UF4 - Sistemas operativos. Gestión de usuarios y procesos

SISTEMAS INFORMÁTICOS

Índice

Gestión de usuarios y procesos en Windows

En las versiones más recientes de Windows, se ha separado la gestión clásica de usuarios de otros aspectos como la vinculación con una cuenta Microsoft.

Para abrir la configuración básica de usuarios y grupos (solo para usuarios administradores), abre *Ejecutar*, escribe **`lusrmgr.msc`** y pulsa *Enter*. Aquí podrás crear y borrar usuarios y grupos, cambiar la descripción de usuarios y grupos, la contraseña de los usuarios y la pertenencia de los usuarios a los grupos del sistema.

- Los usuarios de tipo administrador pertenecen al grupo **Administradores**, lo que implica que tienen todos los permisos del sistema.

Para ver los procesos y recursos consumidos y disponibles, existe el conocido **Administrador de tareas** y herramientas más avanzadas como **Monitor de recursos** y **Monitor de rendimiento**.

Gestión de usuarios en Linux

Usuarios

En Linux, cualquier proceso debe ejecutarse en nombre de un usuario registrado en el sistema, por lo que para usar el sistema se requiere un **inicio de sesión**, donde se introducen las **credenciales** (identificador y una prueba de autenticación, como la contraseña) de un usuario del sistema.

Un **usuario** tiene dos datos que le identifican: un ***login*** (o nombre de usuario) y **UID** únicos.

Los usuarios tienen restringidos permisos para ver, modificar y/o ejecutar la mayor parte de archivos del sistema operativo, aunque cuentan con un directorio personal propio donde cuentan con todos los permisos.

Existe un usuario especial que no tiene ninguna restricción: el usuario **`root`** (administrador). Cualquier usuario con capacidades *sudo* puede suplantar a `root` (u otro usuario) mediante el comando **`sudo`**.

Usuarios y grupos

El SO permitirá o denegará el acceso a ciertos recursos software y hardware dependiendo de qué usuario o proceso quiera utilizarlo. A esto le llamamos **autorización**.

Al crear un usuario, se pueden personalizar sus parámetros o bien dejar que se le apliquen los parámetros por defecto (por ejemplo, si puede iniciar sesión, su carpeta personal, etc.)

Cuando los permisos son compartidos por varios usuarios, se pueden definir políticas o directivas de seguridad para **grupos de usuarios** (por ejemplo, grupos de alumnos, de profesores y de administradores).

- Así, un usuario tiene los privilegios y restricciones **de los grupos a los que pertenece** más los privilegios y restricciones **de su propio usuario**.
- Un usuario pertenece a un **grupo principal** y a cualquier número de **grupos secundarios**.

Los cambios en usuarios y grupos solo los puede realizar un usuario con **privilegios de administrador**.

Perfiles de usuario y grupos en Linux

Los sistemas tipo UNIX diferencian entre **usuarios** y **grupos** de usuarios, cuyos datos se describen en los siguientes archivos:

- **`/etc/passwd`**: guarda información sobre **usuarios** (normales y de sistema). La sintaxis de cada línea es:

  Nombre de usuario **:** contraseña sin cifrar **:** identificador de usuario (UID) **:** identificador de grupo principal (GID) **:** información adicional **:** ubicación del directorio personal **:** intérprete de comandos

- **`/etc/shadow`**: guarda información de usuarios con **contraseñas cifradas**. La sintaxis de cada línea es:

  Nombre de usuario **:** hash de contraseña **:** fecha de última modificación **:** configuraciones de cambio de contraseñas (ver `chage`)

- **`/etc/group`**: guarda información sobre **grupos**. La sintaxis de cada línea es:

  Nombre de grupo **:** contraseña sin cifrar **:** identificador de grupo **:** lista de miembros

- **`/etc/gshadow`**: guarda la información sobre *grupos seguros*. La sintaxis de cada línea es:

  Nombre de grupo **:** contraseña cifrada **:** lista de administradores **:** lista de miembros

Comandos de gestión de usuarios

Existen comandos básicos que permiten realizar acciones sobre los usuarios:

- `useradd`: añade un usuario. Tiene opciones para personalizarlo e incluso para crear su directorio personal en el acto (a partir del directorio `/etc/skel`).
- `usermod`: modifica un usuario, según la(s) opción(es) que se le pase(n). También puede añadir o quitar a un usuario de un grupo.
- `userdel`: elimina un usuario. Con la opción -r, elimina también su directorio personal y su correo local.

