

Sumario

1. Procesos y servicio.....	2
1.1. Proceso.....	2
1.2. Servicio.....	2
2. Procesos en Linux, estados y prioridades.....	2
a) El identificador de proceso en Linux PID.....	2
3. Multitarea.....	3
4. Procesos y recepción de Señales.....	3
5. Estados de los procesos.....	4
6. Control de procesos.....	5
6.1. Ver los procesos en ejecución.....	5
6.2. Algunas opciones:.....	5
6.3. top y htop.....	6
6.4. Ejecución en segundo plano.....	6
6.5. jobs.....	7
7. Servicios, arranque, parada, reinicio y estado.....	8
8. PROGRAMAR TAREAS EN LINUX.....	9
8.1. Cron.....	9
8.2. Crontab.....	9
8.3. Ver las tareas programadas.....	9
8.4. Editar las tareas programadas.....	10
8.5. Selección tiempo.....	10
a) Ejemplo 1.....	11
b) Ejemplo 2.....	11
c) Ejemplo 3.....	11
d) Ejemplo4.....	11
8.6. Palabras reservadas.....	11
a) Ejemplo 5.....	12
8.7. Crear un script y dar permisos de ejecución.....	12
8.8. Añadir una entrada.....	12
8.9. Mostra todas las entradas.....	12

1. Procesos y servicio

1.1. Proceso.

Es un programa en ejecución, en **primer plano** es decir que el usuario puede interactuar con el desde la consola.

Ejemplo: libre Office, Chrome, ..

1.2. Servicio

Es un programa en ejecución, en **segundo plano**, el usuario no interactúa con el lo hace mediante otras aplicaciones o el propio sistema operativo si.

Nos ofrece nuevas funcionalidades.

En ocasiones también se usa el nombre de proceso para referirnos a un servicio.

Ejemplo: sería un servidor DHCP, un cliente DHCP, el programador de tareas Cron, Mysql, Oracle, ...

2. Procesos en Linux, estados y prioridades.

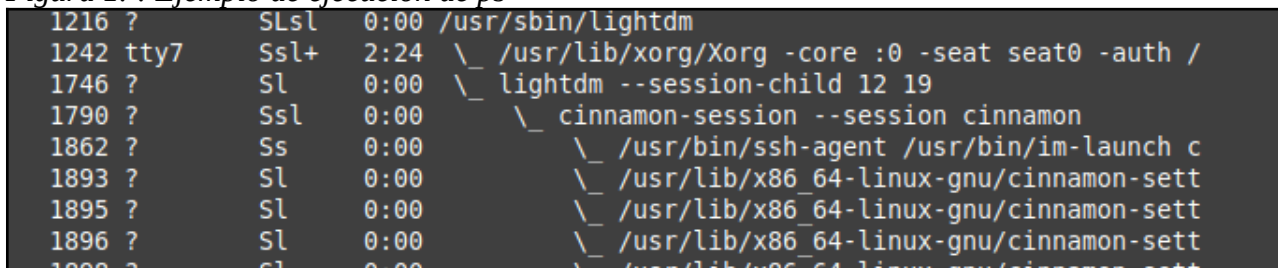
Los procesos en Linux, son creados mediante un mecanismo de clonación.

Un proceso en Linux genera un nuevo proceso para que realice una tarea determinada, y este nuevo proceso es considerado proceso «**hijo**» del proceso anterior, al que llamaremos «**padre**».

Esta estructura de procesos padres e hijos forman un árbol jerárquico de procesos, en los que podemos distinguir esta estructura.

El primer proceso desde el que se crean todos los demás es el proceso **init**.

