Organización de procesos.

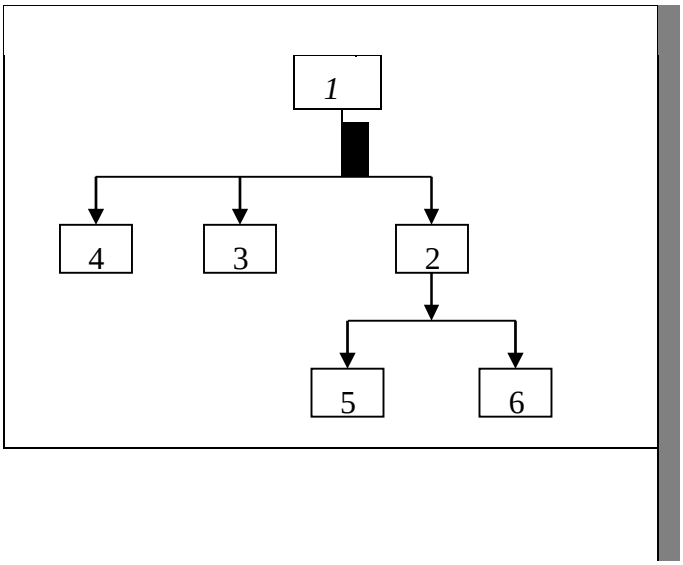
La noción de procesos es uno de los aspectos más importantes del Sistema Linux, junto con los archivos y directorios y el Shell. Un proceso, o tarea, es una instancia o petición de un programa en ejecución. Es importante hacer la distinción entre una orden y un proceso; un proceso se genera cuando se ejecuta una orden.

El Sistema Linux es un sistema multitarea ya que puede ejecutar muchas decenas o incluso centenares o miles de procesos ejecutándose en el sistema.

Cuando un usuario hace su presentación ante el sistema, Linux crea un proceso shell para que éste lo utilice, y el proceso desaparece, o muere, cuando el usuario se despide. Siempre existe al menos un proceso, y frecuentemente más de uno, asociado a cada sesión.

Los procesos se dicen que nacen cuando empiezan y mueren cuando finalizan. Normalmente, un proceso tiene un tiempo de vida relativamente breve que corresponde a la duración de la orden que se introduce en el terminal, aunque algunos comienzan cuando se arranca y perviven hasta que se desconecta.

Cuando un proceso hijo muere, el Kernel envía una señal al padre (muerte de un hijo); hasta que éste no se da por enterado el proceso hijo no desaparece totalmente, el Kernel mantiene entonces a este proceso en un estado llamado defunct (zombie).



El proceso 1 tiene 3 descendencias: proceso 2, 3 y 4.

El proceso 2 tiene 2 descendencias: proceso 5 y 6.

Un proceso puede estar en una de estas situaciones:

- RUNNIG (corriendo), cuando el proceso está siendo atendido por la CPU, es decir, está trabajando.
- READY (listo), está esperando su turno, para que la CPU acceda a él.
- BLOCKED (bloqueado), proceso en espera de algo, por ejemplo, que se pulse una tecla para continuar, etc. Cuando sucede ese algo pasa de estado bloqueado a estado READY.

- DEFUNCT (zombie). Si un padre no reconoce la muerte del hijo (porque el padre esté ignorando la señal, o porque el propio padre esté colgado), el hijo permanece como proceso zombie. Estos procesos aparecen en el listado de ps -f con <defunt> en el lugar del orden. Como un proceso no consume tiempo de CPU y no está ligado a ningún terminal, los campos STIME y TTY están en blanco.

ORDENES DE CONTROL DE PROCESOS:

Orden ps (estado de proceso, process status):

Este comando es usado para obtener información sobre el estado de los procesos que estén activos en ese momento.

- ps [-opción/es.]

Dependiendo de las opciones la información será más o menos completa.

La orden ps sin opciones obtendrá la información referente a los procesos asociados con un terminal. Por ejemplo, la salida ps muestra el ID de proceso (PID), el ID del terminal (TTY), la cantidad de tiempo de UCP en minutos y segundos que la orden ha consumido y el nombre de la orden, tal como se muestra aquí:

```
$ ps
PID      TTY       TIME     COMMAND
3211     term/41   0:05     ksh
12326    term/41   0:01     ps
12233    term/41   0:20     ksh
9046     term/41   0:02     vi
```

Este usuario tiene 4 procesos asociados al ID del terminal term/41; hay 2 shells Korn, ksh, una sesión vi y la propia orden ps.

Los números PID de proceso son asignados secuencialmente cuando se crean los procesos.

