



UNIVERSIDAD DE CÓRDOBA
comprometida con el desarrollo regional



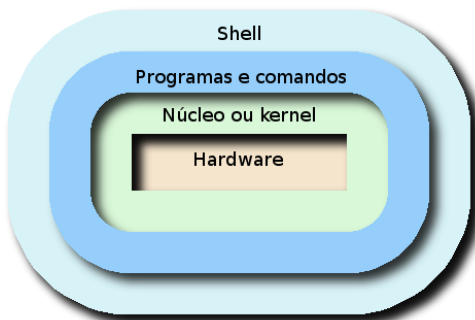
CENTRO DE INNOVACION EN TIC PARA APOYO A LA ACADÉMIA

CINTIA

PROGRAMACIÓN EN SHELL SCRIPT

1.1 ¿Qué es un Shell Script?

Para entender que es un Shell Script se debe analizar primero que es la Shell y como es la estructura del núcleo del sistema operativo GNU/Linux, cabe destacar que el núcleo o kernel de cualquier sistema operativo es el componente central que se encarga de asignar el hardware a las aplicaciones según estas lo requieran, en el kernel también se encuentran subsistemas como gestión de procesos, gestión de memoria, gestión de redes sistema de archivos y otros; todos estos subsistemas son controlados por elementos del núcleo como el bootloader, la Shell, el servidor grafico entre otros. Todo este conjunto de subsistemas lo que permite es que el usuario pueda interactuar fácilmente con el sistema y es aquí donde la Shell juega un papel muy importante ya que esta no es mas que un interprete de comandos entre el usuario y el sistema operativo. Es decir, la Shell es un módulo que actúa como capa externa entre el sistema operativo y el usuario.



Núcleo de Linux [Figura 1]. Recuperado de:
http://wiki.softwarerivre.org/TWikiBar/TWikiBarConversa001#El_Ambiente_Shell

Cabe destacar que a lo largo de la historia del desarrollo del núcleo de Linux se han creado muchos tipos de Shell:

- **Bouerne Shell (sh):** Esta Shell fue utilizada por el S.O de Unix por muchos años, fue desarrollada por Stephen Bourne.
- **Korn Shell (ksh):** es un subconjunto de sh, por lo que posee todas las facilidades de esta y agreraron otras para mejorar su funcionalidad, fue desarrollada por David Korn.

Boune Again Shell (bash): es la Shell más moderna y es con la que viene el sistema operativo Linux por defecto, por lo cual tiene una gran comunidad que le brinda soporte y mejoras, por ello es esta Shell en la que se desarrollara está unida.

Por otra parte, el termino **Script** se puede traducir como “*un guion*” en donde se escriben líneas de comandos que son interpretados por la Shell, es decir un Script no es más que un archivo de texto plano en donde se escribe código que es interpretado y ejecutado por la Shell con la particularidad que al final del nombre se coloca **.sh** por ejemplo “*miscript.sh*”.

Para ejecutar un script en la terminar es necesario darle permisos de ejecución con el comando **chmod** por ejemplo:

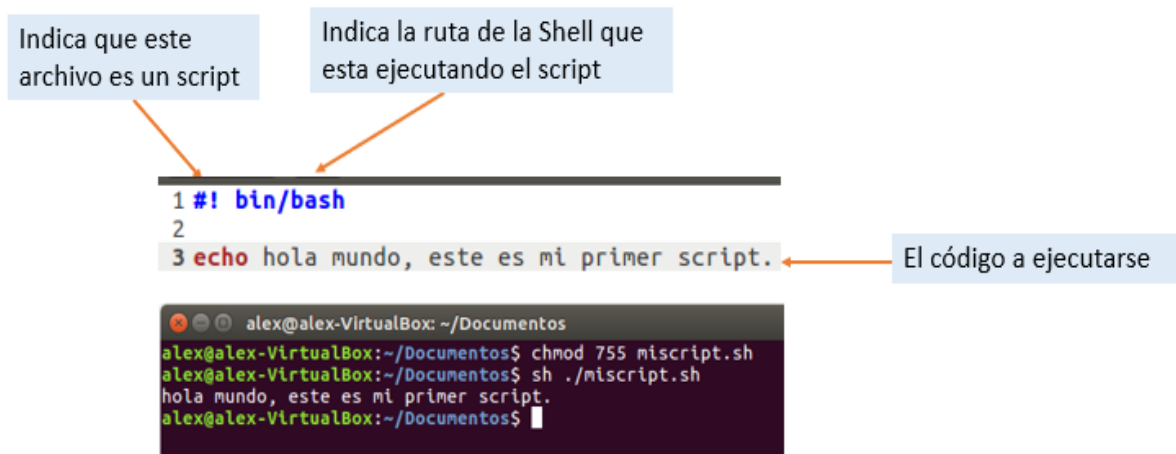
“*chmod 755 /ruta_del_script*” Para el usuario actual

