THE BRIDGE

CYBER SECURITY

Programación Bash

¿Qué veremos?

- **Teoría: Concepto de shell, script y programación shell**
  - Automatización. Problemas de interacción de usuario en scripts
  - Bash y diferentes tipos de shells
  - Programación
    - Cabecera de un script
    - Modo depuración paso a paso
    - Comentarios
    - Detención del programa
    - Variables
    - Parámetros
    - Ejecución con ``. "", ''
    - Operador
    - Lectura de variables por teclado
    - Concepto de condicionales
    - Incrementar variables
    - Concepto de bucles
    - Funciones

Bash y diferentes tipos de shells

- El shell le proporciona una interfaz para el sistema UNIX.
- Un shell es un entorno en el que podemos ejecutar nuestros comandos, programas y scripts de shell.
- Los tipos principales son los siguientes:
  - Shell Bourne (sh). Creado por S. Bourne, es el más utilizado en la actualidad. Su símbolo del sistema es $. Es el shell estándar y el que se monta en casi todos los sistemas UNIX/Linux.
  - C-Shell (csh). Procedente del sistema BSD, proporciona funciones tales como control de trabajos, historial de órdenes, etc. Ofrece importantes características para los programadores que trabajan en lenguaje C. Su símbolo del sistema es %.
  - Korn Shell (ksh). Escrito por David Korn, amplía el shell del sistema añadiendo historial de órdenes, edición en línea de órdenes y características ampliadas de programación.
  - Bourne Again Shell (bash). Fue creado para usarlo en el proyecto GNU. BASH, por lo tanto, es un shell o intérprete de comandos GNU que incorpora la mayoría de distribuciones de Linux. Es compatible con el shell sh. Además, incorpora algunas características útiles de ksh y csh, y otras propias como la edición de línea de comandos, tamaño ilimitado del historial de comandos, control de los trabajos y procesos, funciones y alias, cálculos aritméticos con números enteros, etc. Su símbolo del sistema es nombre_usuario@nombre_equipo.

Cabecera del script

- Cabecera
  - La primera línea del archivo contendrá la referencia al intérprete de shell que será utilizado.
  - En este caso /bin/bash, esta referencia es conocida como “shebang” que le indica al script que lenguaje es utilizado
  - Se utiliza el símbolo #!
  - Se debe coloca la ruta completa del Shell.

- Crear el script prueba1.sh
  - #!/bin/bash
  - echo Hola mundo este es mi primer script
  - Ejecute el script
  - ¿Cuál es la salida?

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

Modo depuración paso a paso

- Depurar en Bash tus scripts, que te permitirán localizar y corregir errores en el caso de que aparezcan. Y es que esto de los errores es algo inevitable.

- Se puede activar en la cabecera
  !#/bin/bash -x

- Se puede activar este modo en la ejecución.
  bash -x prueba1.sh

- O también puedes utilizar la combinación set -x y set +x. De esta forma cuando utilices la primera instrucción comenzará la depuración y cuando utilices la segunda se detendrá.
  !#/bin/bash -x
  set –x
  operacion=$((2+2))
  Set +x
  Echo "Suma 2+2= $operación"

- La siguiente de las opciones que tienes para depurar en Bash es -e.

- Esta opción lo que hace es que cuando en tu script alguna instrucción de como resultado un valor distinto de cero detendrá el script, es decir, directamente se saldrá del script sin continuar.

Comentarios

- Un comentario, un comentario es una línea de código que es ignorada al momento de ejecutar el script, pero que permite documentar la finalidad de este.
- Se usa el Símbolo #.
- Modifique el script prueba1.sh para insertar un comentario que indique
  - **“este es un script de prueba”**

```
# Este programa saluda a la persona que lo utiliza
```

Variable

- Para escribir en una variable lo harás de la siguiente forma Variable=valor
  - Ejemplo
    nom_1=variable_de_prueba

- Mientras que para leer de esa variable antepondrás el simbolo $
  - Ejemplo
    echo $nom_1

- Modifique el script prueba1.sh para que utilice variable

Variables Especiales en Shell

- `$0` representa el nombre del script
- `$1` – `$9` los primeros nueve argumentos que se pasan a un script en Bash
- `$#` el número de argumentos que se pasan a un script
- `$@` todos los argumentos que se han pasado al script
- `$?` la salida del último proceso que se ha ejecutado
- `$$` el ID del proceso del script
- `$USER` el nombre del usuario que ha ejecutado el script
- `$HOSTNAME` se refiere al *hostname* de la máquina en la que se está ejecutando el script
- `$SECONDS` se refiere al tiempo transcurrido desde que se inició el script, contabilizado en segundos.
- `$RANDOM` devuelve un número aleatorio cada vez que se *lee* esta variable.
- `$LINENO` indica el número de líneas que tiene nuestro script.

