

---

# **UNIDAD DIDACTICA 8**

## **GESTIÓN DE PROCESOS**

# 1. GESTIÓN DE PROCESOS.

## INTRODUCCIÓN

---

- ☞ Cada programa que se ejecuta es un proceso con recursos asignados y gestionado por el kernel.
  - ☞ La gestión de procesos comprende la **monitorización, detención y cambio de prioridad** de los procesos
  - ☞ Generalmente los procesos son gestionados automáticamente por el kernel del S.O. (son creados, ejecutados y detenidos sin la intervención del usuario).
  - ☞ Algunas veces los procesos se detendrán por razones desconocidas y será necesario reiniciar el proceso.
  - ☞ Otras veces algún proceso se ejecutará descontroladamente malgastando los recursos del sistema, entonces será necesaria una intervención manual del administrador para detener el proceso.
-

# 1. PARÁMETROS DE UN PROCESO

---

**PROCESS ID (PID):** Cada proceso tiene un número asociado que se le asigna cuando es creado. Los PIDs son números enteros únicos para todos los procesos sistema.

**USER ID & GROUP ID:** Cada proceso tiene que tener asociado unos privilegios que limiten el acceso al sistema de ficheros. Estos privilegios quedan determinados por el user ID y group ID del usuario que creo el proceso.

**PARENT PROCESS:** Todo proceso es creado por otro proceso, el proceso padre (parent process). El primer proceso iniciado por el kernel cuando el sistema arranca es el programa init. Este proceso tiene el PID 1 y es el padre de todos los procesos del sistema.

# 1. PARÁMETROS DE UN PROCESO

---

**PARENT PROCESS ID:** El PID del proceso que inicio el proceso hijo.

**ENVIROMENT:** Cada proceso mantiene una lista de variables y sus correspondientes valores. El conjunto de estas variables recibe el nombre de process enviroment. Normalmente el entorno de un proceso hijo se hereda del proceso padre a menos de que se indique de otra forma.

**CURRENT WORKING DIRECTORY:** Cada proceso tiene asociado un directorio por defecto, donde el proceso leerá/escribirá archivos, a menos que se le especifique explícitamente lo contrario.

**NICE NUMBER:** Permite al usuario modificar la prioridad de ejecución de un proceso.

---

## 2. COMANDOS DE MONITORIZACIÓN DE PROCESOS: PS, PSTREE Y TOP

---

### 🔗 ps

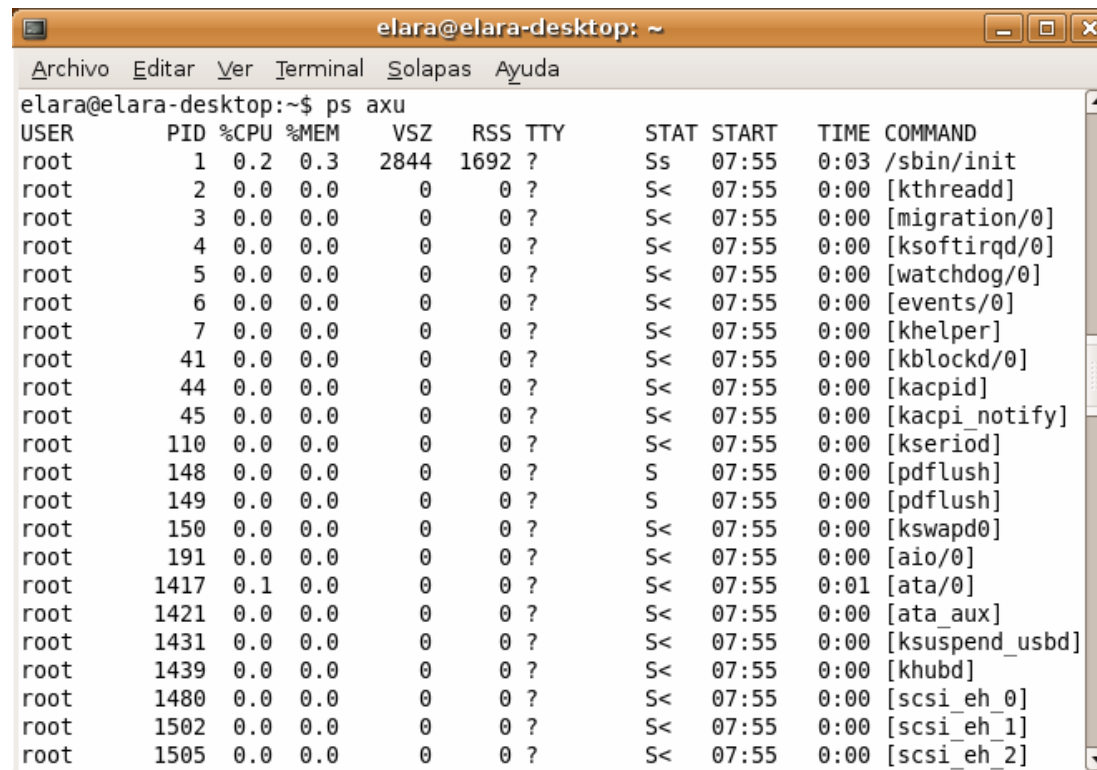
**Sintaxis:** ps [options]