“*chmod 777 /ruta_del_script*” Para cualquier usuario.

Una ves dado los permisos correspondientes se escribe en consola *sh ./nombre_del_script.sh*

Por ejemplo:
“`sh ./miscript.sh`” donde **sh** es el comando para ejecutar script.

Un script en su primera línea de código siempre debe “`#!/ bin/bash`” como se describe en la imagen.



Fuente Propia (2018). Shell Script [Figura 2].

Para mostrar en consola el resultado de un Script se usa el comando “`sh ./Nombre_script.sh`”

1.2 Manejo de variables

El manejo de variables en un script es similar a cualquier lenguaje de programación, pero para entender mejor la sintaxis de este lenguaje se debe tener en cuenta lo siguiente:

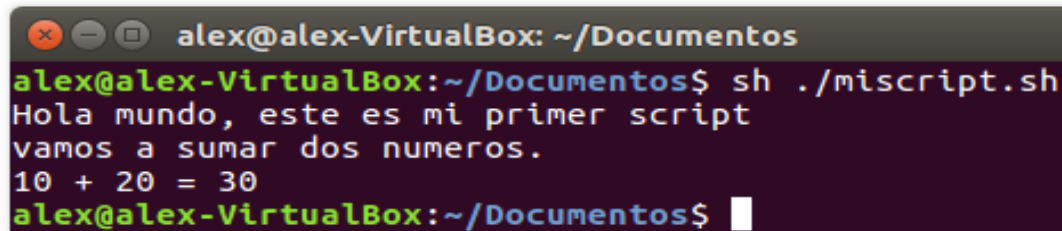
- ✓ Este es un lenguaje NO tipado, es decir, a las variables no se les coloca que tipos es (int string etc), basta con declararlas con el nombre y asignarle el valor. Por ejemplo: “`Nombre_variable=valor`”.
- ✓ La Shell reconoce los espacios en la declaración de variables, por lo tanto, NO se debe dejar espacio entre el nombre el signo igual (=) y el valor.
- ✓ Si a la variable se le pasa como valor una cadena que lleva espacio se debe colocar entre comillas dobles “”.
- ✓ Para hacer el llamado de una variable se le debe anteponer el signo \$
- ✓ Para concatenar variables se debe colocar \$variable1 espacio \$variable2.

En la imagen siguiente se puede observar cada uno de los ítems mencionados.

```

1 #! bin/bash
2
3 saludo="Hola mundo, este es mi primer script"
4 echo $saludo
5
6 echo vamos a sumar dos numeros.
7
8 numero1=10
9 numero2=20
10 suma=$((numero1+numero2))
11
12 echo $numero1 + $numero2 = $suma

```



```

alex@alex-VirtualBox: ~/Documentos
alex@alex-VirtualBox:~/Documentos$ sh ./miscrypt.sh
Hola mundo, este es mi primer script
vamos a sumar dos numeros.
10 + 20 = 30
alex@alex-VirtualBox:~/Documentos$

```

Fuente Propia (2018). Sintaxis de Variables [Figura 3].

