UD6 GNU/Linux Operating System

Shell scripts

Sistemas informáticos

1. Shell scripts

Shell Scripts

- La primera línea indicamos con que Bash se debe ejecutar el script
  - Normalmente: **#!/bin/bash**

- Las variables
  - Se **declaran NombreVariable=Valor**
    - Comillas simples (’): El intérprete de ordenes interpreta el texto entre comillas simples tal cual, sin sustituir nada.
    - Comillas dobles (“): El texto que va entre comillas dobles acepta substituciones: variables, caracteres especiales, etc.
    - Comillas invertidas (`): Especifica que resultado de la orden se asigne a una variable.
  - Se **accede** a su valor anteponiendo el carácter $ al nombre de la variable
    - #!/bin/bash
    - SALUDA=“Hola mundo”
    - echo $saluda

Shell Scripts

- **unset** nombreVariable: Para borrar una variable del intérprete de ordenes utilizamos la orden.
- **echo**: envía un texto a la salida estándar.
  Sintaxis: echo [opciones] [Variable o texto]
  - -n: evita que se haga un salto de línea al final del texto.
  - -e: permite introducir caracteres especiales como:
    - \a: alerta (campana)
    - \n: nueva línea
    - \t: tabulación
    - \\: barra invertida
    - \nnn: el carácter con el número ASCII nnn en octal

Shell Scripts

- **read**: se utiliza para que el usuario pueda dar un valor a una variable de forma interactiva.
  Sintaxis: read [opciones] NombreVariable
  - -p “Texto”: muestra el “Texto” por la pantalla.
  - -n número: lee un número de caracteres sin necesidad de presionar ENTER para finalizar la lectura.
  - -s: silencia el echo, no se ve por pantalla lo que el usuario escribe.

Ejemplo 1:

```
echo  “introduce tu nombre: ”
read nombre
echo “Hola $nombre”
```

Ejemplo 2:

```
read –p “introduce tu nombre: ” nombre
echo “Hola $nombre”
```

Shell Scripts

- **Parámetros posicionales**: Encargados de recibir los argumentos de un script y los parámetros de una función.
  - **$1, $2, …, $9** → primer parámetro, segundo parámetro, …, noveno parámetro. Si son más de nueve se indica con llaves: **${10}**, **${11}**, …
  - **$#** almacena el número de argumentos o parámetros recibidos
  - **$*** nos devuelve todos los argumentos recibidos por el script o función.
  - **shift [n]** Donde n es el número de desplazamientos a la izquierda que queremos hacer con los argumentos. Si se omite n por defecto vale 1

Shell Scripts

- Expansiones en las líneas de comandos
  - **$(orden)** permite ejecutar lo que se encuentre entre los paréntesis, y devuelve su salida.

```
echo pwd        # escribe por pantalla la palabra pwd

echo $(pwd)     # ejecuta la orden pwd, y escribe por pantalla su resultado
```

  - **`orden`** igual que $(orden) pero usando la tilde invertida
  - **$(( ))** o **$[ ]** trata como una **expresión aritmética** lo que este incluido entre los paréntesis o corchetes

```
NUMERO=4

echo $(($NUMERO+3))   # sería lo mismo poner echo $[$NUMERO+3]
```

- **let**
  - Realiza operaciones aritméticas sin usar expansiones ni dólares

```
NUMERO=4

let SUMA=NUMERO+3

echo $SUMA
```

+	Suma
-	Resta
*	Multiplicación
/	División
**	Exponente
%	Modulo (resto)

grep opcion/es 'cadena_de_texto' fichero_donde_buscar

Shell Scripts

grep

- Permite buscar cadenas de texto y palabras dentro de un fichero de texto o de la entrada estándar de la terminal.
- Una vez encontrado el contenido que estamos buscando:
  - Grep mostrará en pantalla la totalidad de la línea/s que contiene/n la cadena de texto o palabra que estamos buscando.
  - Con la opción pertinente únicamente mostrará la cadena de texto o palabra de cada una de las líneas que coincide con nuestro criterio de búsqueda.

SINTAXIS: grep opcion/es 'cadena_de_texto' fichero_donde_buscar

- **Opciones**:
- -w Para encontrar una palabra
- -v Tiene la utilidad de invertir. Por lo tanto esta opción permite mostrar las líneas que no coinciden con una determinada cadena de texto o palabra.
- -n Ver el número de línea/s en el que se encuentra la palabra o cadena de texto buscada.
- -i Para no distinguir entre mayúsculas y minúsculas.
- -r Leer de forma recursiva la totalidad de ficheros que están dentro de un directorio.

grep opcion/es 'cadena_de_texto' fichero_donde_buscar

Shell Scripts

cut

- Permite extraer columnas o campos seleccionados a partir de su entrada estándar o de archivos..

SINTAXIS:

cut -f **f**ield_campo(s) -d caracter_**d**elimitador_del_archivo

Ejemplos:

- **cut** -d " " -f 1 poema.txt Extraerá la 1ª palabra de cada línea de poema.txt
- **cut** -d " " -f 1,3 poema.txt Extraerá la 1ª y la 3ª palabras de cada línea de poema.txt
- **cut** –d: -f1,3 /etc/passwd Delimitador ":" y extraerá las filas 1y3 de passwd

Variables especiales

Variable	Significado
$0	el nombre del script.
$1...$9	Corresponde a los primeros 9 parámetros con los que se llamó el script.
$#	Número de parámetros con los que se ha invocado el script. Es muy útil para comprobar el número de argumentos pasados en la ejecución de un Shell script.
$*	Los parámetros pasados al script.
$@	Los parámetros pasados al script.
$$	El PID de la shell.
$?	Estado de salida del último comando o script ejecutado.

