LINUXTOTAL.COM.MX - Información y servicios en Linux y Open Source
URL: http://www.linuxtotal.com.mx/index.php?cont=info_admon_012

Manual básico de administración de procesos

Autor: Sergio González D. **(sergio.gonzalez.duran@gmail.com)**

La más simple definición de un proceso podría ser que es una instancia de un programa en ejecución (corriendo). A los procesos frecuentemente se les refiere como tareas. El contexto de un programa que esta en ejecución es lo que se llama un proceso. Este contexto puede ser mas procesos hijos que se hayan generado del principal (proceso padre), los recursos del sistema que este consumiendo, sus atributos de seguridad (tales como su propietario y permisos de archivos asi como roles y demás de SELinux), etc.

Linux, como se sabe, es un sistema operativo multitarea y multiusuario. Esto quiere decir que múltiples procesos pueden operar simultáneamente sin interferirse unos con los otros. Cada proceso tiene la "ilusión" que es el único proceso en el sistema y que tiene acceso exclusivo a todos los servicios del sistema operativo.

Programas y procesos son entidades distintas. En un sistema operativo multitarea, múltiples instancias de un programa pueden ejecutarse sumultáneamente. Cada instancia es un proceso separado. Por ejemplo, si cinco usuarios desde equipos diferentes, ejecutan el mismo programa al mismo tiempo, habría cinco instancias del mismo programa, es decir, cinco procesos distintos.

Cada proceso que se inicia es referenciado con un número de identificación único conocido como Process ID PID, que es siempre un entero positivo. Prácticamente todo lo que se está ejecutando en el sistema en cualquier momento es un proceso, incluyendo el shell, el ambiente gráfico que puede tener múltiples procesos, etc. La excepción a lo anterior es el kernel en si, el cual es un conjunto de rutinas que residen en memoria y a los cuales los procesos a través de llamadas al sistema pueden tener acceso.

ps

El comando **ps** es el que permite informar sobre el estado de los procesos. **ps** esta basado en el sistema de archivos */proc*, es decir, lee directamente la información de los archivos que se encuentran en este directorio. Tiene una gran cantidad de opciones, incluso estas opciones varían dependiendo del estilo en que se use el comando. Estas variaciones sobre el uso de **ps** son las siguientes:

- Estilo UNIX, donde las opciones van precedidas por un guión -
-
- Estilo BSD, donde las opciones no llevan guión
- Estilo GNU, donde se utilizan nombres de opciones largas y van precedidas por doble guión --

Sea cual sea el estilo utilizado, dependiendo de las opciones indicadas, varias columnas se mostrarán en el listado de procesos que resulte, estas columnas pueden ser entre muchas otras, las siguientes (y principales):

Columna	Descripción
p o PID	Process ID, número único o de identificación del proceso.
P o PPID	Parent Process ID, padre del proceso
U o UID	User ID, usuario propietario del proceso
t o TT o TTY	Terminal asociada al proceso, si no hay terminal aparece entonces un '?'
T o TIME	Tiempo de uso de cpu acumulado por el proceso
c o CMD	El nombre del programa o camndo que inició el proceso
RSS	Resident Sise, tamaño de la parte residente en memoria en kilobytes
SZ o SIZE	Tamaño virtual de la imagen del proceso
NI	Nice, valor nice (prioridad) del proceso, un número positivo significa menos tiempo de procesador y negativo más tiempo (-19 a 19)
C o PCPU	Porcentaje de cpu utilizado por el proceso
STIME	Starting Time, hora de inicio del proceso
S o STAT	Status del proceso, estos pueden ser los siguientes • R runnable, en ejecución, corriendo o ejecutándose • S sleeping, proceso en ejecución pero sin actividad por el momento, o esperando por algún evento para continuar • T sTopped, proceso detenido totalmente, pero puede ser reiniciado • Z zombie, difunto, proceso que por alguna razón no terminó de manera correcta, no debe haber procesos zombies • D uninterruptible sleep, son procesos generalmente asociados a acciones de IO del sistema • X dead, muerto, proceso terminado pero que sigue apareciendo, igual que los Z no deberían verse nunca

