Sistemas Operativos en Red  
UD 13. Introducción a Bash Scripting - Parte 1

short line

Autores: Sergi García, Gloria Muñoz

Actualizado Abril 2024

Licencia

**Reconocimiento - No comercial - CompartirIgual** (BY-NC-SA): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se ha de hacer con una licencia igual a la que regula la obra original.

Nomenclatura

A lo largo de este tema se utilizarán diferentes símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

📖 **Importante**

❕ **Atención**

💬 **Interesante**

**ÍNDICE**

[**1. Introducción a Bash Scripting 3**](#_gjdgxs)

[**2. Variables en Bash 3**](#_lzci7sdijc2f)

[**3. Mostrar mensajes en la consola con Bash 6**](#_h52lnkjexz38)

[**4. Estructuras If-Else con Bash 8**](#_yfjid3lo2q1g)

[**5. Lectura de entrada estándard en bash 10**](#_uxk9u82mseig)

[**6. Ejercicios resueltos de ejemplo 13**](#_8lygiz2gx6y7)

[**7. Bibliografía 16**](#_bctzfwncaa27)

Unidad 13. Introducción a Bash Scripting - Parte 1

# Introducción a Bash Scripting

Bash (Bourne Again SHell) es una herramienta fundamental de scripting y línea de comandos ampliamente utilizada en entornos Unix y Linux. Es esencial para la administración de sistemas y la automatización de tareas, ofreciendo un entorno poderoso para ejecutar comandos y scripts que manejan operaciones del sistema operativo y aplicaciones.

### **Características clave de Bash:**

#### Automatización:

* **Scripts**: Bash permite a los administradores de sistemas escribir scripts para automatizar tareas repetitivas. Estos scripts pueden ejecutar secuencias de comandos y operaciones que mejoran la eficiencia y reducen los errores humanos, similar a lo que se logra con los cmdlets en PowerShell.
* **Cron Jobs**: Bash utiliza cron para programar tareas que deben ejecutarse en intervalos regulares, proporcionando una herramienta robusta para la automatización de tareas en el backend.

#### Gestión Basada en Texto:

* **Manipulación de Texto y Stream**: Bash trabaja predominantemente con texto plano. Herramientas como grep, awk, y sed permiten una potente manipulación de texto, que es esencial para el procesamiento de archivos de configuración y logs.
* **Pipes y Redirecciones**: Bash permite la combinación de comandos mediante tuberías (|) y la redirección de entrada/salida, facilitando la creación de flujos de trabajo complejos a partir de comandos simples.

#### Universalidad y Flexibilidad:

* **Amplia Adopción y Comunidad**: Como el intérprete de comandos predeterminado en la mayoría de los sistemas Unix y Linux, Bash tiene una amplia adopción y una comunidad activa que contribuye con numerosos scripts y soluciones. Esto asegura que los administradores tengan acceso a una vasta cantidad de recursos y documentación.

Bash es una herramienta indispensable para los profesionales de TI que trabajan en entornos Unix y Linux, proporcionando una plataforma robusta para la gestión y automatización de tareas. Aunque carece de algunas de las características orientadas a objetos de PowerShell, su simplicidad y el enfoque en la manipulación de texto lo hacen ideal para una variedad de tareas de scripting en sistemas basados en Unix.

# Variables en Bash

En Bash, las variables también se utilizan para almacenar información que puede ser utilizada y manipulada dentro del script. Las variables en Bash se definen utilizando el nombre de la variable precedido por el signo $, pero la asignación se realiza sin este signo. Bash es un lenguaje de tipado débil, lo que significa que no se declara explícitamente el tipo de dato de la variable, y el tipo puede cambiar según el contexto de uso.

### **Declaración de Variables en Bash**

A continuación, te proporciono ejemplos detallados de cómo puedes declarar y utilizar variables en Bash, lo cual te ayudará a comprender cómo funcionan las variables y operaciones en este entorno de scripting:

#### Variables numéricas

| numero=25 pi=3.14159 |
| --- |

**Explicación:**

* numero es una variable que almacena el valor entero 25.
* pi es una variable que guarda el valor decimal 3.14159, que es una aproximación del número pi.

