Gestión de procesos:

- En Linux se están ejecutando al momento un gran número de procesos:
  - procesos de sistema (kernel, daemons)
  - procesos de usuarios
- En esta sección trataremos la gestión de los procesos que se están ejecutando:
  - listar procesos en ejecución
  - detener y matar procesos
  - controlar la prioridad de ejecución

Comando ps:

- Existen varias herramientas para ver los procesos en ejecución, la más importante es el comando *ps* (sin opciones, ps sólo muestra los procesos lanzados desde el terminal actual y con el mismo EUID que el usuario quien lo lanzó:

```
$ ps

PID TTY          TIME CMD
 1625 pts/0    00:00:00 bash
 1784 pts/0    00:00:00 ps
```

- ps tiene un gran número de opciones, que se pueden especificar de 3 maneras:
  - opciones UNIX: pueden agruparse y se preceden por un guión: ps -ef
  - opciones BSD: pueden agruparse y van sin guión: ps aux
  - opciones largas GNU: precedidas de dos guiones: ps –user tomas
  - -e ó ax ó -Af: muestra todos los procesos
  - -u (o U o –user) usuario: muestra los procesos de un usuario
  - u: salida en formato de usuario
  - j: salida en formato job (muestra PID, PPID, etc.)
  - l ó -l: salida en formato largo
  - f: muestra un árbol con la jerarquía de procesos
  - -o (ó -o format) formato: permite definir el formato de salida

    ```
    $ ps -o ruser,pid,comm=Comando
    ```

- Ejemplo:

```
$ ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.3   3504  1908 ?        Ss   18:05   0:02 /sbin/init
root          2  0.0  0.0      0     0 ?        S    18:05   0:00 [kthreadd]
root          3  0.0  0.0      0     0 ?        S    18:05   0:00 [ksoftirqd/0]
root          5  0.0  0.0      0     0 ?        S    18:05   0:00 [kworker/u:0]
root          6  0.0  0.0      0     0 ?        S    18:05   0:00 [migration/0]
root          7  0.0  0.0      0     0 ?        S    18:05   0:00 [watchdog/0]
root          8  0.0  0.0      0     0 ?        S<   18:05   0:00 [cpuset]
root          9  0.0  0.0      0     0 ?        S<   18:05   0:00 [khelper]
root         10  0.0  0.0      0     0 ?        S    18:05   0:00 [kdevtmpfs]
root         11  0.0  0.0      0     0 ?        S<   18:05   0:00 [netns]
root         12  0.0  0.0      0     0 ?        S    18:05   0:00 [sync_supers]
```

```
root         13  0.0  0.0      0     0 ?        S    18:05   0:00 [bdi-default]
root         14  0.0  0.0      0     0 ?        S<   18:05   0:00 [kintegrityd]
root         15  0.0  0.0      0     0 ?        S<   18:05   0:00 [kblockd]
root         16  0.0  0.0      0     0 ?        S<   18:05   0:00 [ata_sff]
```

podemos ver los siguientes campos:

- %CPU y %MEM: porcentajes de uso de CPU y memoria.
- VSZ: memoria virtual del proceso, en KBytes.
- RSS: tamaño de la memoria residente (resident set size) en KBytes.
- STAT: estado del proceso; puede ser:

Código	Significado
D	Uninterruptible sleep (usualmente IO
R	Ejecutándose (running) o en cola de ejecución
S	Interruptible sleep (p.e. esperando un evento)
T	Detenido
Z	Proceso zombie

si se utiliza el formato BSD puede aparecer otro código acompañado al principal:

Código	Significado
<	alta prioridad
N	baja prioridad
L	páginas bloqueadas (locked) en memoria
s	líder de sesión
+	Proceso en foreground

Comando pstree

- muestra el árbol de procesos (similar a ps f)

```
init-+-acpid
     |-atd
     |-bonobo-activati
     |-clock-applet
     |-cron
     |-cupsd
     |-dbus-daemon-1
     |-dcopserver
     |-dirmngr
     |-2*[esd]
     |-events/0-+-aio/0
     | |-ata/0
     | |-ata/1
     |           |-kblockd/0
     | |-khelper
```

- Con *pstree -p* podemos visualizar también el PID del proceso.