Figura 1.º: Ejemplo de ejecución de ps



```
1216 ?      Ssl    0:00 /usr/sbin/lightdm
1242 tty7   Ssl+   2:24 \_ /usr/lib/xorg/Xorg -core :0 -seat seat0 -auth /
1746 ?      Sl     0:00 \_ lightdm --session-child 12 19
1790 ?      Ssl    0:00 \_ cinnamon-session --session cinnamon
1862 ?      Ss     0:00 \_ /usr/bin/ssh-agent /usr/bin/im-launch c
1893 ?      Sl     0:00 \_ /usr/lib/x86_64-linux-gnu/cinnamon-sett
1895 ?      Sl     0:00 \_ /usr/lib/x86_64-linux-gnu/cinnamon-sett
1896 ?      Sl     0:00 \_ /usr/lib/x86_64-linux-gnu/cinnamon-sett
1898 ?      Sl     0:00 \_ /usr/lib/x86_64-linux-gnu/cinnamon-sett
```

a) El identificador de proceso en Linux PID

El PID es un número que identifica de forma única a cada proceso.

Todos los PID se almacena en una tabla de procesos, hay una entrada por cada uno de los procesos que están en ejecución en el sistema.

Donde también se almacena, que usuario lo inicio, que tiempo lleva lanzado, que tiempo de uso de CPU tiene, cual es su proceso padre, entre otra información

Para consultar la tabla de procesos disponemos del comando **ps**.

3. Multitarea

El procesador solamente puede estar procesando un proceso a la vez en cada uno de sus núcleos, o «cores».

Pero los SS.OO. Actuales son multiusuario y multitarea.

¿Se pueden ejecutar simultáneamente (en un mismo momento) varios procesos en un solo núcleo? Es decir, si un usuario quiere navegar, copiar datos al penDrive y escuchar música. ¿Puede?

Pues si, el **kernel del sistema operativo** le da dando tiempo de procesamiento alternativamente a todos los procesos (le da un poco a un proceso, se lo retira y se lo da a otro, así mientras queden proceso que quieran CPU), este intercambio se produce de una forma muy rápida, haciendo que el usuario tenga la percepción que todos los procesos se ejecutan a la al mismo tiempo.

4. Procesos y recepción de Señales

Todo proceso que se ejecuta, está preparado para recibir una señales para comunicarnos con el y este (el proceso) actuar en consecuencia.

El comando para enviar señales a los procesos es **kill**, hay que indicar a que proceso se envía la señal usando su PID y que señal se envía.

Tabla 1: Algunos de las señales que se pueden enviar son:

Número de señal	Nombre de señal	Descripción
1	SIGHUP	Hangup. Se indica al programa que debe de salir. Pero es reprogramable para poder usarse en otros comportamientos, como <i>reload</i> .
9	SIGKILL	Kill. Le dice al sistema operativo que debe de terminar de forma forzosa con ese proceso.
15	SIGTERM	Terminación. Es reprogramable para poder realizar tareas específicas antes de salir.
19	SIGSTOP	Stop. Detiene el proceso, pero sin terminarlo, de modo que se pueda volver a reanudar su ejecución.
18	SIGCONT	Continue. Le dice a un proceso parado que continúe su ejecución.

a) Ejemplo:

Ejecutamos un editor en segundo plano **xed &** y posteriormente enviamos la señal de terminar de inmediato el proceso con `kill -9 8112`, usando la señal `-9`

Figura 2.º: Ejemplo envío señal de terminar -9

```
El valiente vive, hasta que el cobarde quiere.  
ricardo@AH512:~$ xed &  
[1] 8112  
ricardo@AH512:~$ kill -9 8112  
ricardo@AH512:~$
```

5. Estados de los procesos

Los procesos pueden estar en diferentes estados, estos son:

Tabla 2: Los procesos pueden estar en diferentes estados, estos son:

Id	Estado	Descripción
S	Sleep	Durmiendo. Quiere decir que está en ejecución, pero en ese momento no se encuentra ejecutándose ningún código dentro de la CPU.
D	Sleep	Es igual que el anterior, pero no es posible interrumpirlo.
T	Stopped	Parado. Quiere decir que se ha detenido su ejecución.
R	Running	En ejecución. Es un proceso que se está ejecutando de forma activa en la CPU.
Z	Zombie	Es un proceso que debería de haber <i>muerto</i> , pero aún tiene dependencias que no es posible terminar. Hasta que no se eliminen sus dependencias no desaparecerá.

