Version1:

**1. Variables en Bash**

Las **variables** son zonas de memoria donde se almacena información. En Bash existen **tres tipos principales**:

1️⃣ **Variables de Entorno**

* Son definidas por el sistema operativo.
* Se pueden visualizar con printenv o env.
* Ejemplo:

bash

CopiarEditar

$ printenv HOME # Muestra el directorio del usuario

$ printenv SHELL # Muestra la shell predeterminada

* Para hacerlas permanentes, deben añadirse al archivo ~/.bash\_profile.

2️⃣ **Variables de Usuario**

* Son definidas manualmente por el usuario.
* Se asignan sin espacios alrededor del =:

bash

CopiarEditar

nombre="Juan"

edad=25

* Se accede a ellas usando $nombre.

3️⃣ **Variables del Sistema**

* Algunas variables especiales:
  + $0: Nombre del script en ejecución.
  + $#: Número de argumentos pasados.
  + $\*: Lista de argumentos pasados.
  + $$: PID del proceso en ejecución.
  + $?: Código de salida del último comando ejecutado.

**🔷 2. Programación en Shell (Shell Scripting)**

Un **script en Bash** es un archivo de texto plano con instrucciones de la terminal.

📌 **Creación de un Script**

1. Crear un archivo de texto con extensión .sh:

bash

CopiarEditar

nano mi\_script.sh

1. Añadir la primera línea indicando el uso de Bash:

bash

CopiarEditar

#!/bin/bash

echo "Hola, mundo"

1. Dar permisos de ejecución:

bash

CopiarEditar

chmod +x mi\_script.sh

1. Ejecutarlo de diferentes formas:

bash

CopiarEditar

./mi\_script.sh # Se ejecuta en la misma terminal

bash mi\_script.sh # Se ejecuta en una nueva sesión de Bash

**🔷 3. Operadores y Expresiones**

Bash permite el uso de **operadores aritméticos, de comparación y lógicos**.

**✅ Operadores Aritméticos**

bash

CopiarEditar

a=10

b=5

echo $((a + b)) # Suma

echo $((a - b)) # Resta

echo $((a \* b)) # Multiplicación

echo $((a / b)) # División

echo $((a % b)) # Módulo

**✅ Comparación de Números**

bash

CopiarEditar

if [ $a -eq $b ]; then echo "Son iguales"; fi

if [ $a -gt $b ]; then echo "$a es mayor"; fi

* -eq: Igual a
* -ne: Diferente a
* -lt: Menor que
* -gt: Mayor que
* -le: Menor o igual
* -ge: Mayor o igual

**✅ Comparación de Cadenas**

bash

CopiarEditar

if [ "$cadena1" = "$cadena2" ]; then echo "Son iguales"; fi

if [ -z "$cadena1" ]; then echo "Está vacía"; fi

* =: Igual
* !=: Distinto
* -z: Cadena vacía
* -n: Cadena no vacía

**🔷 4. Estructuras de Control en Bash**

Bash permite el uso de estructuras de control como if, case, while, until y for para tomar decisiones y automatizar tareas.

**✅ if-then-else**

bash

CopiarEditar

if [ $num -gt 10 ]; then

echo "Mayor que 10"

elif [ $num -eq 10 ]; then

echo "Es 10"

else

echo "Menor que 10"

fi

**✅ case (Selección Múltiple)**

Más eficiente que múltiples if-elif.

bash

CopiarEditar

read -p "Introduce una opción: " opcion

case $opcion in

1) echo "Elegiste la opción 1" ;;

2) echo "Elegiste la opción 2" ;;

\*) echo "Opción no válida" ;;

esac

* \*) representa la opción por defecto si no coincide ninguna otra.

**✅ while (Bucle Condicional)**

bash

CopiarEditar

contador=1

while [ $contador -le 5 ]; do

echo "Número: $contador"

contador=$((contador + 1))

done

* Se ejecuta **mientras la condición sea verdadera**.

**✅ until (Bucle hasta que se cumpla una condición)**

bash

CopiarEditar

contador=10

until [ $contador -eq 0 ]; do

echo "Cuenta atrás: $contador"

contador=$((contador - 1))

done

* Se ejecuta **hasta que la condición se vuelva verdadera**.

**✅ for (Iteración sobre listas)**

bash

CopiarEditar

for i in {1..5}; do

echo "Número: $i"

done

* Se ejecuta un número determinado de veces.

**🔷 5. Funciones en Bash**

Las funciones permiten reutilizar código.

📌 **Definición y Uso**

bash

CopiarEditar

mi\_funcion() {

echo "Hola desde la función"

}

mi\_funcion # Llamada a la función

📌 **Función con Parámetros**

bash

CopiarEditar

suma() {

resultado=$(( $1 + $2 ))

echo "La suma es: $resultado"

}

suma 5 3

* $1, $2 son los argumentos pasados a la función.

**🔷 6. Manipulación de Archivos y Directorios**

Bash permite verificar y manipular archivos y directorios.

