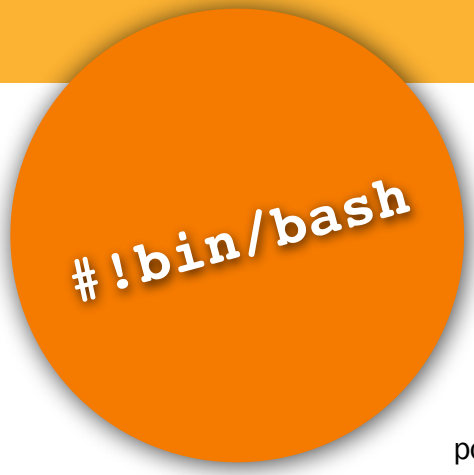
Programación en Shell
Primeros pasos

Francisco de Asís Conde Rodríguez



El sistema operativo Linux ofrece a su administrador cientos de órdenes muy potentes y flexibles listas para ser ejecutadas, con las que podrá realizar la mayoría de sus tareas cotidianas.

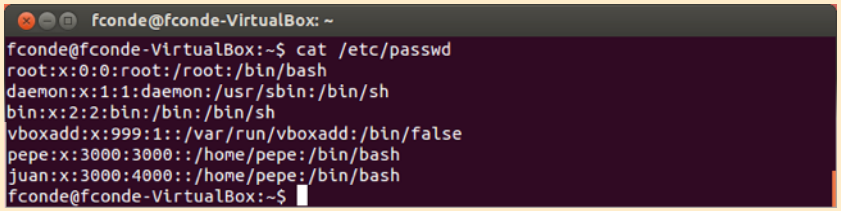
Sin embargo, siempre existen tareas repetitivas, o que incluyen secuencias largas de órdenes, o que incluyen órdenes con sintaxis compleja difícil de recordar, que es mejor tener automatizadas para ahorrar tiempo, y sobre todo, para evitar errores en su realización.

La forma en que se puede automatizar una tarea de administración en Linux es mediante pequeños programas que se pueden ejecutar en la shell, llamados shell scripts.

¿Por qué necesito saber programar shell scripts?

Veamos un ejemplo: Una de las tareas de auditoría que un administrador de sistemas Linux debe realizar es comprobar si existen en el sistema dos usuarios con la misma UID, ya que pueden acceder a los ficheros del otro usuario sin que el sistema pueda distinguir quien es quien. Eso incluso aunque tengan nombre de usuario distinto ya que el sistema usa siempre la UID.

Esta situación anómala, se puede comprobar, listando el contenido del archivo `/etc/passwd` y comprobando que no haya dos líneas en las que la tercera columna sea igual.

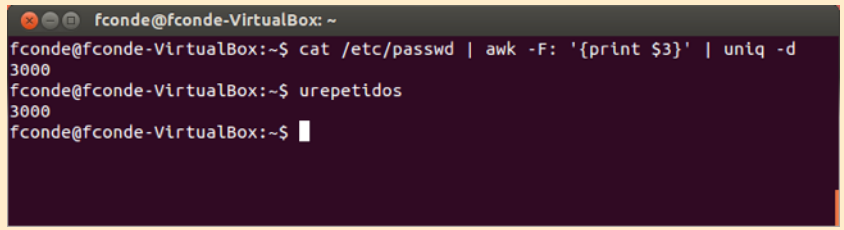


En este ejemplo, los usuarios pepe y juan, ambos tienen UID 3000, lo cual es un error muy grave, que el administrador del sistema debe comprobar y reparar.

El archivo `/etc/passwd` puede contener muchos registros, por lo que la comprobación manual es muy pesada y propensa a errores si las entradas con UID repetida están muy separadas entre sí. Una forma mejor de comprobar esta situación es mediante la orden[1]:

```
cat /etc/passwd | awk -F: '{print $3}' | uniq -d
```

Es una orden difícil de recordar y escribir, es mejor automatizarla mediante un shell script al que podemos llamar `urepetidos` que hace el mismo trabajo pero mucho mas fácil.



[1] No te preocupes si no entiendes la orden. Ese no es el objetivo por ahora.

