1a) Shell Script es un archivo con una secuencia de comandos de un determinado Shell (en nuestro caso Bash). Contiene:

- Comandos
- Manejo de variables (no es fuertemente tipado)
- Comentarios: líneas que comienzan con #
- Estructuras de control: if, case, for, while
- Declaración de funciones
- Puede usar código definido en otros scripts (comando source).

1b) Es ideal para automatizar tareas que implican usar comandos. La ejecución de comandos es directa (con su nombre basta). Es muy simple guardar el resultado de la ejecución de un comando, y no requiere la instalación de ningún intérprete o compilador (todo GNU/Linux tiene al menos un shell). GNU/Linux expone gran parte del sistema como archivos, y es muy fácil manipular archivos desde un Shell.

1c) Los scripts no deben compilarse, porque los interpreta el Shell.

2)

- echo: Imprimir texto. Ej: echo “Hola mundo”

- read: Leer una línea desde la entrada estándar (como el teclado) sobre una variable. Ej: read var

Los comentarios se indican con #. Ej: # Este es un comentario.

Las variables no precisan declaración, sino que se las utiliza directamente. Además, bash no es fuertemente tipado, por lo

que las variables pueden cambiar su tipo durante la ejecución de nuestros scripts.

Las variables tienen básicamente dos tipos: String y Array

Las asignaciones tienen la forma (notar que no hay espacios entre el signo "=" y los operandos):

- variable=valor

Para referenciar su valor, se utiliza el símbolo "$", y opcionalmente "{}". Ej:

```
# Imprime: Mi nombre es Pepe
nombre=Pepe
echo "Mi nombre es $nombre"
```

4) Se pueden acceder a los parámetros que se envían al script al momento de su invocación por medio de los comandos $1..$9 (del $1 al $9). $1 hace referencia al primer parámetro, $2 al segundo... y $9 al noveno. El comando $0 no está incluido porque siempre contiene el nombre del script que se está ejecutando.

Otras variables útiles:

- $#: Contiene el número de parámetros que se han pasado al script.

- $*: Contiene una cadena de texto con todos los parámetros pasados al script, excepto el parámetro 0 ó $0, que es el nombre del script.

Github

- $?: Contiene el estado de salida del último comando ejecutado. Suele usarse para conocer si ciertos comandos realizaron la tarea esperada con éxito o si hubo algún error, y así poder tomar las acciones necesarias en cada caso.

- $HOME: Indica el directorio Home del usuario actual.

5) El comando "exit N" se utiliza para detener la ejecución de un script y salir del mismo con un estado de salida N específico. Si no se agrega un valor N, el estado de salida es el del último comando ejecutado.

Algunos estados de salida de comandos de Linux y su significado:

- 0: Éxito.
- 1: Error General.
- 2: Mal uso de las funciones integradas de shell.
- 126: Comando no encontrado.
- 127: Comando no encontrado.
- 128: Argumento invalido para salir.
- 130: Script terminado por Ctrl+C.
- 255: Estado de salida fuera de rango (los códigos de salida suelen estar limitados al rango 0-255).

6) El comando "expr arg1 op arg2" permite realizar operaciones aritméticas. "expr" es el comando. "arg1" y "arg2" son los argumentos de la operación. "op" es el operando de la operación. Pueden realizarse operaciones de suma, resta, producto, división, resto...

Github

7) El comando "test" puede utilizarse para verificar si un archivo o directorio existe, para verificar el tamaño de una cadena de caracteres, o para comparar números. Siempre retorna un valor de verdad.

- Ej: if test -f archivo; then echo "Existe archivo"; else echo "No existe archivo"; fi

El "if test -f archivo" también puede escribirse como "if [-f archivo]". Son equivalentes.

Opciones de comando "test" para archivos:

- test -e arch: El archivo "arch" existe.
- test -s arch: El archivo "arch" existe y tiene tamaño mayor que cero.
- test -f arch: El archivo "arch" existe y es un archivo regular.
- test -d arch: El archivo "arch" existe y es un directorio.
- test -r arch: El archivo "arch" existe y con derechos de lectura.
- test -w arch: El archivo "arch" existe y con derechos de escritura.
- test -x arch: El archivo "arch" existe y con derechos de ejecución.

Github

Opciones de comando "test" para cadenas de caracteres:

- test -z cadena: El tamaño de cadena es cero.
- test -n cadena: El tamaño de cadena es mayor que cero.
- test cadena: La cadena "cadena" tiene un tamaño mayor que cero.
- test c1 = c2: Las cadenas "c1" y "c2" son idénticas.

Opciones del comando test para evaluar condiciones lógicas:

- n1 -eq n2: "n1" y "n2" son iguales.
- n1 -ne n2: "n1" y "n2" no son iguales.
- n1 -gt n2: "n1" es mayor que "n2".
- n1 -ge n2: "n1" es mayor o igual a "n2".
- n1 -lt n2: "n1" es menor que "n2".
- n1 -le n2: "n1" es menor o igual a "n2".

