El comando `kill` en Linux

Escrito por: *Yoel Monsalve*

Esta es la versión en español del artículo intitulado "The kill command", del mismo autor.

Una guía comprensible del comando `kill`, para desmitificar cosas extrañas que se dicen en el mundo de la programación.

> Hace algún tiempo, escuché alguien decir que "el comando `kill` es para matar un proceso". Bueno, eso no es técnicamente correcto. Estrictamente hablando, el comando `kill` es para *enviar una señal a un proceso*. Esto es un error frecuente que he escuchado de programadores novatos, o la gente que sólo toca los temas superficialmente.

Pero, vamos a una fuente autoritaria, desde la ayuda/documentación de Linux:

```
$ kill --help

kill: kill [-s sigspec | -n signum | -sigspec] pid | jobspec ... or kill -l [sigspec]

    Send a signal to a job.

    Send the processes identified by PID or JOBSPEC the signal named by
    SIGSPEC or SIGNUM.  If neither SIGSPEC nor SIGNUM is present, then
    SIGTERM is assumed.

    Options:
      -s sig    SIG is a signal name
      -n sig    SIG is a signal number
      -l    list the signal names; if arguments follow `-l' they are
            assumed to be signal numbers for which names should be listed
      -L    synonym for -l

    Kill is a shell builtin for two reasons: it allows job IDs to be used
    instead of process IDs, and allows processes to be killed if the limit
    on processes that you can create is reached.

    Exit Status:
    Returns success unless an invalid option is given or an error occurs.
```

Así que la respuesta es muy clara: 'Terminar/matar' un proceso es solo una de las muchas cosas que se pueden hacer a través de este comando. Las "señales" (`signals`) son un tema fascinante, y el lector interesado puede consultar este manpage, o este artículo de *computerhope* , entre otros.

Señalar (`signaling`), es un mecanismo típico para comunicación entre procesos. Se envía una señal a un proceso, esperando que dicho proceso realice una acción específica. Estas acciones pueden ser, por ejemplo, pausar/retomar el proceso (`SIGSTOP` , `SIGCONT`), o terminar el proceso bien sea de forma amigable o no-amigable (`SIGTERM` , `SIGKILL`). Otros comportamientos en respuesta a una señal son menos conocidos, por ejemplo que un proceso padre sea consciente de la terminación de uno de sus hijos (`SIGCHILD`), o para pedir a un proceso que ejecute una acción predeterminada solicitada por el usuario (`SIGUSR1`). Usted puede ver la lista completa del comportamiento estándar ante las señales más comunes, en el artículo de *computerhope* .

Vale la pena señalar que las *señales* son un mecanismo crucial en cualquier sistema bien diseñado, para trabajar con una arquitectura de multi-proceso.

Para ver una lista completa de todas las señales disponibles en su sistema, tipee

```
$ kill -l

 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

La sintaxis para enviar una señal determinada a un proceso, digamos la señal para pedir una terminación amigable `SIGTERM` , es:

```
$ kill -s SIGTERM <pid>
```

donde `<pid>` es el ID de proceso, un entero único que sirve como identificador de dicho proceso. Cada proceso tiene un ID que es automáticamente asignado por el sistema operativo (y no se puede/debe cambiar). Hay algunos comandos que pueden ser útiles para averiguar el PID de un programa, como `pgrep` o `ps` . Sin embargo, esto sería suficiente material para un artículo completo, y no será cubierto aquí.

Otras formas para la sintaxis del comando `kill` son:

```
$ kill -s TERM <pid>
```

y (un especie de manera abreviada de lo anterior):

```
$ kill -TERM <pid>
```

o, usando el *numero* de señal en lugar del nemónico para la misma (**NO recomendado**, ya que los numeros de señal podrían hipotéticamente variar en un futuro, o variar entre un sistema u otro):

```
$ kill -n 15 <pid>
```

o

```
$ kill 15 <pid>
```

Un primer experimento

Supóngase que se tiene un proceso *perezoso* ("lazy") el cual no hace más que "dormir" por un tiempo (cuando se dice "dormir", se está refiriendo a la acción *sleep* del inglés que significa que dicho proceso se coloca en un estado de hibernación o pausado, mientras otro proceso pasa al estado de actividad en su lugar). Ejemplo:

```
"""FILE: lazy.py
---
from time import sleep
import os

print('I\'m a very lazy process -.). Please, don\'t disturb ...')
print(f'My PID = {os.getpid()}')
sleep(10)
print("Lazy program says ... Bye !\n")
```

Lo interesante es que una vez nuestro proceso perezoso está durmiendo y secuestra la consola, nosotros podemos simplemente pausarlo (pero no terminarlo) mediante la combinación de teclas `CTRL + Z` (en sistemas UNIX). Una vez hecho esto, el proceso se detiene:

```
$ python lazy.py
I'm a very lazy process -.). Please, don't disturb ...
My PID = 76750
^Z
[1]+  Stopped                 python lazy.py
```

Observe que el programa está imprimiendo su propio PID, de modo que facilitar al usuario conocer este ID, y enviarle alguna señal si es necesario.

Ahora, una vez que tenemos de nuevo tiempo para dedicarle a nuestro proceso perezoso, vamos a reactivarlo enviándole la señal `SIGCONT` (continue):

