

# MÓDULO I. Administración de Linux

## Sesión 4. Automatización de tareas

2025/07/01

### Índice

1. Introducción . . . . .	1
2. Objetivos Principales . . . . .	2
3. Demonios y Programación de Tareas . . . . .	2
3.1. Características de un proceso demonio . . . . .	2
3.2. Programación de Tareas con at, atq, atrm y batch . . . . .	3
3.3. Salida estándar y Salida de error estándar . . . . .	5
3.4. Control de acceso a la orden at . . . . .	7
3.5. Programación de Tareas Periódicas con cron . . . . .	7
3.6. Gestiona tus tareas programadas con crontab . . . . .	8
3.7. Formato de los archivos crontab: especificando variables de entorno . . . . .	9
3.8. Control de Acceso a la Orden cron . . . . .	11

### 1. Introducción

**AVISO respecto al trabajo en las aulas de prácticas:** Mientras no se diga lo contrario, trabajaremos sobre Ubuntu en nuestra cuenta de usuario personal; no haremos uso de los privilegios como *root*.

Ya sea como usuario normal o como administrador del sistema habrá casos en que tendremos que lanzar órdenes cuya ejecución se realice en un determinado momento de tiempo o bien de forma periódica. El entrenamiento en esta materia es el propósito fundamental de esta práctica. Aparte del uso en sí de las órdenes propias para conseguir la ejecución postergada o periódica, hemos de comprender otros aspectos, como el **tratamiento** que se hace de las **variables de entorno** del proceso que ejecutará nuestra orden, o el **tratamiento** de la **entrada estándar, salida estándar** y salida de error estándar. Además, el administrador del sistema será el responsable del buen funcionamiento de estos servicios del sistema operativo: el demonio *atd* (que proporciona el servicio de ejecución postergada de órdenes) y

demonio cron (que proporciona el servicio de ejecución periódica de órdenes); el mantenimiento de estos demonios también tendrá que ser objeto de nuestro estudio y entrenamiento (parcialmente de momento dada la configuración del software que podemos utilizar en las aulas de prácticas).

## **2. Objetivos Principales**

- Conocer el concepto de proceso demonio y profundizar en los conocimientos adquiridos en la sesión anterior acerca de la información asociada a cada proceso (fundamentalmente a través de la orden ps).
- Conocer y saber utilizar la orden at y sus órdenes asociadas (atq, atrm y batch).
- Comprender qué valores tienen las variables de entorno del proceso que se lanzará para ejecutar una orden de ejecución postergada.
- Conocer el formato de archivo “crontab” y la orden crontab para lanzar órdenes de ejecución periódica.
- Comprender qué valores tienen las variables de entorno del proceso que se lanzará para ejecutar una orden de ejecución periódica.
- Respecto al servicio de ejecución periódica de órdenes, conocer la forma de habilitar y deshabilitar a usuarios para su uso.
- Respecto al servicio de ejecución periódica de órdenes, conocer la forma de iniciar y terminar este servicio.

## **3. Demonios y Programación de Tareas**


### **3.1. Características de un proceso demonio**

Un proceso demonio se caracteriza porque:

- Se ejecuta en background y no está asociado a un terminal o proceso login.
- Muchos se inician durante el arranque del sistema y continúan ejecutándose mientras el sistema esté encendido; otros sólo se ponen en marcha cuando son necesarios y se detendrán cuando dejen de serlo.
- En caso de que termine por algún imprevisto es muy común que exista un mecanismo que detecte la terminación y lo re arranque.
- En muchos casos está a la espera de un evento.
- En el caso frecuente de que el demonio sea un servidor, el evento por el que espera es que llegue una petición de realizar un determinado servicio; habrá siempre, por tanto, algún mecanismo que permita la comunicación entre el demonio en cuanto servidor y los procesos cliente.
- En otros casos, el demonio tiene encomendada una labor que hay que hacer de forma periódica, ya sea cada cierto tiempo o cuando se cumpla cierta condición.
- Es muy frecuente que no haga directamente el trabajo: lanza otros procesos para que lo realicen.
- Para conservar la filosofía de la modularidad propia de Unix, los demonios son programas y no parte del kernel.

- En muchos casos se ejecutan con privilegio de superusuario (UID=0) y tienen por padre al proceso init (PID=1).