Un shell script (o programa shell) es un archivo de texto que contiene una o más órdenes que realizan una tarea concreta, la tarea que se quiere automatizar.

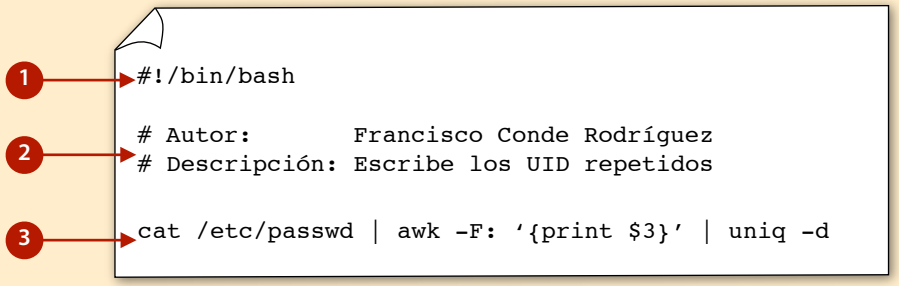
Son las mismas órdenes que escribiríamos si estuviésemos realizando la tarea a mano. La diferencia es que están escritas en un archivo de texto para que el intérprete de órdenes las ejecute cuando se invoque al shell script.

Cuando se ejecuta un shell script, el intérprete de órdenes, lee las órdenes. una a una, del archivo y las ejecuta por orden.

Los programas de shell no se compilan, sólo hay que escribirlos mediante un editor de textos y darles permisos de ejecución para poder usarlos.

Anatomía de un shell script

Un shell script es un archivo de texto que contiene una o varias órdenes para el sistema. Por ejemplo, el archivo `urepetidos` que contiene el programa del ejemplo anterior tiene en su interior:



Archivo `urepetidos`

Todos los shell scripts deberían tener las siguientes partes:

(1) **Cabecera (hash-bang).** La primera línea de un shell script es un tipo especial de comentario. Comienza por los caracteres `#!` y va seguido del nombre de un interprete de ordenes o shell. Dice al sistema qué intérprete de órdenes debe usar para ejecutar el archivo. En este ejemplo `/bin/bash`.

IMPORTANTE: Los caracteres `#!` deben aparecer **justo al comienzo** del archivo, es decir, deben ser los dos primeros bytes del archivo de texto para que el sistema los pueda reconocer.

(2) **Comentarios.** Se indican con el carácter # al comienzo de una línea o palabra. Todos los shell scripts deberían incluir como mínimo quien es su autor y una breve descripción de para qué sirve el script. El intérprete ignora el comentario desde el signo # hasta el final de la línea.

IMPORTANTE: Un carácter #, sólo se interpreta como el comienzo de un comentario, si va justo al principio de una línea o de una palabra de un shell script. Si va en medio o al final de una palabra de un shell script, no se interpreta como comentario.

(3) **Ordenes**. Las órdenes que componen el shell script. Se escriben de la misma forma que se escriben en la consola cuando se ejecutan a mano.

¡Hola mundo! nuestro primer shell script

Vamos a escribir nuestro primer shell script, un sencillo programa llamado `holamundo` que simplemente escriba por pantalla la frase: `¡Hola mundo!`.

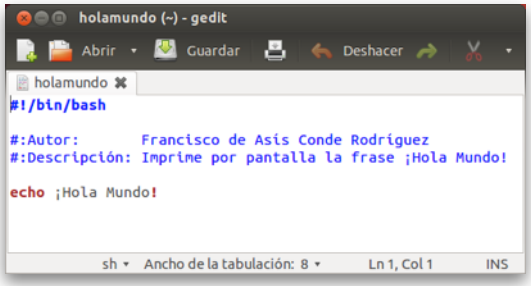
Si no estamos en nuestro directorio base ~, tecleamos la orden `cd` sin argumentos para ir hasta él.

(1) Abrir un editor de textos y escribir el programa.

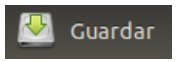
Para ello ejecutamos:

```
gedit holamundo &
```

Escribimos el texto que aparece en la figura.



