Programación en Bash

Mercado Libre - IT Academy

Programación en Bash

BASH (Bourne Again Shell)

- Es un programa informático.
- Intérprete de comandos.
- Es un lenguaje de programación de consola.
- Está basado en la *shell de Unix.*
- Es el shell que proporcionan por defecto muchos sistemas basados en Unix, como Linux o macOS.

Programación en Bash

Introducción a Bash

Programación en Bash

Introducción a Bash

macOS trae preinstalado el shell Bash.

Para saber qué shell está instalado y cuál es la versión que estamos utilizando, en una consola escribamos los siguientes comandos:

```
$ echo $SHELL
$ echo $BASH_VERSION
$ whereis bash
$ cat /etc/shells
```

Para hacer bash por defecto:

```
$ chsh –s /bin/bash
```

Programación en Bash

macOS trae preinstalado el shell Bash.

Para saber qué shell está instalado y cuál es la versión que estamos utilizando, en una consola escribamos los siguientes comandos:

```
$ echo $SHELL
$ echo $BASH_VERSION
$ whereis bash
$ cat /etc/shells
```

Para hacer bash por defecto:

```
$ chsh –s /bin/bash
```

(El comando se ejecuta y hay que reiniciar para que tome los cambios)

Programación en Bash

Los comodines:

`?`	Para referirnos a sólo un carácter.
`*`	Cero o más caracteres.
`[conjunto]`	Uno de los caracteres del *conjunto.*
`[!conjunto]`	Un carácter que no está en el *conjunto*.

Escribamos algunos ejemplos en la consola:

```
$ ls *.txt
... (practicar otros más)
```

Programación en Bash

Los comodines

El comodín ~ se refiere al directorio home de los usuarios

```
$ cd ~
```

El comodín llaves expande una palabra por cada una de las cadenas de caracteres que contiene:

```
$ echo g{at,racios}o
```

También es posible anidarlo:

```
$ echo ca{s{a,erio,ona},racol}.{jpg,txt}
```


Programación en Bash

Los comodines

Se puede usar .. para hacer algo similar a lo que hacen los corchetes obteniendo todos los caracteres ASCII entre dos letras:

```
$ echo l{a..e}
```

La llave debe contener al menos dos cadenas si no, no se realiza la expansión. Probemos con:

```
$ echo c{a}sa
```

Programación en Bash

Los comodines

Ejercicio

Crear 12 archivos vacíos en un nuevo directorio.

`file1.txt, file2.txt …`

Usar el comando `ls` con los comodines vistos, para listar uno, todos o algunos de los archivos del directorio.

NOTA: Una forma de hacerlo rápidamente es escribir:

```
$ touch file{1 .. 12}.txt
```


Programación en Bash

Los comandos internos de Bash

Bash busca los comandos a ejecutar en los directorios indicado en `$PATH`. Sin embargo, existe una serie de comando *internos.*

Ejemplos de estos comandos son `cd, chdir, alias, set` …

Y se puede obtener ayuda sobre estos comandos usando el comando `help`:

```
$ help set
```

Programación en Bash

Redirecciones

Cada programa tiene asociadas una entrada estándar, una salida estándar y una salida de errores estándar.

No obstante, podemos usar el **operador de redirección** <

```
$ cat
```
(usando el teclado como entrada estándar)

```
$ cat < file.txt
```

(sólo para el comando `cat`, podemos ahorrarnos el operador <)

Existen algunos otros comandos que leen de la entrada estándar, realizan una operación sobre el texto y escriben en la salida estándar **¿conocen alguno de ellos?**

Programación en Bash

Redirecciones

El operador de redirección >

```
$ date > ahora
```

Podemos combinar ambos operadores

```
$ cat < file1.txt > file2.txt
```

Podemos cambiar la salida de errores estándar con el operador `2>`

```
$ cat < file1.txt > file2.txt 2> errores
```

¿Sabrían cómo añadir contenido sin sobrescribir un archivo?

Programación en Bash

Redirecciones y pipes

Ejercicio

Crear un archivo `file1.txt` con el contenido de un directorio.