Comando top

ps nos daba una versión estática de los procesos, mientras que top:

- top nos da una lista actualizada a intervalos

```
top - 20:43:56 up  2:38,  2 users,  load average: 2,00, 2,01, 1,96
Tareas:  70 total,   3 running,  67 sleeping,   0 stopped,   0 zombie
%Cpu(s): 98,6 us,  1,4 sy,  0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
KiB Mem:    508344 total,   124508 used,   383836 free,    21908 buffers
```

```
KiB Swap:   521212 total,        0 used,   521212 free,    68708 cached

  PID USER      PR  NI  VIRT  RES  SHR S  %CPU %MEM    TIME+  COMMAND
 2071 nosudo    20   0  5428  280  228 R  49,5  0,1  38:13.33 yes
 2072 nosudo    20   0  5428  284  228 R  49,2  0,1  38:21.20 yes
  357 syslog    20   0 30048 1380 1076 S   0,3  0,3   0:00.94 rsyslogd
  971 root      20   0  9852 1460 1064 S   0,3  0,3   0:22.70 VBoxService
    1 root      20   0  3504 1908 1296 S   0,0  0,4   0:02.00 init
    2 root      20   0     0    0    0 S   0,0  0,0   0:00.02 kthreadd
    3 root      20   0     0    0    0 S   0,0  0,0   0:00.36 ksoftirqd/0
    5 root      20   0     0    0    0 S   0,0  0,0   0:00.49 kworker/u:0
    6 root      rt   0     0    0    0 S   0,0  0,0   0:00.00 migration/0
    7 root      rt   0     0    0    0 S   0,0  0,0   0:01.44 watchdog/0
    ..........
```

- en la cabecera nos muestra un resumen del estado del sistema:
  - hora actual, tiempo que el sistema lleva encendido, el número de usuarios conectados y la carga media del sistema para los últimos 1, 5, y 15 minutos
  - número total de tareas y resumen por estado
  - estado de ocupación de la CPU y la memoria
- por defecto, los procesos se muestran ordenados por porcentaje de uso de CPU (los más costosos arriba).
- pulsando h mientras se ejecuta top, obtenemos una lista de comandos interactivos.
- para salir, q
- Algunos campos de top:
  - VIRT: Tamaño total del proceso (código, datos y librerías com- partidas cargadas), VIRT=SWAP+RES
  - SWAP: Memoria que ha sido swapped out o que aún no ha sido cargada
  - RES: Memoria residente (RAM ocupada por el proceso)
  - CODE y DATA: Memoria ocupada por el código y datos (datos y pila, pero no librerías compartidas) del proceso.
  - SHR: Memoria compartida (memoria que puede ser compartida con otros procesos)
  - P: Última CPU usada (SMP)
  - nFLT: Número de fallos de página para el proceso

Manipulación de procesos:

Mediante la combinación de Ctrl-C y Ctrl-Z podemos terminar y detener un proceso.

- esas combinaciones envían señales a los procesos.
- el comando básico para enviar señales a un proceso es kill.

Commando *kill*

- kill -l lista el conjunto de señales

```
$ 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			

- para ver su significado, ver *man 7 signal* Sintaxis de **kill**

```
kill [señal] PID
```

- *señal* puede indicarse mediante el número o el código:
  - **kill -9** y **kill -KILL** son equivalentes.
  - las señales más comunes son:
    - SIGHUP (1): cuelgue del terminal o muerte del proceso controlador
    - SIGTERM (15): mata el proceso permitiéndole terminar correctamente
    - SIGKILL (9): mata el proceso sin permitirle terminar
    - SIGSTOP (19): para el proceso
    - SIGCONT(18): continúa si parado.
    - SIGINT (2): Interrupción de teclado (Ctrl-C)
    - SIGTSTP (20): stop de teclado (Ctrl-Z)
    - SIGQUIT (3): salida de teclado (Ctrl-\)

    Algunas características de las señales:
  - La señal que se envía por defecto es TERM (15)
    - los procesos pueden ignorar esta señal y no terminar.
    - las señales KILL (9) y STOP(19) no pueden ignorarse.
  - En bash, cuando enviamos una señal SIGHUP a un shell, este se lo reenvía a todos sus hijos.
  - Cuando cerramos un terminal en un entorno gráfico, o abandonamos una sesión, se envía un SIGHUP a todos sus hijos
  - La mayoría de los demonios (daemons) responden a la señal SIGHUP volviendo a leer sus ficheros de configuración:
    - en vez de matar y reiniciar un demonio podemos hacer un kill -HUP para reiniciarlo

Otros comandos útiles

pgrep busca una lista de procesos para localizar el PID a partir del nombre (similar a **ps | grep**)

```
$ pgrep -u root sshd # PID del proceso sshd de root
```

pkill permite enviar señales a los procesos indicándolos por nombre en vez de por PID.

