El shell bash

1. Función

Aunque las distribuciones de Linux destinadas al público en general permiten saltarse la introducción de instrucciones de texto al ofrecer entornos gráficos atractivos, un profesional de Linux no puede obviar el funcionamiento del intérprete de comandos y de los principales comandos asociados. Después de todo, los servidores Linux que disponen de una interfaz gráfica son poco frecuentes.

El intérprete de comandos, o simplemente intérprete, ejecuta las instrucciones introducidas con el teclado o en un script y le devuelve los resultados. Este intérprete es un programa comúnmente llamado shell. Se puede aproximar a la palabra kernel que vimos antes: el kernel significa núcleo. A menudo, está rodeado de una concha dura (piense en un hueso de albaricoque o melocotón, o los frutos secos, como las nueces o las avellanas). Como «shell» significa concha, viene a decir que es lo que rodea al **núcleo** de Linux: se utiliza mediante comandos. Por lo tanto, es una interfaz que funciona en modo texto entre el núcleo de Linux y los usuarios (avanzados) o las aplicaciones.

Hay varios shells: cada uno dispone de especificaciones propias. El Bourne Shell (sh) es el shell más conocido y habitual en los Unix. El C-Shell (csh) retoma la estructura del lenguaje C. El Korn Shell (ksh) es una evolución del Bourne Shell. El Z-Shell (zsh) es a su vez una evolución del Korn Shell. El shell de referencia en Linux se llama Bourne Again Shell (bash). A continuación le presentamos una lista exhaustiva de intérpretes de comandos que puede encontrar en Linux:

- sh: Thompson Shell (ya no existe);
- sh: Bourne Shell (sustituyó al anterior);
- bash: Bourne Again Shell;
- ksh: Korn Shell;
- csh: C Shell;
- zsh: Z Shell;
- tcsh: Tenex C Shell;
- ash: A Shell;

- dash: Debian Almquist Shell.

La lista de los shells disponibles en su instalación de Linux está en el archivo /etc/shells.

2. Bash: el shell por defecto

a. Un shell potente y libre

El bash es un derivado de Bourne Shell. Bourne es el nombre del principal programador de este shell. La expresión «Bourne Again» es un guiño a los orígenes del bash (Bourne) y un juego de palabras en «I born again», lo que significa «he nacido otra vez» o «reencarnado». El bash retoma sh pero también funcionalidades de ksh o chs.

El bash no sólo está en Linux. Al ser un programa libre, se puede compilar o ejecutar en numerosas plataformas. Es el shell de referencia en los sistemas macOS hasta la versión 10.14 (reemplazado por zsh) y existe también para Windows, con la ayuda de soluciones como cygwin, si bien a partir de Windows 10 está integrado de forma directa.

El shell funciona en un terminal. Originalmente, un terminal es una verdadera máquina que sólo dispone de lo necesario para introducir instrucciones (el teclado) y visualizar los resultados (una pantalla o incluso hace mucho tiempo una simple impresora con papel listado). Aunque en la actualidad aún existan terminales físicos, en máquinas corrientes han sido sustituidos por programas que emulan terminales. Se distinguen dos tipos de ellos en Linux:

- las consolas virtuales de texto, el modo por defecto de Linux cuando arranca o funciona bajo entorno gráfico;
- las consolas o terminales gráficos, como xterm, eterm o konsole, que son emuladores de terminales en el seno de ventanas gráficas.

El shell funciona en un terminal. Espera entradas por teclado en la consola o la ventana, y visualiza sus resultados en el mismo lugar. Cualquier usuario avanzado de Linux o Unix en

general tiene al menos un terminal abierto permanentemente o incluso más, o algunas pestañas por terminal. La apertura de un terminal (o consola, en este caso estas palabras son sinónimas) ejecuta automáticamente el shell por defecto.

b. Línea de comandos

El shell espera entradas por el teclado en una línea llamada línea de comandos o prompt. Un cursor, representado por un rectángulo fijo, intermitente o un carácter subrayado, indica la posición actual de su entrada.