Muestra la lista de procesos del sistema, y algunas de sus características: hora de inicio, uso de memoria, estado de ejecución, propietario y otros detalles. Sin opciones lista los procesos creados por el usuario actual y asociados al terminal de usuario

### Opciones:

- a Muestra los procesos creados por cualquier usuario y asociados a un terminal.
  - l Formato largo. Muestra la prioridad, el PID del proceso padre entre otras informaciones.
  - u Formato de usuario. Incluye el usuario propietario del proceso y la hora de inicio.
  - U usr Lista los procesos creados por el usuario "usr"
  - x Muestra los procesos que no están asociados a ningún terminal del usuario. Útil para ver los "demonios" (programas residentes) no iniciados desde el terminal.
-

## 2. COMANDOS DE MONITORIZACIÓN DE PROCESOS: PS, PSTREE Y TOP



USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.2	0.3	2844	1692	?	Ss	07:55	0:03	/sbin/init
root	2	0.0	0.0	0	0	?	S<	07:55	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	S<	07:55	0:00	[migration/0]
root	4	0.0	0.0	0	0	?	S<	07:55	0:00	[ksoftirqd/0]
root	5	0.0	0.0	0	0	?	S<	07:55	0:00	[watchdog/0]
root	6	0.0	0.0	0	0	?	S<	07:55	0:00	[events/0]
root	7	0.0	0.0	0	0	?	S<	07:55	0:00	[khelper]
root	41	0.0	0.0	0	0	?	S<	07:55	0:00	[kblockd/0]
root	44	0.0	0.0	0	0	?	S<	07:55	0:00	[kacpid]
root	45	0.0	0.0	0	0	?	S<	07:55	0:00	[kacpi_notify]
root	110	0.0	0.0	0	0	?	S<	07:55	0:00	[kseriod]
root	148	0.0	0.0	0	0	?	S	07:55	0:00	[pdflush]
root	149	0.0	0.0	0	0	?	S	07:55	0:00	[pdflush]
root	150	0.0	0.0	0	0	?	S<	07:55	0:00	[kswapd0]
root	191	0.0	0.0	0	0	?	S<	07:55	0:00	[aio/0]
root	1417	0.1	0.0	0	0	?	S<	07:55	0:01	[ata/0]
root	1421	0.0	0.0	0	0	?	S<	07:55	0:00	[ata_aux]
root	1431	0.0	0.0	0	0	?	S<	07:55	0:00	[ksuspend_usbd]
root	1439	0.0	0.0	0	0	?	S<	07:55	0:00	[khubd]
root	1480	0.0	0.0	0	0	?	S<	07:55	0:00	[scsi_eh_0]
root	1502	0.0	0.0	0	0	?	S<	07:55	0:00	[scsi_eh_1]
root	1505	0.0	0.0	0	0	?	S<	07:55	0:00	[scsi_eh_2]

