Área de Arquitectura y Tecnología de Computadores

Universidad Carlos III de Madrid

SISTEMAS OPERATIVOS

Ejercicio. Programación en `bash`

Grado de Ingeniería en Informática

Curso 2016/2017

AUTORES (Nombre, Apellidos, NIA y Grupo)

1. Evaluación del ejercicio

Los alumnos deberán responder de manera correcta y razonada a las preguntas formuladas. Así mismo, deberán indicar, al comienzo del cuadernillo, su nombre, apellidos, NIA y grupo. Aquellos alumnos que no respondan a las preguntas **justificando** las respuestas, tendrán suspenso el ejercicio.

Tiempo: 80 minutos

2. Descripción del ejercicio

Los shell scripts son programas interpretados y escritos usando mandatos que permiten el uso de variables y sentencias de control.

La principal ventaja que presentan los shell scripts es que pueden ser portados de una máquina *UNIX* a otra sin problemas, sin necesidad de retocar nada, salvo que se utilicen llamadas a programas muy concretos específicos de alguna versión de *UNIX*, mientras que los programas compilados (desarrollados en *C*, *Pascal*, etc.) deben ser recompilados, pues el código se generará en función del microprocesador de cada máquina. Otra ventaja es la facilidad de lectura e interpretación.

El principal inconveniente que presentan respecto a los programas compilados es la lentitud de ejecución, que se puede paliar usando funciones incluidas en el propio entorno en lugar de llamar a mandatos externos.

Los *scripts* suelen encabezarse con comentarios que indican el nombre de archivo y lo que hace el *script*. Se colocan comentarios de documentación en diferentes partes del *script* para mejorar la comprensión y facilitar el mantenimiento. Un caso especial es el uso de **'#'** en la primera línea, seguido del carácter admiración (**'!'**) y la ruta de la *subshell*, para indicar el intérprete con que se ejecutará el script:

Ejemplo:

#!/bin/bash

Es interesante saber que muchos mandatos devuelven un valor después de ejecutarse, y que este valor indicará si la ejecución ha sido buena o si ha habido algún fallo y qué tipo de fallo se ha producido. Para conocer si un mandato devuelve o no un valor y qué es lo que devuelve en cada caso se deberá consultar la documentación (utilidad man), pero por lo general en caso de una ejecución correcta devolverán el valor 0, y en caso de fallo otro número, positivo o negativo.

NOTAS:

Para poder ejecutar un shell script es necesario que tenga activados, al menos, los permisos de lectura y ejecución (ver mandato chmod para cambiar los permisos). El siguiente ejemplo otorga permisos de ejecución al fichero 'script.sh'.

chmod +x script.sh

Para poder depurar se puede usar la opción **–v** a la hora de definir el *shell*.

Ejemplo:

#!/bin/bash –v

O también:

#!/bin/bash set

-x

Hay que tener mucho cuidado a la hora de establecer asignaciones y el uso de expresiones. Los espacios son significativos en muchos casos:

- Esta asignación funciona **LONGITUD=` echo $NOMBRE | wc -c `**
- Esta asignación no funciona **LONGITUD =` echo $NOMBRE | wc -c `** ya que tiene en cuenta que la variable **LONGITUD** tiene un espacio.

Algo parecido hay que tener en cuenta sobre las sentencias de control ya que éstas implican usar líneas diferentes para cada parte. Ejemplo:

if [$# -gt 0]

then

NOMBRE = $1

fi

EJERCICIO 1.

Escribir en un editor de texto el siguiente programa, y ejecutarlo en una shell:
./ejercicio1.sh

```
#!/bin/bash
VALOR= 3
while [ 1 –le $VALOR ];
do
      echo “$VALOR”
      sleep $VALOR
      VALOR=`expr $VALOR + 1`
done
```

Responda brevemente:

a. Describir el funcionamiento del programa.

El programa es un contador que comienza en 3 y **nunca finaliza**. Para cada iteración del contador se realiza un sleep que duerme al proceso, al menos, el tiempo especificado por el contador, a la vez que se incrementa su valor para la siguiente iteración.

b. ¿Para qué se utiliza la sentencia **while [1 –le $VALOR];** ?

