# Introducción a Bash Scripting

Contenido

[Introducción a Bash Scripting 1](#_Toc198390452)

[Introducción 4](#_Toc198390453)

[¿Qué es un script Bash? 4](#_Toc198390454)

[Crear un script paso a paso 4](#_Toc198390455)

[¿Qué es `#!/bin/bash`? 4](#_Toc198390456)

[Comentarios en Bash 4](#_Toc198390457)

[Variables, Condicionales y Bucles en Bash 5](#_Toc198390458)

[1. Variables 5](#_Toc198390459)

[2. Condicionales 5](#_Toc198390460)

[3. Bucles 5](#_Toc198390461)

[Funciones y Argumentos en Bash 6](#_Toc198390462)

[1. Declaración de funciones 6](#_Toc198390463)

[2. Argumentos en funciones 6](#_Toc198390464)

[3. Retornar valores 6](#_Toc198390465)

[4. Argumentos en scripts 7](#_Toc198390466)

[Argumentos en Bash 7](#_Toc198390467)

[1. Declaración de funciones 7](#_Toc198390468)

[2. Argumentos en funciones 7](#_Toc198390469)

[3. Retornar valores 8](#_Toc198390470)

[4. Argumentos en scripts 8](#_Toc198390471)

[Manipulación de Archivos y Directorios en Bash 8](#_Toc198390472)

[1. Crear y verificar archivos 9](#_Toc198390473)

[2. Crear y verificar directorios 9](#_Toc198390474)

[3. Mover, copiar y eliminar archivos 9](#_Toc198390475)

[4. Leer archivos línea por línea 9](#_Toc198390476)

[5. Buscar archivos y contenido 9](#_Toc198390477)

[6. Modificar contenido con sed y awk 10](#_Toc198390478)

[Buenas Prácticas en Bash Scripting 10](#_Toc198390479)

[1. Uso de set -e y set -u 10](#_Toc198390480)

[2. Validación de argumentos 10](#_Toc198390481)

[3. Uso de funciones para modularizar 10](#_Toc198390482)

[4. Manejo de errores con trap 11](#_Toc198390483)

[5. Comentarios claros 11](#_Toc198390484)

[Scripts Útiles con Archivos y Logs en Bash 11](#_Toc198390485)

[1. Script de backup automático 11](#_Toc198390486)

[2. Monitoreo de espacio en disco 12](#_Toc198390487)

[3. Rotación y compresión de logs antiguos 12](#_Toc198390488)

# Introducción

En este manual aprenderás los fundamentos de los scripts en Bash usando Ubuntu desde WSL (Windows Subsystem for Linux). Veremos cómo crear, ejecutar y entender la estructura básica de un script en Linux.

## ¿Qué es un script Bash?

Un script Bash es un archivo de texto plano que contiene una serie de comandos que pueden ser ejecutados como un programa. Su extensión común es '.sh'. Este tipo de archivo automatiza tareas repetitivas en el sistema.

## Crear un script paso a paso

1. Abre tu terminal Ubuntu (WSL en Windows).

2. Usa un editor como `nano` para crear un archivo:

nano hola.sh

3. Escribe lo siguiente en el archivo:

#!/bin/bash  
echo "Hola, mundo desde Bash!"

4. Guarda y cierra (Ctrl + O, Enter y luego Ctrl + X).

5. Dale permisos de ejecución:

chmod +x hola.sh

6. Ejecuta el script:

./hola.sh

## ¿Qué es `#!/bin/bash`?

`#!/bin/bash` es la primera línea del script y se llama \*shebang\*. Este le dice al sistema que use Bash como intérprete para ejecutar el archivo.

## Comentarios en Bash

Los comentarios en Bash se escriben con el símbolo `#` y permiten documentar el código. No son ejecutados.  
  
Ejemplo:  
# Este script muestra un saludo  
echo "Hola"

# Variables, Condicionales y Bucles en Bash

En este módulo aprenderás a utilizar variables, estructuras condicionales y bucles en Bash. Estos elementos son fundamentales para crear scripts más potentes y dinámicos.

## 1. Variables

Las variables en Bash no necesitan ser declaradas con tipo. Se asignan con el símbolo '=' sin espacios.  
  
Ejemplo:

nombre="José"  
echo "Hola $nombre"

Para acceder a una variable, se usa el símbolo `$` delante del nombre.

## 2. Condicionales

Las estructuras condicionales permiten ejecutar código basado en condiciones. Se usa `if`, `elif`, `else` y `fi`.

