IMPRESCINDIBLES

1. Si no lo pones al principio del script, suspendes:

   #!/bin/bash

2. Llamar función con todos los parámetros pasados:

  llamar_funcion "$@"

3. Llamar función con parámetros específicos:

  llamar_funcion melón 2 * 1.50

4. Crear script en consola:

  nano nombre_script.sh

5. Dar permisos al script:

  chmod +x nombre_script.sh

6. Ejecutar script:

  ./nombre_script.sh

7. Redirección de archivos:

  >: redirigir la salida de un comando a un archivo, sobrescribiendo el contenido del archivo si ya existe. Si el archivo no existe, se creará uno nuevo.
  >>: redirigir la salida de un comando agregándola al final de un archivo, sin sobrescribir su contenido actual. Si el archivo no existe, se creará.

8. Condiciones:

  TEMA 4: (comillas, comparaciones, etc.): https://github.com/STEMgranada/Sistemas-Informaticos/blob/main/UD4/Condiciones.md

  TEMA 5 (modificadores): https://github.com/STEMgranada/Sistemas-Informaticos/blob/main/UD5/condiciones.md

---

FUNCIONES

Función restar números con shift

```
restar_numeros_al_primer_numero() {
  total=$1
  shift
  while [ $# -gt 0 ]; do
    total=$(( total - $1 ))
    shift
  done
  echo "La cifra total es: $total"
}

restar_numeros_al_primer_numero 100 20 1 13
```

Función multiplicar números y restar el último

```
multiplicar_numeros_y_restar_el_ultimo() {
  total=$1
  shift
  while [ $# -gt 1 ]; do
    total=$(( total * $1 ))
    shift
  done

  total=$(( total - $1))
  echo "La cifra total es: $total"
}
```

Función para validar si un número es mayor a cero

```
validar_numero() {
  numero=$1
  resultado="ok"
  if [ "$numero" -le 0 ] 2>/dev/null; then
    resultado="error"
  fi
  echo "$resultado"
}
```

Función para validar el número de argumentos

```
validar_argumentos() {
  esperados=$1
  recibidos=$2
  resultado="ok"
  if [ "$recibidos" -ne "$esperados" ]; then
    resultado="error"
  fi
  echo "$resultado"
}
```

Función para calcular promedio

```
calcular_promedio() {
  suma=0
  cantidad=$#

  while [ "$#" -gt 0 ]; do
    suma=$((suma + $1))
    shift
  done

  promedio=$((suma / cantidad))
  mostrar_mensaje "El promedio de los números es: $promedio"
  mostrar_mensaje "La suma de los números es: $suma"
}
```

Función para concatenar palabras

```
concatenar_palabras() {
  resultado="$1"
  shift

  while [ "$#" -gt 0 ]; do
    resultado="$resultado-$1"
    shift
  done

  echo "$resultado"
}
```

VARIABLES

- $#: número de argumentos pasado al script o función.
Ejemplo:

```
if [ "$#" -eq 0 ] || [ -z "$artista" ]; then
    mostrar_error "No se ha proporcionado un artista y álbumes." (Relación 4, archivar_albumes.sh)
```

- $@: número total de argumentos pasados al script.

MODIFICADORES (Hay más en el archivo UD5/condiciones)

1. -z: verifica que si una cadena está vacía.
Ejemplo 1:

```
if [ -z "$temperatura" ] || [ -z "$unidad" ]; then
    mensaje="Uso: convertir_temperatura <temperatura> <C|F>"
```

Ejemplo 2:

```
if [ -z "$password" ]; then
    mensaje="Error: Debes proporcionar una contraseña."
```

2. -d: verifica si el directorio existe.
Ejemplo:

```
if [ ! -d "$directorio" ]; then
    mensaje="Error: El directorio no existe."
```

3. -f: verifica si un archivo existe:
Ejemplo:

```
if [ ! -f "$archivo" ]; then
    mensaje="Error: El archivo no existe."
```

