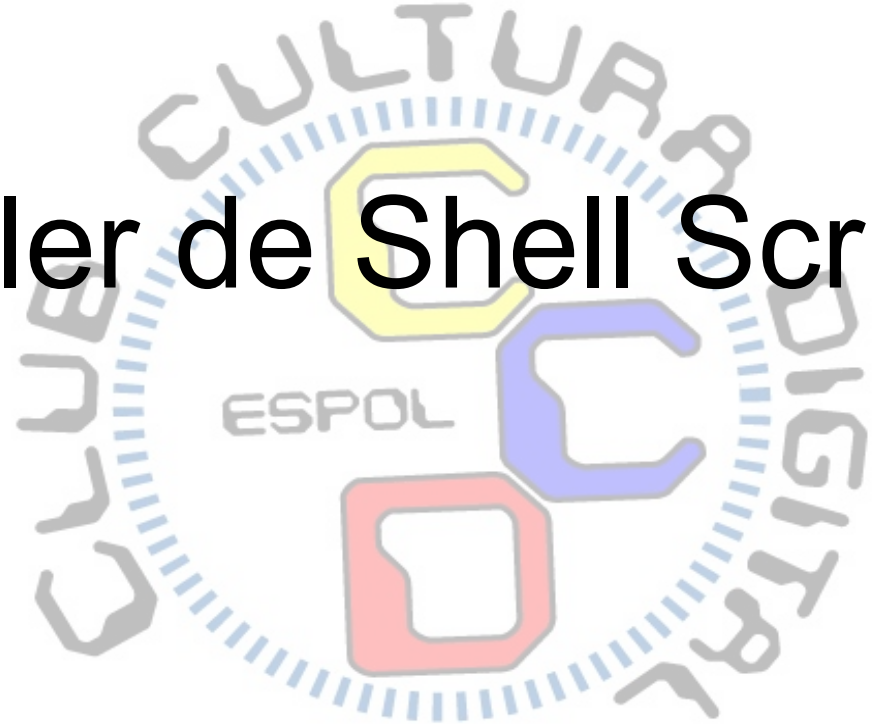
Taller de Shell Script



Leonardo Gavidia Guerra
leogavidia@gmail.com - lgavidia@espol.edu.ec
http://blog.espol.edu.ec/programando

Temas

Manejo básico de la consola

- Mandatos Básicos
- Editando Textos
- Tuberías, ejecuciones
- Mandatos útiles
- Metacaracteres

- **Programando**
- Variables, Operaciones con contenidos de variables
- Priorización de Comandos
- Argumentos
- Operaciones matemáticas
- Evaluación de expresiones
- Sentencias condicionales: if, else, elif
- Sentencias repetitivas: for, while, until.
- Funciones

Manejo Básico de la Consola

Mandatos Básicos

Mostrar el contenido de un directorio
ls *[opciones]* ***<directorio>***
Opciones importantes: -l -a -s -h -R

Desplazamiento por el sistema de archivos
cd ***<rutaObjetivo>***

Copiar archivos/directorios
cp *[opciones]* ***<rutaOrigen> <rutaDestino>***
Opciones importantes: -p -R -v -i -f

Mover archivos/directorios
mv *[opciones]* ***<rutaOrigen> <rutaDestino>***
Opciones importantes: -v -f -i -u

