

Toma el control de la línea de comandos



SHELL PERSONAL

Consejos que nos ayudan a personalizar Bash. POR BRUCE BYFIELD

Una manera de reducir la ansiedad que genera el uso de la línea de comandos es conseguir todo el control sobre la situación que sea posible. Bash, la shell predeterminada en la mayoría de las distribuciones GNU/Linux, no es ninguna excepción. Si sabemos cómo personalizarla, comenzaremos a perder la sensación de angustia (sin duda inducida por una exposición al DOS que usábamos a comienzos de nuestra vida informática) que nos paraliza cuando se plantea la idea de usar la línea de comandos.

Evidentemente, muchas personalizaciones nos interesarán sólo si somos desarrolladores. Francamente, listar cada opción posible requeriría un artículo cinco o seis veces la longitud de éste. Aún así, los ejemplos que se muestran más adelante pueden interesar a usuarios de cualquier nivel y dan una idea de las posibilidades. Varían desde crear nombres cortos para comandos y cambiar permisos por defecto, a personalizar la apariencia y el sentido del prompt del comando y el comportamiento del historial de Bash.

Ficheros Implicados

Antes de comenzar hemos de saber que todas las cuentas de usuario poseen potencialmente cuatro ficheros asociados con Bash. Todos se encuentran normalmente ocultos, aunque podemos ver los que usa nuestra distribución escribiendo `ls -a .bash*`.

Dos ficheros Bash son de interés limitado si estamos aprendiendo cómo personalizarla. El fichero `.bash_history` es una lista de comandos introducidos previamente,

uno por línea. Aunque podemos editarla en un editor de texto, la mayoría de la gente utiliza las teclas Arriba y Abajo para desplazarse a través del historial para encontrar el comando que quieren volver a usar. El fichero opcional `.bash_logout` nos permite ejecutar un script cuando salimos de Bash, aunque no se utiliza a menudo.

Los otros dos ficheros Bash son fundamentales para la configuración. El primero de ellos, `.bashrc`, contiene configuraciones básicas para el historial y opciones del prompt, y está siempre presente. Si se borra, vuelve a crearse automáticamente por el fichero `/etc/bash.bashrc`. El segundo es `.bash_profile`, el cual incluye opciones y configuraciones adicionales. Si no está presente, la cuenta del usuario usa `/etc/profile` en su lugar, que es el predeterminado para el sistema completo.

Estos ficheros pueden editarse directamente o a través de un comando como `export`.

Cambiando la Ruta

La ruta es una lista de directorios que mira el sistema operativo en busca de comandos que introducimos. Consta de todas las rutas definidas en `/etc/profile` más cualquiera añadida a `.bash_profile` en nuestro directorio de inicio. Si un comando no estuviera en la ruta, tendremos que introducir una ruta completa cuando escribimos el comando o bien cambiar al directorio en la que se encuentra – ninguna de las dos resultan tan convenientes como introducir el comando y confiar en Bash para saber dónde buscarlo.

Para añadir un directorio a la ruta, podemos abrir nuestro `.bash_profile` en un editor de texto y buscar el comando `$PATH`. Si no se menciona en ninguna parte del fichero, podemos introducir las líneas manualmente al final del mismo. Por ejemplo, Si deseamos añadir un directorio `/bin` para ejecutables a nuestro directorio de inicio, introduciríamos

```
PATH=$PATH:/home/mi_cuenta/bin
export PATH
```

Para que los cambios tengan efecto inmediato, hemos de ejecutar los comandos contenidos en el fichero con la instrucción `source`:

```
$ source /home/mi_cuenta/
.bash_profile
```

Si no, tendremos que esperar hasta la próxima vez que iniciemos sesión para apreciar los cambios.

Alternativamente, podríamos modificar la ruta desde la línea de comandos declarándolo primero y configurándolo después con:

```
$ PATH=$PATH:/usr/local/bin:
/usr/bin:/bin:/usr/games:
/home/mi_cuenta/bin
$ export PATH
```

Si decidimos que no necesitamos un directorio en nuestra ruta, podemos redefinirlo con las dos mismas líneas, omitiendo simplemente el directorio innecesario.

