Bash

Overview

La primera línea de un script es el shebang que le indica al sistema cómo ejecutar el script. Los comentarios en shell empiezan con `#`. El shebang también es un comentario.

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

Ejemplo sencillo de 'Hola Mundo' en bash:

```
#!/bin/bash
echo ¡Hola mundo!
```

Cada comando empieza con una nueva línea o después de un punto y coma:

```
echo 'Esta es la primera línea'; echo 'Esta es la segunda línea'
```

Para declarar una variable se hace lo siguiente:

```
VARIABLE="Mi string"
```

Pero no así:

```
VARIABLE = "Mi string"
```

Bash decidirá que `VARIABLE` es un comando a ejecutar, dando un error.

Usando la variable:

```
echo $VARIABLE
echo "$VARIABLE"
echo '$VARIABLE'
```

Cuando la variable es usada, asignada, exportada, etcétera… se escribe su nombre sin `$`. Si se quiere saber el valor de la variables, entonces sí se usa `$`. Note que ' (comilla simple) no expandirá las variables.

Sustitución de strings en variables:

```
echo ${VARIABLE/Mi/Una}
```

Esto sustituirá la primera cadena "Mi" con "Una".

Substring de una variable:

```
echo ${VARIABLE:0:7}
```

Esto va a regresar sólo los primeros 7 caracteres del valor.

Valor por defecto de una variable:

```
echo ${FOO:-"DefaultValueIfFOOIsMissingOrEmpty"}
```

Esto trabaja para null (VARIABLE=), string vacío (VARIABLE=""), } cero (VARIABLE=0) regresa 0

Variables del sistema:

```
echo "El valor de regreso del último programa: $?"
echo "PID del sistema: $$"
echo "Número de argumentos: $#"
echo "Argumentos del script: $@"
echo "Argumentos del script separados en variables: $1 $2..."
```

Para leer un valor del input:

```
echo "¿Cuál es tu nombre?"
read NOMBRE # Note que no necesitamos declarar una variable
echo ¡Hola, $NOMBRE!
```

Tenemos la estructura 'if' usual:

```
# use 'man test' para más información sobre condicionales
if [ $NOMBRE -ne $USER ]
    then
    echo "Tu nombre es tu usuario."
else
    echo "Tu nombre no es tu usuario."
fi
```

También hay ejecuciones condicionadas:

```
echo "Siempre ejecutado" || echo "Sólo ejecutado si el primer comando falla"
echo "Siempre ejecutado" && echo "Sólo ejecutado si el primer comando NO falla"
```

Para usar && y || con condicionales, se necesitan múltiples pares de corchetes:

```
if [ $NOMBRE == "Steve" ] && [ $EDAD -eq 15 ]
then
    echo "Esto correrá si $NOMBRE es Steve Y $EDAD es 15."
fi

if [ $NOMBRE == "Daniya" ] || [ $NOMBRE == "Zach" ]
then
    echo "Esto correrá si $NOMBRE es Daniya O Zach."
fi
```

Las expresiones se denotan con el siguiente formato:

```
echo $(( 10 + 5 ))
```

A diferencia de otros lenguajes de programación, bash es shell , así que funciona en un contexto de directorio actual. Puedes listar archivos y directorios en un directorio actual con el comando 'ls':

```
ls
```

Estos comandos tienen opciones que controlan su ejecución:

```
ls -l # Lista todos los archivos y directorios en líneas distintas.
```

Los resultados del comando anterior pueden ser pasados al siguiente como input. El comando 'grep' filtra el input con los comandos provistos. Así es como podemos listar archivos .txt en el directorio actual:

```
ls -l | grep "\.txt"
```

Puedes también redireccionar el input y el error lanzado de algún comando.

```
python2 hello.py < "input.in"
python2 hello.py > "output.out"
python2 hello.py 2> "error.err"
```

El error lanzado eliminará el contenido del archivo si es que existe, para después escribir el error. Para que se concatene (en lugar de eliminar) use el comando ">>".

Los comandos pueden ser sustituidos dentro de otros comandos usando `$()`. El siguiente ejemplo despliega el número de archivos y directorios en el directorio actual:

```
echo "Hay $(ls | wc -l) elementos aquí."
```

Lo mismo puede ser hecho usando comillas invertidas `` ` `` pero no pueden ser anidadas. El método preferido es $()`:

```
echo "Hay `ls | wc -l` elementos aquí."
```

Bash usa una estructura de casos similar al switch de Java o C++:

```
case "$VARIABLE" in
    # Lista de patrones que las condiciones deben cumplir:
    0) echo "Hay un cero.";;
    1) echo "Hay un uno.";;
    *) echo "No es null.";;
esac
```

Para los ciclos, se usa la estructura 'for'. Cicla para cada argumento dado:

```
# El contenido de $VARIABLE se imprime tres veces.
for VARIABLE in {1..3}
do
    echo "$VARIABLE"
done
```

Ciclos `while`:

```
while [true]
do
    echo "cuerpo del ciclo..."
    break
done
```

También se pueden definir sub-rutinas (funciones):

```
# Definición de función
function miFuncion ()
{
    echo "Los argumentos trabajan igual que argumentos de   script: $@"
    echo "Y: $1 $2..."
    echo "Esto es una función"
    return 0
}

miOtraFuncion ()
{
    echo "¡Otra forma de declarar funciones!"
    return 0
}
```

Para llamar a la función:

```
foo "Mi nombre es:" $NOMBRE
```

Hay muchos comandos útiles que puedes aprender:

```
# imprime las últimas 10 líneas del archivo file.txt
tail -n 10 file.txt
# imprime las primeras 10 líneas del archivo file.txt
head -n 10 file.txt
# ordena las líneas del archivo file.txt
sort file.txt
# identifica u omite las líneas repetidas, con -d las reporta
uniq -d file.txt
# imprime sólo la primera columna antes de cada ',' en el archivo
cut -d ',' -f 1 file.txt
```

Enlaces de interés

- https://www.gnu.org/software/bash/

- https://github.com/awesome-lists/awesome-bash
- https://github.com/alebcay/awesome-shell

Licencia