1. **Hacer un shell-script de un juego que consista en acertar un número generado aleatoriamente a partir de la hora del sistema. Cada vez que introducimos un número, se nos indica si el valor correcto es mayor o menor; por último, si acertamos, nos indica el número de intentos que hemos necesitado.**

Sol 1:

```
TRUE=0
FALSE=1
vale=TRUE                    # Condición de terminación
cont=0                       # Número de intentos

#    Cálculo del valor inicial a partir de la hora    #
var1=’date | cut -c12-13’
var2=’date | cut -c15-16’
 var3=’date I cut -c18-19'
res1=’expr $var1 \*   10’
res2=’expr $var2 \*  200’
res3=’expr  $res1 + $res2’
res5=’expr $res3 + $var3’
valor=$res5
clear

while  [ $vale = TRUE ]
do
cont=’expr $cont + 1’
echo
echo -n "Introduce un número: "
read numero
if [ $numero = $valor ]
 then
echo "Acertaste en $cont veces"
vale=FALSE
else
if [ $numero -lt $valor ]
 then
echo " $numero es bajo"
 else
echo " $numero es alto"
 fi
fi
done
```

2. **Hacer un shell-script que se encargue de borrar los archivos que le pasemos como parámetros pero dejando una copia de seguridad en el archivo oculto .papelera. La orden que denominaremos borra admite dos opciones –v y –b. Con la primera se muestra el contenido de la papelera, y con la segunda se borra.**

Sol 2:

```
##########################################
#       Comprobamos si la sintaxis es correcta       #
##########################################
if [ "$1" = “” ]
then
        echo

        echo "Sintaxis: $0 [ -v ] [ -b ] archive [archive … ] “ > &2
        echo
        exit  -1
fi

##########################################
#      Comprobamos si existe en el directorio       #
#      HOME el subdirectorio .papelera, si no       #
#      existe, lo creamos.                                                    #
##########################################
test  -d $H0ME/.papelera
if [ "$?" = "1" ]
        then
        mkdir $HOME/.papelera
 fi

##################################################
#      Comprobamos si el primer parámetro comienza    #
#      con un "-" para tomar las decisiones                 #
#      oportunas.                                                          #
##################################################
param='echo $1 | cut –c 1’
if [ "$param" = "-" ]
        then
        case $1 in
        -v) echo "La papelera incluye los siguientes archivos:"
        ls $H0ME/.papelera;;
        -b) echo "Estoy borrando la papelera"
        rm $HOME/.papelera/*;;
        -*) echo "$0: $1 argumento no válido" >&2
        exit;;
         esac
```

```
########################################
#       Borramos los archivos especificados       #
else
        echo -n "¿Está seguro de que quiere eliminar $*? (s/n):"
        read resp
        if [ "$resp" = "s" -o "$resp" = "S" ]
        then
                for i in $*
                do
                        if [ -f  "$i" ]
                        then
                                mv "$i" $HOME/.papelera > /dev/null 2> /dev/null
                        else
                                echo "$i: No existe" >&2
                         fi
                done
        else
                exit
        fi
fi
```

3. **Realizar un shell-script que lea dos números de teclado y muestre un mensaje indicando cual de ellos es mayor o bien si son iguales.**

Sol 3:

```
echo Introduce numero1
read n1
echo Introduce numero2
read n2
if [ $n1 –gt $n2 ]
then
        echo El numero $1 es mayor que $n2
elif [ $n2 –gt $n1 ]
then
        echo El número $n2 es mayor que $n1
else
        echo Los números $n1 y $n2 son iguales
fi
```

4. **Hacer un shell-script que recupere los archivos que estén guardados en la papelera. Estos archivos deben ser pasados como parámetros a "recupera". Si desea recuperar todos los archivos de la papelera, tiene que pasarle la opción –t.**

```
#       Comprobamos si la sintaxis es correcta       #
##############################################
if [ "$1" = "" ]
then
        echo
        echo "Sintaxis: $0 [-t] archivo [archivo ...]" >&2
        echo
        exit -1
 fi

#####################################
#Comprobamos si el primer parámetro       #
#comienza con un "-" para tomar las          #
#decisiones oportunas.                                                    #
#####################################
param='echo $1 | cut –c1'
if [ "$param" = "-" ]
        then
case $1 in
-t) if [ `ls $HOME/.papelera | wc –w` -eq 0 ]
        then
                echo "No hay archivos en la papelera" exit 0 fi
                echo "Recuperando todos los archivos borrados"
                for i in $HOME/.papelera/*
                        do
                                mv Si .
                        done;;
-*) echo "$0: $1 argumento no válido">&2
esac
#####################################
# # Recuperamos los archivos especificados #
#####################################
                else
                for i in $*
                        do
                                test -f $H0ME/.papelera/$i
                                 if [ $? - 1 ]
                                   then
                                     echo "$1 no existe"
                                   else
                                     mv $H0ME/.papelera/$i
                                 fi
                        done
                fi
```

