BASH

THE BOURNE-AGAIN SHELL

Angel Luis Calvo

ESCUELA IT

En esta sesión:

- Funciones
- Menús
- Comandos sobre servicios

Angel Luis Calvo

Angel Luis Calvo

angelonx@gmail.com

ESCUELA IT

Funciones

Angel Luis Calvo

ESCUELA IT

Funciones

Las funciones son una forma de agrupar secciones de código que se podrán volver a utilizar.

Sintaxis:

La orden function se puede utilizar (antes del nombre), pero está obsoleta.

nombre_función () {
código
}

Ojo. Si se repite el nombre de una función, la segunda anula a la primera.

Para usarlas solo hay que invocar su nombre, las veces que sea.

Funciones

Pueden definirse tantas funciones como se desee. Pero la utilidad que se persigue es no tener que escribir secciones repetidas de código.
Un ejemplo de funciones:

```
ver () {
echo La lista de usuarios: ; cat usuarios.txt
}
listar () {
echo Listado del directorio: ; ls -l
}
ver ; listar
```

Funciones: variables

Las variables, por defecto, son globales a todo el script. Solo cuando se las define con el comando local, se utilizan dentro de la función en la que estén establecidas sin extenderse más allá.

Ejemplo:

```
mi_funcion () {
valor=hola #Esta es global
local valor2=adiós
echo $valor y $valor2
}
mi_funcion # A la salida se verán los dos valores
echo $valor y $valor2 #a la salida solo se verá la global
```


Funciones: variables

Las variables paramétricas ($1, ..,$9, etc.) dentro de una función hacen referencia a la propia función, no al script (salvo $0).

Ejemplo:

```
var=$1
ver () {
echo Para ver ‘$1’ del script, es $var
echo Mis parametros son $1 y $2
}
ver Hola Adiós
```


Ejercicio

Realizaremos un ejemplo de script para crear un comando personalizado de copias mediante parámetros:
-C para copiar con los dos siguientes parámetros: carpeta archivo
-R para copiar como en el caso C, pero el archivo indicando sólo parte del nombre
-A para copiar como en el caso C, pero solo los que tengan fecha de hoy
-H para que se muestre esta cabecera como ayuda

```
#!/bin/bash
#Comando personalizado de copias
# -C para copiar con los dos siguientes parámetros: carpeta archivo
# -R para copiar como en el caso C, pero el archivo indicando solo parte del nombre
# -A para copiar como en el caso C, pero solo los que tengan fecha de hoy
# -H para que se muestre esta cabecera como ayuda
if [ $# -eq 0 ]
   then
```

---------- Continúa en la siguiente----------

BASH
THE BOURNE-AGAIN SHELL

ESCUELA IT

Ejercicio

---------- Viene de la anterior ----------

```
        echo Faltan parámetros
        echo -e "Escriba $0 \e[3m -param Carpeta Archivo \e[0m"
        exit 1 #Terminamos mal
    elif [ "$2" = "" ]
    then
        echo Falta indicar la carpeta
        echo -e "Escriba $0 \e[3m -param Carpeta Archivo \e[0m"
        exit 2
    elif [ "$3" = "" ]
    then
        echo Falta el archivo
        exit 3
fi
```

---------- Continúa en la siguiente----------

ESCUELA IT

Ejercicio

```
-------------------------------- Viene de la anterior ------------------------------------
carpeta="$2" #no quiero que shift afecte a la carpeta
copia () {
  ls -l $1 &> /dev/null
  if [ $? != 0 ]
        then
        echo No hay archivos  $1
        exit 1 # Salimos mal
  if [ “$1” = “-H” ]
        echo -----------AYUDA------------
        head -n 6 $0 | sed '1d'
        exit 0 # Salimos bien
  elif [ ! -d "$2" ]
        then
        ------------------- Continúa en la siguiente----------------------
```

