Bucles "**while**", "**for**" y "**until**".

Por Jorge Fuertes
http://jorgefuertes.com

Revisado a 15 de mayo de 2009

Índice

1. Introducción

1.1. Los bucles

Los bucles o *loops* en inglés, son una parte fundamental del control de flujo en cualquier lenguaje de programación. Estas construcciones nos permiten ejecutar la misma *rutina*[1] repetidamente, en base a una *condición de carrera*[2] o bien hasta el final de una secuencia[3] o de un flujo de entrada de datos[4].

En el *bash scripting*[5], hay tres estructuras que nos permitirán realizar bucles. Estas son:

- **while**: Ejecutará la rutina **mientras se cumpla** una determinada condición de carrera.
- **for**: Ejecutará la rutina **a lo largo de toda una secuencia o entrada**, siendo cada elemento de esta secuencia parte de la *iteración*[6] a través de una variable.
- **until**: Ejecutará la rutina **hasta que se cumpla** una determinada condición de carrera.

2. Bucles *while*

Una estructura *while* se ejecutará **mientras se cumpla** una determinada condición de carrera. Es decir, que la condición especificada debe ser verdadera para que el contenido del bucle pueda ejecutarse. Supuestamente algo en dicho contenido producirá variaciones en la condición de carrera hasta provocar que esta sea falsa, lo que produciría la salida del bucle.

También podría construirse un *while* infinito, por ejemplo con condición de carrera `[ 1 ]`, que siempre sería verdadera, y podríamos forzar la salida en algún momento con una instrucción *exit* o *break*.

2.1. Construcción típica

Una construcción típica de *while* es:

```
#!/bin/bash

CONTADOR=1

while [ $CONTADOR -lt 10 ]
do
    echo "El contador vale ${CONTADOR} y es menor que 10."
    let CONTADOR++
done
```

[1] Bloque de código que se ejecutará en diversas ocasiones y/o que puede ser llamado desde otras partes del programa.
[2] Condición que de cumplirse marca el fin de un bucle.
[3] Por ejemplo de 1 a 10 o de la "a" a la "z".
[4] Por ejemplo un fichero, quizá el resultado de un *cat*, siendo cada línea del fichero objeto de una iteración del bucle.
[5] Programación de guiones de Bash.
[6] Cada ejecución del bucle o rutina.

Este programa producirá la siguiente salida:

```
El contador vale 1 y es menor que 10.
El contador vale 2 y es menor que 10.
El contador vale 3 y es menor que 10.
El contador vale 4 y es menor que 10.
El contador vale 5 y es menor que 10.
El contador vale 6 y es menor que 10.
El contador vale 7 y es menor que 10.
El contador vale 8 y es menor que 10.
El contador vale 9 y es menor que 10.
```

Como se puede observar el bucle ha sido ejecutado 9 veces. Como *$CONTADOR* es igual a 1, cumple la condición de ser menor de 10, así que el bucle comienza a ejecutare. Lo primero que hacemos es un *echo* diciendo cuando vale el contador a la entrada del bucle y acto seguido lo incrementamos (`let CONTADOR++`) y este proceso se repite hasta que contador ha sido incrementado hasta 10, con lo que incumple la condición de carrera.

2.2. Bucle infinito

En programación un bucle infinito se entiende como un error, si hay un bucle infinito en un programa esté se quedará *colgado* en dicho bucle y no continuará nunca. Sin embargo podemos aprovecharnos los bucles *pseudoinfinitos* para conseguir que un programa esté continuamente funcionando hasta que forcemos la salida con una instrucción *break*. Por ejemplo:

```
#!/bin/bash

# La condición [ 1 ] es siempre verdadera.
while [ 1 ]
do
    echo "¿Hay alguien ahí?"
    read ENTRADA

    if [ "$ENTRADA" == "si" ]
    then
        echo "Hola, usuario."
    elif [ "$ENTRADA" == "no" ]
    then
        echo "Pues si no hay nadie me voy."
        break
    else
        echo "No entiendo."
    fi
done

echo "Fin de programa."
```

