*Área de Ingeniería Telemática*
*Departamento de Ingeniería de Sistemas y Automática*
*Escuela Superior de Ingenieros*
*Universidad de Sevilla*

# *PROGRAMACIÓN DEL SHELL*

## 1. OBJETIVO

El objetivo es introducir la programación del procesador de órdenes (shell) para poder desarrollar programas básicos. Un fichero de texto que se va a procesar por el shell se conoce normalmente por shell script. Es habitual que los shell scripts tengan la extensión .sh.

## 2. DESCRIPCIÓN

No existe un único shell. Los más conocidos son el Bourne shell, el C-shell, el Bash (Bourne again shell) entre otros. Cada uno de ellos tiene características particulares, pero todos contienen básicamente lo mismo: variables, estructuras de control… En un shell script se pueden ejecutar varias órdenes destinadas al shell ejecutando un solo fichero, sin necesidad de volver a reescribir todas las órdenes cada vez que se quiera realizar la misma tarea. El shell que vamos a tratar es el Bash.

### Ejemplo básico

### ⇒ Ejemplo hola mundo:

```
#!/bin/bash
# Este script imprime Hola Mundo por pantalla
echo Hola Mundo
```

hola

si ejecutamos el fichero hola:

```
>hola
hola: permiso denegado
```

esto es debido a que el editor no da permiso de ejecución. Sin embargo, para que el fichero script se pueda interpretar, hay que darle permiso de ejecución.

```
>chmod u+x hola
>hola
Hola Mundo
```

Con este script tan básico ya podemos ver algunas cuestiones:

- La primera línea del script:
  #!/bin/bash
  indica el shell que se invoca para realizar la interpretación de este script. Debe empezar por #! indicando a continuación el shell que se debe invocar, en este caso es el bash que se

encuentra en el directorio /bin. No existe ningún espacio entre los caracteres # y !(esto es importante).

- La segunda línea es un comentario:
  # Este script imprime Hola Mundo por pantalla
  Los comentarios se ponen con el carácter # a principio de línea.
- La tercera línea debe será interpretada por el bash:
  echo Hola Mundo

**Variables**

En un shell script se pueden utilizar variables. Una variable puede contener un número, un carácter o una cadena de caracteres. No necesita ser declarada, por defecto todas las variables son cadenas de caracteres.

**⇒ Ejemplo:**

```
#!/bin/bash
# Este script imprime Hola Mundo por pantalla
CAD="Hola Mundo"
echo $CAD
```

```
holavar
```

En este ejemplo se crea una variable de nombre CAD que contiene la cadena "Hola Mundo". Son necesarias las " " porque hay un carácter de espacio dentro de la cadena. Para hacer referencia a una variable se pone $ delante del nombre de la variable. Una vez dado el permiso de ejecución, el resultado de este script es el mismo que el del anterior.

Normalmente existen variables ya definidas por el shell. Con el siguiente ejemplo se imprimen en pantalla algunas de ellas.

**⇒ Ejemplo de impresión de variables predefinidas:**

```
#!/bin/bash
echo Directorio HOME
echo $HOME
echo Directorios incluidos en el PATH
echo $PATH
echo shell por defecto
echo $SHELL
echo Numero de argumentos pasados al shell
echo $#
echo Estado de la ultima orden ejecutada
echo $?
echo Nombre del fichero ejecutado
echo $0
echo Primer argumento de la linea de ordenes
echo $1
echo Segungo argumento de la linea de ordenes
echo $2
echo Todos los argumentos de la linea de ordenes
echo $*
echo Identificador de proceso actual
echo $$
ls &
echo Identificador del ultimo proceso en background
```

```
echo $!
```

