Programación en la shell I

Xaro Benavent Garcia

`xaro.benavent@uv.es`

Herramientas de Programación

Guiones de shell

- Son los programas de shell

```
$ cat primer_programa
```

```
ls
```

```
ls -l
```

- Cómo se ejecutan?
 - Archivo ejecutable: `chmod a+x nombre_guión`
 - `$sh nombre_guión`
 - La primera línea del guión sea:
`#!/bin/sh`

Variables de la shell

- **Variables de entorno**
sirven para personalizar el entorno en el que se ejecuta el `shell`. Algunas de ellas son modificables por el usuario, otras no.
- **Variables definidas por el usuario**
se usan dentro de los guiones de `shell` como lugares de almacenamiento temporal.

Archivos de configuración de bash

- Login shell:
 - Inicialización global:
 - /etc/profile
 - Inicialización personal:
 - \$HOME/.profile
 - \$HOME/.bash_history
- Interactive shell:
 - \$HOME/.bashrc

Variables modificables

Variable de entorno	Propósito de la variable
HOME	Nombre del directorio inicial, válido en el momento en el que se conecta el usuario
PATH	Variable que contiene la ruta de búsqueda del usuario.
PS1	Indicador primario del shell que aparece en la línea de órdenes; normalmente tiene el valor \$
PS2	Indicador secundario del shell
PWD	Nombre del directorio de trabajo actual.
HOSTNAME	Nombre del ordenador actual
USER	Nombre del usuario conectado

\$ set

Variables no modificables

Variable de entorno	Propósito de la variable
\$0	Nombre del programa
\$1-\$9	Valores de los argumentos
\$#	Número total de argumentos de la línea de órdenes
\$@	Valores de todos los argumentos de la línea de órdenes
\$\$	ID del proceso actual
\$?	Estado de salida de la orden más reciente, script o función
#!	PID del proceso de segundo plano más reciente

Uso de variables

- Escritura de una variable:

```
variable=valor
```

- Lectura de una variable:

```
echo $variable
```

```
$ nombre= Xaro  
$ echo $nombre  
$ Xaro  
$ nombre="Xaro Benavent "  
$ echo $nombre
```

Redirecciones

- stdout a fichero:

```
ls -l > salida
```

- stderr a fichero:

```
g++ programa.cpp 2>errores.txt
```

- stderr a stdout:

```
g++ programa.cpp 2>&1
```

- stdout y stderr a fichero:

```
g++ programa.cpp &>/dev/null
```

Exportar el entorno

Sintaxis: `export lista-nombres`

Exporta los nombres y una copia de los valores actuales de 'lista-nombres' a todas las órdenes ejecutadas posteriormente.

```
$ nombre="Xaro"  
$ export nombre  
$ xterm &  
$ echo $nombre  
Xaro  
$
```

```
$export DISPLAY=samba:0.0
```

Sustitución de órdenes

Sintaxis: `orden`

Reemplazar su resultado por orden

```
$ cat demo_fecha
```

```
#!/bin/sh
```

```
fecha=date
```

```
echo "La fecha de hoy es $fecha"
```

```
fecha= `date`
```

```
echo "La fecha de hoy es $fecha"
```

```
La fecha de hoy es date
```

```
La fecha de hoy es mié jul 2 18:51:53 CEST 2003
```

Leer la entrada estándar

Sintaxis: `read lista-variables`

Lee una línea de la entrada estándar

```
$ cat demo_read  
#!/bin/sh  
echo "Escribe una línea"  
read linea  
echo "Línea leída $linea "  
exit 0
```

```
Escribe una línea  
mi linea de prueba  
Línea leída mi linea de prueba
```

Órdenes de flujo de control:if

Sintaxis: `if ... else`

Implementar la bifurcación simple o múltiple

```
if expresión
  then
    [elif expresión
     then
       lista de órdenes
     ...
    [else
      lista-órdenes-else
    ]
  fi
```

Orden test

Sintaxis: `test` expresión

Evaluar expresión y proporcionar un resultado verdadero o falso

Prueba de archivos

Expresión	Resultado
<code>-d arch</code>	True si 'arch' es un directorio
<code>-f arch</code>	True si 'arch' es un archivo ordinario.
<code>-r arch</code>	True si podemos leer 'arch'
<code>-w arch</code>	True si 'arch' es modificable
<code>-x arch</code>	True si 'arch' es ejecutable

Ejemplo de uso de if (1)

```
$ cat demo_if  
#!/bin/sh  
if test $# -eq 0  
then  
    echo "Uso: $0 archivo"  
    exit 1  
fi  
if test $# -gt 1  
then  
    echo "Uso: $0 archivo"  
    exit 1  
fi
```