5.1. Como lanzar un programa en segundo plano.

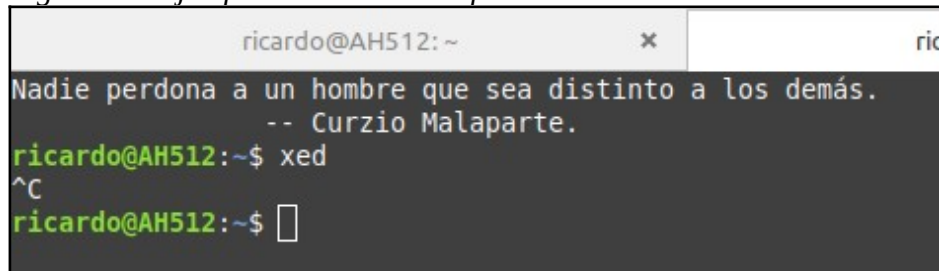
Para lanzar un proceso directamente en segundo plano hay que colocar un **&** al final del mismo.

Como se puede ver en la **Figura 2**

5.2. Cambiar de plano de un proceso

Un proceso que está en ejecución en primer plano, puede ser **detenido mediante** la pulsación de **^C (Ctrl+C)**, que es lo mismo que enviar al proceso activo la señal de SIGTERM (-15)

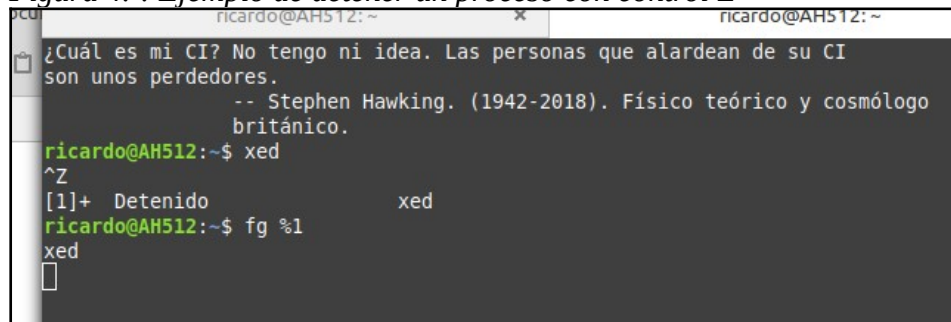
Figura 3.º: Ejemplo de terminar un proceso con control C



```
ricardo@AH512: ~  
Nadie perdona a un hombre que sea distinto a los demás.  
-- Curzio Malaparte.  
ricardo@AH512:~$ xed  
^C  
ricardo@AH512:~$
```

Si se quiere **detener** se usa la combinación de teclas **^Z (Ctrl+Z)**, se enviará la señal de SIGSTOP. Se puede volver a retomar el proceso ejecutar el comando **fg**.

Figura 4.º: Ejemplo de detener un proceso con control Z



```
ricardo@AH512: ~  
¿Cuál es mi CI? No tengo ni idea. Las personas que alardean de su CI  
son unos perdedores.  
-- Stephen Hawking. (1942-2018). Físico teórico y cosmólogo  
británico.  
ricardo@AH512:~$ xed  
^Z  
[1]+ Detenido xed  
ricardo@AH512:~$ fg %1  
xed  