El término castellano “demonio” tiene dos equivalencias en inglés con connotaciones bien diferentes: “daemon” y “demon”. El uso del término “demonio” en el contexto informático no pretende hacer alusión a aspectos malvados que conlleva “demon”, sino que pretende resaltar el aspecto de entidad con vida propia, independiente, que se ejecuta en un plano invisible, a la que alude “daemon”.

 **Actividad 4.1. Consulta de información sobre procesos demonio** A partir de la información proporcionada por la orden `ps` encuentre los datos asociados a los demonios `atd` y `cron`, en concreto: quién es su padre, qué terminal tienen asociado y cuál es su usuario.

**Nota 1:** Si el `demonio atd` no está instalado en su sistema puede descargar su paquete asociado e instalarlo usando las órdenes de las primeras sesiones. Hay que instalarlo y arrancar el servicio con la orden `service atd start`.

**Nota 2:** Si el `demonio cron` no está arrancado, puedes iniciar el servicio con la orden `service cron start`.

### 3.2. Programación de Tareas con `at`, `atq`, `atrm` y `batch`

El demonio `atd` permite la programación puntual de tareas, lo que significa que puedes especificar que una orden se ejecute una única vez en un momento determinado en el futuro.

Como usuarios de este servicio interactuamos con `atd` con las siguientes órdenes:

- `at`: ordenar la ejecución de órdenes a una determinada hora
- `atq`: consultar la lista de órdenes
- `atrm`: eliminar órdenes
- `batch`: ordenar la ejecución de órdenes que se ejecutarán cuando la carga del sistema sea baja.

Su sintaxis completa es:

```
at [-q queue] [-f <script>] [-mldbv] TIME
```

La orden `at` lee órdenes de la entrada estándar o del archivo `<script>` y provoca su ejecución a la hora especificada en `TIME` (una sola vez), usando `/bin/sh`. `TIME` admite muchos formatos, por ejemplo `HH:MM`. Para una descripción de las posibilidades para expresar la hora vea la ayuda de `at` y el contenido del archivo `/usr/share/doc/at/timespec`.


Con el ejemplo siguiente, a una determinada hora (17:10) se generará la lista de archivos del directorio `home` en un archivo de nombre `listahome`:

```
at 17:10
at> ls ~ > listahome
```

at> ^D <EOT>


job 2 at Thu Jul 10 17:10:00 2025

Recuerda ^D significa pulsar la combinación de teclas (EOF), al igual que ^C significa (finalizar el proceso actual), y ^Z es (suspender la ejecución del proceso actual). Si para este momento de las prácticas no tienes clara la diferencia entre estas tres acciones, debes preguntar a tu profesor de forma **urgente**.


 **Actividad 4.2. Ejecución postergada de órdenes con at (I)** Crea un archivo genera-apunte que escriba la lista de hijos del directorio home en un archivo de nombre listahome-date +%Y-%j-%T-\$\$ (consulte la ayuda de date). Lanza la ejecución del archivo genera-apunte un minuto más tarde de la hora actual. ¿En qué directorio se crea el archivo de salida?

La orden **at** gestiona distintas colas de trabajos que esperan ser ejecutados; se designan con una única letra de la a a la z y de A a Z; la cola por omisión es **a**, la cola **b** se usa para los trabajos batch, y a partir de ahí las distintas colas van teniendo menor prioridad al ir pasando de **c en adelante**. Usa la orden **man** para obtener más información.


La orden **nbatch** es equivalente a **at** excepto que no especificamos la hora de ejecución, sino que el trabajo especificado se ejecutará cuando la carga de trabajos del sistema esté bajo cierto valor umbral que se especifica a la hora de lanzar el demonio **atd**.

 **Actividad 4.3. Ejecución postergada de órdenes con at (II)** Lanza varias órdenes **at** utilizando distintas formas de especificar el tiempo como las siguientes: (será de utilidad la opción **-v**):

- a) a medianoche de hoy
- b) un minuto después de la medianoche de hoy
- c) a las 17 horas y 30 minutos de mañana
- d) a la misma hora en que estemos ahora pero del día 25 de diciembre del presente año
- e) a las 00:00 del 1 de enero del presente año

 **Actividad 4.4. Cuestiones sobre at** Utiliza las órdenes **atq** y **atrm** para familiarizarte con su funcionamiento (consulta la ayuda de estas órdenes).