Ejemplo de uso de if (2)

```
if test -f "$1"
then
    nombreadch="$1"
    set `ls -l $nombreadchivo`
    user="$3"
    tamaño="$5"
    fecha="$6"
    echo "Usuario:$user\t Tamaño:\t Fecha:$fecha\n"
else
    echo "$0: el argumento debe ser un archivo \n"
    exit 1
fi
```

Sentencia for

Sintaxis: for

Ejecutar las órdenes de 'lista-órdenes' tantas veces como palabras haya en 'lista-argumentos'. Sin la parte opcional, 'in lista-argumentos', los argumentos se toman de la línea de órdenes.

```
for variable [in lista-argumentos]
do
  lista de órdenes
done
```

Ejemplo de uso de for

```
$cat demo_for
```

```
for persona in Xaro Juan María
```

```
do
```

```
    echo "$persona"
```

```
done
```

```
$ demo_for
```

```
Xaro
```

```
Juan
```

```
María
```

Sentencia `while`

Sintaxis: `while`

Ejecutar las órdenes especificadas en 'lista-órdenes' mientras el resultado de evaluar 'expresión' siga siendo `true`

```
while expresión
do
  lista de órdenes
done
```

Ejemplo de uso de while

```
$cat demo_while
```

```
#!/bin/sh
```

```
cs=agente007
```

```
echo "Adivine el código secreto"
```

```
echo "Escriba su conjetura"
```

```
while [ "$cs" != "$sc" ]
```

```
do
```

```
    echo "Buen intento"
```

```
    echo "Escriba su conjetura\n"
```

```
    read sc
```

```
done
```

```
echo "Enhorabuena!!"
```

Sentencia `until`

Sintaxis: `until`

Ejecutar las órdenes especificadas en 'lista-órdenes' mientras el resultado de evaluar 'expresión' sea `false`

```
until expresión  
do  
  lista de órdenes  
done
```

Ejemplo de uso de until

```
$cat demo_until
```

```
#!/bin/sh
```

```
cs=agente007
```

```
echo "Adivine el código"
```

```
echo "Escriba su conjetura"
```

```
until [ "$cs" = "$sc" ]
```

```
do
```

```
    echo "Buen intento, pero no es eso"
```

```
    echo "Escriba su conjetura\n"
```

```
    read sc
```

```
done
```

```
echo "Enhorabuena!!"
```

Sentencia case

Sintaxis: case

Implementar la ejecución alternativa multivía como un `if` anidado.

```
case cadena-prueba in
  patron1) lista-órdenes1
          ;;
  patron2) lista-órdenes2
          ;;
  ...
  patronN) lista-órdenesN
          ;;
esac
```

Ejemplo de uso de until (1)

```
$cat demo_case
```

```
#!/bin/sh
```

```
echo "Utilice una de estas opciones"
```

```
echo "d: Ver la fecha y hora actuales"
```

```
echo "l: Ver el listado de archivos del directorio de tra
```

```
echo "w: Ver quien está conectado"
```

```
echo "q: salir"
```

```
read opcion
```

Ejemplo de uso de until (2)

```
case "opcion" in
    d|D) date
        ;;
    l|L) ls
        ;;
    w|W) who
        ;;
    q|Q) exit 0
        ;;
esac
exit 0
```

Procesamiento de datos numéricos

Sintaxis: `expr args`

Evalúa los argumentos de la expresión y envía el resultado a la salida estándar.

Operadores aritméticos

`+` `-` `\*` `/` `%`

Operadores de comparación

`=` `\>` `\>=` `\<` `\<=` `!=`

```
$ var=10
```

```
$ var1= `expr $var1 + 1`
```

```
$ echo $var1
```

```
11
```

Ejemplo de uso de expr

```
#!/bin/sh
if [ $# = 0 ]
then
    echo "Uso: $0 lista_numeros"
    exit 1
fi
suma=0
contador=0
for var in $@
do
    suma=`expr $suma + $var`
    contador=`expr $contador + 1`
done
echo "suma total es $suma"
exit 0
```

Funciones en el shell

Declaración: `function nombre_funcion {
codigo }`

Uso: `nombre_funcion lista_parametros`

- Los argumentos que se pasan a las funciones se tratan de la misma manera que los argumentos de los scripts.
- Función puede devolver un valor `return var` y en el script accedemos a ella con el parámetro posicional `$?`

Ejemplo de uso de funciones

```
function suma ()
{
    sum=0
    for var in $@
        do
            sum=`expr $sum + $var`
        done
    return $sum
}
if [ $# != 2 ]
    then
        echo "Uso: $0 num1 num2"
        exit 1
    fi
suma $@
echo "Total $?"
```