Programación en Bash

Mercado Libre - IT Academy

Programación en Bash

Control de Flujo

Programación en Bash

Control de Flujo

Las sentencias condicionales

`if`, `elif` y `else`

```
if condición
then
  sentencias
elif condición
then
  sentencias
else
  sentencias
fi
```

```
if condición ; then
  sentencias
elif condición ; then
  sentencias
else
  sentencias
fi
```

3

Programación en Bash

Control de Flujo

Los códigos de terminación

En Unix los comandos devuelven un código numérico (exit status).

En general un código de terminación 0 significa que el comando terminó correctamente.

En cualquier caso, conviene consultar la documentación del comando para interpretar bien su códigos de terminación.

La sentencia `if` comprueba el código de terminación de un comando en la *`condición`*. Si es 0, se evalúa como cierta.

4

Programación en Bash

Control de Flujo

Los códigos de terminación

```
if comando ; then
  Procesamiento normal
else
  Procesamiento del error
fi
```

En este caso, si un comando falla, se puede informar del error de una manera específica y evitar un comportamiento no controlado.

5

Programación en Bash

Control de Flujo

Las sentencias `return` y `exit`

Para leer el código de terminación del último comando ejecutado disponemos de la variable `?`, cuyo valor es `$?`

```
$ cd directorioErroneo
$ echo $?

$ cd directorioExistente
$ echo $?
```

La variable `?` debe ser leída justo después de ejecutar el comando. Se puede usar una variable auxiliar para guardar dicho valor para su posterior uso.

6

Programación en Bash

Las sentencias `return` y `exit`

Ejercicio:

Crear una función `cd` que sustituya el comando interno `cd` de forma que al ejecutar `cd`, además de cambiar de directorio, se de un mensaje que indique el antiguo y nuevo directorio.

```
function cd {
      builtin cd "$@"
      echo "$OLDPWD -> $PWD"
}
```


Programación en Bash

Las sentencias `return` y `exit`

¿Cómo retornar el código de terminación al hacer la función?

Usamos la sentencia `return`

```
function cd {
      builtin cd "$@"
      local ct=$?
      echo "$OLDPWD -> $PWD"
      return $ct
}
```

¿Por qué tuvimos que guardar el código de terminación en una variable?

Programación en Bash

Control de Flujo

Las sentencias `return` y `exit`

`return` puede ser usada sólo dentro de funciones y `scripts` ejecutados con `source`.

Por el contrario, la sentencia `exit` puede ser ejecutada en cualquier sitio y lo que hace es abandonar el `script` (aún cuando se ejecute dentro de una función).

`exit` se usa para detectar situaciones erróneas que obliguen al programa a detenerse. También se usa para «cerrar» un programa.

Programación en Bash

Control de Flujo

Operadores lógicos y códigos de terminación

`and` representada por `&&`, `or` representada por `||` y `not` representada por `!`

```
if cd /tmp && cp 001.tmp $HOME ; then
      sentencias
fi
```

```
if cd /tmp || cp /tmp/002.tmp ~/d.tmp ; then
      sentencias
fi
```

```
if ! cp /tmp/002.tmp ~/d.tmp ; then
      sentencias
fi
```

Programación en Bash

Test condicionales

La sentencia `if` lo único que hace es evaluar códigos de terminación.

El comando interno `test` nos permite compara otras muchas condiciones y devuelve un código de terminación.

```
$ test cadena1 = cadena2
$ [ cadena1 = cadena2 ]
```


Programación en Bash

Comparación de cadenas

`str1 = str2`	Las cadenas son iguales.
`str1 != str2`	Las cadenas son distintas.
`str1 < str2`	Menor (lexicográficamente) que…
`str1 > str2`	Mayor (lexicográficamente) que…
`-n str1`	Es no nula y tiene longitud mayor a 0.
`-z str1`	Es nula (tiene longitud 0).

```
if [ str1 = str2 ] ; then …
```

