Guía: Scripting con bash

¿Qué es Bash? El poder de programar en terminal

Bash (Bourne Again Shell) es un intérprete de comandos o shell que proporciona una interfaz de línea de comandos para interactuar con el sistema operativo Linux y otros sistemas Unix-like. Es el shell predeterminado en la mayoría de las distribuciones de Linux.

Bash permite a los usuarios ejecutar comandos y scripts para realizar diversas tareas, como administrar archivos y directorios, ejecutar programas, automatizar tareas, administrar procesos y mucho más. Es extremadamente poderoso y versátil, lo que lo convierte en una herramienta esencial tanto para usuarios novatos como para expertos en Linux.

Aquí tienes un ejemplo simple para entender Bash:

Supongamos que queremos listar todos los archivos en un directorio. Podemos hacerlo fácilmente utilizando el comando `ls` en Bash. Por ejemplo:

```
bash
```

```
ls
```

Este comando mostrará una lista de todos los archivos y directorios en el directorio actual.

Ahora, hablemos sobre algunas cosas que puedes hacer con Bash:

1. **Automatización de tareas**: Puedes escribir scripts en Bash para automatizar tareas repetitivas, como realizar copias de seguridad, procesar archivos, administrar usuarios, etc.
2. **Administración del sistema**: Bash te permite administrar archivos y directorios, instalar y desinstalar programas, monitorear procesos, configurar el sistema y realizar otras tareas de administración del sistema.
3. **Programación de shell**: Bash es un lenguaje de programación de shell completo con características como variables, estructuras de control, funciones, bucles, y más. Puedes escribir scripts complejos para realizar diversas tareas utilizando Bash.
4. **Interfaz de usuario**: Aunque Bash es principalmente una interfaz de línea de comandos, también puede ejecutar comandos en modo interactivo, permitiéndote realizar operaciones directamente desde la línea de comandos.

Bash es importante en el contexto de Linux y hacking por varias razones:

- **Flexibilidad y Potencia**: Bash proporciona una amplia gama de herramientas y características que facilitan el trabajo en entornos de Linux y Unix-like. Su flexibilidad y potencia lo hacen ideal para realizar diversas tareas, desde tareas simples hasta operaciones avanzadas.
- **Automatización**: En el hacking ético y la administración de sistemas, la automatización de tareas es fundamental para aumentar la eficiencia y reducir los errores. Bash permite automatizar fácilmente tareas repetitivas mediante la escritura de scripts.
- **Acceso a Herramientas de línea de comandos**: Muchas herramientas y utilidades de hacking ético se ejecutan desde la línea de comandos en Linux. Bash proporciona una interfaz para ejecutar estas herramientas y trabajar con sus salidas.
- **Personalización y Configuración**: Bash se puede personalizar y configurar según las necesidades del usuario, lo que permite adaptar el entorno de línea de comandos a las preferencias individuales y optimizar la productividad.

1. Comentarios

Los comentarios en Bash comienzan con el símbolo `#` y continúan hasta el final de la línea. Son útiles para agregar notas o explicaciones en tu código.

Ejempl

```
bash
```

```
# Este es un comentario en Bash
```

2. Declaración de intérprete

La primera línea de un script de Bash comúnmente especifica el intérprete que se utilizará para ejecutar el script. Por lo general, se usa `#!/bin/bash` para indicar que el script será interpretado por Bash.

Ejemplo:

```
bash
```

```
#!/bin/bash
```

3. Variables

Las variables en Bash se asignan utilizando el operador `=`. No debes dejar espacios alrededor del `=`. Las variables pueden contener letras, números y guiones bajos, pero no pueden comenzar con un número.

Ejemplo:

```
bash
```

```
nombre="Juan"
edad=25
```

4. Substitución de comandos

Puedes ejecutar comandos dentro de una cadena utilizando el símbolo de acento grave "`" o `$(...)` y capturar su salida.

Ejemplo:

```
bash
```

```
fecha_actual=`date`
o
fecha_actual=$(date)
```

5. Estructuras de control

a. Estructura if

La estructura `if` se usa para tomar decisiones basadas en condiciones.

Ejemplo:

```
bash
```

```
if [ "$edad" -ge 18 ]; then
    echo "Eres mayor de edad"
else
    echo "Eres menor de edad"
fi
```

b. Estructura for

La estructura `for` se utiliza para iterar sobre una lista de elementos.

