Arquitectura y Sistemas Operativos

Unidad 2: Introducción a Scripting y Comandos Linux

- Bash scripting en Linux
- Comandos en consola
- etc.

Script

En informática, un ***script***, es un término que se usa para designar a un programa relativamente simple.

Los *scripts* regularmente no se compilan con anticipación a código máquina, sino que son ejecutados por un intérprete que lee el archivo de código fuente al momento; o incluso por una consola interactiva donde el usuario suministra el programa al intérprete paso a paso. Los *scripts* o guiones se pueden usar para prototipar programas, automatizar tareas repetitivas, hacer procesamiento por lotes e interactuar con el sistema operativo y el usuario

Scripting

Para crear un script en bash solo hay que crear un archivo que tenga una extensión .sh (proveniente de shell Script) como por ejemplo: “hola.sh” y agregarle el shebang de bash “#!/bin/bash”

```
#!/bin/bash
# Este es nuestro primer programa
echo Hola Mundo
```

Luego desde la Terminal le daremos permisos de ejecucion con chmod +x ./hola.sh y lo ejecutamos:

```
~$ ./hola.sh.
```

Shebang

Shebang es el nombre que recibe el par de caracteres **#!** que se encuentran al inicio de los programas ejecutables interpretados.

En algunas ocasiones se le denomina también *hash-bang* o *sharpbang*. A continuación de estos caracteres se indica la ruta completa al intérprete de las órdenes contenidas en el mismo (el *SheBang* más habitual suele ser "#!/bin/bash", que es una llamada al intérprete de comandos Bash).

Scripting

Variables

Como cualquier otro lenguaje de programación, necesitamos variables que nos servirán para guardar datos en la memoria del ordenador hasta el momento que los necesitemos.

Podemos pensar en una variable como una caja en la que podemos guardar un elemento (e.g, un número, una cadena de texto, la dirección de un archivo…).

```
nombre_variable=valor_variable
```

Es importante no dejar espacios ni antes ni después del `=`.

Para recuperar el valor de dicha variable sólo hay que anteponer el símbolo de dolar `$` antes del nombre de la variable:

```
$nombre_variable
```

Scripting

Variables

A lo largo de un script podemos asignarle diferentes valores a una misma variable:

```
#!/bin/bash

to_print='Hola mundo'

echo $to_print

to_print=5.5

echo $to_print
```

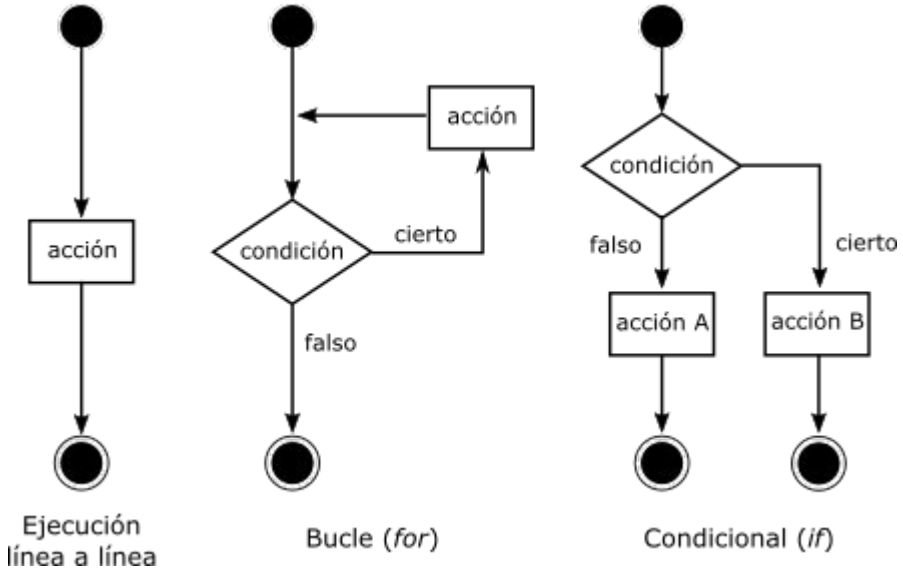
Las variables pueden tomar prácticamente cualquier nombre, sin embargo, existen algunas restricciones:

- Sólo puede contener caracteres alfanuméricos y guiones bajos
- El primer carácter debe ser una letra del alfabeto o “_” (este último caso se suele reservar para casos especiales).
- No pueden contener espacios.
- Las mayúsculas y las minúsculas importan, “a” es distinto de “A”.
- Algunos nombres som usado como variables de entorno y no los debemos utilizar para evitar sobrescribirlas (e.g.,PATH).