4. -s: verifica que el archivo no esté vacío.
Ejemplo:

```
if [ ! -s "$INVENTARIO" ]; then
    echo "? No hay productos en el inventario." (Tanda 1, ikea.sh)
```

COMANDOS

- cantidad=$(find "$directorio" -type f | wc -l): contar cantidad de archivos en el directorio.

```
- cantidad=$(wc -w < "$archivo")
    mensaje="El archivo $archivo tiene $cantidad palabras."
```

- | grep -q '[!@#$%^&*]'; then: busca los caracteres especiales en la cadena.
Ejemplo:

elif ! echo "$contraseña" | grep -q '[!@#$%^&*]'; then (Relación 2, ejercicio 2)

- lineas_codigo=$(grep -v '^\s*$' "$archivo" | grep -v '^\s*#' | wc -l)
    mensaje="El archivo $archivo tiene $lineas_codigo líneas de código.": para contar líneas de código. (Relación 2, ejercicio 3)

- mkdir -p "ruta": crear árbol de directorios.

- mv "ruta_origen" "ruta_destino": mover archivo de sitio.
Ejemplo: mv "discografica/$artista/$1.txt" "discografica/archivados/"

- touch "nombre_archivo.txt" o "ruta_con_archivo": crear archivo.
Ejemplo: touch "discografica/$artista/$nombre_album.txt"

- | bc: permite cálculos matemáticos.
Ejemplo: area=$(echo "3.1416 * $radio * $radio" | bc) (Tanda 1, calculadora_areas.sh)

- | bc -l: permite cálculos con decimales.
Ejemplo: cantidad=$(echo "$cantidad / 1000" | bc -l) (Tanda 1, conversor_longitud.sh)

- **shift**: después de ejecutar shift, el primer argumento se elimina y todos los demás argumentos se desplazan una posición.

Por ejemplo, si la función recibe los argumentos 5 3 7, después de shift:

    $1 será 3 (el segundo argumento original).
    $2 será 7 (el tercer argumento original).

(Ejemplo de su funcinamiento básico: Tanda 1, diferencia_numeros.sh)

- **READs**

  1. read: se utiliza para leer una línea de entrada y almacenarla en una variable.
  2. read -p: permite mostrar un mensaje de solicitud antes de que el usuario ingrese la información. Luego, el valor ingresado por el usuario se almacena en la variable. (Similara a lo que nosotros conocemos por un *prompt*.)
  Ejemplo:

  Uso:
  read -p "¿Cómo te llamas? " nombre
  echo "Hola, $nombre"

  Salida:
  ¿Cómo te llamas? Ana
  Hola, Ana

  3. read -a: se usa para leer una línea de texto y dividirla en un arreglo (array). Cada palabra o conjunto de palabras separadas por espacios se almacena en un elemento diferente del arreglo. (Tanda 1, ikea.sh)

  4. read -r: las barras invertidas no se interpretan y se leen tal cual como son
  No interpretará ninguna secuencia de escape, por lo que si el usuario ingresa hello\nworld, el valor de texto será exactamente hello\nworld (incluyendo la barra invertida), en lugar de separar las líneas.

- sleep: pausar la ejecución de un script o de una línea de comandos durante un período de tiempo específico. (Ejemplo en Tanda 2, carrera_caballos.sh)

  · Segundos: Si no se especifica una unidad, sleep asume que la duración es en segundos.

· Minutos: Puedes usar la unidad m para minutos.
· Horas: Puedes usar la unidad h para horas.
· Días: Puedes usar la unidad d para días.
· Milisegundos: Puedes usar la unidad s para milisegundos (si se especifica un valor decimal).

OPERADORES LÓGICOS

- Ejemplo de operador lógico **OR o ||**:

```
 if [ -z "$num1" ] || [ -z "$operador" ] || [ -z "$num2" ]; then
    mensaje="Uso: calculadora <num1> <operador> <num2>"
```

- Ejemplos de **negación** lógica:

1. elif ! [["$password" =~ [A-Z]]]; then: Negación de comparación con expresión regular.

