Bash Scripting Avanzado: sustitución de parámetros y manipulación de variables

De forma realmente simple y sencilla, es posible manipular variables en bash, para resolver necesidades tan frecuentes como el control de argumentos pasados al guión hasta acciones como renombrar archivos en masa.

En la mayoría de los casos, tan solo se requiere una línea de código y sin embargo, por lo general, nos olvidamos de utilizarla u optamos por complejos algoritmos.

Es notable también, la cantidad de veces que recurrimos a lenguajes interpretados más recientes que bash, creyendo que los mismos cubren ausencias que realmente no son tales. Recurrir a lenguajes como Perl o Python en la creación de guiones de *Shell* es tan frecuente que hasta llegamos a olvidarnos de la grandeza de bash.

También es muy frecuente recurrir al empleo de comandos para resolver necesidades que el lenguaje contempla en su propia sintaxis.

Manipulación de variables: Búsqueda y reemplazo

En bash, es posible efectuar búsquedas y sustituciones de manera rápida y flexible sin recurrir una lista interminable de comandos concatenados. En este aspecto, la necesidad más frecuente con la que nos encontramos, es **sustituir determinadas apariciones en una cadena de texto**. Esto es posible con una simple instrucción:

```
${variable//búsqueda/reemplazo}
```

Por ejemplo:

```
cadena='Hola mundo; Adiós mundo'
buscar='mundo'
reemplazar='¡mundo!'
echo ${cadena//$buscar/$reemplazar}
```

Arrojará el siguiente resultado:

```
Hola ¡mundo!; Adiós ¡mundo!
```

Vale aclarar que el código anterior es equivalen a:

```
cadena='Hola mundo; Adiós mundo'
echo ${cadena//'mundo'/'¡mundo!'}
```

Aunque en este caso en particular, en la búsqueda no serían necesarias las comillas.

Notar que la doble barra diagonal // es la que indica que la sustitución debe hacerse en todas las apariciones

Es posible además, **sustituir solo la primera aparición** de la búsqueda en la cadena, con tan solo omitir una de las barras diagonales del comienzo:

```
cadena='Hola mundo; Adiós mundo'
echo ${cadena/'mundo'/'¡mundo!'}
# Imprime: Hola ¡mundo!; Adiós mundo
```

Los signos #, ## y %, %% en las sustituciones

La almohadilla (#) se utiliza para identificar la búsqueda al comienzo de la cadena mientras que el porcentaje (%), para identificarla al final.

Cuando alguno de estos símbolos está presente, **se eliminará de la cadena** aquella fracción coincidente con **el patrón de búsqueda**:

```
${cadena#patron}
Retorna "cadena" SIN "patron"
```

Una sola almohadilla o porcentaje, encontrará en la cadena la parte más corta coincidente con el patrón de búsqueda mientras que **dos almohadillas o porcentajes, lo harán con la mayor fracción** coincidente. De esta forma, si el patrón a buscar fuese: *mun (el asterisco se leería como *"cualquier cosa"* siendo la expresión completa: *"cualquier cosa hasta la sílaba mun"*), una sola almohadilla encontraría coincidencia en *"Hola mun"*, mientras que dos extenderían la coincidencia hasta *"Hola mundo; Adiós mun"*:

```
cadena='Hola mundo; Adiós mundo'
echo ${cadena#*mun}
do; Adiós mundo

echo ${cadena##*mun}
do
```

Lo mismo sucede con el signo del porcentaje:

```
cadena='Hola mundo; Adiós mundo'
echo ${cadena%mun*}
Hola mundo; Adiós

echo ${cadena%%mun*}
Hola
```

Resumiendo, podríamos decir que:

- El porcentaje busca desde el final de la cadena
- La almohadilla busca desde el principio
- Ambos eliminan de la cadena, el patrón que coincida
- Un solo signo, busca la menor coincidencia mientras que dos, buscan la mayor coincidencia

La parte de la cadena que es suprimida, puede ser sustituida con una sintaxis sumamente simple:

```
sustitución${cadena#patron}
${cadena%patron}sustitución
```

Por ejemplo:

```
cadena='Hola mundo; Adiós mundo'
echo "*******"${cadena#*mun}
*******do; Adiós mundo

cadena='Hola mundo; Adiós mundo'
echo ${cadena%mun*}"*******"
Hola mundo; Adiós *******
```

Este último sistema, nos puede servir para, por ejemplo, **cambiar la extensión de archivos en masa**:

```
for archivo in $HOME/*.gif; do
    echo -n 'Renombrando '
    basename $archivo
    mv $archivo ${archivo%*gif}jpg
done
```

O por qué no, para **crear una copia de seguridad** de los mismos:

```
for archivo in $HOME/*.gif; do
    echo -n 'Creando copia de seguridad para '
    basename $archivo
    cp $archivo ${archivo%*gif}'gif.backup'
done
```

Subcadenas y sustitución

En bash, también es posible **extraer una porción de una cadena** utilizando la siguiente sintaxis:

```
${cadena:posicion}
${cadena:posicion:longitud}
```