El proceso 0 es un proceso del sistema que se crea al principio cuando se arranca el Sistema LINUX, y el proceso 1 es el proceso init a partir del cual se generan todos los restantes.

Los demás PIDs de proceso comienzan en 2 y prosiguen etiquetando cada nuevo proceso con el siguiente número disponible. Cuando se alcanza el máximo número ID de proceso, la numeración comienza de nuevo saltándose cualquier número de ID de proceso en existencia. El máximo número ID puede variar, pero generalmente se fija a 32767.

Cuando se inicia o arranca un Sistema LINUX, el núcleo del Sistema (/linux) se carga en memoria y se ejecuta. El núcleo inicializa las interfaces hardware y las estructuras de datos internas y crea un proceso de sistema, el proceso 0, conocido como el intercambiador (swapper). El proceso 0 se bifurca y crea el primer proceso de nivel usuario, el proceso 1.

El proceso 1 es conocido como proceso init ya que es responsable de la preparación, o inicialización, de todos los procesos subsiguientes en el sistema. Es responsable de disponer el sistema en modo monousuario o multiusuario, de gestionar las líneas de comunicación y de generar los shells de

presentación para los usuarios. El proceso 1 existe en tanto el sistema esté en ejecución y es el antecesor de todos los procesos restantes del sistema.

Usando la orden ps con opciones, es posible controlar la información visualizada referente a los procesos en ejecución.

Principales Opciones:

-e (every) Visualiza todos los procesos que están ejecutándose en el sistema (de todos los terminales). Si en el campo TTY aparece una interrogación (?), indica que el proceso no está asignado a ningún terminal. Sirve de ayuda al administrador del sistema, pues permite saber que ocurre en el sistema en todo momento.

-f (full) Proporciona un listado completo de los procesos.

-l (long) Muestra la información en formato largo. El listado largo repite parte de la información ya vista.

-t termlist Limita el listado a los procesos que estén asociados con los terminales nombrados en la cadena "termilst". Los identificadores de los terminales pueden ser especificados con el nombre del terminal más su número, por ejemplo, tty23; si el terminal comienza por tty sólo es necesario indicar su número de terminal, 23.

-u uidlist Muestra los procesos que están siendo ejecutados por el usuario especificado en uidlist (números o nombres). En el listado aparecerá el número del usuario, UID, a no ser que se utilice la opción -f, la cual imprimirá el nombre del usuario de conexión.

Un ejemplo:

```
$ ps -f
UID    PID   PPID  C    STIME      TTY      TIME  COMMAND
rrr    3321  2187  0    15:57:07   term/41  0:01  /usr/bin/vi perf.rev
rrr    2187  1     0    15:16:16   term/41  0:00   /usr/lbin/ksh
rrr    4311  3321  0    16:35:41   term/41  0:00  sh -i
rrr    4764  4311  27   16:43:56   term/41  0:00  ps -f
```

Los campos significan:

UID Identificación del usuario (PID) propietario del proceso. Con la opción a se imprimirá el nombre del propietario (ID).

PID Número de identificación (ID) del proceso.

PPID Número de identificación (ID) del proceso padre, el número ID del proceso padre.

C Representa un índice de utilización reciente del procesador, que es utilizado por el núcleo para planificación.

STIME Tiempo de inicio del proceso en horas, minutos y segundos; un proceso con más de 24 horas de antigüedad lo da en meses y días (Puede denominarse START).

TTY El número ID del terminal que controla el proceso.

TIME Es el tiempo acumulado de UCP consumido por el proceso.

COMMAND Nombre del comando, nombre de la orden que está siendo ejecutada; el nombre total del comando y sus argumentos son mostrados con la opción -e.

F Especifica un conjunto de indicadores hexadecimales aditivos que identifican características del proceso.

Indicadores (Flags) asociados a los procesos:

00	El proceso ha terminado.
01	El proceso es un proceso del sistema.
10	El proceso está bloqueado.
20	El proceso está en memoria principal.
....	Consultar ayuda en man ps.

PRI	Prioridad del proceso (un valor superior significa una prioridad inferior, inversa al número).
NI	Número utilizado para calcular la prioridad (es el valor nice del proceso). En los S.O versión LINUX SV puede tener valores de 0 a 39 siendo el por defecto 20 (Cuando mas alto es el valor mas baja es la prioridad). En LINUX BSD tiene valores de –20 a +20 siendo por defecto 0.
ADDR ó RSS	Representa la dirección inicial en memoria del proceso.
SZ ó SIZE	Cantidad de memoria, en bloques, usada por el proceso.
WCHAN	Dirección del suceso por el cual está esperando o durmiendo. Si está en blanco el proceso se estará ejecutando.