Programación en Bash

Control de Flujo

Comparación numérica de enteros

`-lt`	Menor que…
`-le`	Menor o igual que...
`-eq`	Igual que…
`-ge`	Mayor o igual que…
`-gt`	Mayor que…
`-ne`	No es igual que…

```
if [ $# -lt 1 ] ; then …
```

13

Programación en Bash

Control de Flujo

Los test de condición (`[ ]`) pueden combinarse usando operadores lógicos

```
if [ condición ] && [ condición ] ; then
```

También pueden combinarse comandos del shell con test de condición

```
if comando && [ condición ] ; then
```

También pueden usarse operadores lógicos dentro de `[ ]`, pero en estos casos se usa `-a` (para and) y `-o` (para or).

```
if [ \($var1 -le $var2\) -a \($var1 -gt $var3\) ]
```

14

Programación en Bash

Comprobar atributos de archivos

`-a archivo`	`archivo` existe
`-d archivo`	`archivo` existe y es un directorio
`-f archivo`	`archivo` existe y es un archivo regular
`-L archivo`	`archivo` existe y es un enlace simbólico
`-r archivo`	`archivo` existe y puede leerse
`-s archivo`	`archivo` existe y no está vacío
`-x archivo`	`archivo` existe y podemos ejecutarlo
…	…

Programación en Bash

El bucle `for` y el comando `xargs`

El bucle `for` en Bash es un poco distinto a los bucles tradicionales.

Se parece más a un bucle `for each`.

```
for var in lista
do
  ...
  Sentencias que usan $var
  ...
done
```

Programación en Bash

Control de Flujo

El bucle `for` y el comando `xargs`

```
for mascota in Perro Gato Canario
do
      echo $mascota
done
```

```
for archivo in *
do
      ls -l "$archivo"
done
```


Programación en Bash

Control de Flujo

El bucle `for` y el comando `xargs`

```
# bucles para recorrer argumentos

for arg in "$*"
do
      echo "Elemento:$arg"
done

for arg in "$@"
do
      echo "Elemento:$arg"
done

$ recibe "El Perro" "El Gato"
```

Programación en Bash

El bucle `for` y el comando `xargs`

Ejercicio:

Crear un script llamado `listapath` que liste todos los archivos que podemos ejecutar en los directorios de la variable de entorno `PATH`. Además, debe avisar si encuentra en `PATH` un directorio que no exista.

NOTA: se puede usar `IFS` para usar : como separador de los elementos de la lista, escribiendo `IFS=':'` (observar los elementos en la lista `$PATH`).

Programación en Bash

El bucle `for` y el comando `xargs`

```
function ListaEjecutables {
     IFS=''
     archivos=$(ls -1 $1)
     for archivo in archivos
     do
          path_archivo="$1/$archivo"
          if [ -x $path_archivo ]; then
               echo $path_arhivo
          fi
     done
     IFS=':'
}
```

Programación en Bash

El bucle `for` y el comando `xargs`

```
IFS=':'
for dir in $PATH
do
     if [ -z "$dir" ]; then
          echo "ENCONTRADO UN DIRECTORIO VACÍO"
          exit 1
     elif ! [ -d "$dir" ]; then
          echo "$dir NO ES UN DIRECTORIO VÁLIDO"
          exit 1
     else
          ListaEjecutables $dir
     fi
done
```


Programación en Bash

El bucle `for` y el comando `xargs`

El comando `xargs` es muy útil para ejecutar una operación sobre una lista de cadenas.

Ejecuta la utilidad pasada como argumento sobre cada elemento de la lista:

```
find -X . -type d | xargs chmod 755
```

`find` escribe en su salida estándar los subdirectorios. El comando `xargs` lee estos directorios y ejecuta sobre cada uno de ellos el comando `chmod 755`. (`-X` permite usar `find` con `xargs` de forma segura, evitando problemas con espacios o comillas)

Programación en Bash

Control de Flujo

Los bucles `while` y `until`

```
while comando
do
    ...
done
```

```
until comando
do
    ...
done
```

En este caso el comando también puede ser una condición encerrada entre `[ ]`.