Variables

- De manera general, y para evitar problemas con las variables de entorno que siempre están escritas en mayúscula, deberemos escribir el nombre de las variables en minúscula.
- Además, aunque esto no es una regla que deba obedecer obligatoriamente, es conveniente que demos a las variables nombres que más tarde podamos recordar. Si abrimos un script tres meses después de haberlo escrito y nos encontramos con la expresión “m=3.5” nos será difícil entender que hace el programa. Habría sido mucho más claro nombrar la variable como “media=3.5”.

Control del Flujo de Programa



Scripting

Condicionales (if)

La sintaxis básica de un condicional es la siguiente

```
if [[ CONDICIÓN ]];
then
  COMANDO 1 si se cumple la condición
fi
```

También se puede especificar qué hacer si la condición no se cumple:

```
if [[ CONDICIÓN ]];
then
  COMANDO 1 si se cumple la condición
else
  COMANDO 2 si no se cumple la condición
fi
```

Scripting

Condicionales (if)

Incluso se pueden añadir más condiciones concatenando más *if*:

```
if [[ CONDICIÓN 1 ]];
then
  COMANDO 1 si se cumple la condición 1
elif [[ CONDICIÓN 2 ]];
then
  COMANDO 2 si se cumple la condición 2
else
  COMANDO 3 si no se cumple la condición 2
fi
```

Condicionales (if)

Ejemplo

```
#!/bin/bash
      num1=$1   # la variable toma el primer valor que le pasamos al script
      num2=$2   # la variable toma el segundo valor que le pasamos al script
      if [[ $num1 -gt $num2 ]];
      then
        echo $num1 es mayor que $num2
      else
        echo $num2 es mayor que $num1
      fi
```

Al comparar números podemos realizar las siguientes operaciones:

operador	significado
-lt	menor que (<)
-gt	mayor que (>)
-le	menor o igual que (<=)
-ge	mayor o igual que (>=)
-eq	igual (==)
-ne	no igual (!=)

Scripting

Condicionales (if)

Al comparar cadenas de texto podemos realizar las siguientes operaciones:

operador	significado
=	igual, las dos cadenas de texto son exactamente idénticas
!=	no igual, las cadenas de texto no son exactamente idénticas
<	es menor que (en orden alfabético ASCII)
>	es mayor que (en orden alfabético ASCII)
-n	la cadena no está vacía
-z	la cadena está vacía

Scripting

Condicionales (if)

```
#!/bin/bash

    string1='reo'

    string2='teo'

    if [[ $string1 > $string2 ]];

    then

      echo Eso es verdad

    else

      echo Eso es mentira

    fi
```

Condicionales (if)

Al comparar archivos podemos realizar las siguientes operaciones:

operador	Devuelve *true* si
-e name	*name* existe
-f name	*name* es un archivo normal (no es un directorio)
-s name	*name* NO tiene tamaño cero
-d name	*name* es un directorio
-r name	*name* tiene permiso de lectura para el user que corre el script
-w name	*name* tiene permiso de escritura para el user que corre el script
-x name	*name* tiene permiso de ejecución para el user que corre el script

Scripting

Condicionales (if)

Por ejemplo, podemos hacer un script que nos informe sobre el contenido de un directorio:

```
#!/bin/bash
for file in $(ls);
do
  if [[ -d $file ]];
  then
    echo directorio: $file
  else
    if [[ -x $file ]];
    then
      echo archivo ejecutable: $file
    else
      echo archivo no ejecutable: $file
    fi
  fi
done
```

Scripting

Bucles (for)

La sintaxis general de los bucles es la siguiente:

```
for VARIABLE in LISTA_VALORES;
do
      COMANDO 1
      COMANDO 2
      ...
      COMANDO N
done
```

Bucles (for)

Donde la lista de valores puede ser un rango númerico:

```
for VARIABLE in 1 2 3 4 5 6 7 8 9 10;

for VARIABLE in {1..10};
```

una serie de valores:

```
for VARIABLE in file1 file2 file3;
```

o el resutlado de la ejecución de un comando:

```
for VARIABLE in $(ls /bin | grep -E 'c.[aeiou]');
```