Estos indicadores pueden variar dependiendo del equipo y la revisión del Sistema Operativo.

STAT ó S Indica el estado actual del proceso.

Abreviatura	***Significado***
0	Proceso en ejecución.
S	Proceso para dormir (sleeping).
R	Proceso para ejecutar en cola (running).
I	Proceso inactivo en creación (intermediate).
Z	Proceso zombie (stopped).
T	Proceso detenido (terminated).
X	Proceso a la espera de más memoria.

Algunos ejemplos:

Ver los procesos que están en curso de un terminal:

$ ps

PID	**TTY**	**TIME**	**COMMAND**
34	02	0:05	ksh
281	02	0:04	ps

Los dos únicos procesos son el shell de conexión (login) del usuario, que tiene el PID 34 y el propio comando ps.

Obtener un listado completo:

$ ps -f

UID PID PPID C STIME TTY TIME COMMAND grp1 34 1 0 15:57:07 02 0:01 -sh
grp1 282 34 80 17:16:16 02 0:05 ps -f

El usuario viene identificado por el nombre de conexión. El proceso padre PPID de ps es shell.

Obtener un listado largo.

$ ps -l

F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	COMD
1	S	4392	34	1	0	30	20	319	25	567a	02	0:01	sh
1	R	4392	283	34	61	80	20	382	24		02	0:00	ps

El usuario viene identificado por su número de identificación de usuario UID. El comando ps se ejecuta mientras el comando sh duerme.

Listar procesos para dos terminales.

$ ps -t "02 03"

PID	TTY	TIME	COMMAND
34	02	0:05	ksh
35	03	0:03	ksh
273	03	0:45	vi
285	02	0:05	ps

El empleo de las comillas es para hacer de la lista de terminales un argumento simple, la parte tty del nombre de terminal no es necesaria. Cada usuario tiene su propia shell.

Listar todos los procesos.

$ ps -e

PID	TTY	TIME	COMMAND
0	?	104:21	swapper
1	?	0:01	init
33	co	0:03	sh
34	02	0:05	sh
20	?	0:03	update
25	?	0:00	lpsched
29	?	0:01	cron
35	03	0:03	sh
36	04	0:01	getty

37	05	0:01	getty
273	03	0:45	vi
281	02	0:04	ps

Mostrar los procesos que están siendo ejecutados por el usuario "grupo2", en su forma completa.

$ ps -f -u grupo2

Orden Kill (Eliminación de procesos):

Se puede terminar o detener un proceso mientras éste está en ejecución, por medio del comando KILL. Por ejemplo, un programa que este ejecutándose puede contener un bucle sin fin, por lo que el proceso nunca se detendrá. O no se desea completar un proceso que haya inicializado, bien porque esté acaparando recursos del sistema o porque esté haciendo algo no deseado.

kill [-señal] [PID] [PID] [...]

Si un proceso foreground está corriendo para detenerlo, sólo es necesario pulsar en el teclado CTRLC, BREAK o DEL (puede que no funcionen todas las teclas en todos los sistemas).

Sin embargo, para terminar un proceso en modo subordinado o asociado a un terminal diferente, se usará el comando kill junto con el PID (Process Identification Number); de esta manera hará que el proceso especificado se cancele.

Esto es útil para los procesos que se ejecuten en modo background. Por ejemplo, para eliminar el proceso con PID 2312 (tal como revela ps), habrá que teclear

$ kill 2312

En LINUX los procesos reciben señales para indicarles algún suceso producido y que actúen en consecuencia. La orden Kill hace uso de algunas de estas señales. Las señales van numeradas generalmente del 0 al 15 (en LINUX SV posee 22 o más señales). y son en gran medida dependientes de la implementación de cada sistema.

La orden kill envía una señal para terminar al proceso especificado. Cuando un proceso recibe una señal puede tratarla de 3 formas diferentes:

El valor por defecto es 15, terminación normal del proceso. Las siguientes instrucciones son Idénticas:

$ kill 2312 *$ kill –15 3212*

La señal 9 (matar) es la de terminación total de un proceso:

$ kill –9 3455

A continuación, se explica el significado de algunas señales (LINUX SV y algunos LINUX):

1. SIGHUP .Desconexión, esta señal se usa para decirle a los procesos de un terminal que el Sistema ha perdido el mismo. Se genera cuando se desconecta el cable, se pierde la conexión vía MODEM, o Internet o cuando el usuario se desconecta del terminal, etc.