5. **Realizar un shell-script que admita un único parámetro correspondiente a un fichero del directorio actual, si existe debe contar el número de líneas del mismo y mostrar un msg indicando dicho número, si no existe debe mostrar un msg de error y salir.**

Sol 5:

```
if [ $# -ne 1 ]
then
        echo Sintaxis $0 fichero
        echo Demasiados parámetros
else
        if [ ! –f $1]
        then
                echo El fichero no existe
                exit
        else
                wc –l $1 > ficheraux
        read numlinea nombre < ficheraux
                echo el número de líneas es $numlinea
        fi
fi
```

6. **1 º parte: Realizar un programa que dado un directorio inicial y el nombre de un fichero, el programa debe buscar el fichero en cualquier subdirectorio que cuelgue del inicial. Los parámetros serán:**
   1. **Directorio inicial, especificando mediante su ruta completa.**
   2. **Nombre del fichero a buscar.**

   Si se encuentra mostrar el fichero y su ruta.
   2ª parte: Si el programa encuentra el fichero se mostrará además de la ruta de la siguiente información máscara de permisos, tamaño en bytes y fecha de la última modificación realizada sobre el fichero en formato mm/dd/hora

Sol 6:

```
if [ $# -ne 2 ]
then
    echo error hay demasiados parámetros
elif [ ! –d $1 ]
then
    echo el directorio no existe
    exit
fi
fi
lista = `ls $1`
for i in lista
do
    if [ -f $i ]
    then
```

```
            if [ $2 = $i ]
            then
                    echo $1 / $i
            fi
    fi
    if [ -d $i ]
    then
            ejercicio $1 / $i $2 → llamada recursiva al programa
    fi
done
```

7. **Realice un Shell-Script que admita como único parámetro el nombre de un directorio, especificado mediante su nombre de ruta completo, si el directorio existe el programa debe mostrar el nombre de todos sus entradas, indicando para cada una de ellas si se trata de un fichero o un directorio. Si el directorio no existe se mostrará un mensaje de error**.

Sol 7:

```
if [ $# -ne 1]
then
    echo error, más de un parámetro
else
if [ ! –d $1 ]
then
    echo el fichero no existe
    exit
else
for i in $1/*     →   $1 es el directorio parámetro, p.e. /tmp y si queremos ver lo que cuelga de él la sintaxis será /tmp/* . También podemos generar lo que cuelga de $1 usando el comando ls, poniendo `ls $1`
    do
            if [ -f $i ]
            then
                    echo $i : fichero
            fi
    done
fi
fi
```

8. **Realizar un shell-script que reciba como parámetro un número indeterminado de nombre de ficheros y como último parámetro un directorio especificado mediante su nombre de ruta completo. El programa debe mostrar un mensaje indicando cuantos de los ficheros especificados permanecen en dicho directorio.**

Sol 9:

```
for i in $*
do
    directorio=$i
done
if [ ! –d $directorio ]
then
    echo error, el directorio no existe
    exit
else
for i in $*
do
if [ ! –d $i ]
    if [ -f $directorio /$i ]
    then
            count = $ ( cont + 1 )
    fi
fi
done
```

–

9. **Realice un programa de shell que reciba desde la línea de órdenes dos palabras y nos indique si son iguales o distintas. Si el número de parámetros no es correcto, se debe visualizar un mensaje de error.**

10. **Realice un shell-script que nos indique si los usuarios del sistema están conectados al sistema o no.**

11. **Realice un shell-script que nos diga los directorios existentes en un cierto directorio.**

12. **Hacer un shell-script que muestre el fichero del directorio activo con más líneas.**

