

En esta práctica veremos algunos mandatos para la gestión de procesos. Aprenderemos a seguir el estado de ejecución de los procesos, a detener un proceso o a matarlo. También veremos cómo ejecutar un proceso en segundo plano y cómo intercambiarlo entre primero y segundo plano. Por último, aprenderemos a programar tareas con el mandato “cron”.

Durante el desarrollo de esta práctica elabora un informe en el que indiques los pasos dados en su realización, y contesta a las preguntas que se te van haciendo en el guión. Añade los pantallazos pertinentes para seguir la práctica. Entrega este informe a través del aula virtual de la asignatura. !

Guión de la práctica:

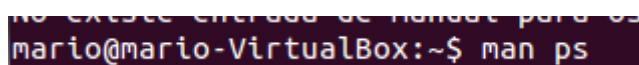
Una de las tareas más importantes del SO es la gestión de los procesos que se están ejecutando en una máquina. El hecho de que todos los procesos deban compartir los recursos hardware disponibles (memoria RAM, CPU) hace que el SO juegue un papel primordial en gestionar esos recursos para que los procesos se ejecuten de forma concurrente. Un proceso es la entidad que crea el SO para la ejecución de un programa. Cada proceso, durante su ejecución, guarda información sobre su “contexto” que incluye, entre otras cosas, información sobre su proceso “padre” (quién lo creó o invocó), los recursos del sistema que se están consumiendo

(segmentos de memoria asignados), permisos de seguridad, etc.

1. Visita el enlace http://structio.sourceforge.net/guias/AA_Linux_colegio/procesos-y-tareas.html y lee el punto 3.1 del mismo.

2. Inicia la máquina virtual Ubuntu.

3. Lee la página del manual sobre el mandato “ps”.



```
mario@mario-VirtualBox:~$ man ps
```

4. Visualiza los procesos que se están ejecutando en este momento. El primer valor que aparece es el identificador de proceso (PID). El segundo es la terminal que está asociada a ese proceso. Después también podemos observar el tiempo acumulado de uso de CPU, y finalmente el nombre del programa que ha dado lugar a este proceso. Apunta los procesos activos y sus valores.

```
mario@mario-VirtualBox:~$ ps
  PID TTY          TIME CMD
 2176 pts/1        00:00:00 bash
 2281 pts/1        00:00:00 ps
```

5. Por defecto, ps sólo ha mostrado los procesos asociados con la misma terminal e iniciados por el mismo usuario. Vamos a recuperar las opciones que nos permiten mostrar todos los procesos. Ejecuta el mandato “ps -e”. Comprueba la lista de procesos que están corriendo en tu máquina. ¿Cuál lleva el PID igual a 1?

```
mario@mario-VirtualBox:~$ ps -e
  PID TTY          TIME CMD
    1 ?           00:00:00 init
    2 ?           00:00:00 kthreadd
    3 ?           00:00:00 ksoftirqd/0
    4 ?           00:00:00 kworker/0:0
    5 ?           00:00:00 kworker/u:0
  LibreOffice Calc
    6 ?           00:00:00 migration/0
    7 ?           00:00:00 watchdog/0
    8 ?           00:00:00 cpuset
    9 ?           00:00:00 khelper
   10 ?           00:00:00 kdevtmpfs
   11 ?           00:00:00 netns
   12 ?           00:00:00 sync_supers
   13 ?           00:00:00 bdi-default
   14 ?           00:00:00 kintegrityd
   15 ?           00:00:00 kblockd
```

6. Recuerda lo que sucedía al pulsar “Ctrl + Alt + F1” hasta “Ctrl + Alt + F6”; comprueba el nombre de cada una de esas terminales. Vuelve a la terminal en la que se encuentra el entorno gráfico (Ctrl + Alt + F7). ¿Qué procesos se están ejecutando en las terminales tty1, tty2...tty6? ¿Qué proceso está corriendo sobre tty7?