Pulsamos el botón Guardar.



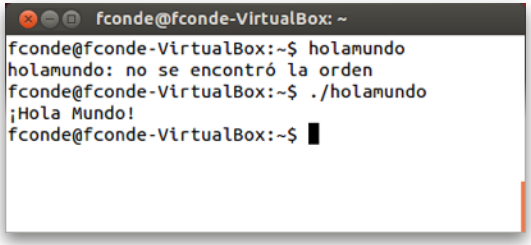
(2) Dar permisos de ejecución al archivo. Para ello ejecutamos:

```
chmod +x holamundo
```

(3) Ejecutar el archivo escribiendo su nombre:

```
./holamundo
```

Importante: si simplemente se escribe `holamundo`, el sistema no encuentra al programa y no lo ejecuta. Por tanto siempre hay que especificar la ruta completa.



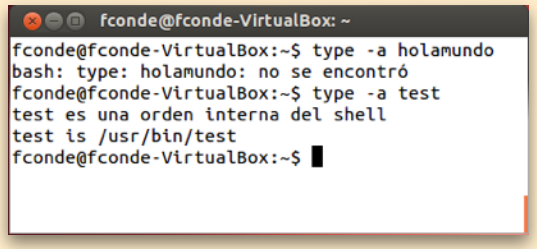
Nota: El signo `&` final de la orden `gedit holamundo &`, indica que el proceso `gedit` se ejecute en background. En este ejemplo se hace así para no bloquear la consola y poder seguir ejecutando órdenes en ella.

¿Cómo llamar a un shell script y dónde colocarlo?

Podemos dar a nuestros shell script cualquier nombre que queramos, sin embargo, si ese nombre ya pertenece a alguna orden interna del intérprete de órdenes, o a alguna orden del sistema, puede causar confusiones.

Por ello, se recomienda que antes de elegir un nombre para nuestro script, comprobemos que no pertenece a ninguna orden del sistema o del intérprete. Para ello existe la orden `type -a [nombre script]` donde `[nombre script]` se sustituye por el nombre que queramos comprobar.

Por ejemplo, podemos comprobar que el nombre `holamundo` no pertenece a ninguna orden, mientras que el nombre `test` sí.

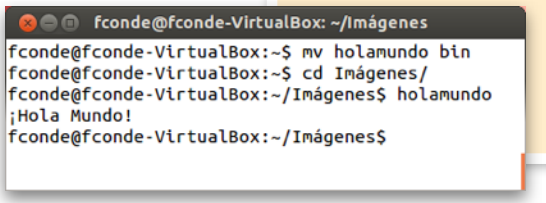


Podemos colocar el shell script en cualquier carpeta donde tengamos permisos, sin embargo se recomienda tener una carpeta específica para los ejecutables llamada `bin`, que cuelgue directamente del directorio base del usuario `~/bin`.

En Ubuntu Linux, la ruta `~/bin` se añade automáticamente al PATH, con lo que los programas que se coloquen en esa carpeta se pueden ejecutar directamente aunque no estemos situados en ese directorio.

Para crear la carpeta `~/bin` escribimos:

```
mkdir ~/bin
```



Cerramos la sesión y volvemos a entrar, a partir de ese momento, para ejecutar los programas que hayamos copiado en la carpeta `bin` sólo hay que escribir su nombre (sin `./`), y el intérprete los encontrará, da igual el directorio en donde estemos.

Parámetros

Para que un shell script sea realmente útil, debe usar parámetros. Según la ayuda de manual que acompaña a Ubuntu Linux, un parámetro es *"una entidad que almacena valores"* y puede ser de tres tipos: parámetros posicionales, parámetros especiales y variables.

(1) **Parámetros posicionales.** Son los argumentos presentes en la línea que invoca a la orden (la llamada) y se referencian mediante números que indican su posición.

Por ejemplo, en la orden `ls /bin`, hay un parámetro posicional, `/bin`, que ocupa la posición 1.

(2) **Parámetros especiales.** Su contenido lo rellena el intérprete de órdenes para guardar información sobre su estado actual, como por ejemplo el número de parámetros posicionales de la orden que se está ejecutando.

