PROCESOS Y TRABAJOS EN LINUX

En el sistema Linux, cuando una aplicación se ejecuta genera lo que llamamos un proceso. Este proceso tiene un identificador de proceso (PID) y, si la propia aplicacion ejecuta otra aplicación, generará otro proceso hijo de la aplicación inicial, que también tendrá su propio (PID).

Esta regla tiene la excepción de los programas internos de la shell, que como ya vimos en el apartado de la shell, utilizan el propio PID de la shell para su ejecución.

A continuación veremos las diferentes aplicaciones que se utilizan en Linux para monitorizar y gestionar procesos.

Monitorizar procesos

Pueden utilizarse diversos comandos para inspeccionar procesos y son especialmente útiles para localizar y finalizar procesos que no son necesarios o son sospechosos.

Una de las herramientas más importantes de administración de procesos es el comando **ps** , que muestra el estado de los procesos. Incluye algunas opciones muy útiles y ayuda a monitorizar lo que está ocurriendo en el sistema. Su sintaxis es bastante simple:

```
ps [options]
```

Algunas de las características más comunes son:

- Mostrar todos los procesos: Por defecto, **ps** muestra solo los procesos arrancados desde la propia terminal (por ejemplo xterm, login en modo texto o login remoto). Las opciones **-A** y **-e** muestran todos los procesos en el sistema, y **x** muestra todos los procesos de los que es propietario el usuario que ejecutó el comando. La opción **x** también incrementa la cantidad de información mostrada sobre cada proceso.

- Mostrar procesos de un usuario: Se pueden mostrar los procesos de un usuario concreto con la opción **-u** *user,* **U** *user,* y **--User** *user.* La variable *user* debe ser un nombre de usuario o el ID de un usuario.
- Mostrar información extra: Las opciones **-f** , **-l** , **j** , **l u** y **v** expanden la información sacada en pantalla. La mayoría de los formatos de salida incluyen una línea por proceso, pero **ps** puede mostrar tal cantidad de información que sea imposible mostrar en una sola línea la información de un proceso concreto.
- Mostrar jerarquía de procesos: Las opciones **-f** , **-H** y **--forest** agrupan procesos y muestran la jerarquía y relaciones entre ellos. Estas opciones pueden ser útiles si se desea saber cual es el padre de un proceso, o viceversa.

Ejemplo del comando **ps** :

```
$ ps -u usuario --forest
PID TTY TIME CMD
1461 ? 00:00:00 ck-launch-sessi
1512 ? 00:00:00 \_ ssh-agent
1587 ? 00:00:02 \_ x-session-manag
1600 ? 00:00:04 \_ xfwm4
1602 ? 00:00:04 \_ xfdesktop
1604 ? 00:00:21 \_ xfce4-panel
1610 ? 00:00:00 \_ xfce4-menu-plug
2189 ? 00:00:06 | \_ pidgin
1615 ? 00:00:00 \_ xfce4-mixer-plu
4631 ? 00:00:14 epdfview
2795 ? 00:00:09 xfce4-terminal
2796 ? 00:00:00 \_ gnome-pty-helpe
2797 pts/0 00:00:00 \_ bash
5989 pts/0 00:00:00 \_ ps
2354 ? 00:01:24 exe
2383 ? 00:03:34 \_ chrome
2388 ? 00:00:00 \_ chrome
2389 ? 00:00:00 \_ chrome-sandbox
2390 ? 00:00:00 \_ chrome
```

```
2393 ? 00:00:00 \_ nacl_helper_boo
2394 ? 00:00:00 \_ chrome
2422 ? 00:00:02 \_ chrome
2427 ? 00:03:43 \_ chrome
2950 ? 00:01:13 \_ chrome
2331 ? 00:00:00 gvfs-afc-volume
2329 ? 00:00:00 gvfs-gphoto2-vo
2325 ? 00:00:00 gvfs-gdu-volume
1643 ? 00:00:00 gnome-keyring-d
1635 ? 00:00:00 gconfd-2
1631 ? 00:00:00 polkit-gnome-au
1629 ? 00:00:00 update-notifier
1627 ? 00:00:01 nm-applet
1622 ? 00:00:02 Thunar
1612 ? 00:00:00 gvfsd
1609 ? 00:00:01 xfce4-settings-
1608 ? 00:00:25 gam_server
1606 ? 00:00:00 xfce4-power-man
1601 ? 00:00:00 xfsettingsd
1594 ? 00:00:00 xfconfd
1591 ? 00:00:00 dbus-daemon
1590 ? 00:00:00 dbus-launch
```

Aquí se muestran los procesos iniciados por el usuario "usuario" y la jerarquía y relación entre ellos.