ESCUELA IT

Ejercicio

```
-------------------------------- Viene de la anterior ------------------------------------
  echo La carpeta no existe, la crearemos.
        mkdir "$2"
        echo Creada
fi
echo Copiando archivos en $2
  cp -v $1 "$2"
  #echo Terminado
}

case "$1" in
        -C) while [ $# -ge 3 ]; do
                copia "$3" "$carpeta"
                shift #si se usan comodines hay que reducir parámetros
            done ;;
        ------------------- Continúa en la siguiente----------------------
```

BASH
THE BOURNE-AGAIN SHELL

ESCUELA IT

Ejercicio

```
-------------------------------- Viene de la anterior ------------------------------------
        -R) while [ $# -ge 3 ]; do
                copia "*$3*" "$carpeta"
                shift #si se usan comodines hay que reducir parámetros
            done ;;
        -A) while [ $# -ge 3 ]; do
                recientes=$(find -maxdepth 1 -name "$3" -mtime -1)
                copia "$recientes" "$carpeta"
                shift
            done ;;
        *) echo Error inesperado
           exit 100 ;;
esac
```

ESCUELA IT

Funciones: matrices

- Las matrices (o arrays) son variables que almacenan varios datos a la vez. Se pueden utilizar en cualquier parte del script, y por supuesto, también en las funciones.
- En bash son unidimensionales, se les llama a veces vectores o listas.
- Se pueden declarar con el comando *declare*, pero se puede hacer directamente:

```
mi_matriz=(valor1 valor2 valor3) # la cantidad de valores es ilimitada, deben separarse por espacios.
```

Funciones: matrices

Cuando se hace referencia a una matriz, puede hacerse a parte de sus elementos, teniendo en cuenta que el primero es el cero (estos números se llaman índices de cada elemento). Ojo a la sintaxis.
Ejemplo:

```
mi_array=(primero segundo tercero)
echo ${mi_array[0]} # mostrará por pantalla “primero”
mio=“un dos tres”
array2=($mio)
echo ${array2[2]} #mostrará por pantalla el tercer valor almacenado en la variable (matriz) “mio” (tres)
```

Funciones: matrices

mi_array[5]=seis #añade otro elemento al array, no tiene por qué ser consecutivo
mi_array[2]=otro_valor #se puede modificar uno existente
echo ${mi_array[*]} #muestra todos los elementos (también vale @)
echo ${#mi_array[*]} #muestra la cantidad de elementos
echo ${!mi_array[*]} #muestra los índices usados
my_array=($(*comando*)) #rellena la matriz con un comando, cuidado!

Ejercicio

Crear un script que lea un archivo con nombres de usuario (usuarios.txt) y les vaya creando una cuenta en el sistema, a la vez que les genera una contraseña de dos posibles formas:

- Automáticamente, con una marca de tiempo
- Manualmente

Para seleccionar la manera se utilizará un parámetro, “auto” para el primero, “manual” para el segundo.
Al final realizará un informe con los resultados.
(Deberá ejecutarse con sudo)

```
#!/bin/bash
matriz=($(cat usuarios.txt))
crearusuario () {
if [ "$1" = "auto" ]
  then
       echo Se van a crear usuarios automáticamente
       echo Pulse CTRL-C para cancelar
       for ((i=0; i<${#matriz[*]}; i++)); do
```

---------- Continúa en la siguiente----------

BASH
THE BOURNE-AGAIN SHELL

ESCUELA IT

Ejercicio

---------- Viene de la anterior ----------

```
                echo Se creará el usuario ${matriz[$i]}
                sleep 0.5
                useradd -m ${matriz[$i]}
                local marca=$(date +%N) #nanosegundos para generar contraseñas pseudoaleatorias
                echo Aqui la password automática: ${matriz[$i]}_$marca #Podría haberse usado $RANDOM
                echo "${matriz[$i]}:${matriz[$i]}_$marca" | chpasswd
                mresumen[$i]=${matriz[$i]}_$marca
        done
fi
if [ "$1" = "manual" ]
   then
        echo Se van a crear usuarios manualmente
        echo Pulse CTRL-C para cancelar
        for ((i=0; i<${#matriz[*]}; i++)); do
```