Este programa estará todo el rato preguntando al usuario que si hay alguien ahí. Sólo se detendrá al contestar "*no*", respuesta que causará la salida del bucle

mediante la instrucción *break*. Observe que la línea final `echo "Fin de programa."` será ejecutada normalmente, ya que sólo hemos ordenado la salida del bucle. Si por el contrario hubiésemos utilizado una instrucción "*exit*", la salida hubiese sido total, de todo el programa, y esta última línea nunca llegaría a ejecutarse.

Recuerde: Utilize **break** para salir de un bucle ignorando la condición de carrera.

3. Bucles *for*

Los bucles de tipo *for* recorren una secuencia de principio a fin, si no se sale con un *break*, e introducen en una variable el valor de la actual iteración.

Son útiles para recorrer secuencias de números, líneas de un fichero de texto, expresiones matemáticas, etc...

La estructura de un bucle *for*, en pseudocódigo, sería:

```
para VAR de 1 a 20
hacer
    decir "Estoy en la iteración número VAR"
hecho

decir "Programa finalizado."
```

O en código real de **Bash**:

```
#!/bin/bash

for VAR in $(seq 1 20)
do
    echo "Estoy en la iteración número $VAR"
done

echo "Programa finalizado."
```

Como se verá el programa real es muy parecido al de pseudocódigo, prácticamente sólo cambia en que hemos introducido el comando *seq* para generar una secuencia de 1 a 20. Si desea ver que hace, haga la prueba en el terminal:

```
#> seq 1 10
1
2
3
4
5
6
7
8
9
10
```

3.1. Construcción típica

Vamos a ver tres construcciones muy típicas, la que recorre una secuencia de números y la que recorre las líneas de un fichero, y adicionalmente veremos como ejecutar un bucle *for* en la **consola**, sin escribir un *script*.

3.1.1. *For* sobre una secuencia de números

```
#!/bin/bash

for VAR in $(seq 1 20)
do
    echo "Estoy en la iteración número $VAR"
done

echo "Programa finalizado."
```

Lo que hará *bash* al ejecutar ese programa es, por cada uno de los números de la secuencia, introducirlo en la variable *VAR* y ejecutar lo que hay entre las líneas *do* y *done*. En el caso de este programa el contenido entre estas líneas será ejecutado veinte veces. El resultado por pantalla será:

```
Estoy en la iteración número 1
Estoy en la iteración número 2
Estoy en la iteración número 3
Estoy en la iteración número 4
Estoy en la iteración número 5
Estoy en la iteración número 6
Estoy en la iteración número 7
Estoy en la iteración número 8
Estoy en la iteración número 9
Estoy en la iteración número 10
Estoy en la iteración número 11
Estoy en la iteración número 12
Estoy en la iteración número 13
Estoy en la iteración número 14
Estoy en la iteración número 15
Estoy en la iteración número 16
Estoy en la iteración número 17
Estoy en la iteración número 18
Estoy en la iteración número 19
Estoy en la iteración número 20
Programa finalizado.
```

3.1.2. *For* sobre las líneas de un fichero

```
#!/bin/bash

echo "Nombres y edades:"
echo "-----------------"
```

```
for LINEA in $(cat nombres-edades.txt)
do
        NOMBRE=$(echo $LINEA|cut -f1 -d":")
        EDAD=$(echo $LINEA|cut -f2 -d":")
        echo "$NOMBRE tiene $EDAD años."
done

echo
echo "Programa finalizado."
```

Siendo el fichero "*nombres-edades.txt*" algo como esto:

```
Pepe:23
Paula:34
Juan:27
María:54
Federico:50
Pablo:32
Sonia:33
Elena:60
```

El programa recorrerá cada una de las líneas del fichero y la pondrá en la variable *LINEA*, pasando a ejecutar el bucle. Dentro del bucle se ejecutan dos comandos *cut* para obtener por separado el nombre y la edad. Lo que hace *cut* es cortar la línea por el seaparador, en este caso "*:*" y darnos el campo que solicitamos en las opciones, si es *-f1* nos dará el primer campo, es decir el nombre.