Parámetros

- Estos son los argumentos pasados al script cuando este es invocado.
- Hay que tener en cuenta que el shell Bourne está limitado a los parámetros del 0 al 9.
  - **./ejemplo1.sh 1 2 3 4 5 6 7 8 9**
- Crear el script prueba2.sh

```
#!/bin/bash
# prueba de parametros
echo "El 1er parámetro es: $1"
echo "El 2do parámetro es: $2"
```

- Ejecutar con los siguientes parámetros:
  - **1 2**
  - **1 dos**
- ¿Cuál es la salida en cada caso y cuál es la diferencia?
- Haga que el script muestre el nombre del usuario y el nombre del directorio del usuario en pantalla usando parámetros

Ejecución con ``, "", ''

- El Bash utiliza el espacio para separar elementos. Así cuando quieres hacer asignaciones que contienen espacios, deberás utilizar las comillas.

- Pero ¿qué comillas utilizar?.
  - Con las **comillas simples** se guarda en la variable literalmente lo que hay entre ellas.
  - Con las **comillas dobles** se interpreta el contenido.
  - Las **comillas Inclinadas(``)** se utilizan para la asignación de un comando a una variable.

- Generar el script **prueba3.sh**

```
variable1=Juan
variable2='Esta es la casa de $variable1'
echo variable2
```

- Cambien las comillas simples por comillas Dobles. ¿Cuál es la salida?

- Incluir en el archivo

```
archivos=`find . -maxdepth 1 -type f | wc -l`
echo "Hay $archivos archivos"
```

- ¿Cuál es la salida?

Operadores

operadores	explicación	Ejemplos
+	adición	`Expr $ a + $ b` resultado es 30.
-	resta	`Expr $ a - $ resultado b` es -10.
*	multiplicación	`Expr $ a \ * $ B` resultados para 200.
/	división	`Expr $ b / $ resultado es a` 2.
%	resto	`Expr $ b $ a`% resultado es 0.
=	asignación	a = b $ variable B se asignará el valor d e a.
==	Iguales. Se utiliza para comparar dos números, mismo rendimiento verdadero.	[$ A == $ b] devuelve falso.
! =	No es igual. Se utiliza para comparar dos números no son los mismos rendimientos cierto.	[$ A! = $ B] devuelve verdadero.

Operadores

operadores	explicación	Ejemplos
-eq	Detectar si dos números son iguales, iguales devuelve true.	[$ A $ eq b] devuelve falso.
-ne	Detectar si dos números son iguales, no iguales como resultado verdadero.	[$ A $ -ne b] devuelve verdadero.
-gt	La izquierda es mayor que el número detectado a la derecha, y si es así, devuelve true.	[$ A $ -gt b] devuelve falso.
-lt	detección de número es inferior a la derecha de la izquierda, y si es así, devuelve true.	[$ A $ -lt b] devuelve verdadero.
-ge	La detección de si el número es igual a la derecha de la parte izquierda de la grande, y si es así, devuelve cierto.	[$ A $ -ge b] devuelve falso.
-le	Si el número de detección de menos de o igual a la derecha a la izquierda, si lo es, se devuelve verdadero.	[$ A $ -le b] devuelve verdadero.

Lectura de variables por teclado

- Se usa la función read. Se puede utilizar solo read y la variable para captura la entrada desde el teclado
  read variable1

  - **Ejemplo**
    echo Introduce tu usuario
    read usuario
    echo Bienvenido , $usuario

- Otra opción es usar el comando read para mostrar el mensaje en pantalla y solicitar la entrada.
  - **Ejemplo**
    echo Introduce tu usuario
    read -p “Introduce tu usuario: ” usuario
    echo Bienvenido , $usuario

Condicionales

```
if [ $1 -lt 5 ]; then
      echo “el primer parámetro es menor que cinco”
else
      echo “el primer parámetro es mayor que cinco”
fi
```

```
case $1 in
         "rm")
             echo "has elegido el comando borrar";;
         "cp")
             echo "has elegido el comando copiar";;
         "ls")
             echo "has elegido el comando listar";;
          *)
             echo "has elegido otro comando";;
esac
```

Bucles

```
for i in {1..1000}
do
    echo "$i.- Yo soy el niño pepino y me siento fresco como una
lechuga"
done
```

```
i=0
while [ $i -lt 1000 ]
do
    echo 'Cemento Fresco, no hay letrero más bello… bueno, sólo Alto
Voltaje.'
    ((i++))
done
```

```
i=10
until [ $i -lt 0 ]
do
    echo $i
    ((i--))
done
```

Incrementar Variables

```
((i+=1))
 let "i+=1"

((i-=1))
 let "i-=1"
```

```
((i++))
 ((++i))
 let "i++"
 let "++i"

((i--))
 ((--i))
 let "i--"
 let "--i"
```

Funciones

```
#!/bin/bash

var1='fuera'
var2='fuera'

funcion_ambito(){
var1='dentro'
local var2='dentro'
var3='dentro'
local var4='dentro'
echo $var1 $var2 $var3 $var4
}

echo $var1 $var2
funcion_ambito
echo $var1 $var2 $var3 $var4
```


THE
BRIDGE