Módulo 5: Variables, entorno y scripting básico

Objetivos

- Declarar y usar variables de shell (locales y de entorno).
- Modificar el entorno con `export` y entender la variable `PATH`.
- Crear alias temporales y permanentes en `/.bashrc` o `/.bash_aliases`.
- Escribir scripts simples de Bash con `#!/usr/bin/env bash`.
- Usar argumentos posicionales (`$1`, `$2`, `$@`, `$#`).
- Implementar condicionales (`if`) con operadores de test (archivos, cadenas, números).
- Escribir bucles `for` para iterar sobre listas.
- Realizar operaciones aritméticas con `$(( ))`.
- Leer entrada del usuario con `read` usando opciones como `-p`, `-t`, `-s`.

Contenido teórico

Variables de shell

Las variables almacenan datos temporales. Se crean con `nombre=valor` (sin espacios alrededor del =). Para acceder a su valor usamos `$nombre` o `${nombre}` (las llaves son necesarias si el nombre va seguido de otros caracteres).

Bash 1: Variables básicas

```
mensaje="Hola Bash"          # variable 'mensaje' <- "Hola Bash"
echo $mensaje                # Hola Bash
numero=42                    # variable 'numero' <- 42
echo "El valor es $numero"   # El valor es 42
```

Variables de entorno

Las variables de entorno son heredadas por los procesos hijos. Para convertir una variable local en variable de entorno usamos `export`.

```
MI_VAR="local"          # solo en este shell
export MI_VAR           # ahora la ven los subprocesos
```

La variable `PATH` es una de las más importantes: contiene la lista de directorios donde Bash busca ejecutables. Para añadir un directorio personal:

```
export PATH="$HOME/mis_scripts:$PATH"
```

Alias

Los alias son atajos para comandos largos. Se definen con `alias nombre='comando'`.

```
alias ll='ls -la'
alias actualizar='sudo apt update && sudo apt upgrade'
```

Para hacerlos permanentes, se suelen añadir a `~/.bashrc`. Sin embargo, muchos sistemas incluyen un archivo específico `~/.bash_aliases` que se carga desde `~/.bashrc` si existe. Por claridad, es recomendable guardar los alias en `~/.bash_aliases`. Luego, para aplicar los cambios, ejecuta `source ~/.bashrc` o abre una nueva terminal.

Scripts de Bash: el shebang

Un script es un archivo con comandos Bash. La primera línea suele ser el **shebang** (`#!`) que indica qué intérprete usar.

Bash 2: Script mínimo

```
#!/usr/bin/env bash
echo "Hola desde un script"
```

Tip 11: `#!/usr/bin/env bash` vs `#!/bin/bash`

`#!/usr/bin/env bash` busca el ejecutable **`bash`** en el **`PATH`** del usuario. Es más portable, especialmente en sistemas donde Bash no está en **`/bin`** (por ejemplo, FreeBSD, NixOS, o entornos virtuales). `#!/bin/bash` asume una ubicación fija que no siempre existe.

Recomendación: usa `#!/usr/bin/env bash` para scripts que compartirás en diferentes entornos.

Para ejecutar un script, primero hazlo ejecutable:

```
chmod +x mi_script.sh
./mi_script.sh
```

Argumentos posicionales

Dentro de un script, puedes acceder a los parámetros pasados:

- `$0` – nombre del script.
- `$1`, `$2`, ... – primer, segundo argumento.
- `$@` – todos los argumentos (como lista).
- `$#` – número de argumentos.

Bash 3: Script que saluda

```
#!/usr/bin/env bash
echo "Hola, $1. Bienvenido a Bash."
echo "Recibiste $# argumentos."
```

Ejecución: `./saludo.sh Ana` → `Hola, Ana. Bienvenido a Bash. Recibiste 1 argumentos`

.

Condicionales: `if`

Sintaxis:

```
if [ condicion ]; then
    comandos
elif [ otra ]; then
    comandos
else
    comandos
fi
```

Tip 12: Espacios en los corchetes

`[ condición ]` requiere espacios después de `[` y antes de `]`. Es un comando (test).

No escribir `[condición]`.

Operadores comunes:

- Archivos: `-f archivo` (existe y es regular), `-d directorio`, `-r`, `-w`, `-x`.
- Cadenas: `"$a" = "$b"` (igual), `"$a" != "$b"` (distinto), `-z "$a"` (vacío).
- Números: `$a -eq $b` (igual), `-ne`, `-lt`, `-le`, `-gt`, `-ge`.

Bash 4: Ejemplo de condicional

```
#!/usr/bin/env bash
if [ -f "$1" ]; then
    echo "$1 existe y es un archivo regular."
elif [ -d "$1" ]; then
    echo "$1 es un directorio."
else
    echo "$1 no existe o no es regular/directorio."
fi
```

Bucles `for`

Iteran sobre una lista de elementos.

