CIMSI – CONFIGURACIÓN, IMPLEMENTACIÓN Y MANTENIMIENTO DE SISTEMAS INFORMÁTICOS

PRÁCTICAS
SESIÓN 1 – SHELL SCRIPTING

Daniel Cascado Caballero
Rosa Yáñez Gómez
Raouf Senhadji Navarro
Mª José Morón Fernández

EL PRIMER *SCRIPT*

Hay que conocer un editor de textos:

vi, nano, emacs (o xemacs).

Iniciar el editor de textos preferido y escribir el siguiente código:

```
#!/bin/bash
echo “Hola Mundo”
```

La primera línea indica a Linux que debe utilizar el intérprete bash para ejecutar el script. Las lineas de comentarios empiezan por # en la columna 1

Hay que hacer que el *script* sea ejecutable

```
$ chmod 700 hola.sh  o bien $ chmod +x hola.sh
```

Se comprueba que es ejecutable:

```
$ ls –l
-rwx------ hola.sh
```

Ejecución del script:

./<nombre_del_script>

VARIABLES

No es necesario declarar una variable ⇒ se crean asignándole un valor a su referencia

Los valores:

- Se almacenan como tipo cadena de texto ⇒ Hay operadores matemáticos que convierten el valor de las variables en número para el cálculo
- Se recuperan anteponiendo un símbolo '$' al nombre

Ejemplo:

```
#!/bin/bash
STR="Hola Mundo!"
echo $STR
```

VARIABLES INTRÍNSECAS (I)

$#: número argumentos, exluyendo el nombre del programa

$*: Cadena de argumentos entera, excluyendo el nombre del programa, como una única cadena

"S*" ≡ "$1 $2 $3"

$@: Cadena de argumentos entera, excluyendo el nombre del programa, como cadenas separadas

"S@" ≡ "$1" "$2" "$3"

VARIABLES INTRÍNSECAS (II)

$-: opciones suministradas a la shell

$?: Código de salida de la última orden ejecutado en primer plano del script

- 0: Ejecución correcta
- ≠0: Ejecución incorrecta

$!: PID del último proceso ejecutado en segundo plano

$$: PID del proceso shell (shell “hijo”) que ejecuta el script

COMILLAS SIMPLES Y DOBLES

Simples: Mostrarán una cadena de caracteres de forma literal sin resolución de variables

> var=’cadena de prueba’
> nuevavar=’Valor de var es $var’
> echo $nuevavar
Valor de var es $var

Dobles: Si, en su contenido se referencia una variable, ésta será resuelta a su valor:

> var=“cadena de prueba”
> nuevavar=“Valor de var es $var”
> echo $nuevavar
Valor de var es cadena de prueba

EVALUACIÓN ARITMÉTICA

La instrucción let se puede utilizar para realizar funciones matemáticas:

```
$ let X=10+2*7
$ echo $X
24
$ let Y=X+2*4
$ echo $Y
32
```

Importante: expresión sin espacios en blanco

No es necesario utilizar $X para referenciar el valor de X

Un expresión aritmética se puede evaluar con $[expression] o $((expression))

```
$ echo $((123+20))
  143
$ VALOR=$[123+20]
$ echo $[123*$VALOR]
  1430
$ echo $[2**3]
$ echo $[8%3]
```

EVALUACIÓN DE EXPRESIONES (I)

Una expresión puede ser: comparación de cadenas, comparación numérica, operadores de fichero y operadores lógicos y se representa mediante [expresión]:

- **Comparación de cadenas**:
  - == Igualdad
  - != Desigualdad
  - -n evalúa si la longitud de la cadena es superior a 0
  - -z evalúa si la longitud de la cadena es igual a 0

Ejemplos:

- [s1 == s2] (true si s1 es igual a s2, sino false)
- [s1 != s2] (true si s1 no es igual a s2, sino false)
- [s1] (true si s1 no está vacía, sino false)
- [-n s1] (true si s1 tiene longitud mayor que 0, sino false)
- [-z s2] (true si s2 tiene longitud 0, sino false)

EVALUACIÓN DE EXPRESIONES (II)

Una expresión puede ser: comparación de cadenas, comparación numérica, operadores de fichero y operadores lógicos y se representa mediante [expresión]:

- **Comparación numérica**:
  - eq igualdad
  - ge mayor o igual
  - le menor o igual
  - ne no igual
  - gt mayor que
  - lt menor que

Ejemplos:

- [n1 -eq n2]
- [n1 -ge n2]
- [n1 -le n2]
- [n1 -ne n2]
- [n1 -gt n2]
- [n1 -lt n2]

EVALUACIÓN DE EXPRESIONES (III)

Una expresión puede ser: comparación de cadenas, comparación numérica, operadores de fichero y operadores lógicos y se representa mediante [expresión]:

- **Operadores de archivos:**
  - **-d** verifica si el path dado es un directorio
  - **-f** verifica si el path dado es un archivo
  - **-s** verifica si el path dado en un link simbólico
  - **-e** verifica si el fichero existe
  - **-s** verifica si el fichero tiene un tamaño mayor a 0
  - **-r** verifica si el fichero tiene permiso de lectura
  - **-w** verifica si el fichero tiene permiso de escritura
  - **-x** verifica si el fichero tiene permiso de ejecución

Ejemplos:

- [-d nombre_fichero]
- [-f nombre_fichero]
- [-e nombre_fichero]
- [-s nombre_fichero]
- [-r nombre_fichero]
- [-w nombre_fichero]
- [-x nombre_fichero]

EVALUACIÓN DE EXPRESIONES (IV)

Una expresión puede ser: comparación de cadenas, comparación numérica, operadores de fichero y operadores lógicos y se representa mediante [expresión]:

- **Operadores lógicos:**
  - **!** **NOT**
  - **-a** **AND**
  - **-o** **OR**

Ejemplo:

```
#!/bin/bash
echo -n “Introduzca un número entre 1 < x < 10:”
read num
if [ “$num” -gt 1 –a “$num” -lt 10 ];
   then
            echo “$num*$num=$(($num*$num))”
   else
            echo “Número introducido incorrecto !”
fi
```

SECUENCIAS DE CONTROL: CONDICIONAL

Importante: hay que poner espacios en blanco para separar los corchetes

if

```
if [ condition ]
then
                    <do something>
else
                    <do something else>
fi
```

Se puede añadir el comando test para hacer cualquier tipo de comparación

```
if [ test $a == $b  ]
then
                    echo $a
fi
```

SECUENCIAS DE CONTROL: BUCLES

for

```
for VAR in LIST
do
        <something>
done
```

Lista de palabras seguidas, separadas por espacio o por CR

while

```
while [ condition ]
do
        <something>
done
```

Atención de nuevo a los espacios en blanco

COMANDO EXIT

El comando exit se puede utilizar para:

- Finalizar la ejecución de un *script*
- Retornar al proceso padre del *script*, un valor

exit nnn

- nnn: el estado de salida
- nnn $\in$ [0,255]

Cuando un script termina con exit sin parámetros, el estado de salida será el del último comando ejecutado en el *script*