Sol 12:

```
NLIN=0
for I in *
do
if [ -f $I ]
then
```

```
N=$(wc -l $I)
if [ $N -gt $NLIN ]
then
NOMBRE=$I
NLIN=$N
fi
fi
done
echo "$NOMBRE tiene $NLIN lineas"
```

13. Realice un shell-script que escriba en el fichero /tmp/usuarios una línea con la fecha y otra con el numero de usuarios distintos que están conectados en ese momento y que lo haga cada 2 minutos

Sol 13:

```
#!/bin/bash
while true
do
   date >> /tmp/usuarios
   who | cut -d ' ' -f 1 | sort -u | wc -l >> /tmp/usuarios
   sleep 120
done
```

14. Realice un shell-script que realice una copia de seguridad del sector de arranque de la partición $PARTITION en el fichero $FICHERO. SI $FICHERO existe, entonces primero se copia a $SEG y luego se sobrescribe

Sol 14:

```
#!/bin/bash
#
if [ "$#" -gt 0 ]; then
   if [ "$1" = "-v" ]; then
         verbose = 1
   else
         echo "$0: No se admiten parámetros, excepto -v"
   exit
fi

PARTITION=/dev/hda3

DIR=/dos
FICHERO=bootsec.lnx
SEG=bootsect.ln~

if [ -f $DIR/$FICHERO ]; then
   if [ $verbose = "1" ];then
         echo "$0: Moviendo $DIR/$FICHERO a $DIR/$SEG "
   fi
```

```
   mv $DIR/$FICHERO $DIR/$SEG
fi
# Realizamos la copia
if [ $verbose = "1" ]; then
   echo "$0: Copiando el sector de arranque de $PARTITION en
$DIR/$FICHERO"
fi
dd if=$PARTITION of=$DIR/$FICHERO bs=512 count=1
```

15. Realice un shell-script que cree un menú que me permita:

1. **Comprobar un listado largo del directorio**
2. **Comprobar que una archivo dado existe y su tamaño es > 0**
3. **Ver el número de usuarios que están conectados en ese momento**
4. **Salir**

Sol 15:

```
#MENU
OPCION=0
clear
while [ $OPCION != 4 ]
do
echo Elige opcion
echo "1.Listado largo del directorio"
echo "2.Comprobar que un archivo dado existe y es >0 su tamaño"
echo "3.Ver el nº de usuarios que estan conectados en este momento"
echo "4.Salir"
read OPCION
clear

case $OPCION in
1) ls -l $HOME;;
2) echo -e "Introducir fichero a leer: \\c"
  read FICHERO
  if (test -s $FICHERO);then
    echo "El fichero existe y su tamaño es mayor que 0"
  else
    echo "ERROR"
  fi;;
3) echo -e "Usuarios conectados\\c"
  who | cut -c 1-2 | sort | uniq | wc -l;
4) echo "ADIOS";;
*) echo "ERROR";;
esac
sleep 3s

clear
done
```

16. Realice un shell-script que mediante un menú me permita hacer las operaciones mas comunes de una calculadora: sumar, restar, dividir y multiplicar

Sol 16:

```
#CALCULADORA POR FUNCIONES
suma()
{
      RESULTADO=`expr $A + $B`
      echo SUMA: $RESULTADO
}
resta()
{
      RESULTADO=`expr $A - $B`
      echo RESTA: $RESULTADO
}
division()
{
      if [ $B -ne 0 ];then
            RESULTADO=`expr $A / $B`
            echo DIVISION: $RESULTADO
        else
            echo "No divisible por 0"
        fi
}
multiplicar()
{
      RESULTADO=`expr $A \* $B`
      echo MULTIPLICACION: $RESULTADO
}

OPCION=2

while [ $OPCION -ne 5 ]
do
      echo \*\*\*\*     CALCULADORA    \*\*\*\*
      echo \*\*\*\* 1. SUMA            \*\*\*\*
      echo \*\*\*\* 2. RESTA           \*\*\*\*
      echo \*\*\*\* 3. DIVISION     \*\*\*\*
      echo \*\*\*\* 4. MULTIPLICACION  \*\*\*\*
      echo \*\*\*\* 5. SALIR           \*\*\*\*
      echo -e "OPCION: \c"
      read OPCION
      clear
      if [ $OPCION -ge 1 ] && [ $OPCION -le 4 ];then
         echo -e "A: \c"
         read A
         echo -e "B: \c"
         read B
      fi
      case $OPCION in
      1) suma $A $B;;
      2) resta $A $B;;
      3) division $A &B;;
      4) multiplicar $A $B;;
      5) echo ADIOS;;
      *) echo Opcion no valida;;
      esac
```

```
done
```

17. Realice un shell-script que reciba un nombre de archivo como parámetro e indique, imprimiendo todas las leyendas que correspondan, si el archivo es legible, modificable y ejecutable por el usuario.