Bash 5: Ejemplos de `for`

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

```bash
for archivo in *.txt; do
    echo "Procesando $archivo"
done

for nombre in Ana Juan Luis; do
    echo "Hola $nombre"
done
```

Aritmética en Bash: `$(( ))`

Para realizar operaciones matemáticas (sumas, restas, multiplicaciones, etc.) usamos la **expansión aritmética** con el formato `$(( expresión ))`. Todo lo que esté dentro se evalúa como una expresión numérica entera.

Bash 6: Ejemplos aritméticos

```bash
echo $(( 5 + 3 ))        # 8
echo $(( 10 - 2 ))       # 8
echo $(( 7 * 6 ))        # 42
echo $(( 20 / 4 ))       # 5 (división entera)
echo $(( 17 % 5 ))       # 2 (resto)
```

También podemos usar variables (sin necesidad de `$` dentro de la expresión, aunque opcionalmente se puede):

```bash
a=10
b=3
echo $(( a + b ))        # 13
echo $(( a * b ))        # 30
suma=$(( a + b ))        # asignamos el resultado a una variable
echo $suma               # 13
```

Esta sintaxis es muy útil en bucles y condicionales para actualizar contadores o acumuladores.

Leer entrada: `read`

El comando `read` lee una línea desde la entrada estándar (normalmente el teclado) y la guarda en una o más variables.

Opciones comunes:

- `-p "texto"` – muestra un prompt antes de leer.
- `-t segundos` – tiempo de espera (timeout) para que el usuario responda.
- `-s` – modo silencioso (no muestra lo que se teclea, útil para contraseñas).

Si no se especifica ninguna variable, el texto se guarda en `$REPLY`.

Bash 7: Ejemplos de `read`

```
# Lectura simple
read nombre
echo "Hola $nombre"

# Con prompt
read -p "Introduce tu edad: " edad
echo "Tienes $edad años."

# Con timeout y modo silencioso (contraseña)
read -t 10 -s -p "Contraseña: " pass
echo -e "\nGracias"
```

Combinando conceptos: script integrador

Bash 8: Script que suma números pasados como argumentos

```
#!/usr/bin/env bash
suma=0
for num in "$@"; do
    suma=$((suma + num))
done
echo "La suma de los argumentos es: $suma"
echo "Cantidad de argumentos: $#"
```

Recordatorio de tips de módulos anteriores

- `Ctrl+R` búsqueda inversa en historial.
- `TAB` autocompletado.
- Expansión de llaves `{a,b,c}`.
- `tee` para ver y guardar salida.

Ejercicios prácticos

1. Crea una variable `NOMBRE` con tu nombre y otra `EDAD` con tu edad. Muestra: "Me llamo [nombre] y tengo [edad] años".

2. Exporta una variable `MI_VAR="desde el shell"` y luego ejecuta `bash` (subshell). Comprueba que la variable está disponible con `echo $MI_VAR`.

3. Define un alias llamado `actualizar` que ejecute `sudo apt update && sudo apt upgrade -y` (si usas Debian/Ubuntu) o el comando equivalente para tu distribución. Añádelo a `~/.bash_aliases`.

4. Escribe un script `backup.sh` que reciba un directorio como argumento y cree una copia comprimida con fecha (usa `tar -czf backup-$(date +%Y%m%d).tar.gz directorio`).

5. Crea un script `verificar.sh` que reciba un archivo como argumento y muestre un mensaje indicando si es un archivo regular, un directorio o no existe.

6. Escribe un script `suma_lista.sh` que sume todos los números pasados como argumentos (usa `for` y `$(( ))`).

7. Haz un script `pregunta.sh` que use `read` para preguntar el nombre del usuario, el directorio que quiere listar, y luego muestre el contenido de ese directorio.

8. (Desafío) Crea un script `organizar.sh` que cree una estructura como por ejemplo `proyecto/{src,doc,backup}` (como vimos en módulo 1) usando un bucle o directamente con expansión de llaves. Luego, dentro de cada subdirectorio, crea un archivo `README.md` con contenido descriptivo.

Referencias

- The Linux Command Line (TLCL) – Capítulos 11-15.
- The Bash Guide – Secciones sobre variables, scripting.
- Manual oficial de Bash.

Resumen

En este módulo aprendiste a:

- Usar variables locales y de entorno, y modificar `PATH`.
- Crear alias temporales y permanentes (en `/.bash_aliases`).
- Escribir scripts portables con `#!/usr/bin/env bash`.
- Acceder a argumentos posicionales y manejar condicionales y bucles.
- Realizar operaciones aritméticas con `$(( ))`.
- Interactuar con el usuario mediante `read` y sus opciones.

Ya puedes automatizar tareas sencillas. En el próximo módulo aplicarás todo lo aprendido en un proyecto integrador.