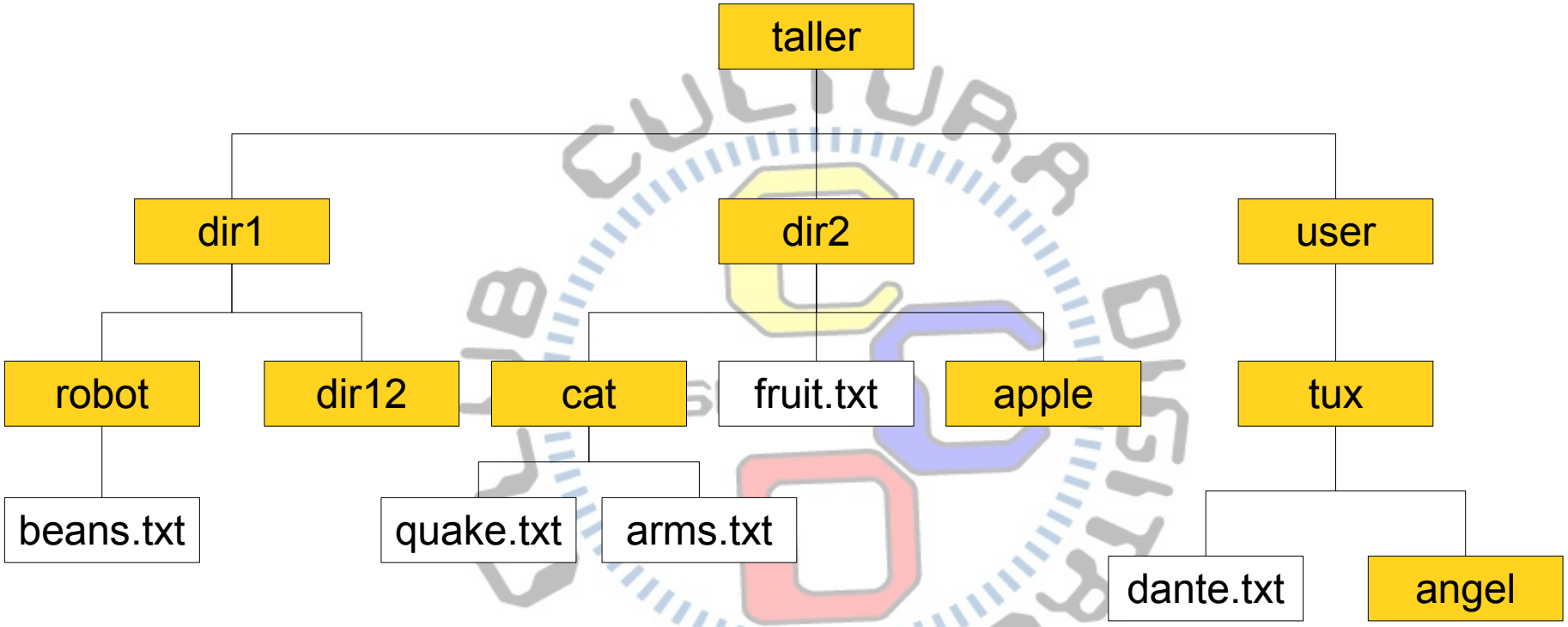
Borrar Archivos
rm *[opciones]* ***<rutaObjetivo>***
Opciones Importantes: -v -r -f -i

Crear Directorios
mkdir *[opciones]* ***<rutaDirectorio>***
Borrar Directorios Vacíos
rmdir *[opciones]* ***<rutaDirectorio>***

Crear Archivos Vacíos
touch <archivo>

Ejercicio:

Crear el siguiente árbol de directorios:



Listar el árbol de directorios
Copiar el archivo fruit.txt en apple/ con el mismo nombre
Copiar el archivo arms.txt en dir12/ con el nombre de file1.txt
Mover el archivo dante.txt a user/
Renombrar el archivo beans.txt y colocarle el nombre spider.txt
Borrar user/
Mover el directorio dir2/ a robot/
Listar el árbol de directorios

Solución:

mkdir taller
cd taller
mkdir dir1 dir2 user
cd dir1
mkdir robot dir12
touch robot/beans.txt
cd ../dir2
mkdir cat apple
touch fruit.txt
cd cat
touch quake.txt arms.txt
cd ../../user
mkdir -p tux/angel
touch tux/dante.txt
cd ../..
ls -R taller
cp taller/dir2/fruit.txt taller/dir2/apple/
cp taller/dir2/cat/arms.txt taller/dir1/dir12/file1.txt
mv taller/user/tux/dante.txt taller/user/dante.txt
mv taller/dir1/robot/beans.txt taller/dir1/robot/spider.txt
rm -rf taller/user
mv -f taller/dir2 taller/dir1/robot
ls -R taller

Editando archivos con vi

vi es un editor de texto muy usado.