---------- Continúa en la siguiente----------

BASH
THE BOURNE-AGAIN SHELL

ESCUELA IT

Ejercicio

---------- Viene de la anterior ----------

```
                echo se creará el usuario ${matriz[$i]}
                useradd -m ${matriz[$i]}
                read -p "Ponga la contraseña para ${matriz[$i]}: " clave
                echo La clave es: $clave
                echo "${matriz[$i]}:$clave" | chpasswd
                mresumen[$i]=$clave
        done
fi
}
resumen (){
echo ${matriz[*]}  > lista_claves.txt
echo ${mresumen[*]} >> lista_claves.txt
echo Resumen de usuarios:
cat lista_claves.txt | column -t
```

---------- Continúa en la siguiente----------

ESCUELA IT

Ejercicio

```
---------------------------------- Viene de la anterior -----------------------------------
}
if [ $# -eq 0 ]
  then
        echo Faltan parámetros
        echo -e "Escriba $0 \e[3m auto o manual \e[0m"
        exit 1 #Terminamos mal
  elif [ "$1" = "auto" ]
  then
        crearusuario auto
        resumen
        echo Terminado
        exit 0 #Sale bien
  elif [ "$1" = "manual" ]
  then
        ------------------- Continúa en la siguiente----------------------
```

BASH
THE BOURNE-AGAIN SHELL

ESCUELA IT

Ejercicio

---------- Viene de la anterior ----------

```
        crearusuario manual
        resumen
        echo Terminado
        exit 0
    else
        echo Opción no válida
        exit 100
fi
```

ESCUELA IT

Funciones: return

El comando *return* es el equivalente a *exit*, pero con la salvedad de que solo se utiliza en funciones.
Así la función termina con un estado de salida definido (siempre un número entero, entre 0 y 255, como exit)
Ejemplo:

```
mi_funcion () {
valor=20
return $(($valor +1))
}
mi_funcion
echo El valor final es $? #se debe consultar inmediatamente
```

Funciones: cálculos

Para cálculos las funciones pueden adquirir el valor numérico de un comando de sustitución, en vez de usar *return*:
Ejemplo:

```
mi_funcion() {
   var=$((1+2))
}
echo El resultado es: $var #Pero se recomienda usar variables locales.
```

Funciones: cálculos

Usando variables locales:

```
mi_funcion() {
    local mi_var=$((1+2))
    echo salida $mi_var
}
resul=$(mi_funcion)
echo $resul
```

Ejercicio

Haremos un script que cuente los días faltantes hasta fin de año. Cuando se introduzca el parámetro “espera”, preguntará por los días de espera, y nos responderá si con esa espera la fecha queda dentro de este año. Cuando se indique el parámetro “semanas”, nos dirá el número de semanas que queda en este año.

```
#!/bin/bash
#Para indicar si una espera de días queda dentro del año en curso
#O para decir cuántas semanas quedan para terminar el año
#Con un parámetro se indica (espera, o ,semanas) la opción a escoger.
#Por defecto, la primera
dias (){
local dias_hoy=$(date +%j)
let disponible=365-$dias_hoy
}
```


---------- Continúa en la siguiente----------

Ejercicio

```
-------------------------------- Viene de la anterior ------------------------------------
if [ "${1-espera}" = "espera" ] #indicamos una opción por defecto
   then
        read -p "Indique días de espera: " x
        dias
        if [ $x -gt $disponible ]
        then
                echo No entra en este año
                exit
        fi
   echo Queda dentro de este año
fi
if [ "$1" = "semanas" ]
   then
        dias
        ------------------- Continúa en la siguiente----------------------
```

BASH
THE BOURNE-AGAIN SHELL

ESCUELA IT

Ejercicio

---------- Viene de la anterior ----------

```
let semanas=$disponible/7
        echo Quedan $semanas semanas completas
fi
echo FIN
```

ESCUELA IT

Funciones: source

Desde un script se pueden ejecutar otros scripts.

