1. ¿Qué es el Shell Scripting? ¿A qué tipos de tareas están orientados los script? ¿Los scripts deben compilarse? ¿Por qué?

El Shell Scripting es la práctica de escribir scripts o programas pequeños utilizando el lenguaje de comandos de un shell en sistemas operativos tipo Unix, como Linux. Estos scripts son secuencias de comandos escritas en un lenguaje de shell (como Bash, sh, zsh, etc.) que se utilizan para automatizar tareas, administrar el sistema, procesar datos y realizar una variedad de acciones en el sistema operativo.

¿Qué es el Shell Scripting?: El Shell Scripting implica escribir secuencias de comandos utilizando el lenguaje de comandos de un shell para automatizar tareas en sistemas Unix-like. Estos scripts pueden ser simples o complejos y se utilizan para realizar una amplia gama de tareas, desde tareas de administración del sistema hasta procesamiento de datos y más.

¿A qué tipos de tareas están orientados los scripts?: Los scripts pueden estar orientados a una variedad de tareas, que incluyen:

Automatización de tareas repetitivas: Por ejemplo, realizar copias de seguridad automáticas, limpiar archivos temporales, descargar y procesar archivos, etc.

Administración del sistema: Como configurar usuarios, gestionar servicios, monitorear recursos del sistema, etc.

Procesamiento de datos: Para transformar, filtrar y analizar datos en archivos de texto o bases de datos.

Interacción con programas y servicios: Para automatizar interacciones con otros programas o servicios, como bases de datos, servidores web, servicios en la nube, etc.

¿Los scripts deben compilarse? ¿Por qué?: No, los scripts de shell generalmente no se compilan en el sentido tradicional. En lugar de eso, los scripts son interpretados directamente por el shell. Esto significa que el código fuente del script es leído y ejecutado línea por línea por el intérprete del shell en tiempo real.

La razón por la que los scripts no se compilan es que el shell scripting se enfoca en la agilidad y la facilidad de uso. Los scripts son fácilmente legibles y editables, lo que facilita la escritura y modificación de código. Además, no se necesita un proceso de compilación antes de ejecutar un script; simplemente se ejecuta directamente. Esto hace que los scripts sean ideales para tareas rápidas y cambios frecuentes en el código.

Sin embargo, es importante señalar que en algunos casos, se pueden utilizar herramientas como shc para compilar scripts de shell en binarios ejecutables para proteger el código fuente o mejorar el rendimiento, pero esto no es común en el uso diario de shell scripting.

2. Investigar la funcionalidad de los comandos echo y read

(a) ¿Como se indican los comentarios dentro de un script?

Comentarios de una sola línea: Se hace con # (#este es un comentario de una linea)

Comentarios de varias lineas: se hace con : (: hola que tal /// no se// enter dsd)

(b) ¿Cómo se declaran y se hace referencia a variables dentro de un script?

Las variables se declaran con =sin espacios( variable=”hola”)

Estas se pueden referenciar en un script con el simbolo $ seguido de llaves que encapsulan la variable (echo “El valor de mi variables es ${variable}.”)

(echo de mostrar el contenido echo $variable)

4. ¿Cómo se acceden a los parámetros enviados al script al momento de su invocación? ¿Qué información contienen las variables $#, $\*, $? Y $HOME dentro de un script?

Al momento de invocar el scrip se ponen al lado parametros para usar, estos dentro del scrip se pueden acceder usar con $#,$\*,y cada uno con $1,$2,$3 etc por orden. $# contiene la cantidad/ valor de parametros que recibe el script, $\* devuelve una cadena con todos los parametros, $? Contiene la salida del ultimo comando ejecutado si es 0 fue exitoso, en otro caso indica error, $HOME contiene la ruta del directorio del usuario actual.

5. ¿Cual es la funcionalidad de comando exit? ¿Qué valores recibe como parámetro y cual es su significado?

El comando exit en un script de shell, ya sea en Bash u otro intérprete de comandos similar, se utiliza para finalizar la ejecución del script y devolver un código de salida al sistema operativo.

La funcionalidad principal del comando exit es:

* Finalizar la ejecución del script: Cuando se ejecuta el comando exit, el script se detiene inmediatamente y el control se devuelve al sistema operativo o al shell que lo llamó.
* Establecer un código de salida: El comando exit puede recibir un valor numérico como parámetro, que se utiliza para indicar el estado de finalización del script. Este valor, conocido como "código de salida" o "código de retorno", se utiliza para comunicar si el script se ejecutó correctamente o si ocurrió algún error. Por convención, un valor de 0 generalmente indica que el script se ejecutó sin errores, y otros valores diferentes de 0 indican que ocurrió algún tipo de error.