Una variante del comando **ps** es **pstree** que muestra procesos activos en formato de árbol genealógico (procesos hijos vinculados a sus respectivos procesos padre).

Para monitorizar continuamente los procesos, mostrando información como utilización de memoria y CPU de cada uno de ellos, se usa el comando **top** . La tecla *[H]* proporciona ayuda sobre la utilización del programa. Puede utilizarse para modificar la prioridad de un proceso. Esta herramienta es un

programa en modo texto que se puede arrancar desde una terminal, aunque también existen algunas variantes en modo gráfico. Como cualquier comando Linux, acepta diferentes opciones, algunas de las más útiles son las siguientes:

-d delay *:* Especifica el intervalo entre actualizaciones que por defecto es 5 segundos. Esto indica el tiempo que tarda el programa en actualizar los datos que se están presentando por pantalla.

-p PID: Para monitorizar procesos específicos se puede usar esta opción.

-n iter: Se puede especificar que muestre cierto número de actualizaciones, y después finalice el programa.

k: Para matar un proceso, el comando preguntará por el PID y matará dicho proceso siempre que esté capacitado de hacerlo.

q: Salir del programa.

r: Se puede cambiar la prioridad de un proceso con esta opción. Habrá que insertar a continuación el PID del proceso seguido de un nuevo valor de prioridad. Un valor positivo decrementará su prioridad y un valor negativo la incrementará, asumiendo que la prioridad por defecto con que inicia un proceso es 0.

P: Ordena la salida por pantalla según el uso de la CPU que hace cada proceso.

M: Ordena la salida por pantalla según el uso de la memoria que hace cada proceso.

Una variante del programa top es **htop** , que a diferencia de top muestra una lista completa de procesos en ejecución en lugar de los procesos que están consumiendo recursos. Además usa colores y da un aspecto visual más amigable ofreciendo información sobre la CPU, la swap y el estado de la memoria.

Se puede obtener el PID de un programa concreto con el comando **pidof** , siempre que dicho programa esté en ejecución. Su sintaxis es como sigue:

```
pidof [options] programa
```

Como programa se debe especificar el nombre del programa cuyo PID deseemos consultar. El PID puede ser uno solo o varios, ya que un programa puede tener distintos procesos en ejecución relacionados entre sí (procesos padre e hijos).

Ejemplo de **pidof:**

```
$ pidof chrome
3061 3016 2645 1888 1855 1851 1849 1844
```

Otra aplicación que permite visualizar los procesos filtrando por cierto criterio es el programa **pgrep** . Funciona de un modo similar al comando **grep** pero esta variante permite filtrar procesos según un criterio dado. La sintaxis básica es la siguiente:

```
pgrep [-flvx] [-d delimitador] [.n|-o] [-P ppid] [-g pgrp,...] [-s sid,...] [-u euid,
...] [-U uid,...] [-G gid,...] [-t term,...] [patrón]
```

Como se puede ver, existen muchos criterios de filtrado para **pgrep** , permitiendo un control total en la selección de criterio a la hora de mostrar procesos. Este comando es muy utilizado en scripts, donde el filtrado y selección de procesos para su gestión se vuelve fundamental para no cometer errores en la automatización de tareas. Un ejemplo sencillo de utilización de este comando es, por ejemplo, el de filtrar los procesos sshd del usuario root. Esto se realiza del siguiente modo:

```
# pgrep -u root sshd
4532
19485
```

FINALIZACIÓN DE PROCESOS

Un programa puede llegar a quedarse totalmente sin respuesta o bien puede ocurrir que se desee finalizar un programa que no debería estar arrancado. En estos casos el comando **kill** es la herramienta a usar.

Este comando envía una señal (es un método que Linux usa para comunicarse con los procesos) a un proceso. La señal estándar cuando no se informa ninguna señal es SIGTERM, con el valor numérico 15 que solicita al programa en cuestión su finalización. El proceso no necesariamente obedece a la señal a menos que la señal sea SIGKILL. En algunos casos la señal SIGHUP puede interpretarse como orden para que el proceso lea nuevamente su(s) archivo(s) de configuración.