Sol 17:

```
#!/bin/bsh
# carsarch.sh: características de un archivo
echo Caracteristicas del archivo $1
if [ -r $1 ]
then
  echo es legible
fi
if [ -w $1 ]
then
  echo es grabable
fi
if [ -r $1 ]
then
  echo es ejecutable
fi
```

18. Realice un shell-script que reciba varios nombres de archivo como parámetros, y para cada uno validar si el nombre corresponde a un archivo común existente, y si es así mostrarlo en pantalla paginando.

Sol 18:

```
#!/bin/bash
# mostrarchs.sh: muestra contenido de varios archivos
for VAR in $*
do
if [ -f $VAR ]
then
  more $VAR
else
  echo No existe $VAR
fi
done
```

19. Realice un shell-script que reciba un nombre de directorio, valide la existencia y condición de directorio y muestre nombres de todos los directorios y subdirectorios bajo él, en formato de página largo 23.

Sol 19:

```
#!/bin/bash
# esdir.sh: verifica directorio y muestra contenido
recursivo
```

```
clear
if [ -d $1 ]
then
  echo Directorios bajo $1
  echo "Digite Enter para continuar"; read; clear
  ls -lR $1 2>/dev/null | grep '^d' | pr -l24 | more -24
  # el valor 24 en more es para visualizar en pantalla
else
  echo No existe el directorio $1
fi
```

20. Realice un shell-script que reciba un nombre de archivo, verifique que existe y que es un archivo común, lo convierta en ejecutable para el dueño y el grupo y muestre el modo final.

Sol 20:

```
#!/bin/bash
# seaejec: convierte un archivo en ejecutable
#
ARCH=$1
if [ -f $ARCH ]          # existe y es archivo regular
then
  chmod ug+x $ARCH
  ls -l $ARCH
else
  echo "seaejec: el archivo $ARCH no pudo ser convertido"
fi
```

21. Escribir un programa que mueva todos los programas del directorio actual (archivos ejecutables) hacia el subdirectorio `bin` del directorio propio del usuario, muestre los nombres de los que mueve e indique cuántos ha movido o que no ha movido ninguno. Si el directorio bin no existe, deberá ser creado.

Sol 21:

```
#!/bin/bash
# copiabin.sh: copia archivos ejecutables hacia $HOME/bin
#
# si el directorio bin no existe lo crea
if [ ! -d $HOME/bin ]
then
  mkdir $HOME/bin
fi
# copia de archivos
N=0            # contador de archivos copiados
for ARCH in *
do
  if [ -x $ARCH -a -f $ARCH ]  # ejecutable y archivo
común (no directorio)
  then
    cp $ARCH $HOME/bin
    echo "  $ARCH fue copiado a $HOME/bin"
    N=`expr $N + 1`
  fi
done
if [ $N -eq 0 ]
```

```
then
  echo "No se copió ningún archivo"
else
  echo "Fueron copiados $N archivos"
fi
```

22. Usando el archivo /etc/passwd escribir el programa `usuarios` que lista los nombres de login, el directorio propio del usuario y el intérprete invocado por defecto de todos los usuarios, ordenados alfabéticamente por nombre de login.

Sol 22:

```
# usuarios: lista datos de usuarios
#
echo "Nombres de usuarios, Directorio propio, intérprete
de comandos"
ypcat passwd | cut -d: -f1,6,7 | sort | more
echo
```

23. Usando solamente el archivo /etc/group, escribir los siguientes programas:
a) 'grupo1': listar los nombres y números de grupo y la lista de usuarios de cada uno, ordenados por nombre.
b) 'grupo2': igual, ordenados por número de grupo.
c) 'grupo3': reúne las dos salidas anteriores, con leyendas explicativas adecuadas para cada parte y para cada columna, así como nombre de la máquina y fecha del día.