Crear un archivo `file2.txt` con el contenido de otro directorio.

Crear un archivo `file3.txt` con el contenido de otro directorio.

Sobrescribir el archivo `file2.txt` con el contenido del archivo `file1.txt`. Redirigir los errores a un archivo `errores.txt`.

Añadir al archivo `file3.txt` el contenido del archivo `file1.txt` . Redirigir los errores a un archivo `errores.txt`, sin sobrescribirlo.

Programación en Bash

Caracteres especiales y entrecomillado

Un carácter especial es, por ejemplo, >

¿Qué sucede si ejecutamos?

```
$ echo 2 * 3 > 5 es verdadero
```

¿Cómo podemos hacer que se imprima?

```
$ echo '2 * 3 > 5 es verdadero'
```

¿Qué diferencia encuentra a continuación? (Pruebe)

```
$ find / -name *.txt
$ find / -name '*.txt'
```

Programación en Bash

Caracteres especiales y entrecomillado

Existe otra forma de solucionar el problema de los caracteres especiales y es mediante los caracteres de escape.

```
$ echo 2 \* 3 \> 5 es verdadero
```

El carácter de escape \ se usa también para que el intérprete no considere los espacios.

```
$ mplayer mi\ serie.avi
```

Finalmente, se usa como escape de sí mismo o para entrecomillar entrecomillados.

Programación en Bash

Caracteres especiales y entrecomillado

¿Cómo escribir textos de varias líneas?

```
$ echo Todos los días de febrero suelen ser \
> calurosos pero a veces refresca como para \
> un abrigo liviano.
```

```
$ echo 'Todos los días de febrero suelen ser
calurosos pero a veces refresca como para un
abrigo liviano.'
```

¿Qué diferencia hay entre una y otra forma de escribir? (Pruebe)

Programación en Bash

Combinaciones de Teclas

Programación en Bash

Combinaciones de Teclas

Historial de comandos

`$ history | grep` *"comando usado recientemente"*

El comando `fc` (sumamente útil)

`$ fc -l`

`$ fc 5` (permite editar, pero se ejecutará!!! el comando)

Ejecutando comandos anteriores

`!!`	ejecuta el último comando
`!n`	ejecuta el comando número *n*
`!cadena`	ejecuta el último que empiece por *cadena*

Programación en Bash

Combinaciones de Teclas

Las teclas de control del terminal

```
$ stty all
```

Ejemplos:

Ctrl+C *detiene el comando actual.*

Ctrl+ ***fuerza*** *la parada del comando actual.*

Ctrl+D *fin de flujo de entrada.*

Ctrl+W *borra desde la posición actual al principio de la palabra.*

19

Programación en Bash

Combinaciones de Teclas

Modos de edición en la línea de comandos

```
$ set –o vi
$ set –o emacs
```

Veamos rápidamente las teclas de edición de emacs que se pueden usar también en Bash.

Escribamos:

```
$ emacs test.txt
```

20

Programación en Bash

Combinaciones de Teclas

Modos de edición en la línea de comandos

Desplazarse en una línea

CTRL+A	Ir al principio de la línea
CTRL+E	Ir al final de la línea
ESC+B	Ir una palabra hacia atrás
ESC+F	Ir una palabra hacia adelante
CTRL+B	Ir una letra hacia atrás
CTRL+F	Ir una letra hacia adelante

Recomendamos consultar un manual sobre los comandos emacs.

21

Programación en Bash

Personalizando el Entorno

22

Programación en Bash

Archivos de configuración de Bash

Cada vez que entramos, se ejecuta el contenido del archivo `/etc/profile`, y luego se ve si en el directorio home existe el archivo `.bash_profile`. De ser así, se ejecuta su contenido para personalizar aspectos de nuestra cuenta.

