1. **¿Qué es el Shell Scripting? ¿A qué tipos de tareas están orientados los scripts? ¿Los scripts deben compilarse? ¿Por qué?**

El Shell scripting es el proceso de escribir un Script o un programa en un lenguaje de comandos (Shell) que se ejecuta en sistema operativo, generalmente basado en Unix o Linux

Los **scripts de Shell** están orientados a automatizar una gran variedad de tareas, especialmente aquellas que implican la administración del sistema o el manejo de procesos repetitivos.

1. Administración de sistemas.
2. automatización de procesos.
3. Automatización de tareas programados.
4. Desarrollo de software.

**Los scripts de Shell no necesitan ser compilados**. Los scripts son **interpretados** en lugar de compilados. Esto significa que el código se ejecuta línea por línea directamente por un intérprete (como el Shell Bash) en lugar de ser transformado en un archivo binario ejecutable como ocurre con los lenguajes de programación compilados (por ejemplo, C o C++).

**2. Investigar la funcionalidad de los comandos echo y read.**

En Bash, los comandos read y echo se utilizan para interactuar con el usuario o el sistema a través de la entrada y salida estándar, respectivamente.

1. **Read**

El comando **read** se usa para leer la entrada del usuario desde el teclado o desde un archivo. Es decir, permite almacenar en una variable el valor que el usuario introduce.

1. **Echo**

El comando echo se usa para imprimir texto o valores en la salida estándar (normalmente la pantalla). Es una manera de mostrar información, ya sea estática o dinámica, en la terminal.

1. Completado
2. **Parametrización**

En un script de Bash, puedes acceder a los parámetros (o argumentos) que se pasan al script al momento de su invocación usando variables especiales. Aquí te explico cómo funcionan estas variables:

**Variables de parámetros**

* $0: El nombre del script.
* $1: El primer parámetro.
* $2: El segundo parámetro.
* $3: El tercer parámetro.
* ...: Puedes seguir así hasta $9 para los primeros nueve parámetros.
* ${10}, ${11}, ...: Para acceder a parámetros más allá del noveno, debes usar llaves: ${10} para el décimo, ${11} para el undécimo, etc.
* $#: El número total de parámetros pasados al script.
* $@: Todos los parámetros pasados al script, tratados como una lista (cada parámetro se considera individualmente).
* $\*: Todos los parámetros pasados al script, pero tratados como una sola cadena.

Las variables especiales en un script de Bash que mencionas ($#, $\*, $?, y $HOME) tienen significados específicos y son útiles para obtener información sobre los parámetros, el estado de la ejecución de comandos y la ubicación del directorio personal del usuario. Aquí te explico cada una:

**1. $#**

* **Descripción**: Contiene el número total de parámetros (argumentos) pasados al script.
* **Ejemplo**: Si ejecutas un script con tres parámetros, $# devolverá 3.

**2. $\***

* **Descripción**: Contiene todos los parámetros pasados al script como una sola cadena. Los parámetros están separados por un espacio.
* **Ejemplo**: Si ejecutas el script con los parámetros param1 param2, $\* devolverá param1 param2.

**3. $?**

* **Descripción**: Contiene el estado de salida del último comando ejecutado. Un valor de 0 indica que el comando se ejecutó con éxito, mientras que cualquier otro valor indica un error.
* **Ejemplo**: Si el último comando se ejecutó correctamente, $? devolverá 0. Si hubo un error, devolverá un número distinto de cero, que representa el código de error.

**4. $HOME**

* **Descripción**: Contiene la ruta al directorio personal del usuario que está ejecutando el script. Este valor es fijo y no depende de los parámetros pasados al script.
* **Ejemplo**: Si tu nombre de usuario es juan, $HOME devolverá /home/juan.

1. **Punto 5**

**¿Cuál es la funcionalidad de comando exit?**

El comando exit en un script de Bash o en la línea de comandos tiene varias funcionalidades clave:

**1. Salir del script**

* Cuando se usa en un script, exit finaliza la ejecución del script en el punto donde se encuentra. No se ejecutarán las líneas que vienen después de exit.

**2. Estado de salida**

* exit también permite especificar un **código de estado de salida**. Este código indica si el script terminó con éxito o si ocurrió un error. Por convención:
  + Un código de salida de 0 indica que el script se ejecutó correctamente.
  + Cualquier otro número (por ejemplo, 1, 2, etc.) indica que ocurrió un error.

