Resumen: Iniciación a los Scripts en Linux

1. Conceptos básicos

- **Script**: Archivo de texto con una serie de comandos y estructuras organizadas secuencialmente que permite automatizar tareas.
- La primera línea debe ser **#!/bin/bash** (shebang) para indicar el intérprete.
- Para ejecutar: dar permisos con **chmod +x script.sh** y ejecutar con **./script.sh**.
- Para depurar: **sh -x nombre_script**.

2. Comandos y Estructuras Principales

2.1 Salida y entrada estándar

- **echo**: echo [opciones] [cadena] - Muestra texto en pantalla
  - echo "Texto \n con salto de línea"
- **Tipos de comillas**:
  - Simples ' ': Se interpreta literalmente
  - Dobles " ": Interpreta caracteres especiales como $ y `
  - Invertidas ` `: Ejecuta el comando entre ellas
- **Variables**:
  - Asignar: nombre_variable=valor
  - Mostrar: echo $nombre_variable
  - Leer entrada: read [opciones] variable
  - Capturar comando: variable=$(comando) o variable=`comando`
  - Exportar: export variable

2.2 Estructuras condicionales

- **if-then-else**:

```
if [ condición ]
then
   bloque1
else
   bloque2
fi
```

- **if-elif-else**:

```
if [ condición1 ]
then
   bloque1
elif [ condición2 ]
then
   bloque2
else
   bloque3
fi
```

- **case**:
  case $variable in
    valor1)
      bloque1
      ;;
    valor2)
      bloque2
      ;;
    *)
      bloque_por_defecto
      ;;
  esac

- ***Operadores de comparación***
  - **Numéricos**: -eq (igual), -ne (distinto), -gt (mayor), -lt (menor), -ge (mayor o igual), -le (menor o igual)
  - **Cadenas**: =, !=, -z (cadena vacía), -n (cadena no vacía)
  - **Ficheros**: -f (existe y es fichero), -d (existe y es directorio), -r (permiso lectura), -w (permiso escritura), -x (permiso ejecución)
  - **Conectores lógicos**: -a (AND), -o (OR)

2.3 Estructuras repetitivas

- **for**:
  for variable in conjunto
  do
    bloque
  done

- **while**:
  while [condición]
  do
    bloque
  done

- **until**:
  until [condición]
  do
    bloque
  done

- **Comandos de control**: break (salir del bucle), continue (saltar a siguiente iteración)

2.4 Operaciones algebraicas

- **Con enteros**: expr arg1 operación arg2
- **Operaciones**: +, -, * (con escape *), /, % (módulo)
- Para decimales se puede usar el comando bc

2.5 Paso de parámetros y funciones

- Parámetros script: $1, $2, etc. para acceder a los argumentos
- $#: Número de parámetros pasados
- $* o $@: Todos los parámetros

- **Funciones**:
  ```
  nombre_funcion() {
     bloque
     return valor
  }
  # Llamada:
  nombre_funcion parámetros
  ```

- $?: Código de retorno (valor devuelto por el último comando o función ejecutada)

3. Cuándo usar cada tipo de estructura en scripts de Linux

3.1 Estructuras condicionales

- **if-then simple**
  - **Cuándo usar**: Cuando necesitas ejecutar un bloque de código solo si se cumple una condición específica.
  - Ejemplo: Verificar si un archivo existe antes de manipularlo.

- **if-then-else**
  - **Cuándo usar**: Cuando necesitas ejecutar un bloque de código si se cumple una condición y otro bloque diferente si no se cumple.
  - Ejemplo: Comprobar si un usuario tiene permisos; si los tiene, continuar; si no, mostrar un mensaje de error.

- **if-elif-else**
  - **Cuándo usar**: Cuando tienes múltiples condiciones que comprobar secuencialmente.
  - Ejemplo: Evaluar diferentes rangos de valores (como calificaciones: A, B, C, etc.)

- **case**
  - **Cuándo usar**: Cuando una variable puede tomar varios valores discretos y necesitas ejecutar código diferente para cada posible valor.
  - Ejemplo: Menús de opciones donde cada opción ejecuta una función diferente.
  - Es más legible que múltiples if-elif anidados cuando trabajas con valores específicos.

3.2 Estructuras de bucle

- **for**
  - **Cuándo usar**: Cuando necesitas iterar sobre un conjunto conocido de elementos o un número definido de veces.
  - Ejemplo: Procesar archivos en un directorio, procesar cada palabra de una lista, o ejecutar algo un número específico de veces.

- **while**
  - **Cuándo usar**: Cuando necesitas repetir un bloque de código mientras se cumpla una condición (sin saber de antemano cuántas iteraciones serán).
  - Ejemplo: Leer líneas de un archivo hasta llegar al final, esperar a que un recurso esté disponible.
  - Ideal para "bucles controlados por centinela" donde la condición de salida puede ocurrir en cualquier momento.

- **until**
  - **Cuándo usar**: Similar al while, pero se usa cuando es más intuitivo expresar "repetir hasta que algo sea verdadero" en lugar de "mientras algo sea falso".
  - Ejemplo: Esperar hasta que un servicio esté activo.

3.3 Parámetros y funciones

- **Scripts con parámetros**
  - **Cuándo usar**: Cuando necesitas que tu script sea flexible y pueda procesar diferentes entradas sin modificar el código.
  - Ejemplo: Scripts de utilidades generales que pueden actuar sobre diferentes archivos o valores.

- **Funciones**
  - **Cuándo usar**: Cuando tienes código que se repite en varias partes del script o cuando quieres modularizar el código para hacerlo más legible y mantenible.
  - Ejemplo: Operaciones comunes como validación de entradas, formateo de salidas, o cualquier tarea reutilizable.

- **Scripts que retornan valores**
  - **Cuándo usar**: Cuando necesitas que un script comunique su resultado a otro script o al sistema.
  - Ejemplo: Scripts que realizan verificaciones y necesitan comunicar éxito/fracaso a quien los invocó.

3.4 Otros tipos específicos

- **Scripts de procesamiento de archivos**
  - **Cuándo usar**: Para automatizar tareas de manipulación, análisis o transformación de archivos.
  - Ejemplo: Scripts como LeerFichero.sh que procesan contenido línea por línea.

- **Scripts interactivos con menús**
  - **Cuándo usar**: Cuando quieres crear una interfaz amigable para usuarios que pueden no estar familiarizados con la línea de comandos.
  - Ejemplo: FechaCalendario.sh, que presenta opciones en un menú y responde a la elección del usuario.

- **Scripts de automatización**
  - **Cuándo usar**: Para tareas repetitivas que siguen un patrón predecible y que consumen tiempo si se hacen manualmente.
  - Ejemplo: Copias de seguridad, monitoreo de sistemas, generación de informes.