Existen comandos básicos que realizan las mismas operaciones sobre grupos:

- `groupadd`: añade un grupo. Tiene opciones para personalizarlo.
- `groupmod`: modifica un grupo, según la(s) opción(es) que se le pase(n).
- `groupdel`: elimina un grupo.

Muchas distribuciones de Linux cuentan con los comandos de alto nivel `adduser`, `deluser`, `addgroup` y `delgroup`, que realizan operaciones adicionales configuradas en el SO.

Múltiples usuarios

Linux pone a nuestra disposición algunos comandos para ver información básica sobre usuarios y grupos:

- `id`: permite ver el nombre y UID del usuario (efectivo) actual, además de su grupo principal y los grupos a los que pertenece. Se puede obtener cada dato de forma aislada según la opción que se especifique.
- `whoami`: permite ver el nombre del usuario (efectivo) actual.
- `who`: permite ver una lista de las sesiones abiertas en el sistema actualmente. Una sesión tiene un usuario, una terminal, una fecha y hora de inicio de sesión y, en caso de ser remota (SSH), una dirección IP de origen.

Contraseñas

Para cambiar la contraseña de un usuario, podemos usar:

- `usermod`: con la opción `-p <contraseña>`, establece la contraseña en `/etc/shadow` directamente, sin cifrar, por lo que deberá estar cifrada previamente. Para cifrar una contraseña:

  ```
  $ openssh passwd [-1|-5|-6] <contraseña>
  ```

  - Algoritmos de hash: `-1` para MD5, `-5` para SHA256, `-6` para SHA512.
- `passwd`: cambia la contraseña del usuario actual o del usuario especificado (en este último caso, requiere permisos de administrador). También permite eliminar (`-d`), bloquear (`-l`) y desbloquear (`-u`) la contraseña de un usuario.

Para cambiar los parámetros de la contraseña de un usuario:

- `chage`: permite ver o modificar la fecha de último cambio de contraseña, los días que puede estar vigente una contraseña como mínimo y como máximo, los días hasta que advierta de que se debe cambiar de contraseña, e incluso la fecha de expiración de la cuenta.
  - Toda esta información se aloja en la línea de `/etc/shadow` correspondiente al usuario en cuestión.

Propietarios de ficheros

Todos los archivos del sistema pertenecen a un usuario y a un grupo. Para modificarlos:

- `chown`: modifica el usuario propietario o ambos propietarios (*`usuario`*`:`*`grupo`*) de el/los ficheros o directorios pasados como argumento.
- `chgrp`: como `chown`, pero solo modifica el grupo propietario.

Cambiar el propietario de un archivo requiere permisos de administrador.

Ambos comandos aceptan la opción `-R` para, en caso de modificar un directorio, modificar también su contenido de forma recursiva.

Permisos de ficheros

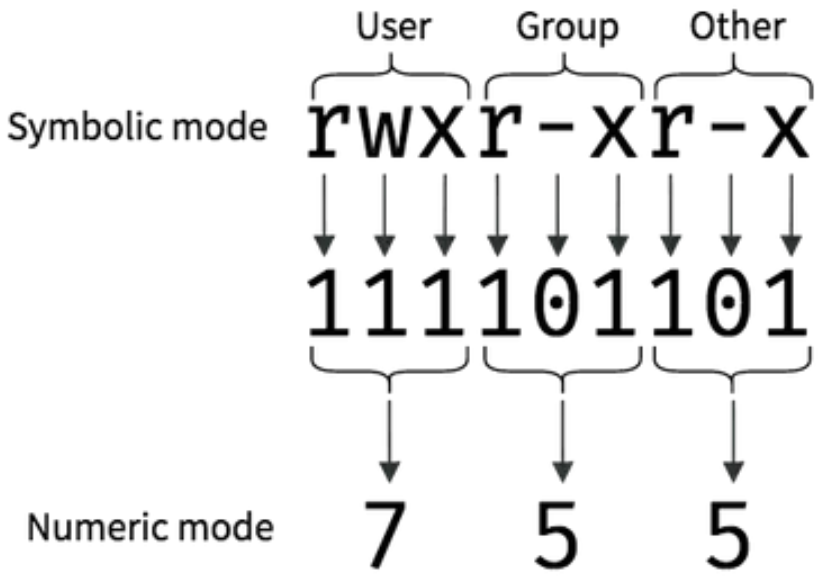
Todos los ficheros tienen una serie de permisos con los que el SO determina si un usuario puede leerlos (`r`), modificarlos (`w`) o ejecutarlos (`x`). Se escriben de la siguiente forma:

```
rwxrwxrwx
```

Donde el primer grupo de 3 corresponde a los permisos del usuario propietario (`u`), el segundo corresponde a los del grupo propietario (`g`) y el tercero corresponde a los del resto de usuarios (`o`).