📌 **Verificación de Archivos**

bash

CopiarEditar

if [ -f "archivo.txt" ]; then

echo "El archivo existe"

fi

* -f: Comprueba si es un archivo.
* -d: Comprueba si es un directorio.
* -e: Comprueba si el archivo existe.

📌 **Crear, Renombrar y Eliminar Archivos**

bash

CopiarEditar

touch nuevo\_archivo.txt # Crea un archivo vacío

mv archivo1.txt archivo2.txt # Renombra un archivo

rm archivo.txt # Elimina un archivo

📌 **Recorrer Archivos en un Directorio**

bash

CopiarEditar

for archivo in \*.txt; do

echo "Procesando: $archivo"

done

**🔷 7. Seguridad y Modificación del PATH**

* La variable PATH define en qué directorios busca Bash los ejecutables.
* Modificar PATH temporalmente:

bash

CopiarEditar

export PATH=$PATH:/nuevo/directorio

* **Evitar riesgos de seguridad:**
  + No incluir . en el PATH (previene ataques con programas maliciosos en el directorio actual).

Version2:

**3. Variables en Bash**

* **Variables de entorno:** Son asignadas por el sistema (HOME, PATH, SHELL, USER, etc.).
* **Variables de usuario:** Definidas manualmente en la sesión (var=valor).
* **Exportación de variables:** Para que una variable sea accesible en subprocesos (export VAR\_NAME).
* **Variables del sistema:** Como $? (estado del último comando), $$ (ID del proceso en ejecución), $0 (nombre del script en ejecución).

**🔹 4. Programación en Shell**

* Se introducen los **shell scripts**, que son archivos de texto que contienen comandos ejecutables.
* Se recomienda usar la extensión .sh y comenzar con la línea #!/bin/bash.
* Se explica cómo ejecutar un script:
  + sh script.sh
  + ./script.sh (si tiene permisos de ejecución)
  + source script.sh (se ejecuta en la misma sesión)

**🔹 5. Comando echo**

* Se usa para mostrar texto en pantalla.
* Opciones avanzadas:
  + echo -n → Evita el salto de línea.
  + echo -e "\tTabulado" → Permite el uso de caracteres especiales como \n (nueva línea) o \t (tabulación).

**🔹 6. Uso de Comillas**

* **Simples (')**: Tratan el contenido de manera literal.
* **Dobles (")**: Permiten la expansión de variables.
* **Inversas (`\**): Ejecutan comandos y almacenan su resultado en una variable (fecha=$(date)).

**🔹 7. Lectura de Datos con read**

* Se pueden capturar datos del usuario mediante read variable.
* Ejemplo:

bash

CopiarEditar

echo "Introduce tu nombre:"

read nombre

echo "Hola, $nombre"

**🔹 8. Evaluación de Expresiones (test y [])**

* Se pueden evaluar condiciones en Bash:

bash

CopiarEditar

if [ $edad -ge 18 ]; then

echo "Eres mayor de edad"

fi

* Operadores:
  + -eq (igual)
  + -ne (diferente)
  + -lt (menor que)
  + -gt (mayor que)
  + -le (menor o igual que)
  + -ge (mayor o igual que)

**🔹 9. Estructuras de Control**

* **Condicional if**:

bash

CopiarEditar

if [ "$var1" = "ok" ]; then

echo "Todo correcto"

else

echo "Error"

fi

* **Switch-Case (case)**:

bash

CopiarEditar

case $opcion in

1) echo "Opción 1";;

2) echo "Opción 2";;

\*) echo "Opción no válida";;

esac

* **Bucles (while, until, for)**:
  + while: Se ejecuta mientras una condición sea verdadera.
  + until: Se ejecuta hasta que una condición sea verdadera.
  + for: Se usa para iterar sobre una lista de elementos.

**🔹 10. Parámetros en Scripts**

* Se pueden pasar parámetros a un script:

bash

CopiarEditar

./mi\_script.sh argumento1 argumento2

* Dentro del script se accede a los argumentos con:
  + $1, $2, $3, ..., $9 (para acceder a los primeros 9 argumentos).
  + $\* (todos los argumentos como una sola cadena).
  + $# (número total de argumentos).

**🔹 11. Manipulación de Archivos y Directorios**

* Se pueden verificar archivos con opciones como:
  + -f (si es un archivo)
  + -d (si es un directorio)
  + -r (si tiene permiso de lectura)
  + -w (si tiene permiso de escritura)
  + -x (si tiene permiso de ejecución)

**🔹 12. Creación y Uso de Funciones**

* Se pueden definir funciones en Bash para reutilizar código:

bash

CopiarEditar

funcion\_ejemplo() {

echo "Esto es una función"

}

funcion\_ejemplo

**🔹 13. Modificación del PATH y Seguridad**

* Se explica cómo modificar la variable PATH para incluir directorios personalizados.
* Se recomienda no incluir . en PATH para evitar ataques con **"caballos de Troya"**.