Todos los terminales anteriores

7. El mandato “ps” todavía nos puede ofrecer más información sobre los procesos en ejecución. Por ejemplo, ¿quién ha iniciado cada uno de los procesos en nuestra máquina? Vamos a usar las siguientes opciones del mandato ps: “a” nos permite conocer todos los procesos que tienen una terminal asociada; “x” aquellos que no tienen terminal; la opción “u” nos muestra la salida en un formato más legible. (ps aux)

```
2010 ps/1 00:00:00 ps
mario@mario-VirtualBox:~$ ps aux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.1	0.0	3512	1956	?	Ss	17:27	0:00	/sbin/init
root	2	0.0	0.0	0	0	?	S	17:27	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	S	17:27	0:00	[ksoftirqd/0]
root	4	0.0	0.0	0	0	?	S	17:27	0:00	[kworker/0:0]
root	5	0.1	0.0	0	0	?	S	17:27	0:00	[kworker/u:0]
root	6	0.0	0.0	0	0	?	S	17:27	0:00	[migration/0]
root	7	0.0	0.0	0	0	?	S	17:27	0:00	[watchdog/0]
root	8	0.0	0.0	0	0	?	S<	17:27	0:00	[cpuset]
root	9	0.0	0.0	0	0	?	S<	17:27	0:00	[khelper]
root	10	0.0	0.0	0	0	?	S	17:27	0:00	[kdevtmpfs]
root	11	0.0	0.0	0	0	?	S<	17:27	0:00	[netns]
root	12	0.0	0.0	0	0	?	S	17:27	0:00	[sync_supers]
root	13	0.0	0.0	0	0	?	S	17:27	0:00	[bdi-default]
root	14	0.0	0.0	0	0	?	S<	17:27	0:00	[kintegrityd]
root	15	0.0	0.0	0	0	?	S<	17:27	0:00	[kblockd]
root	16	0.0	0.0	0	0	?	S<	17:27	0:00	[ata_sff]
root	17	0.0	0.0	0	0	?	S	17:27	0:00	[khubd]
root	18	0.0	0.0	0	0	?	S<	17:27	0:00	[md]

9. Todos los mandatos y opciones que hemos visto hasta ahora ofrecían información estática sobre los procesos. Esta información se extrae del directorio “/proc” del sistema. Hay algunas aplicaciones que también nos permiten conocer en tiempo real las características de cada proceso. Por ejemplo, el mandato “top”.

10. Lee la página del manual sobre el mandato “top”. ¿Qué hace el mandato “top”? Ejecútalo. (top) Como puedes observar, la información sobre el sistema se refresca cada 3 segundos (se puede modificar ese parámetro). Por lo demás, la interfaz de usuario de “top” no es especialmente agradable, aunque sí resulta sencillo modificar ciertas opciones y ajustarla a nuestros requisitos. Pulsa “q” para salir de top.

```
top - 17:31:47 up 4 min,  3 users,  load average: 0.35, 0.33, 0.15
Tasks: 155 total,  1 running, 154 sleeping,  0 stopped,  0 zombie
Cpu(s):  6.7%us,  2.6%sy,  3.1%ni, 76.8%id, 10.8%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   2061540k total,  701196k used, 1360344k free,   48992k buffers
Swap:  2095100k total,    0k used,  2095100k free,  334056k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	3512	1956	1304	S	0.0	0.1	0:00.37	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:00.01	ksoftirqd/0
4	root	20	0	0	0	0	S	0.0	0.0	0:00.07	kworker/0:0
5	root	20	0	0	0	0	S	0.0	0.0	0:00.35	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	0:00.00	watchdog/0
8	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	cpuset
9	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	khelper
10	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kdevtmpfs
11	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	netns
12	root	20	0	0	0	0	S	0.0	0.0	0:00.00	sync_supers
13	root	20	0	0	0	0	S	0.0	0.0	0:00.00	bdi-default
14	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kintegrityd
15	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kblockd
16	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	ata_sff
17	root	20	0	0	0	0	S	0.0	0.0	0:00.07	khubd