Al ejecutar obtendremos lo siguiente por pantalla:

```
#> sh edades.sh

Nombres y edades:
-----------------
Pepe tiene 23 años.
Paula tiene 34 años.
Juan tiene 27 años.
María tiene 54 años.
Federico tiene 50 años.
Pablo tiene 32 años.
Sonia tiene 33 años.
Elena tiene 60 años.

Programa finalizado.
```

3.2. Bucle *for* sobre la consola o shell

En muchas ocasiones necesitaremos ejecutar un bucle en la consola, mientras estamos trabajando, un bucle que sólo vamos a utilizar una vez y para el que no hay necesidad de crear un guión.

Para eso podemos teclearlo en la línea de comandos, separando las líneas por punto y coma (**;**) en lugar de por *intros*, por ejemplo:

```
#> for i in $(seq 1 10); do echo "Iteración número $i";
echo "Podríamos ejecutar cualquier cosa aquí."; done
```

Note que por problemas de espacio el bucle aparece en dos líneas, pero habría que teclearlo en una sola.

El ejemplo es muy sencillo, para no confundir, pero cualquier bucle podría ser ejecutado de esta forma. Por ejemplo uno que recorra un fichero y cree cuentas de usuario con los nombres allí contenidos.

4. Los bucles *until*

Los bucles *until* son los antónimos de los bucles *while*. Es decir que son exactamente iguales pero al revés. Para que un bucle *until* se ejecute, la condición de carrera debe ser falsa, y para que deje de ejecutarse debe pasar a ser verdadera.

Por ejemplo:

```
#!/bin/bash

CONTADOR=1

until [ $CONTADOR -eq 10 ]
do
    echo "El contador vale ${CONTADOR} y es menor que 10."
    let CONTADOR++
done
```

El bucle anterior se ejecutará hasta que la variable *CONTADOR* pase a valer 10:

```
#> sh prueba-until.sh

El contador vale 1 y es menor que 10.
El contador vale 2 y es menor que 10.
El contador vale 3 y es menor que 10.
El contador vale 4 y es menor que 10.
El contador vale 5 y es menor que 10.
El contador vale 6 y es menor que 10.
El contador vale 7 y es menor que 10.
El contador vale 8 y es menor que 10.
El contador vale 9 y es menor que 10.
```

5. Ejercicios

5.1. Enunciados

Cree un script o guión de *Bash* para cada uno de los ejercicios, y llámelo *ej-bucles-num.sh*, siendo *num* el número de ejercicio:

1. Escriba un guión que admita palabras (seguidas de *enter*) por parte del usuario hasta que este teclee "*salir*", después debe aparecer en pantalla el número de palabras que se han introducido y una despedida.

2. Modifique el programa anterior de forma que las palabras se vayan guardando en un fichero "*palabras.txt*".

3. Modifique de nuevo el programa. Consiga que si el usuario introduce un número en lugar de una palabra, se le devuelva la palabra que introdujo en esa posición. Por ejemplo si introduce "5", se le devolverá la palabra número cinco del fichero "*palabras.txt*".

4. Escriba un programa que calcule el resultado de sumar todos los números del 1 al 100 secuencialmente, es decir: 1 + 2 + 3 + 4 + 5...

5. **Medio**: Cree un script que pida números al usuario hasta que este teclee la palabra "*promedio*". Después calcule la media aritmética[7] entre todos ellos.

6. Cree un fichero que contenga una lista de cinco nombres de pila y, separadas por una coma, una edad expresada en años. Por ejemplo una línea sería "Juan,24". Guarde ese fichero como "**nombre-edad.txt**" y escriba un guión que lo recorra y diga por pantalla "*Tal_nombre* tiene *tantos* años." para cada una de las líneas, por ejemplo para la anterior diría "*Juan tiene 24 años.*".