Sol 23:

```
#!/bin/bash
#
# grupo1:
clear
echo "Grupos por nombre:"
echo
-------------------------------------------------------
echo "login:número_de_grupo:lista de usuarios"
echo
-------------------------------------------------------
ypcat group | cut -d: -f1,3,4 | sort | more -18
echo
-------------------------------------------------------
echo Digite Enter para continuar
read
clear

#!/bin/bash
#
# grupo2:
echo "Grupos por número:"
echo
-------------------------------------------------------
echo "login:número_de_grupo:lista de usuarios"
echo
-------------------------------------------------------
ypcat group | cut -d: -f1,3,4 | sort -t: -n +1 | more -18
```

```
echo
------------------------------------------------------------
echo

#!/bin/bash
#
# grupo3:
clear
./grupo1
echo
./grupo2
echo
echo Máquina: `hostname`
echo Fecha: `date`
```

24. Escribir un programa `usugrup` que dado un nombre de login de usuario determine si existe en el sistema, y si es así, presente su nombre de usuario, , número de usuario (UID), grupo primario y grupos secundarios si los tiene, con leyendas adecuadas.

Sol 24:

```
#!/bin/bash
# usugrup: datos y grupos de un usuario
#
USUARIO=$1
id $USUARIO 1>/dev/null 2>&1
ERROR=$?
if [ $ERROR -ne 0 ]
then
  echo "El usuario " $USUARIO "no existe"
  exit
fi
NOMBRE=`id $USUARIO | cut -f1 -d" "`
echo \(UID\) y nombre: $NOMBRE
GRUPO1=`id $USUARIO | cut -f2 -d" "`
echo \(GID\) y grupo primario: $GRUPO1
if test `id $USUARIO | tr " " "\n" | wc -l` -gt 2
then
  GRUPOS2=`id $USUARIO | cut -f3 -d" "`
echo \(GIDs\) y grupos secundarios: $GRUPOS2
fi
```

25. Escribir un programa `grupusu` que dado un nombre de grupo determine si existe en el sistema, y si es así, presente su nombre, número de grupo (GID), y lista de usuarios que pertenezcan a él, ya sea como grupo primario (en /etc/passwd) o como grupo secundario (lista en /etc/group).

Sol 25:

```
#!/bin/bash
# determina usuarios en un grupo

GRUPO=$1        # nombre de variable significativo
```

```
EXISTE=`ypcat group | grep "^$GRUPO"`
if [ ! $EXISTE ]
then
  echo "El grupo $GRUPO no existe."
  exit
fi

# extrae número del grupo

GID=`echo $EXISTE | cut -d: -f3`
echo "El número del grupo $GRUPO es $GID"
# busca usuarios con este grupo primario

echo Usuarios en este grupo como primario:
  # corta campos usuario e id grupo,

# selecciona líneas con $GID al final,
# luego corta el GID, deja nombre usuario
ypcat passwd | cut -d: -f1,4 | grep :$GID$ | cut -d: -f1

# busca usuarios con este grupo secundario

echo Usuarios en este grupo como secundario: echo $EXISTE
| cut -d: -f4 | tr "," " "
```

26. Escribir los siguientes programas:
a) `ligass`: muestra los nombres de archivo que son enlaces simbólicos.
b) `ligash`: muestra los archivos que tiene enlaces hard.
Ambos programas reciben un nombre como como parámetro, y validarán que corresponda a un directorio del sistema.

Sol 26:

```
#!/bin/bash
# ligass: lista archivos que son enlace simbólico o tienen
enlaces hard
# simbólicos: en ls -l se busca que empiece con l
if [ ! -d $1 ]
then
  echo Error: ligas: $1 no es un directorio
  exit
fi
echo Archivos que son enlace simbólico:
ls -l | grep "^l" | cut -56 -
echo
# hard: se busca 3 espacios y distinto de 1 como contador de
enlaces
echo Archivos que tienen enlace hard:
ls -l | grep -v "^d" | cut -c11 - | grep -v "^ 1" | cut -c46 -
```

27. Escribir un programa `saludo` que, según la hora, escriba el saludo correspondiente al nombre de pila del usuario. En el archivo /etc/passwd los usuarios deben estar ingresados con nombre y apellido separados por blanco. Los saludos corresponden a las siguientes horas: Buenos días, de 05:00 hasta 12:59; Buenas tardes, de 13:00 hasta 19:59; Buenas noches 20:00 hasta 04:59. Ejemplo de mensaje: Buenos días, Juan.