2. if [! -f "$archivo"]; then: negación tradicional.

- Ejemplo de operador lógico **AND o &&**:

```
if [ "$operador" = "/" ] && [ "$numero" -eq 0 ]; then
    resultado="error"
```

OTROS OPERADORES

1. ${#variable}: devuelve el número de caracteres de la cadena de texto contenida en la variable.

Ejemplo:

```
 elif [ ${#password} -lt 8 ]; then
    mensaje="Error: La contraseña debe tener al menos 8 caracteres."
```

Si password="abc123", entonces ${#password} devolverá 6, que es la cantidad de caracteres en la cadena.

2. =~: comparación con una expresión regular.

Ejemplo:

```
elif ! [[ "$password" =~ [A-Z] ]]; then
    mensaje="Error: La contraseña debe contener al menos una letra mayúscula."
elif ! [[ "$password" =~ [0-9] ]]; then
    mensaje="Error: La contraseña debe contener al menos un número."
```

ARRAYs

Declarar un arreglo

```
frutas=("manzana" "banana" "cereza")
```

Acceder a elementos

```
echo ${frutas[0]}  # manzana
echo ${frutas[1]}  # banana
```

Modificar un elemento
```
frutas[2]="kiwi"
echo ${frutas[2]}  # kiwi
```

Obtener el tamaño del arreglo
```
echo "Número de frutas: ${#frutas[@]}"  # 3
```

Agregar un elemento
```
frutas+=("mango")
echo ${frutas[@]}  # manzana banana kiwi mango
```

Recorrer el arreglo
```
for fruta in "${frutas[@]}"; do
    echo $fruta
done
```

SINTÁXIS BÁSICA

- Bucle **WHILE**

```
 while [ condicion ]; do

 done
```

- **IF, ELIF, ELSE**

```
if [ condición ]; then
    # Código a ejecutar si la condición es verdadera
elif [ otra_condición ]; then
    # Código a ejecutar si la primera condición es falsa y esta segunda es
verdadera
else
    # Código a ejecutar si todas las condiciones anteriores son falsas
fi
```

- Operaciones **MATEMÁTICAS**

```
1. Operaciones con enteros: (( ... ))
Ejemplo: resultado=$((a + b))

2. Operaciones con decimales: echo " ... " | bc
Ejemplo: resultado=$(echo "$a + $b" | bc)

3. Incremento: ((a++))
4. Decremento: ((a--))
```

- **MENÚ** (Versión con saltos de línea, en Tanda 1, ikea.sh hay una versión SIN)

```
mostrar_menu() {

echo "? IKEA - Sistema de Gestión de Inventario ?"
echo -e "\n1. Agregar producto al inventario
\n2. Mostrar productos en inventario
\n3. Realizar un pedido
\n4. Listar pedidos realizados
\n5. Salir"

}
```

- **RANDOMs**

1. Generar un número aleatorio dentro de un rango ente 0 y número.

   echo $((RANDOM % N))

   Ejemplo: echo $((RANDOM % 100)) --> (0-99)

2. Generar un número aleatorio en un rango específico.

   echo $((A + RANDOM % (B - A + 1)))

   Ejemplo: echo $((50 + RANDOM % (100 - 50 + 1))) --> (50-100, ambos incluídos)

MANEJO DE ERRORES (Con return)

Ejemplo:

```
mi_funcion() {
if [ "$1" -lt 0 ]; then
  echo "Error: El número no puede ser negativo."
  return 1  # Devuelve un código de error
else
  echo "El número es positivo."
  return 0  # Indica éxito
fi
}

mi_funcion -5
echo "El valor de retorno fue: $?"  # 1 (error)

mi_funcion 5
echo "El valor de retorno fue: $?"  # 0 (éxito)
```

return *se utiliza para finalizar funciones y devolver un valor a la función que la llamó, sin detener el script.*
exit *se utiliza para terminar el script completo y devolver un valor de salida al sistema operativo.*