Por ejemplo, para enviar la señal SIGTERM al proceso número 4902:

```
# kill -SIGTERM 4902
```

Algunas de las señales más utilizadas se listan a continuación:

- **SIGHUP:** Termina o reinicia el proceso. Valor numérico 1.
- **SIGINT:** Interrumpe el proceso, igual a [ClrI]+[C]. Valor numérico 2.
- **SIGQUIT:** Cierra el proceso. Valor numérico 3.
- **SIGKILL:** Fuerza la finalización del proceso. Valor numérico 9.
- **SIGTERM:** Solicita al proceso para finalizar. Valor numérico 15.

Existen otros comandos muy relacionados con **kill** para la eliminación de procesos. El comando **pkill** funciona del mismo modo que el comando **pgrep** visto anteriormente, salvo que la única diferencia es que el primero le envía la señal por defecto **SIGTERM** (se puede modificar con la opción *-singal*) y el segundo solo muestra el PID del proceso por pantalla.

Otro de los comandos utilizados para eliminar procesos es **killall** . Este comando envía una señal a todos los procesos en ejecución que se relacionen con el patrón dado. Por defecto, al igual que **pkill** , envía la señal **SIGTERM**, siendo modificable por sus parámetros. La diferencia con **pkill** reside en su posibilidad de seleccionar el criterio de eliminación de procesos mediante

expresiones regulares, eliminación de procesos de forma interactiva y verificación de que todos los procesos se han eliminado con éxito. Su sintaxis básica es la siguiente:

```
killall [-Z patron] [-e] [-g] [-i] [-q] [-r] [-s señal] [-u usuario] [-v] [-w] [--ign
ore-case] [-V] patron
```

Las opciones mas destacadas de este comando son las siguientes:

- **--ignore-case:** Ignora la sensibilidad de mayúsculas a la hora de buscar el *patrón* dado.
- **-i, --interactive:** Pregunta de forma interactiva antes de eliminar los procesos.
- **-r, --regexp:** Permite el uso de expresiones regulares en el *patron* dado.
- **-u, --user:** Elimina solo los procesos del *usuario* dado.
- **-w, --wait:** Espera a que todos los procesos se eliminen antes de finalizar el comando. Esto hace que el comando verifique cada segundo y si algún proceso todavía existe espera hasta que no quede ninguno.

Tareas en Primer y Segundo Plano (jobs), tareas desatendidas

Una de las tareas más básicas de la administración de procesos es controlar si un proceso se ejecuta en primer o segundo plano. Cuando se inicia un programa, éste normalmente toma el control de la terminal impidiendo realizar cualquier otra tarea. Para liberar la terminal basta con pulsar *control + Z* que pausa el programa.

El inconveniente de este procedimiento es que el programa no sigue ejecutándose. Si se pausan procesos se pueden recuperar con **fg** y si hay más de uno debe añadirse el número de tarea que se quieren recuperar. El número de tarea asociado con un terminal se obtiene con el comando **jobs** .

```
$ jobs
[1]-  Stopped                 top
[2]+  Stopped                 htop
```

Para recuperar el proceso *htop* a primer plano, se ejecuta lo siguiente:

```
$ fg 2
```

Una variante de **fg** es **bg** que, a diferencia de **fg** que devuelve la tarea al primer plano, devuelve una tarea a un segundo plano.

El comando **fg** es útil para programas que no necesiten interacción con el usuario y puedan realizar su tarea manteniendo libre la shell. Si se inicia un programa GUI (gráfico) desde una shell, ésta quedará bloqueada por el programa. Para retomar el control de la shell, hay que pulsar *control* + *Z* y recuperar el trabajo con **bg** , lo que permite al programa ejecutarse en segundo plano y dejar la terminal libre.

Sintaxis **bg** :

```
bg [job_spec]
```

La alternativa a no tener que pausar y volver a ejecutar los programas con **bg** es añadir un caracter ampersand [&] al final de la línea del comando a ejecutar que permitirá iniciar un programa sin que tome el control de la terminal.

```
$ gedit &
[2] 23055
```

Los números que se muestran al ejecutar el comando en segundo plano corresponden al número de la tarea y al PID, respectivamente. El número de una tarea puede utilizarse como argumento del comando **kill** , siempre y cuando esté precedido del símbolo de tanto por ciento [%]:

```
$ kill %2
```