Programación en Shell
Cadenas

Francisco de Asís Conde Rodríguez

El intérprete de órdenes trata todos los datos como cadenas de caracteres, por lo que proporciona un conjunto muy potente de herramientas para analizar y procesar cadenas de caracteres. Es lo que se llama Expansión de parámetros.

Hasta ahora, cada vez que nuestros shell scripts debían escribir algo en la pantalla, usábamos la orden echo, es una orden muy útil, sin embargo hay otras alternativas mejores como printf que permite dar formato a la salida.

En esta sesión nos vamos a dedicar a estudiar qué herramientas pone el interprete de órdenes a nuestra disposición y cómo usarlas en nuestros shell scripts.

Expansión de parámetros

En shell script, los parámetros tienen valores que son cadenas de caracteres. La **expansión de parámetros** es un mecanismo muy potente que permite trabajar con esas cadenas de forma sencilla. A continuación vamos a ver las expansiones de parámetros estándar:

- **`${#nombrepar}`**

Se expande a la longitud de la cadena que es el valor del parámetro cuyo nombre es `nombreparam`. Por ejemplo, si en el parámetro posicional `2` está la cadena `adios` el siguiente fragmento de shell script:

```
var="hola mundo"
echo ${#2} ${2}
echo ${#var} ${var}
```

da como resultado:

```
5 adios
10 hola mundo
```

- **`${nombrepar%patrón}`**

Se elimina del parámetro cuyo nombre se pasa como primer argumento, **la subcadena más pequeña**, empezando por el final, que coincide con el patrón que se pasa como segundo argumento. Por ejemplo, el siguiente fragmento de shell script:

```
var=Toronto
echo ${var%o*}
```

da como resultado

```
Toront
```

ya que la `o` final coincide con el patrón `o*` y es la subcadena mas pequeña del final que lo hace.

Un uso muy habitual de esta expansión de parámetros es encontrar el nombre del directorio en el que está un archivo, dentro de una ruta completa hasta ese archivo. Por ejemplo:

```
fichero=/usr/bin/calendar
echo ${fichero%/*}
```

da como resultado:

```
/usr/bin
```

- **`${nombrepar%%patrón}`**

Se elimina del parámetro cuyo nombre se pasa como primer argumento, **la subcadena más grande**, empezando por el final, que coincide con el patrón que se pasa como segundo argumento. Por ejemplo, el siguiente fragmento de shell script:

```
var=Toronto
echo ${var%%o*}
```

da como resultado

```
T
```

ya que la `o` inicial y el resto de la palabra coincide con el patrón `o*` y es la subcadena mas grande desde el final que lo hace.

- **`${nombrepar#patrón}`**

Se elimina del parámetro cuyo nombre se pasa como primer argumento, **la subcadena más pequeña**, empezando **por el principio**, que coincide con el patrón que se pasa como

Expansión de parámetros (continuación)

segundo argumento. Por ejemplo, el siguiente fragmento de shell script:

```
var=Toronto
echo ${var#*o}
```

da como resultado

```
ronto
```

- **${nombrepar##patrón}**

Se elimina del parámetro cuyo nombre se pasa como primer argumento, **la subcadena más grande**, empezando **por el principio**, que coincide con el patrón que se pasa como segundo argumento. Por ejemplo, el siguiente fragmento de shell script:

```
var=Toronto
echo ${var##*o}
```

da como resultado la cadena vacía.

Un uso muy habitual de esta expansión de parámetros es encontrar el nombre del archivo, dentro de una ruta completa hasta ese archivo. Por ejemplo:

```
fichero=/usr/bin/calendar
echo ${fichero##*/}
```

da como resultado:

```
calendar
```

- **${nombrepar//patrón/cadena}**

Sustituye dentro del parámetro cuyo nombre se pasa en `nombrepar`, todas las instancias del patrón que se da como argumento por la cadena de caracteres que se pasa en `cadena`.