[7] http://es.wikipedia.org/wiki/Media_aritm%C3%A9tica

7. Utilizando el fichero del ejercicio anterior haga un guión que diga quién de la lista es el mayor y el menor.

8. Con la misma lista, haga un guión que cree un usuario para cada uno de ellos en su máquina, el grupo debe ser "*ejercicios*", y que haga después un tar.gz (uno sólo) de todos los directorios de usuario de la máquina (excluyendo a root) con el nombre *homes-backup.tgz* y que se guarde en */root/backups*.

9. Cree un script que borre todos los usuarios pertenecientes al grupo "*ejercicios*" pero antes de borrar a cada uno debe mostrar su nombre por pantalla y pedir confirmación.

10. Haga un guión que mire todos los directorios que haya dentro de */home* y que por cada uno de ellos compruebe si existe dicho usuario en el */etc/passwd*, si el usuario no existe debe dar la opción de comprimir en un backup y borrar el directorio de ese usuario fantasma. Antes de probarlo cree varios directorios dentro de */home*.

11. Escriba un guión que repase todos los ficheros de ejercicios que ha hecho hasta ahora, y que compruebe que la primera línea de cada uno de ellos es "*#!/bin/bash*".

12. Escribe un programa que dado un nombre de paquete de Debian, compruebe si está instalado o no, y si no lo está de la opción de obtener más información sobre él o de instalarlo.

13. Construya un programa que escriba por pantalla una secuencia de 1 a 99, utilice un bucle until para ello.

5.2. Soluciones

1. Escriba un guión que admita palabras (seguidas de *enter*) por parte del usuario hasta que este teclee "*salir*", después debe aparecer en pantalla el número de palabras que se han introducido y una despedida.

```
#!/bin/bash

# Variables:
CONTADOR=0
PALABRA=""

# Limpiar pantalla:
clear

# Bucle:
while [ "$PALABRA" != "salir" ]
do
    # Entrada de usuario:
    read -p "Introduzca palabra: " PALABRA
    let CONTADOR++
done

# Quitamos 1 por "salir":
let CONTADOR--

# Total de palabras:
echo "Has tecleado $CONTADOR palabras."
```

2. Modifique el programa anterior de forma que las palabras se vayan guardando en un fichero "*palabras.txt*".

```
#!/bin/bash

clear

# Variables:
CONTADOR=0

# Creamos o borramos el fichero auxiliar:
cat /dev/null > palabras.txt

while [ 1 ]
do
    # Entrada de usuario:
    read -p "Introduce palabra: " PALABRA

    # Convertir a minúsculas:
```

```
    PALABRA=$(echo $PALABRA|tr "A-Z" "a-z")

    # Salir del bucle si PALABRA es "salir":
    if [ "$PALABRA" == "salir" ] || [ -z "$PALABRA" ]
    then
        break
    else
        # Sumamos 1 palabra:
        let CONTADOR++
        # Agregar palabra al fichero:
        echo $PALABRA >> palabras.txt
    fi
done

echo "Has introducido $CONTADOR palabras."
```

3. Modifique de nuevo el programa. Consiga que si el usuario introduce un número en lugar de una palabra, se le devuelva la palabra que introdujo en esa posición. Por ejemplo si introduce "5", se le devolverá la palabra número cinco del fichero "*palabras.txt*".

```
#!/bin/bash

# Limpiar pantalla:
clear

# VariableS:
DATO=""
CON=0

# Instrucciones:
echo "Introduzca las palabras que quiera, que las memorizaré."
echo "Pulsa ('salir') para terminar."

# Borrar fichero auxiliar:
cat /dev/null > palabras.txt

# Bucle:
while [ "$DATO" != "salir" ]
do
    read -p "¿Palabra a guardar? " DATO

    # Comprobar, mediante exp.regular si es un número:
    if [[ $DATO =~ [0-9] ]]
    then
        # Comprobar que la palabra solicitada es de
        # que ya han sido memorizadas:
        if [ $DATO -gt $CON  ]
        then
```

```
            echo -e "\nERROR: No has introducido tantas palabras.\n"
        else
            cat palabras.txt|grep -E "^${DATO}"|cut -f2 -d":"
        fi
    else
        if [ "$DATO" != "salir" ]
        then
            # Cuento una palabra más:
            let CON++
            # Guardo la palabra con su número correspondiente:
            echo "$CON:$DATO" >> palabras.txt
        fi
    fi
done

echo -e "\nTecleaste $CON palabras.\nAdiós.\n"
```