#### Variables de cadena

| saludo="Hola, mundo" nombre='Juan' |
| --- |

**Explicación:**

* saludo contiene la cadena de texto "Hola, mundo".
* nombre guarda la cadena "Juan". Bash permite el uso de comillas simples o dobles para definir cadenas de texto.

#### Variables booleanas

| verdadero=true falso=false |
| --- |

**Explicación:**

* verdadero es una variable booleana que almacena el valor true.
* falso almacena el valor false. Estas variables se usan comúnmente para controlar el flujo de ejecución en scripts.

#### Array

| numeros=(1 2 3 4 5) nombres=("Ana" "Beto" "Carlos") |
| --- |

**Explicación:**

* numeros es un array que contiene cinco elementos numéricos.
* nombres es un array de cadenas que incluye los nombres "Ana", "Beto", y "Carlos". Los arrays son útiles para almacenar colecciones de datos.

#### Hashtable (Asociative array)

| declare -A persona persona[Nombre]="Elena" persona[Edad]=32 persona[Ciudad]="Madrid" |
| --- |

**Explicación:**

* persona es un array asociativo que almacena datos en pares clave-valor. Aquí, Nombre, Edad, y Ciudad son las claves, y cada una tiene un valor asociado, permitiendo un acceso rápido y organizado a cada elemento.

### **Operaciones con variables en Bash**

#### Concatenación de cadenas

| nombreCompleto="$nombre Perez" echo "Nombre completo: $nombreCompleto" |
| --- |

**Explicación:**

* Concatena el valor de $nombre con la cadena " Perez" para formar el nombre completo y luego lo imprime.

#### Operaciones matemáticas

| suma=$((numero + 15)) multiplicacion=$((numero \* 2)) echo "Suma: $suma, Multiplicación: $multiplicacion" |
| --- |

**Explicación:**

* suma calcula la suma de 25 y 15, resultando 40.
* multiplicacion realiza la multiplicación de 25 por 2, dando como resultado 50.

#### Accediendo a elementos de un array

| primerElemento=${numeros[0]} ultimoElemento=${nombres[2]} *# Bash no tiene índice negativo directo* echo "Primer elemento: $primerElemento, Último elemento: $ultimoElemento" |
| --- |

**Explicación:**

* primerElemento extrae el primer elemento del array $numeros, que es 1.
* ultimoElemento obtiene el último elemento del array $nombres, que es "Carlos".

#### Modificando y añadiendo datos a un array asociativo

| persona[Edad]=33 persona[Profesion]="Ingeniero" |
| --- |

**Explicación:**

* Modifica la edad en el array asociativo persona a 33.
* Añade un nuevo par clave-valor, con la clave "Profesion" y el valor "Ingeniero".

#### 

#### Interpolación de Cadenas

| mensaje="Tu nombre es $nombre y tienes ${persona[Edad]} años." echo $mensaje |
| --- |

**Explicación:**

* Forma una cadena interpolada que incluye el nombre y la edad almacenada en el array asociativo persona, y luego la imprime.

#### Variables de Entorno

| path=$PATH echo "El PATH actual es: $path" |
| --- |

**Explicación:**

* Recupera la variable de entorno PATH y la imprime. Esta variable de entorno contiene las rutas de los directorios donde el sistema operativo y otros programas buscan ejecutables.

# Mostrar mensajes en la consola con Bash

En Bash, mostrar mensajes en la consola es una tarea común y se realiza principalmente utilizando comandos como echo, printf, y otras utilidades de la línea de comandos. A continuación, te explico cómo y cuándo usar cada uno de estos comandos y técnicas para la impresión en la consola:

### **Comandos comunes para mostrar mensajes en Bash**

#### echo

* **Uso básico**: echo se utiliza para mostrar texto simple en la consola.
* **Control de nueva línea**: Por defecto, echo añade una nueva línea al final del mensaje, pero esto se puede controlar con la opción -n.