varpredef

si se ejecuta este script, el resultado es el siguiente:

```
>varpredef hola pepe
Directorio HOME
/home/alumnos/pepe
Directorios incluidos en el PATH
/usr/ucb:/bin:/usr/sbin:/usr/bin:/usr/etc:/usr/ccs/bin:/usr/local/gnu/bin:/us
r/local/bin:/usr/games:/usr/local/games:/usr/X11R6/bin:/usr/openwin/bin:/usr/
openwin/bin/xview:/home/alumnos/pepe/bin:.:/usr/local/TeX/bin
Shell por defecto
/bin/tcsh
Numero de argumentos pasados al shell
2
Estado de la ultima orden ejecutada
0
Nombre del fichero ejecutado
varpredef
Primer argumento de la linea de ordenes
hola
Segungo argumento de la linea de ordenes
pepe
Todos los argumentos de la linea de ordenes
hola pepe
Identificador de proceso actual
8303
a.out             ejemplofuncion2~  esexe           varpredef
ejcomillas        ejemplofuncion~   esexe2          varpredef~
ejcomillas~       ejemploif         esexe2~
ejemexport        ejemploif2        esexe~
ejemexport~       ejemploif2~       hola
ejemplocase       ejemploif3        hola.c
ejemplocase2      ejemploif3~       holavar
ejemplocase2~     ejemploif~        holavar~
ejemplocase~      ejemplountil      hola~
ejemplofor        ejemplountil~     idioma~
ejemplofor2       ejemplowhile      listado~
ejemplofor2~      ejemplowhile~     saludo
ejemplofor~       ejforseqNO        saludo~
ejemplofuncion    ejforseqNO~       variable
ejemplofuncion2   ejforseq~         variable1
Identificador del ultimo proceso en background
8304
```

Se pueden utilizar otras formas de sustitución de variables diferentes a la vista anteriormente (con el $ delante del nombre de la variable). Por ejemplo:

${variable}
Se necesita usar las llaves cuando a la variable le va a seguir una letra, dígito o subrayado. Por ejemplo en ${DIR}fichero.tex.
${variable:-cadena}
Si la variable está definida y su valor es no-nulo se toma el valor de la variable. En caso contrario, se toma cadena. Por ejemplo en ${DIR:-/tmp}, si DIR no está definida o es nula, se tomaría /tmp.

${variable:=cadena}
Si la variable no está definida o es nula, se toma cadena y a partir de aquí variable vale cadena. Por ejemplo en ${DIR:=/tmp}, si DIR no está definida o es nula, se toma /tmp y DIR valdrá /tmp a partir de aquí.

A continuación se dan algunos ejemplos simples:

**⇒ Ejemplo:**

```
#!/bin/bash
DIR=/tmp
FICH=$DIR/temporal
echo El fichero es $FICH
echo El fichero .c es ${FICH}.c
```

variable

resultado:

```
>variable
El fichero es /tmp/temporal
El fichero .c es /tmp/temporal.c
```

**⇒ Ejemplo:**

```
#!/bin/bash
echo El fichero es ${FICH:-fichero.txt}
echo El fichero es $FICH
```

variable1

resultado:

```
>variable1
El fichero es fichero.txt
El fichero es
```

**⇒ Ejemplo:**

```
#!/bin/bash
echo El fichero es ${FICH:=fichero.txt}
echo El fichero es $FICH
```

variable2

resultado:

```
>variable2
El fichero es fichero.txt
El fichero es fichero.txt
```

Para declarar variables de tipo entero se utiliza declare –i VAR. En el ejemplo siguiente al no haber sido declarada como tipo entero el resultado es el que se muestra:

**⇒ Ejemplo sin declare:**

```
#!/bin/bash
VAR=1
VAR=VAR+1
echo $VAR
```

variable3

resultado:

```
>variable3
VAR+1
```

**⇒ Ejemplo con declare:**

```
#!/bin/bash
declare –i VAR
VAR=1
VAR=VAR+1
echo $VAR
```

variable4

resultado:

```
>variable4
2
```

**Estructuras de control**

A continuación se verán algunas de las estructuras de control que se utilizan en los scripts.

**Estructura de control: if-then-else**

**⇒ Ejemplo de if:**

```
#!/bin/bash
FILE=hola
if [ –f $FILE ]; then
   echo el fichero $FILE existe
else
   echo fichero no encontrado
fi
```

ejemploif

Este script contiene un ejemplo de la estructura if-then-else e imprime por pantalla `el fichero hola existe` en el caso de que el fichero exista o `fichero no encontrado` en el caso de que