```

6. Control de procesos

6.1. Ver los procesos en ejecución

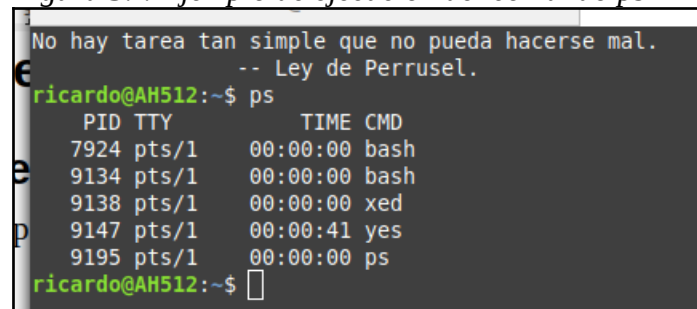
Existen varias herramientas para ver los procesos en ejecución, la más importante es el comando **ps**.

a) Comando **ps**

ps [opciones]

Ps lista los procesos con su PID, datos de usuario, tiempo, identificador del proceso y línea de comandos usada

Figura 5.º: Ejemplo de ejecución del comando **ps**



```
No hay tarea tan simple que no pueda hacerse mal.  
-- Ley de Perrusel.  
ricardo@AH512:~$ ps  
  PID TTY          TIME CMD  
 7924 pts/1    00:00:00 bash  
 9134 pts/1    00:00:00 bash  
 9138 pts/1    00:00:00 xed  
 9147 pts/1    00:00:41 yes  
 9195 pts/1    00:00:00 ps  
ricardo@AH512:~$
```

Sin opciones, sólo muestra los procesos lanzados desde el terminal actual y del usuario que lo lanzó.

6.2. Algunas opciones:

- -e o ax: muestra todos los procesos
- -u (o U o -user) usuario: muestra los procesos de un usuario
- u: salida en formato usuario
- j: salida en formato job (muestra PID, PPID, etc.)
- -f o l: salida en formato largo
- f: muestra un árbol con la jerarquía de procesos

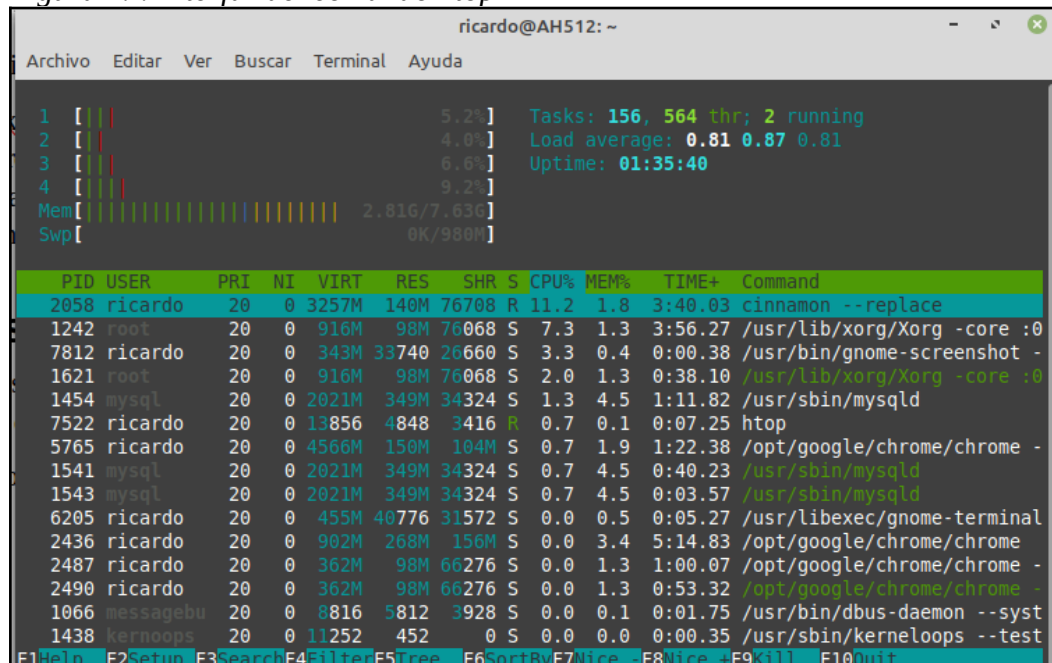
Figura 6.º: Ejemplo de ejecución del comando ps

```
ricardo@AH512:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
ricardo      7924    6205  0 00:37 pts/1        00:00:00 bash
ricardo      9134    7924  0 01:05 pts/1        00:00:00 bash
ricardo      9138    7924  0 01:05 pts/1        00:00:00 xed
ricardo      9147    9134  99 01:05 pts/1        00:06:10 yes
ricardo      9387    7924  0 01:11 pts/1        00:00:00 ps -f
ricardo@AH512:~$
```