4. Escriba un programa que calcule el resultado de sumar todos los números del 1 al 100 secuencialmente, es decir: 1 + 2 + 3 + 4 + 5...

```
#!/bin/bash

# VariableS:
RES=0

# Bucle:
for VAR in $(seq 1 100)
do
    echo "Sumando $RES + $VAR."
    # Sumar VAR a lo anterior:
    RES=$(expr $RES + $VAR)
done

# Resultado:
echo " El resultado es $RES."
```

5. **Medio**: Cree un script que pida números al usuario hasta que este teclee la palabra "*promedio*". Después calcule la media aritmética[8] entre todos ellos.

```
#!/bin/bash

clear

# Variables:
CONTADOR=0
```

[8] http://es.wikipedia.org/wiki/Media_aritm%C3%A9tica

```
SUMA=0

while [ 1 ]
do
    # Entrada de usuario:
    read -p "Diga un número: " NUMERO

    # Comprobar si nos piden el promedio:
    if [ "$NUMERO" = "promedio" ] || [ "$NUMERO" = "PROMEDIO" ]
    then
        # Romper bucle:
        break
    fi

    let CONTADOR++
    SUMA=$(expr $SUMA + $NUMERO)
done

echo "Ha introducido $CONTADOR numeros."

PROMEDIO=$(echo "scale=2; $SUMA / $CONTADOR"|bc|tr "." ",")

echo "El promedio es $SUMA entre $CONTADOR igual a ${PROMEDIO}."
```

6. Cree un fichero que contenga una lista de cinco nombres de pila y, separadas por una coma, una edad expresada en años. Por ejemplo una línea sería "Juan,24". Guarde ese fichero como "**nombre-edad.txt**" y escriba un guión que lo recorra y diga por pantalla "*Tal_nombre* tiene *tantos* años." para cada una de las líneas, por ejemplo para la anterior diría "*Juan tiene 24 años.*".

```
#!/bin/bash

clear

for LINEA in $(cat nombre-edad.txt|tr " " "_")
do
NOMBRE=$(echo $LINEA|cut -f1 -d",")
EDAD=$(echo $LINEA|cut -f2 -d",")
echo "$(echo $NOMBRE|tr "_" " ") tiene $EDAD años."
done
```

7. Utilizando el fichero del ejercicio anterior haga un guión que diga quién de la lista es el mayor y el menor.

```
#!/bin/bash

clear
```

```
# Variables:
EDADMENOR=999
EDADMAYOR=0
NOMBREMENOR=""
NOMBREMAYOR=""

# Bucle:
for LINEA in $(cat nombre-edad.txt|tr " " "_")
do
    NOMBRE=$(echo $LINEA|cut -f1 -d",")
    EDAD=$(echo $LINEA|cut -f2 -d",")
    echo "$(echo $NOMBRE|tr "_" " ") tiene $EDAD años."

    if [ $EDAD -lt $EDADMENOR ]
    then
        EDADMENOR=$EDAD
        NOMBREMENOR=$NOMBRE
    fi

    if [ $EDAD -gt $EDADMAYOR ]
    then
        EDADMAYOR=$EDAD
        NOMBREMAYOR=$NOMBRE
    fi
done

echo
echo "El menor es $(echo $NOMBREMENOR|tr "_" " ") con $EDADMENOR años."
echo "El mayor es $(echo $NOMBREMAYOR|tr "_" " ") con $EDADMAYOR años."
echo
```

