Mi primer script en bash

Así que has decidido meterte en el mundo del scripting? Interesante, pues es tu día de suerte, en los próximos pasos te guiaremos para que crees tu primer script! y además sentaremos las bases para que hagas scripting básico por tu cuenta, empecemos:

Paso 1:

Lo primero que debemos hacer es abrir nuestra terminal, hecho esto creamos un archivo de texto plano. Podemos usar *vim* o el comando *touch*.

Ej1: `touch miPrimerScript.sh`
Ej2: `vim miPrimerScript.sh`

Es muy importante que tenga la extensión *.sh*!

Para este punto deberías tener en el directorio en el que estás parado un archivo vacío llamado miPrimerScript.sh, (usa *ls* para comprobarlo!). Como dijimos anteriormente es un archivo de texto plano, esto quiere decir que lo podemos editar como si fuera un archivo .txt de toda la vida!

Paso 2:

Ahora siguiendo con la tradición de solo usar la terminal, editaremos nuestro archivo.sh usando vim. (si nunca usaste vim ve a la sección *vim en 5 minutos*)

Abrimos el archivo con el comando `vim miPrimerScript.sh`.

Al igual como ocurre en otros entornos de la programación, necesitamos tipear una serie de instrucciones básicas antes de empezar a desarrollar, en nuestro caso son 3 lineas que debe ir al principio de todo archivo script que vayamos a crear:

```
# !/bin/bash
# Author: ElBrujas
# Mi primer script
```

La primera línea es obligatoria para que funcione nuestro script.
Las siguientes líneas indican el nombre del autor y la descripción de lo que hace el script, respectivamente, estas son opcionales, pero se considera buena práctica agregarlas.

Paso 3:

Ahora podemos introducir todos los comandos que vimos hasta el momento y muchos mas!
Debes respetar siempre la regla de un comando por línea!
Recuerda que los comandos se irán ejecutando secuencialmente de arriba hacia abajo.

Empecemos con algo sencillo; hagamos que se imprima por pantalla la frase mas célebre en el mundo de la programación:

```
echo “HOLA MUNDO!”
```

El archivo debería tener esta pinta para este momento:

```
# !/bin/bash
# Author: ElBrujas
# Mi primer script

echo “HOLA MUNDO!”
```

(*no olviden guardar los cambios*)

Paso 4:

Bueno, ya creamos nuestro *sh*, ahora qué? Ahora debemos ejecutarlo!!
Devuelta en la terminal, vamos al directorio donde se encuentra nuestro *sh*, y procedemos a introducir el siguiente comando:

```
./miPrimerScript.sh
```

Epa! A menos que seas superusuario, el comando anterior seguro explotó!
El problema es que no tiene permisos de ejecución (para ver mas sobre permisos ir a la sección *permisos en 5 minutos*).

Vamos a arreglar esto con el siguiente comando:

```
chmod +x miPrimerScript.sh
```

y volvamos a intentarlo…

Paso 5:

¡Felicidades! Para este punto deberías haber visto el siguiente mensaje en tu terminal:

```
HOLA MUNDO!
```

Esto es solo un punto de inicio, puedes concatenar tantos comandos como quieras y se ejecutarán todos en una milésima de segundo con solo ejecutar el script!!

Por ejemplo, analicemos lo que pasa en el siguiente script:

```
# !/bin/bash
# Author: ElBrujas
# Mi primer script

pwd
cd /
ls
```

Primero imprimirá por pantalla el directorio donde se encuentra parado el script.
`/home/elbrujas/…`

Luego se moverá al *root.*

Y para finalizar mostrará por pantalla los archivos del root.

```
bin   dev  home  lib32  libx32      media  opt   root  sbin  sys  usr
boot  etc  lib   lib64  lost+found  mnt    proc  run   srv   tmp  var
```

Además de todo esto también puedes utilizar las redirecciones de inputs y outputs (y pipes que lo veremos mas adelante). Por ejemplo un script como el siguiente:

```
# !/bin/bash
# Author: ElBrujas
# Mi primer script

history > history.txt
```

Al ejecutarlo debería generarte un archivo .txt con la información del historial de comandos de tu terminal.

