Sintáxis y Semántica de los lenguajes 2021
K2054
JUEVES Noche

Expresiones regulares y expresiones regulares extendidas en Bash

Nombre y apellido	Leandro Tomás Rouillet
Legajo	16.3443-4
Correo	lrouillet@frba.utn.edu.ar
Usuario Github	lrouillet
Repositorio	github.com/lrouillet/SSL-K2054

Fecha de Presentación	03/06
Fecha de Devolución	
Calificación	
Firma Profesor	

Universidad Tecnológica Nacional
Facultad Regional Buenos Aires

Variables

Una variable, al igual que en los lenguajes de programación, es un lugar donde se guarda un valor o información. De esta manera, se puede reutilizar dicha información en el momento que se desee. Por lo tanto, es necesario poder escribirla y leerla.

Para escribir una variable, se hace de esta manera:

variable=valor

Y para leerla, basta con anteponer un signo $ delante del nombre de la variable en el momento que se la quiera usar. Por ejemplo, si queremos mostrar en pantalla la variable, haremos lo siguiente:

echo $variable

Existen además variables especiales, que ya vienen definidas de antemano, y que refieren al contexto del script. Estas son:

- `$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.

Por ejemplo, si tenemos el siguiente script prueba.sh con el contenido:

#!/bin/bash
echo $1

Y ejecutamos el siguiente comando:

bash prueba.sh variable1

Como resultado obtendremos variable1

Sentencias condicionales

Como adelanta su nombre, una sentencia condicional es aquella que se encuentra sujeta a una *condición*. Es decir, si sucede X, entonces ejecuto la acción Y, caso contrario, ejecuto otra o ninguna acción.

En bash, estas sentencias se escriben como *if then else* y *case*. Como decíamos anteriormente, independientemente de cuál elijamos, en ambos tenemos que resolver una comparación, y así decidir qué camino tomar.

Para comparar en bash se utiliza *test, [ ]* o *[[ ]],* siendo estos últimos una mejora de los corchetes simples.

Algunos comparadores son los siguientes:

CADENAS

[[	[
>	>
<	<
=	=
!=	!=

ENTEROS

[[	[	
-gt	-gt	mayor
-lt	-lt	menor
-ge	-ge	mayor o igual
-le	-le	menor o igual
-eq	-eq	igual
-ne	-ne	distinto

OPERADORES BOOLEANOS

[[	[
&&	-a
||	-o

If, then, else

Estos condicionales se usan de la misma manera en la que suenan: Si (if) pasa esto, entonces (then) hago esto, si no (else), hago aquello. Veámoslo en un ejemplo:

```
#!/bin/bash

if [[ $(($1 % 2)) == 0 ]]
then
   echo Par
else
   echo Impar
fi
```

En este ejemplo, estamos preguntando si el módulo 2 del primer parámetro es 0, es decir, si el parámetro es par. Si lo es, imprimimos Par, caso contrario, Impar.

Case

Esta opción es más útil si una misma variable debe ser comparada muchas veces. Usando if, then y else, esto nos dejaría un código bastante difícil de leer, ya que debemos concatenar sentencias. En cambio, usando una sentencia case, nos quedaría de la siguiente manera:

```
case <expresión> in
   <patrón 1>)
      Comandos
      ;;
   <patrón 2>)
      Comandos
      ;;
   *)
      Comandos
      ;;
esac
```

Donde expresión es la expresión a comparar, y patrón n son los diferentes patrones con los cuales se compara. En caso de que coincida con alguno, se ejecutará lo que está inmediatamente abajo hasta los punto y coma ;;.

Por ejemplo:

```
case $1 in
   naranjas)
      echo "$50 el kilo"
      ;;
   bananas)
      echo "$80 el kilo"
      ;;
   peras)
      echo "$60 el kilo"
      ;;
   manzanas)
      echo "$70 el kilo"
      ;;
   *)
      echo "No vendemos esa fruta actualmente"
      ;;
```

Sentencias cíclicas

El concepto de sentencia cíclica o bucle hace referencia a la realización repetitiva de determinada tarea, bien sea infinitamente o hasta que cierta condición sea cumplida.

While

Supongamos que necesitamos realizar un conteo hasta 50. Sin bucles, esta tarea sería algo complicada:

```
echo 1
echo 2
echo 3
…
echo 50
```

Para ahorrarnos esto es que existen los bucles. Esta tarea se puede resolver con un blucle *while* de la siguiente manera:

```
i=1
while [ $i -le 50 ]
do
   echo “$i”
   ((i++))
done
```

Lo que pasa aquí es que se repetirá la acción debajo de done **mientras** que la condición de while se cumpla. Si la condición hubiese sido [true], este bucle se ejecutaría infinitamente.

Until

La sintáxis es idéntica a la del ciclo while. La diferencia es que ahora el bucle se repetirá **hasta** que la condición se cumpla. Si queremos hacer lo mismo que hicimos arriba, pero con un bucle until, sería de la siguiente manera:

```
i=1
until [ $i -gt 50 ]
do
   echo “$i”
   ((i++))
done
```

For

Este tiene una sintáxis diferente a las que venimos viendo. De vuelta, si queremos contar hasta 50, deberíamos hacer lo siguiente:

```
for i in {1..50}
do
   echo "$i"
done
```

Donde i irá tomando los valores dentro del rango especificado luego de in

Otra manera es la siguiente, muy similar a la que se usa en C:

```
for ((i=1;i<=50;i++))
do
   echo "$i"
done
```

Subprogramas

Subprogramas, o funciones, son porciones de código a las cuales se las identifica, como si fuesen variables, para luego poder ser utilizadas en otras partes de nuestro programa. El hecho de tener funciones nos ayuda a que el código sea mucho más mantenible, ya que hace nuestro código reutilizable y, en caso de necesitar modificarse algo, sólo deberían hacerse esos cambios en la función correspondiente, y no en cada lugar donde hubiese estado si no teníamos esta capa de abstracción.

Para declarar una función, lo hacemos de la siguiente manera:

mi_primera_funcion(){ echo Hola Mundo; }

Y si queremos usar esta función, solo debemos llamarla por su nombre, sin ningún otro agregado.

Tenemos que tener en cuenta que declarar no es lo mismo que llamar o instanciar. Que hayamos declarado una función (como hicimos arriba) no implica que esta se ejecute. Para ejecutarla debemos llamarla. Además, debemos saber que, si queremos usar una función, esta debe estar previamente declarada en nuestro código.

Algo importante en las funciones es el ámbito de sus variables respecto de su contexto. Supongamos que tenemos la siguiente porción de código:

```
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
```

El resultado será:

```
fuera fuera
dentro dentro dentro dentro
dentro fuera dentro
```

Y esto es así porque se indicó que var2 y var4 sean locales a la función mediante el comando *local,* mientras que var1 y var3 no, por lo que lo que se modifique dentro de la función afectará a las variables globales previamente creadas.