La orden ps proporciona una información muy interesante sobre los procesos que tenemos en ejecución. Podemos saber el pid del proceso qué programa o originó el proceso, cuanta memoria ocupa, cuanta CPU consume, cuanto tiempo de ejecución lleva, ...

## 2. COMANDOS DE MONITORIZACIÓN DE PROCESOS: PS, PSTREE Y TOP

---

### 🔗 **pstree**

**Sintaxis:** `pstree [options] [PID | user]`

Este comando muestra la jerarquía de los procesos mediante una estructura de árbol. Si se especifica el PID de un proceso, el árbol empezará desde ese proceso, de lo contrario el árbol empezará por el proceso init (PID=1) y mostrará todos los procesos del sistema. Si se especifica un usuario válido se mostrará la jerarquía de todos los procesos del usuario.

#### **Opciones:**

- a Incluye en el árbol de procesos la línea de comandos que se usó para iniciar el proceso.
- c Deshabilita la unión de procesos hijos con el mismo nombre (réplicas de un mismo proceso).
- G Usa los caracteres de línea para dibujar el árbol. La representación del árbol es más clara, pero no funciona al redireccionar la salida.
- h Remarca la jerarquía del proceso actual (normalmente el terminal). No funciona al redireccionar la salida.
- n Por defecto los procesos con mismo padre se ordenan por el nombre. Esta opción fuerza a ordenar los procesos por su PID.
- p Incluye el PID de los procesos en el árbol.

## 2. COMANDOS DE MONITORIZACIÓN DE PROCESOS: PS, PSTREE Y TOP

---

### ☞ top

**Sintaxis:** top [options]

El comando top ofrece una lista de los procesos similar al comando ps, pero la salida se actualiza continuamente. Es especialmente útil cuando es necesario observar el estado de uno o más procesos o comprobar los recursos que consumen.

### **Opciones:**

- i Ignora los procesos inactivos, listando únicamente los que utilizan recursos del sistema.
- d Especifica el ritmo de actualización de la pantalla en segundos. Es posible especificar decimales.

### **Ordenes interactivos:**

- h Muestra una pantalla de ayuda.
- q Sale del programa.
- k Kill. Permite detener un proceso.
- r Renice. Permite alterar la prioridad de un proceso.



### 3. EJECUCIÓN EN 1º Y 2º PLANO

---

- œ Un proceso ejecutado en 1º plano, se ejecuta bloqueando el terminal desde el que se lanzó.
- œ Un proceso se lanza en 1º plano simplemente introduciendo su nombre en el terminal y pulsando intro.

`$firefox`

- œ Un proceso ejecutado en 2º plano, se ejecuta sin bloquear el terminal desde el que se lanzó.
- œ Los programas se pueden iniciarse en 2º plano añadiendo el carácter & al final del comando.

`$firefox &`

`[1] 1748`

El sistema operativo le asigna el trabajo 1 (indicado mediante [1]) y el PID 1748.

### 3. EJECUCIÓN EN 1º Y 2º PLANO

---

- ∞ Cuando un proceso se inicia en segundo plano, se crea un trabajo (job), al cual se le asigna un número entero, empezando por 1 y numerando secuencialmente.
- ∞ Cuando un proceso se ejecuta en segundo plano (background), la única manera de comunicarse con el proceso es mediante el envío de mensajes (signals).
- ∞ Se puede mover un programa ejecutado en primer plano al segundo plano, deteniéndolo escribiendo Ctrl-Z y después reiniciando el proceso en 2º plano, mediante la orden bg.