Se referencian mediante caracteres no alfanuméricos, com por ejemplo `*` o `#`.

(3) **Variables.** Las usa el programador del shell script para almacenar cualquier información que necesite. Se referencian mediante un nombre.

El nombre de una variable puede ser cualquier combinación de letras, números o el guión bajo _ y siempre comienza por una letra o el guión bajo.

Por ejemplo, nombres válidos de variables son: `resultado`, `res_1`, o `_res`, pero no `1res`.

Accediendo al valor de los parámetros

Para acceder al valor de cualquier parámetro, se usa el signo $ precedido del identificador del parámetro. También se llama **sustitución de parámetros.**

Por ejemplo, para acceder al contenido del parámetro posicional que ocupa la segunda posición se escribe `$2`, para acceder al contenido del parámetro especial `#`, se escribe `$#`, y para acceder al contenido de la variable resultado, se escribe `$resultado`.

También se puede encerrar entre llaves el identificador del parámetro. Esto permite delimitar bien cual es ese identificador y es el método preferible[1] aunque sea más difícil de escribir. Así, los ejemplos anteriores también se pueden escribir de la siguiente forma: `${2}`, `${#}` o `${resultado}`.

Estableciendo el valor de variables

Para crear una variable y asignarle un valor sólo hay que escribir el nombre de la variable, un signo igual = y el valor. Por ejemplo:

```
nombre=Francisco
```

IMPORTANTE, no debe haber espacios en esa asignación o el sistema dará un error.

```
nombre = Francisco
```

no es correcto.

Para eliminar una variable se usa `unset`. Por ejemplo: `unset nombre`.

[1] En algunos casos es obligatorio. Por ejemplo, si se quiere acceder a un parámetro posicional con identificador mayor que 9, debe hacerse obligatoriamente usando llaves. Por ejemplo ${15}.

Ejemplos de uso de parámetros posicionales

Vamos a escribir un shell script que reciba como argumento un parámetro posicional y que imprima un saludo seguido del valor de ese parámetro posicional. Llamaremos al script `saludo`.

saludo

```
#!/bin/bash

# Autor:       Francisco de Asis Conde Rodriguez
# Descripción: Imprime un saludo usando un parametro
#              posicional

echo Buenos dias ${1}
```

Hace referencia al parámetro posicional que ocupa la primera posición.

```
fconde@fconde-VirtualBox: ~
fconde@fconde-VirtualBox:~$ saludo Juan Antonio
Buenos dias Juan
fconde@fconde-VirtualBox:~$
```

Fíjate que en la llamada al script `saludo` del ejemplo anterior, hay dos parámetros posicionales, juan y antonio, pero sólo se imprime el primero, porque en la orden `echo` sólo se hace referencia al primer de ellos.

¿Para qué se usan los parámetros posicionales?

Para poder pasar argumentos a un shell script, como por ejemplo el directorio en el que trabajar, o el archivo que comprobar; y para pasar opciones a un shell script (recuerda que las opciones se preceden de un guión).

Aquellos shell scripts que necesiten trabajar con argumentos u opciones pueden hacer referencia a ellos mediante el uso de los parámetros posicionales.

Ejemplos de uso de parámetros especiales

Vamos a escribir un shell script que haciendo uso de los parámetros especiales, nos muestre información relevante sobre la ejecución de ese shell script. Llamaremos a ese shell script `muestrainfo`.

*muestrainfo

```
#!/bin/bash

# Autor:       Francisco de Asís Conde Rodríguez
# Descripción: Muestra información relevante sobre la
#              ejecución de este shell script

echo La ruta del script es: ${0}
echo El número de parámetros posicionales: ${#}
echo Los parámetros son: ${*}
echo El PID del proceso actual: ${$}
```

sh ▾ Ancho de la tabulación: 8 ▾ Ln 10, Col 37 INS

fconde@fconde-VirtualBox: ~

```
fconde@fconde-VirtualBox:~$ muestrainfo hola 23 dato
La ruta del script es: /home/fconde/bin/muestrainfo
El número de parámetros posicionales: 3
Los parámetros son: hola 23 dato
El PID del proceso actual: 2353
fconde@fconde-VirtualBox:~$
```

Como podemos ver, en el shell script `muestrainfo`, se usan cuatro parámetros especiales[1]: `0`, `#`, `*` y `$`.