Hay que tener en cuenta que si pasamos un listado de valores pero lo ponemos entrecomillado, el ordenador lo enterá como un única línea:

```
for VARIABLE in "file1 file2 file3";
```

Bucles (for)

Ejemplo simple de *for* sería:

```
#!/bin/bash

for numero in {1..20..2};

do

  echo Este es el número: $numero

done
```

Scripting

Bucles (until)

El ciclo *until*, a diferencia del ciclo *while*, permite ejecutar un bloque de instrucciones mientras no se cumpla una condición dada. A diferencia del *while*, verifica que la condición sea falsa, entonces ejecuta el segmento de código contenido entre las palabras *do*y *done*, hasta que la condición se cumpla.

La sintaxis general de los bucles o ciclos until es la siguiente:

```
until[ condición ]
do
//Bloque de instrucciones
done
```

Ejemplo:

```
#!/bin/bash
contador=0
termina=10

until[ $termina -ge $contador ]
do
 echo $contador
 let contador=$contador+1
done
```

Bucles (while)

La sintaxis general de los bucles while es la siguiente:

```
while [ condición ]
do
//Bloque de instrucciones
done
```

Ejemplo:

```
#!/bin/bash
contador=0
termina=10

while [ $termina -ge $contador ]
do
 echo $contador
 let contador=$contador+1
done
```

Scripting

Operaciones aritméticas

Bash también permite la operaciones aritméticas con número enteros:

- + - : suma, resta

```
~$ $num=10
~$ echo $((num + 2))
```

- ** : potencia

```
~$ echo $((num ** 2))
```

- * / % : multiplicación, división, resto (módulo)

```
~$ echo $((num * 2))
~$ echo $((num / 2))
~$ echo $((num % 2))
```

Scripting

Operaciones aritméticas

- VAR++ VAR– : post-incrementa, post-decrementa

```
~$ echo $((num++))
~$ echo $num
```

- ++VAR –VAR : pre-incrementa, pre-decrementa

```
~$ echo $((++num))
~$ echo $num
```

Manipulación de cadenas de texto (strings)

Extraer subcadena

Mediante `${cadena:posicion:longitud}` podemos extraer una subcadena de otra cadena. Si omitimos :longitud, entonces extraerá todos los caracteres hasta el final de cadena.

Por ejemplo en la cadena `string=abcABC123ABCabc`:

- `echo ${string:0}`: abcABC123ABCabc
- `echo ${string:0:1}`: a (primer caracter)
- `echo ${string:7}` : 23ABCabc
- `echo ${string:7:3}`: 23A (3 caracteres desde posición 7)
- `echo ${string:7:-3}`: 23ABCabc (desde posición 7 hasta el final)
- `echo ${string: -4}`: Cabc (atención al espacio antes del menos)
- `echo ${string: -4:2}`: Ca (atención al espacio antes del menos)

Manipulación de cadenas de texto (strings)

Borrar subcadena

Hay diferentes formas de borrar subcadenas de una cadena:

- `${cadena#subcadena}`: borra la coincidencia más corta de subcadena desde el principio de cadena
- `${cadena##subcadena}`: borra la coincidencia más larga de subcadena desde el principio de cadena

Por ejemplo, en la cadena `string=abcABC123ABCabc`:

- `echo ${string#a*C}`: 123ABCabc
- `echo ${string##a*C}`: abc

Manipulación de cadenas de texto (strings)

Reemplazar subcadena

También existen diferentes formas de reemplazar subcadenas de una cadena:

- `${cadena/buscar/reemplazar}`: Sustituye la primera coincidencia de *buscar* con *reemplazar*
- `${cadena//buscar/reemplazar}`: Sustituye todas las coincidencias de *buscar* con *reemplazar*

Por ejemplo, en la cadena `string=abcABC123ABCabc`:

- `echo ${string/abc/xyz}`: xyzABC123ABCabc.
- `echo ${string//abc/xyz}`: xyzABC123ABCxyz.

Scripting

El comando read interrumpe la ejecución del shell hasta que el usuario introduzca una cadena de caracteres (aunque sea vacía) en su entrada estándar.

Las palabras que componen la cadena de caracteres escrita por el usuario se asignan a las variables cuyos nombres se pasan como argumentos al comando read

```
$ read a b c

 1 2 3

$ echo $a ; echo $b ; echo $c

1

2

3
```