### 3. OPERACIONES CON PROCESOS EN PRIMER PLANO

---

Con un proceso en 1º plano podemos realizar dos acciones desde el terminal que tiene asociado:

☞ Matar el proceso: C-c (Ctrl-c ) se cancela el proceso y se liberan todos los recursos que tuviera asignados. Esta acción realmente envía una señal de interrupción al proceso, indicándole que tiene que detenerse.

Sólo podremos matar procesos sobre los que tengamos permiso. Si intentamos matar el proceso de otro usuario el sistema no nos lo permitirá, salvo a root.

☞ Parar el proceso: C-z (Ctrl-z) . En este caso sólo se detiene la ejecución del proceso, conservando su estado y sus recursos para poder continuar en el momento que se dé la orden adecuada.

## 4. ENVIAR MENSAJES A UN PROCESO: KILL

---

- ☞ Cada proceso que se ejecuta en el sistema está alerta de los mensajes enviados por el kernel o por el usuario.
- ☞ Las órdenes kill y killall se utilizan para enviar señales a un proceso.
- ☞ Estos mensajes (signals) son números enteros predefinidos y conocidos por los procesos.
- ☞ Cuando un proceso recibe un mensaje, este realiza una determinada acción.
- ☞ Existen 30 diferentes señales definidas en LINUX. Cada señal tiene un nombre y un entero.

## 4. ENVIAR MENSAJES A UN PROCESO: KILL

Nombre	Entero	Descripción
SIGHUP	1	Colgar. Esta señal es enviada automáticamente cuando un modem se desconecta. También se usa por muchos demonios para forzar la relectura del archivo de configuración. Por ejemplo, en los procesos init y inetd.
SIGINT	2	Parar el proceso y desaparece. Esta señal se envía con la secuencia de teclas Ctrl-C.
SIGKILL	9	Kill. Mata un proceso incondicionalmente e inmediatamente. Enviar esta señal es un método drástico de terminar el proceso, ya que no se puede ignorar
SIGTERM	15	Terminar. Mata un proceso de forma controlada.
SIGCONT	18	Continuar. Cuando un proceso detenido recibe esta señal continúa su ejecución.
SIGSTOP	19	Parar, pero preparado para continuar. Esta señal se envía con la secuencia de teclas Ctrl-Z.

## 4. ORDEN KILL: ENVÍO DE SEÑALES A PROCESOS

---

☞ kill

**Sintaxis:** kill [-s sigspec | -sigspec] [pids]

El comando kill permite enviar un mensaje arbitrario a un proceso, o varios, con un PID igual a pids. El parámetro sigspec es el valor entero de la señal o el nombre de la señal que enviaremos al proceso. El valor de sigspec se puede especificar en minúsculas o mayúsculas, pero normalmente se especifica en mayúsculas. Para especificar el tipo de señal se pueden usar -s sigspec o simplemente -sigspec. Si se omite sigspec en el comando kill, se toma por defecto el valor SIGTERM (señal 15, salir de manera correcta).

El valor de pids tendremos que averiguarlos utilizando la orden ps.

**Ejemplos:**

kill -9 337    ↩️ ↪️    kill -KILL 337

## 4. ORDEN KILL: ENVÍO DE SEÑALES A PROCESOS

### ☞ Ejemplos kill:

Comandos equivalentes, envían la señal SIGTERM a los procesos con PID 1000 y 1001. Si estos procesos se están ejecutando de manera "educada", recibirán la señal SIGTERM y saldrán de manera correcta después de limpiar todo lo que tenían abierto.