Cuando sea el momento que se especificó en la orden **at**, se lanzará una shell **/bin/sh** para ejecutar el archivo **<script>** que se indicó (o si no se especificó, se lanza el conjunto de órdenes que se tomaron de la entrada estándar); podemos preguntarnos sobre cuál es el entorno de este nuevo proceso.

 **Actividad 4.5. Relación padre-hijo con órdenes ejecutadas mediante at** El proceso nuevo que se lanza al cumplirse el tiempo que se especificó en la orden **at**...:

- ¿qué directorio de trabajo tiene inicialmente? ¿hereda el que tenía el proceso que invocó a **at** o bien es el **home**, directorio inicial por omisión?

- ¿qué máscara de creación de archivos `umask` tiene? ¿es la heredada del padre o la que se usa por omisión?
- ¿hereda las variables locales del proceso padre?

Experimenta con la orden `at` lanzando las órdenes adecuadas para encontrar las respuestas. (Puede encontrar información en la ayuda de `at`).

 **Actividad 4.6. Script para orden `at`** El proceso nuevo que se lanza al cumplirse el tiempo que se especificó en la orden `at`... ¿de quién es hijo? Investiga lanzando la ejecución retardada de un script que muestre la información completa sobre los procesos existentes y el `pid` del proceso actual; el script podría contener lo que sigue:

```
nombearchivo=`date +%Y-%j-%T`
ps -ef > $nombearchivo
echo Mi pid = $$ >> $nombearchivo
```

#### **Actividad 4.7. Trabajo con la orden `batch`**

- a) Cree un script llamado `scriptTarea.sh` que ejecute las siguientes tareas:
  1. Obtener el año actual.
  2. Obtener un listado de los ficheros del directorio actual.
- b) Programe el script para que se ejecute de forma puntual (no repetitiva) el 22 de diciembre, a las 11 de la noche del año actual; Cuando se ejecute el script (el 22 de diciembre), este debe crear un archivo llamado `Tareapuntual.txt` en el directorio actual donde se guardarán los resultados de la ejecución de las tareas 1 y 2.

### **3.3. Salida estándar y Salida de error estándar**

Si ejecutando el intérprete de órdenes de forma interactiva lanzamos la ejecución de un script, la nueva ejecución del shell que se lanza tiene como entrada estándar, salida estándar y salida de error estándar al teclado (para la entrada) y pantalla (para salida y salida de error) del terminal desde el que estamos trabajando.


Sin embargo, al lanzar la ejecución asíncrona de una tarea con la orden `at` **NO** estamos creando un proceso hijo de nuestra shell, este nuevo proceso **NO** tendrá como entrada estándar, salida estándar y salida de error estándar los asociados a nuestra consola de terminal. Cuando la orden lanzada de forma retardada sea ejecutada, lo que se genere en la salida estándar y la salida de error estándar se envía al usuario que envió la orden como un correo electrónico usando la orden `/usr/bin/sendmail` (la ubicación concreta puede depender de la instalación), si el servicio está disponible. Hemos de tener esto en cuenta y considerar sus consecuencias.


Por ejemplo, hemos de vigilar que no se generen un número excesivamente elevado de salidas que pudieran crear problemas de espacio. En el caso de lanzar la ejecución de órdenes que generan muchos mensajes en la salida estándar o salida de error estándar podríamos redirigirlas a `/dev/null` para que se “pierda” y no inunde el buzón de correo:

```
at now + 1 minute
at> tar cvf /backups/backup.tar . 1>> ~/salidas 2> /dev/null
at> ^D <EOT>
job 3 at Thu Jun 10 09:58:00 2025
```


También puede ser conveniente, como se ve arriba, redirigir la salida estándar del programa a un archivo para que pueda consultarse en caso de necesidad. En el ejemplo anterior estamos seguros del buen funcionamiento de la línea en que está la orden `tar`, de modo que sabemos que sus mensajes de error no serán de nuestro interés.

