C.F.G.S. DESARROLLO DE APLICACIONES INFORMÁTICAS

MÓDULO:

Sistemas Informáticos

Programación Shell

INDICE DE CONTENIDOS

1. INTRODUCCIÓN

Un Shell script es un programa que interpreta órdenes. Antes de ejecutar un Shell script hay que escribirlo con el editor de textos vi.

Los Shell scripts son muy útiles para ciertos tipos de tareas:

- Tareas administrativas: algunas partes de los sistemas UNIX son Shell scripts, para poder entenderlos y modificarlos es necesario tener alguna noción sobre la programación de scripts.
- Tareas tediosas que solo se van a ejecutar una o dos veces, no importa el rendimiento del programa resultante pero si conviene que su programación sea rápida.
- Hacer que varios programas funcionen como un conjunto de una forma sencilla.
- Pueden ser un buen método para desarrollar prototipos de aplicaciones más complejas que posteriormente se implementarán en leguajes más potentes.
- Conocer a fondo la shell aumenta tremendamente la rapidez y productividad a la hora de usarla, incluso fuera de los scripts.

AUTOEVALUACIÓN

Un Shell script es…

a) Un virus informático

b) Una serie de instrucciones en lenguaje C

c) Un programa que interpreta órdenes

d) Ninguna es correcta

2. VARIABLES

Las variables se definen de forma similar a como se hace en cualquier lenguaje de programación.

Para escribir los nombres de las variables se diferencia entre mayúsculas y minúsculas. Y no pueden empezar por un número.

Para acceder a contenido de una variable, se utiliza el símbolo $.

El valor a una variable se puede dar de forma directa. Si el valor es alfanumérico, tendrá que ir entre comillas.

Sintaxis

```
variable = valor_numérico
variable = “valor_alfanumérico”
variable_nueva = $ variable
```

ACONSEJABLE: Para una buena organización, crear los Shell scripts en el directorio “/home/usuario1/Documents/shells”

1 EJEMPLO

Realizar un Shell script llamado “prueba” para crear dos variables: Una con un contenido numérico y otra con un contenido alfanumérico. Al final visualizar sus contenidos.

Los pasos a seguir son:

- Escribir las órdenes del Shell “prueba” con el editor de textos vi.

  ```
  vi prueba
  var1=12
  var2=”datos de var2”
  echo “var1=$var1”
  echo “var2=$var2”
  ```

- Salir del Shell script guardando los cambios

  ```
  Esc
  :wq
  ```

- Ejecutar el Shell script “prueba”

  ```
  sh prueba
  ```

var1=12

var2=datos de var2

2 AUTOEVALUACIÓN

Es cierto que:

a) Para escribir los nombres de las variables en los Shell strip de Unix se diferencia entre mayúsculas y minúsculas. Y no pueden empezar por un número

b) Para escribir los nombres de las variables en los Shell strip de Unix no se diferencia entre mayúsculas y minúsculas. Y no pueden empezar por un número

c) Para escribir los nombres de las variables en los Shell strip de Unix se diferencia entre mayúsculas y minúsculas. Y pueden empezar por un número

3. ÓRDENES BÁSICAS

3.1. read

Se utiliza para introducir un valor desde teclado.

Sintaxis read variable

2 EJEMPLO

Crear una variable “nombre” en la que se pide el nombre del usuario. Y al final visualizar ese nombre.

#Shell visualizar

echo “Introduce tu nombre:”

read nombre

echo “El nombre que has introducido es: $nombre”

3.2. readonly

Se utiliza para hacer que una variable tenga un valor fijo durante toda la sesión.

Sintaxis readonly variable

3 EJEMPLO

Crear una variable con un valor fijo y mostrar su contenido. A continuación intentar modificar el valor de esa variable.

- #Shell visualizar1

  var1=”Prueba”

  readonly var1

  echo “var1 ahora vale: $var1”

  var1 ahora vale: Prueba

- Si introducimos la siguiente línea:

  var1=”Prueba”

  readonly var1

  var1=”Cambiar”

  echo “var1 ahora vale: $var1”

  var1: readonly variable

  Nos muestra ese error indicando que var1 es una variable de sólo lectura, entonces no puede cambiar

3 AUTOEVALUACIÓN

La orden readonly se utiliza:

a) Pedir el valor de una variable por teclado

b) Hacer que una variable tenga un valor fijo durante toda la sesión

c) Inicializar el valor de una variable

3.3. echo

Sirve para visualizar cadenas de caracteres o contenido de variables.