Para recibir el valor de una variable por teclado se usa el comando “**read**”, y al comando echo se le agrega el atributo -p, por ejemplo.

```

echo -p "Digite valor de la variable"
read valor

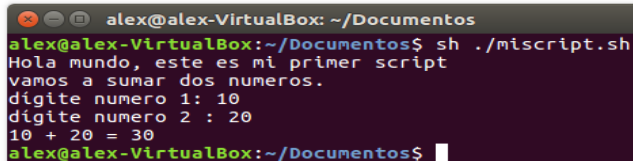
```

Como se puede observar en la imagen.

```

1 #! bin/bash
2
3 saludo="Hola mundo, este es mi primer script"
4 echo $saludo
5 echo vamos a sumar dos numeros.
6 echo -n "digite numero 1: "
7 read numero1
8 echo -n "digite numero 2 : "
9 read numero2
10 suma=$((numero1 + numero2))
11
12 echo $numero1 + $numero2 = $suma

```



```

alex@alex-VirtualBox: ~/Documentos
alex@alex-VirtualBox:~/Documentos$ sh ./miscrypt.sh
Hola mundo, este es mi primer script
vamos a sumar dos numeros.
digite numero 1: 10
digite numero 2 : 20
10 + 20 = 30
alex@alex-VirtualBox:~/Documentos$

```

Fuente Propia (2018). Sintaxis de Variables [Figura 4].

Existen diferentes clases de variables que se clasifican según su localización:

Variables locales:

- Presentes en la instancia actual del Shell
- No están disponible para programas iniciados des el Shell
-

Variables de entorno:

- Disponible para todo proceso hijo del Shell.
- Son muy útiles para la estructura de Script.
- Pueden visualizarse mediante el comando “**env**”
- Se pueden agregar mediante el comando “**export**” (export Nombre_variable=Valor).
- Por convención los nombres de estas variables se colocan en mayúscula.

Variables de Shell:

- Establecidas y utilizadas por el Shell para su funcionamiento.
- Algunas son variables de entorno otras son locales.
- Pueden visualizarse mediante el comando “**set**”.
- Por convención los nombres de estas variables se colocan en mayúscula.

1.3 Manejo de argumentos (parámetros)

Los parámetros son valores que se le pasan a un Script desde la línea de comando cuando este se ejecuta, para que el Shell reconozca que es un parámetro que se ha pasado se coloca el símbolo \$, (\$Numero_parametro), los parámetros normalmente van del \$1 al \$9. Existen tres parámetros especiales que son:

\$*: Devuelve todos los parámetros separados por espacio.

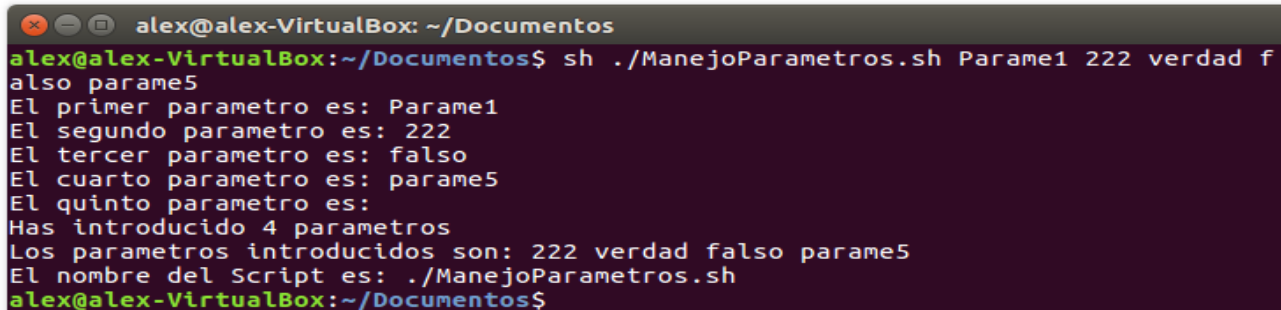
\$#: Devuelve el número de parámetros que se han pasado.

\$0: Devuelve parámetro 0, es decir, el nombre del Script o la función.

