarde el valor del segundo argumento
4. Usando un if haz que el programa imprima el texto "es mayor" si la primera variable es mayor que la segunda

Ejercicio

1. Crea un script llamado iguales.sh
2. Declara una variable que guarde el valor del primer argumento
3. Declara otra variable que guarde el valor del segundo argumento
4. Usando un if haz que el programa imprima el texto "son iguales" si la primera variable es igual que la segunda

If … then … else

```
if [[ condicion  ]]
then
    # si es verdad
else
    # si no es verdad
fi
```

Ejercicio

1. Crea un script llamado distintos.sh
2. Declara una variable que guarde el valor del primer argumento
3. Declara otra variable que guarde el valor del segundo argumento
4. Usando un if haz que el programa imprima el texto "es distinto" si la primera variable es distinta que la segunda y "son iguales" en caso contrario

Ejercicio

1. Crea un script llamado menor.sh
2. Declara una variable que guarde el valor del primer argumento
3. Declara otra variable que guarde el valor del segundo argumento
4. Usando un if haz que el programa imprima el texto "es menor" si la primera variable es menor que la segunda y "no es menor" en caso contrario

If … then … elif … then … else

```
if [[ condicion1  ]]
then
    # Si condicion1 es verdadero
elif [[ condicion2 ]]
then
    # Si las anteriores condiciones son falsas
    # y condicion2 verdadera
else
    # Si ninguna de las anteriores es verdadera
fi
```

Ejercicio

1. Crea un script llamado compara.sh
2. Declara una variable que guarde el valor del primer argumento
3. Declara otra variable que guarde el valor del segundo argumento
4. Usando un if haz que el programa imprima el texto "es mayor" si la primera variable es mayor que la segunda, "son iguales" en caso de que sean iguales y "es menor" si ninguno de las anteriores condiciones se ha cumplido

Operadores lógicos

- !: es la negación de una condición
- &&: nos permite combinar dos condiciones con el operador AND
  - True && True -> True
  - True && False -> False
  - False && True-> False
  - False && False -> False
- ||: nos permite combinar dos operaciones con el operador OR
  - True || True -> True
  - True || False -> True
  - False || True-> True
  - False || False -> False

Operadores lógicos

```
if [[ $NUM1 -gt 0 && $NUM1 -lt 100 ]]
then
    echo Es mayor de 0 y menor de 100
elif [[ $NUM1 -lt 0 || $NUM1 -gt 1000 ]]
then
    echo Es menor que 0 o mayor de 1000
else
    echo Es mayor o igual que 100 y menor o igual que 1000
fi
```

Ejercicio

1. Crea un script llamado operadores_logicos.sh
2. Declara una variable que guarde el primer argumento
3. Si el valor de la variable es un numero par y es positivo imprime el texto "Par positivo"
4. Si es negativo o impar imprime "Negativo o impar"
5. Si es cero imprime "Es cero"

For loop

```
for V in <lista>
do
        # comandos del loop
done
```

Lista de argumentos

- Hemos visto que para acceder a un argumento podemos utilizar su posición, por ejemplo para acceder al primero haríamos $1.
- También es posible cargar todos los argumentos como una lista utilizando la construcción **$@**.
- Gracias a esto, podemos utilizar un bucle para recorrer todos los argumentos del script
- Para saber cuántos argumentos se han pasado se puede utilizar la variable **$#**

Ejercicio

1. Crea un script llamado imprime_argumentos.sh
2. Imprime en pantalla el número de argumentos
3. Declara una variable que guarde la lista de argumentos
4. Crear un bucle for que recorra los argumentos
5. Imprime cada argumento por pantalla

Ejercicio

1. Crea un script llamado crea_directorios.sh
2. Declara una variable que guarde la lista de argumentos
3. Crear un bucle for que recorra los argumentos
4. Crea un directorio con el valor de cada argumento. Para crear un directorio usa el siguiente comando

mkdir -p <nombreDirectorio>

Manejo de la Shell interactiva

- La shell interactiva nos permite ejecutar comandos para interactuar con el sistema.
- La terminal nos puede servir como lugar de pruebas d los comandos que se van a usar en el script.
- Si no estamos seguros de cómo usar un comando podemos acceder a información del mismo de dos maneras:
  - Help: ejecutando <comando> --help, nos da un resumen de las opciones disponibles
  - Man: ejecutando man <comando>, se abre el manual del comando, mucho más detallado que el help