8. Con la misma lista, haga un guión que cree un usuario para cada uno de ellos en su máquina, el grupo debe ser "*ejercicios*", y que haga después un tar.gz (uno sólo) de todos los directorios de usuario de la máquina (excluyendo a root) con el nombre *homes-backup.tgz* y que se guarde en */root/backups*.

```
#!/bin/bash

clear

# Comprobar si exsiste el grupo:
cat /etc/group|grep "ejercicios" &> /dev/null
if [ $? -ne 0 ]
then
    echo "Añadiendo grupo ejercicios."
    addgroup ejercicios
fi
```

```
# Crear directorio de backups:
mkdir -p /root/backups

# Bucle:
for i in $(cat nombre-edad.txt|tr " " "-"|tr "A-Z" "a-z")
do
    NOMBRE=$(echo $i|cut -f1 -d",")
    echo "Creando usuario $NOMBRE..."
    useradd -g ejercicios -m $NOMBRE
    if [ $? -eq 0 ]
    then
        echo "Creado correctamente."
    else
        echo "ERROR creando usuario."
        # exit 1
    fi
done

echo -e "FIN de creación de usuarios.\n\n"

echo -n "Creando copia de seguridad de /home..."
tar czf /root/backups/homes-backup.tgz /home

if [ $? -eq 0 ]
then
    echo "OK"
else
    echo "ERROR"
    exit 1
fi
```

9. Cree un script que borre todos los usuarios pertenecientes al grupo "*ejercicios*" pero antes de borrar a cada uno debe mostrar su nombre por pantalla y pedir confirmación.

```
#! /bin/bash

clear

# Bucle:
for linea in $(cat /etc/passwd|tr " " "_")
do
    # Si el usuario tiene UID < 1000 es de sistema
    # y hay que saltarlo.
    USUUID=$(echo $linea|cut -f3 -d":")
    if [ $USUUID -lt 1000 ]
        then
        echo "$USUUID es un usuario de sistema."
```

```
    else
        usuario=$(echo $linea|cut -f1 -d":")
        # Sacar grupos del usuario y ver si tiene 'ejercicios':
        groups $usuario|grep "ejercicios" &> /dev/null
        if [ $? -eq 0 ]
        then
            read -n 1 -p "¿Borrar al usuario ${usuario}? (Y/N): " YN
            echo
                        if [[ "$YN" == "y" || "$YN" == "Y" ]]
            then
                echo -n "Borrando usuario $usuario..."
                userdel -r $usuario
                if [ $? -eq 0 ]
                then
                    echo "OK"
                else
                    echo "ERROR"
                fi
            fi
        fi
    fi
done
```

10. Haga un guión que mire todos los directorios que haya dentro de */home* y que por cada uno de ellos compruebe si existe dicho usuario en el */etc/passwd*, si el usuario no existe debe dar la opción de comprimir en un backup y borrar el directorio de ese usuario fantasma. Antes de probarlo cree varios directorios dentro de */home*.

```
#!/bin/bash

clear

for DIR in $(ls -C1 /home)
do
echo "Comprobando directorio $DIR..."

cat /etc/passwd | grep $DIR &> /dev/null

if [ $? -eq 0 ]
then
# El usuario existe.
echo "El usuario existe."
else
# El usuario no existe.
echo "El usuario $DIR no existe."
read -p "¿Comprimimos el directorio y lo eliminamos? (s/n): " \
-n1 SN
echo
```

```
if [[ "$SN" == "s" || "$SN" == "S" ]]
then
# Por tanto hay que borrar el directorio fantasma,
# comprimiendo un backup previamente.
echo "Comprimiendo backup de $DIR..."
tar czf /root/backups/$DIR.tgz /home/$DIR
echo "Borrando /home/$DIR..."
rm -Rf /home/$DIR
echo "El directorio $DIR ha sido comprimido y eliminado."
fi
fi

done
```