```
$ kill -CONT 76750
$ Lazy program says ... Bye !
```

Luego de retomar el proceso, este termina su ciclo de espera y luego de ello imprime un mensaje de despedida y termina.

Este proceso de detención de procesos puede ser usado, por ejemplo, para tomar 'capturas' a programas que dirigen su salida a la consola (y de este modo observar el flujo de actividades del proceso, imprimir valores de variables, etc). Es realmente un mecanismo muy útil, aunque rudimentario, de hacer *debug* de programas.

Empezando a usar *jobs* (tareas)

Trabajar en terminales de sistema UNIX puede volverse productivo cuando empezamos a usar los *jobs*, o tareas. Un *job* es un proceso corriendo bajo una terminal. Posiblemente usted notó el mensaje particular

```
^Z
[1]+  Stopped                 python lazy.py
```

cuando detuvimos el programa perezoso. Esto significa que la terminal a cargo está identificando dicho proceso con el *job number* (número de tarea) igual a 1 (esto podría ser un número mayor si usted tiene varias tareas corriendo en la misma terminal).

De acuerdo a la ayuda del comando `kill`, es también posible usar la sintaxis:

```
kill [-s sigspec | -n signum | -sigspec] jobspec
```

donde `jobspec` es la identificación para la tarea objetivo (que se debe pasar incluyendo el carácter '%' delante del número). Es decir:

```
$ python lazy.py
I'm a very lazy process -.). Please, don't disturb ...
My PID = 77330
^Z                              <-- you press CTRL + Z
[1]+  Stopped                 python lazy.py

$ kill -CONT %1                 <-- this will resume process
$ Lazy program says ... Bye !   <-- process is done
```

La forma fácil.

Si usted solo está interesado en detener/continuar un proceso, puede profundizar un poco más en el concepto de los *jobs*. Presionar `CTRL + Z` pausa la tarea y la pone en segundo plano o *background*. Podemos luego reactivar el proceso pausar, trayéndolo de vuelta al primer plano o *foreground*.

ESTADO	SIGNIFICADO
background	el proceso pasa a estado pausado
foreground	el proceso está despierto y activo para interactuar con el teclado

Podemos mover un proceso al segundo plano (`background`), y luego al primero (`foreground`) usando, respectivamente, los comandos `fg` y `bg` . Ejemplo

```
fg %1    .... pone el job ID#1 en estado background (stop)
bg %1    .... pone el job ID#1 en estado foreground (resume)
```

Usted puede alternar un proceso entre el segundo plano y el primer plano tantas veces como quiera, mientras el proceso esté aún vivo y siendo manejado por dicha terminal.

Presionar `CTRL + Z` envía implícitamente una señal `STOP` a un proceso, de modo que este es de hecho puesto en estado background. Para restaurarlo, use `fg` . Puede moverlo de nuevo al background con `bg` , luego de nuevo al foreground con `fg` , ... etc.

```
$ python lazy.py
I'm a very lazy process -.). Please, don't disturb ...
My PID = 77771
^Z                              <-- you press CTRL + Z
[1]+  Stopped                 python lazy.py
$ fg %1                         <-- this will resume child, until its completion
python lazy.py
Lazy program says ... Bye !
```

Note que la orden `fg` envía implícitamente una señal `CONT` al proceso hijo , de modo que usar `fg` / `bg` es prácticamente equivalente al uso "manual" de comandos `kill -STOP|-CONT` .

Explorando otras opciones

Quizá usted decidió que ya tuvo suficiente de este proceso perezoso, y quiere simplemente terminarlo. Entonces, usted debe despachar la señal `TERM` para terminar de una vez con esta molestia:

```
$ python lazy.py
I'm a very lazy process -.). Please, don't disturb ...
My PID = 77907
^Z                              <-- you press CTRL + Z
[1]+  Stopped                 python lazy.py
$ kill -TERM %1                 <-- kill the annoying bug

[1]+  Stopped                 python lazy.py
$                               <-- press ENTER again, and you will see
[1]+  Terminated              python lazy.py
                                (the annoying thing is gone!)
```

OK, esto puede lucir un poco extraño. Una vez que se ha enviado la señal de terminación, la consola responde diciendo que el proceso está en estado de pausa. Presione ENTER de nuevo y la terminal informará que el hijo ha sido terminado. Este es sólo uno de esos "comportamientos `UNIX` " (y que puede ser modificando al configurar algunas variables shell, pero no profundizaremos en ello ahora.)

Una nota final: La señal `TERM` envía una solicitud de terminación amigable al proceso objetivo. Este se supone que ejecuta algunos procedimientos "on-close" antes de su terminación, como cerrar descriptores de fichero abiertos, propagar la señal a sus procesos hijos, y otros. Existe también la otra señal más ruda llamada `KILL` que no puede ser ignorada por el proceso, y su efecto es terminar el proceso objetivo inmediatamente, e incondicionalmente. Pero, sea consciente y emplee cosas como esta:

```
$ kill -KILL <pid>|<jobspec>
```

solo como último recurso. Lo ideal es primero enviar la señal amigable `TERM` , luego después de algunos segundos intentar `TERM` nuevamente, y después de algunos intentos más infructuosos, entonces enviar la señal ruda `KILL` .

Y, esto es todo por ahora. Deseando que haya disfrutado este material y esperando compartir más artículos pronto !