11. Ejecuta ahora el mandato “top -u alumno”. ¿Cómo ha cambiado la salida del mandato? ¿A quién pertenecen los procesos que observas ahora?

```
mario@mario-VirtualBox:~$ top -u mario
```

```
top - 17:32:42 up 5 min,  3 users,  load average: 1.19, 0.57, 0.24
Tasks: 154 total,  1 running, 153 sleeping,  0 stopped,  0 zombie
Cpu(s):  4.8%us,  1.4%sy,  7.9%ni, 79.0%id,  6.9%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   2061540k total,  713064k used, 1348476k free,   49204k buffers
Swap:  2095100k total,    0k used,  2095100k free,  334812k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2826	mario	20	0	90768	15m	11m	S	1.3	0.8	0:00.30	gnome-terminal
1890	mario	20	0	148m	23m	16m	S	1.0	1.2	0:00.99	nautilus
1705	mario	9	-11	98.6m	5588	4100	S	0.7	0.3	0:00.62	pulseaudio
1500	mario	20	0	19072	1224	840	S	0.3	0.1	0:00.69	VBoxClient
1505	mario	20	0	6520	2772	620	S	0.3	0.1	0:00.90	dbus-daemon
1892	mario	20	0	52208	9.8m	8036	S	0.3	0.5	0:01.07	bamfdaemon
2312	mario	30	10	157m	56m	37m	S	0.3	2.8	0:06.01	update-manager
1429	mario	20	0	56424	4164	3540	S	0.0	0.2	0:00.02	gnome-keyring-d
1440	mario	20	0	52552	9972	8012	S	0.0	0.5	0:00.12	gnome-session
1479	mario	20	0	16872	272	0	S	0.0	0.0	0:00.00	VBoxClient
1480	mario	20	0	18568	1708	1264	S	0.0	0.1	0:00.00	VBoxClient
1489	mario	20	0	16872	268	0	S	0.0	0.0	0:00.00	VBoxClient

16. Vuelve a tu directorio personal (cd \$HOME, cd ~, cd /home/alumno, ...) Veremos ahora algunos atajos de teclado que nos permiten gestionar procesos.

```
file [--help]
mario@mario-VirtualBox:~$ cd ~
mario@mario-VirtualBox:~$
```

17. Lee la página del manual sobre el comando “yes”. Aunque el mismo pueda no parecer de gran utilidad, a nosotros nos va a servir para comprobar cómo podemos detener y “matar” procesos.

```
mario@mario-VirtualBox:~$ man yes
```

18. Ejecuta el mandato “yes hola”. Observa que el mensaje aparece indefinidamente. Vamos a “matar” esta tarea. Intenta salir de la tarea con “q”. La tecla “q” (quit, salir) nos permite salir de ciertas aplicaciones en ejecución, pero no acabar con una tarea. Teclea el atajo “Ctrl + C”. El mismo debería terminar con el proceso activo. ¿Qué ha sucedido? El atajo de teclado “Ctrl + C” se encarga de terminar (o matar) una tarea. El atajo “Ctrl + Z” se encarga únicamente de detenerla (aunque el proceso siga “vivo” y se pueda retomar en el estado en que se detuvo).

[illegible]

19. Podemos ahora redirigir la salida del mandato a un fichero (observa que esto podría darnos serios problemas de memoria en nuestra máquina). Para poder redirigir la salida de mandatos a un fichero sin peligro de que eso colapse nuestra memoria, Linux dispone de un fichero cuya localización es `"/dev/null"`. Busca en la Wikipedia `"/dev/null"` para comprender mejor qué es y ejecuta el mandato: (yes adios `> /dev/null`)

20. Como puedes observar, la tarea en ejecución no permite seguir utilizando la terminal. Teclea “Ctrl + C” para detenerlo. ¿Qué tamaño ocupa ahora en disco el fichero /dev/null? ¿Dónde ha ido a parar toda la información que hemos enviado? (stat /dev/null)

21. Ejecutamos de nuevo el mandato “yes”. (yes ‘que tal’ > /dev/null. Abre una nueva terminal y localiza el PID del proceso “yes” iniciado.

```
mario 3209 0.1 0.1 8512 3576 pts/1 Ss 17:36 0:00 bash
mario 3273 88.0 0.0 5408 280 pts/1 R+ 17:37 0:13 yes 'que tal'
```

22. Lee la página del manual sobre el mandato “kill”. Como puedes observar, kill nos permite mandar señales a un proceso. El tipo de señales que permite mandar lo puedes encontrar, por ejemplo, en http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_12_01.html Lee las Secciones 12.1.1.3 y 12.1.2 del anterior enlace. Apunta algunas de las señales más comunes que se pueden enviar a procesos en ejecución y la forma de hacerlo (por teclado, o por medio de kill). Anota la diferencia entre SIGKILL y SIGTERM.

23. Interrumpe el proceso activo “yes” por medio del mandato “kill”