El comando **shift** es usado en el manejo de parámetros, ya que este permite eliminar el primer parámetro de la lista y corre a los otros una posición así atrás; es decir; elimina \$1 y el parámetro \$2 pasa a ser \$1.

En la imagen se ve que el tercer parámetro (verdad) no se muestra porque se puso el comando shift y el quinto parámetro pasa a la cuarta posición.

```
1 #! bin/bash
2 echo "El primer parametro es: $1"
3 echo "El segundo parametro es: $2"
4 shift
5 echo "El tercer parametro es: $3"
6 echo "El cuarto parametro es: $4"
7 echo "El quinto parametro es: $5"
8 echo "Has introducido $# parametros"
9 echo "Los parametros introducidos son: $*"
10 echo "El nombre del Script es: $0"
```



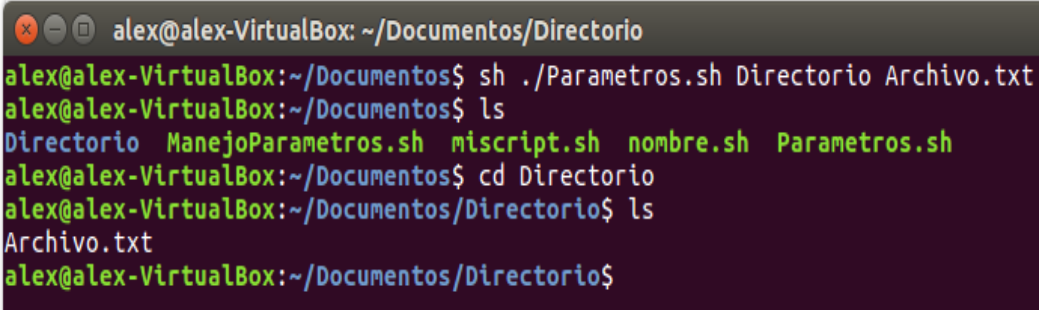
```
alex@alex-VirtualBox: ~/Documentos
alex@alex-VirtualBox:~/Documentos$ sh ./ManejoParametros.sh Parame1 222 verdad f
also param5
El primer parametro es: Parame1
El segundo parametro es: 222
El tercer parametro es: falso
El cuarto parametro es: param5
El quinto parametro es:
Has introducido 4 parametros
Los parametros introducidos son: 222 verdad falso param5
El nombre del Script es: ./ManejoParametros.sh
alex@alex-VirtualBox:~/Documentos$
```

Fuente Propia (2018). Uso de parámetros [Figura 5].

Una aplicación del uso de parámetros es el siguiente:

Ejemplo: *Crear un Script que cree un directorio y dentro de este cree un archivo.txt*

```
1 #! bin/bash
2 mkdir $1 # Crea el directorio
3 :>$2     # Crea el archivo.txt
4 mv $2 $1 # Mueve el Archivo.txt al directorio
```



Fuente Propia (2018). Uso de parámetros [Figura 6].

1.4 Operaciones de comparación.

Las operaciones de comparación son muy útiles en la programación de Script, ya que estas permiten comparar cadenas, números y ficheros; estas operaciones también son conocidas como expresiones “test”. Las expresiones test para facilitar su uso se agrupan según la funcionalidad que brinda cada comando, como se muestra en la tabla 1.