Scripts con parámetros:

Al igual como las funciones en la mayoría de lenguajes de programación, en tus scripts bash también puedes usar parámetros, a continuación te diremos como.

Como ejemplo haremos un script básico que recibe el nombre y el apellido como parámetros y los imprime por pantalla.

Paso 1:

Primero debemos definir las variables en nuestro archivo:

```
# !/bin/bash
# Author: ElBrujas
# Script identificador

nombre = $1
apellido = $2
```

El símbolo $1 indica que el primer parámetro que se espera, será asignado a la variable *nombre* y el segundo a la variable *apellido*.

Es decir, se espera que a la hora de ejecutar el script se lo haga de la siguiente forma:

```
./identificarme.sh unNombre unApellido
```

Paso 2:

Ahora ya podemos utilizar esas variables para trabajar en nuestro script, por ejemplo, si queremos imprimirlas por pantalla con un texto saludando.

```
# !/bin/bash
# Author: ElBrujas
# Script identificador

nombre = $1
apellido = $2

echo “Hola, tu nombre es $nombre y tu apellido es $apellido”
```

El símbolo $ en las variables es justamente para decirle al intérprete que queremos usar la variable que definimos previamente y que no es solo una palabra mas.

Nota:

A la hora de ejecutar los scripts con parámetros, bash utiliza los espacios para separar los mismos, es decir:

```
./script.sh parametro1 parametro2 parametro3
```

Y si tengo un parámetro que lleva espacios? ¿Puedo hacer esto?:

```
./script.sh parametro con espacio1 parametro2 parametro3
```

La respuesta es NO, estaría interpretando partes del primer parámetro como el segundo y el tercero, y eventualmente explotará o funcionara incorrectamente.

Entonces como hago?

Para estos casos es que existe el uso de “ ”.
Así la forma correcta de mandar un parametro que lleva espacios seria:

```
./script.sh “parametro con espacio1” parametro2 parametro3
```

Paso 3:

¡Tu script ya esta listo y a la espera de que lo ejecutes! Pruébalo por tu cuenta y esperamos te animes a crear uno nuevo usando esta característica.

Vim en 5 min

Si llegaste hasta acá es por que quieres aprender lo básico de vim, lo suficiente como para editar un archivo de texto y guardar los cambios, bueno es justo lo que mostraremos a continuación, sin rodeos.

Abrir vim

Vim funciona sobre la terminal, por lo que necesitaremos un comando para abrirlo.
El comando en cuestión es:

```
vim
```

Pero si introduces el comando sin mas solo aparecerá una ventana con datos de asistencia.
Para empezar a editar un archivo con vim debemos invocarlo como parámetro:

```
vim rutaDelArchivoAEditar
```

Si el archivo no existe este se creará uno nuevo.

Dentro de vim

Ahora que estás dentro de vim, de seguro intentaste tipear unos cuantos caracteres, y nada paso. Eso es por que vim tiene 2 estados, el estado de visualización, y el estado de edición. Por default al abrir un archivo se encuentra en estado de visualización. Si queremos pasar al estado de edición, debemos apretar la tecla *i*. Para salir de este estado debemos apretar la tecla *Esc*.

Comandos en vim?

Bueno ahora que ya hiciste tus primeras líneas, te preguntarás ¿cómo salgo de aca? cierro la terminal de una?
La respuesta a esto son los comandos de vim, si, parecido a la terminal.

Para empezar a escribir un comando deberás apretar la tecla *“ : ”* , y verás que se habilita la barra del input en la parte inferior.

Primero el comando mas solicitado por toda latinoamerica unida (de programadores), como salir de vim, solo debemos ingresar el comando:

```
:q
```

Pero no te dejara salir así sin mas si no guardaste los cambios, para eso tenemos el comando:

```
:w
```

que guarda los cambios hechos.