Por ejemplo, el siguiente fragmento de shell script:

```
contrasena=aJds34.$
echo "${contrasena//?/*}"
```

da como resultado:

```
********
```

El símbolo `?` es un patrón que significa: cualquier carácter y la doble barra `//` indica **todas** las ocurrencias del patrón, por tanto se sustituyen todos los caracteres por el asterisco.

Nota: hay que escribir `${contrasena//?/*}` entre comillas dobles para impedir que se realice la expansión del `*` como equivalente al contenido del directorio actual.

- **${nombrepar/patrón/cadena}**

Sustituye dentro del parámetro cuyo nombre se pasa en `nombrepar`, la primera instancias del patrón que se da como argumento por la cadena de caracteres que se pasa en `cadena`. Por ejemplo, el siguiente fragmento de shell script:

```
contrasena=aJds34.$
echo "${contrasena/?/*}"
```

da como resultado:

```
*Jds34.$
```

- **${nombrepar/inicio/tamano}**

Devuelve una subcadena de la cadena que se encuentra en el parámetro que se indica en nombrepar, de tamano caracteres, empezando a contar en el carácter que se pasa en inicio.

Por ejemplo, el siguiente fragmento de shell script:

```
dato="abc123"
echo ${dato:2:3}
```

devuelve:

```
c12
```

Que es la subcadena de tres caracteres que comienza en el carácter 2 (se empieza a contar en 0).

Para quedarse con un sólo carácter de la la cadena basta con poner la posición en la que está y tamaño 1. Por ejemplo:

```
dato="abc123"
echo ${dato:3:1}
```

devuelve:

```
1
```

Que es el carácter que ocupa la posición 3 empezando a contar en 0.

- **${nombrepar^}**

Devuelve la cadena que se encuentra en `nombrepar`, con **el primer carácter** en mayúsculas. Por ejemplo:

```
dato="hola mundo"
echo ${dato^}
```

devuelve la cadena:

```
Hola mundo
```

- **${nombrepar^^}**

Devuelve la cadena que se encuentra en `nombrepar`, con **todos** los caracteres en mayúsculas.

Por ejemplo:

```
dato="hola mundo"
echo ${dato^^}
```

devuelve:

```
HOLA MUNDO
```

- **${nombrepar,}**

Devuelve la cadena que se encuentra en `nombrepar`, con **el primer carácter** en minúsculas. Por ejemplo:

```
dato="HOLA MUNDO"
echo ${dato,}
```

devuelve la cadena:

```
hOLA MUNDO
```

- **${nombrepar,,}**

Devuelve la cadena que se encuentra en `nombrepar`, con **todos** los caracteres en minúsculas.

Por ejemplo:

```
dato="HOLA MUNDO"
echo ${dato,,}
```

devuelve:

```
hola mundo
```

Expansión de parámetros (continuación)

Ejemplo: Escribe un shell script que reciba como argumento una palabra y que la escriba al revés (su palíndromo).

```
#!/bin/bash
# Autor:       Francisco de Asís Conde Rodríguez
# Descripción: Recibe como argumento una palabra y
#              la escribe al revés (su palíndromo)

if [ ${#} -ne 1 ]
then
       echo "Error. Debes escribir un sólo argumento."
else
       i=$(( ${#1} - 1 ))

1 →    while [ ${i} -ge 0 ]
       do
2 →           echo -n ${1:${i}:1}
              i=$(( ${i} - 1 ))
       done
3 →    echo
fi
```

(1) Se recorren todos los caracteres de la cadena que se da en el parámetro posicional nº 1 de atrás hacia adelante.

(2) La opción `-n` del `echo` es para que no se escriba un salto de renglón después de cada carácter.

`${1:${i}:1}` es una expansión de parámetros que toma un carácter (último `1`) del parámetro posicional nº 1 (primer `1`) a partir de la posición cuyo nº se tiene en la variable `i`.