Pero situémonos ahora en el caso de que hemos lanzado una orden con `at` sin haber redirigido la salida de error. Entonces nos pasará totalmente desapercibido el error, no tendremos ningún resultado de la ejecución de nuestro trabajo y nos parecerá que no ha funcionado el mecanismo de ejecución postergada. Entre los errores más frecuentes está el de crear un script sin darle inicialmente permiso de ejecución, de modo que cuando llegue su momento la ejecución no será posible, no tendremos a la vista el mensaje de error “Permiso denegado” y parecerá que el mecanismo de la orden `at` no funciona. Otro error frecuente es no tener incluido en la lista de búsqueda al directorio actual, de modo que invocar a un script que cuelga de donde estamos simplemente por su nombre sin anteponerle `./` dará el error “orden no encontrada” que, en el caso de una ejecución retardada, se perderá. Si el mecanismo `sendmail` está funcionando entonces recibiremos un correo con los mensajes de error. Si no es el caso, hemos de ser nosotros mismos quienes redirijamos las salidas de error para poder rastrear la ejecución de nuestra orden.

 **Actividad 4.8. Utilización de las colas de trabajos de at** Construye un script que utilice la orden `find` (haz `man find` y repasa Fundamentos del Software), para generar en la salida estándar los archivos modificados en las últimas 24 horas (partiendo del directorio `home` y recorriéndolo en profundidad), la salida deberá escribirse el archivo de nombre `modificados_<año><día><hora>` (dentro del directorio `home`). Con la orden `at` provoca que se ejecute dentro de un día a partir de este momento.

 **Actividad 4.9. Relación padre-hijo con órdenes ejecutadas mediante crontab** Lanza los procesos que sean necesarios para conseguir que exista una gran carga de trabajo para el sistema de modo que los trabajos lanzados con la orden `batch` no se estén ejecutando (puede simplemente construir un script que esté en un ciclo infinito y lanzarla varias veces en segundo plano). Utiliza las órdenes oportunas para manejar este conjunto de procesos (la orden `jobs` para ver los trabajos lanzados, `kill` para finalizar un trabajo, ...y tal vez también las órdenes `fg`, `bg` para pasar de segundo a primer plano y viceversa, `<Ctrl-Z>` para suspender el proceso en primer plano actual, etc). Experimenta para

comprobar cómo al ir disminuyendo la carga de trabajos habrá un momento en que se ejecuten los trabajos lanzados a la cola batch.

 **Actividad 4.10. Ejecución de scripts con crontab (I)** Construye tres scripts que deberás lanzar a las colas c, d y e especificando una hora concreta que esté unos pocos minutos más adelante (no muchos para ser operativos). Idea qué actuación deben tener dichos scripts de forma que se ponga de manifiesto que de esas colas la más prioritaria es la c y la menos es la e. Visualiza en algún momento los trabajos asignados a las distintas colas.

### 3.4. Control de acceso a la orden at

Los dos archivos de configuración `/etc/at.deny` y `/etc/at.allow` determinan qué usuarios pueden usar la orden `at`. El archivo `at.allow`, si existe, contiene una lista con el nombre de todos los usuarios habilitados para ello (uno por línea). Si no existe el archivo `allow`, se comprobará el contenido del archivo `deny` y se entenderá que un usuario podrá usar `at` a no ser que esté presente en `deny`. Si ninguno de estos dos archivos existe entonces qué usuarios están autorizados depende del sistema Linux concreto y de los parámetros de configuración.

### 3.5. Programación de Tareas Periódicas con cron

Muchas de las tareas de administración se tienen que llevar a cabo de forma periódica, a continuación veremos cómo el sistema operativo nos brinda facilidades para ello. Uno de los demonios básicos es `cron`, responsable de ejecutar órdenes con una periodicidad determinada. La especificación de las tareas que se desea que ejecute se hace construyendo un archivo (llamado archivo “crontab”) que deberá tener un determinado formato, (llamado formato “crontab”). Será con la orden `crontab` con la que indicamos el archivo con formato “crontab” que deseamos comunicar al demonio `cron`.

Cada línea de código de un archivo `crontab` (excepto los comentarios que están precedidos por `#`) puede contener estos campos (que representan una orden):

`minuto hora día-del-mes mes día-de-la-semana orden`

Los campos `minuto`, `hora`, `día-del-mes`, `mes` y `día-de-la-semana` se encargan de indicar la periodicidad con que deseamos que se ejecute la orden. Si se trata de una orden shell o un script se lanzará `/bin/sh` para su ejecución, y si es la ruta de un archivo ejecutable entonces se provocará su ejecución.

