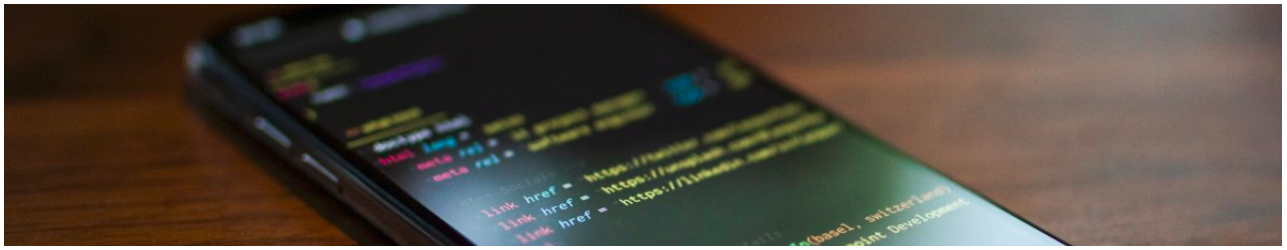
SCRIPTS EN BASH

Actualizado el 7 de junio de 2021 por atareao *52.7K*

Por poco que hayas trabajado con el terminal de Linux, seguro que en mas de una ocasión te has sorprendido a ti mismo repitiendo instrucciones. Realizar **tareas repetitivas** tiene dos problemas, por un lado que es algo realmente tedioso. Por otro lado que estás abocado al error. Si, es cierto, que cuanto mas veces repites una tarea menos te equivocarás, porque habrás adquirido mas destreza, pero la probabilidad de equivocarte es la misma. Pero, lo que es seguro, es que si **automatizas una tarea repetitiva**, no te volverás a equivocar nunca. Precisamente este es el objetivo de este tutorial, crear scripts en Bash, que te permitan automatizar tus tareas.

Aunque puedas pensar que los scripts en Bash son algo mas para sistemas, lo cierto es que lo puedes emplear para cualquier cosa que te puedas imaginar. Así, por ejemplo, si trabajas en *Markdown*, puedes automatizar la conversión a PDF, cada vez que edites un archivo, o cada cierto tiempo. Y por supuesto, no puedo dejar de lado las **copias de seguridad**. Una copia de seguridad es la tarea repetitiva por excelencia, y por tanto es **carne de cañón** para realizar scripts en Bash. Y por supuesto, no puedes dejar de lado a tu pequeña Raspberry Pi, donde puedes automatizar tareas que *ella* se encargue de llevar adelante.

Estas son algunas razones para ponerte manos a la obra y aprender a crear tus propios scripts en Bash. Eso si, antes que te lances a aprender a crear tus propios scripts en Bash, te recomiendo que si no te desenvuelves con soltura en el terminal, le des un repaso al tutorial sobre el terminal.



SCRIPTS EN BASH

¿QUE ES BASH?

Bash es un intérprete de comandos (lo que normalmente puedes encontrar en la literatura anglosajona como *shell*). Se trata de una *interfaz de usuario* con la que relacionarte con el sistema operativo con el que estás trabajando. Así, **esta aplicación es capaz de leer y ejecutar instrucciones**. Básicamente es el encargado de decirle al sistema operativo lo que tu quieres hacer. Posteriormente, se encargará de devolverte la respuesta del sistema operativo, en el caso de que la haya.

Además de Bash existen otros muchos intérpretes de comandos, como `sh`, o el

intérprete de comandos de Korn, conocido como `ksh` o el *C shell*, `csh`. Precisamente, una de las características de Bash, es que es altamente compatible con estos otros intérpretes de comandos.

UN POQUITO DE HISTORIA SOBRE BASH

Por ponerte en contexto, indicarte que **Bash** es un acrónimo de **B**ourne-**a**gain **sh**ell. Un juego de palabras que viene a significar algo así como **nacido de nuevo**. *Bourne* fue el intérprete de la versión 7 de Unix, desarrollado inicialmente por el investigador *Stephen Bourne*. Mientras que **Bash**, es el equivalente implementado por *Brian Fox*, para el proyecto GNU, cuya versión beta fué liberada en 1989.

Desde ese momento, Bash se ha convertido en el intérprete de comandos mas popular de los utilizados en Linux. No solo eso, sino que además, hoy por hoy, es el intérprete por defecto de la mayoría de distribuciones Linux, e incluso de macOS. Actualmente, Bash, también lo puedes encontrar en algunas versiones de Linux. Todo esto te puede hacer una idea del alcance de este intérprete de comandos.