Modo de uso:
vi *<ruta del archivo>*

Comandos:

i	Inserta texto antes del carácter sobre el cual se encuentra el cursor (cambia al modo de edición)
yy	Copia una línea de texto
p	Pega una línea de texto
dd	Corta o elimina una línea de texto
o	Inserta una línea en blanco después de la línea en la que se encuentra el cursor (cambia al modo de edición)
r	Reemplaza un carácter (se presiona r y luego se ingresa el carácter deseado)
gg	Desplaza el cursor hacia una línea indicada
ZZ	Guarda los cambios y sale del editor
:w	Guarda los cambios en el archivo
:wq	Es equivalente al comando ZZ
:/	Busca texto en el archivo.
:q	Permite salir del editor si no se han realizados cambios en el archivo.
:q!	Salida de modo forzado, sale del editor sin guardar los cambios realizados al archivo.

Tuberías

Un proceso en un sistema UNIX posee 3 canales abiertos

STDIN (entrada stándar)
STDOUT (salida stándar)
STDERR (salida de error)

Es posible redirigir la salida de un comando a un archivo

ls > salida.txt	Envía STDOUT a salida.txt
ls >> salida.txt	Envía STDOUT al final de salida.txt
ls 2> error.txt	Envía STDERR a error.txt
ls 2>> error.txt	Envía STDERR al final de error.txt
ls > salida.txt 2>&1	Envía STDOUT y STDERR a salida.txt

Podemos enviar el contenido de un archivo a STDIN

comando < archivo.txt Envía el contenido de archivo.txt a STDIN

También podemos dirigir la salida de un comando a la entrada de otro.

comando1 | comando2

Ejecuciones

Ejecuciones Consecutivas
comando1; comando2; comando3
comando2 se ejecutará después de **comando1** y **comando3** se ejecutará después de **comando2**.

Ejecuciones Simultáneas
comando1 & comando2
comando2 se ejecutará al mismo tiempo que **comando1**

Ejecuciones Condicionales
comando1 && comando2
comando2 se ejecutará solo si comando1 finaliza correctamente.

Si se desea enviar un proceso a segundo plano se coloca **&** al final del comando, si se desea pausar un proceso se usa **"Ctrl+z"**, si se desea reanudar este proceso y enviarlo a segundo plano se usa el comando **bg.**

Mostrado Archivos

cat	Concatena un archivo y coloca su contenido en la salida estándar.
tail	Muestra la última parte de un archivo.
more	Muestra el contenido de un archivo una pantalla a la vez.

Buscando archivos

find	Busca archivos en un directorio
whereis	Busca la localización de binarios, fuentes y manuales de un comando.

Filtrando Texto

grep	Muestra líneas que contengan un patrón indicado
tr	Translitera los caracteres de entrada
sed	Filtra y edita texto.
cut	Remueve secciones de texto.

Usuarios y Grupos

id	Muestra la identificación del usuario.
whoami	Muestra el nombre del usuario.
who	Muestra todos los usuarios conectados.

Útiles

date	Muestra la hora y fecha.
cal	Muestra el calendario en pantalla.
sleep	Realiza un proceso durante cierto tiempo
tar	Agrupa/desagrupa ficheros.
man	Muestra el manual de un comando.
pwd	Muestra la ruta del directorio actual.