Shell Scripts

- **if** y **else**
  - Entre el corchete y la condición debe de haber un espacio

```
if [ condicion ]
then
   sentencias
else
   sentencias
fi
```

```
If [ condicion ]; then
   sentencias
else
   sentencias
fi
```

```
if [ exp1 ]; then
                realizar si exp1 es verdadera
elif [ exp2 ]; then
                realizar si exp1 es falsa, pero es verdadera exp2
elif [ exp3 ]; then
                realizar si exp1 y exp2 son falsas, pero es verdadera  exp3
else
                realizar si todas las expresiones anteriores son falsas
fi
```

  - Ejemplos:

```
NOMBRE=”Raúl”
if [ $NOMBRE = “Raúl” ]; then
   echo Bienvenido Raúl
fi
```

```
NUMERO=12
if [ $NUMERO -eq 12 ]; then
   echo Efectivamente, el número es 12
fi
```

Shell Scripts

- Condiciones **alfanuméricas**

Operadores de comparación de cadenas alfanuméricas	
Cadena1 **=** Cadena2	Verdadero si Cadena1 o Variable1 es **IGUAL** a Cadena2 o Variable2
Cadena1 **!=** Cadena2	Verdadero si Cadena1 o Variable1 **NO es IGUAL** a Cadena2 o Variable2
Cadena1 **<** Cadena2	Verdadero si Cadena1 o Variable1 es **MENOR** a Cadena2 o Variable2
Cadena1 **>** Cadena2	Verdadero si Cadena1 o Variable1 es **MAYOR** a Cadena2 o Variable2
-n Variable1	Verdadero si Cadena1 o Variable1 **NO ES NULO** (tiene algún valor)
-z Variable1	Verdadero si Cadena1 o Variable1 **ES NULO** (esta vacía o no definida)

Condiciones **numéricas**

Operadores de comparación de valores numéricos.	
Numero1 **-eq** Numero2	Verdadero si Numero1 o Variable1 es **IGUAL** a Numero2 o Variable2
Numero1 **-ne** Numero2	Verdadero si Numero1 o Variable1 **NO es IGUAL** a Numero2 o Variable2
Numero1 **-lt** Numero2	Verdadero si Numero1 o Variable1 es **MENOR** a Numero2 o Variable2
Numero1 **-gt** Numero2	Verdadero si Numero1 o Variable1 es **MAYOR** a Numero2 o Variable2
Numero1 **-le** Numero2	Ver. si Numero1 o Variable1 es **MENOR O IGUAL** a Numero2 o Variable2
Numero1 **-ge** Numero2	Ver. si Numero1 o Variable1 es **MAYOR O IGUAL** a Numero2 o Variable2

Shell Scripts

- Condiciones: usar la función test del bash

Operaciones condicionales usando test.	
-a fichero	Verdadero si fichero **existe**
-d fichero	Verdadero si fichero **existe**, y es un fichero de tipo **directorio**
-f fichero	Verdadero si fichero **existe**, y es un **fichero** regular.
-r fichero	Verdadero si fichero **existe** y se puede **leer**
-w fichero	Verdadero si fichero **existe** y se puede **escribir**
-x fichero	Verdadero si fichero **existe** y se puede **ejecutar**
-s fichero	Verdadero si fichero **existe** y **no está vacío**
-e fichero	Verdadero si fichero **existe**
fichero1 -nt fichero2	Verdadero si fichero1 es **más nuevo** que fichero2
fichero1 -ot fichero2	Verdadero si fichero1 es **más viejo** que fichero2

- Anidar expresiones usando:
  - && (AND)
  - || (OR).
- Operador ! (NOT) para indicar una negación.

Estructura **case**

```
case VARIABLE in
        valor1)
                Se ejecuta si VARIABLE tiene el valor1
        ;;
        valor2)
                Se ejecuta si VARIABLE tiene el valor2
        ;;
        *)
                Se ejecuta si VARIABLE no tiene ninguno de los valores anteriores
        ;;
esac
```

Shell Scripts

- Estructura **for**

```
for variable in conjunto; do
        estas líneas se repiten una vez por cada elemento del conjunto,
        y variable va tomando los valores del conjunto
done
```

```
#!/bin/bash
for dia in lunes martes miércoles jueves viernes sabado domingo; do
        echo “el dia de la semana procesado es $dia”
done
```

- **seq** muestra una secuencia de números

  seq primero incremento último

```
#!/bin/bash
for numero in $( seq 1 1 20 ); do
        echo “Número vale :” $NUMERO
done
```

Shell Scripts

- Estructura **while**

```
while [ expresión ]; do
        estas líneas se repiten MIENTRAS la expresión sea verdadera
done
```

  - Leer un archivo linea por linea

```
while read linea
do
  comando
done < archivo
```

  - "IFS" el separador de campo adecuado (el espacio es el separador por defecto).

```
while [IFS=separador] read campo1 campo2
do
  comando
done < archivo
```

Shell Scripts

- Estructura **until**: muy parecida al while.

```
until [ expresión ]; do
        estas líneas se repiten HASTA que la expresión sea verdadera
done
```

- Estructura **select**:
  - Realiza una iteración o bucle, pero presentando un menú por pantalla para que el usuario escoja una opción
  - El bucle no tiene fin con lo que para salir debemos usar la sentencia break o exit

```
select VARIABLE in conjunto_opciones; do
        Aquí variable toma el valor de una de las opciones del conjunto
done
```