Para ver cambios en este archivo sin salir de nuestra cuenta, escribimos:

```
$ source .bash_profile
$ . .bash_profile
```


Programación en Bash

Archivos de configuración de Bash

El orden de ejecución de los archivos que «pueden» existir es el siguiente:

```
.bash_profile
.bash_login
.profile
```

Al salir de la cuenta se ejecutará, de existir, el archivo:

```
.bash_logout
```

Programación en Bash

Personalizando el Entorno

Los alias

Son nombres cortos utilizados sólo para referir un comando que usamos mucho:

```
$ alias ll = 'ls –laF'
```

(hace un listado detallado del directorio actual)

`$ alias`	obtiene un listado de todos los alias
`$ unalias ll`	borra una definición determinada

25

Programación en Bash

Personalizando el Entorno

Variables de Entorno

```
$ NOMBRE='IT Academy'
$ echo $NOMBRE
$ unset NOMBRE
```

Variables y entrecomillado

```
$ articulo='Café'
$ precio='25.00'
$ echo "El $articulo cuesta $precio pesos"
```

26

Programación en Bash

Personalizando el Entorno

Variables de Entorno Internas

SHELL	Ruta del archivo que estamos usando
LINES	Líneas del terminal en caracteres
COLUMNS	Columnas del terminar en caracteres
HOME	Directorio home del usuario
PWD	Directorio actual
OLDPWD	Anterior al directorio actual
USER	Nombre del usuario en la cuenta actual

27

Programación en Bash

Programación Básica del Shell

28

Programación en Bash

Scripts

Un **script** es un archivo que contiene comandos Bash ejecutables.

Se crea un archivo con el editor de texto.

Puede ejecutarse con el comando `source`.

Funciones

Se ejecutan dentro de la memoria del propio proceso Bash, con lo que es más eficiente que ejecutar scripts aparte, pero tienen el inconveniente de que tienen que estar siempre cargadas en la memoria del proceso Bash.

Programación en Bash

Para definir una función escribiremos:

```
function nombrefn {
   …
   comados bash
   …
}

nombrefn() {
   …
   comados bash
   …
}
```

Una función se almacena como una EV

Programación en Bash

Variables del shell

Bash utiliza (principalmente) variables tipo cadena.

Por convención las variables de entorno exportadas se escriben en mayúsculas y las que no en minúscula.

Parámetros Posicionales

Son los encargados de recibir los argumentos de un script y los parámetros de una función. Sus nombre son 1, 2, 3, etc. por lo que para acceder a ellos, utilizaremos $1, $2, $3, etc.

El parámetro posicional 0 almacena el nombre del script donde se ejecuta.

Programación en Bash

Ejercicio

Crear el script:

```
echo "El script $0"
echo "Recibe los argumentos $1 $2 $3 $4"
```

Guardarlo como `recibe` con el permiso `x` activado.

Ejecutar:

```
$ recibe hola adiós
```

Programación en Bash

Ejercicio

Definir la función:

```
$ function recibe {
> echo "El script $0"
> echo "Recibe los argumentos $1 $2 $3 $4"
> }
```

Ejecutar:

```
$ recibe buenos días Bash
```


Programación en Bash

Variables locales y globales

Los parámetros posicionales (por defecto) son locales

En el archivo `saluda` escribamos el siguiente script

```
# Script que llama a una función para saludar
function DiHola {
    echo "Hola $1"
}
DiHola
$ saluda Juan
```

Programación en Bash

Variables locales y globales

Modifiquemos el script anterior:

```
# Script que llama a una función para saludar
function DiHola {
    echo "Hola $1"
}
DiHola $1
$ saluda Juan
```


Programación en Bash

Variables locales y globales

A diferencia de los parámetros posicionales, el resto de las variables son globales. Escribamos el script `dondeestoy`

```
# Ejemplo de variable global
function EstasAqui {
     donde='Dentro de la función'
}
donde='En el script'
echo $donde
EstasAqui
echo $donde

$ dondeestoy
```

Programación en Bash

Variables locales y globales

Si queremos que una variable *no* posicional sea local, debemos ponerle el modificador `local`.

```
# Ejemplo de variable global
function EstasAqui {
      local donde='Dentro de la función'
}
donde='En el script'
echo $donde
EstasAqui
echo $donde

$ dondeestoy
```