2. *SIGINT. Interrupción. Se genera cuando se pulsa alguna de las teclas de interrupción (CTRL-C, BREAK, o la tecla DEL, SUPR en los teclados Españoles)*

3. *SIGQUIT. Salir. Similar a la anterior. Cuando se aborta un proceso con CTRL-\ (en LINUX con CTRL-4).*

4. *SIGHILL. Instrucción ilegal. Es enviada al proceso cuando intenta ejecutar alguna instrucción ilegal. Por defecto termina la ejecución.*

5. *SIGTRAP. Ejecución de un proceso paso a paso.*

6. *SIGIOT. Fallo Hardware.*

7. *SIGEMT. Fallo Hardware.*

8. *SIGFPE. Error en coma flotante. Se termina el proceso por defecto.*

9. **SIGKILL. Matar. Terminación del proceso. No se puede ignorar.**

10. *SIGBUS. Error de acceso a memoria.*

11. *SIGSEGV. Violación de segmento de memoria.*

12. *SIGSYS. No se usa.*

13. *SIGPIPE. Error en procesos con tuberías.*

14. *SIGALARM. Señal de finalización de temporizador.*

15. **SIGTERM. Terminar. Indica al proceso que debe terminar su ejecución, es la señal que envía la orden kill por defecto. Puede ser ignorada.**

Cuando se trata de detener un proceso se debe usar siempre:

$ kill PID *o* *$ killl –15 PID*

Solo cuando el proceso no se detiene la orden anterior, hay que utilizar:

$ Kill – 9 PID

La señal de asesinar (El LINUX es u lenguaje mórbido en muchas ordenes) solo se debe emplear si tenemos claro el proceso que deseamos eliminar. Tener en cuenta que cuando un proceso ignora la señal de terminal normal (kill PID), suele tener sus razones y la eliminación del mismo puede provocar daños laterales inesperados.

Se puede eliminar todos los procesos que se hayan creado durante la sesión de presentación de su terminal actual. Un grupo de procesos es un conjunto de procesos relacionados, por ejemplo, todos los que tienen como antecesor común un shell de presentación.

Para terminar con todos los procesos se usará:

$kill 0

Eliminándose el shell de presentación actual, (comprobarlo en cada Sistema concreto).

Orden nohup (Mantenimiento de trabajos activos cuando se despide el usuario):

Una razón para ejecutar una orden en modo subordinado es que pueda durar mucho tiempo. A veces sería deseable poder iniciar la ejecución de este tipo de órdenes y después despedirse. Si se está ejecutando un trabajo en modo subordinado y el usuario se despide, éste terminará.

La orden *nohup* permite mantener en ejecución un trabajo incluso cuando uno se despide (a pesar de las señales de finalización que genera el proceso de log-off del sistema).

Cuando un terminal es desconectado, el núcleo (kernel) envía la señal SIGHUP (señal 01, página 63) a todos los procesos que estaban asociados al terminal. La intención de esta señal es hacer que todos los restantes procesos terminen.

Para asegurar que un proceso permanezca vivo después de que se despida, se utiliza la orden nohup como sigue:

$nohup orden &

Este comando se utiliza normalmente junto a procesos background.

Si la salida del resultado del proceso no es redireccionada por el usuario, tanto la salida estándar como la de error *son enviadas a un archivo llamado* ***nohup.out*** que se crea en ese momento en el directorio actual, incluyéndose en él tanto la salida deseada como el error estándar.

Si no se tiene permiso de escritura en el directorio actual, el archivo nohup.out es creado en el directorio HOME.

Ejemplos:

$ nohup find / -name ".c " &*

Permite desconectar el terminal y que find continúe su ejecución después de salir, el resultado de la búsqueda y los errores que se produzcan al intentar acceder a directorios sin permiso son enviados a un fichero llamado *noup.out.*

$ nohup find / -name ".c" > f1 2> f1.err &*

Permite desconectar el terminal y que find continúe su ejecución después de salir, En este caso, la salida estándar se he redirigido al fichero f1 y el error estandar a f1.err, de forma que se registre en ellos todo lo que suceda.

$ nohup date; who; ps -ef

Se ha efectuado una interpretación errónea, sólo la primera orden obedece a *nohup* las restantes órdenes de la línea acabarán cuando se cuelgue.

Para utilizar nohup con múltiples órdenes, o bien se precede a cada una de ellas con nohup, o preferiblemente, se coloca todas las órdenes en un archivo y se utiliza nohup nombre archivo.

Ordenes nice y renice (Cambiar la prioridad de los procesos):

A los procesos en un Sistema LINUX se les asignan los recursos para ejecución secuencialmente. El núcleo asigna la UCP a un proceso durante un intervalo de tiempo; cuando el tiempo ha transcurrido, el proceso es colocado en una de varias colas de prioridad.