(3) El echo final es para que haya un salto de renglón al final de la palabra.

printf

En shell script se tiene la orden `echo`, que es útil para imprimir información por la pantalla, pero hay ocasiones en que se necesita que esa información esté formateada. Entonces es necesario usar `printf` en su lugar.

Por ejemplo, el siguiente bucle:

```
for i in 10 100 1000
do
  echo ${i}
done
```

Muestra por pantalla:

```
10
100
1000
```

Dado que se trata de una columna de números, quedaría mejor si se alinea a la derecha, para este tipo de situaciones se usa la orden `printf`. Por ejemplo:

```
for i in 10 100 1000
do
    printf "%4d\n" ${i}
done
```

Muestra por pantalla:

```
  10
 100
1000
```

Sintaxis de `printf`

La sintaxis de `printf` es:

```
printf formato argumentos...
```

Donde `argumentos...` es una lista de argumentos separados por espacios y `formato` es una cadena de caracteres que indica con qué formato se imprimen esos argumentos.

En el formato se pueden escribir caracteres normales, secuencias de escape y especificadores de formato.

- Los caracteres normales se escriben tal cual. Permiten aclarar un poco el significado de la información que se imprime por pantalla. Por ejemplo: `"el resultado es:"`
- Las secuencias de escape son un único carácter precedido de una barra invertida `\`. Sirven para escribir caracteres no imprimibles. Los más usados son:

  `\n` Salto de renglón.

  `\t` Tabulador.

  `\\` Permite escribir la barra invertida como un carácter normal usando `printf`.

- Los especificadores de formato son un único carácter precedido del signo `%`. Tiene que haber tantos como argumentos. Cada uno indica cómo se imprime su argumento asociado. Los más usuales son:

%s El argumento correspondiente se imprime como una cadena de caracteres literal.

%d El argumento correspondiente se imprime como un número entero.

%f El argumento correspondiente se imprime como un número real.

printf (continuación)

Especificar el tamaño

Si justo después del signo % se escribe un número, ese número indica el tamaño en caracteres con que se imprime el argumento. Si la cadena que contiene el argumento tiene menos caracteres de los especificados, se imprime alineada a la derecha.

Por ejemplo:

```
printf "%6s\n" hola
```

imprime por pantalla:

```
  hola
```

mientras que:

```
printf "%8s\n" hola
```

imprime por pantalla:

```
    hola
```

Si el número que especifica el tamaño en caracteres se precede de un signo -, entonces la cadena se alinea a la izquierda.

Por ejemplo:

```
printf "%-8s:%s\n" hola mundo
```

imprime por pantalla:

```
hola    :mundo
```

Si se especifica un tamaño decimal para el especificador de formato `f`, la primera cifra corresponde con el total de caracteres que se usan para imprimir el número, mientras que la segunda indica cuantos de ellos son decimales. Por ejemplo:

```
printf "%10.3f\n" 1234,45433
```

imprime por pantalla:

```
  1234,454
```

También se puede usar un tamaño decimal para el especificador de formato de cadenas de caracteres s. En este caso, la segunda cifra indica la anchura máxima de la cadena. Por ejemplo:

```
printf "%10.3s %8.2s\n" hola mun
```

imprime por pantalla:

```
       hol       mu
```

Imprimir con formato en una variable

Si en la orden `printf` se usa la opción `-v` seguida del nombre de una variable, la cadena de caracteres formateada no se imprime por la salida estándar, sino que se guarda en la variable.

Por ejemplo:

```
printf -v res "%10.3f\n" 1234,45433
echo ${res}
```

imprime por la pantalla:

```
1234,454
```

Fíjate que se ignoran los caracteres iniciales.

