Práctica 3. Programación en *shell* de *UNIX*.

1.- Objetivos:

- Utilizar comandos elementales de *UNIX*.
- Utilizar el funcionamiento básico del editor de textos *vi* de *UNIX*.
- Modificar la fuente de los datos de entrada o el destino de los resultados de las órdenes *UNIX*.
- Filtrar el contenido de archivos de texto para seleccionar solamente parte la la información que contienen.
- Establecer y modificar permisos de acceso al sistema de archivos *UNIX*.
- Optimizar el acceso al sistema de archivos mediante el uso de caracteres especiales.
- Utilizar variables de *shell* y realizar operaciones aritméticas con ellas.
- Obtener nociones básicas de la programación y ejecución de *scripts* de *shell*.

2.- Shell scripting:

Introducción.

La programación en *shell* surge como respuesta a tareas que se realizan de manera secuencial y repetitiva. Por ejemplo, dar de alta cuentas de usuario en el laboratorio.

Un *script* de *shell* no es más que una secuencia de comandos escritos en un fichero de texto plano. Para poder ejecutar dicha secuencia de comandos es preciso introducir la siguiente cabecera al principio del mismo:

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

Un ejemplo de script válido sería el siguiente:

```
#!/bin/bash
 echo ¡Hola Mundo!
```

Para ejecutar un script basta con darle permisos de ejecución al fichero que lo contiene y después ejecutarlo normalmente indicando su ruta (absoluta o relativa).

Normalmente, la secuencia de comandos dentro del *script* se separa mediante saltos de línea. También podemos introducir varios comandos en una línea separándolos por el carácter *;*.

Se pueden incluir comentarios en el *script* precediéndolos el carácter *#*. Por ejemplo:

```
# Esto es un comentario
```

Declaración de variables.

Las variables de *shell* no necesitan ser declaradas previamente. Para acceder a su contendido hay que anteponer el carácter *$*.

Por ejemplo:

```
#!/bin/bash
var1=¡Hola!
echo $var1
var2=¡Mundo!
echo $var2
```

Paso de argumentos a un script.

El paso de parámetros se realiza escribiéndolos en la línea de comandos detrás del nombre del *script*. Dentro del mismo podremos acceder a ellos con las variables *$1*, *$2*, *$3*, etc.

Las variables *$#* y *$** tienen un significado especial:

- *$#*: Contiene el número de parámetros pasados al *script*.
- *$**: Contienen todos los parámetros pasados al *script*.

Por ejemplo, dado el script *eco.sh* cuyo contenido es:

```
#!/bin/bash
echo $0, $1, $2, $3
echo $#
echo $*
```

El resultado de ejecutar el comando `./eco.sh` uno dos tres sería:

```
./holamundo.sh, uno, dos, tres
3
uno dos tres
```

Lectura de variables desde la entrada estándar.

Utilizaremos la orden *read*. Con *read* podemos leer más de una variable a la vez.

Por ejemplo, si ejecutamos el siguiente *script*:

```
#!/bin/bash
read var1 var2 var3
echo $var1, $var2, $var3
```

E introducimos '`uno dos tres`' en la entrada, el resultado sería:

```
uno, dos, tres
```

Expresiones númericas.

Para evaluar expresiones numéricas utilizaremos la orden *expr* entre caracteres `.

Por ejemplo, la ejecución del *script*:

```
#!/bin/bash
a=1
b=2
c=`expr $a + $b`
echo $c
```

Producirá la siguiente salida:

```
3
```

Expresiones lógicas.

Para evaluar expresiones condicionales utilizaremos la orden *test*.

Sintaxis: `test` *`EXPRESION`*

Comparaciones numéricas:

Significado	**Expresión**
`INTEGER1 = INTEGER2`	`INTEGER1 -eq INTEGER2`
`INTEGER1 < INTEGER2`	`INTEGER1 -lt INTEGER2`
`INTEGER1 <= INTEGER2`	`INTEGER1 -le INTEGER2`
`INTEGER1 <> INTEGER2`	`INTEGER1 -ne INTEGER2`
`INTEGER1 > INTEGER2`	`INTEGER1 -gt INTEGER2`
`INTEGER1 >= INTEGER2`	`INTEGER1 -ge INTEGER2`

Expresiones condicionales sobre cadenas de caracteres:

Significado	**Expresión**
`STRING1 = STRING2`	`STRING1 = STRING2 (Con espacios a ambos lados para distinguirla de la asignación)`
`STRING1 <> STRING2`	`STRING1 != STRING2`
`STRING` es una cadena vacía	`-z STRING`
`STRING` no es una cadena vacía	`-n STRING`

Expresiones condicionales sobre ficheros:

Significado	**Expresión**
`FILE` existe y es un archivo o un subdirectorio	`-a FILE`
`FILE` existe y es un archivo	`-f FILE`
`FILE` existe y es un subdirectorio	`-d FILE`
`FILE` existe y es un enlace simbólico	`-h FILE`

`FILE` tiene permisos de lectura	`-r FILE`
`FILE` tiene permisos de ejecución	`-x FILE`
`FILE` tiene permisos de escritura	`-w FILE`
`FILE` existe y no está vacío	`-a FILE`

Comparación de expresiones lógicas:

Significado	**Expresión**
`EXPRESION1` AND `EXPRESION2`	`EXPRESION1 -a EXPRESION2`
`EXPRESION1` OR `EXPRESION2`	`EXPRESION1 -o EXPRESION2`
NOT `EXPRESION`	`!EXPRESION`

Ejecución condicional.

Lo primero que tenemos que conocer es que la variable *$?* contiene el código de salida del último comando ejecutado en el sistema. Toma el valor *0* si el último comando se ha ejecutado con éxito y *<>0* si no ha sido así.

Ejecución condicional. Sentencia if.

Sintaxis:

```
if EXPRESION
then
...
#parte opcional
else
...
#fin de la parte opcional
fi
```

Ejemplo:

```
#!/bin/bash
a=$1
b=$2
if test $a -eq $b
then
  echo $a es igual a $b
else
  echo $a no es igual a $b
fi
```

Ejecución condicional. Bucle for.

Sintaxis:

```
for variable in conjunto_de_valores
do
...
done
```

Ejemplo:

```
#!/bin/bash
for i in $( ls . )
do
  echo He encontrado el fichero $i
done
```

Ejecución condicional. Bucle while.

Sintaxis:

```
while EXPRESION
do
  ...
done
```

Ejemplo:

```
#!/bin/bash
numFiles=`ls | wc -l`
count=0
while test $count -lt $numFiles
do
  count=`expr $count + 1`
  echo He encontrado $count fichero/s
done
```

Ejecución condicional. Bucle until.

Sintaxis:

```
while EXPRESION
do
  ...
done
```

Ejemplo:

```
#!/bin/bash
numFiles=`ls | wc -l`
count=0
while test $count -eq $numFiles
do
  count=`expr $count + 1`
  echo He encontrado $count fichero/s
done
```

Ejecución condicional. Sentencia case.

Sintaxis:

```
case $variable in
valor1)
  ...;;
valor2)
  ...;;
...
# Opcional el otherwise
*)
  ...;;
esac
```

Ejemplo:

```
#!/bin/bash
echo Por favor, introduce un número entre el 1 y el 3
read input
case $input in
1) echo Has introducido el número $input;;
2) echo Has introducido el número $input;;
3) echo Has introducido el número $input;;
*) echo "¡Dije un número entre el 1 y el tres!";;
esac
```

Sentencias break, countinue y exit.

Dentro de los bucles *for*, *while* y *until* podemos usar:

- *break* para detener la ejecución del bucle.
- *continue* para detener la iteración actual y continuar con la siguiente iteración del bucle.

Podemos usar *exit* para terminar la ejecución del *script*. Podemos incluir también un código de retorno al sistema operativo, por ejemplo *0* si la ejecución es correcta.

Ejemplo:

```
#!/bin/bash
while true
do
  echo Por favor, introduce un número entre el 1 y el 3
  read input
  case $input in
    1) echo Has introducido el número $input
       exit 0;;
    2) echo Has introducido el número $input
       exit 0;;
```

```
    3) echo Has introducido el número $input
       exit 0;;
    *) echo "¡Dije un número entre el 1 y el tres!";;
  esac
done
```

Funciones y procedimientos.

Las funciones en un script de shell han de ser declaradas al principio del script.

Sintaxis:

```
function NOMBRE_FUNCION()
{
...
}
# Resto del script
...
```

Una función devolverá un valor, siempre que haga una única salida por *stdout* que será almacenada en la variable a la que sea asignada la llamada de la función.

