Programación en Shell

Comando test y Estructuras de Control

```
#!/bin/bash
```

¿Qué es la programación en shell?

Definición

Es la capacidad de escribir y ejecutar **scripts** en el intérprete de comandos del sistema operativo.

Automatización de tareas

Simplifica tareas repetitivas y complejas mediante scripts

Administración del sistema

Gestión eficiente de archivos, procesos y configuraciones

Rapidez y flexibilidad

Desarrollo ágil y adaptación a diferentes escenarios

El comando test

¿Qué es?

Comando que evalúa la **validez** de una condición o expresión

Valores de retorno

0	Condición verdadera (true)
1	Condición falsa (false)

Sintaxis

```
test [expresión]
```

```
[ expresión ]
```

Ejemplo de uso

```
#!/bin/bash

# Verificar si dos números son iguales
num1=10
num2=10

if test "$num1" -eq "$num2"; then
    echo "Los números son iguales"
else
    echo "Los números son diferentes"
fi
```

Resultado: `Los números son iguales`

El comando test: Opciones de archivos

Verifica si existe

Es un directorio

Archivo regular

No está vacío

Permisos de lectura

Permisos de escritura

Permisos de ejecución

Verificar archivo existe

```
if [ -f "archivo.txt" ]; then
  echo "Archivo encontrado"
fi
```

Comprobar directorio

```
if [ -d "/tmp" ]; then
  echo "Es un directorio"
fi
```

Verificar permisos

```
if [ -x "script.sh" ]; then
  echo "Ejecutable"
fi
```

El comando test: Opciones de cadenas y números

Cadenas

`=` Iguales	`!=` Diferentes
`-n` No vacío	`-z` Vacío

Números

`-eq` Igual	`-ne` Diferente	`-gt` Mayor
`-ge` Mayor o igual	`-lt` Menor	`-le` Menor o igual

Condicionales

`-a` Y (AND)	`-o` O (OR)	`!` NO (NOT)

Comparación de cadenas

```
if [ "$str1" = "$str2" ]; then
  echo "Cadenas iguales"
fi
```

Comparación numérica

```
if [ $num1 -gt $num2 ]; then
  echo "num1 es mayor"
fi
```

Operadores condicionales

```
if [ -f "file.txt" -a -r "file.txt" ]; then
  echo "Archivo existe y es legible"
fi
```

Formato con corchetes

Corchetes simples

- Compatibilidad POSIX
- Operadores: -a, -o, !
- Estándar en todos los shells

Corchetes dobles

- Extensiones Bash
- Operadores: &&, ||, =~
- Más eficiente y potente
- Expresiones regulares

Corchetes simples [ ]

```
if [ "$a" -gt 10 -a "$a" -lt 20 ]; then
  echo "a está entre 10 y 20"
fi
```

Usa operadores -a y -o

Corchetes dobles [[ ]]

```
if [[ "$a" > 10 && "$a" < 20 ]]; then
  echo "a está entre 10 y 20"
fi
```

Usa operadores && y ||

[[ ]] solo funciona en Bash, Zsh y Ksh. [ ] es más portable.

Estructuras de control: if

`if` `then` `elif` `else` `fi`

Sintaxis

`if` inicia la condición

`then` ejecuta si la condición es verdadera

`elif` evalúa condiciones alternativas

`else` ejecuta si ninguna condición es verdadera

`fi` cierra la estructura if

<> Ejemplo completo

```
#!/bin/bash

# Verificar edad
edad=18

if [ "$edad" -lt 18 ]; then
  echo "Menor de edad"
elif [ "$edad" -eq 18 ]; then
  echo "Exactamente 18 años"
else
  echo "Mayor de edad"
fi
```

Las estructuras if pueden anidarse para evaluar múltiples condiciones jerárquicas

Estructuras de control: case

`case` `in` `;;` `esac`

Sintaxis

`case` inicia la estructura

`in` separa la variable de los patrones

`;;` finaliza cada caso

`esac` cierra la estructura case

Más legible y eficiente que múltiples if-elif

Ejemplo: Menú de opciones

```
#!/bin/bash

opción="A"

case $opción in
  A)
    echo "Opción A seleccionada"
    ;;
  B)
    echo "Opción B seleccionada"
    ;;
  *)
    echo "Opción inválida"
    ;;
esac
```

Coincidencia de patrones

Soporta patrones como [0-9], [a-z], *, ? y expresiones regulares simples

Estructuras de control: for

`for` `do` `done`

Sintaxis

`for` inicia el bucle iterativo

`do` inicia el bucle de código a ejecutar

`done` cierra el bucle for

Itera sobre listas, rangos, archivos o comandos

Ejemplo: Iteración simple

```
#!/bin/bash

for i in 1 2 3 4 5
do
  echo "Número: $i"
done

# Salida:
# Número: 1
# Número: 2
# Número: 3
# Número: 4
# Número: 5
```

Rangos y secuencias

`{1..5}` o `seq 1 5` generan secuencias numéricas

Estructuras de control: while

`while` `do` `done`

Sintaxis

`while` inicia el bucle condicional

`do` inicia el bloque de código

`done` cierra el bucle while

Se ejecuta mientras la condición sea

Ejemplo: Contador

```
#!/bin/bash

count=1

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

# Salida:
# Contador: 1
# Contador: 2
# Contador: 3
```

Importante

¡Cuidado con bucles infinitos! Asegúrate que la condición cambie

Estructuras de control: until

`until` `do` `done`

Sintaxis

`until` inicia el bucle inverso

`do` inicia el bloque de código

`done` cierra el bucle until

While se ejecuta mientras es | Until se ejecuta hasta que sea

Ejemplo: Contador

```
#!/bin/bash

count=1

until [ $count -gt 3 ]
do
  echo "Contador: $count"
  count=$((count + 1))
done

# Salida:
# Contador: 1
# Contador: 2
# Contador: 3
```

Uso principal

Ideal para esperar un evento o hasta que se cumpla una condición específica

Preguntas de evaluación

¿Qué valor devuelve el comando `test` cuando la condición es verdadera?

A) 1

B) 0

C) -1

D) 2

¿Qué opción del comando `test` verifica si un archivo existe?

A) -f

B) -d

C) -e

D) -s

¿Cuál es el operador correcto para comparar dos números iguales en el comando `test`?

A) ==

B) =

C) -eq

D) -equal

¿Qué estructura de control es más adecuada para múltiples condiciones similares?

A) if-elif-else

B) for

C) case

D) while

¿Cuál es la diferencia principal entre corchetes `[ ]` y

¿Qué bucle se ejecuta mientras la condición sea