

## Estructura y ejecución

### Shebang (#!)

- Ruta absoluta al intérprete en la linea 1
- #!/bin/bash (estandar para scripts robusteros)
- Permite ejecutar el script directamente (`./script.sh`)

en lugar de involucrarse

(Permisos)

- Error common : Permission denied
- Solucion : Chmod +x Script.sh

## ② Argumentos y variables

- variables = valor ( Si expresión en el interpolación :  
= )
- uso : "fvariables" ( comillas dobles permiten expansión )
- Comillas simples '\$var' interpretan el texto literalmente ( útil para regex )
- Command substitution
  - \* Guardar output de un comando de variable
- Sintaxis moderna : Fecha = \$(date)

variables especiales

\$0 nombre al script

\$1, \$2 argumentos posicionales pasados al script

\$? exit code de ultimo comando ejecutando ( 0 = éxito , 1 - SS = error )

### ③ Logica de Control

if / Else

```
if [ "$1" = "PROD" ]; then
    echo "Desplegando en Producción..."
elif [ "$1" = "DEV" ]; then
    echo "Ambiente de Pruebas"
else
    echo "Error: Ambiente desconocido"
    exit 1 # Forzar error
fi
```

test operador ( test o [ ] )

• archivos

- f : existen y es archivo regular

- d : existe y es directorio

- S : existe y tamaño > 0 (no estro  
vacio)

• Strings

= / != : Comparacion exacta

- z String vacio ( length zero )

- **Entorno**

- eq (equal), -ne (not equal)
- gt (greater than), -lt (less than)

- **Login**

- a (and), -o (or)

①

## Bucle

- **For loops**

uso crítico en hilo: proceso múltiples cvs en un directorio

### Bash

```
# Globbing (*) para iterar archivos
for ARCHIVO in /data/incoming/*.csv; do
    echo "Procesando $ARCHIVO..."
    # Aquí iría tu script de Python o comando de carga
    python process_data.py "$ARCHIVO"
done
```

- **While Loops:**

- Ejecutar mientras una condición sea verdadera (ej. esperar a que aparezca un archivo).

### Bash

```
CONTADOR=0
while [ $CONTADOR -lt 10 ]; do
    # Lógica aritmética
    CONTADOR=$((CONTADOR + 1))
done
```