Kill 1000 1001	Kill -15 1000 1001
Kill -SIGTERM 1000 1001	Kill -s 15 1000 1001
Kill -s sigterm 1000 1001	kill -s TERM 1000 1001

En algunos casos los procesos se ejecutan descontroladamente e ignorarán la señal SIGTERM. Para detener un proceso en estos casos es necesario usar la señal KILL que termina el proceso incondicionalmente.

kill -9 1000 1001 ☞☞ kill -s KILL 1000 1001

## 4. ORDEN KILL: ENVÍO DE SEÑALES A PROCESOS

---

### ☞ killall

**Sintaxis:** killall [-s sigspec | -sigspec] nombre\_proceso

Esta orden es ligeramente diferente a la orden kill por dos motivos; en primer lugar utiliza el nombre de proceso en lugar del pid, y además le envía la señal a todos los procesos que tengan el mismo nombre. Por lo demás, su comportamiento es idéntico, por lo que serían equivalentes:

**\$ kill -HUP 1 ☞☞ \$ killall -HUP init**

al haber un único proceso init, con pid igual a 1.



## 5. CONTROL DE TRABAJOS DESDE EL TERMINAL: BG, FG, JOBS

---

☞ Cuando lanzamos un proceso en segundo plano obtenemos un PID y un número de trabajo. El PID es el número de proceso, que es utilizado por el S.O. para identificar de forma única al proceso. En cambio el número de trabajo es un identificador de uno o varios procesos correspondientes a un usuario.

☞ Anteriormente vimos que `C-z` detiene (suspende) un proceso y lo deja en segundo plano. También vimos como podíamos enviarle un señal para que continuara su ejecución en segundo plano. Ahora lo que vamos ver son los mecanismos para realizar una gestión más completa de esos trabajos.

## 5. CONTROL DE TRABAJOS DESDE EL TERMINAL: BG, FG, JOBS

---

### ☞ bg (background)

**Sintaxis:**       bg [jobspec]

Mueve el trabajo jobspec a segundo plano, como si se hubiese iniciado con &. Si el trabajo especificado está detenido, el comando bg lo reiniciará en segundo plano.

### Ejemplo:

\$ sleep 200 (y pulsar C-z)	Ejecutamos en 1º plano y lo detenemos
\$ bg + jobs	Lo continuamos en 2º plano
\$ sleep 201 & + jobs	Ejecutamos en 2º plano
\$ fg 2 (y pulsar C-z)	Pasamos a 1º plano el ultimo comando
\$ fg 1 (y pulsar C-z)	Pasamos a 1º plano el primer comando
\$ bg %1 \$ bg %2	Pasamos a 2º plano los dos comandos
\$ ps	

## 5. CONTROL DE TRABAJOS DESDE EL TERMINAL: BG, FG, JOBS

---

☞ **fg (foreground)**

**Sintaxis:**        fg [jobspec]

Se utiliza para traer a primer plano un trabajo que está en segundo plano, bien esté activo o bien esté detenido.

**Ejemplo:**

\$ sleep 200 (y pulsar C-z)	Ejecutamos en 1º plano y detenemos
\$ sleep 300 (y pulsar C-z)	Ejecutamos en 1º plano y detenemos
\$ fg 1 (y pulsar C-z)	Pasamos a 1º plano y detenemos
\$ bg 2	Pasamos a 2º plano
\$ fg 1	Pasamos a 1º plano

## 5. CONTROL DE TRABAJOS DESDE EL TERMINAL: BG, FG, JOBS

---

### ❧ jobs

**Sintaxis:** jobs [options] [jobspec]

Lista todas las tareas activas. Si se incluye jobspec únicamente listará la información sobre esas tareas.

Con la orden jobs podemos obtener una lista de los trabajos que hemos lanzado en el sistema. La orden jobs se utiliza como:

### Opciones:

-l Lista los PID de las tareas.