Significado de cada parámetro

EXPRESIONES TEST MÁS COMUNES		
Comando	Sintaxis	Función
Comparación de cadenas		
=	"cad1"="cad2"	Comprueba si las cadenas son iguales
-Z	-Z cadena	Comprueba si la cadena esta vacia
!=	"cad1"!="cad2"	Comprueba si la cad1 es diferente de cad2.
Comprobaciones de expresiones numéricas		
-eq	N1 -eq N2	Comprueba si N1 = N2
-ge	N1 -ge N2	Comprueba si N1 >= N2
-ne	N1 -ne N2	Comprueba si N1 es distinto de N2
-gt	N1 -gt N2	Comprueba si N1 > N2
-le	N1 -le N2	Comprueba si N1 <= N2
-lt	N1 -lt N2	Comprueba si N1 < N2
Comprobación de directorios		
-f	-f/ruta/nombreDir	Comprueba si es un fichero normal
-l	-l/ruta/nombreDir	Comprueba si es un enlace suave
-d o -a	-d/ruta	Comprueba que el directorio exista
-x	-x/ruta/nombreDir	Comprueba si es un ejecutable
-r	-r/ruta/nombreDir	Comprueba si el directorio existe y se puede leer
-w	-r/ruta/nombreDir	Comprueba si el directorio existe y se puede escribir
-u	-u/ruta/nombreDir	Comprueba si tiene activados los permisos suid
-g	-g/ruta/nombreDir	Comprueba si tiene activados los permisos sgid
-s	-s/ruta/nombre	Comprueba que su tamaño es mayo que cero.
Comprobaciones de fecha		
-N	-N/ruta/nombreDir	Comprueba la última modificación del directorio
-nt	Dir1 -nt Dir2	Dir1 es más nuevo que Dir2
-ot	Dir1 -ot Dir2	Dir1 es más antiguo que Dir2
Concatenar expresiones a evaluar		
-o = OR	Exp1 -o Exp2	
-a = AND	Exp1 -a Exp2	
!= NOT	Exp1! Exp2	

Fuente Propia (2018). Expresiones test [Tabla 1].

1.5 Manejo del condicional (if)

En un Script el **if** se utiliza al igual que en la mayoría de los lenguajes de programación, esta sentencia lo que hace es que si se cumple la condición se ejecuta un bloque de código, en caso que esta no se cumpla se ejecuta otro bloque de código, su sintaxis en un Script es:

```
if [ condición ]; then
    Sentencias
else
    Sentencias
fi.
```

```
if [ condición ]
then
    Sentencias
else
    Sentencias
fi.
```

Se debe tener en cuenta el espacio que hay entre los corchetes [] y la condición. Este condicional también se puede anidar, para ello su sintaxis es:

```
if [ condición ]; then
    Sentencias
else
    if [ condición ];
    then
        Sentencia
    else
        sentencia
    fi
fi.
```

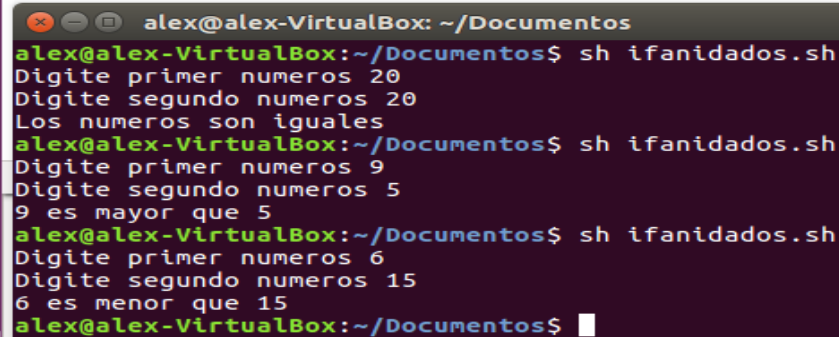
Cuando se usan expresiones **test** en un condicional se coloca *if test condición*. Para entender mejor el uso del condicional if se analizará el siguiente ejemplo.

Ejemplo: crear un Script que pida al usuario dos números y me diga cual es el mayor o si son iguales.

```
1 #! bin/bash
2 read -p "Digite primer numeros " n1
3 read -p "Digite segundo numeros " n2
4 if [ $n1 = $n2 ]; then
5     echo "Los numeros son iguales"
6 else
7     if test $n1 -gt $n2; then
8         echo "$n1 es mayor que $n2"
9     else
10        echo "$n1 es menor que $n2"
11    fi
12 fi
```

En la línea 4 se observa la sintaxis del **if** para validar la igualdad de los dos números.

