Shell Scripting

Estructura básica del shell scripts.

Como ya hemos mencionado, un script no es más que un archivo simple de texto que contiene un conjunto de órdenes para realizar una determinada tarea. Para ayudar a la identificación del contenido a partir del nombre de archivo, es recomendable que los shell scripts se almacenen con la extensión **sh.** Por ejemplo, el siguiente fichero de texto, almacenado como *script_ejemplo.sh*, sería un shell script, que además de comandos, contiene otros elementos para mejorar la funcionalidad del mismo.

script_ejemplo.sh

```
#!/bin/bash
# Primer shell script
echo Hola a todos
```

La primera línea de nuestro script debe comenzar con el símbolo **#!** conocido como **shebang,** y que simplemente indica al sistema qué tipo de intérprete utilizar para interpretar el script creado. Inmediatamente seguido de este símbolo se adiciona la ruta completa donde se encuentre el shell a emplear, en nuestro caso, se utilizará el Bash Shell y cuya ruta predeterminada es **/bin/bash** (compruébalo con *which bash*).

La segunda línea es un comentario. Todas las líneas que comiencen por el signo almohadilla **(#)** son ignoradas por el interpretador de comandos y sirven para incluir anotaciones legibles que pretenden explicar lo que con código no se puede. La única excepción a esta regla es cuando la primera línea (como hemos visto) empieza con **#!.**

Debemos recalcar que es muy importante al escribir scripts, que estos presenten una estructura ordenada y entendible y por ello, es una buena práctica organizar el código en bloques, usar sangrados, asignar variables y funciones con nombres descriptivos, etc. Es muy recomendable consultar las guías de estilo proporcionadas para cada lenguaje de programación. De hecho, una forma de mejorar su legibilidad es utilizar los comentarios. Esto ahorrará mucho tiempo y esfuerzo al mirar su código en un futuro o al compartirlo entre otros usuarios o programadores para comprender dicho código y la finalidad o propósito del mismo. Los comentarios se pueden agregar al principio de la línea o en línea con otros códigos, por ejemplo:

```
#!/bin/bash
# Primer shell script
echo Hola a todos # This is a inline Bash comment.
```

Como veis existe un espacio en blanco después de la almohadilla, esto no es necesario, pero mejorará la legibilidad del comentario. Los comentarios se adicionarán línea a línea de formar individual (bash no permite comentarios multilíneas).

Finalmente, en la tercera línea tenemos el comando echo que sirve para imprimir un determinado texto en la pantalla.

Una vez completado el script, lo deberemos guardar y para ello, como hemos comentado, utilizaremos la extensión .sh.

Ejecución del Shell Script

Para ejecutar el archivo de script, el archivo debe tener permiso de ejecución. Para verificar usamos el comando:

```
>ls -l hola.sh
```

Cuando el archivo es creado, los permisos suelen tener la siguiente configuración

```
-rw-r—r—
```

Indicando que el usuario tiene permisos para leer y escribir (rw), los usuarios del grupo tienen permiso de lectura (r) y el resto de los usuarios también tienen permiso de lectura (r). Para poder ejecutar el script debemos otorgar permiso de ejecución mediante el comando chmod.

```
>chmod u+x hola.sh
>ls -l hola.sh

-rwxr-xr-x
```

Para ejecutar el script que hemos creado escribiremos:

```
>./hola.sh
Hola a todos
```

Para ejecutar el script, incluimos ./ delante del nombre del archivo para indicar que dicho comando debe buscarlo en el directorio actual en donde nos encontramos. Si el archivo se encuentra en otro directorio debemos incluir el path o ruta ya sea de forma absoluta o relativa al directorio actual.