Sintaxis

echo texto

~~Opciones: -~~

- ‘texto’: Interpreta todo como texto (‘ se obtiene pulsando la tecla que

tiene el caráncter?)

- “texto”: Interpreta el símbolo $

4 EJEMPLO

Probar el siguiente Shell Script:

```
#Shell orden_echo
var1=prueba
echo “Esto visualizará $var1”
echo ‘Esto visualizará $var1’
```

Esto visualizará prueba

Esto visualizará $var1

3.4. expr

Se utiliza para realizar operaciones aritméticas: +, -, *, /, %.

Si el valor del resultado lo queremos asignar a otra variable, tendremos que introducir la expresión aritmética entre comillas simples.

Sintaxis

resultado = `expr $variable1 operador $variable2 ….`

Operador:

- +: Suma
- -: Resta
- *: Multiplicación
- /: División
- %: Resto de la división

En este tipo de operaciones se sigue la prioridad de las operaciones aritméticas: Primero división o multiplicación y después suma o resta.

` `: Se obtienen pulsando la tecla que tiene el carácter “[“ mas un espacio

5 EJEMPLO

Probar el siguiente Shell Script en donde se realizan operaciones aritméticas con tres variables.

```
#Shell exprexion

a=2; b=3; c=4

resultado1=`expr $a + $b`

resultado2=`expr $a \* $b + $c`

#1º multiplica

echo “El valor del resultado1 es: $resultado1”

echo “El valor del resultado2 es: $resultado2”

El valor del resultado1 es: 5

El valor del resultado2 es: 10
```

4 AUTOEVALUACIÓN

La orden expr se utiliza para:

a) Realizar operaciones aritméticas: +, -

b) Realizar operaciones aritméticas: *, /

c) Realizar operaciones aritméticas: /, %

d) Realizar operaciones aritméticas: +, -, *, /, %

4. PARÁMETROS

Podemos pasar parámetros en Unix. Y estos van desde: $1, $2, …., $9

Además de los números del 1 a 9 precedidos del símbolo $, Unix genera una secuencia de variables con la misma estructura, pero con un significado totalmente diferente:

- $0: Muestra el nombre del programa ejecutado
- $#: Muestra el número de argumentos introducidos
- $?: Almacena el valor devuelto después de ejecutar un comando. Si $?=0 □ el comando se ejecutó correctamente.

  Ejemplo:

  ls

  echo “$?”

.....

0

Muestra 0 □ Eso quiere decir que ejecutó la orden ls correctamente

- $$: Muestra el número de identificación del proceso

6 EJEMPLO

Probar el siguiente Shell Script que tiene dos parámetros. A la hora de ejecutarlo tendremos que indicar, tras el nombres del Shell, dos cadenas de caracteres. Cada una de ellas se sustituirá por uno de los parámetros posicionales.

#Shell parametros

echo “Este es el valor del 1º parámetro: $1”

echo “Este es el valor del 2º parámetro: $2”

Al ejecutar el Shell, habrá que escribir lo siguiente:

parametros uno dos

Este es el valor del 1º parámetro: uno

Este es el valor del 2º parámetro: dos

5 AUTOEVALUACIÓN

Los parámetros en Unix van desde:

a) $0, $2, …., $9

b) $1, $2, …., $10

c) $1, $2, …., $9

d) $0, $2, …., $10

5. COMPARACIÓN Y COMPROBACIÓN

Podemos evaluar una expresión con la orden test, que compara valores. Con esta orden se puede chequear la existencia o no de ficheros, se pueden comparar cadenas de caracteres, números enteros o longitud de cadenas.