```
Creación: -
mario@mario-VirtualBox:~$ yes que tal > /dev/null
^C
mario@mario-VirtualBox:~$ yes 'que tal' > /dev/null
^C
mario@mario-VirtualBox:~$
mario@mario-VirtualBox:~$ ps aux
mario 2244 0.0 0.4 91352 9216 ? S 17:28 0:00 /usr/lib/notify
mario 2250 0.0 0.4 42752 8860 ? Sl 17:28 0:00 gnome-screensav
mario 2277 0.0 0.5 44584 11948 ? Sl 17:28 0:00 /usr/lib/notify
mario 2282 0.0 0.6 61728 12512 ? Sl 17:29 0:00 update-notifier
root 2305 0.0 0.4 18056 9300 ? S 17:29 0:00 /usr/bin/python
mario 2312 1.1 2.8 160776 58008 ? SNL 17:29 0:06 /usr/bin/python
mario 2355 0.0 0.2 35692 4124 ? Sl 17:30 0:00 /usr/lib/deja-d
mario 2484 0.0 0.3 11196 6244 tty2 S+ 17:30 0:00 -bash
mario 2708 0.0 0.3 11196 6240 tty6 S+ 17:30 0:00 -bash
root 2902 0.0 0.0 0 0 ? S 17:32 0:00 [worker/0:1]
mario 3023 0.8 1.9 74476 40352 ? Sl 17:36 0:00 /usr/bin/python
lp 3092 0.0 0.0 6208 1376 ? S 17:36 0:00 /usr/lib/cups/n
mario 3159 0.6 0.8 92280 17108 ? RL 17:36 0:00 gnome-terminal
mario 3204 0.0 0.0 2384 752 ? S 17:36 0:00 gnome-pty-helpe
mario 3209 0.1 0.1 8512 3576 pts/1 Ss 17:36 0:00 bash
mario 3273 88.0 0.0 5408 280 pts/1 R+ 17:37 0:13 yes 'que tal'
root 3274 0.0 0.0 0 0 ? S 17:37 0:00 [worker/0:2]
mario 3297 2.1 0.1 8512 3484 pts/2 Ss 17:37 0:00 bash
mario 3355 0.0 0.0 6148 1156 pts/2 R+ 17:37 0:00 ps aux
mario@mario-VirtualBox:~$ kill 3273
mario@mario-VirtualBox:~$
```

24. En nuestra terminal original vuelve a ejecutar el proceso “yes que tal > /dev/null” y, desde la segunda terminal que hemos abierto, envíale ahora una señal de SIGKILL. Comprueba que el resultado externo ha sido el mismo que antes.

No me encuentra la orden

25. todos los procesos que ejecutamos en una terminal deben ejecutarse en primer plano (bloqueando así la terminal). También podemos hacer lo que se conoce como ejecución en segundo plano. Puedes leer en UD8 información acerca de las principales diferencias entre ejecutar un proceso en primero o segundo plano (esencialmente tiene que ver con la prioridad del mismo). La forma de hacer que un programa se ejecute en segundo plano es escribiendo el programa en el intérprete de mandatos seguido de un símbolo “&”. Comprueba, en nuestra terminal original, el siguiente mandato:(yes otra vez > /dev/null &)

```
mario@mario-VirtualBox:~$ yes otra vez > /dev/null &
[1] 3366
```


26. Vamos a hacer ahora uso del mandato “jobs”. Comprueba en primer lugar su función por medio de “help jobs”, y el significado de la opción “-l”. La diferencia entre una “tarea” (job) y un proceso (process) es que los “jobs” son obligatoriamente iniciados desde una terminal y están asociados a ella (son procesos “hijos” de la terminal).

```
mario@mario-VirtualBox:~$ help jobs
jobs: jobs [-lnprs] [idtrabajo ...] o jobs -x orden [args]
Muestra el estado de los trabajos.

Muestra los trabajos activos. IDTRABAJO restringe la salida a
ese trabajo. Sin opciones, se muestra el estado de todos los trabajos
activos.

Opciones:
-l muestra los id's de los procesos, además de
  la información normal
-n solo muestra los procesos que han cambia de estado desde
  la última notificación
-p solo muestra los id's de los procesos
-r restringe la salida a los trabajos en ejecución
-s restringe la salida a los trabajos detenidos

Si se especifica -x, la ORDEN se ejecuta después de que todas las
especificaciones de trabajo que aparecen en ARGS se han reemplazado
```

27. Ejecuta el mandato “jobs” en la misma terminal en la que has ejecutado “yes”. ¿En qué estado se encuentra el proceso? Compruébalo también con los mandatos “ps” y “top”. Anota el porcentaje de CPU que consume.