Ejercicio

1. Ejecuta man ls
2. ¿Para qué sirve el comando ls?
3. ¿Cuáles son los parámetros más importantes (salen con la opción --help)
4. Haz algunas pruebas en la terminal para entender las diferentes opciones

Ejercicio

1. Ejecuta man mkdir
2. ¿Para qué sirve el comando mkdir?
3. ¿Cuáles son los parámetros más importantes (salen con la opción --help)
4. Haz algunas pruebas en la terminal para entender las diferentes opciones

Ejercicio

1. Ejecuta man rm
2. ¿Para qué sirve el comando rm?
3. ¿Cuáles son los parámetros más importantes (salen con la opción --help)
4. Haz algunas pruebas en la terminal para entender las diferentes opciones (pero con mucho cuidado)

Ejercicio

1. Ejecuta man cat
2. ¿Para qué sirve el comando cat?
3. ¿Cuáles son los parámetros más importantes (salen con la opción --help)
4. Haz algunas pruebas en la terminal para entender las diferentes opciones

Sustitución de comandos

- Existen determinados comandos que al ser ejecutados producen una salida textual.
- Si queremos capturar esa salida y almacenarla en una variable podemos utilizar el mecanismo de sustitución de comandos.
- Por ejemplo, el comando cat saca por pantalla el contenido de un fichero. Podríamos querer guardar ese contenido en una variable de esta forma

  FILE_CONTENT=$(cat <nombre_archivo>)
- Cuando la salida de un comando se captura con este mecanismo ya no aparece en pantalla.

Ejercicio

1. Crea un fichero carpetas.txt con un nombre de carpeta por cada línea
2. Crea un script llamado crea_carpetas.sh
3. El script cargará en una variable FOLDERS el contenido de un fichero llamado carpetas.txt que debe contener una serie de nombres de carpetas cada una en una línea (utiliza el comando cat para extraer los nombres del archivo)
4. Usando un bucle recorre la lista FOLDERS y crea una carpeta por cada elemento usando el comando mkdir –p <carpeta>

Escribir la salida de un comando a un fichero

```
echo "Hola mundo" > hola.txt
```

```
echo "Adiós mundo" >> hola.txt
```

```
ls -lah > ls_out.txt
```

Expansión de expresiones aritméticas

- Debido a que no existen tipos de datos en Bash es necesario marcar las expresiones aritméticas que pongamos en nuestros scripts.
- Existen varias formas de escribir una expresión aritmética:
  - Entre backticks (en combinación con la palabra expr):

    z=`expr $z + 3`
  - Entre dobles paréntesis: la versión con backticks ya no se utiliza

    ((z= z + 3))
  - Después de la palabra let:

    let z=z + 3
- En caso de que queramos utilizar el resultado de una expresión para algo haremos:

  echo $((z +3))

Ejercicio

1. Crear un script llamado suma.sh
2. Utiliza un bucle para calcular la suma de todos los argumentos. Recuerda que es necesario realizar la expansión de expresiones aritméticas
3. Imprime en pantalla el resultado de la suma

Funciones

- Es posible definir bloques de código invocables a través de la definición de funciones.
- Las funciones se corresponden con la misma idea que los métodos en Java
- Al igual que en Java, las funciones pueden recibir parámetros, pero aquí no se declaran entre paréntesis, sino que son accedidos a través del $1, $2, etc...
- También podemos usar $@ para acceder a la lista de parámetros y $# para sacar el número de parámetros que se han pasado

Funciones

```
#! /bin/bash

my_func (  )
{
        echo "Recibidos los mensajes "$1" y "$2
}

my_func "Hola mundo" "Adiós mundo"
```

Ejercicio

1. Crear un script llamado funciones.sh
2. Define una función que muestre en pantalla el mensaje "Invocada la función con los siguientes parámetros: " y en cada línea uno de los parámetros recibidos por la función
3. Invoca la función pasándole los valores "val1", "val2" y "val3"

Scripts con opción de ayuda

- Cuando creamos un script, es conveniente proporcionar una opción de ayuda para mostrar explicar cómo se usa el script.
- Para ello, es habitual crear una función que lo único que hace es imprimir mensajes explicatorios.
- Esta función será invocada cuando el primer argumento del script tome el valor --help

Ejercicio

1. Crea un script llamado mkdirs_ayuda.sh
2. Define una función llamada help que imprime en pantalla un texto de ayuda que explique cómo usar este script.
3. El script debe ejecutar la funcion help cuando el primer argumento toma el valor –help
4. Si no se pasa el argumento –help, debe funcionar igual que el script crea_carpetas.sh