¿PORQUE NECESITAS SCRIPTS EN BASH?

Uno de las principales razones para iniciarte en el mundo del scripting es sin lugar a dudas la **automatización**. Pero, ¿para que automatizar? Simplemente para reducir el número de errores. Un proceso que hayas automatizado, que hayas mecanizado, es un proceso que tiene menos probabilidades de que falle. Que no te digo que no falle, pero, has quitado un factor intermedio, el factor humano.

Por ejemplo, un trabajo que deberías de hacer, es una **copia de seguridad** de tus documentos mas importantes. O de tus archivos de configuración o de lo que tu consideres. Esta trabajo, es un trabajo realmente tedioso, y que normalmente pospones en favor de otros. Es una procrastinación en toda regla. O simplemente olvidas que lo tienes que hacer, o piensas que ya lo has hecho.

Si las copias de seguridad, las tienes automatizadas, y no solo esto, sino que además programas las tareas con cron, tal y como te explico en el artículo del enlace, ya no te tienes que preocupar de nada. Bueno, de tanto en tanto, tendrás que comprobar que se ha realizado la copia de seguridad. O ni siquiera de eso. Puedes hacer que, en caso de que se produzca un error al realizar la copia de seguridad, te mande un mensaje, por ejemplo a tu cuenta de Telegram o Twitter, por decirte algo.

Pero no solo esto, y si tienes que hacer una operación de forma repetitiva, por ejemplo, hacer un `ping` a todas las direcciones de una red. ¿Tienes que hacerlo una a una?¿No será mejor programarlo para no estar delante del teclado repitiendo una y otra vez el mismo comando?

Y LLEGÓ LA RASPBERRY

Por último, no quiero dejar de lado a la pequeña **Raspberry Pi**. Si te gusta el *cacharreo*,

si quieres sacarle el máximo partido a este pequeño, gran ordenador, no puedes dejar de aprender a hacer tus propios scripts en Bash. Porque, es posible que te suceda como a mi. Que empieces por comprar una Raspberry Pi, y encontrarte al cabo de los meses, con toda una familia de Raspberry... y ahora, tienes que instalar y configurar a cada una de ellas, el sistema operativo. Instalar las herramientas que sueles utilizar, vamos, dejarlas como tu quieres... **¿es necesario repetir una y otra vez lo mismo?**, **NO**, con tus scripts en Bash, puedes pasar por encima de esto sin ningún problema. Simplemente ejecutas tus scripts en Bash, y ya tienes tu Raspberry como tu quieres....

¿QUE NECESITAS PARA APRENDER A REALIZAR SCRIPTS EN BASH?

Lo cierto es que poco. Actualmente, como te indicaba anteriormente, en cualquier sistema operativo ya puedes empezar a realizar tus scripts en Bash. De hecho, con `docker` puedes empezar a realizar tus scripts en Bash, dentro de un Ubuntu, en un *periquete*. O de un Debian, así, por ejemplo, si quieres hacer tu primer script en un Debian, tan solo tienes que ejecutar las siguientes líneas,

```
docker pull debian
docker run -it debian
```

UN EDITOR DE TEXTO

A parte de un sistema operativo que tenga Bash, también necesitarás un editor de texto. Aquí, puedes utilizar **cualquier editor de texto** que venga con tu sistema operativo. Así, por ejemplo, puedes utilizar `gedit` en Ubuntu. Sin embargo, mi recomendación, es que **te decantes por uno que esté en el propio terminal**. De esta forma, lo tendrás todo de la mano.

Si no conoces que editores puedes utilizar en el terminal, te recomiendo el capítulo sobre visores y editores. En particular, si estás empezando, y todavía no te manejas con soltura, **mi recomendación** es nano. Este editor, es al que me referiré durante este tutorial para hacer los scripts, de forma que te resulte mas sencillo e intuitivo.

ACABAS DE LLEGAR A LINUX

Si acabas de llegar a Linux o todavía no has empezado a utilizar el terminal, **ha llegado el momento**. En este sentido, te recomiendo que comiences con el tutorial sobre el terminal.

Este tutorial trata de ver las características mas interesantes del terminal. El objetivo, no solo es que aprendas a utilizar el terminal, sino que llegues a sacarle el máximo partido posible. En este sentido, puedes ver como navegar por el sistema de archivos o gestionar archivos, hasta operaciones como procesar archivos o incluso realizar búsquedas.