Ejemplo: Escribe un shell script que reciba como argumento un número menor o igual a 20 y escribe por pantalla una pirámide de asteriscos de tantos pisos como el número que se indica.

```
#!/bin/bash
# Autor:       Francisco
# Descripción: Imprime por pantalla una pirámide de
#              asteriscos del tamaño que se pasa como
#              argumento

(1) if [ ${#} -ne 1 ] || [[ ${1} = *[^0-9]* ]]
then
       echo "Error. Debes dar sólo un argumento numérico."
else
(2)    if [ ${1} -gt 20 ]
       then
              echo "Error. Número mayor que 20"
       else
(3)           cad="********************"
              i=1
(4)           while [ ${i} -le ${1} ]
              do
(5)                  printf "%${1}.${i}s%-${1}.${i}s\n" \
                            "${cad}" "${cad}"
                     i=$(( ${i}+1 ))
              done
       fi
fi

fconde@fconde-VirtualBox:~$ piramideasteriscos 6
     **
    ****
   ******
  ********
 **********
************
```

(1) Se comprueba si el número de argumentos no es 1 o si incluye caracteres no numéricos. Esto último es para asegurar que el argumento es un número.

(2) Se comprueba si el número que se ha dado es menor que 20.

(3) Se define una cadena de 20 asteriscos, el máximo que se van a tener que escribir. Posteriormente sólo se imprimirá una parte de esa cadena, la que corresponda al número de asteriscos que haya especificado el usuario.

(4) El bucle va variando desde i igual a 1 hasta el número de asteriscos que haya especificado el usuario.

(5) Se imprime con formato la cadena formada por los 20 asteriscos.

El especificador de formato incluye indicación sobre la anchura con que se debe imprimir el argumento.

Como puede verse, los valores de anchura pueden tomarse de parámetros. En este caso `${1}` y `${i}`

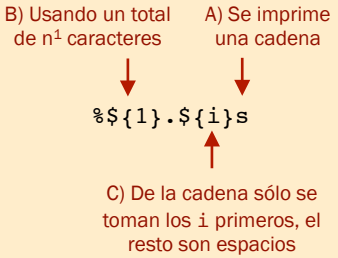
Hay dos especificadores de formato:

`%${1}.${i}s`

y

`%-${1}.${i}s`

cada uno de ellos se aplica a un argumento: `${cad}`



La diferencia entre los dos es que el primero alinea a la derecha y el segundo a la izquierda. De esta forma sale la pirámide.

[1] n es el número que indica el usuario y que se está en el parámetro posicional 1.

Ejercicios

1) Los directorios base de usuario suelen estar ubicados en el directorio `/home`, pero no tiene por qué ser así. Escribe un shell script sin argumentos que analice el archivo `/etc/passwd` y que escriba por pantalla aquellos usuarios, que no sean de sistema, cuyo directorio base no cuelgue de `/home`.

2) Escribe un shell script que renombre todos los archivos ejecutables del directorio `~/bin`, de forma que le añada los caracteres `.sh`, sólo si el archivo no temina ya en .sh. Así si, por ejemplo, en `~/bin` existe un archivo ejecutable llamado `programa`, debería renombrarlo para que se llame `programa.sh`, pero si ya existe un ejecutable llamado `archivo.sh` lo deja tal y como está.

3) La orden `ls -l` lista todos los archivos del directorio de trabajo actual con información completa (incluyendo permisos, tamaño, etc). Escribe un shell script que analice los archivos del directorio `~/bin` e informe si hay archivos cuyos permisos de lectura o escritura estén activados para el grupo y otros usuarios.

4) En el directorio **~/bin** se tienen varios shell scripts. Algunos de ellos tienen nombre terminado en `.sh`, otros no tienen extensión y otros tienen extensión `.exe`. Escribe un shell script que unifique todos los nombres de archivo de forma que todos terminen en `.sh`.

5) En el directorio `~/bin` se tienen varios shell scripts. Escribe un shell script que los renombre usando el siguiente nombre: `nombre.###.sh` donde `nombre` es el nombre que tenga cada escript sin extensión, *###* es un número secuencial comenzando en `000` que se incrementa para cada nuevo shell script que se renombra.