Cada uno de los campos de determinación del tiempo (`minuto`, `hora`, `día-del-mes`, `mes`, `día-de-la-semana`) puede contener:

- un asterisco (\*), que indica cualquier valor posible
- un número entero, que activa ese valor determinado
- dos enteros separados por un guión, que indican un rango de valores

- una serie de enteros o rangos separados por una coma, activando cualquier valor de los que aparecen en la lista.

Ejemplo de especificación de una hora:

```
1 20 * * 1-5
```

Esto significa “a las 20 horas y 1 minuto de lunes a viernes”.

Si se han indicado valores concretos para día-del-mes y día-de-la-semana, se entenderá que la condición es que se cumpla día-del-mes OR día-de-la-semana (en lugar de AND como podría pensarse); por ejemplo:

```
0,30 * 13 * 5
```

Significa “cada media hora el viernes y cada media hora el día 13 del mes” y no “cada media hora de todos los viernes 13”.

Consulte la ayuda del formato de archivo crontab (en la sección 5 del manual) para una descripción completa de todas las posibilidades sobre la especificación del tiempo.

La orden crontab se encarga de instalar, desinstalar o listar los trabajos que procesará el demonio cron. La sintaxis para especificar el archivo <file> como archivo con formato “crontab” que contiene la lista de trabajos para cron es:

```
crontab <file>
```


### 3.6. Gestiona tus tareas programadas con crontab


crontab es una herramienta fundamental en Unix/Linux que te permite programar la ejecución automática de comandos o scripts en momentos específicos. Aquí te presento las tres opciones principales para interactuar con crontab:


- `crontab -e`: Esta opción te permite editar tu archivo crontab personal con tu editor favorito, definido en la variable de entorno EDITOR, puedes redefinir esta variable en cualquier momento. Al ejecutarla, se abrirá un editor de texto (generalmente vi o nano) donde podrás añadir, modificar o eliminar las tareas programadas. Cada línea en este archivo representa una tarea (cron job) con su horario de ejecución y el comando a ejecutar.
- `crontab -l`: Con esta opción, puedes listar el contenido actual de tu archivo **crontab**. Es útil para verificar rápidamente qué tareas tienes programadas sin necesidad de abrir el editor.
- `crontab -r`: Esta opción se utiliza para eliminar completamente tu archivo **crontab** personal. ¡Ten mucha precaución al usarla! Una vez ejecutada, todas tus tareas programadas serán borradas y no podrán recuperarse fácilmente.




Recuerda que cada usuario tiene su propio archivo **crontab**. Esto significa que las tareas que programas con `crontab -e` solo se ejecutan bajo tu usuario.

 **Actividad 4.11. Ejecución de scripts con crontab (II)** Al igual que se investigó en la Actividad 4.5 sobre quién es el proceso padre del nuestro, lanza el script construido en dicha actividad con una periodicidad de un minuto y analiza los resultados.

 **Actividad 4.12. Ejecución de scripts con crontab (III)** Construye un script que sea lanzado con una periodicidad de un minuto y que borre los nombres de los archivos que cuelguen del directorio `/tmp/varios` y que comiencen por “core” (cree ese directorio y algunos archivos para poder realizar esta actividad). Utiliza la opción `-v` de la orden `rm` para generar como salida una frase de confirmación de los archivos borrados; queremos que el conjunto de estas salidas se añadan al archivo `/tmp/listacores`. Prueba la orden `crontab -l` para ver la lista actual de trabajos (consulte la ayuda para ver las restantes posibilidades de esta orden para gestionar la lista actual de trabajos).

 **Actividad 4.13. Variables de entorno en archivos crontab** Para asegurar que el contenido del archivo `/tmp/listacores` no crezca demasiado, queremos que periódicamente se deje dicho archivo solo con sus 10 primeras líneas (puede ser de utilidad la orden `head`). Construye un script llamado `reducelista` (dentro del directorio `~/SO`) que realice la función anterior y lance su ejecución con periodicidad de un minuto.

