Guía de Bash Script

Bash es una herramienta popular de scripts disponible en **Unix**. Es la abreviación de Bourne Again Shell. Es una herramienta poderosa para todo usuario de Linux o administrador de sistemas. ¡Entonces empecemos a aprender sobre Bash script en Linux!

Unix tiene 2 categorías principales de shells.

- Shell tipo Bourne
- C shell

Bourne shell se clasifica además como:

- Korn shell (ksh)
- Bourne shell (sh)
- POSIX shell (sh)
- Bourne Again shell (bash)

C shell también se clasifica además como:

- C shell (csh)
- TENEX (TOPS) C shell (tcsh)

Los Bash scripts son un componente muy potente y útil para el desarrollo. Puede reducir tareas repetitivas y cortas a una sola línea. Se pueden consolidar muchos comandos largos en un solo código ejecutable.

Bash está disponible en casi todas las versiones de Linux y no requiere de instalación adicional. La lista de shells disponibles se puede verificar escribiendo el siguiente comando:

```
cat /etc/shells
```

El resultado será algo similar a esto:

```
/bin/bash
/bin/sh
/bin/tcsh
/bin/csh
```

¿Por qué usar los scripts Bash?

Las funciones Bash pueden:

- Eliminar tareas repetitivas
- Ahorrar tiempo
- Proporciona una secuencia de actividades bien estructurada, modular y formateada
- Con scripts, podemos proporcionar valores dinámicos a comandos usando argumentos de línea de comando
- Puede simplificar comandos complejos en una sola unidad en ejecución
- Una vez creado, se puede ejecutar cualquier cantidad de veces por cualquier persona. Construye una vez y ejecuta muchas veces.
- Los flujos lógicos se pueden construir utilizando funciones bash
- Las funciones Bash se pueden ejecutar al inicio del servidor o agregando un cron job programado
- Los comandos pueden ser depurados
- Puede tener comandos de shell interactivos

Bash es definitivamente una gran herramienta para facilitar tu trabajo y mejorar tus proyectos. Los usos potenciales son ilimitados, así que hoy solo te enseñaremos los conceptos básicos. ¡Prepárate para escribir tu primer script de bash en Linux!

Comenzando con Bash

```
man bash
```

A continuación, tendremos que crear un archivo **.sh**.

Para esto usaremos **VIM o nano Editor**. Para crear un archivo, usa un comando como este:

```
vim sampleFunction.sh
```

Ahora seremos llevados al archivo **.sh**, donde podemos editarlo.
Esto generará un resultado con los comandos Bash y su uso. Todo script de bash en Linux debe comenzar con la siguiente línea:

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

El siguiente comando muestra la ruta del script bash.

```
which bash
```

Esto mostrará el siguiente resultado:

```
/bin/bash
```

La sintaxis común de bash es:

```
function functionName {
first command
second command
}
```

Esto también se puede escribir como:

```
functionName (){
first command
second command
}
```

En una sola línea, esto se puede escribir así:

```
functionName() { first command; second command; }
```

Un ejemplo de dicha función se muestra a continuación, donde primero creamos un directorio y luego cambiamos la ruta para que apunte al nuevo directorio:

```
sampleFunction () {
mkdir -p $1
cd $1
}
```

$1 representa el argumento de entrada de la línea de comando. Bash puede crear entradas dinámicas dentro del comando. Para verificar esta función, puedes ejecutar:

```
sampleFunction myDir
```

Aquí **myDir** es un nombre de directorio válido. Si revisas el directorio de trabajo actual utilizando el comando **pwd**, puedes ver que actualmente estás dentro del **myDir** recién creado.

Igualmente, cualquier comando de uso común se puede agregar como una función bash.

Recuerda que cuando hayas terminado de usar el editor VIM para editar el archivo **.sh**, puedes guardar y salir presionando **ESC** para ingresar al modo de comando, y luego escribir **:wq** para guardar y salir.

Funciones básicas de Bash

Uno de los ejemplos básicos de la función bash en Linux se muestra a continuación:

```
#!/bin/bash
testfunction(){
   echo "My first function"
}
testfunction
```

Si guardas este script en **testFunction.sh** y lo ejecutas como **./testFunction.sh**, podrás ver el resultado como:

```
My first function
```

Echo imprime la salida en la consola. Si intercambias la posición de la definición de la función con la llamada, esto generará un error. El siguiente fragmento dará un error.

```
#!/bin/bash
testfunction
testfunction(){
   echo "My first function"
}
```

Entonces, primero tendrás que definir la función y luego invocarla.
Las funciones bash pueden aceptar cualquier número de parámetros. El siguiente ejemplo acepta dos parámetros:

```
#!/bin/bash
testfunction(){
   echo $1
   echo $2
}
testfunction "Hello" "World"
```

También puedes usar entradas interactivas y realizar funciones bash. Uno de estos ejemplos es el que se muestra a continuación:

```
#!/bin/bash
addition(){
   sum=$(($1+$2))
   return $sum
}
read -p "Enter a number: " int1
read -p "Enter a number: " int2
add $int1 $int2
echo "The result is : " $?
```

En el ejemplo anterior, el valor de addition se asigna en una suma variable, y esto es lo que entrega la función. La entrada interactiva se toma usando **read** para ambos números. Finalmente, el resultado se imprime con **$?** que almacena el valor $sum generado por la función.
Las funciones bash siempre devuelven un único valor.

Puedes dejar comentarios dentro del archivo agregando el símbolo **#** para dejar notas útiles.

Los scripts bash soportan:

- Bucle *while*
- Bucle *for*
- Declaración *if*
- Elemento lógico *and*
- Elemento lógico *or*
- Declaración *Else If*
- Declaración *case*

A continuación se muestra un breve ejemplo del bucle While.

```
#!/bin/bash
isvalid=true
count=1
while [ $isvalid ]
do
echo $count
if [ $count -eq 5 ];
then
break
fi
((count++))
done
```

El ejemplo anterior usa las declaraciones *while* y *if*. Esto ejecuta el bucle *while* 5 veces antes de salir después de verificar la declaración condicional *if*.

El resultado de esto será:

```
1
2
3
4
5
```

El bucle *for* se puede usar para incrementar o disminuir los contadores. Un ejemplo del bucle *for* es el que se muestra a continuación:

```
#!/bin/bash
for (( count=10; count>0; count-- ))
do
echo -n "$count "
done
```

La salida de este bucle será:

```
10 9 8 7 6 5 4 3 2 1
```

En Bash **&&** representa el elemento lógico AND, mientras que **||** representa a OR.

Con las declaraciones *If*, también podemos definir *Else if*. Veamos un ejemplo:

```
#!/bin/bash
echo "Enter a valid number"
read n
if [ $n -eq 101 ];
then
echo "This is first number"
elif [ $n -eq 510 ];
then
echo " This is second number "
elif [ $n -eq 999 ];
then
echo " This is third number "
else
echo "No numbers over here"
fi
```

El mismo ejemplo anterior también se puede escribir utilizando la declaración *case* como se muestra a continuación:

```
#!/bin/bash
echo " Enter a valid number"
read n
case $n in
101)
Echo " This is the first number " ;;
510)
echo " This is the second number " ;;
999)
echo " This is the third number " ;;
*)
echo " No numbers over here " ;;
esac
```

En las declaraciones *case* **;;** representa una ruptura de case.