11. Escriba un guión que repase todos los ficheros de ejercicios que ha hecho hasta ahora, y que compruebe que la primera línea de cada uno de ellos es "*#!/bin/bash*".

```
#!/bin/bash

clear

echo "Comprobando forma 1."
echo

for nombre in $(ls ej-bucles*)
do
    linea=$(cat $nombre|head -n 1)
    if [ "$linea" != "#!/bin/bash" ]
    then
        echo El archivo $nombre no tiene la primera línea correcta.
    fi
done

echo -e "\nComprobando forma 2:\n"

for nombre in $(ls ej-bucles*)
do
    head -n1 $nombre|grep "\#\!/bin/bash" &> /dev/null
    if [ $? -ne 0 ]
    then
        echo "La primera línea de $nombre es incorrecta."
    fi
done
```

12. Escribe un programa que dado un nombre de paquete de Debian, compruebe si está instalado o no, y si no lo está de la opción de obtener más información sobre él o de instalarlo.

```
#! /bin/bash

clear

read -p "Introduzca el nombre del paquete: " nombre

aptitude search $nombre|grep -E "^i.*$nombre\ " &> /dev/null

if [ $? -ne 0 ]
then
    echo el paquete $nombre no está instalado
    read -n1 -p "Quiere más información del paquete (P) o desea instalarlo (I): " opc
    echo
    if [ "$opcion" == "P" ]
    then
        echo información sobre el paquete $nombre
        aptitude show $nombre
    else
        echo instalando el paquete $nombre
        aptitude install $nombre
    fi
else
    echo el paquete $nombre  está instalado
fi
```

13. Construya un programa que escriba por pantalla una secuencia de 1 a 99, utilice un bucle until para ello.

```
#!/bin/bash

clear

contador=1

until [ $contador -eq 100 ]
do
    echo -n "${contador} "
    let contador++
done

echo -e "\nFin de programa.\n"
```

6. Sobre esta unidad didáctica

6.1. Notas y advertencias

Debian: Esta guía está basada en el sistema *Debian GNU/Linux*, podría haber pequeños cambios si se aplica a otras distribuciones de *GNU*, pero en su mayor parte funcionará bien con la excepción de lo referido al sistema de paquetería de programas, los comandos que empiezan por *apt*, ya que otras *distros* no basadas en *Debian* podrían incorporar sistemas diferentes para el manejo de sus paquetes.

6.2. Derechos

Esta guía se cede bajo contrato Coloriuris. Sólo puede ser utilizada previa aceptación del contrato de cesión sito en:

- http://www.coloriuris.net/contratos/ef5af6aaa441ab9c213273fade56dca1

Dicho contrato garantiza que estoy cediendo los derechos de uso y modificación sin ánimo de lucro.

6.3. Agradecimientos

El autor quiere reflejar su agradecimiento a todas las páginas de Internet que ponen a disposición de todo el mundo sus contenidos, así como a todo aquél que publica artículos, manuales y experiencias en Internet, ya que eso favorece a la difusión del conocimiento y al desarrollo humano. *La información quiere ser libre.*

Un agradecimiento muy especial a toda la comunidad del Software Libre. Sin ellos el autor viviría en la oscuridad: Programadores, traductores, asociaciones, hacktivistas, webmasters, etc...

También quiero agradecer muy especialmente su ayuda a mis alumnos y lectores, por tomarse la molestia de comunicarme las erratas y por darme ideas para mejorar los ejercicios.

6.4. Revisiones

El autor irá eventualmente publicando revisiones de esta unidad en su página personal, y estará encantado de recibir sugerencias y dudas en la misma o en su email:

- http://jorgefuertes.com.
- cursos@jorgefuertes.com.

Por supuesto se puede contactar con el autor para contratarle para hacer nuevas unidades, adaptaciones, modificaciones, cursos, etc...