Bash Scripting Cheatsheet

> **Curso:** Google IT Automation with Python
> **Módulo:** Bash Scripting
> **Tema:** Automatización de tareas en Linux/Unix con scripts Bash

Introducción

Bash es un **shell** y **lenguaje de scripting** usado para automatizar tareas en sistemas Linux/Unix.
Permite ejecutar comandos, manipular archivos y crear scripts reutilizables.

- Archivo de script: extensión `.sh`
- Ejecutable con:

```
bash script.sh
# o
./script.sh   # si tiene permisos de ejecución
```

- Dar permisos de ejecución:

```
chmod +x script.sh
```

Estructura Básica de un Script

```
#!/bin/bash
# Esto es un comentario

echo "Hola Mundo"        # Imprime texto en pantalla
```

- `#!/bin/bash` → shebang, indica que se usará Bash para ejecutar el script
- `#` → comentarios
- `echo` → imprimir en terminal

Variables

```
nombre="Juan"
echo "Hola $nombre"

NUM1=5
```

```
NUM2=10
SUM=$((NUM1 + NUM2))
echo "Suma: $SUM"
```

- No se usan espacios alrededor de `=`
- Referencia con `$`
- Expresiones aritméticas: `$(( ... ))`

◈ Leer entrada del usuario

```
read -p "Ingrese su nombre: " NOMBRE
echo "Hola $NOMBRE"
```

◈ Condicionales

```
if [ $NUM1 -gt $NUM2 ]; then
    echo "$NUM1 es mayor que $NUM2"
elif [ $NUM1 -eq $NUM2 ]; then
    echo "Son iguales"
else
    echo "$NUM1 es menor que $NUM2"
fi
```

Comparaciones comunes

Operador	Descripción
`-eq`	Igual a
`-ne`	Diferente de
`-gt`	Mayor que
`-ge`	Mayor o igual que
`-lt`	Menor que
`-le`	Menor o igual que

◈ Bucles

For loop

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

- Itera sobre una lista de valores
- También se puede usar `for i in {1..5}`

While loop

```
COUNT=1
while [ $COUNT -le 5 ]; do
    echo "Count: $COUNT"
    ((COUNT++))
done
```

- `((COUNT++))` → incremento de variable

◈ Funciones

```
saludar() {
    echo "Hola $1"
}

saludar "Ana"
```

- `$1, $2, ...` → argumentos de la función

◈ Operaciones con Archivos

- Listar archivos: `ls`
- Copiar: `cp origen destino`
- Mover/renombrar: `mv origen destino`
- Borrar: `rm archivo` / `rm -r carpeta`
- Comprobar existencia:

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

if [ -d "carpeta" ]; then
    echo "Directorio existe"
fi
```

Redirecciones y Pipes

Símbolo	Función	Ejemplo
`>`	Sobrescribir archivo	`echo "Hola" > archivo.txt`
`>>`	Agregar a archivo	`echo "Mundo" >> archivo.txt`
`<`	Entrada desde archivo	`sort < archivo.txt`
`|`	Pipe, pasar salida de un comando a otro	`cat archivo.txt | grep "hola"`
`2>`	Redirigir errores	`ls noexiste 2> error.log`

Comandos Útiles

- `pwd` → muestra directorio actual
- `cd` → cambiar directorio
- `touch` → crear archivo vacío
- `mkdir` → crear directorio
- `rm` → eliminar archivo/directorio
- `chmod` → cambiar permisos
- `chown` → cambiar propietario
- `grep` → buscar texto dentro de archivos
- `cut` → extraer columnas de texto
- `awk` / `sed` → procesamiento avanzado de texto

Buenas Prácticas

- Siempre usar **shebang** `#!/bin/bash`
- Comentar bloques de código importantes
- Verificar existencia de archivos antes de operar
- Evitar sobrescribir archivos importantes
- Usar funciones para código repetitivo
- Probar scripts con `bash -n script.sh` para detectar errores de sintaxis

Referencias oficiales

- GNU Bash Manual
- Bash Scripting Tutorial – LinuxConfig
- Shell Scripting – tldp.org