#### printf

* **Formato avanzado**: printf permite un control más detallado del formato de salida, similar a otros lenguajes de programación como C.
* **Colores y formato**: Se pueden usar secuencias de escape ANSI para agregar color y otros efectos de texto.

### **Ejemplos de Impresión en Consola en Bash**

#### Imprimir texto simple

| echo "Hola, mundo." |
| --- |

**Explicación**:

* echo es usado aquí para imprimir el mensaje "Hola, mundo." directamente en la consola. Este es el uso más básico y directo de echo para mostrar información simple al usuario.

#### 

#### Imprimir con cambio de línea personalizado

| echo -n "Hola,"; echo " mundo." |
| --- |

**Explicación**:

* echo -n se utiliza para evitar que se añada un salto de línea después de "Hola,". Inmediatamente después, echo imprime " mundo." Esto resulta en la salida "Hola, mundo." en la misma línea.

#### Imprimir con colores

| echo -e "\e[32mTexto en verde.\e[0m" |
| --- |

**Explicación**:

* Este comando cambia el color del texto a verde usando secuencias de escape ANSI. Es útil para destacar mensajes importantes.

#### Imprimir salida de un comando

echo $(date)

**Explicación**:

* echo se usa para imprimir la salida del comando date, que obtiene la fecha y hora actual.

#### Mostrar una advertencia

| echo -e "\e[33mAdvertencia: Queda poco espacio en disco.\e[0m" |
| --- |

**Explicación**:

* echo emite una advertencia, que se muestra en color amarillo. Este tipo de mensajes sirve para alertar al usuario sobre condiciones que podrían requerir atención.

#### Emitir un error

| echo -e "\e[31mError crítico: Fallo de operación.\e[0m" |
| --- |

**Explicación**:

* echo muestra un mensaje de error en color rojo, indicando problemas que han ocurrido durante la ejecución de un script.

#### Combinar textos y variables

| nombre="Ana" echo "Hola, $nombre" |
| --- |

**Explicación**:

* Aquí se muestra cómo incluir el contenido de una variable (nombre) en un mensaje impreso.

#### Imprimir múltiples líneas con un solo comando

| echo -e "Primera línea\nSegunda línea" |
| --- |

**Explicación**:

* echo se usa para imprimir múltiples líneas con un solo comando, utilizando \n para separar las líneas.

#### Imprimir listado de objetos

| colores=("Rojo" "Verde" "Azul") **for** color **in** "${colores[@]}"; **do**  echo $color **done** |
| --- |

**Explicación**:

* Utilizando un bucle for, cada color en el array colores se imprime en una línea nueva.

#### Imprimir texto alineado

| echo -n "Inicio de línea"; echo " y fin de línea." |
| --- |

**Explicación**:

* Combina dos echo, el primero con -n para mantener la salida en la misma línea.

#### Imprimir con formato tabular

| printf "%-10s %-5s\n" "Nombre" "Edad" printf "%-10s %-5s\n" "Luis" "30" printf "%-10s %-5s\n" "Marta" "25" |
| --- |

**Explicación**:

* printf se utiliza para crear una tabla bien formateada, especificando el ancho y la alineación de las columnas.

#### Imprimir y redirigir a un archivo

| echo "Este es un mensaje" > mensaje.txt |
| --- |

**Explicación**:

* Redirige la salida de echo a un archivo llamado mensaje.txt, guardando el mensaje en el archivo.

# Estructuras If-Else con Bash

En Bash, las estructuras condicionales if-else permiten la ejecución de bloques de código de forma condicional. Estas estructuras evalúan expresiones que devuelven verdadero o falso y ejecutan bloques de código basados en el resultado de esas evaluaciones.