```
$ pkill -9 proceso
```

- **Cuidado con pkill:** si hay varios procesos con el mismo nombre los mata a todos.

- **killall** similar a pkill, pero no admite patrones en el nombre, y tiene otras opciones.

- **nohup** normalmente, cuando salimos de un login shell (logout) o cerramos una un terminal, se envía una señal SIGHUP a todos los procesos hijos.

– si lanzamos un proceso en background y salimos de la sesión el proceso muere al morir el shell desde que lo iniciamos. El comando **nohup** permite lanzar un comando ignorando las señales SIGHUP.

```
$ nohup comando # la salida del comando se redirige al fichero nohup.out
```

- **exec** es otra forma de crear un proceso que no se muere al morirse el shell.
- **Ejecución en background y foreground**: las órdenes lanzadas en background permiten tener el control de la consola mientras se ejecuta en segundo plano. Si queremos que el proceso se ejecute en primer plano, tenemos que hacernos valer de la orden fg.

```
$ yes > /dev/null &  # lanzado en background
[1] 27058
$ fg 1
yes > /dev/null
```

- Ejemplos

Ejemplo1:

```
$ yes >/dev/null &
[1] 9848
$ yes >/dev/null &
[2] 9849
$ ps
PID TTY          TIME CMD
9834 pts/7    00:00:00 bash
9848 pts/7    00:00:02 yes
9849 pts/7    00:00:01 yes
9850 pts/7    00:00:00 ps
$ kill –STOP 9849
[2]+ Stopped
$ jobs
[1]– Running
[2]+ Stopped
$ kill –CONT 9849
$ jobs
[1]– Running
[2]+ Running
$ kill –KILL 9848
$ kill –1 9849
[1]– Matado
[2]+ Colgar
```

Ejemplo2:

```
$ yes > /dev/null &
[1] 14724
$ yes > /dev/null &
[2] 14725
yes >/dev/null &
yes >/dev/null &
yes >/dev/null
yes >/dev/null
$ ps
PID TTY          TIME CMD
```

```
7083 pts/3 00:00:00 bash
14724 pts/3 00:00:02 yes
14725 pts/3 00:00:02 yes
14726 pts/3 00:00:00 ps
$ pgrep yes
14724
14725
$ pkill -9 yes
$ ps
PID TTY
7083 pts/3 00:00:00 bash
14730 pts/3 00:00:00 ps
[1]- Matado yes > /dev/null
[2]+ Matado yes > /dev/null
```

Ejemplo 3

```
$ nohup yes > /dev/null &
[1] 9620
$ kill -HUP 9620
$ ps
PID TTY TIME CMD
8293 pts/5 00:00:00 bash
9620 pts/5 00:00:13 yes
9621 pts/5 00:00:00 ps
$ kill 9620
[1]+ Terminado nohup yes > /dev/null
```

Comandos nice y renice

Cuando un proceso se ejecuta, lo hace con una cierta *prioridad*.

- las prioridades van desde -20 (más alta) hasta 19 (más baja).
- los procesos se ejecutan con prioridad 0 por defecto. Además, un usuario normal sólo puede asignar prioridades más bajas (números positivos), mientras que *roto* puede asignar prioridades más altas (números negativos).

Sintaxis:

```
$ nice -n <ajuste> <comando>  # la prioridad es modificada por ajuste
```

Ejemplo:

```
$ cat script1.sh
#!/bin/bash
yes > /dev/null &
$ nice -n 10 ./script1.sh
$ ps -o pid,pri,ni,stat,cmd
  PID PRI  NI STAT CMD
 4387  19   0 Ss   -bash
 4585  19   0 R    yes
 4613  19   0 R+   ps -o pid,pri,ni,stat,cmd
$ nice -n -1 4585 # No va a funcionar al bajar la prioridad y no ser root.
```

Con *renace* podemos modificar la prioridad:

Sintaxis:

```
$ renice pri [-p pid] [-g pgrp] [-u user] # la prioridad es modificada por ajuste
```

Si por ejemplo:

```
$ ./script1.sh
[1] 4455
$ renice 10 -p 4455 # cambia por pid
$ renice 3 -u user01 # cambia para los procesos de usuario especificado.
```

Análisis básico del rendimiento del sistema

Además del ps y el top, existen otro comandos básicos que nos pueden mostrar el estado del sistema en cuanto a uso de CPU y consumo de memoria, no obstante, veremos otros más relacionados con estos usos (vmstat y sar)

ultime: muesta la hora actual, el tiempo que el sistema lleva encendido, el número de usuarios conectados y la carga media del sistema para los últimos 1, 5 y 15 minutos (lo mismo que en la primera línea de top).