Metacaracteres:

Son tres:

*	Asterisco
[ ]	Corchetes
?	Signo de Interrogación

*** Representa a un número indeterminado de caracteres, incluso a ningún carácter.**
Ej:
$ ls **.txt*

prueba.txt	texto.txt	file.txt
newfile.txt	aux.txt	austin.txt

[ ] Representa a un único carácter de un conjunto o rango de caracteres.
Ej:
$ ls hol[aeiou012_]***.txt***

hola.txt	holi.txt	hol0.txt
hol_.txt		

Nota: Los rangos pueden definirse de la forma carácterInicial-carácterFinal
Ej: [a-z] [a-zA-Z] [0-9a-z] (minúsculas, minúsculas y mayúsculas, números y minúsculas respectivamente).

? Representa a un único caráter cualquiera que sea.

$ ls ?ro***.txt***

aro .txt	_ro.txt	0ro.txt
Aro.txt	Bro.txt	nro.txt

Ejercicio:

Mostrar sólo la dirección IP de una interfaz de red.
(Usar el comando **ifconfig**)

Solución:
ifconfig *eth0* **| sed** *"**s/[ ][ ]*/:/g**"* **| grep** M*ask* **| cut** *-**f4*** -d**:**

ifconfig eth0	Muestra los detalles de eth0
sed "s/[ ][ ]*/:/g"	Reemplaza los espacios ([ ][ ]*) por **:**
grep Mask	Busca la línea que contenga **Mask** (esta línea también contiene la IP)
cut	Muestra la 4ta columna delimitada por **:**

Usar el comando ifconfig y cambiar las letras *minúsculas* por letras *MAYÚSCULAS*

Solución:
ifconfig | tr a-z A-Z

tr a-z A-Z	Cambia las letras minúsculas a MAYÚSCULAS

Empezando a programar en Shell

Mostrando un mensaje por pantalla
echo “Bienvenidos ”
printf “Hola”

Variables
Las variables no se declaran ni se les especifica un tipo de dato, se crean una vez que se les asigna un valor.
Ejemplos:
A=10
B=”Hola que tal”

Para acceder a los datos contenidos en las variables usamos el operador ***$***
Ejemplo:
A=”Leonardo”
echo “Hola $A!! ¿Como estás?”
Muestra por pantalla: **Hola Leonardo!! ¿Como estás?**

Las cadenas de caracteres se delimitan con ' ' (comillas simples) si se desea que no se interprete nada dentro de la misma (cadenas constantes) y con “”(comillas dobles) si se desea que alguna expresión sea interpretada (cadenas dinámicas). Ejemplo:

A=”noName”
echo “Hola $A”
Muestra por pantalla: **Hola noName**
echo 'Hola $A'
Muestra por pantalla: **Hola $A**

Podemos leer variables con el comando read.
Modo de uso:
read variable

Operaciones con variables

{<variable>:<inicio>:<longitud>}

Extrae una subcadena de **<variable>** empezando desde **<inicio>** y con un tamaño de **<longitud>**

Reemplazar texto en una variable

${<variable>/<texto1>/<texto2>}
Sustituye **<texto1>** por **<texto2>** en **<variable>**. Sólo se reemplaza la primera aparición.
${<variable>//<texto1>/<texto2>}
Ésta forma hace que se sustituyan todas las apariciones de **<texto1>** por **<texto2>** en **<variable>**.

Cortando texto

Al principio de una variable
${<variable>#<texto>}
Corta **<texto>** de **<variable>** si esta empieza por **<texto>,** si **<variable>** no empieza con **<texto>**, **<variable>** no se usará alterada.

Al final de una variable
${<variable>%<texto>}
Corta **<texto>** de **<variable>** si esta termina en **<texto>,** si **<variable>** no termina en **<texto>**, **<variable>** no se usará alterada.