Variables

Las variables servirán para guardar datos en memoria. Para asignar el valor a una variable simplemente usamos el signo de igualdad (=). Es importante no dejar espacios ni antes ni después del signo de igualdad.

```
Nombre_variable=valor_variable
```

Para recuperar el valor de una variable sólo hay que anteponer el símbolo de dolar ($) antes del nombre de la variable. A lo largo del un script podemos asignarle diferentes valores a la misma variable.

```
#!/bin/bash
# Primer shell script
mensaje="Hola a todos"
echo $mensaje
mensaje= 10.50
echo $mensaje
```

Nombre de las variables

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 un guión bajo
- No pueden contener espacios

- Las mayúsculas y las minúsculas importan, “a” es distinto de “A”
- Algunos nombres son usados como variables de entorno y no los debemos utilizar para evitar sobrescribirlas, por ejemplo PATH

De manera general, y para evitar problemas con las variables de entorno que siempre están escritas en mayúscula, es conveniente escribir el nombre de las variables en minúscula. Además, es conveniente que usemos nombres de variables que permitan identificar el contenido de las mismas.

Control de flujo

Los scripts se ejecutan línea a línea hasta llegar al final, sin embargo, muchas veces nos interesará modificar ese comportamiento de manera que el programa pueda responder de un modo u otro dependiendo de las cirscunstancias o pueda repetir trozos de código.

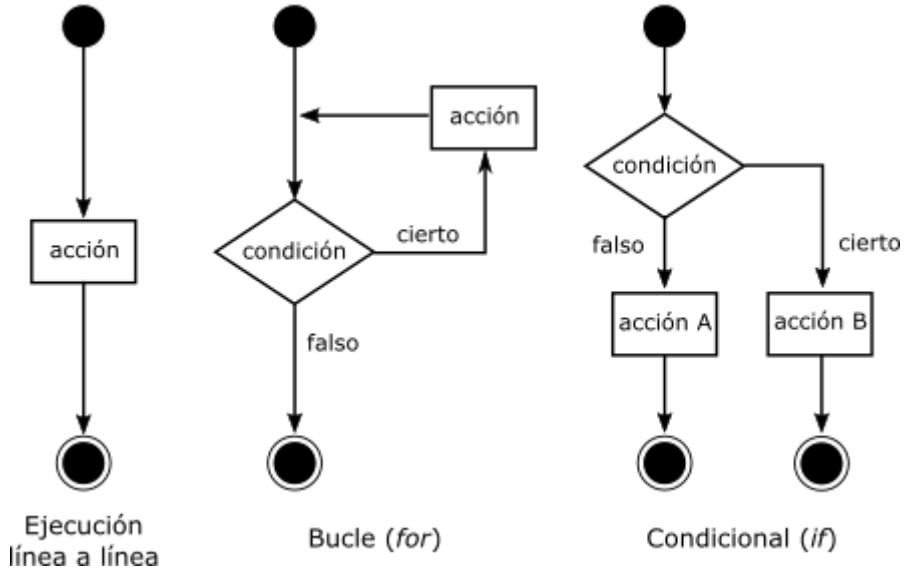

Figura 9.1. https://bioinf.comav.upv.es/courses/unix/scripts_bash.html

Bucles (for)

La sintaxis general de los bucles en un script es la siguiente:

```
for VARIABLE in LISTA_VALORES;
do
      COMANDO_1
      COMANDO_2
      ...
done
```

Donde LISTA_VALORES puede ser un rango númerico:

```
for VARIABLE in 1 2 3 4 5 6;
for VARIABLE in {1..6};
```

una serie de valores:

```
for VARIABLE in archivo1 archivo2 archivo3;
```

o el resultado de la ejecución de un comando:

```
for VARIABLE in $(ls ./ | grep -E 'datos.[abc]');
```

```
for numero in {1..10..2};
do
    echo Iteracion: $numero
done
```

```
Salida
Iteracion: 1
Iteracion: 3
Iteracion: 5
Iteracion: 7
Iteracion: 9
```

Condicionales (if)

La sintaxis básica de un condicional es:

```
if [[ condición ]];
then
    commando_1 si se cumple la condición
fi
```