```
mario@mario-VirtualBox:~$ jobs
[1]+  Ejecutando yes otra vez > /dev/null &
```

28. Entre los procesos de Linux siempre existe una jerarquía definida, ya que cada proceso debe tener un proceso padre (excepto el proceso de inicio, llamado init). Esta jerarquía adquiere relevancia ya que “matar” a un proceso padre por lo general conlleva acabar también con los procesos hijos. En algunos casos, un proceso padre y sus hijos pueden incluso compartir memoria. Comprueba la jerarquía de procesos en tu máquina por medio del mandato pstree (puedes ver alguna de sus opciones en man pstree). Comprueba sus ancestros. Apunta en tu informe de qué procesos desciende “yes”.

```
mario@mario-VirtualBox:~$ pstree
init--NetworkManager--dhclient
                        |
                        +--dnsmasq
                        |
                        +--2*[{NetworkManager}]
--2*[VBoxClient]--VBoxClient--{VBoxClient}
--VBoxClient--VBoxClient
--VBoxClient--VBoxClient--2*[{VBoxClient}]
--VBoxService--7*[{VBoxService}]
--accounts-daemon--{accounts-daemon}
--acpid
--atd
--avahi-daemon--avahi-daemon
--bamfdaemon--2*[{bamfdaemon}]
--bluetoothd
--colord--2*[{colord}]
--console-kit-dae--64*[{console-kit-dae}]
--cron
--cupsd--dbus
--2*[dbus-daemon]
--dbus-launch
```


29. Vuelve a ejecutar “yes” en la misma terminal y también en segundo plano. (yes mensaje > /dev/null &) Vuelve a comprobar el árbol de procesos por medio de “pstree -h”. Usando el mandato “top”, anota en tu informe el porcentaje de CPU (aproximado) que suman estos dos procesos yes.

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3376	mario	20	0	5408	284	228	R	49.5	0.0	0:10.30	yes
3366	mario	20	0	5408	280	228	R	49.2	0.0	2:10.95	yes
913	root	20	0	132m	86m	15m	S	0.3	4.3	0:15.42	Xorg
930	root	20	0	26136	1332	972	S	0.3	0.1	0:00.23	VBoxService
1739	mario	20	0	271m	65m	35m	S	0.3	3.3	0:05.77	unity-2d-shell
3159	mario	20	0	92424	16m	11m	S	0.3	0.8	0:01.19	gnome-terminal

30. Vuelve a comprobar el estado de los procesos iniciados en esta shell por medio de “jobs”. Comprueba que aparecen las dos tareas iniciadas y que en la segunda aparece el símbolo + indicando que es la última que se ha ejecutado. Apunta el estado de ambas en tu informe.

```
mario@mario-VirtualBox:~$ jobs
[1]-  Ejecutando                yes otra vez > /dev/null &
[2]+  Ejecutando                yes mensaje > /dev/null &
```

32. A través de “jobs”, cada tarea que se está ejecutando desde nuestra terminal recibe un nuevo número (1, 2, ...). Este número aparece entre corchetes. Mata la segunda tarea iniciada (la de mayor PID). Por ejemplo, puedes ejecutar top y capturar su PID y enviarle una señal de kill (en lugar del PID también puedes usar el símbolo “%” seguido del número de tarea). Comprueba por medio de jobs que sólo queda una tarea activa.

```
mario@mario-VirtualBox:~$ jobs
[1]-  Ejecutando                yes ot
[2]+  Ejecutando                yes me
mario@mario-VirtualBox:~$ kill %2
mario@mario-VirtualBox:~$ jobs
```

```
mario@mario-VirtualBox:~$ jobs
[1]+  Ejecutando                yes otra vez > /dev/null &
mario@mario-VirtualBox:~$
```

33. Ejecuta el mandato “yes mensaje2 > /dev/null”. Por medio del teclado (Ctrl + Z), o por medio de kill (con la señal SIGSTOP ó 19 y con el PID correspondiente) envía al proceso una señal de “detenido”. Comprueba que el proceso está detenido por medio de jobs.