También se puede insertar en una variable la salida de un comando usando ` ` (acento grave).
Los acentos graves ejecutan en el shell todo lo que esté entre ellos.
Ejemplo:
A=`**whoami**`
echo $A
Muestra en pantalla el nombre de usuario (en mi caso **root**)

También se usan para priorizar los comandos.
Ejemplo:
echo “Bienvenido `whoami`”
Muestra en pantalla: **Bienvenido root**
Paréntesis
Todo lo que está entre paréntesis ( ) se ejecuta en una nueva shell y luego ésta “muere”.
Ejemplo:
 # pwd ; **cd** /tmp;**pwd**
/root
/tmp
 # pwd
/tmp

 # (pwd ; **cd** /tmp;**pwd)**
/root
/tmp
 # pwd
/root
Para acceder a los datos retornados por estos comandos se se usa el operador $
Ejemplo:
A=**$(whoami)**

Los programas se guardarán en un archivo con extensión .sh y la primera línea de éste debe contener la ruta de la shell que se usará precedido por #!
Ejemplo de un archivo.sh:
#!/bin/sh
echo "Hola Mundo"

Una vez guardado este archivo debemos añadirle el permiso de ejecución
chmod +x <rutaDelArchivo>
Ej:
chmod +x /root/primerScript.sh

Para ejecutar este script debemos introducir la ruta completa del archivo, si nos encontramos en el directorio que contiene nuestro programa escribimos ./archivo.sh

Es posible pasarle argumentos a nuestro script, de la siguiente forma:
 # rutaScript arg1 arg2 arg3 … argN

Cuando escribimos nuestro programa, los argumentos que pasamos se tratan como variables ordenadas numéricamente de la siguiente forma:
$1 $2 $3 … ${10} ${11} …
arg1 arg2 arg3 … arg10 arg11 ..

El argumento **$0** es el propio script

basename $0	Es el nombre del script
dirname $0	Es la ruta del script
shift	Rota los argumentos hacia la izquierda $1 vale $2, $2 vale $3 , $3 vale $4... sucesivamente.

Argumentos especiales

$#	Número de argumentos que nos han pasado.
$*	Todos los argumentos. “$*” = “$1 $2 $3...”
$@	Todos los argumentos. “$@” = “$1” “$2” “$3”...
$_	Comando anteriormente ejecutado.
$$	PID del propio proceso shell.
$?	Almacena el estado en que terminó el último comando

Operaciones matemáticas

Operadores: suma(**+**), resta(**-**), producto(*****), división(**/**), resto(**%**)

Usamos el comando **let**

```
let A=5+3
echo $A
```

Atención: No hay espacios en la operación.

Muestra en pantalla: 8

Incrementando una variable

```
let A=3
let A=$A+1
```

Agrupando términos

```
let A=\(10+5\)*2
```

Nota: Si deseamos agrupar con paréntesis, éstos deben estar precedidos de una va barra invertida ****

Otra forma:

```
A=`expr 10 + 1`
B=`expr \( 10 + 1 \) \* 2`
```

Atención: **expr** requiere espacios en la operación.

Si queremos multiplicar usamos *****

Evaluación de expresiones

Para evaluar expresiones usamos **test**
Modo de uso: test expresion

Opciones:
-f fichero existe el fichero y es un fichero regular
-r fichero existe el fichero y es de lectura
-w fichero existe el fichero y es de escritura
-x fichero existe el fichero y es ejecutable
-h fichero existe el fichero y es un enlace simbólico.
-d fichero existe el fichero y es un directorio

-z <cadena1>	La longitud de la **<cadena1>** es cero
-n <cadena1>	La longitud de la cadena **<cadena1>** es distinta de cero
<cadena1>=<cadena2>	**<cadena1>** y **<cadena2>** son iguales
<cadena1>*!*=<cadena2>	**<cadena1>** y **<cadena2>** son distintas
<numero1> *-eq* <numero2>	**<numero1>** y **<numero2>** son iguales.
<numero1> *-ne* <numero2>	**<numero1>** y **<numero2>** no son iguales.
<numero1> *-gt* <numero2>	**<numero1>** es estrictamente mayor que **<numero2>**
<numero1> *-ge* <numero2>	**<numero1>** es mayor o igual que **<numero2>**
<numero1> *-lt* <numero2>	**<numero1>** es estrictamente menor que **<numero2>**
<numero1> *-le* <numero2>	**<numero1>** es menor o igual que **<numero2>**