Los procesos del sistema tienen una prioridad superior a las de todos los procesos de usuario.

La prioridad de los procesos de usuario depende de la cantidad de tiempo UCP que hayan empleado. Los procesos que han utilizado grandes cantidades de tiempo de UCP tienen asignadas prioridades inferiores; los que utilizan poco tiempo de UCP tienen asignadas prioridades más altas. Con la orden nice se puede influenciar esa planificación.

La orden nice **permite a un usuario ejecutar una orden** con una prioridad más baja de lo normal.

$ nice [-prioridad] orden

Si se reduce la prioridad de una orden, ésta utiliza menos tiempo de UCP y se ejecuta más lenta.

El rango de prioridades está comprendido entre -19 (la más alta) y 20 (la prioridad más baja) o entre 0 (la más alta) y 39 (la más baja) dependiendo del Sistema LINUX.

El valor del nice de los procesos, en SCO SV suele ser por defecto de 20.

Los usuarios normales del sistema pueden únicamente bajar la prioridad de un proceso usando el comando nice. Los valores válidos pueden depender de cada Sistema LINUX, por lo general van desde 1 hasta 19. Si no se especifica incremento, se supone el valor por omisión que da el sistema, 10. Por ejemplo, la orden

$ nice prog

Ejecutará la orden prog con un valor de prioridad reducido en las 10 unidades por omisión. La orden

$ nice -19 prog

lo reducirá en 19. El incremento proporcionado a nice es una unidad arbitraria, nice -19 correrá más lento que nice -9.

Únicamente el administrador puede incrementar la prioridad de un proceso usando valores negativos dentro de la orden nice:

 # nice --10 miprograma

Los sistemas LINUX recientes soportan cambiar la prioridad de un **proceso que está ya iniciado**, para poder modificar su prioridad hay que conocer el PID del proceso. La orden es renice y su sintaxis es:

renice –n prioridad PID

$ renice –n 5 10434

Como antes, el administrador puede aumentar la prioridad a un proceso en ejecución:

$ renice –n -15 1254

Para conocer el estado del nice de los procesos, se usa la orden ps, observando en el listado las columnas cuya cabecera es PRI y NI(NICE):

La columna PRI indica la prioridad de cada proceso.indica la priorid

La columna NI (NICE) del listado de ps, figura el nice (por defecto 20.) El sistema usara este valor para determinar la prioridad (PRI) del proceso.

El funcionamiento de nice y renice puede variar entre diferentes sistemas, incluso algunos no soportan estas ordenes

Ejecución en modo FOREGROUND.

Hasta ahora los comandos se han ejecutado uno a uno poniendo cada uno de los comandos en líneas individuales (un comando, una línea).

Cada comando shell que se ejecuta es considerado por el sistema como un programa independiente o un "proceso". Normalmente, cada comando (o pipeline de comandos) que se introduzca, se ejecutará mientras que el operador tendrá que estar esperando a su terminación para proseguir la secuencia.

Se utiliza el término foreground para referirse al modo de ejecución de dichos comandos o procesos.

Estos son procesos interactivos y requieren la intervención del usuario y no permiten ejecutar otro proceso o comando hasta que el anterior comando foreground no termine.

Ejemplo:

$ date
Mon Jun 9 09:50:58 EDT 1987
$ ls -l; who; wc file

El usuario deberá esperar que salga la fecha por el terminal para poder introducir la siguiente línea de comandos.

El shell como ya comentamos también permite establecerse más de un comando en una línea de comandos.

Los comandos deberán ir separados por punto y coma (;). Primero se ejecuta el comando situado más a la izquierda de la línea de comandos, después se ejecuta el inmediatamente situado a su derecha, y así sucesivamente hasta alcanzar el último comando de la línea. **comando1; comando2; comando3...**

El primer comando a ejecutarse sería comando1. El segundo, sería comando2. Y el tercer comando a ejecutarse sería comando3.

Ejemplo:

$ ls -l; who; rm documento

Esto es equivalente a:

$ ls -l
$ who
$ rm documento

Una línea de comandos se puede extender a más de una línea de pantalla introduciendo un símbolo "\" antes de pulsar la tecla <CR>. El shell mostrará el prompt secundario ">" para indicarle que puede seguir introduciendo el resto de la línea de comando. Cuando el <CR> sea pulsado sin haber pulsado previamente la tecla "\", la línea completa de comandos se ejecutará.