A efectos de pruebas, se recibirán la hora y el nombre de login como parámetros, dejando comentados los comandos donde se extrae la hora real y se toma el usuario real.

Sol 27:

```
#!/bin/bash
# saludo.cmd
# script en UNIX que saludo al usuario por su nombre
NOMBRE=`cat /etc/passwd | grep "^$LOGNAME" | \
cut -d: -f5 | cut -d' ' -f1`
# si el usuario no tiene ingresado un nombre, toma
"Nadie"
NOMBRE=${NOMBRE:-Nadie}
HORA=`date | cut -c12-13 | tr -d ' '`
if expr "$HORA" \<= 5 > /dev/null
then
  echo 'Buenas noches, '$NOMBRE
elif expr "$HORA" \<= 12 > /dev/null
then
  echo 'Buenos dias, '$NOMBRE
elif expr "$HORA" \<= 19 > /dev/null
then
  echo 'Buenas tardes, '$NOMBRE
elif expr "$HORA" \<= 24 > /dev/null
then
  echo 'Buenas noches, '$NOMBRE
fi
```

28. Un script de respaldo produce, entre otros mensajes, líneas del tipo
`"Total bytes written 18804023"`
Guarda su salida en el archivo `respaldo.error`. Escribir un script `total.cinta` que sume los bytes grabados e indique el total en bytes, Mb y Gb.
Crear un archivo `respaldo.error` de prueba, con un contenido tal como
`Total bytes written 1800`
`Total bytes written 1000`

Sol 28:

```
#
# total.cinta: cantidad de bytes grabados en cinta
#
TOTAL=0
LISTA=`cat respaldo.error | tr -dc "[0-9] "`
for NUM in $LISTA
do
  TOTAL=`expr $TOTAL + $NUM`
done
echo "Total General de bytes respaldados: "$TOTAL
TOTALMB=`expr $TOTAL \/ 1000000`
echo "Total General de MB respaldados: "$TOTALMB
TOTALGB=`expr $TOTALMB \/ 1000`
echo "Total General de GB respaldados: "$TOTALGB
```

29. Escribir un shell-script que construya un menú con 4 opciones. El menú opera sobre un fichero llamado datos. Cada línea del fichero contiene dos cadenas: un nombre y un numero de teléfono. El fichero esta ordenado alfabéticamente por el nombre. Las opciones del menú son:

1. **Búsqueda de un teléfono, dado el nombre**
2. **Alta, dado el nombre**
3. **Baja, dado el nombre**
4. **Listado por pantalla del fichero completo**

Sol 29:

```
# Conviene crear el fichero vacio si no existe
if [ ! -f datos ]
then
  echo > datos
fi


while true
do
  clear
  echo AGENDA TELEFONICA
  echo
  echo   1. Buscar entrada
  echo   2. Insertar entrada
  echo   3. Borrar entrada
  echo   4. Mostrar listado
  echo   5. Salir
  echo
  echo -n Opcion elegida:

  read opcion
  case $opcion in
    1) clear
       echo BUSCAR ENTRADA
       echo
       echo -n Introduzca el nombre del usuario:
       read nombre
       if [ -z $nombre ]
       then
          break
       fi
       echo
       grep -i $nombre datos
       if [ $? -ne 0 ]
       then
         echo La entrada no existe
         echo
       fi
       echo
       echo Pulse INTRO para continuar...
       read intro;;

    2) clear
       echo INSERTAR ENTRADA
       echo
```

```
            echo -n Introduzca el nombre del usuario:
            read nombre
            if [ -z $nombre ]
            then
               break
            fi
            echo
            echo -n Introduzca el telefono:
            read telefono
            if [ -z $telefono ]
            then
               break
            fi
            echo
            echo "$nombre $telefono" >> datos
            sort -d datos > temp
            mv temp datos;;

         3) clear
            echo BORRAR ENTRADA
            echo
            echo -n Introduzca el nombre del usuario:
            read nombre
            if [ -z $nombre ]
            then
               break
            fi
            echo
            # Opcion 1
            grep -v $nombre datos > temp
            mv temp datos;;
            # Opcion 2
            # grep -d "^[\<$nombre\>]" datos > temp
            # mv temp datos;;

         4) clear
            echo LISTAR ENTRADAS
            echo
            more datos
            echo
            echo Pulse INTRO para continuar...
            read intro;;

         5) clear
            exit 0;;
      esac
   done
```

