UNIVERSIDAD NACIONAL DE SAN LUIS
FACULTAD DE CIENCIAS FÍSICAS, MATEMÁTICAS Y NATURALES

Programación Avanzada de Sistemas Embebidos

Trabajo Práctico 2 - Bourne Again Shell

Autores: Leandro Marsó, Guillermo Larregay, Mauricio Gomez.
Profesores: Mario Beron, Martín Murdocca, Alejandro Nuñez Manquez

---

abril 2018

Ejercicio 1

Escriba[1] un script que reciba como parámetros un paso, un archivo que contiene una lista de archivos. El script realiza una copia de seguridad de los archivos especificados en el segundo parámetro el directorio Backup el cual se encuentra en el paso especificado en el primer parámetro.

```
#!/bin/bash
paso=$1
files=$2

echo "Copiando ... "
for i in `cat $files`
do
    echo " $i"
    cp $i $paso
done
echo "en el directorio $paso"
```

Ejercicio 2

Escriba el script myBasicBC. Este script recibe como parámetros dos números enteros y dependiendo de una opción especificada por el usuario en la línea de comandos realiza la suma, resta, multiplicación o división de los números recibidos como parámetros.

Ejercicio 3

Escriba el script miOperacion. Este script utiliza dos opciones, la primera indica si la operación que se desea realizar es una suma o una multiplicación. La segunda opción utiliza un parámetro que indica el límite superior de una sumatoria o productoria dependiendo de la operación especificada por la primera opción. El script realiza la sumatoria o productoria desde 1 hasta el parámetro indicado en la segunda opción de los números que comienzan a partir de un parámetro del script miSumatoria. A continuación se muestra como el script se debería invocar:

```

if [ $# -eq 4 ]
then
   while [ -n "$1" ]
   do
      case "$1" in
            -s) #echo "Sumatoria:"
                operation="Sumatoria";;
            -p) #echo "Se realizará una productoria..."
                operation="Productoria";;
            -i) limiteSuperior=$2
                escalar=$3
                echo "$operation con i desde 1 hasta $limiteSuperior ... "
                shift 3;;
            *) echo "$1 no es una opción válida";;
      esac
      shift
   done
else
#
# Si hubo error de cantidad de parámetros:
   echo "Ejemplos de uso:"
   echo "El siguiente comando realiza la \
 sumatoria desde i=1 hasta 8 de 5+i:"
   echo "$0 -s -i 8 5 "
   echo ""
   echo "El siguiente comando realiza\
```

[1]Para descargar todos los archivos fuentes de este trabajo práctico, visitar el siguiente link: `https://github.com/3l134ndr0/ese/tree/master/pase/tp/tp2/src`

```
 la productoria desde i=1 hasta 8 de 5*i"
   echo "$0 -p -i 8 5"
   fi

#
# Acá defino que hacer, si la sumatoria o productoria
# en base al valor de la variable $operation
if [ $operation = "Sumatoria" ]
then
    echo "de $escalar + i"
    acum=0
    for (( i=1; i <=limiteSuperior ; i++ ))
    do
        (( acum += i + escalar ))
    done
    echo "El resultado es $acum"
elif [ $operation = "Productoria" ]
then
    echo "de $escalar * i"
    acum=1
    for (( i=1; i <=limiteSuperior ; i++ ))
    do
        (( acum *= i * escalar ))
    done
    echo "El resultado es $acum"
fi
```

Ejercicio 4

Escriba un script que defina la función menú. Dicha función imprime por pantalla los siguientes carteles: 1) Sumar; 2) Restar; 3) Salir y retorna como resultado la opción seleccionada por el usuario. El script debe invocar a la función y luego realizar la tarea indicada por el usuario.

```
#!/bin/bash

#Ejercicio 4: Escriba un script que defina la función menú. Dicha función imprime por pantalla los
    siguientes carteles: 1) Sumar; 2) Restar; 3) Salir y retorna como resultado la opción seleccionada
    por el usuario. El script debe invocar a la función y luego realizar la tarea indicada por el
    usuario.

function menu {
    echo "Elija una opción:"
    echo "1) Sumar"
    echo "2) Restar"
    echo "3) Salir"
    read opcion
    echo "Ud. eligió la opción $opcion"
    return $opcion
}

function echoParams {
while [ -n "$1" ]
do
    echo $1
    shift
done
}

# Llama al menú para obtener la opción del/a usuario/a
#menu. Como la variable opcion es global no la asigno a nada
opcionUser= menu
case $opcion in
    1)read -p "Ingrese dos números: " sum1 sum2
    echo $((sum1 + sum2));;
```

```
    2)read -p "Ingrese dos números: " sum1 sum2
    echo $((sum1 - sum2));;
    *)echo "¡Adios!";;
esac
```

