Bash Scripting Avanzado: utilizando declare para la definición de tipos

En lo que a Scripting de Shell respecta, GNU Bash es el lenguaje que el 99,9% de nosotros elige. Sin embargo, es el lenguaje que más abandonado tenemos. Escribimos guiones de Shell en GNU Bash sin importarnos siquiera la legibilidad del código y a pesar de ello nos sentimos satisfechos. Pero ¿por qué al menos no avanzar en el conocimiento del lenguaje? Y a eso, apunta esta nueva serie de artículos.

El comando `declare` (`typeset` en *ksh*), es un comando incorporado en en el conjunto de herramientas de GNU Bash, técnicamente denominado *built in*.

Al igual que en lenguajes como C, este *builtin* nos permite **declarar variables definiendo su tipo de datos** y haciendo de éste, un tipo inmutable.

Pero las características de `declare` no concluyen allí. Entre otras cosas, también **permite la emulación de constantes** como lo es `const` para C y PHP, por ejemplo.

Por definición, una variable es interpretada por Bash como *string* (una cadena de texto), por lo cual, en principio, no podríamos ver la necesidad de especificar dicho tipo, puesto que incluso cuando se le asignara un número como valor, sería necesario recurrir a `expr` o a `let` para tratarlo como tal y efectuar, por ejemplo, cálculos aritméticos sobre dicho número.

Sin embargo, en el caso contrario, podríamos ver la necesidad con claridad si lo que se deseara es evitar el uso de `expr` o de `let`.

Definición de constantes en Bash

En bash es posible "declarar" constantes si se utiliza el comando `declare` con el argumento `-r` o `readonly` con la siguiente sintaxis:

```
declare -r NOMBRE_CONSTANTE="valor fijo de la constante"
```

De esta forma, si se quisiera modificar posteriormente el valor de la constante declarara, el *script* arrojaría un error:

```
#!/bin/bash

MAX_TRY=3
(( MAX_TRY++ ))
echo $MAX_TRY
# Salida: 4

declare -r MAX_TRY=3
echo $MAX_TRY
# Salida: 3

(( MAX_TRY++ ))
# Genera un error de solo lectura:
# MAX_TRY: variable de sólo lectura
```

Declaración de tipos: enteros

Podemos además, definir una variable como entero, utilizando **`-i`** como argumento:

```
declare -i variable=5

(( variable++ ))

echo $variable

# Imprimirá 6
```

Si se quisiera modificar el tipo mediante reasignación de una *string*, por ejemplo, **bash evaluaría la cadena de texto como `0`**, tal como se muestra a continuación:

```
declare -i variable=5

variable='foo'

echo $variable
# Imprimirá 0

variable=bar

echo $variable
# También imprimirá 0
```

> *Bash evalúa una cadena de texto (string) como 0 si la variable ha sido declarada como entero*

Si por el contrario, se intentara modificar el valor de la variable hacia un tipo flotante, se produciría un error ya

que no podría ser evaluado como entero:

```
#!/bin/bash

declare -i variable=1500
variable=14.8

# Salida: error sintáctico: operador aritmético inválido (el elemento de error es ".8")
```

Aritmética directa: prescindir de let y expr

Si una variable ha sido declarada como entero, permitirá la realización de cálculos aritméticos básicos prescindiendo de los comandos let y expr:

```
declare -i anio_nacimiento=1978

declare -i edad=2014-anio_nacimiento

echo $edad
# Imprimirá 36

declare -i espacio_total=1500
declare -i usuarios=10
declare -i espacio_por_usuario=espacio_total/usuarios

echo $espacio_por_usuario
# Imprimirá 150
```

Restricción de ámbito

Cuando una variable es definida mediante el comando `declare`, se restringe su ámbito de aplicación. Esto significa que si la variable es declarada dentro de una función, solo será accesible desde dentro de esa función. Para entenderlo mejor, lo mostraré con dos ejemplos: utilizando el comando `declare` y sin utilizarlo.

```
function set_data() {
    declare -i espacio_total=1000
}


function calcular_vps() {
    set_data
    declare -i cantidad_vps=espacio_total/20
    echo Es posible crear $cantidad_vps VPS sobre este disco
}


calcular_vps
# Salida: Es posible crear 0 VPS sobre este disco
```

Sin embargo, cuando se trabaja de forma tradicional sin recurrir al comando declare, la variable se encuentra

disponible fuera del ámbito de la función que la define:

```
function set_data() {
    espacio_total=1000
}


function calcular_vps() {
    set_data
    declare -i cantidad_vps=espacio_total/20
    echo Es posible crear $cantidad_vps VPS sobre este disco
}


calcular_vps
# Salida: Es posible crear 50 VPS sobre este disco
```

Por favor, notar que si en el último ejemplo NO se hubiese definido la variable cantidad_vps mediante declare, la operación aritmética de división, no podría haberse llevado a cabo sin la utilización de expr

Otros usos

El comando declare también puede utilizarse para definir *arrays* y funciones, aunque sin dudas, no nos otorgará tantos beneficios como utilizado para los casos explicados anteriormente.

```
declare -a usuarios=([0]='juan' [1]='pepe' [2]='ana' [3]='eugenia')

echo ${usuarios[@]}
# Salida: juan pepe ana eugenia
```

Si se quisiera obtener una impresión literal del array, el comando declare sería muy útil:

```
declare | grep usuarios
# Salida: usuarios=([0]="juan" [1]="pepe" [2]="ana" [3]="eugenia")
```

Puede utilizarse también, para imprimir la lista completa de funciones declaradas en el script:

```
function foo() {
    echo 1
}

declare -f  # imprimirá la función foo literalmente
```