Nombres Cortos o Alias

Los comandos pueden llegar a ser largos y complejos. Dependiendo del comando, podemos tener el comando básico, cualquier número de variables, un fichero o directorio fuente y un objetivo. En algunos casos, como con el comando `apt-get`, también podemos tener un subcomando además del comando principal. Esta estructura puede ser difícil de recordar, y, para empeorar las cosas, un error en la sintaxis puede tener resultados no esperados o invalidar el comando.

De modo que Bash nos permite definir y usar atajos. Podremos crearlos editando `.bashrc` en un editor de texto, aunque es más rápido usar los comandos precompilados `alias` y `unalias`. Como indican sus nombres, `alias` crea atajos y `unalias` los borra.

Las estructuras de estos comandos son simples. Por ejemplo, si queremos ver siempre las listas de los contenidos del directorio codificadas en color, deberemos introducir el comando *alias ls = 'ls —color= auto'*.

Técnicamente, deberíamos comenzar con el *alias -p*, aunque la opción *-p*, que envía los resultados a una salida estándar, es innecesaria en todas las distribuciones que he probado, así que no necesitamos preocuparnos por ella.

Una vez que hemos definido este alias, en lugar de introducir siempre *ls —color = auto*, lo único que tenemos que hacer es escribir *ls*. Esto nos ahorra pulsaciones de tecla si usamos la línea de comandos para la administración de ficheros. Podemos hacer lo mismo para cualquier comando Bash o aplicación, incluyendo uno para el escritorio, si así lo decidimos. La limitación obvia es que hay que seleccionar atajos que sean poco probables de introducirse por accidente, aunque supongo que también puede que descubramos – al menos en teoría – algún alias que crea un conflicto entre opciones incompatibles.

Borrar un atajo es incluso más simple: Escribimos *unalias* seguido del nombre del atajo. Por ejemplo, si decidimos que utilizar colores en un listado de directorio no es algo que prefiramos después de todo (probablemente porque seamos daltónicos, o porque prefiramos la opción *-F* para indicar los tipos de ficheros por un carácter al final del nombre), entonces introducimos *unalias ls*.

Este comando borrará el alias, pero no – y permítanme que subraye esto – el comando *ls* en sí. Si deseamos borrar todos los alias, el comando es más simple aún: *unalias -a*.

Para ver una lista de atajos definidos, escribimos *alias* sin más. Si usamos muchos, deben añadirse líneas de comentarios (#) en *.bashrc* y organizar nuestros atajos por función para verlos luego en un editor de texto cuando necesitemos un recordatorio. En un fichero de muestra *.bashrc* que he visto recientemente, se habían separado los alias por categorías: programación, aplicaciones de escritorio, scripts y otra media docena de clases. Otros incluían errores de tecleo comunes, para que el usuario no recibiera un error al escri-

\d	fecha
\h	nombre de la máquina
\t	hora actual en formato 24 horas HH:MM:SS
\T	hora actual en formato 12 horas HH:MM:SS
\@	hora actual en formato 12 horas am/pm
\A	hora actual en formato 24 horas HH:MM
\u	nombre de usuario del usuario actual
\w	directorio de trabajo actual, con el directorio de nivel superior indicado con una tilde (~)
\W	nombre base del directorio de trabajo actual, con el directorio de nivel superior indicado con una tilde (~)
*\	barra invertida

bir *yum intsall* en vez de *yum install* cuando añade paquetes a su sistema Fedora. Tal y como muestran estos ejemplos, cómo de útil encontremos los alias dependerá completamente de nuestra paciencia e ingenio.

Configurando Permisos Predeterminados

Los permisos definen cómo pueden usarse los ficheros o directorios. Cuando creamos un nuevo fichero, se está dando automáticamente un juego de permisos predeterminado denominado *umask*.

Una manera de resumir los permisos es utilizando tres dígitos. De izquierda a derecha, son los permisos de la cuenta de usuario desde la que se creó, para los que se encuentran en el mismo grupo que el creador y para todos los demás registrados. Aparentemente, sin más razón que la de que la idea pareció buena en su momento, cada dígito es la suma de números en base 8 que definen un permiso. El permiso de lectura es 4, el de escritura 2, el de ejecución 1, y ningún permiso es 0. A esta forma corta se le denomina modo absoluto, permisos simbólicos o permisos octales, según prefiramos.

