ÍNDICE

1. CARACTERES ESPECIALES....2
2. VARIABLES Y PARÁMETROS....3
3. REDIRECCIONAMIENTOS....4
4. TUBERÍAS....4
5. TESTS....4
6. EVALUACIÓN MATEMÁTICA....5
7. ESTRUCTURAS DE CONTROL....6
If Se evalúa la expresión y si se obtiene un valor cierto, entonces se ejecuta los comandos....6
case....6
for....7
for variable in lista-de-valores; do....7
while: ejecuta comandos mientras la expresión es cierta....7
until: igual que while excepto que el bucle se ejecuta mientras expresión no es 0....7
break: Termina la ejecución del bucle mas interior causando la ejecución de la instrucción más cercana....7
continue :Causa la ejecución de la instrucción while, until o for....7
8. FUNCIONES....8
Ejemplo de funciones....8
Ejemplo de funciones con parámetros....8
9. FILTROS....8
grep....8
tr....9

1. CARACTERES ESPECIALES

#!/bin/bash

Todos los scripts empiezan con esta línea.

Sirve para decir que es un **fichero ejecutable e indicar cual es el intérprete** de comandos.

**Comentario**: todo lo que haya tras él es ignorado.

; **Separa dos comandos**: echo "la fecha de hoy es: " ; date

“” **para cadenas**.

‘’ **para caracteres**. No interpreta los caracteres especiales.

`` **ejecuta un comando:** echo `date`

$ Accede al valor de una **variable**: echo $PATH

~ Equivale al **directorio 'home' del usuario**, es equivalente a leer el valor $HOME

& Escrito después de un comando, lo **ejecuta en segundo plano**.

EJEMPLO DE CARACTERES ESPECIALES E INICIALIZACIÓN DE VARIABLES

```
1 #!/bin/bash
2 variable=10
3 echo "contenido de la variable $variable" # si
4 echo 'contenido de la variable $varable' #no se
5 echo "la fecha es:"`date`
6 echo 'la fecha es:'; date
```

```
usuario@virtual22:~/Documentos/scripts$ ./ej1.sh
contenido de la variable 10
contenido de la variable $varable
la fecha es:lun 13 mar 2023 20:06:19 CET
la fecha es:
lun 13 mar 2023 20:06:19 CET
```

2. VARIABLES Y PARÁMETROS

VARIABLES DE ENTORNO

Ver el valor de todas las variables de entorno mediante comando **set** sin argumentos.

Algunas variables de entorno:

- **$HOME** Directorio 'home' del usuario.
- **$PATH** Rutas de los ejecutables cuando se invoca un comando.
- **$?** Contiene el valor de salida del último comando ejecutado. Es útil para saber si un comando ha finalizado con éxito o ha tenido problemas. Un '0' indica que no ha habido errores, otro valor indica que sí ha habido errores.
- **$UID** Identificador del usuario que ejecuta el script.
- **$!** Identificador de proceso del último comando ejecutado en segundo plano.

VARIABLES DEFINIDAS POR USUARIO

- No es necesario declarar las variables,
- Crear variable: Se asigna un valor a una variable: **variable = valor**
- Para **acceder** al valor que contiene una variable se usa el carácter $: **echo $variable**

```
#!/bin/bash
#EJEMPLO DE VARIABLE SOLICITADA
read -p "Introduce número " num
echo "El número introducido ha sido" $num

#EJECUCIÓN1
sh ejemplo1.sh
#EJECUCIÓN2
chmod 777 ejemplo2.sh
./ejemplo2.sh
#RESULTADO
Introduce número 56
El número introducido ha sido 56
```

PARÁMETROS RECIBIDOS

- Los parámetros recibidos se guardan en una serie de variables que el script puede consultar. $1 $2 $3 .... $10 $11....
- La variable **$0** contiene el **nombre** del **script**.
- El comando **shift** mueve los parámetros una posición a la izquierda, esto hace que el parámetro que haya en $1 desaparezca, y sea reemplazado por el que había en $2.
- **$#** contiene el número de parámetros que ha recibido el script.
- **$@** contiene todos los parámetros.
- $* contiene todos los parámetros.