6.3. top y htop

Muestra información de proceso en una consola de texto interactiva, además permite ejecutar comandos de una forma más sencilla. ¿?

Figura 7.º: Interfaz del comando htop



The screenshot shows the htop interface in a terminal window. At the top, there's a menu bar with options: Archivo, Editar, Ver, Buscar, Terminal, Ayuda. Below the menu, system statistics are displayed: Tasks: 156, 564 thr; 2 running; Load average: 0.81 0.87 0.81; Uptime: 01:35:40. Memory usage is shown as 2.81G/7.63G and swap as 0K/980M. The main part of the screen is a table of running processes. The table has columns: PID, USER, PRI, NI, VIRT, RES, SHR, S, CPU%, MEM%, TIME+, and Command. The processes listed include cinnamon, gnome-screenshot, xed, yes, htop, chrome, mysql, gnome-terminal, and dbus-daemon. At the bottom, there's a status bar with function key shortcuts: F1Help, F2Setup, F3Search, F4Filter, F5Tree, F6SortBy, F7Nice, F8Nice, F9Kill, F10Quit.

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
2058	ricardo	20	0	3257M	140M	76708	R	11.2	1.8	3:40.03	cinnamon --replace
1242	root	20	0	916M	98M	76068	S	7.3	1.3	3:56.27	/usr/lib/xorg/Xorg -core :0
7812	ricardo	20	0	343M	33740	26660	S	3.3	0.4	0:00.38	/usr/bin/gnome-screenshot -
1621	root	20	0	916M	98M	76068	S	2.0	1.3	0:38.10	/usr/lib/xorg/Xorg -core :0
1454	mysql	20	0	2021M	349M	34324	S	1.3	4.5	1:11.82	/usr/sbin/mysqld
7522	ricardo	20	0	13856	4848	3416	R	0.7	0.1	0:07.25	htop
5765	ricardo	20	0	4566M	150M	104M	S	0.7	1.9	1:22.38	/opt/google/chrome/chrome -
1541	mysql	20	0	2021M	349M	34324	S	0.7	4.5	0:40.23	/usr/sbin/mysqld
1543	mysql	20	0	2021M	349M	34324	S	0.7	4.5	0:03.57	/usr/sbin/mysqld
6205	ricardo	20	0	455M	40776	31572	S	0.0	0.5	0:05.27	/usr/libexec/gnome-terminal
2436	ricardo	20	0	902M	268M	156M	S	0.0	3.4	5:14.83	/opt/google/chrome/chrome -
2487	ricardo	20	0	362M	98M	66276	S	0.0	1.3	1:00.07	/opt/google/chrome/chrome -
2490	ricardo	20	0	362M	98M	66276	S	0.0	1.3	0:53.32	/opt/google/chrome/chrome -
1066	messagebu	20	0	8816	5812	3928	S	0.0	0.1	0:01.75	/usr/bin/dbus-daemon --syst
1438	kernoops	20	0	11252	452	0	S	0.0	0.0	0:00.35	/usr/sbin/kerneloops --test

6.4. Ejecución en segundo plano

Por defecto, los comandos corren en primer plano (**foreground**): el shell espera a que termine el comando antes de aceptar uno nuevo.