En la línea 7 se usó la expresión **test -gt** que valida si n1 es mayor que n2



```
alex@alex-VirtualBox: ~/Documentos
alex@alex-VirtualBox:~/Documentos$ sh ifanidados.sh
Digite primer numeros 20
Digite segundo numeros 20
Los numeros son iguales
alex@alex-VirtualBox:~/Documentos$ sh ifanidados.sh
Digite primer numeros 9
Digite segundo numeros 5
9 es mayor que 5
alex@alex-VirtualBox:~/Documentos$ sh ifanidados.sh
Digite primer numeros 6
Digite segundo numeros 15
6 es menor que 15
alex@alex-VirtualBox:~/Documentos$
```

Ejemplo: Hacer un Script que compruebe si un archivo.txt existe y me diga si este es regular, en caso de que no exista debe de crearlo.

1.6 Uso del Case

Case hace la misma función que el **if**, pero de distinta forma.

Al igual que el **if**, que se cierra con **fi**, el **case** se cierra con **esac**.

Esta estructura nos permite ejecutar varias acciones, dependiendo del valor de una variable o expresión.

Su sintaxis es la siguiente.

```
case $variable in
    condicion1)
        instrucciones;;
    condicion2)
        instrucciones;;
        .
        .
        .
    condicionn)
        instrucciones;;
esac
```

Ejemplo.

Hacer un Script que diga que carácter se digito por teclado.

```
1 #!/bin/bash
2 echo "Digite un caracter:\c"
3 read var
4 case $var in
5 [a-z])
6     echo "Digitates una letra minúscula";;
7 [A-Z])
8     echo "Digitates una letra mayúscula";;
9 [0-9])
10    echo "Digtates un número entero";;
11
12 ?)
13    echo "Digitates un caracter especial";;
14
15 *)
16    echo "Digitates más de un caracter";;
17 esac
```

El resultado del Script es el que se muestra en la siguiente imagen.

```

alex@alex-VirtualBox:~/Documentos$ chmod 777 ejemplo_case.sh
alex@alex-VirtualBox:~/Documentos$
alex@alex-VirtualBox:~/Documentos$ sh ejemplo_case.sh
Digite un caracter:d
Digitates una letra minúscula
alex@alex-VirtualBox:~/Documentos$
alex@alex-VirtualBox:~/Documentos$ sh ejemplo_case.sh
Digite un caracter:U
Digitates una letra mayúscula
alex@alex-VirtualBox:~/Documentos$
alex@alex-VirtualBox:~/Documentos$ sh ejemplo_case.sh
Digite un caracter:4
Digitates un número entero
alex@alex-VirtualBox:~/Documentos$
alex@alex-VirtualBox:~/Documentos$ sh ejemplo_case.sh
Digite un caracter:#
Digitates un caracter especial
alex@alex-VirtualBox:~/Documentos$
alex@alex-VirtualBox:~/Documentos$ sh ejemplo_case.sh
Digite un caracter:linux
Digitates más de un caracter
alex@alex-VirtualBox:~/Documentos$

```

1.7 Bucles (for y while)

1.7.1 Sintaxis y usos del for.

El **for** es una estructura iterativa que se ejecuta hasta que se deje de cumplir la condición para seguir y el incremento depende de la expresión de paso que se coloque. El incremento normalmente es de uno, pero se puede cambiar a que se incremente de 2 en 2, de 3 en 3, ect, dependiendo del caso particular para resolver cada problema.

La sintaxis del for es la siguiente:

```

for ((var_inicio; condición_para_seguir; expresión_de_paso))
Do
    instrucciones
done

```

Otra forma de escribir el ciclo for y la más usada para hacer Script es la siguiente.