Sea como fuere, el paso lógico, sería comenzar con el tutorial sobre el terminal, para continuar con este tutorial, sobre scripts en Bash. Pero con, independencia de lo que quieras hacer, sin lugar a dudas, **anímate y comienza con tus scrips en Bash**.

NO TIENES NI IDEA DE PROGRAMACIÓN

Otra excusa que puedes ponerte a ti mismo, para no querer hacer scripts en Bash es que *no tienes ni idea de programación*. Pero, precisamente, esto no debería ser un escollo, mas bien todo lo contrario, **debería ser un acicate**.

Gracias a este tutorial, empezarás a realizar tus primeros programas. Porque si, en Bash, también puedes hacer programas tan complicados como tu quieras. Al final, un script no es mas que un programa, mas o menos complicado. Ahí es tu imaginación la que pone los límites al programa o script que quieras hacer.

MI PRIMER SCRIPT

Aunque este sea un capítulo de introducción, **no puedes irte sin crear y ejecutar** tu primer script. Por supuesto, que será un `Hola mundo`, como debe ser, pero será tu primer script... De paso verás la anatomía de un script.

INSTALANDO EL EDITOR

El primer paso es instalar el editor que utilizarás durante todo el tutorial, que como te he comentado será **nano**. Para hacer esto, tan solo tienes que ejecutar la siguiente instrucción en el terminal,

```
sudo apt update
sudo apt install nano
```

PRIMEROS PASOS CON NANO

Una vez has instalado nano, una instalación que dura apenas unos segundos, lo tienes que iniciar. Para iniciarlo, tan solo tendrás que ejecutar `nano`. Directamente ya puedes escribir lo que quieras. Para trabajar con `nano` necesitas conocer algunos atajos de teclado imprescindibles que podrás encontrar en el artículo sobre nano. Sin embargo, te digo aquí lo más básico, para que puedas empezar con tu primer script,

- `Ctrl+R` para abrir un archivo
- `Ctrl+O` para guardar el archivo
- `Ctrl+X` para salir

Vamos allá, crea tu primer script. Ejecuta `nano primero.sh`. Pega el siguiente contenido, y guarda utilizando `Ctrl+O`. Una vez terminado, utiliza `Ctrl+X` para salir del editor.

Si te das cuenta, en la parte inferior del editor están las principales operaciones que puedes realizar con `nano`.

```
#!/bin/bash
echo Hola mundo
```

Si todo ha ido bien, cuando ejecutes `cat primero.sh` verás el contenido introducido. Si ahora, ejecutas `bash primero.sh` te mostrará el esperado `Hola mundo`. Pero podemos hacerlo todavía mas cómodo, podemos darle permisos de ejecución. Para ello, simplemente ejecuta,

```
sudo chmod +x primero.sh
```

Ahora ya puedes ejecutar tu script directamente con `./primero.sh`. Si te preguntas la razón para poner `./` antes del nombre de tu script, es que Bash interpreta, **únicamente** los comandos que se encuentran en `$PATH`, y el directorio en el que te encuentras no se encuentra por razones de seguridad.

ANATOMÍA DE UN SCRIPT

En este primer script has introducido dos líneas. **La primera línea** es la que indica el intérprete que se debe utilizar. Se conoce como shebang. Es importante que recuerdes que tienes que indicar la ruta completa.

En el caso de tu primer script, en la primera línea indicas, que el intérprete será `bash`, indicando la ruta completa, `/bin/bash`. Igualmente que indicas que es `bash`, podría ser cualquier otro, como `sh` o incluso `python` o `nodejs` en el caso que quisieras hacer el script con Node y JavaScript.

La segunda de las líneas haces un `echo`. Este comando escribe el argumento que le pasas a la salida estándar. Así cuanto le has dicho `echo Hola mundo`, lo que hace es escribir en la salida estándar `Hola mundo`. En el caso quisieras escribirlo en un archivo en lugar de en la salida estándar, tan solo tienes que redirigir la salida, tal y como explico en el capítulo sobre redirigir entrada y salida del tutorial sobre el terminal.

CONCLUSIÓN SOBRE SCRIPTS EN BASH

Ya has visto lo sencillo que ha sido crear tu primer script. Si además tampoco tienes conocimientos de programación, has visto que esto del `Hola mundo` está sobrevalorado.

Sea como fuere, ahora todo es cuestión de profundizar en el maravilloso mundo del scripting. Así que te invito a próximas entregas.

Listado de capítulos

Estos son todos los capítulos del tutorial Scripts en Bash,