```
{zeitgeist-fts}
mario@mario-VirtualBox:~$ ps
  PID TTY          TIME CMD
 3209 pts/1        00:00:00 bash
 3366 pts/1        00:05:09 yes
 3383 pts/1        00:00:01 yes
 3386 pts/1        00:00:00 ps
mario@mario-VirtualBox:~$ kill 2283
bash: kill: (2283) - No existe el proceso
mario@mario-VirtualBox:~$ kill 3383
mario@mario-VirtualBox:~$
```

34. Los números de tarea pueden ser usados por el mandato “fg” (foreground) para traer dichas tareas a primer plano (fg 1, fg 2, ...), o por el mandato “bg” para mandarlas a segundo plano (bg 1, bg 2, ...). Para recuperar una tarea detenida sólo tienes que ejecutar fg (foreground) o bg (background), dependiendo de que quieras que la tarea se ejecute en primer o segundo plano. Comprueba con fg que la tarea vuelve a primer plano.

```
mario@mario-VirtualBox:~$ fg
yes mensaje2 > /dev/null
Terminado
mario@mario-VirtualBox:~$
```

38.Linux también dispone de utilidades para la programación de tareas; las tareas programadas son procesos que se ejecutarán (siempre y cuando la máquina esté encendida) de forma planificada. El programa que nos permite programar tareas desde línea de mandatos se llama “cron”. Puedes leer en la UD8 información sobre el cron y crontab.

39.Comprueba que “cron” está activo en tu ordenador (puedes observar si aparece en pstree). El proceso “cron” se debe encontrar siempre en ejecución, para que a la hora y día que tenga programada alguna tarea pueda ejecutar la misma. También te puedes asegurar de que el mismo está en marcha por medio del mandato: (sudo service cron start) (También con sudo start cron)

```
Terminado
mario@mario-VirtualBox:~$ sudo service cron start
[sudo] password for mario:
start: Job is already running: cron
mario@mario-VirtualBox:~$
```

40.Existen diversas formas de programar nuevas tareas. El fichero en el que se encuentran las tareas programadas se llama “/etc/crontab”. Lee la página del manual sobre el fichero “/etc/crontab”. (man 5 crontab)

```
mario@mario-VirtualBox:~$ man crontab
```

41.El fichero “conrab” es un fichero de texto, así que puedes editarlo y modificarlo, por ejemplo, con nano (pero deberás disponer de permisos de superusuario). Ejecuta: (sudo nano /etc/crontab)Por defecto deberían aparecer varias tareas programadas del sistema. La estructura de cada una de las líneas es la siguiente:

minuto(s) - hora - día del mes – mes - día de la semana – usuario - mandato De esta forma, la siguiente línea: 5,20,35,50 * * * * alumno cd /home/alumno/Escritorio;wget http://www.larioja.com significará que a los minutos 5, 20, 35, 50 de todas las horas (*), de todos los días del mes (*), de todos los meses y de todos los días de la semana (*), el usuario “alumno” ejecutará los mandatos “cd /home/alumno/Escritorio; wget http://www.larioja.com”. Inserta la línea anterior en la última línea de tu fichero crontab, pero de forma que la acción se ejecute cada cinco minutos. (Escribe */5)

```
mario@mario-VirtualBox: ~
GNU nano 2.2.6 Archivo: /etc/crontab Modificado

# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab`
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 * * * * root    cd / && run-parts --report /etc/cron.hourly
25 6 * * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --repo$
47 6 * * 7 root    test -x /usr/sbin/anacron || ( cd / && run-parts --repo$
52 6 1 * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --repo$
5,20,35,50 * * * * alumno cd /home/alumno/Escritorio || wget http://www.larioja$
#

^G Ver ayuda ^O Guardar ^R Leer Fich ^Y RePág. ^K Cortar Tex ^C Pos actual
^X Salir ^J Justificar ^W Buscar ^V Pág. Sig. ^U PegarTxt ^T Ortografía
```

43. ¿Cuál es la diferencia entre el comando AT y utilizar cron/crontab?

La diferencia entre AT y Cron es que el primero no es persistente, por lo que si reiniciamos la PC se perderá la tarea que le encomendamos. ¿Cómo funciona AT? Pues muy sencillo, la forma básica sería escribir en el terminal