- Se puede ejecutar sin más que invocarlo como se haría en la línea de comandos. Este script se ejecuta en una subshell diferente de la principal.
- Si se utiliza la orden source, se ejecutan en el mismo proceso que el script principal, conservando sus variables.
- . es equivalente (POSIX) a source

Ejercicio

Ejemplo de un script que llama a otros (principal.sh) Hay que crear 1.sh, 2.sh y 3.sh tal cómo se muestra.

principal.sh

```
#/bin/bash
echo valor del PID $$ y del padre $PPID
var=prueba
source 1.sh
./2.sh #Este se ejecuta cómo script externo, sin relación con el principal
source 3.sh
```


1.sh

```
echo Primer script
echo valor del PID $$ y del padre $PPID
echo la variable var es $var
```

---------- Continúa en la siguiente----------

Ejercicio

---------- Viene de la anterior ----------

2.sh

```
#/bin/bash
echo Segundo script
echo valor del PID $$ y del padre $PPID
echo la variable var es $var
```

3.sh

```
echo Tercer script
echo valor del PID $$ y del padre $PPID
echo la variable var es $var
```

ESCUELA IT

Funciones: source

Con la orden source, no se necesita ni shebang ni que los llamados sean archivos ejecutables.
Se puede hacer una biblioteca con funciones en archivos de texto, para ser llamadas por diferentes scripts.
Ejemplo: Se guarda como archivo de texto tal cual, por ejemplo copiar.sh

```
copias (){
cp -v $1 $2
}
```

En otro script se le hace la llamada *source copiar.sh*, luego se ejecuta la función donde se quiera.

Ejercicio

Haremos un script que cuente las líneas que contienen dos ficheros (indicados como parámetros) y nos dé la suma de ambos. En caso de no poner parámetros calculará la media del tamaño de los archivos de la carpeta actual. Se usará una función externa, almacenada en el archivo calcular.sh, que sirva para efectuar sumas, restas, multiplicaciones y divisiones.

calcular.sh

```
#no es necesario shebang ni que sea ejecutable
calculo (){
suma=$(echo "scale=2; $1+$2" | bc)
resta=$(echo "scale=2; $1-$2" | bc)
producto=$(echo "scale=2; $1*$2" | bc)
division=$(echo "scale=2; $1/$2" | bc)
}
```

cuenta_lineas.sh

---------- Continúa en la siguiente----------

Ejercicio

---------- Viene de la anterior ----------

```
#!/bin/bash
#Cuenta las líneas de los dos archivos indicados como parámetros y las suma
# O indica la media de tamaño que consumen los archivos de la carpeta actual
source calcular.sh
if [ $# -eq 0 ]
   then
        tamano=$(du -s | cut -f 1) #cut delimita por tabulador por defecto
        n_archivos=$(ls -l | wc -l)
        calculo $tamano $n_archivos
        echo la media es $division bytes
        echo Finalizado
        exit 0
fi
```

---------- Continúa en la siguiente----------

ESCUELA IT

Ejercicio

---------- Viene de la anterior ----------

```
primero=$(wc "$1" -l | cut -d " " -f 1)
segundo=$(wc "$2" -l | cut -d " " -f 1)
calculo $primero $segundo
echo la suma es $suma
echo Finalizado
```

ESCUELA IT

Menús

Angel Luis Calvo

ESCUELA IT

Menús

Con los comandos condicionales vistos se pueden realizar menús sencillos, elaborados con *echo* por ejemplo, para darle algo de vistosidad (Quizá el más adecuado sea *case*)
De lo que se trata es de mostrar al usuario una interfaz fácil e intuitiva a la hora de tomar diferentes opciones.
Combinando esto con funciones se consigue cierta versatilidad.

Menús: select

El comando select está especialmente pensado para ofrecer menús al usuario.

Sintaxis:

```
select nombre in [lista]
do
  comandos
done
```

Ejemplo:

```
select algo in primero segundo
do
echo Seleccionaste $algo
done
```

En el ejemplo, para salir habrá que pulsar CTRL-C

Menús: select

Consideraciones:

- Es necesario prever una salida al menú, por ejemplo *break* o *exit*.
- La variable PS3 controla la solicitud de entrada.
- Se puede combinar muy bien con *if* y sobre todo con *case*.
- La variable REPLAY almacena la respuesta.