no exista. La condición del if [ -f $FILE ]; es verdad si el fichero FILE existe. Es importante respetar en la condición del if los corchetes, los espacios y el carácter ; final (en general, el intérprete de órdenes es muy estricto en la sintaxis). Otros ejemplos de condiciones podrían ser los siguientes:

```
[ $FILE = "hola" ];       # verdad si FILE es hola
[ -d /usr ];              # verdad si /usr es un directorio
[ -d $DIR ];              # verdad si DIR es un directorio
[ -r $FILE ];             # verdad si FILE existe y podemos leerlo
[ -w $FILE ];             # verdad si FILE existe y podemos escribir en él
[ -x $FILE ];             # verdad si FILE existe y podemos ejecutarlo
[ -r $FILE ];             # verdad si FILE existe y podemos leerlo
[ n1 -eq n2 ];            # verdad si los enteros n1 y n2 son iguales
[ n1 -ne n2 ];            # verdad si los enteros n1 y n2 no son iguales
[ n1 -gt n2 ];            # verdad si n1 es mayor que n2
[ n1 -ge n2 ];            # verdad si n1 es mayor o igual que n2
[ n1 -lt n2 ];            # verdad si n1 es menor que n2
[ n1 -le n2 ];            # verdad si n1 es menor o igual que n2
```

En el siguiente ejemplo se pregunta si el número de argumentos es 0, es este caso se imprime el mensaje Falta el nombre del fichero, en caso contrario, si el primer argumento es un fichero se mira si es ejecutable.

**⇒ Ejemplo de if:**

```
#!/bin/bash
if [ $# -eq 0 ]; then
   echo Falta el nombre del fichero
else
   if [ -f $1 ]; then
     if [ -x $1 ]; then
        echo El fichero $1 es ejecutable
     else
        echo El fichero $1 NO es ejecutable
     fi
   else
      echo El argumento $1 no es un fichero
   fi
fi
```

esexe

si se ejecuta este script, el resultado es el siguiente:

```
>esexe
Falta el nombre del fichero
>esexe .
El argumento . no es un fichero
>esexe esexe
El fichero esexe es ejecutable
```

Se puede utilizar el operador -a (and lógico) de la siguiente manera:

```
#!/bin/bash
if [ $# -eq 0 ]; then
   echo Falta el nombre del fichero
else
```

```
   if [ -f $1 -a -x $1 ]; then
     echo $1 es un fichero ejecutable
   else
      echo $1 no es un fichero ejecutable
   fi
fi
```

El operador lógico or es -o.

**Estructura de control: for**

El siguiente ejemplo muestra por pantalla los nombres de los ficheros del directorio actual que tienen el permiso x. La variable i en cada iteración del bucle for es el nombre de un fichero que se ha obtenido de la ejecución de ls.

**⇒ Ejemplo de for:**

```
#!/bin/bash
Declare -i i
for i in $(ls); do
   if [ -x $i ]; then
      echo $i
   fi
done
```

ejemplofor

El siguiente ejemplo hace lo mismo que el anterior. En este ejemplo se ha sustituido $(ls); por `ls`; . Las comillas simples (hacia la derecha) le indican al shell que ejecute la orden que hay entre comillas.

**⇒ Ejemplo de for:**

```
#!/bin/bash
for i in `ls`; do
   if [ -x $i ]; then
      echo $i
   fi
done
```

ejemplofor2

Otra forma del for muy parecida a la de C es la siguiente:

**⇒ Ejemplo de for:**

```
#!/bin/bash
for (( i=0 ; i<10 ; i++ )) ; do
 {
   echo $i
 }
done
```

ejemplofor3

Con este ejemplo se imprime por pantalla del 0 al 9. Esta forma del bucle for no funciona en versiones antiguas del bash.

**Estructura de control: while**

**⇒ Ejemplo de while:**

```
#!/bin/bash
CONTADOR=0
while [ $CONTADOR -lt 10 ]; do
   echo El contador es $CONTADOR
   let CONTADOR=$CONTADOR+1
done
```

ejemplowhile

resultado:

```
>ejemplowhile
El contador es 0
El contador es 1
El contador es 2
El contador es 3
El contador es 4
El contador es 5
El contador es 6
El contador es 7
El contador es 8
El contador es 9
```

**Estructura de control: until**

Las órdenes dentro del until se ejecutan si la condicion es falsa.

**⇒ Ejemplo de until:**

```
#!/bin/bash
CONTADOR=0
until [ $CONTADOR -ge 10 ]
do
   echo El contador es $CONTADOR
   let CONTADOR=$CONTADOR+1
done
```

ejemplountil

El resultado de este ejemplo es el mismo que el del ejemplowhile.