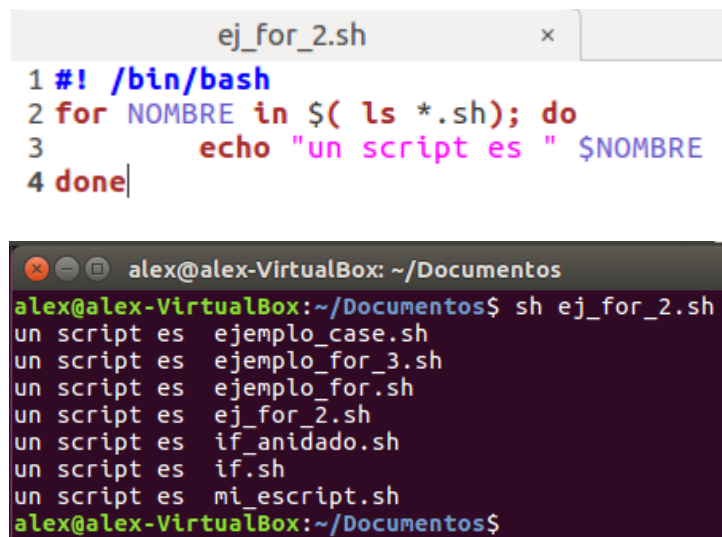
```

for variable in conjunto;
do
    instrucciones

```

Las líneas se repiten una vez por cada elemento del conjunto, y variable va tomando valores del conjunto uno por uno.

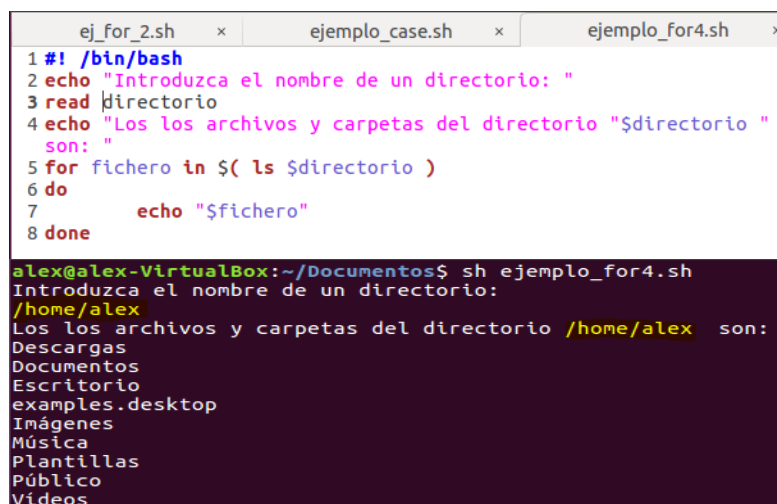
Ejemplo. Crear un Script que liste todos los archivos **.sh** que están el directorio desde donde estoy ejecutando el Script.



```
ej_for_2.sh x
1 #! /bin/bash
2 for NOMBRE in $( ls *.sh ); do
3     echo "un script es " $NOMBRE
4 done

alex@alex-VirtualBox: ~/Documentos
alex@alex-VirtualBox:~/Documentos$ sh ej_for_2.sh
un script es  ejemplo_case.sh
un script es  ejemplo_for_3.sh
un script es  ejemplo_for.sh
un script es  ej_for_2.sh
un script es  if_anidado.sh
un script es  if.sh
un script es  mi_escript.sh
alex@alex-VirtualBox:~/Documentos$
```

Ejemplo. Crear un script que liste todos los archivos de un directorio ingresado por teclado.



```
ej_for_2.sh x  ejemplo_case.sh x  ejemplo_for4.sh x
1 #! /bin/bash
2 echo "Introduzca el nombre de un directorio: "
3 read directorio
4 echo "Los los archivos y carpetas del directorio "$directorio "
   son: "
5 for fichero in $( ls $directorio )
6 do
7     echo "$fichero"
8 done

alex@alex-VirtualBox:~/Documentos$ sh ejemplo_for4.sh
Introduzca el nombre de un directorio:
/home/alex
Los los archivos y carpetas del directorio /home/alex son:
Descargas
Documentos
Escritorio
examples.desktop
Imágenes
Música
Plantillas
Público
Vídeos
```

1.3.2 Sintaxis y usos del while.

Cuando no queremos recorrer un conjunto de valores, sino repetir algo mientras se cumpla una condición, o hasta que se cumpla una condición, podemos usar las estructuras while. La sintaxis es la siguiente:

```
while [expresión]; do  
    instrucciones  
done
```

El ciclo se ejecuta mientras que la condición sea verdadera y se repiten las instrucciones un número de veces no preestablecidos.