**Parámetros de exit**

1. **Códigos de éxito**
   * **0**: Este valor indica que el script o comando se ejecutó correctamente sin errores. Es el valor por defecto si no se proporciona ningún argumento a exit.
2. **Códigos de error**
   * **1 a 255**: Estos valores representan distintos códigos de error. Puedes definirlos según la lógica de tu script. Cada número puede tener un significado específico dependiendo de la aplicación, pero comúnmente:
     + **1**: Error genérico.
     + **2**: Error relacionado con la sintaxis.
     + **126**: Comando invocado no ejecutable.
     + **127**: Comando no encontrado.
     + **128+n**: Comando terminado con un error de señal n.
     + **255**: Indica un error en el script, aunque puedes usar cualquier número hasta 255 para describir diferentes errores.
3. **El comando expr permite la evaluación de expresiones. Su sintaxis es: expr arg1 op arg2, donde arg1 y arg2 representan argumentos y op la operación de la expresión. Investigar qué tipo de operaciones se pueden utilizar**

**Operaciones Aritméticas**

En Bash se pueden utilizar las operaciones entre dos variables utilizando los signos que seguramente ya sabéis:

* "**+**": Para sumar valores.
* "**-**": Para restar valores.
* "**\***": Para multiplicar valores.
* "**/**": Para dividir valores.
* "**%**": Para calcular el resto de una división.

**La lógica de Bash**

Para el terminal, los caracteres que tienen un significado lógico en la comparación o evaluación de operaciones aritméticas y de archivos son:

**Evaluación de operaciones aritméticas**

* "**-lt**": Menor que
* "**-le**": Menor o igual que
* "**-eq**": Igual que
* "**-ge**": Mayor o igual que
* "**-gt**": Mayor que
* "**-ne**": Distinto que

1. **El comando “test expresión” permite evaluar expresiones y generar un valor de retorno, true o false. Este comando puede ser reemplazado por el uso de corchetes de la siguiente manera [ expresión ]. Investigar que tipo de expresiones pueden ser usadas con el comando test. Tenga en cuenta operaciones para: evaluación de archivos, evaluación de cadenas de caracteres y evaluaciones numéricas.**

**. Evaluación de archivos**

Se pueden usar varias expresiones para verificar propiedades de archivos:

* **Existencia de un archivo**:

test -e nombre\_archivo

[ -e nombre\_archivo ]

* **Archivo regular**:

test -f nombre\_archivo

[ -f nombre\_archivo ]

* **Directorio**:

test -d nombre\_directorio

[ -d nombre\_directorio ]

* **Archivo de enlace simbólico**:

test -L nombre\_archivo

[ -L nombre\_archivo ]

* **Archivo ejecutable**:

test -x nombre\_archivo

[ -x nombre\_archivo ]

* **Archivo vacío**:

test -s nombre\_archivo

[ -s nombre\_archivo ]

**2. Evaluación de cadenas de caracteres**

Las expresiones de prueba para cadenas incluyen:

* **Comprobar si una cadena está vacía**:

test -z "cadena"

[ -z "cadena" ]

* **Comprobar si una cadena no está vacía**:

test -n "cadena"

[ -n "cadena" ]

* **Comprobar igualdad de cadenas**:

test "cadena1" = "cadena2"

[ "cadena1" = "cadena2" ]

* **Comprobar desigualdad de cadenas**:

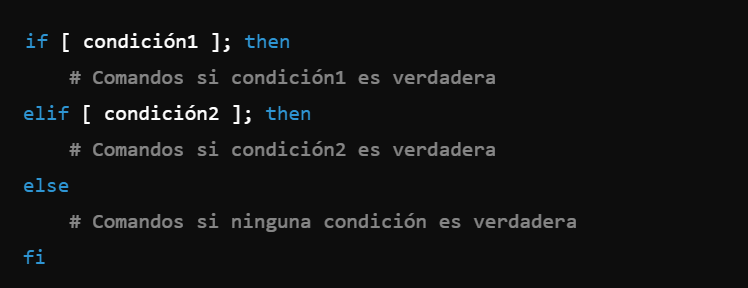
test "cadena1" != "cadena2"

[ "cadena1" != "cadena2" ]

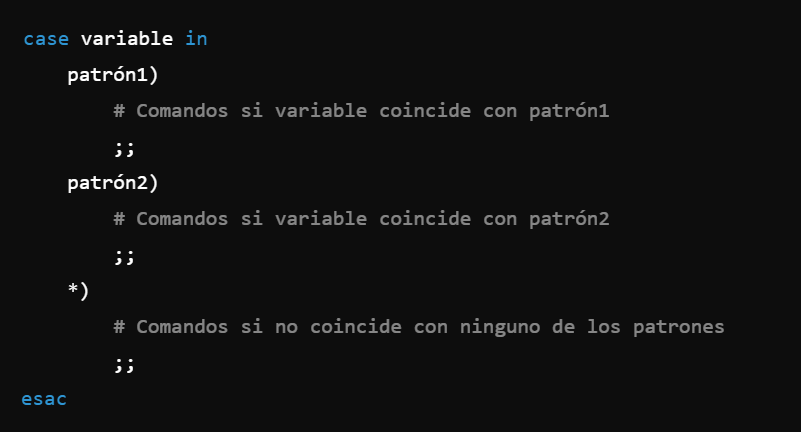
1. **8. Estructuras de control. Investigue la sintaxis de las siguientes estructuras de control incluidas en shell scripting:**

**1. Estructura if**

La estructura if se utiliza para tomar decisiones basadas en una condición.