Para **ejecutar** un comando en **segundo plano** (background) hacerlo con **&**

\$ sleep 10

\$ sleep 10 &

Para **terminar** un proceso **en foreground** **Ctrl-C**

Para **pausar** un comando en **foreground** **Ctrl-Z**

Con **bg** pasa el proceso de **foreground** a **background**, pero antes debemos detenerlo.

Con **fg** lo devuelve a **foreground** un proceso que esta en **background**.

6.5. jobs

Permite ver la lista de comandos (jobs) en background lanzados desde el shell (terminal) **actual**, así como su estado (fg y bg pueden referirse a uno de los jobs).

a) Tareas sobre procesos

1. Queremos mostrar una ventana con tu nombre dentro de 10 minutos y que mientras no salga no podemos hacer nada.

```
sleep 600 ; zenity --info --text "hola"
```

2. Lanzar xed en segundo plano.

```
Xed &
```

3. Ahora queremos cerrar el programa xed (que se lanzo así xed & y sin pulsar la x)

```
kill -9 (pid o %n)
```

otra forma

```
fg %n  
control + z
```

4. Consola recién abierta y lanzar xed en primer plano.

```
Xed
```

5. Sigue el 4, ahora quiero detener el xed

```
control+z
```

otra forma

```
Abro una terminal nueva.  
Ejecuto ps -u  
Busco el pid de xed  
Envío señal de detener, kill -19 pid_de_xed
```

6. Sigue el 5, ahora quiero volver a poner en ejecución en primer plano.

```
Fg %1
```

otro forma mas larga (sin sentido)

```
fill -18 %2  
fg %2
```


Manejo de procesos y servicios

- 7. Sigue el 5, ahora quiero volver a poner en ejecución en segundo plano.**

```
bg %2
```

- 8. Lanzar el xed en una terminal y detenerlo desde otra terminal.**

En la primera terminal ejecutamos
xed

En la segunda terminal ejecutamos
ps
kill -19 9918 (9918 es el pid de xed)

- 9. Sigue el 8, queremos poner en marcha otra vez el xed.**

```
kill -18 9918
```

```
zenity --info --text "hola"
```

7. Servicios, arranque, parada, reinicio y estado.

- Para **iniciar** un servicio en Linux, necesitas ejecutar el siguiente comando:

- `sudo systemctl start [nombre_servicio]`
 - `sudo /etc/init.d/nombre_servicio start`

- Si quieres **detenerlo**, usa el siguiente comando:

- `sudo systemctl stop [nombre_servicio]`
 - `sudo /etc/init.d/nombre_servicio stop`

- Para verificar el estado de un servicio puedes usar:

- `sudo systemctl status [nombre_servicio]`
 - `sudo /etc/init.d/nombre_servicio status`

- Para parar y iniciar

- `sudo systemctl restart [nombre_servicio]`
 - `sudo /etc/init.d/nombre_servicio restart`

8. PROGRAMAR TAREAS EN LINUX

8.1. Cron

Es un demonio (proceso en segundo plano, servicio) que se ejecuta desde el mismo instante en el que arranca el sistema. Comprueba si existe alguna tarea (job) para ser ejecutado de acuerdo a la hora configurada en el sistema.

8.2. Crontab

Crontab es un simple archivo de texto que guarda una lista de comandos a ejecutar en un tiempo especificado por el usuario. Crontab verificará la fecha y hora en que se debe ejecutar el script o el comando, los permisos de ejecución y lo realizará en el background. Cada usuario puede tener su propio archivo crontab, de hecho el `/etc/crontab` se asume que es el archivo crontab del usuario root, cuando los usuarios normales (e incluso root) desean generar su propio archivo de crontab, entonces utilizaremos el comando `crontab`.

8.3. Ver las tareas programadas.

Para ver las tareas programadas del usuario actual ejecutamos **crontab -l**

8.4. Editar las tareas programadas.

Para editar las tareas programadas del usuario actual ejecutamos **crontab -e** y al final del fichero añadimos una entrada como la figura.