Cuando una orden o serie de órdenes terminan en & se dice que se ejecutan en 2º plano background:

$ (comando1; comando2; comando3) &

La ejecución de los tres comandos en el grupo se realiza en background con lo que el sistema operativo recupera de inmediato el control, y se puede proseguir con otras tareas. Si faltasen los paréntesis, los comandos "comando1" y "comando2" se ejecutarían en foreground, y solo "comando3" sería ejecutado en background.

Ejecución en modo BACKGROUND.

Una de las características que diferencia a LINUX de MS-DOS es el ser multitarea. La multitarea hace posible que cada usuario tenga más de un proceso activo en el sistema en cualquier momento.

A veces resulta útil iniciar una orden y después ejecutar otra inmediatamente, sin esperar a que la primera finalice. Esto es especialmente cierto cuando hay que ejecutar una orden o un programa que tarda mucho tiempo en finalizar.

El Shell tiene una característica especial que permite arrancar un programa y dejarlo desatendido, mientras el usuario puede seguir introduciendo comandos sin tener que esperar su terminación para lanzar el siguiente proceso. Este programa que funciona desatendido, se dice que corre en **background**, mientras que los comandos que normalmente se introducen, se ejecutan inmediatamente y están en foreground.

Para lanzar un proceso en modo background se deberá:

1. Introducir el comando.
2. Poner un ampersand (&) como último elemento de la línea de comandos.
3. Pulsar el retorno de carro.

Línea de comando &

Por ejemplo, si se tiene un programa llamado "act_mov", que consume mucho tiempo, se puede aprovechar todo ese tiempo, ejecutando el programa en background.

```
$ act_mov &
[1] 13523
$
```

Una vez lanzado a ejecución en modo background,, el sistema mostrará dos números: Un primer número, que está entre corchetes, es el ID del trabajo de esta orden. Es un número pequeño que identifica cuál de entre sus trabajos actuales es éste.

El número de identificación del proceso (PID de proceso, process id number) asignado a dicho comando en background. Dicho número, lo proporciona el Shell, es un número único que identifica este proceso entre todos los del sistema y le permitirá al usuario mantener un seguimiento sobre dicho proceso.

A continuación, el shell mostrará de nuevo el promt ($) del shell para permitirle una nueva orden de ejecución, mientras el anterior comando se ejecuta independientemente del terminal.

Por ejemplo, find es una orden que tarda mucho tiempo en ejecutarse, por eso se suele ejecutar como un proceso background:

$ find / -name ".odl" -print > old_file &*

Esta orden busca archivos desde el directorio raíz cuyos nombres terminen por .old y los guarda en el archivo old_file.

En LINUX, los procesos en background continúan ejecutándose, aunque el usuario se haya despedido del sistema.

Orden jobs (Listar los trabajos en 2º plano):

La orden jobs visualiza la lista de todos los trabajos actuales en modo principal y subordinado, así como los trabajos que están parados o suspendidos.

El número entre corchetes al comienzo de cada línea de trabajo es el ID del trabajo (JOBID). El signo + indica el trabajo actual (el último iniciado o restaurado), el signo - indica el anterior.trabajo.

```
$ jobs
[2]  +  Running                 find / -name *.c > /tmp/salida  &
[1 ] -  Stopped (SIGTSP)        man find
$
```

Para **suspender el trabajo actual que esté en modo principal** hay que usar CTRL-Z. Esto suspende el programa y devuelve el control al shell:

```
$ ^Z
 [2]  +  Stopped (SIGTSP)       find / -name *.c > /tmp/salida &
```

Si no aparece u mensaje similar a este, es que nuestro shell no soporta el control de trabajos o bien que el LINUX que se está usando no soporta la utilización de CTRL-Z para suspender trabajos.

Si todo va bien, aparecerá el indicador del sistema, podemos usar jobs para observar lo sucedido:

```
$ jobs
[2]  +  Stopped (SIGTSP)        find / -name *.c > /tmp/salida &
[1 ] -  Stopped  (SIGTSP)       man find
$
```

Si un trabajo está suspendido significa que el sistema operativo no le va asignar tiempo de procesador.

Podemos matar un proceso con la orden kill, pero indicando el JOBID del proceso en vez del PID, con la siguiente sintaxis:

kill %JOBID Finaliza un proceso indicando su JOBID

kill %% Terminar le proceso más reciente.

kill %%2 Terminar el 2º proceso más reciente.

Podemos detener un proceso con la orden stop, indicando su JOBID con la siguiente sintaxis:

- stop %JOBID Detiene el proceso indicado por JOBID

La orden jobs admite la opción –l que da un listado largo que incluye el PID ademas del JOBID.