5.1. TIPOS DE COMPARACIONES

5.1.1. COMPARAR CADENAS

Sintaxis

test cadena1 argumento cadena2

Argumento: -

- =: Devuelve 0 (true) si cadena1 es igual a cadena2, 1 (false) en caso contrario
- !=: Devuelve 0 (true) si cadena1 es distinta de cadena2, 1 (false) en caso contrario
- >: Devuelve 0 (true) si cadena1 es mayor que cadena2, 1 (false) en caso contrario
- <: Devuelve 0 (true) si cadena1 es menor que cadena2, 1 (false) en caso contrario

7 EJEMPLO

Comparar el contenido de la siguiente cadena:

```
#Shell cadenas
cadena1=”PRUEBA”
test $cadena1 = “prueba”
echo $?
1
```

Devuelve 1 (false), pues el contenido de cadena1 “PRUEBA” en mayúsculas es distinto de “prueba” en minúsculas.

5.1.2. COMPARAR ENTEROS

Sintaxis

test numero1 argumento numero2

Argumento: -

- -eq: Igual a
- -ne: Distinto de
- -lt: Menor que
- -le: Menor o igual que

- -gt: Mayor que
- -ge: Mayor o igual que

Operadores lógicos:

- !: Negación
- -a: And o Y lógico
- -o: Or u O lógico

8 EJEMPLO

Comparar el contenido de los siguientes números:

```
#Shell numeros
numero1=7
test $numero1 -eq 7
echo $?
0
```

Devuelve 0 (true), pues el contenido de numero1 es 7=7

9 EJEMPLO

Comparar los siguientes números:

```
#Shell numeros1
test 1 –eq 2
echo $?
test 1 –eq 1
echo $?
1
0
```

5.1.3. COMPARAR FICHEROS

Sintaxis

test argumento fichero

Argumento:

- -e: El fichero o cadena existe
- -w: Para averiguar si tiene permiso de escritura
- -r: Para averiguar si tiene permiso de lectura
- -x: Para averiguar si tiene permiso de ejecución

10 EJEMPLO

Comprobar si el fichero “cadenas” tiene permiso de lectura, escritura y ejecución.

```
#Shell fichero
test –r cadenas
echo $?
 test –w cadenas
echo $?
test –x cadenas
echo $?
0
0
1
```

Según esa salida, el fichero “cadenas” tiene permiso de lectura, escritura, pero no de ejecución.

6 AUTOEVALUACIÓN

Indica la respuesta correcta:

a) En los Shell Scripts de Unix se pueden realizar 2 tipos de comparaciones: De cadenas, de ficheros

b) En los Shell Scripts de Unix se pueden realizar 3 tipos de comparaciones: De cadenas, de ficheros, de enteros

c) En los Shell Scripts de Unix se pueden realizar 4 tipos de comparaciones: De cadenas, de ficheros, de enteros, de reales

d) En los Shell Scripts de Unix se pueden realizar 3 tipos de comparaciones: De cadenas, de ficheros, de reales

6. ESTRUCTURAS

6.1. CONDICIONALES

6.1.1. if

Funciona igual que en cualquier lenguaje de programación.

Sintaxis

~~FORMATO1~~

```
if expresión;
then
        órdenes;
fi
```

~~FORMATO2~~

```
if expresión;
then
        órdenes;
else
        órdenes;
fi
```

11 EJEMPLO

```
var1=7
var2=”PRUEBA”
if test $var1 –eq 7;
then
        echo “$var1”;
fi
if test $var2 = “prueba”;
then
```

```
        echo “$var2”;
fi
```

7

Si cambiamos el segundo condicional por: if test $var2 = “prueba”; entonces mostraría:

7

PRUEBA

6.1.2. case

Sirve para seleccionar una opción entre diferentes opciones. Suele utilizarse para construir menús de opciones.

Sintaxis

```
case variable in
        valor1) orden1;;
        valor2) orden2;;
        .....
        valorn) ordenn;;
        *) otros mandatos;
esac
```

*: Se usa para analizar si el valor introducido no coincide con ninguno de los anteriores.