Ejercicio 5

Escriba un script que defina la función mayor. La función recibe como parámetros dos números y retorna como resultado el mayor. El script pide al usuario dos números e invoca a la función. Luego de realizada dicha tarea imprime la leyenda “El número mayor es:” y seguidamente el valor retornado por la función.

```
#!/bin/bash
# Ejercicio 5: Escriba un script que defina la función mayor. La función recibe como parámetros
# dos números y retorna como resultado el mayor. El script pide al usuario dos números e invoca
# a la función. Luego de realizada dicha tarea imprime la leyenda "El número mayor es:" y seguidamente
# el valor retornado por la función.


mayor () {
if [ $1 \> $2 ]
then
    echo $1
else
    echo $2
fi
}

read -p "Ingrese dos números: " var1 var2
echo "El mayor de los dos es: $(mayor $var1 $var2)"
```

Ejercicio 6

Escriba un script que defina una función que permita contar el número de líneas de un archivo que contienen una palabra específica. La función recibe como parámetros el archivo y la palabra y retorna como resultado el número de líneas. El script debe invocar a la función y mostrar el resultado.

```
# Ejercicio 6: Escriba un script que defina una función que permita contar el número de líneas de
# un archivo que contienen una palabra específica. La función recibe como parámetros el archivo y
# la palabra y retorna como resultado el número de líneas. El script debe invocar a la función
# y mostrar el resultado.

cuantasVeces () {
    # Esta función cuenta la cantidad de apariciones en
    # todo el archivo (no es lo que se pide)
    local contador=0
while read line
do
    for word in $line
    do
        if [ $word = $2 ]
        then
            ((contador++))
        fi
    done
done <$1
echo $contador
}

cuantasLineas () {
    # Esta versión es la que si encuentra una línea
    # con la palabra buscada, incrementa el contador
    # y pasa a la siguiente línea.
```

```
    local contador
while read line
do
    for word in $line
       do
               if [ $word = $2 ]
               then
                   ((contador++))
                   break # Sólo cuenta una vez
               fi
       done
done <$1
echo $contador
}


# Main
read -p "Ingrese un nombre de archivo: " archivo
read -p "Ingrese una palabra a buscar en el archivo $archivo: " palabra
echo "Se encontraron $(cuantasLineas $archivo $palabra) líneas que contienen esa palabra"
echo "La palabra $palabra aparece $(cuantasVeces $archivo $palabra) veces en ese archivo"
```

Ejercicio 7

Escriba un script que defina la función mostrarTodo. Esta función recibe como parámetro una extensión de archivo y muestra todos los archivos que se encuentran en la carpeta actual que tienen la extensión recibida como parámetro.

```
#!/bin/bash

# Ejercicio 7: Escriba un script que defina la función mostrarTodo. Esta función recibe como
# parámetro una extensión de archivo y muestra todos los archivos que se encuentran en la carpeta
# actual que tienen la extensión recibida como parámetro.

mostrarTodo () {

    for file in $PWD/* #Current directory
    do
        if [ -f $file ]
        then
            if [ $1 = ${file: -${#1 ]thenecho file*/fifidone
```

Ejercicio 8

Escriba un script que defina la función apliqueATodos. Esta función recibe como parámetros un comando simple (comando con un parámetro y sin opciones) y una lista de archivos. La función aplica el comando a cada uno de los archivos recibidos como parámetros.

```
# Ejercicio 8: Escriba un script que defina la función apliqueATodos. Esta función recibe
# como parámetros un comando simple (comando con un parámetro y sin opciones) y una lista
# de archivos. La función aplica el comando a cada uno de los archivos recibidos como parámetros.

apliqueATodos () {
    comando=$1
    shift
    while [ -n "$1" ]
    do
        $comando $1
        shift
    done
}
```

jjj

Ejercicio 9

Diga que hace el siguiente script:

```
DIRECTORIO=$1
ARCHIVO_BUS=$2
shift 2
PALABRAS=$*
 if [ $# -ge 3 ]
  then
    ls $DIRECTORIO | grep $ARCHIVO_BUS | while read ARCHIVO
      do
          grep "$PALABRAS" ${DIRECTRIO}/${ARCHIVO}
           done
           else
            echo "Número de argumentos insuficientes"
             echo "Use: $0 <directorio> <archivo_a_buscar> <palabras>"
             fi