Ejercicio

Por medio de un menú y usando funciones, se desea que un script pueda dar opciones de copiar, y borrar dos archivos, así como renombrar uno, solicitando al usuario 2 nombres para realizar esas opciones.

```
#!/bin/bash
clear
ayuda (){
echo Las opciones piden dos nombres de archivo,
echo pero para copiar o borrar se pueden usar comodines.
}
leer (){
read -p "Nombre archivo 1: " nombre1
read -p "Nombre archivo 2: " nombre2
}
PS3="Hola $USERNAME escoge: "
select algo in copiar borrar renombrar ayuda salir
```


---------- Continúa en la siguiente----------

Ejercicio

---------- Viene de la anterior ----------

```
do
   case $algo in
        copiar)
                echo Indica el archivo a copiar y su destino:
                leer
                cp -v $nombre1 $nombre2;;
        borrar)
                echo Indica hasta dos archivos a eliminar:
                leer
                rm -v $nombre1 $nombre2;;
        renombrar)
                echo Indica el archivo a renombrar, y el nuevo nombre:
                leer
                mv $nombre1 $nombre2;;
```

---------- Continúa en la siguiente----------

BASH
THE BOURNE-AGAIN SHELL

ESCUELA IT

Ejercicio

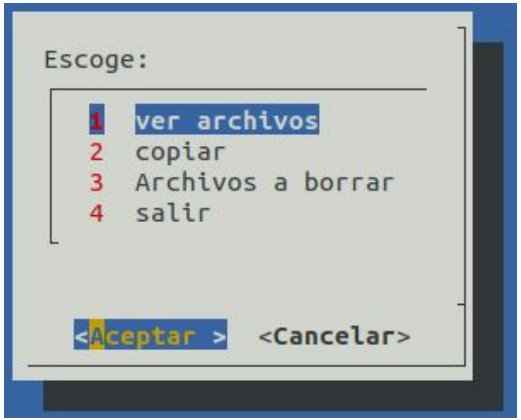
---------- Viene de la anterior ----------

```
        ayuda)
                ayuda;;
        salir)
                echo Saliendo
                break;;
        *)
                echo La opción $REPLY no es válida
  esac
done
```

ESCUELA IT

Menús: dialog

Para hacer menús más visuales podemos apoyarnos en *dialog*, una herramienta añadida para construir toda clase de cuadros de diálogo, con menús y checklist, entre otras cosas.



Menús: dialog

Para actuar es común que tenga que combinarse con condicionales *if* y *case*, o incluso con bucles.
(Hay otros programas alternativos, como Whiptail)

Se instala con apt install dialog.

Para más info:
https://aplicacionesysistemas.com/dialog-crear-menus-tus-scripts/

Ejercicio

Ejemplo de un menú de selección para indicar si se quiere ver dos archivos, copiarlos, o borrar alguno de tres archivos concretos.

```
#!/bin/bash
leer (){
read -p "Nombre archivo 1: " nombre1
read -p "Nombre archivo 2: " nombre2
}

#caja de menú, la salida se almacena en la salida de errores stderr(2)
# Los números indican tamaños de ventana, número y orden de opciones
dialog --menu "Escoge:" 12 30 4 1 "ver archivos" 2 "copiar" 3 "Archivos a borrar" 4 "salir" 2>/tmp/output.tmp
#almacenamos el resultado anterior
salida=$(cat /tmp/output.tmp)
rm -f /tmp/output.tmp
if [ $salida -eq 1 ]; then
```


---------- Continúa en la siguiente----------

Ejercicio

---------- Viene de la anterior ----------

```
        leer
        cat $nombre1 $nombre2
  elif [ $salida -eq 2 ]; then
        leer
        cp -v $nombre1 $nombre2
  elif [ $salida -eq 3 ]; then
        dialog --checklist "Marca los archivos a borrar" 10 40 3 \
        1 usuarios_cesados on \
        2 usuarios_nuevos off \
        3 res.log off  2> /tmp/output2.tmp
        #Las opciones on estárán marcadas por defecto
        archivo=$(cat /tmp/output2.tmp)
        rm -f /tmp/output2.tmp
```