### **Sintaxis básica de If-Else en Bash**

| **if** [ condición ]; **then**  *# código si la condición es verdadera* **elif** [ otra condición ]; **then**  *# código si la otra condición es verdadera* **else**  *# código si ninguna de las condiciones anteriores es verdadera* **fi** |
| --- |

**Explicación**:

* Esta es la estructura básica para crear lógica condicional en Bash. Comienza con un if que evalúa una condición. Si la condición es verdadera, ejecuta el código dentro del bloque. Si no es verdadera, pasa a evaluar otra condición en un elif (si existe), y finalmente tiene un bloque else para ejecutar un código si todas las condiciones anteriores son falsas.

### **Ejemplos de rstructuras condicionales If-Else en Bash**

#### If simple

| edad=20 **if** [ $edad -ge 18 ]; **then**  echo "Eres mayor de edad." **fi** |
| --- |

**Explicación**:

* Verifica si la variable edad es mayor o igual a 18. Si es cierto, imprime "Eres mayor de edad." Este tipo de condicional es útil para decisiones rápidas basadas en un único criterio.

#### If-Else

| edad=16 **if** [ $edad -ge 18 ]; **then**  echo "Eres mayor de edad." **else**  echo "No eres mayor de edad." **fi** |
| --- |

**Explicación**:

* Similar al anterior, pero con un bloque else que cubre el caso en que la condición no se cumpla. Aquí, si edad no es al menos 18, se ejecuta el código en else, imprimiendo "No eres mayor de edad."

#### If-ElseIf-Else

| calificacion=85 **if** [ $calificacion -ge 90 ]; **then**  echo "Excelente" **elif** [ $calificacion -ge 80 ]; **then**  echo "Bueno" **else**  echo "Necesita mejorar" **fi** |
| --- |

**Explicación**:

* Este ejemplo maneja múltiples condiciones para clasificar una calificación. Dependiendo del rango en el que la calificación se encuentre, imprime un mensaje diferente.

#### Condiciones compuestas

| edad=25 tieneLicencia=true **if** [ $edad -ge 18 ] && [ $tieneLicencia = true ]; **then**  echo "Puede conducir" **else**  echo "No puede conducir" **fi** |
| --- |

**Explicación**:

* Evalúa dos condiciones simultáneamente (edad debe ser al menos 18 y tieneLicencia debe ser verdadero). Solo si ambas condiciones son verdaderas, se permite conducir.

#### Negación de condición

| usuarioLogueado=false **if** [ $usuarioLogueado = false ]; **then**  echo "Usuario no logueado" **fi** |
| --- |

**Explicación**:

* Utiliza la negación para verificar si usuarioLogueado es falso, es decir, si el usuario no está logueado.

#### Operador -or

| temperatura=30 **if** [ $temperatura -lt 15 ] || [ $temperatura -gt 25 ]; **then**  echo "Temperatura fuera del rango cómodo" **fi** |
| --- |

**Explicación**:

* Verifica si la temperatura está fuera de un rango cómodo (menor que 15 o mayor que 25). Si alguna de las condiciones es verdadera, ejecuta el bloque de código.

# Lectura de entrada estándard en bash

En Bash, leer la entrada estándar se puede manejar de manera simple utilizando el comando read. Este comando permite interactuar con los usuarios de scripts, recogiendo datos en tiempo de ejecución. A continuación, te proporciono ejemplos de cómo realizar estas operaciones en Bash:

### **Ejemplos de lectura de la entrada estándar en Bash**

#### Leer una cadena simple

| echo "Introduce tu nombre:" read nombre echo "Hola, $nombre!" |
| --- |

**Explicación**:

* Este ejemplo pide al usuario que introduzca su nombre. El valor introducido se almacena en la variable nombre y luego se usa para saludar al usuario.

#### Convertir la entrada en un número entero

| echo "Introduce tu edad:" read edad echo "Tienes $edad años." |
| --- |

**Explicación**:

* Aquí se solicita la edad del usuario. read recoge la entrada como una cadena, y Bash la trata como un número entero automáticamente cuando se usa en un contexto que requiere un entero.