8)

- if: Decisión. Si la condición es verdadera, ejecuta el "block1", sino el "block2". "fi" determina el final de la estructura.
  Ej:

Github

```
if [ condition ]
then
    block1
else
    block2
fi
```

- case: Selección. Dependiendo el valor de la variable "var", se ejecutará un bloque u otro. Si el valor de la variable no se corresponde con ningún valor, se ejecutará el bloque "*". "esac" determina el final de la estructura. Ej:

```
case $variable in
"valor 1")
    block
;;
"valor 2")
    block
;;
*)
    block
;;
esac
```

- while: Iteración. Se ejecutará el bloque de código hasta que se cumpla una condición de corte. "done" determina el final de la estructura.
  Hay 2 formas: While y Until

- While:

```
while [ condition ] # Mientras se cumpla la condición.
do
    block
done
```

- Until:

```
until [ condition ] # Mientras NO se cumpla la condición.
do
    block
done
```

- for: Repetición. Se ejecutará el bloque de código una cantidad establecida de veces. "done" determina el final de la estructura. Hay 2 formas:

  - Contador:

```
for ((i=0; i < 10; i++))
do
    block
done
```

  - Lista de valores:

```
for i in valor1 valor2 valor3 .. valorN;
do
    block
done
```

[Github]

- select: Utilizado para la creación de menús. Se creará una lista con las opciones ingresadas luego del "in" en el encabezado. El resultado será guardado en la variable del encabezado. "done" determina el final de la estructura. Ej:

```
select marca in Samsung Sony iPhone Symphony
Walton
    do
        echo "Se eligió la marca $marca"
    done
```

Al ejecutarse, se mostrará en pantalla un menú de selección de este tipo:

1) Samsung
2) Sony
3) iPhone
4) Symphony
5) Walton

Luego, si se ingresa 2, por ejemplo, se mostrará en pantalla el siguiente mensaje:

Se eligió la marca Sony

9)

- break: "break [n]" corta la ejecución de n niveles de loops. Ejs:

```
while [ condition ] #Loop 1.
do
    while [ condition ] #Loop 2.
    do
        break #Esto cortará solo el loop 2.
    done
done

while [ condition ] #Loop 1.
do
    while [ condition ] #Loop 2.
    do
        break 2 #Esto cortará el loop 2 y el loop 1.
    done
done
```

- continue: "continue [n]" salta a la siguiente iteración del enésimo loop que contiene esta instrucción. Ej:

```
i=0
while [[ $i -lt 5 ]]; do
    # Incrementa i en 1.
    ((i++))

    # Si i es igual a 2...
    if [[ "$i" == '2' ]]
    then
        # Continua a la siguiente iteración del while.
        continue
    fi
    # Imprime en pantalla el valor de i.
    echo "Number: $i"
done
```

Github

Imprimirá:

Number: 1
Number: 3
Number: 4
Number: 5

10) No existen tipos de datos. Una variable de bash puede contener un número, un carácter o una cadena de caracteres. No se necesita declarar una variable, sino que esta se creará sólo con asignarle un valor a su referencia. Por lo tanto, Shell Script no es fuertemente tipado.

- Arreglos: Se pueden definir de 2 maneras distintas:
    - arreglo_a = () # Se crea vacío.
    - arreglo_b = (1 2 3 5 8 13 21) # Inicializado.
- Para asignar un valor en una posición concreta:
    - arreglo_b[2] = spam
- Para acceder a un valor en concreto (las llaves son obligatorias):

```
# Imprimo en pantalla el contenido de esa posición.
echo ${arreglo_b[2]}
# Me guardo el contenido de esa posición en la variable "copia".
copia = ${arreglo_b[2]}
```

Github

- Para acceder a todos los valores del arreglo:
    - echo ${arreglo[*]}
- Para acceder al tamaño del arreglo:
    - ${#arreglo[*]}
- Borrado de un elemento (reduce el tamaño del arreglo pero no elimina la posición, solamente la deja vacía):
    - unset arreglo[2]

11) Las funciones permiten modularizar el comportamiento de los scripts. Se pueden declarar de 2 formas:

- function nombre { block }
- nombre() { block }

Las funciones reciben parámetros en las variables $1, $2, etc. Con la sentencia "return" se retorna un valor entre 0 y 255, y ese valor de retorno se puede evaluar mediante la variable "$?".

Github

Ej:

```
# Recibe 2 argumentos y devuelve:
# 1 si el primero es el mayor
# 0 en caso contrario

mayor()
{
    # “$*” cadena de texto con los parámetros pasados al script.
    echo "Se van a comparar los valores $*"
    if [ $1 -gt $2 ]; then # Si $1 es mayor que $2...
        echo "$1 es el mayor"
        return 1
    fi

    echo "$2 es el mayor"
    return 0
}

# Programa Principal.

mayor 5 6 # Invocación
echo $? # Imprime el exit Status de la función.
```

Github