Dentro de una función accedemos a los argumentos que le pasemos como si de un *script* independiente se tratara, es decir, a través de las variables *$1*, *$2*, etc.

Ejemplo:

```
#!/bin/bash
function cuadrado()
{
  result=`expr $1 \* $1`
  echo $result
}

echo Por favor, introduce un número
read num
num2=`cuadrado $num`
echo El cuadrado de $num es $num2
```

Operadores && y ||.

Podemos utilizar los operadores && y || para realizar ejecución de comandos condicionada:

Sintaxis	**Significado**
`COMANDO1 && COMANDO2`	`COMANDO2` se ejecuta sí, y sólo sí, `COMANDO1` se ha ejecutado con éxito.
`COMANDO1 || COMANDO2`	`COMANDO2` se ejecuta sí, y sólo sí, `COMANDO1` NO se ha ejecutado con éxito.

3.- Desarrollo de la práctica

Escribe un *script UNIX* que calcule ciertas estadísticas sobre un directorio que reciba como argumento. El script debe controlar el número de argumentos:

- Si no recibe ninguno trabajará sobre el directorio actual.
- Si recibe un argumento lo tomaremos como directorio de trabajo. Si no es un directorio válido debemos finalizar la ejecución del *script* e indicar el motivo al usuario.
- Si recibe más de un parámetro debemos finalizarla ejecución del *script* e indicar el motivo al usuario.

Una vez establecido el directorio de trabajo, se debe mostrar al usuario un menú con las siguientes opciones:

1) Numero de archivos: Cuando el usuario escoja esta opción se visualizará el número de archivos dentro del directorio de trabajo.
2) Numero de subdirectorios: Cuando el usuario escoja esta opción se visualizará el número de subdirectorios dentro del directorio de trabajo.
3) Fichero más grande: Cuando el usuario escoja esta opción se visualizará el nombre del fichero de mayor tamaño dentro del directorio de trabajo.
4) Fichero más pequeño: Cuando el usuario escoja esta opción se visualizará el nombre del fichero de menor tamaño dentro del directorio de trabajo.
5) Espacio total ocupado : Cuando el usuario escoja esta opción se visualizará el espacio total ocupado en disco por el directorio de trabajo en un formato legible (Mb, Gb, etc.)
6) Número de ficheros con permiso de lectura para el usuario que ejecuta el script: Cuando el usuario escoja esta opción se visualizará el número de archivos con permiso de lectura para el usuario que ejecuta el script dentro del directorio de trabajo.
7) Número de ficheros con permiso de escritura para el usuario que ejecuta el script: Cuando el usuario escoja esta opción se visualizará el número de archivos con permiso de escritura para el usuario que ejecuta el script dentro del directorio de trabajos
8) Número de ficheros con permiso de ejecución para el usuario que ejecuta el script: Cuando el usuario escoja esta opción se visualizará el número de archivos con permiso de ejecución para el usuario que ejecuta el script dentro del directorio de trabajos
9) Ficheros con permiso de ejecución para todos los usuarios: Cuando el usuario escoja esta opción se visualizarán todos los archivos con permiso de ejecución para cualquier usuario.
10) Salir: Cuando el usuario escoja esta opción se finalizará la ejecución del script.

El usuario debe escoger una opción de la lista, si se introduce un valor diferente, se debe informar al usuario y preguntar de nuevo.

Una vez escogida una opción válida, se ejecutará la tarea que corresponda y se le volverá a preguntar al usuario que desea hacer a continuación. Repetiremos el proceso hasta que el usuario seleccione la opción *salir*, en cuyo caso finalizaremos la ejecución del script.

Además de los comandos que hemos visto hasta ahora puede que alguno de los siguientes te sea de utilidad:

Comando	**Significado**
`man COMANDO`	Visualiza el fichero de ayuda disponible para `COMANDO`. Este comando es muy importante ya que nos permite conocer los modificadores de cualquier comando.
`head -X FILE`	Visualiza las `X` primeras líneas de `FILE`.
`tail -X FILE`	Visualiza las `X` últimas líneas de `FILE`.
`du`	Visualiza el espacio utilizado en el directorio actual o en el que se le indique como argumento.
`awk`	Herramienta para la búsqueda de patrones y procesado de textos. Mucho más compleja y potente que la orden `cut`.