### Ejemplo:

```
elara@elara-desktop:~$ sleep 600 &
[1] 28185
elara@elara-desktop:~$ sleep 700 &
[2] 28221
elara@elara-desktop:~$ jobs
[1]-  Running                  sleep 600 &
[2]+  Running                  sleep 700 &
elara@elara-desktop:~$
```

Entre corchetes tenemos el número de trabajo, y los signos + y menos indican:

- + El trabajo es el primero de la lista
- El trabajo es el segundo de la lista

## 5. CONTROL DE TRABAJOS DESDE EL TERMINAL: BG, FG, JOBS

---

### ☞ nohup

**Sintaxis:**     nohup orden [argumentos]

La orden nohup lanza un proceso y lo independiza del terminal que estamos usando. Los procesos se organizan de forma jerárquica, de forma que si abandonamos la shell que nos conectó al sistema (abandonamos la sesión de trabajo) automáticamente se matarán todos los procesos que dependan de ella. Pero en muchas ocasiones no puede interesar lanzar un proceso y dejarlo en ejecución aun cuando hayamos cerrado la sesión de trabajo. Para esto se usa la orden nohup.

## 6. CAMBIAR LA PRIORIDAD DE LOS PROCESOS: NICE & RENICE

---

- ⌘ Cada proceso tiene una determinada prioridad de ejecución, al necesitar +/- tiempo de CPU que otros.
- ⌘ Normalmente la prioridad de los procesos es gestionada automáticamente por el kernel.
- ⌘ No obstante, LINUX ofrece la posibilidad de modificar estas prioridades y favorecer la ejecución de ciertos procesos respecto a otros.
- ⌘ La prioridad de un proceso puede determinarse examinando la columna PRI en los resultados de los comandos top y ps -l. Los valores mostrados son relativos, cuanto mayor es el PRI, mayor es el tiempo de CPU dedicado por el kernel a ese proceso.

## 6. CAMBIAR LA PRIORIDAD DE LOS PROCESOS: NICE & RENICE

```
elara@elara-desktop: ~  
Archivo Editar Ver Terminal Solapas Ayuda  
top - 08:05:47 up 10 min, 2 users, load average: 0.30, 0.64, 0.56  
Tasks: 116 total, 1 running, 115 sleeping, 0 stopped, 0 zombie  
Cpu(s): 4.4%us, 2.0%sy, 0.0%ni, 93.3%id, 0.0%wa, 0.3%hi, 0.0%si, 0.0%st  
Mem: 515580k total, 427036k used, 88544k free, 40712k buffers  
Swap: 192740k total, 0k used, 192740k free, 232332k cached  


| PID  | USER  | PR | NI | VIRT  | RES  | SHR  | S | %CPU | %MEM | TIME+   | COMMAND        |
|------|-------|----|----|-------|------|------|---|------|------|---------|----------------|
| 5137 | root  | 20 | 0  | 38124 | 12m  | 6368 | S | 4.6  | 2.5  | 0:13.30 | Xorg           |
| 5768 | elara | 20 | 0  | 74984 | 20m  | 11m  | S | 1.3  | 4.0  | 0:01.62 | gnome-terminal |
| 5653 | elara | 20 | 0  | 25736 | 11m  | 8208 | S | 0.7  | 2.3  | 0:02.08 | nm-applet      |
| 5788 | elara | 20 | 0  | 2308  | 1116 | 852  | R | 0.7  | 0.2  | 0:00.20 | top            |
| 5582 | elara | 20 | 0  | 28620 | 5808 | 3372 | S | 0.3  | 1.1  | 0:00.95 | pulseaudio     |
| 1    | root  | 20 | 0  | 2844  | 1692 | 544  | S | 0.0  | 0.3  | 0:03.06 | init           |
| 2    | root  | 15 | -5 | 0     | 0    | 0    | S | 0.0  | 0.0  | 0:00.00 | kthreadd       |
| 3    | root  | RT | -5 | 0     | 0    | 0    | S | 0.0  | 0.0  | 0:00.00 | migration/0    |
| 4    | root  | 15 | -5 | 0     | 0    | 0    | S | 0.0  | 0.0  | 0:00.00 | ksoftirqd/0    |
| 5    | root  | RT | -5 | 0     | 0    | 0    | S | 0.0  | 0.0  | 0:00.00 | watchdog/0     |
| 6    | root  | 15 | -5 | 0     | 0    | 0    | S | 0.0  | 0.0  | 0:00.08 | events/0       |
| 7    | root  | 15 | -5 | 0     | 0    | 0    | S | 0.0  | 0.0  | 0:00.04 | khelper        |
| 41   | root  | 15 | -5 | 0     | 0    | 0    | S | 0.0  | 0.0  | 0:00.60 | kblockd/0      |
| 44   | root  | 15 | -5 | 0     | 0    | 0    | S | 0.0  | 0.0  | 0:00.00 | kacpid         |
| 45   | root  | 15 | -5 | 0     | 0    | 0    | S | 0.0  | 0.0  | 0:00.00 | kacpi_notify   |
| 110  | root  | 15 | -5 | 0     | 0    | 0    | S | 0.0  | 0.0  | 0:00.02 | kseriod        |
| 148  | root  | 20 | 0  | 0     | 0    | 0    | S | 0.0  | 0.0  | 0:00.00 | pdflush        |