```
$ uptime
 11:35:34 up  1:40,  5 users,  load average: 9,80, 8,26, 5,81
```

w: nos da la información sobre los usuarios y sus procesos.

```
$ w
 12:08:15 up  2:13,  5 users,  load average: 9,73, 9,58, 9,21
USUARIO  TTY      DE              LOGIN@   IDLE   JCPU   PCPU WHAT
admin    tty1                     19abr14  1:05m  2.74s  2.63s -bash
admin    pts/0    192.168.1.100   19abr14 10días  0.43s  0.43s -bash
admin    pts/1    10.1.80.169     jue19    4días  0.31s  0.31s -bash
admin    pts/2    10.1.80.169     jue19    4días  0.33s  0.33s -bash
admin    pts/5    192.168.1.100   11:27    1.00s  1:43   0.00s w
```

Tenemos:

- LOGIN la hora a la que se conectó el usuario.
- IDLE tiempo que lleva ocioso el terminal.
- JCPU el tiempo de CPU consumido por los procesos que se ejecutan en el TTY.
- PCPU tiempo consumido por el proceso actual (el que aparece en la columna WHAT).

free: muestra la cantidad de memoria libre y usada en el sistema, tanto para la memoria física como para el swap, así como los bufares usados por el kernl (similar a la cabecera de top).

Ejemplo:

```
$ free
             total       usado       libre     compart.     búffers      almac.
Mem:        508344      201480      306864           0       29324      130460
-/+ buffers/cache:       41696      466648
Intercambio:       521212           0      521212
```

- Columna *shared* está obsoleta.
- Opciones: -b, -k, -m, -g memoria en bytes/Kb/Mb/Gb.
- Opción: -t muestra una línea con el total de memoria (física + swap).
- Opción: -s delay, muestra la memoria de forma continua, delay = segundos.

htop: Es una herramienta gráfica, basada en top, pero con muchas más opciones de gestión de procesos. Es como el panel de tareas de Windows. Se pueden finalizar procesos y otras tareas.

Ejercicios:

(a) Estudia la orden uptime:

- ¿Cuánto tiempo lleva funcionando el sistema?
- ¿Cuántos usuarios hay trabajando?
- ¿Qué orden ofrece en su cabecera la misma información que uptime?

(b) La orden potree muestra el árbol de procesos que hay en ejecución. Comprueba, haciendo uso de la orden ps -la y de valores "PID" y "PPID" mostrados para cada proceso y que efectivamente los procesos son padre e hijo.

(c) Crea el fichero /home/usuario/loop.sh con el siguiente contenido:

```
#!/bin/bash
while :
do
echo 'nada' > /dev/null &
done
```

(d) Ejecuta la orden top en una terminal y comprueba el estado del sistema, a continuación lanza loop.sh en otra. Observa cómo cambia el estado del sistema al lanzar el script. En una tercera terminal, comprueba con ps los procesos en ejecución.

(e) Usando la combinación de teclas "Control-Z" para el proceso loop.sh. Una vez parado comprueba que la información mostrada por top va cambiando, hasta llegar un momento en el que no muestra información sobre dicho proceso. Fíjate que ha aumentado el número de procesos parados.

(f) Reinicia el proceso con la orden fg y comprueba que vuelve a aparecer la información sobre el proceso.

(g) Observa si mientras está en ejecución ese proceso cambia la carga media del sistema.

(h) ¿Por qué aparece siempre el proceso loop.sh con el mismo PID si se lanza a sí mismo una y otra vez durante su ejecución?

(i) Cambia la velocidad de referesco de top a 2s.

(j) Desde el top, cambia la prioridad del proceso, dándole un valor menor, por ejemplo 10.

(k) Usando la orden nice lanza otro proceso bucle con la prioridad de 5.

(l) Observa que la CPU se le asignará más al segundo loop.sh lanzado, que tiene más prioridad, que al primero que se ejecutó, que tiene menos prioridad.

(m) Asigna mediante renice una prioridad de 19 al bucle que lanzaste con prioridad 5. ¿Cómo afecta esto a la ejecución de los dos procesos?

(n) Desde el top mata el loop.sh con prioridad 10. Fíjate que ahora, a pesar de que el que queda tiene prioridad 19, se le asigna más de la CPU que antes.

(o) Haciendo uso de la orden kill, para el proceso loop.sh que aún queda en ejecución. Después usando también kill reanúdalo y finalmente elimínalo.