Pero si de todas formas queremos salir sin guardar los cambios podemos gritarle:

```
:q!
```

Esto hará salir el programa ignorando los cambios no guardados.

Podemos combinar comandos, por ejemplo:

```
:wq
```

Guardará los cambios y saldrá del programa (el comando `:x` hace lo mismo)

FIN

Felicidades ya sabes lo necesario de vim para editar un archivo de texto, para mas comandos e información del programa puedes usar el comando *:help* dentro del editor o el comando *man vim* en bash.

Permisos en 5 min

¿Qué es esto de los permisos? Bueno los archivos en linux tienen distintos grados de permisos para ciertos usuarios.
De cierta forma puedes visualizarlos con el comando *ls -l*. Hace lo mismo que el comando *ls* solo pero ahora puedes ver al principio de los nombres de los archivos sus respectivos permisos.

```
elbrujas@supmku-unix:~/Documentos/carpetaPruebas$ ls -l
total 28
-rw-rw-r-- 1 elbrujas elbrujas 13830 abr 13 21:17 history.txt
-rw-rw-r-- 1 elbrujas elbrujas   165 abr 13 21:40 scriptbase
-rwxrwxr-x 1 elbrujas elbrujas    65 abr 17 23:54 scriptPiola.sh
-rwxrwxrwx 1 elbrujas elbrujas    12 abr 13 21:00 texto.txt
-rw-rw-r-- 1 elbrujas elbrujas     0 abr 17 23:05 text.txt
```

Ignorando el primer guión, puedes ver como hay 9 caracteres, cada carácter representa un tipo de permiso:

- r: permiso de lectura
- w: permiso de escritura
- x: permiso de ejecución

Pero, ¿por qué hay 9? Es por que se dividen en ternas entre los 3 distintos tipos de usuario:

- La primera terna hace referencia a los permisos del dueño del archivo
- La segunda terna hace referencia a los permisos de los usuarios del mismo grupo que el dueño
- La tercera terna hace referencia a los permisos de los usuarios que no pertenecen a ninguno de los dos grupos mencionados anteriormente.

Entonces, si en una terna tenemos la siguiente combinación de permisos:

```
rw-
```

Quiere decir que para ese usuario el archivo tiene permisos de escritura y de lectura, pero no de ejecución (como el script.sh que probamos al principio).

Una terna con todos los permisos se ve así:

rwx

Ahora veamos el siguiente ejemplo:

rwx `---` `---`

Este archivo tiene todos los permisos habilitados, pero solo para el dueño (recordar la primera terna son para los permisos del dueño).
Para los usuarios del mismo grupo y todos los demás, no tiene ningún permiso habilitado.

Cambiar permisos

¿Puedo cambiar estos permisos? Si, con el comando *chmod*.
Este comando recibe un número de 3 cifras como primer parámetro y la ruta del archivo que vamos a modificar como segundo parámetro:

Ej: `chmod 347 miScript.sh`

Pero qué son estos números? Bueno cada dígito representa una terna (el primero para el dueño, el segundo para el grupo y el tercero para todos los demás)

El número que mandemos, representa la combinación de permisos que queremos para esa terna.

Por ejemplo un 7 representa una combinación rwx y un 5 una combinación r-x.