El while se usa para repetir un conjunto de instrucciones dependiendo de si se cumple o no la condición. Las condiciones que se ponen en el while son expresiones test o una variable definida antes del ciclo.

El **break** se usa solo si se quiere salir bruscamente del ciclo.

Ejemplo. Hacer un Script que muestre los números del 1 al 10.

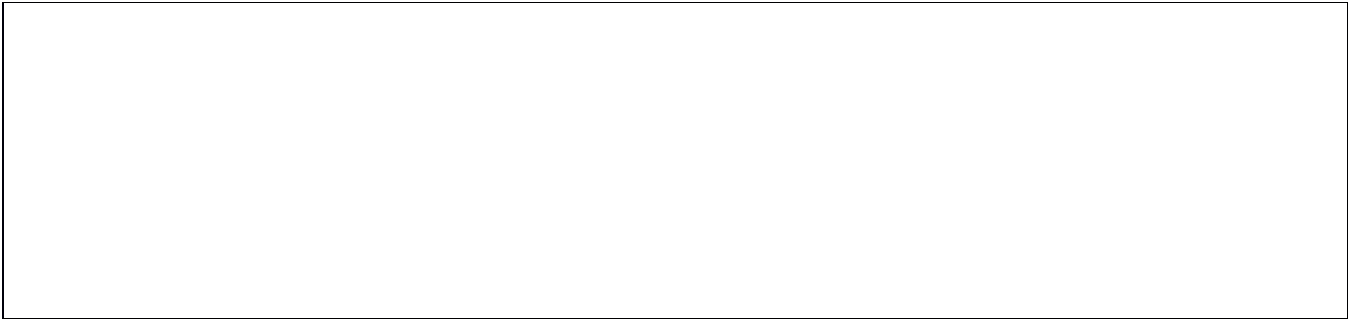
1.4 Creación de Shell Script avanzado.

Funciones

Usar funciones en un script es muy simple. Basta con usar la siguiente estructura al principio del script.

```
Function nombre_funcion{  
    instrucciones  
}
```

Las instrucciones de la función no se ejecutaran al procesar el Shell script, sino que solo se ejecutarán cuando en el cuerpo del script se use el nombre de la función.



EJERCICIOS/ACTIVIDADES

Actividad en clase 1.

Realizar un taller utilizando la consola y un editor de texto donde se evidencia la aplicación del manejo de variables y el condicional if.

Metodología

Cada estudiante va a realizar un taller donde documente varios ejemplos prácticos del uso de variables y el condicional if y el case.

Actividad en clase 2.

Realizar un taller utilizando la consola y un editor de texto para colocar en practica el uso de las sentencias repetitivas y funciones.

Metodología.

En grupos de dos estudiantes se realizará donde documenten varios ejemplos del uso de las sentencias repetitivas y de las funciones.

BIBLIOGRAFIA

- Isaac Paneque. Linux 4 you. Ed 2013 (Pág 352). Recuperado de: <https://books.google.com.co/books?id=jSEXTiZqvYC&pg=PA343&dq=programacion+de+shell+scripts+en+linux&hl=es-419&sa=X&ved=0ahUKEwiOgsWq5JbbAhVMzFMKHcCdCciQ6AEITzAI#v=onepage&q=programacion%20de%20shell%20scripts%20en%20linux&f=false>
- Juan Garcés. Programación en Shell Script Linux. (2013). Recuperado de: <http://www.jgarces.info/programacion-de-shell-scripts-linux/>
- Adrián de los Santos. Programación Shell. Recuperado de: <https://www.freeshell.de/~rasoda/programacion/guia-shell.pdf>
- Francisco J. Fernandez & Manuel Muños. Programación Shell-Script en Linux. Recuperado de: <http://trajano.us.es/~fifi/shell/shellscrip.htm>