Las opciones completas de **`ps`** las encuentras en las páginas del manual (**`man ps`**), o escribiendo en la terminal **`ps L`**, y para ver un resumen de sus opciones más comunes usa **`ps --help`**:

```
#> ps --help
********* simple selection *********  ********* selection by list *********
-A all processes                      -C by command name
-N negate selection                   -G by real group ID (supports names)
-a all w/ tty except session leaders  -U by real user ID (supports names)
-d all except session leaders         -g by session OR by effective group name
-e all processes                      -p by process ID
T  all processes on this terminal     -s processes in the sessions given
a  all w/ tty, including other users  -t by tty
g  OBSOLETE -- DO NOT USE             -u by effective user ID (supports names)
r  only running processes             U  processes for specified users
x  processes w/o controlling ttys     t  by tty
*********** output format **********  *********** long options ***********
-o,o user-defined  -f full            --Group --User --pid --cols --ppid
-j,j job control   s  signal          --group --user --sid --rows --info
-O,O preloaded -o  v  virtual memory  --cumulative --format --deselect
-l,l long          u  user-oriented   --sort --tty --forest --version
-F   extra full    X  registers       --heading --no-heading --context
                    ********* misc options *********
-V,V  show version      L  list format codes  f  ASCII art forest
-m,m,-L,-T,H  threads   S  children in sum    -y change -l format
-M,Z  security data     c  true command name  -c scheduling class
-w,w  wide output       n  numeric WCHAN,UID  -H process hierarchy
```

A continuación algunos cuantos ejemplos de **ps** con la salida recortada.

```
>#  ps -e     (-e muestra todos los procesos)
  PID TTY          TIME CMD
    1 ?        00:00:01 init
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00 migration/0
    4 ?        00:00:00 ksoftirqd/0

#> ps -ef     (-f muestra opciones completas)
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 10:12 ?        00:00:01 init [5]
root         2     0  0 10:12 ?        00:00:00 [kthreadd]
...
root      6130  5662  0 10:24 pts/0    00:00:00 su -
root      6134  6130  0 10:24 pts/0    00:00:00 -bash
sergon    6343  5604  0 10:28 ?        00:00:00 kio_file [kdeinit] file /home/sergon/tmp/
root      6475  6134  0 10:38 pts/0    00:00:00 ps -ef

#> ps -eF     (-F muestra opciones completas extra)
UID        PID  PPID  C    SZ   RSS PSR STIME TTY          TIME CMD
root         1     0  0   412   556   1 16:59 ?        00:00:01 init [5]
root         2     0  0     0     0   1 16:59 ?        00:00:00 [kthreadd]
sergon    8326  8321  0   902  1272   0 17:07 ?        00:00:00 /bin/sh /usr/lib/firefox-
sergon    8331  8326  4 53856 62604   0 17:07 ?        00:00:50 /usr/lib/firefox-2.0.0.8/
sergon    8570  7726  2 15211 37948   0 17:17 ?        00:00:10 quanta

#> ps ax      (formato BSD sin guión, a muestra todos, x sin mostrar tty)
  PID TTY      STAT   TIME COMMAND
    1 ?        Ss     0:01 init [5]
    2 ?        S<     0:00 [kthreadd]
    3 ?        S<     0:00 [migration/0]
    4 ?        S<     0:00 [ksoftirqd/0]

#> ps aux    (formato BSD sin guión, u muestra usuarios y demás columnas)
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   1648   556 ?        Ss   16:59   0:01 init [5]
root         2  0.0  0.0      0     0 ?        S<   16:59   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S<   16:59   0:00 [migration/0]
root         4  0.0  0.0      0     0 ?        S<   16:59   0:00 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S<   16:59   0:00 [migration/1]

#> ps -eo user,pid,tty    (-o output personalizado, se indican los campos separados por coma, ver ps --h
USER       PID TT
root         1 ?
root         2 ?
sergon    8570 tty 1
root      8876 pts/1

#> ps -eH  (muestra árbol de procesos)
#> ps axf  (lo mismo en formato BSD)
#> ps -ec  (el comando  que se esta ejecutando, sin la ruta, solo el nombre real)
#> ps -el  (muestra formato largo de varias columnas, muy práctico)
```

```
#> ps L     (No muestra procesos, lista todos los códigos de formatos)
```

pstree

Muestra los procesos en forma de árbol, **`pstree --help`** te da las opciones más comunes. Recomiendo uses lo uses con la opción *-A* y *-G* para que te un árbol con líneas con líneas estilo ASCII y de terminal VT100 respectivamente, puedes añadir también *-u* para mostrar entre paréntesis al usuario propietario del proceso:

```
#> pstree -AGu
init---acpid
     |-atd(daemon)
     |-automount----2*[{automount}]
     |-avahi-daemon(avahi)
     |-beagled(sergon)----7*[{beagled}]
     |-beagled-helper(sergio)----3*[{beagled-helper}]
     |-compiz(sergon)----kde-window-deco
     |-console-kit-dae----61*[{console-kit-dae}]
     |-crond
     |-dbus-daemon(messagebus)
     |-dbus-daemon(sergio)
     |-dbus-launch(sergio)
     |-dcopserver(sergio)
     |-dhclient
     |-gam_server(sergio)
     |-gconfd-2(sergio)
     |-hald(haldaemon)----hald-runner(root)----hald-addon-acpi(haldaemon)
     |                                        |-hald-addon-cpuf
     |                                        |-hald-addon-inpu
     |                                        |-hald-addon-stor
     |-httpd---8*[httpd(apache)]
     |-2*[ifplugd]
     |-ipw3945d
     |-kaccess(sergio)
...
```