- **0** El parámetro especial `0`, se sustituye por la ruta completa hacia el shell script que se está ejecutando.
- **#** El parámetro especial #, se sustituye por el número de parámetros posicionales que se hayan pasado al shell script durante la llamada, en el ejemplo 3: hola, 23 y dato.
- ***** El parámetro especial *, se sustituye por el valor de todos los parámetros posicionales que se hayan pasado al shell script.
- **$** El parámetro especial $, se sustituye por el identificador del proceso actual.

[1] Hay nueve parámetros especiales en total. El resto: `@`, `?`, `_`, `!` y `–`, se irán viendo en prácticas posteriores, a medida que su uso se vaya haciendo necesario.

Ejemplos de uso de variables

Vamos a escribir un shell script que ilustre el uso de variables. Llamaremos a ese shell script `usavariables`.

usavariables

```
#!/bin/bash

# Autor:       Francisco de Asís Conde Rodríguez
# Descripción: Usa algunas variables para demostrar su uso

var1=hola
var2=1+2
var3=${1}

echo El contenido de var1 es: ${var1}
echo El contenido de var2 es: ${var2}
echo El contenido de var3 es: ${var3}
```

sh ▾ Ancho de la tabulación: 8 ▾ Ln 11, Col 38 INS

fconde@fconde-VirtualBox: ~

```
fconde@fconde-VirtualBox:~$ usavariables parametro
El contenido de var1 es: hola
El contenido de var2 es: 1+2
El contenido de var3 es: parametro
fconde@fconde-VirtualBox:~$
```

Aspectos importantes que hay que conocer sobre las variables

En los programas shell script, los valores de todas las variables se interpretan como **cadenas de caracteres.** Esa es parte de la potencia de los shell script. Permiten un procesamiento muy potente de cadenas de caracteres. Eso se puede comprobar en las líneas 7 y 11 del script `usavariables`. Como puede verse a la variable `var2` se le asigna el valor `1+2` y el intérprete de órdenes lo asigna como la cadena de caracteres `"1+2"`, no como el valor entero `3`.

En la asignación de variables se puede usar el valor de otros parámetros (posicionales, especiales o variables), para ello se usa la sustitución apropiada. En el ejemplo esto se puede ver en la línea 8 del script. Ahí, se le asigna a la variable `var3`, el valor del parámetro posicional `1`.

Ejercicios

1) Escribe y ejecuta el siguiente shell script. Llámalo `pruebavariables01`:

```
#!/bin/bash

# Autor:       Francisco de Asís Conde Rodríguez
# Descripción: Pruebas de uso de variables

var1=hola
var2="hola mundo"
var3=hola mundo

echo El contenido de var1 es: ${var1}
echo El contenido de var2 es: ${var2}
echo El contenido de var3 es: ${var3}
```

¿Qué resultado se obtiene? ¿A qué crees que se debe? ¿Cómo crees que se resuelve?

2) Escribe y ejecuta el siguiente shell script. Llámalo `pruebavariables02`:

```
#!/bin/bash

# Autor:       Francisco de Asís Conde Rodríguez
# Descripción: Pruebas de uso de variables

var1=3
var2=${var1}hola
var3=$var1hola

echo El contenido de var1 es: ${var1}
echo El contenido de var2 es: ${var2}
echo El contenido de var3 es: ${var3}
```

¿Qué resultado se obtiene? ¿A qué crees que se debe? ¿Por qué no se produce un error?

3) Escribe y ejecuta el siguiente shell script. Llámalo `pruebavariables03`:

```
#!/bin/bash

# Autor:       Francisco de Asís Conde Rodríguez
# Descripción: Pruebas de uso de variables

var1=3

echo El contenido de var1 es: ${var1}
unset var1
echo El contenido de var1 es: ${var1}
```

¿Qué resultado se obtiene? ¿Por qué no se produce un error?