```

top

ps -l

```
elara@elara-desktop: ~  
Archivo Editar Ver Terminal Solapas Ayuda  
elara@elara-desktop:~$ ps -l  
F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD  
0 S 1000 5771 5768 0 80 0 - 1402 - pts/0 00:00:00 bash  
0 R 1000 5792 5771 0 80 0 - 607 - pts/0 00:00:00 ps  
elara@elara-desktop:~$
```

## 6. CAMBIAR LA PRIORIDAD DE LOS PROCESOS: NICE & RENICE

---

- ∞ El parámetro que permite al usuario modificar la prioridad de ejecución de un proceso recibe el nombre de nice number.
- ∞ Por defecto los procesos poseen un valor de nice igual a cero. Con este valor el kernel no modifica la prioridad del proceso.
- ∞ El parámetro nice puede tomar valores comprendidos entre -20 y +19. Cualquier usuario puede aumentar el valor de nice, y bajar la prioridad del proceso, pero únicamente el usuario root puede asignar números negativos a nice, e incrementar la prioridad y por lo tanto el tiempo de CPU.



## 6. CAMBIAR LA PRIORIDAD DE LOS PROCESOS: NICE & RENICE

---

### ❧ nice

Sintaxis: `nice [-n nicenumber] command`

`nice [-nicenumber] command`

El comando `nice` se usa para iniciar un proceso y proporcionarle un determinado valor al parámetro `nice`. Para los usuarios normales `nicenumber` es un entero comprendido entre 1 y 19. Para el usuario `root`, `nicenumber` también puede tomar valores negativos (y así incrementar la prioridad de un proceso) y los valores permitidos están comprendidos entre -20 y 19. `command` es cualquier orden válida del intérprete de comandos, incluyendo opciones, argumentos, redireccionamientos y el carácter especial `&`.

## 6. CAMBIAR LA PRIORIDAD DE LOS PROCESOS: NICE & RENICE

---

### ☞ Ejemplos nice:

Ejecuta el editor vi con un valor de nice igual a +10. El primer - se refiere a la opción.

```
nice -10 vi /etc/hosts.deny
```

```
nice -n 10 vi /etc/hosts.deny
```

Ejecuta el editor vi con un valor de nice igual a -10. El primer - se refiere a la opción, el segundo se refiere al valor negativo del parámetro nice. Únicamente el usuario root puede asignar un valor negativo al parámetro nice de un proceso.

```
nice --10 vi /etc/hosts.deny
```

```
nice -n -10 vi /etc/hosts.deny
```

## 6. CAMBIAR LA PRIORIDAD DE LOS PROCESOS: NICE & RENICE

---

### 🔗 renice

**Sintaxis:** `renice [+|- nicenumber] [options] targets`

Este comando permite modificar el parámetro nice de un proceso ya iniciado.