Bajo este sistema, un fichero que todo el mundo pueda leer, escribir o ejecutar debería tener un permiso de 777. Por el contrario, uno que el propietario podría leer, escribir y ejecutar y que nadie más podría usar, tendría un permiso de 700.

Cuando creamos un fichero y el sistema asigna el juego predeterminado de permisos, el *umask* está definido en */etc/login.defs*. Sin embargo, podemos establecer el *umask*

seguido de los permisos (por ejemplo, *umask 700*) a *.bashrc*.

Personalizando el Prompt

El prompt es el texto recurrente que aparece a la izquierda de donde comenzamos a escribir el comando. Si somos un usuario nuevo, posiblemente no le hayamos prestado demasiada atención.

Tampoco es absolutamente necesario en la actualidad. Antes de que los escritorios se convirtieran en la norma, el prompt contenía información útil que necesitábamos saber de un vistazo, como el directorio actual, la fecha y la hora. Hoy en día, que podemos conseguir toda esta información del escritorio, no deberíamos preocuparnos demasiado por él.

Aún así, incluso hoy en día, encontraremos útil disponer de información básica visible siempre, como la cuenta actual y el directorio, incluso a pesar de que estemos usando casi siempre un terminal virtual. Nunca sabemos cuándo necesitaremos reparar nuestro sistema y no tengamos disponible el escritorio. Además, si miramos algunas distribuciones, encontraremos a menudo que el carácter final en el prompt nos dice si estamos usando un usuario normal o una cuenta de root; los prompts de la cuenta del usuario acaban en un signo dólar, mientras que los de root lo hacen en un signo almohadilla (#).

De hecho, una vez que lo miramos, veremos que distribuciones diferentes poseen prompts diferentes basándose en qué información piensan que querrán ver los usuarios. Por ejemplo, si estuviera trabajando en el directorio */usr/share* en un ordenador llamado *nanday*, en Fedora, mi prompt predeterminado será *[bruce@conure share]\$*, mientras que en Debian será *bruce@nanday:/usr/share\$*. De esta diferencia deberíamos inferir que los valores predetermina-

cer el *umask* para cada cuenta añadiendo el comando

```
bruce@nanday:~$ ls -a .bash*
.bash_history  .bash_logout  .bash_profile .bash_profile~ .bashrc
```

Figura 1: El comportamiento de Bash está configurado en cuatro ficheros principales en nuestro directorio de inicio.

```
bruce@nanday:~$ PS1="\u $ "
bruce $ █
```

Figura 2: Un cambio en el prompt de Bash dura solamente hasta que se cierra la sesión actual. Cualquier cambio en el prompt requiere referencias a una serie de símbolos arcanos.

dos de Fedora suponen que los usuarios permanecen gran parte del tiempo en sus directorios de inicio porque no se les da la ruta completa, mientras que los de Debian suponen que sus usuarios más avanzados pueden estar en cualquier parte en el sistema y prefieren no tener que usar el comando *pwd* para descubrir dónde se encuentran.

Además, posiblemente deseemos acortar el prompt, especialmente si estamos listando rutas a directorios anidados a cinco o seis niveles de profundidad. También puede que queramos cambiar el color del prompt para hacerlo más visible, para complacer nuestro sentido de la estética, o para resaltar mejor la diferencia entre la cuenta de root y todas las demás.

Sea cual sea nuestra razón, el mejor sitio para comenzar si queremos cambiar nuestro prompt es cambiar temporalmente el parámetro prompt *PS1*. Estos cambios permanecerán efectivos hasta que los cambiemos de nuevo o cerremos nuestra terminal virtual actual. El terminal siguiente que abramos nos volverá al prompt predeterminado.

Para comenzar, desearemos ver las configuraciones actuales para *PS1*, la variable para el prompt, introduciendo el comando *echo PS1*. Probablemente obtendremos una respuesta que dice algo como: `[\u@\h \W]$`, que es la lectura que consigo con una máquina con Fedora 10 instalado. Comparándolo con el prompt de Fedora mencionado antes, podemos entender lo que significa cada entrada. Nótese, sin embargo, los elementos extra, tales como la *@* entre el nombre de usuario (*\u*) y el nombre del host (*\h*) y el espacio entre el nombre de host y el directorio actual (*\W*).