 **Actividad 4.14. Archivos crontab de diferentes usuarios** Construye un sencillo script que escriba en el archivo `~/SO/listabusqueda` una nueva línea con la fecha y hora actual y después el valor de la lista de búsqueda, por ejemplo:

```
...
2011-297-12:39:10 - /usr/local/bin:/usr/local/bin:/usr/bin...
...
```

Ejecuta este script desde el lenguaje de órdenes y también lánzalo como trabajo `crontab` y compara los resultados, ¿se tiene en ambos casos la misma lista de búsqueda?

### 3.7. Formato de los archivos `crontab`: especificando variables de entorno

Una línea de un archivo `crontab` puede ser o bien una línea de especificación de una orden para `cron` como hemos explicado hasta ahora, o bien una línea de asignación de valores a variables de entorno, con la forma:

`<nombre>=<valor>`

Algunas variables de entorno son establecidas automáticamente por el demonio `cron`, como las siguientes:

- SHELL se establece a `/bin/sh`
- LOGNAME y HOME se toman del archivo `/etc/passwd`

Un detalle importante es que sobre `<valor>` no se realizan sustituciones de variables, de modo que una línea como la siguiente no funcionaría para utilizar los elementos de la lista de búsqueda actual:


```
PATH=$HOME/SO:$PATH
```


 **Actividad 4.15. Ejecución de scripts con crontab (IV)** Practicamos ahora lo que acabamos de explicar situándonos en lo que hemos realizado en la Actividad 4.11.

Construye un script que generará un archivo crontab llamado `crontab-reducelista` que deberá contener:

- Como primera línea la asignación a la variable PATH de la lista de búsqueda actual y además el directorio `$HOME/SO`
- Después la indicación a cron de la ejecución con periodicidad de 1 minuto del script `reducelista`

Una vez construido `crontab-reducelista` lánzalo con la orden `crontab`. Comprueba que con esta nueva lista de búsqueda podremos hacer alusión a `reducelista` especificando únicamente su nombre independientemente del directorio de trabajo en que nos situemos (no como ocurría en la Actividad 4.11 en que el directorio `$HOME/SO` no estaba en la lista de búsqueda).

 **Actividad 4.16. Gestión del servicio cron como usuario root** Vamos a lanzar un archivo crontab cuyo propietario es otro usuario. Visualiza el contenido del archivo `/fenix/depar/lsi/so/ver-entorno` y `/fenix/depar/lsi/so/crontabver`. Comprueba con `ls -l` que el propietario es el usuario `lsi`. Sin copiarlos, úsalos para lanzar la ejecución cada minuto del script `/fenix/depar/lsi/so/ver-entorno`. Analiza el archivo de salida: ¿de qué línea del archivo `/etc/passwd` se toman LOGNAME y HOME, de la línea del propietario del archivo crontab o de la línea del usuario que lanza el archivo crontab?

 **Actividad 4.17. Gestión de tareas automáticas con cron** El objetivo es ejecutar todos los días a las 0 horas 0 minutos una copia de los archivos que cuelgan de `$HOME` que se hayan modificado en las últimas 24 horas. Vamos a programar este salvado incremental utilizando la orden `find` que usábamos en la Actividad 4.6; ahora queremos que se copien los archivos encontrados por `find` utilizando la orden `cpio`:

```
<orden find de la Actividad 4.6> | cpio -pmduv /tmp/salvado$HOME
```

Una característica interesante de esta orden (y que no tiene `cp`) es que puede tomar de la entrada estándar los archivos origen a copiar; con las opciones que se presentan en el ejemplo anterior replica la estructura original manteniendo metadatos de especial interés como usuario propietario y grupo propietario (consulte la ayuda de `cpio` para obtener información sobre cada una de las opciones). Por

ejemplo, una vez al mes se puede hacer un salvado global, y diariamente salvar únicamente los archivos modificados en ese día.

No es necesario que lo pruebes, pero una posibilidad interesante es que la copia se haga en un dispositivo que acabamos de montar, y justo al terminar lo desmontemos; esto aumentaría la seguridad de esa copia ante posibles ataques:


```
mount /dev/loop0 /directoriosalvados  
<orden find> | <orden cpio>  
umount /dev/loop0
```

Como última observación, si el dispositivo y punto de montaje usados en esa orden mount no están en el fstab serán más difíciles de detectar por un intruso que acceda a nuestro sistema.