Se puede poner exit seguido de un numero para indicar explicitamente el estado de finalizacion, ejemplo exit 0 para informar que fue exitoso, o exit 1 para informar error, con $? Podemos ver este valor

6. expr se usa para obtener una operación entre argumentos, el resultado y poder usarlo en cadenas. Tambien se pueden manipular cadenas de texto. Ejemplo:

resultado=$(expr 5 + 3)

echo $resultado #Esto imprimirá "8"

El comando expr admite diversas operaciones aritméticas y de manipulación de cadenas. A continuación, se presentan algunas de las operaciones que puedes utilizar con expr:

Operaciones Aritméticas:

Suma: +

Resta: -

Multiplicación: \*

División: /

Resto (Módulo): %

Exponenciación: \*\* (dependiendo de la implementación de expr, no está disponible en todas las versiones)

Comparaciones Numéricas:

Igual: =

No igual: !=

Menor que: <

Menor o igual que: <=

Mayor que: >

Mayor o igual que: >=

Operaciones de Manipulación de Cadenas:

Concatenación: : (por ejemplo, expr "Hola, " : '\(.\*\)' para concatenar cadenas)

Obtener una subcadena: index, length, match, substr (para manipular cadenas)

Operadores Lógicos:

& (y)

| (o)

Operadores de Bits:

& (y a nivel de bits)

| (o a nivel de bits)

^ (o exclusivo a nivel de bits)

~ (complemento a nivel de bits)

<< (desplazamiento a la izquierda)

>> (desplazamiento a la derecha)

Funciones:

length (obtener la longitud de una cadena)

index (encontrar la posición de una subcadena en una cadena)

match (encontrar una subcadena coincidente en una cadena)

substr (obtener una subcadena de una cadena)

7. test o [] para evaluar expresiones. Por ejemplo en if, if [-e archivo1.txt]; then....

El comando test, también conocido como [ ], es una utilidad en sistemas Unix y Linux que se utiliza para evaluar expresiones y generar un valor de retorno que representa "verdadero" (exit code 0) o "falso" (exit code 1). Estas expresiones se utilizan comúnmente en scripts de shell para realizar evaluaciones condicionales. Pueden ser usadas para diversas finalidades, incluyendo:

1. Evaluación de archivos:

-e archivo: Comprueba si el archivo existe.

-f archivo: Comprueba si el archivo es un archivo regular (no un directorio u otro tipo de archivo especial).

-d archivo: Comprueba si el archivo es un directorio.

-s archivo: Comprueba si el archivo tiene un tamaño mayor que cero.

-r archivo: Comprueba si el archivo es legible por el usuario actual.

-w archivo: Comprueba si el archivo es escribible por el usuario actual.

-x archivo: Comprueba si el archivo es ejecutable por el usuario actual.

2. Evaluación de cadenas de caracteres:

cadena1 = cadena2: Comprueba si cadena1 es igual a cadena2.

cadena1 != cadena2: Comprueba si cadena1 no es igual a cadena2.

-z cadena: Comprueba si cadena está vacía (longitud cero).

-n cadena: Comprueba si cadena no está vacía.

3. Evaluaciones numéricas:

num1 -eq num2: Comprueba si num1 es igual a num2.

num1 -ne num2: Comprueba si num1 no es igual a num2.

num1 -lt num2: Comprueba si num1 es menor que num2.

num1 -le num2: Comprueba si num1 es menor o igual que num2.

num1 -gt num2: Comprueba si num1 es mayor que num2.

num1 -ge num2: Comprueba si num1 es mayor o igual que num2.

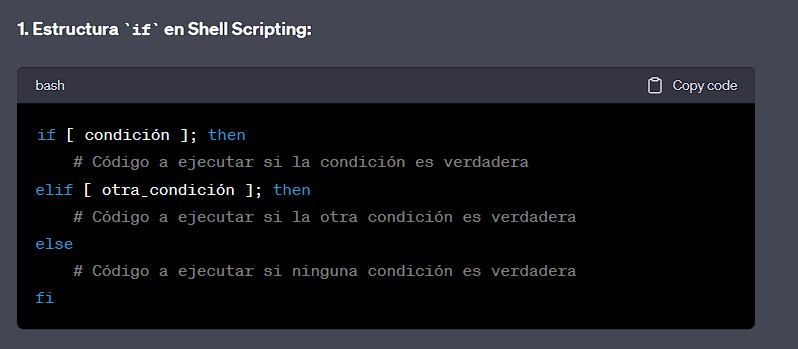
Otros operadores:

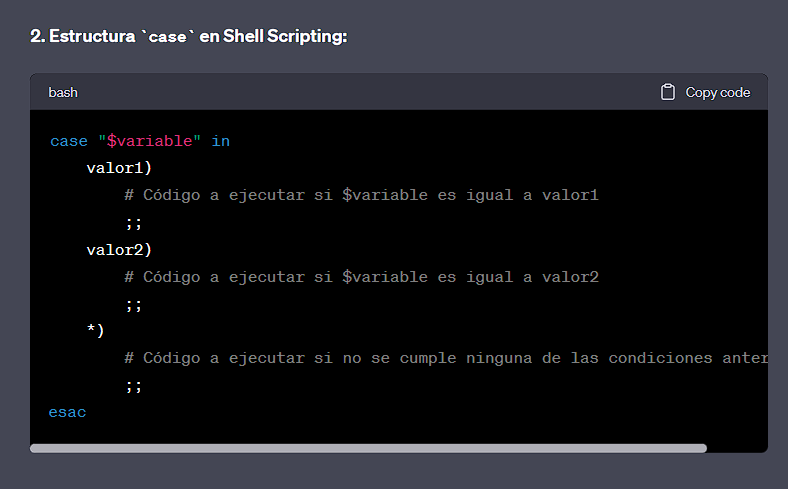
-o: Operador OR lógico.

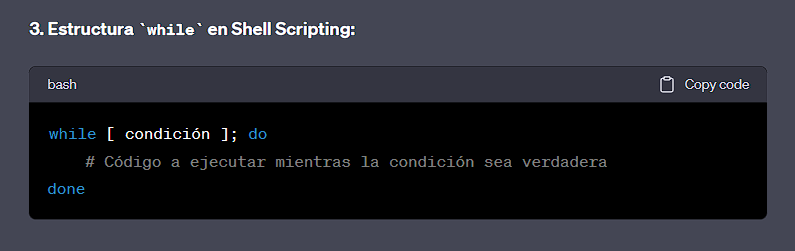
-a: Operador AND lógico.

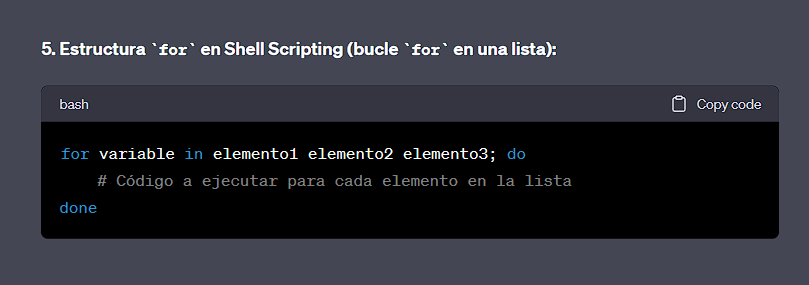
!: Operador NOT lógico.

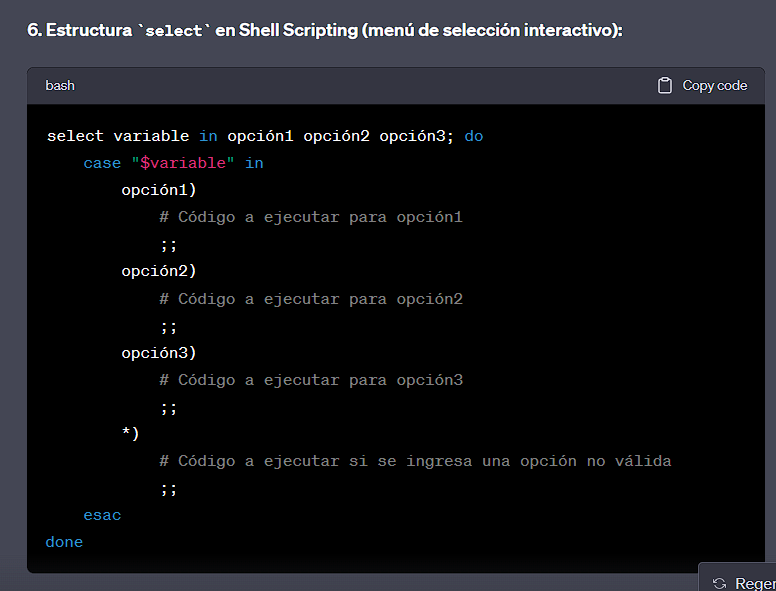
8. Estructuras de control (if, case, while, for, select)











9. ¿Qué acciones realizan las sentencias break y continue dentro de un bucle? ¿Qué parámetros reciben?