Programación en Bash

Variables locales y globales

Finalmente, las variables globales también se pueden definir dentro de la función.

```
# Ejemplo de variable global
function EstasAqui {
      local donde='Dentro de la función'
      quiensoy='Juan'
}
donde='En el script'
echo $donde
EstasAqui
echo $donde
echo "Soy $quiensoy"

$ dondeestoy
```

Programación en Bash

Las variables $*, $@ y $#

Ejemplo:

Escribamos el script recibe con el siguiente contenido:

```
for i in $@; do echo "\$@ $i"; done
for i in $*; do echo "\$* $i"; done
for i in "$@"; do echo "\"\$@\" $i"; done
for i in "$*"; do echo "\"\$*\" $i"; done
```

```
$ ./recibe hola adiós
```


Programación en Bash

Las variables $*, $@ y $#

Ejemplo:

```
function CuentaArgumentos {
    echo "Recibidos $# argumentos"
}
```

Lo guardamos en archivo `recibe`

Probemos pasar los argumentos recibidos por el script a la función usando cualquiera de las siguientes variables:

Programación en Bash

Las variables $*, $@ y $#

```
CuentaArgumentos $*
CuentaArgumentos $@
$ recibe Gato Perro
$ recibe "El gato" "El perro"

CuentaArgumentos "$*"
CuentaArgumentos "$@"
$ recibe Gato Perro
$ recibe "El gato" "El perro"
```


Programación en Bash

Expansión de variables usando llaves

`$variable` es la forma simplificada de `${variable}`

Sin embargo:

```
$ nombre=Juan
$ apellido=Perez
$ echo "$nombre_$apellido"
$ echo "${nombre}_$apellido"
$ echo "${nombre}_${apellido}"
```

Programación en Bash

Expansión de variables usando llaves

Otro lugar donde necesitamos usar llaves es cuando deseamos obtener el décimo parámetro posicional. `$10` tomará el `1`, si queremos que tome el `10`, usaremos: `${10}`

Programación en Bash

Operadores de cadena

Los operadores de cadena nos permiten realizar las siguientes operaciones:

- Asegurarnos de que una variable exista (que esté definida y no es nula).
- Asignar valores por defecto a las variables.
- Tratar errores debidos a que una variable no tiene un valor asignado.
- Tomar o eliminar partes de una cadena que cumplen un determinado patrón.

Programación en Bash

Operadores de cadena: Operadores de sustitución

`${var:-valor}`

Si `var` existe y no es nula retorna su valor, sino retorna `valor`.

`${var:+valor}`

Si `var` existe y no es nula retorna `valor`, sino retorna una cadena nula.

`${var:=valor}`

Si `var` existe y no es nula retorna su valor, sino asigna `valor` y retorna su valor.

Programación en Bash

Operadores de cadena: Operadores de sustitución

`${var:?mensaje}`

Si `var` existe y no es nula retorna su valor, sino imprime `var:mensaje` y aborta el script que se está ejecutando (en shells no interactivos). Si se omite `mensaje` imprime un mensaje por defecto.

`${var:offset:longitud}`

Retorna una subcadena de `var` que empieza con `offset` y tiene una longitud `longitud` de caracteres. El primer caracter de `var` empieza en la posición 0. Si se omite la longitud, devuelve todos los caracteres hasta el final de la cadena.

Programación en Bash

Operadores de cadena: Operadores de sustitución

Ejercicio

Crear un archivo que contenga 5 líneas con un formato: (numero – nombre).

Escribir e interpretar el siguiente script de nombre `select`:

```
archivo=$1
archivo=${archivo:?'no suministrado'}
cuantos=$2
defecto=5
sort -nr $archivo | head -${cuantos:=$defecto}
```


Programación en Bash

Operadores de cadena: Operadores de búsqueda de patrones

```
$ var="El gato y el ratón corren en el patio"
$ {var#patron}              Ej: ${var#El*n}
${var##patron}              Ej: ${var##El*n}
${var%patron}               Ej: ${var%e*patio}
${var%%patron}              Ej: ${var%%e*patio}
${var/patrón/cadena}        Ej: ${var/el/mi}
${var//patrón/cadena}       Ej: ${var//el/mi}
```

Operadores de cadena: Operador longitud

```
${#var}1
```