Programación en Bash

Control de Flujo

Los bucles `while` y `until`

Ejercicio:

Modificar el script `listapath` pero usando un bucle while y aprovechando el hecho de que una cadena en un test de condición evalúa por verdadero cuando no está vacía y por falso cuando lo está.

Programación en Bash

La sentencia `case`

```
case cadena in
   patrón1)
      Sentencias;;
   patrón2)
      Sentencias;;
   ...
esac
```

El default se consigue agregando un patrón `*)`

Programación en Bash

La sentencia `case`

Ejercicio:

Crear un script llamado `tipoarchivo` que guarde el nombre en un archivo `imagen`, `texto` y `otros` los nombres de los archivos que reciba en una lista de parámetros según su extensión.

.txt	al archivo `texto`
.jpg .bmp .tga .xmp .pcx .tif .gif	al archivo `imagen`

Programación en Bash

La sentencia `select`

Nos permite generar fácilmente un menú simple.

```
select variable in lista
do
      sentencias que usan $variable
done
```

Tiene una sintaxis similar al `for`.

La sentencia genera un menú con los elementos de lista.

El bucle se puede abandonar con `break`.

Programación en Bash

La sentencia `select`

Ejercicio:

Escribir una función `elegirdir()` que permita al usuario elegir un directorio de los disponibles en la variable de entorno $HOME. Al elegir un directorio, se listarán todos los archivos ejecutables que contenga.

Programación en Bash

La sentencia `shift`

Por defecto desplaza los argumentos de la línea de órdenes una posición a la izquierda. Se puede desplazar un número de posiciones mayor a uno.

```
shift[n]
```

Un ejemplo clásico de uso de este comando son los scripts con opciones.

Programación en Bash

La sentencia `shift`

```
if [ $1 = -o ]
then
    Ejecuta las operaciones con $2 y $3
else
    Ejecuta las operaciones con $1 y $2
fi
```

```
if [ $1 = -o ] ; then
    Procesa -o
    shift
fi
Ejecuta las operaciones con $1 y $2
```

Programación en Bash

Control de Flujo

Ejercicio 1:

Crear un script de nombre `opciones` que pueda tener una de estas dos sintaxis:

```
opciones –o archivo1 archivo2
opciones archivo1 archivo2
```

En cualquiera de los dos casos copiará el archivo1 en el archivo2, comprobando que el archivo1 existe y que el archivo2 no existe. Si se utiliza la primera sintaxis, el script copiará el archivo1 en el archivo2, pero ordenando previamente el archivo1.

31

Programación en Bash

Control de Flujo

Ejercicio 2:

Crear un script de nombre `mezclaArchivos` que pueda recibir un número indeterminado de parámetros. Los n primeros serán archivos con el mismo número de líneas y el último parámetro el archivo de salida.

El script debe mezclar las líneas de los n primeros archivos y guardarlos en archivo de salida.

El número mínimo de parámetros a procesar serán 3, no estando limitado el número máximo. Si se invoca con menos parámetros debe mostrar un mensaje de error.

32

Programación en Bash

Control de Flujo

Ejercicio 3:

Crear un script de nombre `menuArchivos` que pueda recibir un archivo o directorio como argumento. A través del menú, el script debe realizar las siguientes operaciones sobre el argumento:

a. Contar el número de archivos en el directorio. Si es un archivo, debe mostrar el correspondiente error.
b. Mostrar el tamaño del archivo.
c. Crear el archivo (antes debe asegurarse de que no exista).
d. Mostrar toda la información del archivo.

33

Programación en Bash

Control de Flujo

Ejercicio 4:

Crear un script de nombre `cambiaExtension` que pueda recibir dos extensiones como argumentos. El script deberá cambiar la extensión de todos los archivos que tengan la extensión pasada como primer argumento, por la extensión pasada como segundo argumento.