La sentencia es un bucle que itera sobre una variable ‘valor’ que inicialmente tiene un valor de 3. El bucle itera tantas veces como se cumpla la condición (1 es menor o igual que ‘valor’).

c. ¿Qué ocurre si se cambia **–le** por **–gt** ?

Si se modifica la condición de iteración, en este caso el programa ya no sería un bucle infinito, ya que directamente no se cumpliría la condición (1 es mayor o igual que ‘valor’), finalizando sin realizar acciones.

b. Describa brevemente para qué se utilizan los símbolos **$** y **#** en los scripts de shell.

El símbolo $ se utiliza para que la Shell reconozca que la palabra que le precede es una variable.
El símbolo # se utiliza para indicar que lo que continúa es un comentario de línea.

EJERCICIO 2.

Escribir en un editor de texto el siguiente programa y ejecutarlo en una Shell:
./ejercicio2.sh

```
#!/bin/bash
echo “Inserte una cadena:”
read CADENA
LONGITUD=` echo $CADENA | wc –c `
while [ $LONGITUD –gt 0 ]
do
      CADENA2=”$CADENA”`echo $CADENA | cut –c$LONGITUD`
      LONGITUD=`expr $LONGITUD – 1`
done
echo “$CADENA2”
```

Responda brevemente:

a. ¿Qué operación realiza el programa?

El programa realiza una operación de inversión de una cadena: da la vuelta a los caracteres de la cadena, almacenando éstos en otra variable que al final se muestra por pantalla.

b. ¿Para qué sirve **wc –c**? ¿y **cut –c**?

El mandato wc se utiliza para contar (por defecto es un contador de palabras [Word count]), en este caso el flag -c indica que se cuentan bytes (caracteres). Es decir, calcula la longitud de una cadena.
El mandato cut es un extractor de información (similar a un Split), que junto al flag -c tiene la función de hacer una división de la cadena por un byte que se debe indicar.

c. Modifique el programa para que reciba la cadena directamente desde un argumento que se pase al programa.

```
#!/bin/bash
CADENA=$1
LONGITUD=` echo $CADENA | wc –c `
while [ $LONGITUD –gt 0 ]
do
      CADENA2=”CADENA”`echo $CADENA | cut –c$LONGITUD`
      LONGITUD=`expr $LONGITUD – 1`
done
echo “$CADENA2”
```

EJERCICIO 3.

Escribir un programa en C (con extensión .c) que contenga el siguiente código, e insertar 3 copias en una carpeta llamada e3_test que estará situada en el mismo directorio que el script:

```
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char* argv[]){
      printf("Programa compilado con un shellscript\n");
      return 0;
}
```

En una cuarta copia se debe suprimir la línea de retorno return 0. Esto provocará un error en la compilación que será útil para su posterior verificación.

Escribir en un editor de texto un programa ejecutable de tipo shellscript:

./ejercicio3.sh

El programa debe tener las siguientes características:

a. Se recibe por argumento un directorio del sistema de ficheros que contiene ficheros de tipo C. Se proporciona un ejemplo, pero se recomienda probar con más.
b. Se asume que el directorio pasado existe y únicamente contiene ficheros tipo C (extensión .c)
c. Se debe obtener un listado del directorio y recorrer cada elemento realizando su compilación con la instrucción:
   gcc –Wall –Werror –c nombre_fichero.c
d. Los ficheros compilados deben estar incluidos en la misma carpeta que los ficheros C.
e. Al final de la ejecución, se debe mostrar el número de ficheros compilados sin error, y se mostrarán los ficheros objeto (los ya compilados, que tienen extensión .o).
   i. Para saber si la ejecución del último comando ha sido exitosa, se puede consultar el valor de $?, teniendo un valor de 0 si todo es correcto.

SOLUCIÓN:

```
#!/bin/bash
DIRECTORIO=`ls $DIRECTORIO`
COUNT=0
for file in $LISTADO
do
        NEWFILE=`echo "$file" | cut -d'.' -f1`
        Gcc -Wall -Werror -c "$DIRECTORIO/$file" -o "$DIRECTORIO/$NEWFILE.o"
if [ $? -eq 0 ]
then
                COUNT=`expr $COUNT + 1`
fi
done
echo "Numero de ficheros compilados con éxito = $COUNT"
echo `ls $DIRECTORIO | grep ".o"`
```