#### Leer un booleano

| echo "¿Eres mayor de 18 años? (sí/no)" read respuesta **if** [ "$respuesta" = "sí" ]; **then**  echo "Acceso concedido." **else**  echo "Acceso denegado." **fi** |
| --- |

**Explicación**:

* Se pide al usuario que confirme si es mayor de edad. Dependiendo de la respuesta, el script toma una decisión de control de acceso.

#### Validación de entrada

| **while** true; **do**  echo "Introduce tu correo electrónico:"  read email  **if** [[ "$email" =~ ^[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}$ ]]; **then**  echo "Email válido: $email"  break  **else**  echo "Formato de email no válido, intenta de nuevo."  **fi** **done** |
| --- |

**Explicación**:

* Solicita al usuario su correo electrónico repetidamente hasta que introduzca un formato válido (simple validación con expresión regular).

#### Lectura de contraseña

| echo "Introduce tu contraseña:" read -s password echo "Contraseña guardada de forma segura." |
| --- |

**Explicación**:

* Pide al usuario que introduzca una contraseña, utilizando la opción -s para read que evita que la entrada se muestre en la pantalla.

#### Elección múltiple

| echo "Elige tu color favorito (Rojo, Verde, Azul):" read colorFavorito **case** $colorFavorito **in**  Rojo) echo "Has elegido Rojo." ;;  Verde) echo "Has elegido Verde." ;;  Azul) echo "Has elegido Azul." ;;  \*) echo "No has elegido un color válido." ;; **esac** |
| --- |

**Explicación**:

* Este script permite al usuario elegir entre tres opciones y utiliza case para manejar la respuesta.

#### Confirmación Sí/No

| echo "¿Deseas continuar? (sí/no)" read confirmacion **if** [ "$confirmacion" = "sí" ]; **then**  echo "Proceso continuado." **else**  echo "Proceso detenido." **fi** |
| --- |

**Explicación**:

* Pide al usuario que confirme si desea continuar con el proceso. La acción se toma en base a su respuesta.

#### Entrada de número con validación

| **while** true; **do**  echo "Introduce un número (0-100):"  read numero  **if** [[ "$numero" =~ ^[0-9]+$ ]] && [ "$numero" -ge 0 ] && [ "$numero" -le 100 ]; **then**  echo "Número válido: $numero"  break  **else**  echo "Introduce un número válido."  **fi** **done** |
| --- |

**Explicación**:

* Pide un número entre 0 y 100, asegurando que la entrada sea un entero válido dentro del rango especificado.

#### Lectura de fecha

| echo "Introduce tu fecha de nacimiento (formato: YYYY-MM-DD):" read fecha echo "Fecha de nacimiento: $fecha" |
| --- |

**Explicación**:

* Solicita la fecha de nacimiento del usuario y simplemente la imprime. Bash no tiene capacidad nativa para validar o convertir fechas sin comandos externos o scripts adicionales.

#### Lista de valores separados por comas

| echo "Introduce una lista de artículos, separados por comas:" read items arrayItems=(${items//,/ }) echo "Has introducido ${#arrayItems[@]} artículos." |
| --- |

**Explicación**:

* Pide al usuario que introduzca una lista de artículos separados por comas y luego divide la cadena en un array de artículos individuales utilizando sustitución de patrones de Bash.

# Ejercicios resueltos de ejemplo

### **Ejercicio 1: Cálculo de descuentos de tienda en Bash**

**Enunciado:** Desarrolla un script en Bash para ayudar a una tienda a calcular descuentos en el punto de venta. El script debe:

* Solicitar al usuario el precio original de un producto.
* Solicitar al usuario el porcentaje de descuento a aplicar.
* Calcular y mostrar el precio final después del descuento.
* Manejar cualquier error de entrada (por ejemplo, entradas no numéricas o porcentajes inválidos).