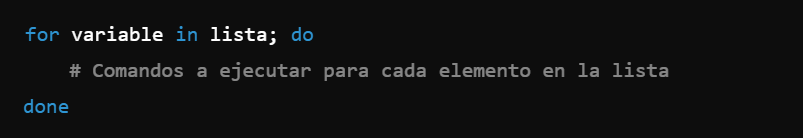
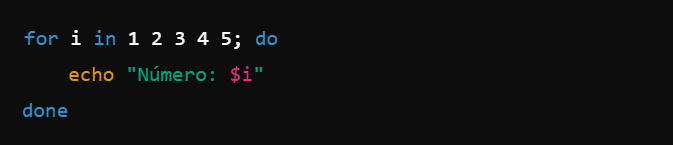


**2. Estructura case**

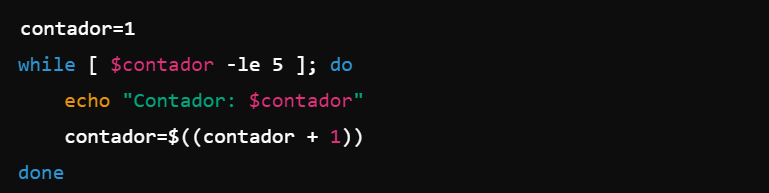
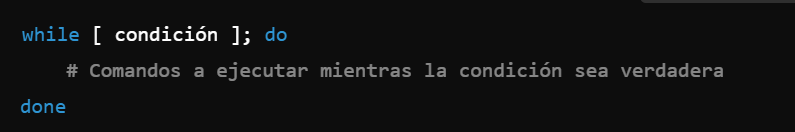
La estructura case se utiliza para realizar múltiples comparaciones sobre 

1. **Estructura for**

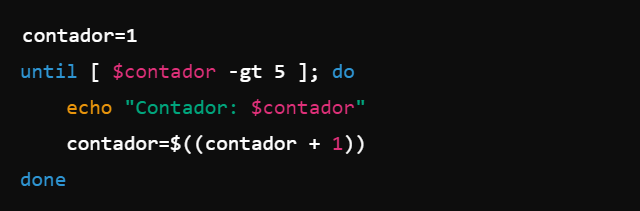
La estructura for se utiliza para iterar sobre una lista de elementos**.**

****

**4. Estructura while**

La estructura while se utiliza para ejecutar un bloque de comandos mientras se cumpla una condición.

**5. Estructura until**

La estructura until se utiliza para ejecutar un bloque de comandos hasta que se cumpla una condición.

1. **¿Qué acciones realizan las sentencias break y continue dentro de un bucle? ¿Qué parámetros reciben?**

**1. Sentencia break**

Acción:

* La sentencia break se utiliza para salir inmediatamente de un bucle (ya sea un for, while, o until). Cuando se ejecuta break, el control se transfiere a la siguiente instrucción después del bucle.

**2. Sentencia continue**

Acción:

* La sentencia continue se utiliza para saltar la iteración actual del bucle y pasar a la siguiente iteración. No termina el bucle, sino que omite el resto de los comandos en la iteración actual y continúa con la próxima.

1. **¿Qué tipo de variables existen? ¿Es shell script fuertemente tipado? ¿Se pueden definir arreglos? ¿Cómo?**

**¿Qué tipo de variables existen?**

**Variables de entorno**:

* Disponibles para todos los procesos.
* Se definen con export.
* Ejemplo: export VAR="valor"

**Variables locales**:

* Solo accesibles dentro de un script o función.
* Ejemplo: VAR="valor"

**Variables posicionales**:

* Contienen argumentos pasados a un script.
* Ejemplo: $1, $2, $@, $\*

**Variables especiales**:

* Tienen significados específicos.
* Ejemplos: $? (código de salida), $$ (ID de proceso), $# (número de argumentos)

**Arrays**:

* Almacenan múltiples valores en una sola variable.
* Ejemplo: mi\_array=(valor1 valor2 valor3)

**Variables de cadena**:

* Almacenan texto.
* Ejemplo: texto="Hola, mundo"

**¿Es shell script fuertemente tipado?**

No, Shell Script no es un lenguaje fuertemente tipado

Sí, en **Shell Script** (especialmente en Bash), se pueden definir y usar arreglos (o arrays).

**Arreglo unidimensional**:

* Puedes definir un arreglo unidimensional utilizando la siguiente sintaxis:

nombre\_arreglo=(elemento1 elemento2 elemento3)

1. **¿Pueden definirse funciones dentro de un script? ¿Cómo? ¿Cómo se maneja el pasaje de parámetros de una función a la otra?**

¿Pueden definirse funciones dentro de un script?

**Definición de Funciones**

1. **Sintaxis básica**:

nombre\_funcion() {

# Comandos a ejecutar

}