La línea (prompt) proporciona información en el terminal y su posición en el sistema de archivos.

```
seb@slyserver:/home/public>
```

o

```
seb@slyserver:/home/public$
```

o incluso

```
[seb@slyserver public]$
```

En esta clásica línea, obtiene cuatro datos:

- seb: es el nombre de inicio de sesión o login del usuario actualmente conectado al terminal;
- slyserver: es el nombre de anfitrión (hostname), el nombre lógico de la máquina conectada al terminal;
- /home/public: es la posición actual del shell en el sistema de archivos. Si solo se indica un directorio, es el último del árbol donde se esté ubicado.
- > o $: es la terminación estándar del bash para un usuario sin privilegios.

Esta línea le informa de que el usuario sin privilegios de administración seb es el que

utiliza el terminal (está conectado) en la máquina slyserver y que se encuentra actualmente en /home/public.

El carácter de terminación puede tener otros significados:

- $ indica que el usuario no tiene privilegios particulares, como con >.
- # indica que el usuario es el administrador root que tiene todos los privilegios.

La ruta puede variar:

seb@slyserver:~$: el carácter tilde ~ indica que se encuentra en su directorio personal.

seb@slyserver:~/test$: al combinarse con ~ (su directorio personal), indica que está en el subdirectorio test de éste.

La línea de comando se puede personalizar modificando una variable de entorno llamada PS1.

Para cualquier otra ruta se suele sustituir la línea de comandos por un simple símbolo de dólar, $, o un #, con el fin de ganar espacio.

3. Utilizar el shell

a. La introducción de datos

En el terminal, el teclado se utiliza de forma intuitiva. Puede desplazarse en la línea con las flechas de derecha e izquierda del teclado y borrar caracteres con las teclas [Tab] y [Supr]. Ejecute el comando que ha introducido presionando la tecla [Entrar].

Hay varios métodos abreviados que son prácticos:

- **[Ctrl] a**: ir al principio de la línea.
- **[Ctrl] e**: ir al final de la línea.
- **[Ctrl] l**: borrar el contenido del terminal, y mostrar la línea de comandos en la parte superior.
- **[Ctrl] u**: borrar la línea hasta el principio.

- **[Ctrl] k**: borrar la línea hasta el final.

Ha llegado la hora de probar comandos. El comando **date** indica la fecha y la hora actuales. Por supuesto no obtendrá el mismo resultado, y no siempre en el mismo idioma; depende de su instalación de Linux.

```
$ date
mar 23 mar 2021 20:28:30 UTC
```

Un comando práctico, **pwd**, permite saber en qué directorio está.

```
$ pwd
/home/seb
```

El shell indica que está en el directorio /home/seb.

b. Sintaxis general de los comandos

Los comandos o instrucciones (las dos palabras son sinónimas en este caso) GNU tienen a menudo una sintaxis con la misma estructura:

```
Comando [parámetros] [argumentos]
```

Un comando puede no tener ni parámetros ni argumentos. En este caso, ejecuta la acción por defecto para la cual se programó, o muestra un mensaje de error, de haberlo.

Un parámetro es una opción del comando. Las dos palabras son sinónimas aquí. A menudo es una simple letra o una simple cifra precedida de un guión: `-l`, `-p`, `-s`, etc. Si el comando acepta varios parámetros, los puede introducir unos tras otros separándolos por espacio: `-l -r -t`, o escribiendo un solo guión y luego todos los parámetros: `-lrt`. Se aceptan las dos sintaxis: producen el mismo resultado. Simplemente, la segunda es más corta.

En ciertos casos, un parámetro necesita un argumento, por ejemplo un nombre de archivo. En este caso, es preferible separar este parámetro de los demás: `-lrt -f miarchivo` o de ubicarlo al final de los parámetros : `-lrtf miarchivo`.

Los argumentos son los valores en los que un comando debe ejecutar su acción. El tipo de valor (archivo, texto, números) depende del comando.