Con la primera, obtenemos la porción desde `posicion` hasta el final mientras que con la segunda, la obtenemos desde `posicion` hasta alcanzar la longitud indicada:

```
cadena='Hola mundo; Adiós mundo'
echo ${cadena:5}
mundo; Adiós mundo

echo ${cadena:5:5}
mundo
```

Utilizando el mismo método anterior, también podríamos simular una sustitución:

```
cadena='Hola mundo; Adiós mundo'
echo '*****'${cadena:5}
*****mundo; Adiós mundo

echo '***'${cadena:5:5}'***'
***mundo***
```

Teniendo en cuenta que con **${#cadena}** se obtiene la **longitud total de una cadena**, podemos utilizar: **${#cadena}-cantidad** para obtener una porción calculando una determinada posición desde el final:

```
cadena='Hola mundo; Adiós mundo'
echo ${cadena:${#cadena}-11}
Adiós mundo
```

Control de argumentos

Cuando escribimos guiones cuyos argumentos pueden ser opcionales, a veces establecemos valores por defecto para los mismos. Seguramente, en más de una oportunidad te habrás encontrado haciendo lo siguiente:

```
if [ $1 ]; then
    file_path=$1
else
    file_path=$DEFAULT_PATH
fi
```

Sin embargo, bash dispone de una sintaxis mucho más simple para esto:

```
variable=${argumento:-VALOR_POR_DEFECTO}

Ejemplo:
file_path=${1:-$DEFAULT_PATH}
```

Pero lo anterior, es solo una de las tantas posibilidades que tenemos. A continuación, veremos todas ellas con mayor detalle.

Uso y Asignación de Parámetros por defecto

En bash, se pueden **utilizar valores por defecto** en diferentes casos:

```
${variable-valor a usar si la variable no fue definida}
${variable:-valor a usar si la variable no fue definida o su valor es nulo}

Algunos Ejemplos:

echo ${foo-25}              # foo no está definida. Imprime: 25
foo=                        # se define foo con valor nulo
echo ${foo-30}              # No imprime nada porque foo ya fue definida en la línea anterior
echo ${foo:-55}             # Imprime: 55 ya que foo fue definida pero su valor es nulo
```

Este método es ideal para utilizar en el control de argumentos, cuando éstos, no son pasados al guión:

```
declare -r DEFAULT_PATH=/etc/apache2/sites-available
apache_path=${1:-$DEFAULT_PATH}
```

En el ejemplo anterior, si no se pasa un argumento al guión, la variable `apache_path` toma como valor el de la constante `DEFAULT_PATH`.

Vale aclarar que en todos los casos anteriores, el valor por defecto NO ES ASIGNADO a la variable, solamente es UTILIZADO.

Eso significa que (retomando el ejemplo con la variable `foo`) si posteriormente se quisiera imprimir la variable $foo, arrojaría un valor nulo ya que nunca se le ha asignado uno:

```
echo ${foo-75}  # Imprime: 75
echo $foo       # no imprime nada
```

Pero de ser necesario, también es posible **asignar valores por defecto** utilizando el signo de igualdad en vez del guión medio:

```
${variable=valor de variable si no fue definida}
${variable:=valor de variable si no fue definida o su valor es nulo}


echo ${foo=25}   # foo no está definida. Imprime: 25 asigna a foo el valor 25
echo $foo        # Imprime: 25

foo=             # se define foo con valor nulo
echo ${foo=30}   # No imprime nada porque foo ya fue definida. El valor de foo no cambia
echo ${foo:=43}  # Imprime: 43 ya que foo fue definida pero su valor es nulo.
                 # El valor de foo se establece en 25
```

Resumen de búsquedas y sustituciones en Bash

```
================================================================================
BÚSQUEDA Y REEMPLAZO
================================================================================

${VAR/A/B}        Reemplaza la primera aparición de A en VAR, por B
${VAR//A/B}       Reemplaza todas las apariciones de A por B en VAR


================================================================================
CONTAR CARACTERES
================================================================================

${#VAR}           Retorna la longitud de VAR


================================================================================
EXTRACCIÓN
================================================================================

${VAR#PATRON}     Elimina la coincidencia más corta de PATRON del comienzo de VAR
${VAR##PATRON}    Elimina la coincidencia más larga de PATRON del comienzo de VAR
${VAR%PATRON}     Elimina la coincidencia más corta de PATRON del final de VAR
${VAR%%PATRON}    Elimina la coincidencia más larga de PATRON del final de VAR


================================================================================
SUBCADENAS
================================================================================

${VAR:POS}        Retorna la porción de VAR desde POS hasta el final
${VAR:POS:LEN}    Retorna la porción de VAR desde POS hasta alcanzar la longitud LEN


================================================================================
VALORES POR DEFECTO
================================================================================

${VAR-VALOR}      Usar VALOR si VAR no fue definido
${VAR:-VALOR}     Usar VALOR si VAR no fue definido o su valor es nulo
${VAR=VALOR}      Asignar VALOR a VAR si VAR no fue definido
${VAR:=VALOR}     Asignar VALOR a VAR si VAR no fue definido o su valor es nulo
```