Ejemplo:

```
bash
```

```
for i in {1..5}; do
    echo "Número: $i"
done
```

c. Estructura while

La estructura `while` se utiliza para ejecutar un bloque de código mientras una condición sea verdadera.

Ejemplo:

```
bash
```

```
contador=0
while [ $contador -lt 5 ]; do
    echo "Contador: $contador"
    contador=$((contador + 1))
done
```

d. Estructura case

La estructura `case` se utiliza para comparar una variable con una lista de patrones y ejecutar acciones basadas en la coincidencia.

Ejemplo:

```
bash
```

```
fruta="manzana"
case $fruta in
    manzana)
        echo "Es una manzana"
        ;;
    naranja)
        echo "Es una naranja"
        ;;
    *)
        echo "No reconocido"
        ;;
esac
```

6. Funciones

Puedes definir funciones en Bash usando la palabra clave `function` seguida por el nombre de la función y luego el bloque de código entre llaves `{}`.

Ejemplo:

```
bash
```

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

saludar "Juan"
```

7. Entrada del usuario

Puedes solicitar entrada al usuario utilizando el comando `read`.

Ejemplo:

```
bash
```

```
echo "¿Cuál es tu nombre?"
read nombre_usuario
echo "Hola $nombre_usuario, ¡bienvenido!"
```

8. Redirección y tuberías

Bash permite redirigir la entrada, salida y errores de los comandos utilizando `<`, `>`, `>>`, `|`, entre otros.

Ejemplo:

```
bash
```

```
# Redireccionamiento de salida estándar a un archivo
echo "Hola Mundo" > salida.txt

# Tubería para pasar la salida de un comando como entrada a o
tro
ls | grep ".txt"
```

Estos son los elementos básicos de la sintaxis de Bash con ejemplos para cada uno. ¡Espero que te resulte útil! Si tienes alguna pregunta adicional o necesitas más ejemplos, ¡no dudes en preguntar!

Si estás trabajando con VIM

Para salir de Vim y guardar los cambios realizados en el archivo, sigue estos pasos:

1. Asegúrate de estar en el modo de comando presionando la tecla `Esc`.
2. Escribe `:wq` y presiona `Enter`.

Esto guardará los cambios realizados en el archivo y luego saldrá de Vim. La secuencia `:wq` significa "write" (guardar) y "quit" (salir). Si el archivo aún no tiene un nombre, se te pedirá que lo guardes antes de salir.

¿Qué significa #!/bin/bash?

`#!/bin/bash` es una línea especial que se encuentra al principio de muchos scripts de Bash. Esta línea se conoce como "declaración de intérprete" o "shebang". Su significado es el siguiente:

- `#!`: El shebang es un carácter especial que indica al sistema operativo que el archivo debe ser ejecutado por el intérprete especificado a continuación.
- `/bin/bash`: Es la ruta al intérprete de comandos que se utilizará para ejecutar el script. En este caso, `/bin/bash` indica que el script será interpretado por el shell de Bash, que es un intérprete de comandos comúnmente utilizado en sistemas basados en Unix y Linux.

En resumen, `#!/bin/bash` le dice al sistema operativo que el archivo debe ser ejecutado por el intérprete de comandos Bash. Esto es esencial cuando ejecutas un script de Bash directamente desde la línea de comandos o desde otro script.

Operadores de Comparación:

1. **eq**: Igual a (equal)

   Ejemplo:

   ```
   bash
   ```

- ```
  if [ "$numero" -eq 10 ]; then echo "El número es igual a 10."
  fi
  ```
- **ne**: No es igual a (not equal)

  Ejemplo:
  ```

```
bash
```

- `if ["$numero" -ne 10]; then echo "El número no es igual a 10."
fi`
- **gt**: Mayor que (greater than)

  Ejemplo:

```
bash
```

- `if ["$numero" -gt 10]; then echo "El número es mayor que 10."
fi`
- **lt**: Menor que (less than)

  Ejemplo:

```
bash
```

- `if ["$numero" -lt 10]; then echo "El número es menor que 10."
fi`
- **ge**: Mayor o igual que (greater than or equal to)

  Ejemplo:

```
bash
```

- `if ["$numero" -ge 10]; then echo "El número es mayor o igual a 10."
fi`
- **le**: Menor o igual que (less than or equal to)

  Ejemplo:

```
bash
```

1. `if ["$numero" -le 10]; then echo "El número es menor o igual a 10."
fi`

Operadores Lógicos:

1. **&&**: Y lógico (AND)

   Ejemplo:

   ```
   bash
   ```

- `if [ "$edad" -ge 18 ] && [ "$ciudad" == "New York" ]; then echo "Eres mayor de edad y vives en Nueva York."`
  `fi`
- **||**: O lógico (OR)

  Ejemplo:

  ```
  bash
  ```

- `if [ "$tipo" == "administrador" ] || [ "$tipo" == "superusuario" ]; then echo "Tienes privilegios especiales."`
  `fi`
- **!**: Negación lógica (NOT)

  Ejemplo:

  ```
  bash
  ```

1. `if ! [ -e "$archivo" ]; then echo "El archivo no existe."`
   `fi`

Estos son algunos de los operadores de comparación y lógicos más comunes en Bash, junto con ejemplos de cómo se utilizan en declaraciones condicionales (`if`). Puedes combinar estos operadores para construir expresiones condicionales más complejas según sea necesario.

Condicionales en Bash:

1. **Estructuras de control condicional: if, elif y else:**

- En Bash, la estructura de control `if` se utiliza para tomar decisiones basadas en condiciones. Si la condición especificada se evalúa como verdadera, se ejecuta un bloque de código; de lo contrario, se pasa al siguiente bloque `elif` o `else`, si está presente.
- `elif` se utiliza para evaluar múltiples condiciones en secuencia si la condición `if` no se cumple.
- `else` se utiliza para especificar un bloque de código que se ejecuta si ninguna de las condiciones anteriores se cumple.

Ejemplo:

```
bash
```

- ```
  if [ "$edad" -ge 18 ]; then echo "Eres mayor de edad."
  elif [ "$edad" -ge 13 ]; then echo "Eres adolescente."
  else echo "Eres un niño."
  fi
  ```

- **Operadores de comparación en Bash: -eq, -ne, -lt, -gt, -le, -ge:**
  - Estos operadores se utilizan para comparar valores en Bash. Devuelven verdadero (0) o falso (1) dependiendo del resultado de la comparación.
  - `eq`: Igual a
  - `ne`: No igual a
  - `lt`: Menor que
  - `gt`: Mayor que
  - `le`: Menor o igual que
  - `ge`: Mayor o igual que

  Ejemplo:

  ```
  bash
  ```

- ```
  if [ "$edad" -ge 18 ]; then echo "Eres mayor de edad."
  fi
  ```

- **Uso de operadores lógicos: && (AND), || (OR), ! (NOT):**
  - Estos operadores se utilizan para combinar o negar condiciones en Bash.
  - `&&`: Operador AND, que devuelve verdadero si ambas condiciones son verdaderas.
  - `||`: Operador OR, que devuelve verdadero si al menos una de las condiciones es verdadera.
  - `!`: Operador NOT, que invierte el resultado de una condición.

  Ejemplo:

  ```
  bash
  ```

1. `if ["$edad" -ge 18] && ["$ciudad" == "New York"]; then echo "Eres mayor de edad y vives en Nueva York."
fi`

Bucles en Bash:

1. **Bucles for: cómo iterar sobre una secuencia de valores:**
   - Los bucles `for` se utilizan para iterar sobre una secuencia de valores, como elementos de un array, números en un rango, etc.
   - La sintaxis básica es `for variable in lista; do ...; done`, donde `lista` puede ser una lista de elementos separados por espacios.

   Ejemplo:

   ```
   bash
   ```

- `for i in {1..5}; do echo "Número: $i"
done`
- **Bucles while: cómo repetir un bloque de código mientras se cumpla una condición:**
  - Los bucles `while` se utilizan para repetir un bloque de código mientras se cumpla una condición específica.

- La sintaxis básica es `while condición; do ...; done`, donde `condición` es una expresión que se evalúa como verdadera o falsa.

Ejemplo:

bash

- `contador=0`
  `while [ $contador -lt 5 ]; do echo "Contador: $contador" contador=$((contador + 1))`
  `done`

- **Uso de la sentencia break y continue en bucles:**
  - `break` se utiliza para salir de un bucle de manera prematura cuando se cumple una condición específica.
  - `continue` se utiliza para saltar al siguiente ciclo de iteración cuando se cumple una condición específica.

Ejemplo:

bash

```
for i in {1..10}; do
    if [ $i -eq 5 ]; then
        break
    fi
    echo "Número: $i"
done
```

bash

```
for i in {1..10}; do
    if [ $((i % 2)) -eq 0 ]; then
        continue
    fi
```

```
    echo "Número impar: $i"
done
```