---------- Continúa en la siguiente----------

ESCUELA IT

Ejercicio

```
-------------------------------- Viene de la anterior ------------------------------------
                        for archivos in $archivo; do
                                case $archivos in
                                        1) rm -v usuarios_cesados;;
                                        2) rm -v usuarios_nuevos;;
                                        3) rm -v res.log;;
                                esac
                        done
        elif [ $salida -eq 4 ]; then
                clear
                echo Adiós
        exit
fi
```

ESCUELA IT

Comandos sobre servicios

Angel Luis Calvo

ESCUELA IT

Comandos sobre servicios

Por línea de comandos se pueden ejercer acciones sobre sobre toda clase de servicios.
Depende de la naturaleza de cada uno los comandos son muy diversos.
Por ejemplo, para conectarse a un servidor empleando SSH:

ssh *usuario@host [comando]*

Se necesitará cuenta en el servidor remoto y solicitará contraseña.
Normalmente, con esto accedemos y operamos desde el servidor.

Comandos sobre servicios

Si se desea emplear el comando ssh dentro de un script, una manera sería mediante la autenticación SSH sin contraseña, usando claves públicas. Así se evita el tener que meter la contraseña cada vez.
Otra opción es usar *sshpass* (se instala aparte)

Para más info:
https://www.neoguias.com/utilizar-ssh-shell-de-unix-linux/

Comandos sobre servicios: mysql

Mientras el servidor de base de datos tenga configurado el acceso remoto, podremos acceder a mysql con los comandos de shell.
mysql --host=*servidor* --user=*usuario* --password=*password*

Ya dentro se pueden ejecutar sentencias de mysql: selecciones diversas, inserción de registros, creación de tablas, etc.
Ejemplo:
mysql -u root -pClave
> show databases;

Más info:
https://desarrolloweb.com/articulos/2408.php

Ejercicio

Script para ver las tablas de una base de datos mysql. Hace falta tener instalado *mysql-client*.

```
#!/bin/bash
#Script que muestra las tablas de una base de mysql
#Primero se generan variables para usar en la sentencia de mysql, bash no las conoce!!!
host="localhost"
usuario="root"
clave="Clave"
database="fop2"
#Juntamos esas variables como parámetros del comando mysql,
#con -s para modo silencioso y -e para que salga de mysql al terminar
mysql_param="-h $host -u $usuario -p$clave -D $database -s -e"
#Ejecución de mysql
mysql $mysql_param "show tables;"
```


Comandos sobre servicios: samba

Con el servicio samba se comparten archivos.
Para gestionarlo se crean usuarios específicos para samba, con sus permisos, para ello hay sus propios comandos.
Cuando tenemos instalado samba, estos comandos pueden ser interpretados por bash.
En el servidor, a los usuarios del sistema se les deshabilita el login para usar samba (usermod -L)
smbpasswd → con esto añadimos el usuario a los usuarios de samba y ponemos clave

Comandos sobre servicios

Para la gestión de servicios existe el comando antiguo, *service*, y el más moderno basado en SystemD, *systemctl*.
Sintaxis:
service [start, restart, stop, status] *nombre*

systemctl *nombre* [start, restart, stop, status, disable, enable]

Ejercicio

Realizar un script para crear usuarios en el servidor samba, con cuadros de diálogo, añadiéndolos a los grupos que se deseen. (Habrá que ejecutar con sudo)

```
#!/bin/bash
#Por parámetro se introduce el nombre del usuario a crear
useradd $1
usermod -L $1
clear
echo "Escriba la clave del usuario $1"
smbpasswd -L -a $1
smbpasswd -L -e $1
#Para que tenga efecto debemos reiniciar el servicio
systemctl restart smbd
#Hacemos matrices para almacenar la secuencia del comando y las opciones del mismo
dial=(dialog --separate-output --checklist "Selecciona los grupos a los que pertenece:" 22 76 16)
```