kill

El comando **`kill`**, que literalmente quiere decir matar, sirve no solo para matar o terminar procesos sino principalmente para enviar señales (signals) a los procesos. La señal por default (cuando no se indica ninguna es terminar o matar el proceso), y la sintaxis es **`kill PID`**, siendo PID el número de ID del proceso. Asi por ejemplo, es posible enviar una señal de STOP al proceso y se detendrá su ejecución, después cuando se quiera mandar una señal de CONTinuar y el proceso continuara desde donde se quedo.

```
#> kill -l     (lista todas las posibles señales que pueden enviarse a un proceso)
 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 lista previa presenta una lista de todas las posibles señales que pueden mandarse a un proceso y estas pueden ser invocadas a través del número de la señal o de su código, por ejemplo:

```
#> kill -9 11428         (termina, mata un proceso completamente)
#> kill -SIGKILL 11428   (Lo mismo que lo anterior)
```

Las señales más comunes son la 19 y 20 que detienen momentáneamente la ejecución de un proceso o programa, 18 la continua, 1 que es la señal de hang up que obliga al proceso a releer sus archivos de configuración estando en ejecución y 9 que termina rotundamente un proceso.

killall

El comando **killall**, que funciona de manera similar a **kill**, pero con la diferencia de en vez de indicar un PID se indica el nombre del programa, lo que afectará a todos los procesos que tengan ese nombre. Asi por ejemplo si se tienen varias instancias ejecutándose del proxy server squid, con **killall squid** eliminará todos los procesos que se esten ejecutando con el nombre 'squid'

```
#> killall -l              (lista de posibles señales)
#> killall -HUP httpd      (manda una señal de "colgar", detenerse releer sus archivos de configuración y
#> killall -KILL -i squid (manda señal de matar a todos los procesos squid pero pide confirmación en c
```

nice

Permite cambiar la prioridad de un proceso. Por defecto, todos los procesos tienen una prioridad igual ante el CPU que es de 0. Con **nice** es posible iniciar un programa (proceso) con la prioridad modificada, más alta o más baja según se requiera. Las prioridades van de -20 (la más alta) a 19 la más baja. Solo root o el superusuario puede establecer prioridades negativas que son más altas. Con la opción -l de **ps** es posible observar la columna NI que muestra este valor.

```
#> nice              (sin argumentos, devuelve la prioridad por defecto )
0
#> nice -n -5 comando   (inicia comando con una prioridad de -5, lo que le da más tiempo de cpu)
```

renice

Asi como **nice** establece la prioridad de un proceso cuando se incia su ejecución, **renice** permite alterarla en tiempo real, sin necesidad de detener el proceso.

```
#> nice -n -5 yes    (se ejecuta el programa 'yes' con prioridad -5)
                     (dejar ejecutando 'yes' y en otra terminal se analiza con 'ps')
#> ps -el
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0 12826 12208  4  75  -5 -   708 write_ pts/2    00:00:00 yes

#> renice 7 12826
12826: prioridad antigua -5, nueva prioridad 7
#> ps -el
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0 12826 12208  4  87   7 -   708 write_ pts/2    00:00:15 yes
```

(obsérvese el campo NI en el primer caso en -5, y en el segundo con renice quedó en 7, en tiempo real)

nohup y &

Cuando se trata ejecutar procesos en background (segundo plano) se utiliza el comando **nohup** o el operador & . Aunque realizan una función similar, no son lo mismo.

Si se desea liberar la terminal de un programa que se espera durará un tiempo considerable ejecutándose, entonces se usa . Esto funciona mejor cuando el resultado del proceso no es necesario mandarlo a la salida estándar (stdin), como por ejemplo cuando se ejecuta un respaldo o se abre un programa Xwindow desde la consola o terminal. Para lograr esto basta con escribir el comando en cuestión y agregar al final el símbolo & (ampersand).