Operadores lógicos
! negación
-a operador AND binario
-o operador OR binario
También se puede evaluar expresiones de la siguiente forma [**expresion**]

Las expresiones evaluadas con **test** o con **[ ]** modifican el valor de **$?**

Sentencia if: Es una estructura condicional. Si la condición definida se cumple, se realizan las acciones especificadas

Modo de uso:

```
if expresión
then
        acciones
fi
```

```
if   expresión
then
     acciones
else
     acciones
fi
```

```
if   expresión
then
        acciones
elif  expresión
        acciones
else
        acciones
fi
```

Ejemplo:

```
if [ 10 -lt 8 ]
then
      echo “10 es menor que 8”
else
      echo “10 no es menor que 8”
fi
```

Ejercicio:

Escriba un programa que solicite el ingreso de un número y que indique si es par o no.
Solución:

```
#!/bin/sh
printf "Ingrese un numero: "
read numero
let A=$numero%2
if [ $A -eq 0 ]
then
        echo "es par"
else
        echo "es impar"
fi
```

Ejercicio:

Escriba un programa que reciba como argumento la ruta de un directorio y si existe que se lo indique al usuario de lo contrario que consulte si desea que éste sea creado.

Solución:

```
#!/bin/sh
if [ -d $1 ]
then
        echo "El directorio existe"
else
        printf "El directorio no existe... Desea crearlo?? (s/n): "
        read opcion
        if [ $opcion='s' ]
        then
                mkdir -p $1
                if [ $? -eq 0 ]
                then
                        echo "Directorio creado con éxito"
                else
                        echo "No se pudo crear el directorio"
                fi
        else
                exit 0
        fi
fi
```

Sentencia case

Es una sentencia de selección, permite seleccionar las acciones a realizarse de acuerdo al valor que tome la variable pasada como parámetro.

Modo de uso:

```
case $VARIABLE in
        valor1)
                comandos
        ;;
        valor2)
                comandos
        ;;
        ...
        valorN)
                comandos
        ;;
        *)
                comandos
        ;;
esac
```

Escriba un programa que solicite un nombre, si este nombre está en la lista de opciones, muestre un saludo en particular para el mismo, si no se encuentra en la lista, le indique al usuario que el nombre no existe.

Solución:

```
#!/bin/sh
printf "Ingrese un nombre: "
read nombre
case $nombre in
        "Leo")
                echo "Hola Leo..."
        ;;
        "Angel")
                echo "Bienvenido Angel"
        ;;
        "Arturo")
                echo "Buenas tardes Arturo"
        ;;
        *)
                echo "El nombre no existe"
        ;;
esac
```

Sentencia Repetitivas

Sentencia repetitiva for:
Esta sentencia repite un conjunto de acciones mientras recorre una lista, asignando el valor de un elemento de la lista a la variable definida.

for variable **in** lista
do
    comandos
done

Ejemplo:

for *nombres* **in** *valor1 valor2 valor3 valor4*
do
    echo “$nombre”
done

Esto muestra en pantalla:
valor1
valor2
valor3
valor4

Ejercicios:

Escriba un programa que dado un archivo que contenga una lista de nombres, muestre por pantalla esta lista de nombres enumerada.

Solución:

```
#!/bin/sh
i=0
archivo=$1
for nombre in `cat $archivo`
do
	echo "$i   $nombre"
	let i=$i+1
done
```

Escriba un programa que dados como parámetros nombres de ciudades, muestre éstas ciudades enumeradas.

