Bash Scripts

Introducción

El shell de `bash` y `sh` permiten ejecutar *scripts* diseñados para automatizar la ejecución de comandos, agregando algunas estructuras adicionales, que permitirán generar archivos ejecutables para automatizar tareas.

Los *scripts* son archivos de texto marcados como ejecutables (con el modo `--x`) y generalmente se indica el shell de ejecución con `#!/bin/bash` o `#!/bin/sh`. Generalmente estos se guardan con la extensión `.sh`.

> Ejemplo de un script de Bash que despliega un "Hola mundo"

```
#!/bin/bash

echo Hola mundo :D
```

Para ejecutar un script usaremos el comando `sh`.

```
[$] sh <script>.sh
```

Adicionalmente podemos usar `sudo` para ejecutar el script con el usuario `root`.

Variables

Podemos crear variables que guarden valores (generalmente números y texto) indicando su nombre y el valor asignado.

```
SINTAXIS: Crear una variable

<name>=<value>
```

- **NOTA:** Es importante no poner espacio entre el nombre y el valor, por ejemplo `a=hola` significará que la variable `a` contiene el texto `"hola"`. Pero `a = hola` intentará ejecutar el comando `a`.

Para usar una variable podemos anteponer el símbolo de dólar antes del nombre de la variable.

```
SINTAXIS: Usar una variable

$<name>

EJEMPLO:

nombre=Ana

echo Hola $nombre, ¿cómo estás?
```

En la siguiente tabla se muestran algunas variables ya asignadas en un script.

Variable	Descripción
`$0`	Nombre del script
`$<n>`	Parámetro `<n>` pasado al script. Ejemplo `sh hello.sh a b c` implica `$1 -> a`, `$2 -> b`, `$3 -> c`
`$#`	Número de parámetros pasados al script
`$@`	Todos los parámetros pasados al script
`$?`	Salida del último proceso ejecutado
`$$`	Id del proceso del script
`$USER`	Nombre del usuario que ejecuta el script
`$HOSTNAME`	Nombre de la máquina que ejecuta el script
`$SECONDS`	Tiempo transcurrido en segundos desde que se ejecutó el script
`$RANDOM`	Un valor aleatorio cada que se usa la variable
`$LINENO`	Número de líneas del script

El sistema también tiene algunas variables llamadas **Variables de Entorno** las cuales se pueden consultar con el comando `env`.

```
[$] env
```

Podemos usar estas variables de entorno dentro de nuestro script como `$<env>`, donde `<env>` es el nombre de la variable de entorno.

La siguiente tabla muestra algunas variables de entorno útiles.

Variable	Descripción
`$PATH`	Las rutas exportadas donde se encuentran binarios y comandos ejecutables

Variable	Descripción
$SHELL	El nombre de la terminal shell
$EDITOR	El editor de texto usado por defecto
$USERNAME	El nombre de usuario
$HOME	La ruta de la carpeta principal del usuario

Comillas

Existen dos tipos de caracteres.

- **Genéricos** - Son caracteres comunes
- **Cita** - Son caracteres especiales usados por bash para procesar

En la siguiente tabla se muestran algunos caracteres especiales importantes.

Caracter	Descripción
~	Ruta de la carpeta principal del usuario
`	Sustitución de comandos (evaluación previa)
#	Comentario sobre la línea
$	Uso de variable
&	Procesamiento en segundo plano
*	Comodín
(	Comienzo de un subshell
)	Final de un subshell
\	Escape de caracter
|	Pipe / Tubería
[	Comienzo de un conjunto de caracteres comodines
]	Final de un conjunto de caracteres comodines
{	Comienzo de un bloque de comandos
}	Final de un bloque de comandos
;	Separador de comandos
'	Comilla Simple (literal)
"	Comilla Doble (natural)
<	Redirección de entrada
>	Redirección de salida

Caracter	Descripción
/	Separador de rutas (separador de carpetas)
?	Comodín de único caracter
!	Pipeline Lógico

Las comillas simples sirven para agrupar caracteres literales.

```
nombre=Ana

echo 'Hola $nombre'

>>> Hola $nombre
```

Las comillas dobles procesan variables y algunas otras cosas.

```
nombre=Ana

echo "Hola mundo $nombre"

>>> Hola Ana

NOTA:

Si la variable se pega con otros caracteres, podemos usar ${<name>}
para determinar correctamente a la variable.

EJEMPLO:

peso=5

# INCORRECTO
echo "El precio es $pesogr."

# CORRECTO
echo "El precio es ${peso}gr."
```

Podemos crear variables que contengan el resultado de una ejecución usando `$(<expression>)`.

```
fecha=$(date -u)
```

Alternativamente podemos crear variables que contengan el resultado de una ejecución usando `<expression>`.

```
carpeta=`pwd`
```

También podemos hacer evaluaciones sobre números usando `$((<expression>))`

```
resultado=$((2 + 2))
```

Redirecciones

Los comandos pueden aceptar redirecciones provenientes de archivos y de la entrada estándar, la salida estándar y el error estándar.