**Estructura de control: case**

Un ejemplo simple de case es el siguiente:

**⇒ Ejemplo de case:**

```
#!/bin/bash
```

```
for NUM in 0 1 2 3
do
  case $NUM in
      0)
        echo $NUM es igual a cero ;;
      1)
        echo $NUM es igual a uno ;;
      2)
        echo $NUM es igual a dos ;;
      3)
        echo $NUM es igual a tres ;;
  esac
done
```

ejemplocase

En este ejemplo se ejecuta un bucle for en donde la variable NUM va tomando los valores 0, 1, 2 y 3. En cada iteracción del bucle se imprime un mensaje diferente.

resultado:

```
>ejemplocase
0 es igual a cero
1 es igual a uno
2 es igual a dos
3 es igual a tres
```

En el siguiente ejemplo se comprueba si el parámetro del case $1 encaja con los patrones dados. En el caso de tener extensión .c se compila. En el caso por defecto, es decir, si no encaja con los patrones anteriores se da un mensaje.

**⇒ Ejemplo de case:**

```
#!/bin/bash
case $1 in
   *.c)
      echo compilando $1
      gcc $1 ;;
   *)
      echo $1 no es un fichero C ;;
esac
```

ejemplocase2

resultado:

```
>ejemplocase hola.c
compilando hola.c
>ejemplocase a.out
a.out no es un fichero C
```

Existen otras órdenes de control tales como `select, break, continue, exit …`

**Funciones**

En un shell script tambien se pueden utilizar funciones. Un ejemplo básico de función es el siguiente. En este ejemplo se tienen dos funciones: la función suma y la función resta.

**⇒ Ejemplo de función:**

```
#!/bin/bash

let A=100
let B=200

#
# Funcion suma()
# Suma las variables A y B
#
function suma(){
 let C=$A+$B
 echo "Suma: $C"
}

#
# Funcion resta()
# Resta las variables A y B
#
function resta(){
 let C=$A-$B
 echo "Resta: $C"
}

suma
resta
```

ejemplofuncion

resultado:

```
>ejemplofuncion
Suma: 300
Resta: -100
```

En el siguiente ejemplo se muestra como se realiza el paso de parámetros

**⇒ Ejemplo de función con parámetros:**

```
#!/bin/bash

let A=100
let B=200

#
# Funcion suma
# Suma los dos parametros que se le dan
#
function suma () {
 let C=$1+$2
 echo "Suma: $C"
}

#
```

```
# Funcion resta()
# Resta las variables A y B
#
function resta () {
 let C=$1-$2
 echo "Resta: $C"
}

suma $A $B
resta $A $B
```

ejemplofuncion2

resultado:

```
>ejemplofuncion2
Suma: 300
Resta: -100
```

Los parámetros dentro de una función se tratan de la misma forma que los parámetros suministrados al script.

En este último ejemplo de función se muestra cómo se puede devolver un valor desde una función.

**⇒ Ejemplo de función que devuelve un valor:**

```
#!/bin/bash

let A=100
let B=200

#
# Funcion suma
# Suma los dos parametros que se le dan
#
function suma () {
 let C=$1+$2
 return $C
}

#
# Funcion resta()
# Resta las variables A y B
#
function resta () {
 let C=$1-$2
 return $C
}

suma $A $B
echo Suma: $?
resta $A $B
echo Resta: $?
```

Ejemplofuncion3

resultado:

```
>ejemplofuncion3
Suma: 300
Resta: -100
```

### Exportando variables

#### export

Las variables son locales por defecto. Si se pretende ejecutar en un mismo script varios scripts, para que las variables definidas se pasen a estos scripts se deben exportar con la orden export.

En el siguiente ejemplo, se llama a un script llamado saludo que imprime hola o hello dependiendo del idioma definido.

**⇒ Ejemplo de export:**

```
#!/bin/bash
export IDIOMA=$1
saludo
```

ejemexport

```
#!/bin/bash

if [ $IDIOMA = "ESP" ]; then
   echo hola
else
   echo hello
fi
```

saludo

resultado:

```
>ejemexport ESP
hola
>ejemexport ING
hello
```

En caso de no haber exportado la variable IDIOMA en el script ejemexport no se habría podido utilizar en el script saludo.

#### La orden "."

Cada vez que se llama a un script se invoca a un nuevo shell. La orden "." sirve para ejecutar un script dentro de otro script sin abrir un nuevo shell.

Por ejemplo, si incluimos la orden ps en el script ejemexport y en saludo, se puede ver que hay dos procesos que están ejecutando el bash.