30. Escribir un Shell-Script que tome como parámetro el nombre de un directorio y realice las siguientes acciones:
1. Comprobar que el numero de parametros es correcto
2. Comprobar que el parametro es un directorio
3. Para cada entrada del directorio especificado el programa genera:
El numero de la entrada
Nombre de la entrada
Tipo de la entrada: fichero, directorio, ...
Tamaño
Numero de enlaces
Día y Mes de la última modificación

Sol 30:

```
# Comprobamos el numero de parametros

if [ $# -ne 1 ]
then
   echo Numero de parametros incorrecto
   echo
   echo Sintaxis: $0 directorio
   echo
   exit 0
fi

# Comprobamos que el parametro es un directorio

if [ ! -d $1 ]
then
   echo El parametro especificado no es un directorio
   echo
   echo Sintaxis: $0 directorio
   echo
   exit 0
fi

# Listado de los ficheros del directorio

cont=0
for i in $1/*
do
  echo Entrada numero: $cont
  ls -l $i
  cont=`expr $cont + 1`
done
```

31. Realizar un shell-script en el que:
1. Si el primer parámetro es -l o -L, cree un enlace del fichero o directorio especificado en el segundo parámetro con el nombre especificado en el tercer parámetro.
2. Si el primer parámetro es -c o -C, contara y mostrara el numero de líneas, palabras y caracteres de todos los ficheros que se pasen por parámetro.

3. Si el primer parámetro es un nombre de fichero existente en el directorio actual, cambiar al directorio a cual nos vamos al hacer cd sin parámetros, crear en este un subdirectorio cuyo nombre se leerá por teclado, cambiar a este subdirectorio y copiar el fichero dado como primer parámetro con el nombre especificado en el segundo parámetro.

Sol 31:

```
# CASO 1

if [ $1 = "l" -o $1 = "L" ]
then
  if [ $# -ne 3 ]
  then
    echo Sintaxis $0 -l|L fichero/directorio nombre
    echo
    exit
  else
    if [ -f $2 ]
    then
      ln $2 $3
    else
      ln -s $2 $3
    fi
    echo Enlace realizado con exito
    exit
  fi
fi

# CASO 2

if [ $1 = "c" -o $1 = "C" ]
then
  if [ $# -lt 2 ]
  then
    echo Sintaxis $0 -c|C lista_ficheros
    echo
    exit
  else
    for i in $*
    do
      if [ -f $i ]
      then
        wc $i > temp
        read lineas palabras caracteres x < temp
        echo $i: $lineas lineas, $palabras palabras, $caracteres
caracteres
        rm temp
      else
        echo Atencion: $i no es un fichero
      fi
    done
    exit
  fi
fi

# CASO 3

if [ $# -lt 2 ]
```

```
then
  echo Sintaxis $0 fichero1 fichero2
  echo
  exit
else
  if [ -f $1 ]
  then
    actual=$PWD
    cd $HOME

    # Comprobacion del directorio
    while true
    do
      echo -n Introduzca el nombre del directorio:
      read directorio
      if [ -z $directorio ]
      then
        continue
      else
        break
      fi
    done

    # Si no existe el directorio lo creamos
    if [ -d $directorio ]
    then
      echo El directorio ya existe
    else
      mkdir $directorio
    fi

    cd $directorio
    cp $actual/$1 $2
    echo Copia realizada con exito
  else
    echo El fichero especificado no existe
    echo
    exit
  fi
```

fi

32. Realizar un shell-script en el que el usuario debe introducir 5 palabras separadas por espacios que serán ordenadas alfabéticamente en orden inverso y mostrará en pantalla únicamente la primera palabra resultado de la ordenación.