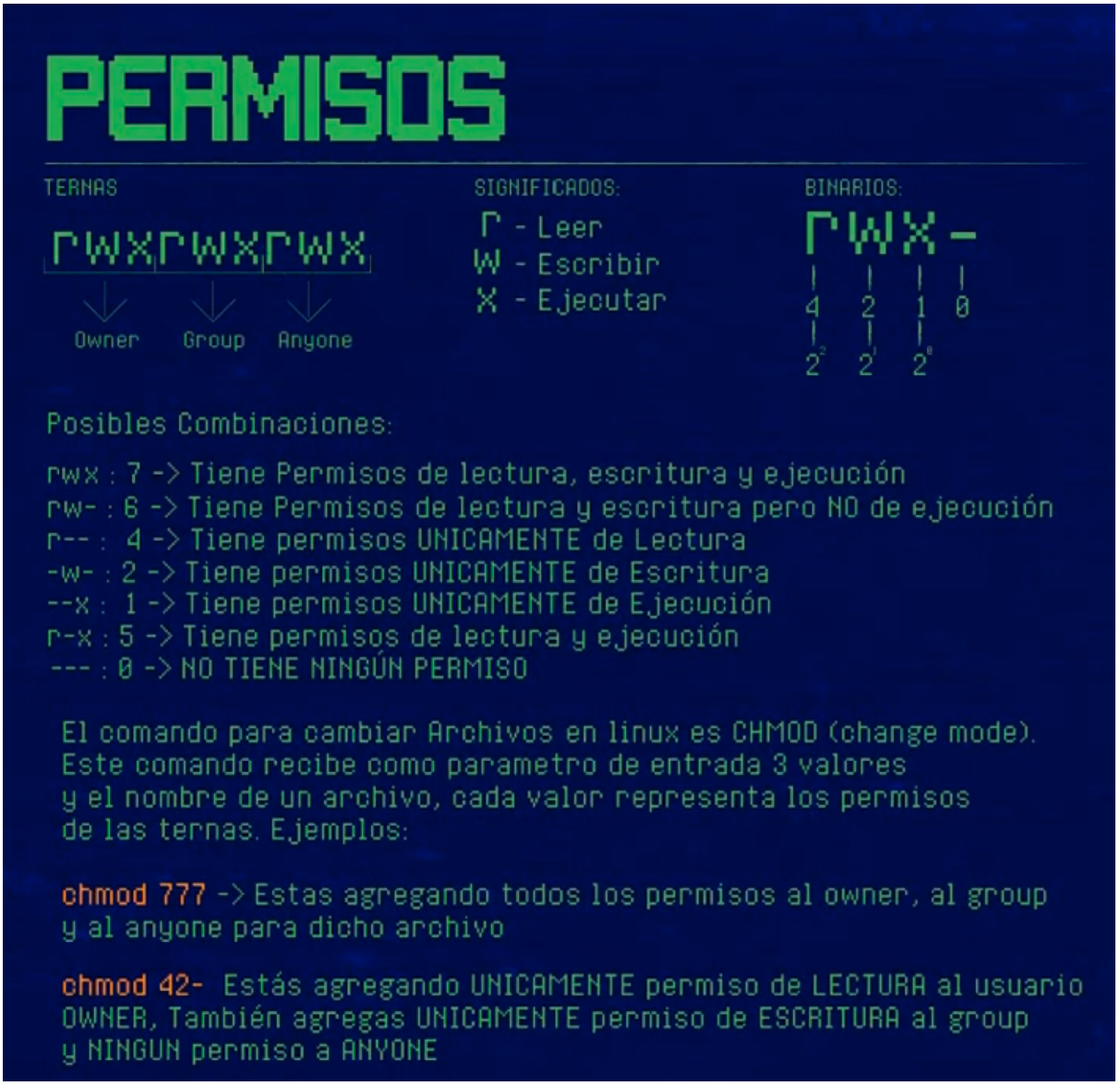
Entonces si queremos que un archivo tenga todos los permisos habilitados para todos los tipos de usuarios usamos el siguiente comando:

```
chmod 777 miScript.sh
```

Si quisiéramos quitarle los permisos de escritura solo a la tercera terna usamos el siguiente comando:

```
chmod 775 miScript.sh
```

Pero, ¿qué número uso? Te dejo la tabla de abajo para que sepas que combinación representa cada número:



Che y el comando que nos hiciste poner en la parte de mi primer script?

```
chmod +x miScript.sh
```

Es un comando especial que agrega permisos de ejecución a las 3 ternas si es que no lo tienen habilitado. Es un comando mas especifico, hay mas como estos, le animamos a investigar por su cuenta todo lo que puede hacer este comando.