```

donde:

- $1: Nombre de un directorio.
- $2: Nombre de un archivo.
- $3: Una palabra a buscar.

```
#!/bin/bash

DIRECTORIO=$1
ARCHIVO_BUS=$2
shift 2
PALABRAS=$*
echo "Cantidad de parámetros: $#"
if [ $# -ge 3 ]
then
   ls $DIRECTORIO | grep $ARCHIVO_BUS | while read ARCHIVO
                                        do
                                          grep "$PALABRAS" ${DIRECTRIO}/${ARCHIVO}
                                        done
else
   echo "Número de argumentos insuficientes"
   echo "Use: $0 <directorio> <archivo_a_buscar> <palabras>"
fi

 # Este script recibe 3 parámetros: Un directorio, un archivo y una o más palabras. Supuestamente
 # busca esas palabra en el archivo de ese directorio, si la encuentra nos dice en que líneas aparece.
 #
 # El problema es que nunca hace eso, ya que la condición de test nunca se cumple, debido
 # al shift 2. Por lo tanto para que funcione el test sería: -ge 1
 # Además hay un error de tipeo cuando usamos la variable DIRECTORIO en la línea 10. Corrijo los dos
    bugs
 # y el script queda así:

ejercicio9Corregido () {
DIRECTORIO=$1
ARCHIVO_BUS=$2
shift 2
PALABRAS=$* # Agrupa todos los parámetros restantes en una sola variable.
if [ $# -ge 1 ]  Luego del shift 2 deberían quedar al menos un parámetro (la/s palabra/s a buscar)
then
  ls $DIRECTORIO | grep $ARCHIVO_BUS | while read ARCHIVO
                                       do
                                        echo "grep \"$PALABRAS\" ${DIRECTRIO}/${ARCHIVO}"
                                        grep "$PALABRAS" ${DIRECTORIO}/${ARCHIVO}
```

```
                                        done
else
  echo "Número de argumentos insuficientes"
  echo "Use: $0 <directorio> <archivo_a_buscar> <palabras>"
fi

}
```

Ejercicio 10

Haga un script que permita que los archivos pasados como parámetros, los cuales son todos ejecutables, se puedan ejecutar.

```
#!/bin/bash

# Ejercicio 10: Haga un script que permita que los archivos pasados como parámetros,
# los cuales son todos ejecutables, se puedan ejecutar.

while [ "$1" ]
do
    echo "$1"
    $1 # Ejecuta el archivo
    shift # pasa al siguiente parámetro
done
```

Ejercicio 11

Escriba un script que reciba como parámetro un paso donde se encuentran archivos `.c` (sin dependencias entre si) los compile y luego cree dos carpetas una Ejecutables y la otra Fuentes y copie los archivos fuentes en la carpeta Fuentes y los ejecutables en la carpeta Ejecutables.

```
#!/bin/bash

# Ejercicio 11: Escriba un script que reciba como parámetro un paso donde se
# encuentran archivos .c (sin dependencias entre si) los compile y luego cree
# dos carpetas una Ejecutables y la otra Fuentes y copie los archivos fuentes
# en la carpeta Fuentes y los ejecutables en la carpeta Ejecutables.

# Zona para configurar
# Directorio donde se guardarán los archivos fuentes:
FUENTES="Fuentes"
# Directorio donde se guardarán los ejecutables:
EJECUTABLES="Ejecutables"
# Compilador a usar:
GCC=gcc
#

source ejercicio7.sh # Importa la función mostrarTodo

originalDir=`pwd` # Recuerda el directoria desde donde fue ejecutado, para volver.
directorio=$1
cd $directorio

#
if [ -d $FUENTES ] && [ -d $EJECUTABLES ]
then
    echo "Los directorios $FUENTES y $EJECUTABLES ya existen"
else
    mkdir $FUENTES ; mkdir $EJECUTABLES
fi
#

# Compilo
```

```
33 mostrarTodo .c $directorio | while read SOURCE
34                       do
35                           base=${SOURCE%%.*} # Le borra la extensión al archivo
36                           echo "$GCC $SOURCE -o $base"
37                           $GCC $SOURCE -o $base # Para hacer el ejecutable sin extensión
38                           echo "cp $SOURCE $FUENTES ; cp $base $EJECUTABLES "
39                           cp $SOURCE $FUENTES ; cp $base $EJECUTABLES
40                       done
```