### Opciones:

- u Interpreta targets como un nombre de usuario. Cambia el parámetro nice a todos los procesos propietarios del usuario especificado.
- p Interpreta targets como un PID (comportamiento por defecto).

### Ejemplos:

**renice 19 501** ➡ Baja la prioridad del proceso con PID 501 incrementando su número nice al máximo.

**renice -20 -u pepe 501** ➡ Aumenta la prioridad de todos los procesos del usuario pepe y el proceso con PID 501. Únicamente el root puede ejecutar el comando renice con valores negativos.

# PRACTICA 7. GESTIÓN DE PROCESOS

---

**Paso 1.** ¿Qué es el PID de un proceso en GNU/Linux?

**Paso 2.** ¿Qué diferencia hay entre la opción -a y la opción -x de la orden ps?

**Paso 3.** ¿Qué es el número NICE de un proceso? ¿Qué valores puede tomar el parámetro NICE? ¿Qué usuarios pueden cambiar este parámetro?

**Paso 4.** Listar todos los archivos que contengan la cadena ".jpg" dentro de la estructura de directorios del sistema. Usar la orden ls y grep.

**Paso 5.** Ejecutar otra vez la orden anterior, pero esta vez con la prioridad más baja posible.

**Paso 6.** Abrir otro terminal, y mientras la orden anterior se ejecuta, mediante la orden ps listar todos los procesos asociados al terminal del usuario actual y obtener el PID del proceso ls. Con la orden renice otorgar la máxima prioridad a la orden ls. Después con la orden kill terminar el proceso de la manera "más correcta" posible.

# PRACTICA 7. GESTIÓN DE PROCESOS

---

**Paso 7.** Visualizar mediante el editor vi el archivo que contiene la información de los usuarios del sistema. Ejecutar el programa vi con la máxima prioridad posible.

**Paso 8.** En otro terminal, obtener el PID del editor vi y el PID de su proceso padre mediante la orden pstree.

**Paso 9.** Mediante la orden top establecer una prioridad normal al editor vi y después terminar el proceso del editor.

**Paso 10.** Ejecutar el navegador WEB mozilla desde el terminal en segundo plano. Indicar el número de trabajo y su PID.

**Paso 11.** Listar todos los archivos terminados en .gif del sistema de directorios del sistema y almacenarlo en el archivo todoslosgif. Ejecutar esta orden en segundo plano.

**Paso 12.** Mediante la orden jobs listar todos los trabajos en segundo plano del terminal.

# PRACTICA 7. GESTIÓN DE PROCESOS

---

## Procesos en 1º plano

**Paso 13.** Mirar los procesos existentes en el sistema y lanzar un proceso que dure 600 segundos en 1º plano (p.e. `sleep 600`)

**Paso 14.** Mata el proceso ¿Qué observamos? ¿Aparecen los mismos procesos que al principio?

**Paso 15.** Repetimos el paso 13 y el 14, pero ahora sólo queremos detener el proceso (no cancelarlo). ¿Qué observamos? ¿Aparecen los mismos procesos que al principio?

**Paso 16.** ¿Cómo podemos hacer para que continúe el proceso en 1º plano? ¿Se puede hacer con la orden `kill`?

**Paso 17.** Indica dos maneras para que un proceso detenido, continúe en 2º plano.

NOTA: En lugar del comando `sleep`, prueba con `firefox`, para ver mejor el efecto

# PRACTICA 7. GESTIÓN DE PROCESOS

---

## Ejecución en 2º plano

**Paso 18.** Lanzar un proceso en 2º plano y obtener su PID. ¿Cuál es su número de trabajo y nº de proceso?

**Paso 19.** En un proceso lanzado en 2º plano ¿seguirá mostrando su salida en la pantalla desde la que se dio la orden de ejecución? Busca un ejemplo.

**Paso 20.** Detén y vuelve a recontinuar en 2º plano un proceso lanzado en 2º plano