**AVISO:** para realizar este apartado hemos de trabajar como usuario root.

### 3.8. Control de Acceso a la Orden cron

Los dos archivos de configuración /etc/cron.deny y /etc/cron.allow determinan qué usuarios pueden ejecutar la orden crontab. Su significado es equivalente a los archivos /etc/at.deny y /etc/at.allow explicados anteriormente.

 **Actividad 4.18. Lanzamiento y parada del servicio cron** Prueba las siguientes operaciones sobre el demonio cron:

1. Como usuario root, deshabilita/habilita a un determinado usuario para que pueda utilizar el servicio cron; comprueba que efectivamente funciona.
2. Iniciar y terminar el servicio cron. Prueba las siguientes órdenes para iniciar y terminar este servicio:
  - Iniciar el servicio cron: `/sbin/service cron start`
  - Terminar el servicio cron: `/sbin/service cron stop`

 **Actividad 4.19. Programación de tareas periódicas con scripts** La actividad consiste en programar una tarea periódica utilizando un script. Este script llevará a cabo la siguiente acción: Monitoreo de Memoria: Usará un comando que permita mostrar información acerca del uso de memoria del sistema, con el objetivo de monitorear su estado.

El script deberá crear automáticamente un archivo llamado LOG-fecha en tu directorio home. El formato de este nombre de archivo será LOG-AAAA (por ejemplo, LOG-2025 si el año actual es 2025).

Además, debes conseguir que este script se ejecute del 15 al 20 de diciembre y junio, a las 22:30h. Los resultados de cada ejecución se guardarán en el archivo LOG-fecha correspondiente, permitiendo así observar la evolución del uso de la memoria a lo largo del tiempo.

Para que puedas entrenar, aquí tienes algunas otras acciones comunes que podrías probar:

**1. Backup Diario de un Directorio** Programa un script en Bash que comprima el directorio `~/proyecto` en un archivo `proyecto-YYYYMMDD.tar.gz` y lo almacene en `~/backups` con una tarea diaria de cron.

**2. Monitorización de tamaño de archivos** Crea un script que:

Ejecute `du` sobre el directorio `/var/log` para comprobar el tamaño del directorio de logs del sistema.

Si el tamaño supera los 500M, cree un archivo de aviso en `$HOME/reports/log_alerta.txt` con la fecha y el tamaño encontrado.

Programa este script en cron para que se ejecute diariamente a las 02:00.

**3. Organización automática de ficheros** Crea un script que:

a) Busque en el directorio `~/Downloads` todos los archivos que:

1. No se hayan accedido en los últimos 30 días,
2. Y superen 100M de tamaño.

b) Mueva estos archivos a una carpeta de respaldo en `~/Downloads/antiguos/`, que se debe crear si no existe.

c) Registre en `~/logs/limpieza_descargas.log` la fecha y los nombres de los archivos movidos.

Programa este script en cron para que se ejecute todos los lunes a las 9:00 am.

**4. Limpieza Diaria de Archivos Temporales** Escribe un script que elimine ficheros con más de 7 días en `/tmp` y `$HOME/.trash//var/tmp` usando `find ... -mtime +7 -delete` y configúralo para que corra diariamente.

**5. Rotación Diaria de Logs** Implementa un script que rote `app.log` renombrándolo a `app-YYYYMMDD.log`, vacíe el archivo original y lo ejecute a diario mediante `crontab`.

**6. Informe Mensual de Procesos** Genera un script que recopile los 10 procesos con mayor uso de CPU y memoria (`ps aux --sort=-%cpu | head -n11` y `ps aux --sort=-%mem | head -n11`), guarde el informe `top-YYYYMM.txt` en `$HOME/reports` y prográmalo mensualmente.

**7. Log Semanal de Actividad de Usuarios** Establece un archivo LOG en tu directorio home y programa un job en cron para que todos los martes a las 12:15 durante el mes de noviembre ejecute un script que inserte la fecha actual, muestre el listado de usuarios y su tiempo de uso, y liste los archivos modificados en el home en los últimos 30 minutos, guardando toda la salida en LOG.

**8. Registro Diario Horario (Excluyendo Fines de Semana)** Planifica una tarea en cron que añada la fecha y la hora al fichero /tmp/hora-{usuario}.log cada día a las 15:00 durante el mes de febrero, excluyendo sábados y domingos (donde {usuario} es el nombre del usuario que ha planificado la tarea).