Sustitución de comandos y funciones

Si una función imprime mensajes utilizando echo, estos pueden ser capturados utilizando el mecanismo de sustitución de comandos que hemos visto antes.

```
#! /bin/bash

my_func (  )
{
        echo "Hola mundo"
}

MESSAGE=$( my_func )
echo "La función ha devuelto: " $MESSAGE
```

Referencia de comandos bash

Comando	Descripción
ls	Listado de ficheros y directorios
mkdir	Crea directiorios
touch	Crea ficheros
cat	Muestra el contenido de un fichero
echo	Imprime en pantalla
history	Muestra el historial de comandos ejecutados por el usuario
wc	Cuenta líneas, palabra y/o bytes
grep	Realiza búsquedas de texto sobre un fichero o sobre la entrada
cd	Navegación por el sistema de archivos
cp	Copia un archivo
mv	Mueve un archivo
rm	Elimina archivos y directorios

Piping de comandos

- En sistemas UNIX es posible "conectar" directamente la salida de un comando a la entrada de otro a través del uso de pipes (|)
- La forma de hacerlo es poner un comando con sus parámetros seguido de un pipe (|) y luego otro comando dejando algún argumento sin pasar para que se rellene con la salida del anterior comando
- Es posible encadenar múltiples comandos en una sola línea
- Los comandos se irán ejecutando de izquierda a derecha hasta llegar al último

Ejercicios de piping (I)

1. Muestra el historial de ejecuciones del comando cat.
2. Cuenta el número de ejecuciones del comando cat que hay en el historial de ejecuciones (history)
3. Crea un fichero de texto llamado mi_texto.txt con varias líneas, algunas deben contener la palabra "Hola". Muestra todas las líneas del fichero de texto que contengan la palabra "Hola".
4. Cuenta las líneas que contienen la palabra "Hola" del fichero mi_texto.txt.
5. Muestra todas las líneas del fichero mi_texto.txt que no contengan la palabra "Hola".
6. Cuenta las líneas que no contienen la palabra "Hola" del fichero mi_texto.txt.

Ejercicios de piping (II)

1. Lee el fichero mi_texto.txt, quédate con las líneas que tienen la palabra "Hola" y guárdalas en un fichero llamado hola.txt. El fichero debe crearse si no existiese y sobreescribirse en caso de ya existir.
2. Lee el fichero mi_texto.txt, quédate con las líneas que NO tienen la palabra "Hola", ordénalas en orden alfabético y guárdalas en un fichero llamado hola.txt. Si el fichero ya existiese las líneas deben añadirse al final del mismo.
3. Muestra el historial de ejecuciones del comando cat, haz que salgan en orden alfabético.
4. Muestra todas las líneas del fichero de texto que no contengan la palabra "Hola", haz que salgan en orden alfabético inverso.

Scripts con piping

1. Crea un script que lea de un fichero, se quede con las líneas que contienen un texto y guarda el resultado en otro fichero. Este script debe recibir tres argumentos: el fichero que se lee, el texto a buscar y el fichero de salida.
2. Crea un script que lea de un fichero, se quede con las líneas que NO contienen un texto, las ordena en orden alfabético y guarda el resultado en otro fichero. Este script debe recibir tres argumentos: el fichero que se lee, el texto a buscar y el fichero de salida. Si el fichero ya existiese las líneas deben añadirse al final del mismo.
3. Crea un script que muestre el historial de ejecuciones del comando cat, haz que salgan en orden alfabético.
4. Crea un script que muestree todas las líneas de un fichero de texto que no contengan un texto, haz que salgan en orden alfabético inverso. Este script debe recibir dos argumentos: el fichero y el texto a buscar.

Scripts interactivos

- Es posible hacer scripts interactivos que vayan pidiendo al usuario los datos que se vayan necesitando
- Para pedirle un dato al usuario se utiliza el comando read

  read –p "<mensaje pidiendo el dato>" VARIABLE
- Una vez el usuario ha introducido el dato y ha pulsado ENTER el texto introducido se cargará en VARIABLE

Ejercicio

Crear un script llamado busqueda_interactiva.sh que reconozca los siguientes comandos:

- count: Contar líneas que contengan un texto en un fichero. El script pedirá al usuario el fichero del que se quiere leer y el texto que se quiere buscar.
- remove: Elimina un fichero. El script pedirá al usuario el fichero que se quiere eliminar
- list: Lista los ficheros y directorios en una ruta. El script pedirá al usuario la ruta en la que se quieren listar los ficheros