Ejemplo básico:

numero=5  
if [ $numero -gt 3 ]; then  
 echo "Es mayor que 3"  
else  
 echo "No es mayor que 3"  
fi

Notas:

- Se usan corchetes `[ ]` con espacios.  
- Operadores: `-gt` (mayor), `-lt` (menor), `-eq` (igual), etc.

## 3. Bucles

### Bucle for

Ejecuta un bloque de código varias veces.

for i in 1 2 3  
do  
 echo "Número: $i"  
done

### Bucle while

Ejecuta mientras una condición sea verdadera.

contador=1  
while [ $contador -le 3 ]  
do  
 echo "Contador: $contador"  
 ((contador++))  
done

# Funciones y Argumentos en Bash

Este módulo te enseñará a declarar funciones dentro de un script Bash y cómo pasar argumentos a los scripts o funciones. Las funciones permiten estructurar mejor el código y reutilizar lógica.

## 1. Declaración de funciones

Una función se define con la palabra clave `function` (opcional) o directamente con el nombre seguido de paréntesis.

Ejemplo:

saludar() {  
 echo "Hola, bienvenido a Bash!"  
}  
  
saludar

## 2. Argumentos en funciones

Los argumentos se acceden con `$1`, `$2`, etc., dentro de la función.

Ejemplo:

saludo\_personalizado() {  
 echo "Hola $1, bienvenido!"  
}  
  
saludo\_personalizado "José"

## 3. Retornar valores

Las funciones pueden retornar un valor numérico usando `return`, aunque el resultado principal suele comunicarse por `echo`.  
  
Ejemplo:

sumar() {  
 resultado=$(( $1 + $2 ))  
 echo $resultado  
}  
  
suma=$(sumar 3 4)  
echo "Resultado: $suma"

## 4. Argumentos en scripts

Los scripts también pueden recibir parámetros desde la línea de comandos.

Ejemplo:

#!/bin/bash  
echo "Primer argumento: $1"  
echo "Segundo argumento: $2"  
echo "Número de argumentos: $#"  
echo "Todos los argumentos: $@"

Lo ejecutamos:

./script.sh valor1 valor2

# Argumentos en Bash

Este módulo te enseñará a declarar funciones dentro de un script Bash y cómo pasar argumentos a los scripts o funciones. Las funciones permiten estructurar mejor el código y reutilizar lógica.

## 1. Declaración de funciones

Una función se define con la palabra clave `function` (opcional) o directamente con el nombre seguido de paréntesis.

Ejemplo:

saludar() {  
 echo "Hola, bienvenido a Bash!"  
}  
  
saludar

## 2. Argumentos en funciones

Los argumentos se acceden con `$1`, `$2`, etc., dentro de la función.

Ejemplo:

saludo\_personalizado() {  
 echo "Hola $1, bienvenido!"  
}  
  
saludo\_personalizado "José"

## 3. Retornar valores

Las funciones pueden retornar un valor numérico usando `return`, aunque el resultado principal suele comunicarse por `echo`.  
  
Ejemplo:

sumar() {  
 resultado=$(( $1 + $2 ))  
 echo $resultado  
}  
  
suma=$(sumar 3 4)  
echo "Resultado: $suma"

## 4. Argumentos en scripts

Los scripts también pueden recibir parámetros desde la línea de comandos.

Ejemplo:

#!/bin/bash  
echo "Primer argumento: $1"  
echo "Segundo argumento: $2"  
echo "Número de argumentos: $#"  
echo "Todos los argumentos: $@"

Lo ejecutamos:

./script.sh valor1 valor2

# Manipulación de Archivos y Directorios en Bash

En este módulo aprenderás a trabajar con archivos y directorios desde scripts Bash. Dominar estos comandos es clave para automatizar tareas como copias de seguridad, organización de archivos y limpieza de logs.