**Solución:**

| **#!/bin/bash** echo "Ingrese el precio original del producto:" read precioOriginal echo "Ingrese el porcentaje de descuento (sin '%'):" read descuento  **if** ! [[ "$precioOriginal" =~ ^[0-9]+(\.[0-9]+)?$ ]] || ! [[ "$descuento" =~ ^[0-9]+(\.[0-9]+)?$ ]]; **then**  echo "Error: Entrada no válida." **elif** [ "$descuento" -lt 0 ] || [ "$descuento" -gt 100 ]; **then**  echo "El porcentaje de descuento debe estar entre 0 y 100." **else**  precioFinal=$(echo "scale=2; $precioOriginal - ($precioOriginal \* $descuento / 100)" | bc)  echo "El precio final después del descuento es: $precioFinal" **fi** |
| --- |

**Explicación:** El script pide al usuario que introduzca el precio original y el porcentaje de descuento. Ambos valores se validan para asegurar que son numéricos y que el descuento está en un rango aceptable. Se calcula el precio final aplicando el descuento al precio original.

### **Ejercicio 2: Registro de participantes en evento en Bash**

**Enunciado:** Crea un script en Bash para registrar participantes en un evento. El script debe:

* Solicitar el nombre, edad y correo electrónico de un participante.
* Verificar que la edad es mayor o igual a 18 años.
* Verificar que el correo electrónico tenga un formato válido.
* Imprimir un mensaje de confirmación con los datos del participante o un mensaje de error si hay problemas con la entrada.

**Solución:**

| **#!/bin/bash** echo "Ingrese su nombre:" read nombre echo "Ingrese su edad:" read edad echo "Ingrese su correo electrónico:" read email  **if** ! [[ "$edad" =~ ^[0-9]+$ ]] || [ "$edad" -lt 18 ]; **then**  echo "Debe ser mayor de 18 años para registrarse." **elif** ! [[ "$email" =~ ^[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}$ ]]; **then**  echo "Formato de correo electrónico no válido." **else**  echo "Registro completado con éxito. Nombre: $nombre, Edad: $edad, Email: $email" **fi** |
| --- |

**Explicación:** El script recoge información básica del participante y verifica la edad para asegurarse de que cumpla con un mínimo requerido. También verifica que el correo electrónico introducido tenga un formato adecuado usando una expresión regular. Si alguna validación falla, se muestra un mensaje de error correspondiente. Si todo es correcto, se imprime un mensaje de confirmación.

### **Ejercicio 3: Simulador de autenticación en Bash**

**Enunciado:** Implementa un script en Bash que simule el proceso de autenticación de un usuario en un sistema. El script debe:

* Solicitar un nombre de usuario y una contraseña.
* Verificar que el nombre de usuario y la contraseña no estén vacíos.
* Simular la verificación de las credenciales (por ejemplo, nombre de usuario "admin" y contraseña "admin123").
* Imprimir un mensaje de éxito o de error en la autenticación.

**Solución:**

| **#!/bin/bash** echo "Ingrese su nombre de usuario:" read username echo "Ingrese su contraseña:" read password  **if** [[ -z "$username" ]] || [[ -z "$password" ]]; **then**  echo "Ni el nombre de usuario ni la contraseña pueden estar vacíos." **elif** [[ "$username" == "admin" ]] && [[ "$password" == "admin123" ]]; **then**  echo "Autenticación exitosa. Bienvenido $username!" **else**  echo "Nombre de usuario o contraseña incorrectos." **fi** |
| --- |

**Explicación:** Este script pide al usuario que introduzca su nombre de usuario y contraseña. Verifica que ninguno de los campos esté vacío. Comprueba si las credenciales coinciden con un conjunto predefinido ("admin" y "admin123"). Dependiendo de la validez de las credenciales, imprime un mensaje apropiado.

# Bibliografía

* Bash Scripting Guide <https://www.tldp.org/LDP/Bash-Beginners-Guide/html/>
* Advanced Bash-Scripting Guide <https://www.tldp.org/LDP/abs/html/>
* Bash Guide for Beginners <https://www.tldp.org/LDP/Bash-Beginners-Guide/html/Bash-Beginners-Guide.html>
* GNU Bash manual <https://www.gnu.org/software/bash/manual/>