Un trabajo suspendido se puede reanudar con la orden ***fg***(foreground) o ***bg***(background).

Ordenes fg y bg (Continuar trabajos en segundo o primer plano):

Estas órdenes continúan un trabajo detenido, en primer plano o en 2º plano.

- fg [%JOBID] Continua un trabajo en primer plano.
- bg [%JOBID] Continua un trabajo en 2º plano.

Si hay más de un trabajo detenido, hay que indicar su JOBID.

Orden *stty tostop* (Suspender todos los trabajos en background antes de que produzcan salida):

Muchos de los trabajos en 2º plano están ejecutándose durante mucho tiempo y al finalizar muestran el resultado por pantalla.

Puede ser interesante que la fase de proceso que sse ejecute en 2º plano se detenga automáticamente justo antes de mostrar su salida. Eso es lo que hace la orden:

- stty tostop

Una vez efectuada esta orden, esta forma e trabajar durará hasta que se desconecte del sistema.

Esta orden es muy util, ya wue pdremos comprobar con jobs de vez en cuando si un proceso en 2º plano se ha detenido, y reanulador en primer plano con fg para ver la salida en pantalla cuando se desee.

Orden time (Ejecutar un proceso mostrando información de los tiempos empleados en su ejecución):

La función de este comando consiste en determinar el tiempo necesario para que se ejecute un proceso dado.

- time [opciones] comandoshell

Opciones:

-v: Devuelve el tiempo real, de usuario y de sistema.
-p: Devuelve un informe completo del uso de los recursos del sistema.

La ejecución de la orden produce primero la salida por pantalla del comando y después nos indica información de los tiempos en segundos de la misma:

$ time ps -f

UID	PID	PPID	C	STIME	TTY	TIME	COMMAND
grp1	34	1	0	15:57:07	02	0:01	-sh
grp1	282	34	80	17:16:16	02	0:05	ps -f

real 0.5 user 0.2 sys 0.1

real (elapsed) Tiempo total de ejecución desde que se teclea el comando y se pulsa ENTER hasta que vuelve a aparecer el prompt del Sell. Esto incluye el tiempo de E/S, de espera a otros usuarios, etc. EL tiempo real puede ser bastante mayor que el tiempo total de la CPU.

user Tiempo dedicado por la CPU para ejecutar el código de la orden del usuario.

sys Es el tiempo tarda se tarda en ejecutar las rutinas del núcleo de LINUX que dan servicio a la orden.

El tiempo total de CPU es la suma del de *user* y *sistema* y siempre es menor que el *real*. Los tiempos de medición varían dependiendo de las condiciones de carga.

Esta es la información estándar, pero puede aparecer más dependiendo de cada sistema.

ORDENES DE PLANIFICACIÓN DE PROCESOS:

Orden sleep (Dormir):

Esta orden no hace nada durante un determinado periodo de tiempo en segundos. Es decir provoca un retardo en la ejecución.

sleep tiempo

Incluida en un guión (script) retardará este durante el tiempo especificado:

Ejemplo:

```
$ cat alarma sleep
3600   banner
REUNION

$ chmod u+x alarma
$ alarma &
```

Este ejemplo hace que se nos muestre un mensaje de aviso dentro de una hora.

Ejemplo:

```
$ (sleep 1800; who >> registro) &
```

Proporciona un registro del número de usuario en el sistema una vez cada media hora. Crea un proceso en modo subordinado que duerme (suspende la operación) durante 1.800 segundos; luego se despierta, ejecuta la orden who y coloca la salida en un archivo de nombre registro. Colocada en un Script puede servir para tener información sobre los usuarios del sistema cada cierto tiempo.

Orden at (Ejecución diferida):

El sistema LINUX proporciona un método para permitir la ejecución retardada de comandos. Suele utilizarse para enviar mensajes a una hora determinada o ejecutar una serie de procesos en un momento determinado, sin necesidad de estar identificado ante el sistema.

Las ordenes at y batch utilizan los servicios de un mecanismo de planificación que esta siempre corriendo dentro del sistema LINUX, se trata de la orden **cron** (por cronografo)

La orden **cron** es un ***Daemon*** (demonio) del sistema que se ejecuta al comienzo de la secuencia de arranque de todos los sistemas LINUX. Se despierta una vez cada minuto, examina un fichero de control para ver si hay algún trabajo que ejecutar en ese minuto y lo lanza. Si no hay trabajos planificados para ese minuto, *cron* vuelve a dormir hasta el siguiente minuto.