Dependiendo de las distribuciones, las versiones de las herramientas y los comandos pueden presentar diferencias y, por lo tanto, la sintaxis de los ejemplos aquí estudiados podría variar.

c. Primer ejemplo concreto con cal

Tomemos el ejemplo del comando **cal**. Admite varios parámetros y argumentos (los únicos de la norma POSIX). Si se le invoca sin argumentos, muestra el calendario del mes en curso y resalta el día actual.

```
$ cal

    Marzo 2021
do lu ma mi ju vi sá
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

El comando **cal** admite dos argumentos opcionales. Si se precisa sólo uno, se trata del año, y se muestra el calendario de ese año en su totalidad. Si se le indican dos argumentos, el primero es el mes; el segundo, el año.

```
$ cal 12 1975
   diciembre 1975
do lu ma mi ju vi sá
   1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

El comando admite también unos parámetros, según la versión instalada. Aquí se trata de CentOS 9 o openSUSE 15.2. Observará que por defecto se prevé la visualización para los anglosajones: la primera columna es un domingo, que representa el primer día de la semana. En España, es el lunes. El parámetro `-m` (de monday, en inglés) permite precisarlo:

```
$ cal -m 12 1975
   diciembre 1975
lu ma mi ju vi sá do
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

Si no ha constatado la visualización (inversa en el formato español) y requiere una visualización anglosajona, el parámetro será `-s`.

Un segundo parámetro `-3` permite visualizar los meses que preceden y siguen al mes determinado (o el mes en curso).

```
$ cal -m -3 12 1975
  noviembre 1975       diciembre 1975       enero 1976
lu ma mi ju vi sá do  lu ma mi ju vi sá do  lu ma mi ju vi sá do
                1  2   1  2  3  4  5  6  7          1  2  3  4
 3  4  5  6  7  8  9   8  9 10 11 12 13 14   5  6  7  8  9 10 11
10 11 12 13 14 15 16  15 16 17 18 19 20 21  12 13 14 15 16 17 18
```

```
17 18 19 20 21 22 23  22 23 24 25 26 27 28  19 20 21 22 23 24 25
24 25 26 27 28 29 30  29 30 31              26 27 28 29 30 31
```

Y como puede agrupar los parámetros, el comando siguiente produce el mismo resultado.

```
$ cal -m3 12 1975
```

En una distribución Ubuntu o Debian, puede usar el comando **ncal** que produce el mismo resultado pero con una sintaxis diferente, el comando **cal** solo acepta el mes y el año.

```
$ ncal -b -M -3 7 1978
    Junio 1978          Julio 1978         Agosto 1978
lu ma mi ju vi sá do lu ma mi ju vi sá do lu ma mi ju vi sá do
        1  2  3  4              1  2     1  2  3  4  5  6
 5  6  7  8  9 10 11   3  4  5  6  7  8  9   7  8  9 10 11 12 13
12 13 14 15 16 17 18  10 11 12 13 14 15 16  14 15 16 17 18 19 20
19 20 21 22 23 24 25  17 18 19 20 21 22 23  21 22 23 24 25 26 27
26 27 28 29 30        24 25 26 27 28 29 30  28 29 30 31
                      31
```

d. Encadenar los comandos

Puede ejecutar varios comandos en una sola línea, unos tras otros. Para ello, basta con separarlos con un punto y coma.

```
# date;pwd;cal -m  ene 2020

mar 23 mar 2021 21:02:15 UTC
/home/seb
    Enero 2020
do lu ma mi ju vi sá
        1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
```

```
26 27 28 29 30 31
```

e. Visualizar texto con echo

No hay nada más sencillo que visualizar texto. El comando **echo** está hecho para ello. Como casi todos los comandos, acepta los parámetros, además de los argumentos, en forma de texto. Para visualizar un texto sencillo:

```
$ echo Hola amigos
Hola amigos
```

Puede colocar el texto entre comillas o simples apóstrofes para aclarar y agrupar el texto que va a visualizar. Podrá ver más adelante que esto tiene un significado particular.

Observe que, por defecto, su texto se visualiza y echo efectúa solo un retorno de carro para empezar otra línea. Puede modificar su texto para añadirle secuencias de caracteres que tienen una acción particular. Si ya conoce el lenguaje C, son las mismas. En la lista se encuentran sólo las más utilizadas:

Secuencia	Acción
`\n`	Pasar a la línea
`\t`	Tabulación horizontal
`\c`	Suprimir el salto de línea final
`\b`	Retorno de un carácter atrás
`\\`	Visualiza la barra oblícua
`\nnn`	Visualiza el carácter especificado en octal

Para utilizar estas secuencias, añada el argumento `-e`:

```
$ echo -e "Hola.\tMe llamo Javier\b\b\bNadie\n"
Hola. Me llamo Nadie
```

f. Comandos internos y externos

Existen dos tipos de comandos:

- Los comandos externos son programas binarios presentes como archivos en su disco duro (o cualquier otro soporte de datos). Cuando ejecuta el comando, se carga este archivo en memoria y se inicia como proceso (esta noción se explicará en este mismo capítulo).
- Los comandos internos son del proprio shell y se ejecutan en él. Estos comandos forman parte del programa shell, el bash. Los comandos **cd** o **pwd** son dos ejemplos de comandos internos. Cuando los ejecuta, el shell ejecuta las funciones definidas en su interior que les corresponden.

Puede distinguir un comando interno de un comando externo con la ayuda del comando interno **type**. Así, **date** es un comando externo. Puede observar que es un archivo presente en `/bin`, mientras que **pwd** es interno al shell.

```
$ type date
date is /bin/date
$ type pwd
pwd es una orden interna del shell
```

También puede encontrar otros tipos, como los alias de comandos que son atajos de comandos propios del shell. Así el shell bash de ciertas distribuciones Linux proponen alias como ll, que corresponde en realidad a ls -l.

```
$ type ll
ll is aliased to `ls -l'
```

g. Algunos atajos útiles

Se deben conocer algunas secuencias de atajos de comandos:

- **[Ctrl] c**: interrupción del programa: se termina.
- **[Ctrl] z**: para el programa (ver los procesos).
- **[Ctrl] d**: interrumpe una introducción de datos en un símbolo del sistema >.

4. El historial de comandos

Le resultará muy útil poder volver a llamar a un comando que ya ejecutó navegando por el historial de los comandos con las teclas [Flecha arriba] y [Flecha abajo]. La flecha arriba

vuelve atrás en el historial. Si ha tecleado los dos comandos anteriores (**date** y luego **pwd**), al presionar una primera vez en la flecha arriba se visualiza en la línea de comandos **pwd**, y a la segunda se visualiza el comando **date**. La flecha abajo navega en el otro sentido hasta la línea de origen. Si pulsa la tecla [Entrar] vuelve a ejecutar el comando.

Cuantos más comandos teclee, más se amplía el historial. El shell conserva así un gran número de entradas en el historial (se puede modificar el número de líneas guardadas). Este historial se conserva en un archivo caché de su directorio personal llamado .bash_history. Puede ver el contenido del historial con el comando **history**. El resultado siguiente está truncado de manera voluntaria, ya que la lista es demasiado larga.

```
$ history
...
 1000  date
 1001  pwd
 1002  uname -a
 1003  ls
 1004  fc -l -5
 1005  history
```

El comando **fc** funciona como history cuando se utiliza con el parámetro `-l`. Por defecto, se limita a los últimos quince comandos. También puede pasar el número de últimos comandos, como a continuación para los diez últimos:

```
$ fc -l -10
109   cal -m -3 12 1975
110   date;pwd;cal
111   date;pwd;cal -m
112   echo "toto\n"
113   echo -e "toto\n"
114   type date
115   type pwd
116   type ll
117   fc -l -10
118   uname -a
```

Puede volver a llamar a un comando con fc y el parámetro `-s` seguido del número del

comando. Entonces se ejecutará de manera automática.

```
$ fc -s 1001
uname -a
Linux slyserver 2.6.22.17-0.1-default #1 SMP 2008/02/10 20:01:04 UTC
x86_64 x86_64 x86_64 GNU/Linux
```

Finalmente, puede sustituir un elemento del comando por otro antes de ejecutarlo. Por ejemplo, puede sustituir `fc` por `ls` en la entrada 1002 del historial:

```
$ fc -s fc=ls 1002
ls -l
total 775944
-rw-r--r--   1 seb users    15391 may 14  2007 AR-1179161176460.pdf
...
```