Podemos cambiar este prompt temporalmente tomando como referencia la Tabla 1. No me he molestado con algunas opciones que probablemente carezcan de interés para los modernos no-desarrolladores. Si deseamos consultarlas, podremos encontrar una lista completa online con una búsqueda rápida.

Por ejemplo, si quisiéramos presentar la cuenta de usuario actual, haríamos un cam-

```
bruce@nanday:~$ export PS1="\u $ "PS1="\e[0;31m[\u@\h \W]\$ \e[m "
bruce@nanday ~]$
```

Figura 3: El prompt de Bash se cambia permanentemente, toma los efectos inmediatamente y los mantiene la próxima vez que iniciamos la shell.

bio temporalmente ejecutando *PS1 = "\u \$ "* para obtener el prompt *bruce \$*. Nótese que el espacio después del signo dólar no es un error, sino que se añade deliberadamente para que lo que escribo esté separado del prompt.

Si deseamos añadir color, podemos cambiar el de los caracteres de acuerdo a la fórmula:

```
"\e[x;ym\e[m "
```

En ella, *\e/* marca el inicio de los caracteres a los que se les aplica el color, *\e/m* señala el final y *x;ym* es el código de color (Tabla 2). El comando para cambiar el prompt a rojo oscuro sería:

```
PS1="\e[0;31m[\u@\h \W]\$ \e[m "
```

Si sustituimos un *1* por un *0*, obtendremos una versión más suave del mismo color.

Podemos experimentar con estos cambios temporalmente hasta que consigamos el prompt que deseemos. Si cometemos un error, podemos usar las teclas de flecha para encontrar una entrada en el historial de comandos desde la que podemos cambiar el error o bien simplemente cerrar la ventana si estamos trabajando en un terminal virtual.

Cuando encontremos la fórmula del prompt que queremos, abrimos el fichero *.bashrc* para nuestra cuenta en un editor de texto y sustituimos la fórmula existente por la que hemos ideado. Si preferimos, podemos usar también el comando *export*: por ejemplo, `export PS1 = "\e[0;31m[\u@\h \W]\$ \e[m`.

Opciones del Historial

El historial Bash es una lista de comandos usados previamente almacenados en

Negro	0;30
Azul	0;34
Verde	0;32
Turquesa	0;36
Rojo	0;31
Morado	0;35
Marrón	0;33

.bash_history. Si escribimos continuamente el mismo comando u otros similares, podemos usar las teclas de flecha Arriba y Abajo para desplazarnos hasta el comando que queremos usar.

El comando *export* configura las variables de la shell y es una manera de personalizar el historial Bash. O, editar directamente *.bashrc*. Para establecer el número de comandos almacenados en el historial a 1,200, bien lo introducimos en la línea de comandos o bien lo añadimos a *.bashrc*:

```
export HISTFILESIZE=1200
```

Como *.bash_history* es un fichero de texto, no requiere demasiada memoria, así que no hay razón alguna por la que no usemos esta configuración para incrementar el tamaño por defecto de 500 entradas. Evidentemente, cuanto mayor sea el fichero, más desplazamientos tendremos que hacer para encontrar un viejo comando, aunque siempre podemos abrir *bash_history* en un editor de texto y usar su función de búsqueda.

Otra personalización es mantener comandos duplicados que están añadidos a *.bash_history*. Para esta opción, introducimos:

```
export HISTCONTROL=erasedups
```

Esta personalización asegura que *.bash_history* contiene solamente comandos únicos, aunque podría significar que el comando que deseamos requiere desplazamiento extra, especialmente si hemos usado uno que ya aparece muy atrás en la lista.

Sólo el Principio

Para llevar todo esto más lejos, necesitaremos saber cómo escribir scripts en Bash. Los scripts amplían las posibilidades radicalmente. En Debian, por ejemplo, un script se usa en *.bash_logout* para limpiar la pantalla para asegurar nuestra privacidad cuando salimos de la línea de comandos. Los ejemplos descritos en este artículo son lo suficientemente simples como para ayudarnos a comenzar y darnos una perspectiva de las posibilidades de personalización de Bash. ■