Para pasar parámetros en el momento de ejecutar un script se indican separados por espacios después del nombre del script

```
#!/bin/bash
echo El parámetro '$1' es $1
echo El parametro '$2' es $2
echo El parametro '$3' es $3
echo Numero de parametros $#
echo Todos los parametros $*
echo Todos los parametros $@
echo Nombre de archivo $0
echo $1;shift;echo $2
```

```
usuario@virtual22:~/Documentos/scripts$ ./ej1.sh María Juan Pedro
El parámetro $1 es María
El parametro $2 es Juan
El parametro $3 es Pedro
Numero de parametros 3
Todos los parametros María Juan Pedro
Todos los parametros María Juan Pedro
Nombre de archivo ./ej1.sh
María
Pedro
```

3. REDIRECCIONAMIENTOS

Salida a un fichero

ls -l > fichero.txt #La salida de ls -l se escriba en el fichero fichero.txt

Salida de error a un fichero

grep da * 2> errores-de-grep.txt #Se creará un fichero llamado 'errores-de-grep.txt' que contendrá los errores de la salida del comando grep da *

4. TUBERÍAS

Permiten utilizar la salida de un programa como la entrada de otro

ls -l | grep "*.txt" # La salida del comando ls -l se envía al comando grep

5. TESTS

Un test es una expresión que permite evaluar si una expresión es verdadera o falsa.

Hay dos formas de escribir un test, [ ] y [[ ]]. No son equivalentes, (por ejemplo los operadores && || solo funcionan en la última)

```
if [ test ];then
   comando
else
   comando
fi
```

Tests de ficheros. Toman el nombre de un fichero y devuelven verdadero o falso:

```
if [ -e fichero ];then
   echo "fichero existe"
fi
```

- **-e** Si el fichero existe.
- **-f** Si es fichero regular (que no es ni un directorio, ni un dispositivo).
- **-d** Si es directorio.
- **-h** Si el fichero es un enlace simbólico.
- **-r** Si se tiene permiso para leer el fichero.
- **-w** Si se tiene permiso para escribir el fichero.
- **-x** Si se tiene permiso para ejecutar el fichero.

Operadores de comparación de enteros.

```
[ "$a" -eq "$b" ]
```

- **-eq** igual a
- **-ne** no es igual a
- **-gt** es mayor que
- **-ge** es mayor o igual que
- **-lt** es menor que
- **-le** es menor o igual que

Operadores de comparación de cadenas.

= ó == igualdad

!= desigualdad.

! se puede colocar delante de cualquier test para negar su resultado.

< Menor que

>Mayor que.

6. EVALUACIÓN MATEMÁTICA

No se puede realizar operaciones con : echo 1 + 1

Para ello se utiliza:

`echo $((1+1))`

`echo $[1+1]`

Para usar fracciones, u otras matemáticas, puede utilizar **bc:** `echo 3/4 | bc -l`

7. ESTRUCTURAS DE CONTROL

if Se evalúa la expresión y si se obtiene un valor **cierto**, entonces se ejecuta los comandos.

```
if [expresión];then
    comandos
else
    comandos
fi
```

```
#!/bin/bash
#EJEMPLO IF:Pide una nota numerica y
read -p "Introduce una nota: " nota
if [ $nota -ge 9 ];then
        echo "sobresaliente"
    elif [ $nota -ge 7 ];then
        echo "notable"
    elif [ $nota -ge 6 ];then
        echo "bien"
    elif [ $nota -ge 5 ];then
        echo "suficiente"
else
        echo "Insuficiente"
fi
```

if se puede anidar

```
if [expresión1]; then ...
        elif [expresión2];then      ...
        else
          ...
fi
```

En la **expresión** se puede usar:**El operador && (y) y El operador || (o)**

case

El flujo del programa se controla en base a una palabra.

Esta palabra se compara con cada patrón hasta que se encuentra uno que haga juego.

Cuando lo encuentra, se ejecuta el comando asociado y se termina la instrucción.

```
case palabra in
  patrón 1) comandos ;;
  patrón 2 | patrón3) comandos ;;
  patrón N) comandos ;;
esac
```

Un comando puede asociarse con **mas de un patrón**. Los patrones deben separarse con |

El **orden** de chequeo es el orden en que aparecen los patrones.