Las sentencias break y continue son utilizadas en bucles (como for, while, o until) en lenguajes de programación, incluyendo el scripting de shell, para controlar el flujo de ejecución dentro del bucle. A continuación, se explican las acciones que realizan y cómo se utilizan:

1. break: (salir de inmediato del bucle)

Acción: La sentencia break se utiliza para salir inmediatamente de un bucle antes de que se complete su iteración normal. Cuando se encuentra una instrucción break, el bucle se interrumpe y la ejecución continúa fuera del bucle.

Parámetros: La sentencia break no recibe parámetros adicionales.

2. continue: (saltarse la interacción actual e ir a la siguiente)

Acción: La sentencia continue se utiliza para saltar la iteración actual del bucle y pasar a la siguiente iteración. Cuando se encuentra una instrucción continue, el código restante dentro del bucle para la iteración actual se omite, y se inicia la siguiente iteración.

Parámetros: La sentencia continue no recibe parámetros adicionales.

10. ¿Qué tipo de variables existen? ¿Es shell script fuertemente tipado? ¿Se pueden definir arreglos? ¿Cómo?

BAsh soporta variables String y Arrays, este es fuetemente tipado Case sesitive.

Si se pueden definir arreglos, de la forma

* arreglo\_a=() #este av a estar vacio
* Arreglo\_b=(1 2 3 4 5 6) #inicializado

Se debe poner los vcalores del arreglo separados por un espacio

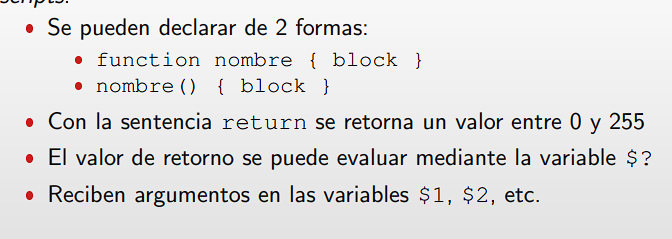
Se puede ver con

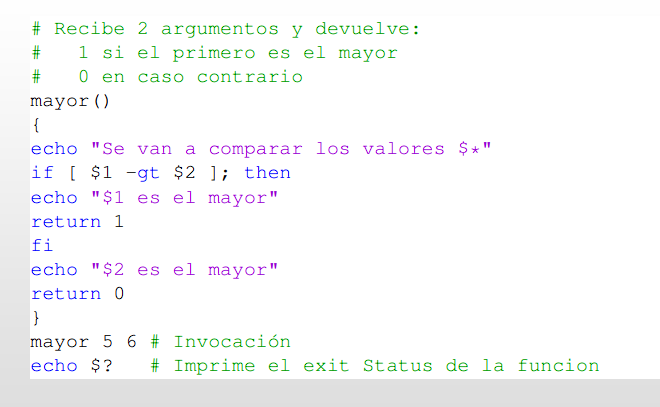
echo ${arreglo\_a[2]} # ver contenido de la pos 2. ESTAS EMPIEZAN EN 0

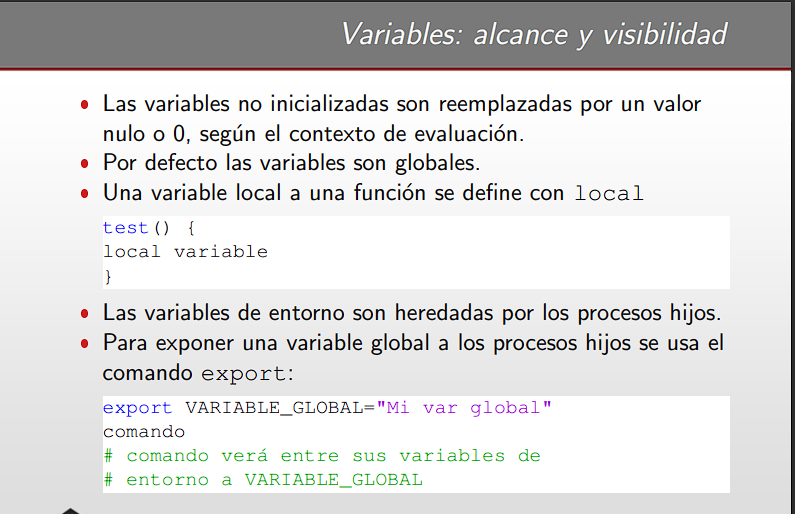
echo ${arreglo\_a[\*]} # Ver todos los elementos

Unset areglo\_b[2] eliminar en esa posicion. El espacio sigue, pero queda vacia esa posicion

11. ¿Pueden definirse funciones dentro de un script? ¿Cómo? ¿Cómo se maneja el pasaje de parámetros de una función a la otra?







17. TAR