Solución:

```
#!/bin/sh
i=0
for nombre in $@
do
	echo "$i   $nombre"
	let i=$i+1
done
```

Sentencia Repetitiva while

Ésta sentencia repite acciones mientras se cumpla una condición
Modo de uso

```
while condicion
do
      acciones
done
```

Ejemplo

```
i=0
while [ $i -lt 100 ]
do
      echo $i
      let i=$i+1
done
```

Este programa muestra la serie del 0 al 100

Ejercicio:
Escriba un programa que solicite el ingreso por teclado de un numero y que finalice el programa hasta que el número ingresado sea impar.

Solución:

```
#/bin/sh
numero=0
while  [ `expr $numero % 2` -eq 0 ]
do
        printf "Ingrese un numero"
        read numero
done
```

Leyendo archivos línea por línea

```
cat archivo | while read variable
do
      acciones
done
```

No solo podemos leer **archivos** línea por línea

```
ls | while read archivo
do
      mv $archivo otro$archivo
done
```

Estas instrucciones renombran todos los archivos contenidos en el directorio actual, anteponiéndole la palabra “otro” al nombre de cada archivo.

Ejercicios:

Escriba un programa que lea un archivo pasado como parámetro y que coloque en un nuevo archivo las líneas pares.

Solución:

```
#!/bin/sh
i=0
cat $1| while read linea
do
        if [ `expr $i % 2` -eq 0 ]
        then
                echo $linea >> nuevo.txt
        fi
        let i=$i+1
Done
```

Escriba un programa que lea un archivo pasado como parámetro y muestre por pantalla todas las líneas que NO empiezan con un **#**

Solución:

```
#!/bin/sh
cat $1| while read linea
do
        if [ ${linea:0:1} != "3" ]
        then
                echo “$linea”
        fi
done
```

Sentencia Repetitiva until

Esta sentencia repite un grupo de acciones hasta que la condición se cumpla
Modo de uso:

```
until condicion
do
      acciones
done
```

Ejemplo

```
#!/bin/sh
n=100
until [ $n -le 0 ]
do
       echo "$n"
       let n=$n-1
done
```

Esto muestra por pantalla los valores que va tomando **n** hasta que este se hace 1, nótese que cuando **n** se hizo **0** el programa terminó.

Ejercicio:

Escriba un programa que consulte cada 6 si un archivo pasado como parámetro existe segundos, si este archivo existe, el programa termina. Si no existe, se solicita al usuario crearlo, si el usuario no lo crea se repite el proceso.

Solución:

```
#!/bin/sh
until [ -f $1 ]
do
        sleep 6
        printf "El archivo no existe... Desea crear el archivo $1? [s/n]: "
        read opcion
        if [ $opcion = "s" ]
        then
                touch $1
        else
                echo "Piénselo 6 segundos más"
        fi
done
```

Funciones

Modularizamos los Scripts agrupando tareas en funciones.
Es NECESARIO que una función sea declarda ANTES de que sea llamada.
Dentro de una función $1, $2, $3, etc. Serán parámetros pasados a la función. Estos parámetros son diferentes a los del script.

Modo de uso

```
function nombreFunción
{
      acciones
}
```

Ejemplo

```
function nombreApellido
{
      echo "Nombre: $1"
      echo "Apellido: $2"
}
```

La llamamos así:
nombreApellido Linus Torvalds
Muestra en pantalla:
Nombre: Linus
Apellido: Torvalds

Ejercicio:

Haga una función que sume todos sus parámetros.

```
function sumaTodo
{
	total=0
	for num in $@
	do
		let total=$total+$num
	done
	echo $total
}
```

También es posible incluir todas las funciones de un archivo en nuestro script.
Para esto usamos source
Modo de Uso:
source archivo

Ejemplo:
Si la función sumaTodo estuviera en miscript.sh

```
#!/bin/sh
source miscript.sh
sumaTodo 10 100 15 25 30
```