```
$> yes > /dev/null &
$> tar czf respaldo /documentos/* > /dev/null/ &
$> konqueror &  (con estos ejemplos se ejecuta el comando y se libera la terminal regresando el prompt)
```

Sin embargo lo anterior produce que el padre del proceso PPID que se invocó sea el proceso de la terminal en si, por lo que si cerramos la terminal o salimos de la sesión también se terminaran los procesos hijos que dependan de la terminal, no muy conveniente si se desea que el proceso continué en ejecución.

Para solucionar lo anterior, entonces se usa el comando **nohup** que permite al igual que '&' mandar el proceso y background y que este quede inmune a los hangups (de ahí su nombre nohup) que es cuando se cuelga o termina la terminal o consola de la cual se ejecutó el proceso.

```
$> nohup yes > /dev/null &
$> nohup czf respaldo /documentos/* > /dev/null/
$> nohup konqueror
```

Asi se evita que el proceso se "cuelgue" al cerrar la consola.

jobs

Si por ejemplo, se tiene acceso a una única consola o terminal, y se tienen que ejecutar varios comandos que se ejecutarán por largo tiempo, se pueden entonces como ya se vió previamente con **nohup** y el operador '&' mandarlos a segundo plano o background con el objeto de liberar la terminal y continuar trabajando.

Pero si solo se está en una terminal esto puede ser difícil de controlar, y para eos tenemos el comando **jobs** que lista los procesos actuales en ejecución:

```
#> yes > /dev/null &
[1] 26837
#> ls -laR > archivos.txt &
[2] 26854
#> jobs
[1]-  Running                 yes >/dev/null &
[2]+  Running                 ls --color=tty -laR / >archivos.txt &
```

En el ejemplo previo, se ejecutó el comando **yes** y se envió a background (&) y el sistema devolvió [1] 26837, indicando asi que se trata del trabajo o de la tarea [1] y su PID, lo mismo con la segunda tarea que es un listado recursivo desde la raíz y enviado a un archivo, esta es la segunda tarea.

Con los comandos **fg** (foreground) y **bg** background es posible manipular procesos que esten suspendidos temporalmente, ya sea porque se les envió una señal de suspensión como STOP (20) o porque al estarlos ejecutando se presionó ctrl-Z. Entonces para reanudar su ejecución en primer plano usaríamos **fg**:

```
#> jobs
[1]-  Stopped                 yes >/dev/null &
[2]+  Stopped                ls --color=tty -laR / >archivos.txt &
#> fg %1
#> jobs
[1]+  Running                 yes >/dev/null &
[2]-  Stopped                ls --color=tty -laR / >archivos.txt &
```

Obsérvese como al traer en primer plano al 'job' o proceso 1, este adquirió el símbolo [+] que indica que esta al frente. Lo mismo sería con **bg** que volvería a reinicar el proceso pero en segundo plano. Y también es posible matar los procesos con **kill** indicando el número que devuelve **jobs**: **kill %1**, terminaría con el proceso en **jobs** número 1.

top

Una utilería muy usada y muy útil para el monitoreo en tiempo real del estado de los procesos y de otras variantes del sistema es el programa llamado **top**, se ejecuta desde la línea de comandos, es interactivo y por defecto se

actualiza cada 3 segundos.

```
$> top
top - 13:07:30 up 8 days,  6:44,  4 users,  load average: 0.11, 0.08, 0.08
Tasks: 133 total,   1 running, 131 sleeping,   0 stopped,   1 zombie
Cpu(s):  0.0%us,  0.2%sy,  0.0%ni, 99.7%id,  0.0%wa,  0.0%hi,  0.2%si,  0.0%st
Mem:    497356k total,   472352k used,    25004k free,    21500k buffers
Swap:  1156640k total,   257088k used,   899552k free,    60420k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
26156 sergon    15   0  2160 1016  784 R    1  0.2   0:00.93 top
    1 root      15   0  2012  616  584 S    0  0.1   0:00.98 init
    2 root      RT   0     0    0    0 S    0  0.0   0:00.29 migration/0
    3 root      34  19     0    0    0 S    0  0.0   0:00.00 ksoftirqd/0
    4 root      RT   0     0    0    0 S    0  0.0   0:00.00 watchdog/0
    5 root      RT   0     0    0    0 S    0  0.0   0:00.38 migration/1
...
```

Estando adentro de la apliación, presionando 'h' muestra una ayuda de los posibles comandos que permiten configurar **`top`**, por ejemplo, al presionar 's' pregunta por el tiempo en segundos de actualización, etc.

Estas son algunas de las herramientas, las más importantes y usadas, para adminsitrar procesos, hay varios programas en ambientes gráficos que en una sola pantalla permiten todo lo anterior y más, y en línea de comandos te recomiendo **`htop`** (http://htop.sourceforge.net/), que es como un **`top`** pero en esteroides.