12 EJEMPLO

Introducir por teclado un valor y comprobar si se ha pulsado un 1, un 2 u otra cosa.

```
#Shell orden_case
echo “Introduce un valor a evaluar:”
read a
case $a in
        1) echo “Has pulsado la tecla 1”;;
```

```
        2) echo “Has pulsado la tecla 2”;;
        *) echo “No has pulsado ni 1 ni 2”;;
esac
```

7 AUTOEVALUACIÓN

La estructura condicional case:

a) Sirve para comparar cadenas de caracteres

b) Sirve para seleccionar varias opción entre diferentes opciones

c) Sirve para seleccionar una opción entre diferentes opciones

d) Ninguna es correcta

6.2. BUCLES

6.2.1. while

Se utiliza para ejecutar un conjunto de instrucciones de forma repetitiva mientras la condición evaluada sea cierta.

Sintaxis

```
while condición;
do
        orden;
done
```

13 EJEMPLO

Repetir un bucle hasta que el usuario introduzca “N”. En caso contrario (mientras el usuario introduzca “S”) saldrán los siguientes mensajes:

- Continuas dentro del while
- ¿Deseas continuar (S/N)?

#Shell orden_while

```
echo “¿Deseas entrar en el bucle (S/N)?”; read respuesta
while test $respuesta = “S”;
do
        echo “Continuas dentro del while”;
        echo “¿Deseas continuar (S/N)?”; read respuesta;
done
```

ES NECESARIO: En la condición del while que antes del igual y después del igual debe ir un espacio en blanco.

6.2.2. until

Realiza la misma función que la orden anterior, pero el conjunto de instrucciones se estará repitiendo hasta que la condición sea cierta.

Sintaxis

```
until condición;
do
        orden;
done
```

14 EJEMPLO

Repetir un bucle hasta que el usuario introduzca “N”. En caso contrario (mientras el usuario introduzca “S”) saldrán los siguientes mensajes:

- Continuas dentro del until
- ¿Deseas continuar (S/N)?

```
#Shell orden_until
echo “¿Deseas entrar en el bucle (S/N)?”; read respuesta
until test $respuesta = “N”;
do
        echo “Continuas dentro del until”;
```

```
        echo “¿Deseas continuar (S/N)?”; read respuesta;
done
```

8 AUTOEVALUACIÓN

La estructura while:

a) Se utiliza para ejecutar un conjunto de instrucciones de forma repetitiva mientras la condición evaluada sea cierta

b) Se utiliza para ejecutar un conjunto de instrucciones de forma repetitiva hasta que la condición evaluada sea cierta

c) Sirve para seleccionar una opción entre diferentes opciones

6.2.3. for

Al igual que las anteriores sirve para ejecutar un conjunto de comandos durante un número determinado de veces.

Sintaxis

```
for variable in lista;
do
        orden;
done
```

15 EJEMPLO

Visualizar en pantalla los números del 1 al 5 con la estructura for.

```
#Shell orden_for
for a in 1 2 3 4 5;
do
        echo $a;
done
```

9 AUTOEVALUACIÓN

La estructura for:

a) Ejecuta un conjunto de instrucciones de forma repetitiva mientras la condición sea cierta

b) Ejecutar un conjunto de instrucciones de forma repetitiva hasta que la condición sea cierta

c) Ejecutar un conjunto de comandos durante un número determinado de veces

7. ÓRDENES AVANZADAS

7.1. DE RUPTURA

7.1.1. break

Sirve para romper la ejecución de una estructura repetitiva y saltar a la instrucción siguiente en la que finaliza la estructura repetitiva.

Se utiliza con if, while, until o for.

Sintaxis break

7.1.2. exit

Sirve para forzar la salida del Shell que se está ejecutando sin terminarlo de forma lógica.

Sintaxis exit

7.2. PARA EL MANEJO DE CADENAS DE CARACTERES

7.2.1. index

Devuelve la posición del carácter seleccionado en una cadena. Si hay más de uno devuelve la posición del primero.

Sintaxis index=”cadena” carácter

16 EJEMPLO

Realizar un Shell en el que se le pasa la cadena “cadena de caracteres” y nos devuelva la posición en la que está el carácter “e” dentro de la cadena.

#Shell orden_index

variable=”Cadena de caracteres”

expr index = “$variable” e

4

10 AUTOEVALUACIÓN

Para realizar los Shell Scripts de Unix disponemos de las siguientes órdenes de ruptura:

a) break, exit

b) break

c) exit

d) No se dispone de ninguna orden de ruptura