También podemos especificar qué hacer cuando la condición no se cumpla:

```
if [[ condición ]];
then
    comando_1 si se cumple la condición
else
    commando_2 si no se cumple la condición
fi
```

Se pueden añadir condiciones dentro de condicionales.

```
if [[ condición_1 ]];
then
    comando_1 si se cumple la condición_1
elif [[ condición_2 ]];
then
    commando_2 si se cumple la condición_2
else
    commando_3 si no se cumple la condición_2
fi
```

Cuando en la condición usamos valores numéricos podemos usar las siguientes operaciones:

Operador	Significado
-lt	Menos que (<)
-gt	Mayor que (>)
-le	Menor o igual que (≤)
-ge	Mayor o igual que (≥)
-eq	Igual (=)
-ne	No igual (≠)

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

```
then
    echo $num1 es mayor que $num2
elif [[ $num1 -lt $num2 ]];
then
    echo $num1 es menor que $num2
else
    echo $num1 es igual a $num2
fi
```

Para la ejecución del script debemos pasar los valores de num1 y num2

```
>./comparar.sh 10 5
10 es mayor que 5
>./comparar.sh 5 10
5 es menor que 10
>./comparar.sh 4 4
4 es igual a 4
```

Para comparar cadenas de texto podemos utilizar las siguientes operaciones:

Operador	Significado
=	Las dos cadenas de texto son idénticas
!=	Las dos cadenas de texto no son idénticas
<	Es menor que
>	Es mayor que
-n	La cadena no está vacía
-z	La cadena está vacía

```
#!/bin/bash
cadena1='casa'
cadena2='palo'

if [[ $cadena1 > $cadena2 ]];
then
    echo $cadena1 es mayor que $cadena2
else
```

```
    echo $cadena1 es menor que $cadena2
fi
```

También se pueden usar *wildcards* (*) en las comparaciones.

```
#!/bin/bash
cadena1='casa'
if [[ $cadena1 = "*s*" ]];
then
    echo $cadena1 contiene una s
else
    echo $cadena1 no contiene una s
fi
```

Condicionales con archivos

Operador	Significado
-e name	*name* existe
-f name	*name* es un archivo normal (no es directorio)
-s name	*name* tiene un tamaño mayor que cero
-d name	*name* es un directorio
-r name	*name* tiene permiso de lectura para el usuario que ejecuta el script
-w name	*name* tiene permiso de escritura para el usuario que ejecuta el script
-x name	*name* tiene permiso de ejecución para el usuario que ejecuta el script

El siguiente script nos informa 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
```

Extraer subcadena

Podemos extraer una subcadena de otra cadena de caracteres usando ${cadena:posición:longitud}. El argumento posición indica la ubicación desde donde queremos extraer la subcadena. Hay que tomar en cuenta que la primera posición de una cadena viene identificada con la posición 0. El argumento longitud indica cuántos caracteres se van a extraer a partir de posición. A continuación algunos ejemplos utilizando la cadena string=ABCDEFGHIJKL:

- `echo ${string:0}` : ABCDEFGHIJKL
- `echo ${string:0:1}` : A
- `echo ${string:3}` : DEFGHIJKL
- `echo ${string:3:4}` : DEFG
- `echo ${string:2:-3}` : CDEFGHI

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

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.

Operaciones aritméticas

También podemos realizar operaciones aritméticas con números enteros en nuestros scripts de bash.

Operación	Ejemplo	Resultado a partir de numero=5
Suma	`echo $(( numero + 5 ))`	10
Resta	`echo $(( numero – 5 ))`	0
Potencia	`echo $(( numero ** 2 ))`	25
Multiplicación	`echo $(( numero * 2 ))`	10
División	`echo $(( numero / 2 ))`	2
Resto/Módulo	`echo $(( numero % 2 ))`	1
Post-incremento	`echo $(( numero++ ))`	6
Post-decremento	`echo $(( numero-- ))`	4
Pre-incremento	`echo $(( ++numero ))`	6
Pre-decremento	`echo $(( --numero ))`	4