La orden cron se encuentra en el directorio /etc/cron y aunque es ejecutable no debe ser usada nunca ni por los usuarios e incluso ni por el superusuario (dos copias de *cron* corriendo en la máquina hará que esta sufra un desorden considerable). Puede que alguna versión del S.O LINUX disponen de programas de temporización con un nombre diferente a *cron*.

:

Los formatos de la orden at es el siguiente:

- at tiempo [fecha] [incremento]
- at tiempo [fecha] [incremento] < guión

- at –l Listado de todas las ordenes at pendientes de ejecutar
- at –r nº_trabajo... Eliminar tarea.
- at –m Enviar mensaje corto de confirmación al terminar tarea.

1.- Con el primer formato las ordenes se escriben directamente hasta pulsar CTRL-D

$ at 23
ventas
resumen
ordenar
CTRL-D
$

En este ejemplo se usa el comando at para aplazar la ejecución de los programas ventas, resumen y ordenar a las 23 horas (11 de la noche).

2.- Con el segundo formato las ordenes se escriben previamente en un guion:

$ cat > misordenes
ventas
resumen
ordenar

CTRL-D
$ chmod u+x misordenes
$ at 23 < misordenes

Las opciones de tiempo, fecha e incremento son

- Especificando solo una hora:

  hh[:]mm[am|pm] [today]

$ at 3	*3 de la mañana.*
$ at 17	*5 de la tarde.*
$ at 0704	*7 y 04 de la mañana.*
$ at 7:04pm	*7 y 04 de la tarde.*
$ at 2200 today	*A las 10 de la noche hoy.*

  La hora se asume en formato de 24 horas a no ser que se especifique am o pm.

- Especificando fecha y hora:

  hh[:]mm[am|pm] [mes] [dia][, año] [tomorrow]

$ at 2pm Sat	*Sabado a las 2 de la mañana*
$ at 19:23 Aug 24	*24 de Agosto a las 19 y 23*
$ at 7:04pm tomorrow	*Mañana a las 7 y 04 de la tarde.*
$ at 2:15pm Jul 16, 2020	*16 de Julio del 2020 a las 14: 15 (Con suerte).*

- Especificando los caracteres especiales de mediodia y media noche:

  [noon][midnight]

$ at noon	A mediodia.
$ at noon Wed	El miercoles a mediodia.
$ at midnight	A medianoche.

- Especificando un desfase respecto a una fecha u hora:

  [now]

  Se pueden usar los especificadores:

  $ at now + 5 min
  $ at now + 5 minutes
  $ at now + 2 days
  $at now + 6 months

- Existen combinaciones híbridas además de los formatos especificados, por ejemplo:

  $ at 2:15pm May 13, 2003 + 30 years Ejecutará el trabajo 30 años después del dia 5 de mayo del 2003 a las 14 y 30(Aviso de pagar la hipoteca).

Cuando se ordena un trabajo con at, se recibe una información como la que sigue:

$ at 11.45 < miscosas job 550414980.a-904:0 at Thu May 13 11:45:00 2003.
$

El nº de trabajo 550414980.a es la referencia del mismo (El número representa un valor en segundos denominado tiempo mágico).

Listado de Ordenes pendientes creadas con at:

- at -l

  Muestra en pantalla los trabajos pendientes, solo muestra la identificación del mismo y la fecha cuando se ejecutará, pero no muestra el comando LINUX que se ejecutará.

  $ at -l *job 550414980.a-904:0 at Thu May 13 11:45:00 2003.*
  $

  Los trabajos quedan registrados en un archivo propiedad del administrador llamado /usr/spool/cron/atjobs.

Eliminar de Ordenes pendientes creadas con at:

- at -r [número_de_proceso]

  Elimina de la cola de procesos pendientes al nº referenciado:

  $ at -l
  job550414980.a-904:0 at Thu May 13 11:45:00 2003.
  $ at –r 550414980.a-904:0

Debido al peligro que puede entrañar este comando, el administrador puede permitir o prohibir su utilización a nivel de usuario individual. Para ello dispone de dos archivos:

- ***at.allow*** contentiene los nombres de los usuarios que tienen permitido el uso del comando *at*, uno en cada línea.
- ***at.deny*** en el que figurarán los nombres de aquellos a los que se les deniega el permiso.

En caso de que sólo exista el archivo *at.deny* , todos los usuarios podrán utilizar el comando at excepto los referenciados en *at.deny*.

Cuando no exista ninguno de estos archivos, tan sólo el administrador tiene permitido el uso del comando at.

Si existen los dos archivos, se ignora el at.*deny.*

Si el sistema está parado el día y hora que se tenía que ejecutar el trabajo, el trabajo at será ejecutado en cuanto arranque el sistema,