34

Programación en Bash

Arrays

Para declarar un array podemos usar el comando `declare -a`

```
$ declare -a A
```

Podemos crearlo asignándole directamente valores:

```
$ B=(Banana 5 Manzana)
```

Podemos preguntar por alguna de esas variables escribiendo:

```
$ declare -p A
```

Se cuenta desde 0 aunque podemos cambiar los índices:

```
$ C=([5]=Pera [0]=Uva [4]=400)
$ C=([5]=Pera Naranja Pomelo)
```


Programación en Bash

Arrays

Otra forma de llenar un array es a partir de lo que devuelve un comando:

```
$ E=($(ls -1))
```

Accedemos a los elementos mediante el operador corchete:

```
$ echo ${B[2]}
```

Podemos inicializar el array:

```
$ F[2]=casa
$ F[0]=auto
```

Programación en Bash

Arrays

Podemos iterar sobre los parámetros de un array:

```
for e in ${C[*]}
     do
          echo $e
     done
```

Podemos consultar la longitud de un array:

```
$ echo ${#C[@]}
```

Podemos consultar la longitud de un elemento del array:

```
$ echo ${#C[5]}
```


Programación en Bash

Edición de stream con `sed`

El comando `sed` es un editor de texto orientado a stream. Acciones como buscar un patrón y reemplazarlo por otro contenido se realizan sobre el stream que circula desde la entrada estándar a la salida estándar de `sed`.

`sed [opciones] instrucciones archivo`

Normalmente las instrucciones se encierran entre comillas fuertes para evitar que el shell sustituya símbolos como `$` o `*`.

Si no se especifica el archivo, `sed` lee de su entrada estándar.

Programación en Bash

Herramientas de manipulación de texto

Edición de stream con `sed`

Las `instrucciones` constan de una **dirección** y un **procedimiento**.

Tanto la una como la otra suelen hacer uso de patrones (una expresión regular encerrada entre barras (/).

El comando `sed` procesa línea a línea el contenido del archivo.

Veamos un ejemplo que permita entender mejor cómo funciona:

39

Programación en Bash

Herramientas de manipulación de texto

Edición de stream con `sed`

Supongamos el archivo `domicilios.txt`

```
Juan Pérez, Av. Maipú 234, CBA
José Fernández, 24 de Setiembre 342, CBA
Ana García, Oliva 2312, MZA
María Pintos, Las Playas 160, BA
Jorge Funes, Laprida 3456, CBA
```

Para reemplazar `CBA` por `Córdoba` podemos invocar `sed`:

```
$ sed 's/, CBA/, Córdoba/' domicilios.txt
```

40

Programación en Bash

Herramientas de manipulación de texto

Edición de stream con `sed`

Para reemplazar en un archivo un patrón por otro lo que hacemos al invocar `sed` con el procedimiento `s` por (sustitute)

```
sed 's/patron1/patron2/' archivo
```

Podemos expresar múltiples instrucciones

```
sed -e 's/patron1/, patron2/', -e 's/
patron3/patron4/' archivo
```

Las instrucciones se pueden colocar en un archivo de instrucciones `.sed` e invocarlo:

```
sed -f archivo_instrucciones archivo
```


Programación en Bash

Herramientas de manipulación de texto

Edición de stream con `sed`

El archivo sobre el que se trabaja no se modifica. Si queremos modificarlo, debemos redireccionar la salida de `sed` a otro archivo y una vez ejecutado, reemplazar el archivo original con `mv`.

Podemos también suprimir la salida estándar automática:

```
sed -n 's/patron1/patron2/p' archivo
```

suprime la salida automática

imprime

Programación en Bash

Generación de informes con `awk`

El comando `awk` es similar al comando `sed`:

`awk [opciones] instrucciones archivo`

Para imprimir el primer argumento de una lista:

```
awk '{print $1}' archivo
awk '/patron/{print $1}' archivo
awk –F"delimitador" '/patron/{print $1}' archivo
awk '/patron/{print $1; print $2}' archivo
```