## 1. Crear y verificar archivos

Para crear archivos vacíos:

touch archivo.txt

Verificar si un archivo existe:

if [ -f "archivo.txt" ]; then  
 echo "El archivo existe"  
fi

## 2. Crear y verificar directorios

Crear un directorio:

mkdir mi\_carpeta

Verificar existencia de directorio:

if [ -d "mi\_carpeta" ]; then  
 echo "Directorio encontrado"  
fi

## 3. Mover, copiar y eliminar archivos

Copiar archivo: cp origen.txt destino.txt

Mover archivo: mv archivo.txt carpeta/

Eliminar archivo: rm archivo.txt

Eliminar directorio y contenido: rm -r carpeta/

## 4. Leer archivos línea por línea

Ejemplo para leer un archivo línea por línea:

while IFS= read -r linea  
do  
 echo "Línea: $linea"  
done < archivo.txt

## 5. Buscar archivos y contenido

Buscar archivos por nombre:

find . -name '\*.log'

Buscar contenido en archivos:

grep 'ERROR' archivo.log

## 6. Modificar contenido con sed y awk

Reemplazar texto:

sed -i 's/viejo/nuevo/g' archivo.txt

Extraer campos de un archivo CSV:

awk -F',' '{print $1,$3}' archivo.csv

# Buenas Prácticas en Bash Scripting

Este módulo presenta una serie de buenas prácticas recomendadas al escribir scripts en Bash. Seguir estos principios mejora la legibilidad, mantenimiento y seguridad de tus scripts.

## 1. Uso de set -e y set -u

- `set -e`: Hace que el script se detenga si ocurre un error (fallo en un comando).  
- `set -u`: Hace que el script se detenga si se utiliza una variable no definida.

Ejemplo:

#!/bin/bash  
set -eu  
echo "Inicio del script"  
echo "Valor: $VARIABLE\_NO\_DEFINIDA"

## 2. Validación de argumentos

Es importante verificar si se han pasado los argumentos necesarios antes de usarlos. Esto evita errores inesperados en la ejecución.

Ejemplo:

if [ $# -lt 1 ]; then  
 echo "Uso: $0 archivo"  
 exit 1  
fi

## 3. Uso de funciones para modularizar

Divide tu script en funciones pequeñas con responsabilidades claras.

Ejemplo:

hacer\_backup() {  
 cp "$1" "$2"  
 echo "Backup hecho de $1 a $2"  
}  
  
hacer\_backup archivo.txt /copia/archivo.txt

## 4. Manejo de errores con trap

`trap` permite ejecutar código cuando el script finaliza o encuentra errores. Es útil para limpiar recursos o registrar fallos.

Ejemplo:

trap "echo Fallo detectado; exit 1" ERR  
comando\_que\_falla

## 5. Comentarios claros

Documenta cada sección importante de tu script. Usa comentarios (`#`) para explicar el propósito del código.

Ejemplo:

# Este script crea un respaldo del archivo dado  
cp "$1" "$2"

# Scripts Útiles con Archivos y Logs en Bash

Este módulo contiene ejemplos reales y prácticos de scripts Bash orientados a la automatización de tareas generales. Se enfocan en el manejo de archivos, directorios y generación de logs para tareas cotidianas de administración y respaldo.

## 1. Script de backup automático

Este script hace una copia de respaldo de un archivo y genera un log con la fecha y hora de la operación.

#!/bin/bash  
set -e  
ORIGEN="$1"  
DESTINO="$2"  
LOG="backup.log"  
  
echo "$(date "+%F %T") - Iniciando backup de $ORIGEN a $DESTINO" >> "$LOG"  
cp "$ORIGEN" "$DESTINO" >> "$LOG" 2>&1  
echo "$(date "+%F %T") - Backup completado" >> "$LOG"

## 2. Monitoreo de espacio en disco

Este script verifica el uso del disco y genera alertas si supera cierto umbral.

#!/bin/bash  
UMBRAL=80  
LOG="espacio.log"  
  
USO=$(df / | tail -1 | awk '{print $5}' | sed 's/%//')  
  
if [ "$USO" -gt "$UMBRAL" ]; then  
 echo "$(date "+%F %T") - ALERTA: Uso de disco en $USO%" >> "$LOG"  
else  
 echo "$(date "+%F %T") - Estado OK: Uso de disco en $USO%" >> "$LOG"  
fi

## 3. Rotación y compresión de logs antiguos

Este script comprime logs si superan un tamaño definido y conserva un historial comprimido.

#!/bin/bash  
LOG="aplicacion.log"  
TAM\_MAX=100000 # en bytes  
  
if [ -f "$LOG" ] && [ $(stat -c%s "$LOG") -gt "$TAM\_MAX" ]; then  
 TIMESTAMP=$(date "+%Y%m%d%H%M%S")  
 mv "$LOG" "$LOG.$TIMESTAMP"  
 gzip "$LOG.$TIMESTAMP"  
 echo "$(date "+%F %T") - Log rotado y comprimido."  
fi