**⇒ Ejemplo de export:**

```
#!/bin/bash
export IDIOMA=$1
ps
echo saludo
saludo
```

ejemexport

```
#!/bin/bash

if [ $IDIOMA = "ESP" ]; then
   echo hola
else
   echo hello
fi

ps
```

saludo

resultado:

```
>ejemexport ING
   PID TT       S  TIME COMMAND
  7502 pts/1    R  0:00 -tcsh
  7571 pts/1    R  0:00 /bin/bash ejemexport ING
saludo
hello
   PID TT       S  TIME COMMAND
  7502 pts/1    S  0:00 -tcsh
  7571 pts/1    R  0:00 /bin/bash ejemexport ING
  7573 pts/1    R  0:00 /bin/bash ./saludo
```

Notar que en realidad la ejecución de un script es la ejecución de un shell que lo interpreta.

Si al ejecutar saludo utilizamos "." como en el ejemplo siguiente, se puede observar que no se ha abierto un nuevo shell para ejecutar saludo.

**⇒ Ejemplo de ".":**

```
#!/bin/bash
IDIOMA=$1
ps
echo saludo
. saludo
```

ejemexport

```
#!/bin/bash

if [ $IDIOMA = "ESP" ]; then
   echo hola
else
   echo hello
fi
```

```
ps
```

saludo

resultado:

```
>ejemexport ING
   PID TT       S  TIME COMMAND
  7502 pts/1    R  0:00 -tcsh
  7571 pts/1    R  0:00 /bin/bash ejemexport ING
saludo
hello
   PID TT       S  TIME COMMAND
  7502 pts/1    S  0:00 -tcsh
  7571 pts/1    R  0:00 /bin/bash ejemexport ING
```

En este caso no hace falta utilizar export para que la variable IDIOMA se entienda en saludo, ya que es el mismo shell y por tanto conoce esta variable.

**Entrecomillado**

Las comillas utizadas en las cadenas de caracteres pueden ser dobles ("cadena") o simples ('cadena'). No se debe confundir con las comillas simples hacia la derecha utilizadas para órdenes que se van a interpretar por el shell (`orden`).

Tanto las dobles como las simples permiten introducir espacios en la cadena e interpretar las variables que se incluyen en ellas ("La variable es $VAR").

Para incluir comillas en una cadena se puede hacer de la forma que se muestra en el ejemplo:

**⇒ Ejemplo de comillas:**

```
#!/bin/bash
COMENT=comentario
echo "Esto es un $COMENT"
VAR1="Esto es un 'comentario' "
echo $VAR1
VAR2='Esto es un "comentario" '
echo $VAR2
```

ejcomillas

resultado:

```
>ejcomillas
Esto es un comentario
Esto es un 'comentario'
Esto es un "comentario"
```

También se puede realizar utilizando el carácter de escape \ como en el siguiente ejemplo:

**⇒ Ejemplo de carácter de escape:**

```
#!/bin/bash
```

```
echo "Esto es un \"comentario\""
```

ejescape

resultado:

```
>ejcomillas
Esto es un "comentario"
```

El carácter de escape hace que el carácter siguiente se interprete como un carácter normal sin ningún significado especial. Por ejemplo si se quisiera imprimir $VAR en vez de interpretar la variable y poner su valor habría que poner \$VAR.

Otras secuencias de escape son \n (nueva línea), \t (tabulador), \\ (una \ sencilla), \r (retorno), \v (tabulación vertical), entre otras.

**Línea de órdenes**

En la línea de órdenes se puede dar la orden `history` para ver las últimas órdenes que se han dado.

Con !cad se puede encontrar la última orden que empieza con cad que se ejecutó en el intérprete. Esto facilita la introducción de órdenes.

**⇒ Ejemplo history:**

```
> more comillas
#!/bin/bash
VAR1="Esto es un 'comentario' "
echo $VAR1
VAR2='Esto es un "comentario" '
echo $VAR2
> ls
bin/                comillas~           practicas/
comillas*           html/               progshell/
> history
    1  more comillas
    2  ls
    3  history
> !mo
more comillas
#!/bin/bash
VAR1="Esto es un 'comentario' "
echo $VAR1
VAR2='Esto es un "comentario" '
echo $VAR2
>
```

Se pueden ejecutar secuencias de control desde la línea de órdenes. Por ejemplo de la siguiente forma:

```
> for i in 1 2 3 4 5 ; do
> echo $i
> done
```

```
w1
2
3
4
5
```

También se puede utilizar el “;” para separar las órdenes como en el ejemplo siguiente:

```
> for i in 1 2 3 4 5 ; do echo $i ; done
1
2
3
4
5
```