Los descriptores de archivos se componen de los primeros tres niveles para la entrada, salida y error estándares y de ahí los archivos enlazados.

```
0 - Entrada estándar
1 - Salida estándar
2 - Error estándar
n - El descriptor `n-ésimo`
```

Cuándo se ejecuta un comando se puede redireccionar una entrada o salida mediante el caracter `<` (entrada) o el caracter `>` (salida).

```
SINTAXIS: Redirección de la entrada estándar

# Redirección desde un archivo

`command` < `file`

# Redirección desde el descriptor `n`

`command` <&`n`

# Redirección desde una línea de texto

`command` <<< "..."

# Redirección desde múltiples líneas de texto

`command` <<EOF
...
...
EOF

# Redirección por tubería
```

```
`command 1` | `command 2`
```

Para el caso de la salida estándar tenemos también la alternativa de redirigir la salida a un archivo, otro comando (tubería) o a un descriptor.

```
SINTAXIS: Redirección de la salida estándar

# Redirección a un archivo (crea o reemplaza)

`command` > `file`

# Redirección a un archivo (crea o agrega al final)

`command` >> `file`

# Redirección hacía el descriptor `n`

`command` >&`n`

# Redirección hacía la nada

`command` > /dev/null

# Redirección por tubería

`command 1` | `command 2`
```

Cuándo un comando genera error, los errores y advertencias irán hacía el error estándar, este se puede redirigir a un archivo, otro comando (tubería) o a un descriptor.

```
SINTAXIS: Redirección del error estándar

# Redirección a un archivo (crea o reemplaza)

`command` 2> `file`

# Redirección a un archivo (crea o agrega al final)

`command` 2>> `file`

# Redirección hacía el descriptor `n`

`command` 2>&`n`

# Redirección hacía la nada
```

```
`command` 2> /dev/null

# Redirección de la salida y el error estándar hacía un archivo

`command` &> `file`

# Redirección de la salida y el error estándar hacía un descriptor

`command` >&`n` 2>&1

# Redirección de la salida y el error estándar hacía la nada

`command` &> /dev/null

# Redirección de la salida y el error estándar hacía la nada (alternativo)

`command` > /dev/null 2>&1
```

Las redirecciones se pueden componer como en el caso de `a> [descriptor] b>&a`, significaría que el descriptor `a` apunta hacía el archivo o descriptor especificado y el descriptor `b` apunta hacía el descriptor `a`. De esta manera ambos descriptores estarían enlazados.

Podemos generalizar las redirecciones de los descriptores como:

```
# Descriptores de lectura

`command` `a`<`x` `b`<`y` `c`<`z` ...

# Descriptores de escritura

`command` `a`>`x` `b`>`y` `c`>`z` ...
```

Y podemos establecer una tubería de la salida estándar como

```
# Tubería

`command 1` | `command 2` | ... | `command n`
```

Para crear un nuevo descriptor podemos usar `exec`.

```
SINTAXIS: Crear un descriptor

# Descriptor de entrada

exec `n`< `file`
```

```
# Descriptor de salida

exec `n`> `file`

# Descriptor de entrada y salida

exec `n`<> `file`

# Cerrar un descriptor

exec `n`<&-

# Ejecutar un archivo `fifo`

mkfifo `file`
exec < `file`

# Enviar comandos al archivo `fifo`

exec `n`> `file`

echo `command` >&`n`
```

Veamos algunos ejemplos útiles.

```
# Leer la primer línea de un archivo

read < `file`

# Leer la primer línea de un descriptor

read <&`n`

# Lee los primero `k` caracteres de un archivo

read -n `k` < `file`

# Lee los primero `k` caracteres de un descriptor

read -n `k` <&`n`

# Escribe texto de la salida estándar hacía un archivo

echo `texto` > `file`

# Escribe texto de la salida estándar hacía un descriptor

echo `texto` >&n`
```

```
# Escribe el error estándar hacía un archivo

echo `texto` 2> `file`

# Escribe el error estándar hacía un descriptor

echo `texto` 2>&n`
```

Alias

El comando `alias` nos permite definir un sobrenombre para un comando.

```
SINTAXIS: Crear un alias `name` para un comando `command [options]`

alias `name`=``command [options]`'

EJEMPLO:

alias fecha='date -u'
```

Los alias pueden dejarse permanentes editando el archivo `~/.bashrc`.

Ubicación de comandos

Podemos usar `type <command>` para mostrar la ubicación de comandos.

```
[$] type <command>
```

También podemos usar `whereis <command>` para mostrar la ubicación de comandos y la ubicación de sus manuales.

```
[$] whereis <command>
```

Sustitución de comandos

Se pueden crear variables que sustituyan los comandos mediante `$(<command>)`.

```
SINTAXIS: Sustituir un comando por una variable

<name>=$(<command>)

EJEMPLO:
```

```
fecha=$(date -u)

echo "La fecha es $fecha"
```

Opciones del shell Bash

Podemos consultar las opciones activadas y desactivadas del shell con `set -o`.

```
# LISTA DE OPCIONES

[$] set -o
```

De la lista podemos activar una opción (`on`) con `set -o <option>`.

```
# ACTIVAR UNA OPCIÓN

[$] set -o <option>
```

De la lista podemos desactivar una opción (`off`) con `set +o <option>`.

```
# DESACTIVAR UNA OPCIÓN

[$] set +o <option>
```


Instructor Alan Badillo Salas

Estudié **Matemáticas Aplicadas** en la Universidad Autónoma Metropolitana, posteriormente realicé una Maestría en **Inteligencia Artificial** en el Instituto Politécnico Nacional.

He impartido cursos de Programación Avanzada en múltiples lenguajes de programación, incluyendo *C/C++, C#, Java, Python, Javascript* y plataformas como *Android, IOS, Xamarin, React, Vue, Angular, Node, Express*. Ciencia de Datos en *Minería de Datos, Visualización de Datos, Aprendizaje Automático y Aprendizaje Profundo*. También sobre *Sistemas de administración basados en Linux, Apache, Nignx* y *Bases de Datos SQL y NoSQL* como MySQL, SQL Server y Mongo. Desde hace 7 años en varios instituciones incluyendo el *IPN-CIC, CST, KMMX, The Inventor's House, Auribox*. Para diversos clientes incluyendo al *INEGI, CFE, PGJ, SEMAR, Universidades, Oracle, Intel y Telmex*.