---------- Continúa en la siguiente----------

Ejercicio

```
-------------------------------- Viene de la anterior ------------------------------------
opciones=(1 "sala" on 2 "direccion" off  3 "ventas" off  4 "compras" off)
seleccion=$("${dial[@]}" "${opciones[@]}" 2>&1 >/dev/tty)
clear
for sel in $seleccion
do
        case $sel in
                1) adduser $1 sala;; #Este comando añade el usuario a un grupo
                2) adduser $1 direccion;;
                3) adduser $1 ventas;;
                4) adduser $1 compras;;
        esac
done
clear
echo $1 pertenece a los grupos: ; groups $1
```

ESCUELA IT

Comandos sobre servicios

Para comprobar los servicios instalados y su estado general tenemos el comando:
systemctl list-unit-files

Y para ver los procesos en ejecución ya usamos *ps*.
Con ps aux vemos todos los procesos de todos los usuarios

Ejercicios

1.- Crear un script que lea un archivo con nombres de usuario (usuarios_datos.txt) y que vaya solicitando cubrir sus datos personales (teléfono y edad)
Esos datos se mostrarán por pantalla y se almacenarán en un archivo. 30_

2.- Script ver_archivo.sh, que dependiendo del nombre del usuario, muestre el contenido de un determinado archivo u otro. 31_

3.- Crear una función para mostrar mensajes por pantalla en diferentes formatos, que acepte dos parámetros, el primero indicando el mensaje a mostrar, el segundo que codifique el formato a usar (cursiva, color rojo, negrita y todos a la vez), dicha función podrá ser usada por cualquier script. 32_

ESCUELA IT

1.-

```
#!/bin/bash
clear
for user in $(cat usuarios_datos.txt)
do
        read -p "Indique tlfno. y edad de $user: " tfno edad
        echo "$user: $tfno, $edad" >> usuarios_completado.txt
done
echo Los datos almacenados son:
echo
cat usuarios_completado.txt
echo FIN de ARCHIVO
```

2.-

```
#!/bin/bash
#angel y el jefe pueden ver el archivo más restringido
#pepe y nacho otros menos restringidos, el resto no accede a nada
case $USERNAME in
        pepe)
                cat restringido.txt;;
        angel | jefe)
                cat muy_restringido.txt;;
        nacho)
                cat poco_restringido.txt;;
        *)
                echo Usted no puede ver nada;;
esac
```

ESCUELA IT

Ejercicios

3.-

mensaje.sh

```
#Función de uso externo para que los scripts puedan mostrar mensajes
# en diferentes formatos
formatexto (){
mensaje=
local mensaje=$1
formato=
local formato=$2
case "$formato" in
        cursiva)
                echo -e "\e[3m $mensaje \e[0m";;
        rojo)
                echo -e "\e[91m $mensaje \e[0m";;
```

---------- Continúa en la siguiente----------

ESCUELA IT

Ejercicios

---------- Viene de la anterior ----------

```
        negrita)
                echo -e "\e[1m $mensaje \e[0m";;
        todos)
                echo -e "\e[1;3;91m $mensaje \e[0m";;
esac
}
#Sería bonito con dialog
```

prueba_mensaje.sh

```
#!/bin/bash
#Para probar con la función de mensaje
source mensaje.sh
echo Quiero ver lo que es cursiva
```

---------- Continúa en la siguiente----------

ESCUELA IT

Ejercicios

---------- Viene de la anterior ----------

```
texto="Esto es cursiva"
formatexto "$texto" cursiva
echo Quiero ver lo que es color rojo
texto="Esto es rojo"
formatexto "$texto" rojo
echo Quiero ver lo que es negrita
texto="Esto es negrita"
formatexto "$texto" negrita
echo Quiero ver lo que es todo junto
texto="Y esto todo junto"
formatexto "$texto" todos
```

ESCUELA IT

BASH

THE BOURNE-AGAIN SHELL

GRACIAS!!

Angel Luis Calvo

ESCUELA IT