Los asteriscos indican la frecuencia de ejecución y al final de la línea se indica el comando a ejecutar (hay que incluir la ruta y asegurarse que tiene los permisos de ejecución).

Dibujo 1.º: Ejemplo de línea para ejecutar todos los minutos, de todos los días del año del programa script.sh

```
* * * * * /bin/ejecutar/script.sh
```

8.5. Selección tiempo.

Dibujo 2.º: Representación gráfica de las opciones de una entrada de crontab



De izquierda a derecha, los asteriscos representan:

- Minutos: de 0 a 59.
- Horas: de 0 a 23.
- Día del mes: de 1 a 31.
- Mes: de 1 a 12.
- Día de la semana: de 0 a 6, siendo 0 el domingo.

Si se deja un asterisco, quiere decir “cada” minuto, hora, día de mes, mes o día de la semana.

a) Ejemplo 1

```
* * * * * /bin/MiWeb/script.sh
```

Ejecutaria el script:

1. Cada minuto
2. De cada hora
3. De cada día del mes
4. De cada mes
5. De cada día de la semana

b) Ejemplo 2

```
30 2 * * 1 /bin/ejecutar/script.sh
```

Ejecutaría el script:

1. En el minuto 30
2. De las 2 de la noche
3. De cada día del mes
4. De cada mes
5. Sólo si es lunes

Todos los viernes a las 2:30 horas se ejecutará el script.

c) Ejemplo 3

Ejecutar un script de lunes a viernes a las 2:30 horas:

```
30 2 * * 1-5 /bin/MiWeb/script.sh
```

d) Ejemplo4

Ejecutar un script de lunes a viernes cada 10 minutos desde las 2:00 horas durante una hora:

```
0,10,20,30,40,50 2 * * 1-5 /bin/MiWeb/script.sh
```

otra opción sería

```
*/10 2 * * 1-5 /bin/MiWeb/script.sh
```

8.6. Palabras reservadas

@reboot Ejecuta una vez, al inicio

@yearly ejecuta sólo una vez al año: 0 0 1 1 *

@annually igual que @yearly

@monthly ejecuta una vez al mes, el día primero: 0 0 1 * *

@weekly Semanal el primer minuto de la primer hora de la semana. 0 0 * * 0".

@daily diario, a las 12:00A.M. 0 0 * * *

@midnight igual que @daily

@hourly al primer minuto de cada hora: 0 * * * *

a) Ejemplo 5

```
@hourly /home/usuario/scripts/molestorecordatorio.sh
```

```
@monthly /home/usuario/scripts/respaldo.sh
```

```
@daily apt-get update
```

Tareas de crontab

Ejecutar el script actualizar.sh (esta la carpeta bin de tu directorio personal.)a las 10:15 a.m. todos los días,

```
15 10 * * * usuario /home/usuario/scripts/actualizar.sh
```

Ejecutará el script actualizar.sh a las 10:15 p.m. todos los días

```
15 22 * * * usuario /home/usuario/scripts/actualizar.sh
```

Ejecutar apt -y update todos los domingos a las 10:00 a.m

```
00 10 * * 0 apt -y update
```

El día 20 de noviembre a las 7:30 se ejecutar script.sh

```
30 7 20 11 * usuario /home/usuario/scripts.sh
```

El día 11 de noviembre a las 7:30 a.m. y que sea domingo

Manejo de procesos y servicios

30 7 11 11 sun usuario /home/usuario/scripts.sh

Cuando se ejecuta esta tarea 01 * * * * usuario /home/usuario/scripts/molesto.sh

Cuando se ejecuta esta tarea 30 17 * * 1,2,3,4,5

A las 12 del día todos los días primero, quince y 28 de cada mes

00 12 1,15,28 * *

zenity --info --text "hola"