Para especificar un **patrón por defecto** : ***)**

```
read -p "¿Orden estalbecido, ascendente o descentente?(o/a/d) " orden
case $orden in
    o|O)
    cat numeros.txt;;
    a|A)
            cat numeros.txt | sort -n;;
    d|D)
            cat numeros.txt | sort -n -r;;
    *)
            echo Opción no válida;;
esac
```

for

for variable in lista-de-valores; do

comandos

done

```
#!/bin/bash
for i in `seq 1 5`;do
    echo $i
done
```

```
usuario@virtual22:~/Documentos/scripts$ ./ej1.sh
1
2
3
4
5
```

La variable puede tomar el valor de una lista

```
#!/bin/bash
for i in `ls *.sh`;do
if [ -x "$i" ]; then
echo "El fichero $i es ejecutable"
fi
done
```

```
usuario@virtual22:~/Documentos/scripts$ ./ej1.sh
El fichero ej1.sh es ejecutable
El fichero ej2.sh es ejecutable
usuario@virtual22:~/Documentos/scripts$
```

while: ejecuta comandos mientras la expresión es cierta

while [expresión];do

comandos

done

```
#!/bin/bash
number=10
while [ $number -gt 5 ];do
    echo $number
    number=$(($number-1))

done
```

```
usuario@virtual22:~/Documentos/scripts$ ./ej1.sh
10
9
8
7
6
usuario@virtual22:~/Documentos/scripts$
```

until: igual que while excepto que el bucle se ejecuta mientras expresión **no es 0**.

until [expresión];do

comandos

done

```
#!/bin/bash
number=10
until [ $number -le 5 ];do
    echo $number
    number=$(($number-1))

done
```

```
usuario@virtual22:~/Documentos/scripts$ ./ej1.sh
10
9
8
7
6
usuario@virtual22:~/Documentos/scripts$
```

break: Termina la ejecución del bucle mas interior causando la ejecución de la instrucción más cercana.

continue : Causa la ejecución de la instrucción while, until o for.

Ejemplo :Mientras el usuario introduzca un comando o un nulo el script continua funcionando. Para pararlo el usuario debe teclear fin.

```
#!/bin/bash
while read -p "Por favor introduce un comando: "  respuesta;do
case "$respuesta" in
    'fin') break;;  # no mas comandos
     "") continue;; # continua
     *) $respuesta;; # ejecuta el comando almacenado en variable
esac
done
```

```
usuario@virtual22:~/Documentos/scripts$ ./ej1.sh
Por favor introduce un comando: ls
ej1.sh  ej2.sh
Por favor introduce un comando: date
lun 13 mar 2023 21:33:31 CET
Por favor introduce un comando: fin
usuario@virtual22:~/Documentos/scripts$
```

8. FUNCIONES

Declarar una función: **miFunc() { código }**

Llamar a la función : escribir su nombre: **miFunc**

Ejemplo de funciones

declaración

```
salir() {
        exit
}
hola() {
        echo “Hola!”
}
```

llamadas

```
hola
salir
```

Ejemplo de funciones con parámetros

declaración

```
fej1() {
        echo $1
}
```

llamada

```
ej1 María
```

```
1 #!/bin/bash
2 hola(){
3     echo "Esta es la función hola"
4 }
5 adios(){
6     echo "Esta es la función adiós"
7 }
8
9 prueba (){
10    echo "Esta es una función con el parámetro: " $1
11 }
12 hola
13 adios
14 prueba Maria
```

```
usuario@virt
usuario@virtual22:~/Documentos/scripts$ ./ej3.sh
Esta es la función hola
Esta es la función adiós
Esta es una función con el parámetro:  Maria
usuario@virtual22:~/Documentos/scripts$
```

9. FILTROS

grep

Busca las líneas que contienen una cadena de caracteres (PATRÓN).

```
grep patrón archivos
grep -w “usuario” /etc/passwd
```

cut

Cortar por campo usando:

du -a | grep -w "fichero" | cut -f2

Cortar usando carácter delimitador:

ls -l | grep -w “fichero” | cut -d" " -f1

sort

Ordenación puede ser por caracteres ASCII o por valor numérico.

```
sort arch1 #ordena según el código ASCII.
sort -n arch2.num  #ordena numéricamente.
sort -t: -k1,3 arch1.txt #ordena campos separados por ":", campo 1 hasta el último del campo 3.
```

tr

Traduce los caracteres, sustituyendo unos caracteres por otros.

Opciones:

-d para **borrar** caracteres;

-c para **sustituir** complemento(contrarios) de los caracteres y

-s para **comprimir caracteres repetidos en uno solo**.

```
cat dias.txt | tr a-z A-Z #convierte todo a mayúsculas.
cat dias.txt | tr -d aeiou #borra todas las vocales del archivo días.
cat dia.txt | tr -s “  “ “  “ # sustituye los espacios blancos por uno sólo
```

```
cat fichero.txt | tail -1 | tr -s " " " | cut -d" " -f1 #visualiza la última línea de un fichero y obtiene la primera palabra (si hay varios espacios los sustituye por uno sólo, muy importante para que funcione bien el cut)
```