Sol 32:

```
clear
echo "Introduzca 5 elementos (separados por espacios):"
read uno dos tres cuatro cinco
echo $uno > temp
echo $dos >> temp
echo $tres >> temp
echo $cuatro >> temp
echo $cinco >> temp
sort -r temp > temp2
resultado=`head -1 temp2`
rm temp
```

```
rm temp2
echo El resultado es $resultado
```

33. Comprobar si el numero de ficheros existentes en el directorio actual es mayor que el número de usuarios conectados al sistema

El shell-script debe responder:

usuarios: si el numero de usuario es mayor
ficheros: si el numero de ficheros es mayor

Sol 33:

```
clear
who > usuarios
wc -l usuarios > temp1
ls -l > ficheros
wc -l ficheros >> temp1
sort -r temp1  > temp2
head -1 temp2 > temp1
read x resultado < temp1
rm temp1
rm temp2
rm usuarios
rm ficheros
echo Contiene mayor numero de lineas: $resultado
```

34. Realizar un shell script que cree dos directorio D1 y D2. Dentro de D2 debe crear un fichero 'hola' que contenga el nombre del usuario conectado. Después debemos movernos al directorio D1 y desde allí crear un enlace simbólico al fichero hola especificando su ruta completa.

Sol 34:

```
mkdir D1
mkdir D2
cd D2
logname > hola
ruta=`pwd`
cd ../D1
ln -s $ruta/hola
```

35. En un sistema de información disponemos de un fichero de texto de la forma:

Primer_Apellido Segundo_Apellido Nombre

La prueba consistirá en escribir un shell-script que tenga tres parámetros:

1. Directorio en el que se encuentra el fichero (D)
2. Nombre del fichero (F)
3. Numero de líneas (N)

El shell-script debe realizar las siguientes acciones:

1. Comprobar la existencia del directorio (D).
2. Comprobar la existencia y derechos de lectura sobre el fichero (F)dentro del directorio indicado
3. Localizar el fichero (F) todas las líneas correspondientes a sujetos cuyo primer apellido sea Felipe
4. Crear un fichero (f1) ordenado alfabéticamente segun el segundo apellido, que contenga los N primeros nombres localizados

Sol 35:

```
# Comprobacion del numero de parametros
if [ $# -ne 3 ]
then
   echo Sintaxis: $0 directorio fichero numero_lineas
   echo
   exit
fi

# Comprobar que el directorio existe
if [ ! -d $1 ]
then
   echo Error: El directorio no existe
   echo
   exit
fi

# Comprobar que el fichero existe
if [ ! -f $1/$2 ]
then
   echo Error: El fichero no existe
   echo
   exit
fi

# Comprobar que el derecho de lectura del fichero
if [ ! -r $1/$2 ]
then
   echo Error: El fichero debe tener permiso de lectura
   echo
   exit
fi

grep "^Felipe" $1/$2 > $1/temp

if [ $? -ne 0 ]
then
   echo No se encontro el patron
   rm $1/temp
   exit
else
   echo "Las entradas con el primer apellido \"Felipe\" son:"
   cat $1/temp
   echo
fi

sort +1 -2 $1/temp > $1/f1
echo "Los $3 primeros nombres del fichero ya ordenado son: "
head -$3 $1/f1
```

```
rm $1/temp
```

36. Realizar un shell-script que reciba como parámetro una extensión de archivo sin punto y el nombre de un directorio especificado mediante su nombre de ruta completo. El shell-script preguntará al usuario si desea copiar o mover los archivos del directorio actual que tengan esa extensión al directorio especificado como segundo parámetro. Una vez realizada la acción escogida se mostrará por pantalla un listado de los ficheros afectados, ordenados alfabéticamente.

Sol 36:

```
if [ $# -ne 2 ]
then
        echo error, número de parámetros incorrecto
        exit
fi
if [ ! –d $2 ]
then
        error directorio no existe
        exit
fi
echo Elija una opción:
echo 1. Para copiar
echo 2. Para mover
read opcion
while [ $opcion –ne 1 ] || [ $opcion –ne 2 ]
do
        echo error, opción incorrecta
        echo vuelva a teclear opcion
done
if [ $opcion –eq 1 ]
then
        cp *.$1 $2  //copia todos los archivos con extensión $1 a el directorio $2
fi
if [ $opcion –eq 2 ]
then
        mv *.$1 $2  //copia todos los archivos con extensión $1 a el directorio $2
fi
```