Cada una de estas agrupaciones se representa como una cifra octal, donde cada permiso activo es un `1`.

- Por ejemplo, `rw-rw-r--` se representa como `664`.



Permisos especiales

Existen tres permisos especiales que forman una cifra octal adicional:

- **Set UID o SUID (4000)**: este bit permite a un fichero ser ejecutado con un usuario efectivo igual al usuario propietario del archivo. Se puede añadir el permiso con `u+s`.
- **Set GID o SGID (2000)**: este bit permite a un fichero ser ejecutado con un grupo efectivo igual al grupo propietario del archivo. En directorios, hace que los ficheros que se creen dentro tengan el mismo grupo que dicho directorio. Se puede añadir el permiso con `g+s`.
- **Sticky bit (1000)**: este bit permite a un directorio con permisos abiertos de escritura prohibir a un usuario eliminar ficheros contenidos en él sobre los que no tienen permiso de escritura. Se puede añadir el permiso con `+t`.

Modificar permisos

Se pueden modificar los permisos de un fichero o ficheros mediante:

- `chmod`: su primer argumento son los permisos que se quieren establecer. Pueden definirse en octal (en cuyo caso se modifican por completo) o de forma simbólica:

  `[<ugo>]<+-=><rwx>`

  - Se puede especificar para qué clase de usuarios se quiere modificar el permiso (`u`, `g`, `o`, `go`, `ugo`, etc.)
  - `+` para añadir permiso, `−` para quitarlo, `=` para dejar exactamente estos permisos.
  - Los permisos que se quieren añadir o quitar (`r`, `w`, `x`, `rw`, `rwx`, etc.). `s` y `t` para permisos especiales

Con la opción `−R`, los permisos se aplican también al contenido de directorios.

Por ejemplo:

```
$ chmod u+x script.sh
$ chmod go-r,u+w file1.txt file2.txt
$ chmod -R 755 dir/
```

Permisos por defecto

Al crear un nuevo fichero o directorio, se crea con unos permisos por defecto.

Los permisos originales de los directorios son `0777`, y los de los ficheros regulares son `0666`.

A estos permisos se les aplica una máscara de permisos que se puede ver y modificar con el comando `umask`.

Para calcular los permisos de un nuevo fichero o directorio, se realiza un AND entre los permisos originales y los bits negados (NOT) de la máscara. Por ejemplo:

- Si creo un nuevo fichero regular y mi máscara es `0022`:

  `0666 AND (NOT 0022) = 0666 AND 7755 =` **`0644`**
- Si creo un nuevo directorio y mi máscara es `0027`:

  `0777 AND (NOT 0027) = 0777 AND 7750 =` **`0750`**

Tipos de shells

Un **shell** es la interfaz con la que un usuario o proceso interactúa con el sistema operativo. Un shell puede ser:

- **Shell con inicio de sesión**: cuando al iniciar el shell se solicita autenticación del usuario.
  - Por ejemplo, cuando abrimos un terminal virtual (`CTRL+ALT+F`*`1-6`*), cuando nos conectamos por SSH, etc.
- **Shell sin inicio de sesión**: cuando al iniciar el shell no se solicita autenticación.
  - Por ejemplo, cuando abrimos un emulador de terminal en sistemas con GUI, o al crear un subshell al ejecutar la mayoría de comandos, etc.
- **Shell interactivo**: están asociados a un terminal y permiten interactuar escribiendo comandos, interrumpiendo los comandos, etc.
  - Por ejemplo, terminales virtuales, emuladores de terminal, comando bash, SSH, etc.
- **Shell no interactivo**: no está asociado a un terminal, como suele ser en un subshell, ya que normalmente se ejecuta de forma automática sin intervención del usuario.
  - Por ejemplo, cuando se ejecuta un script y prácticamente cualquier comando de la terminal que cree un subshell.

Configuración de perfiles

Los archivos de configuración más importantes para un usuario son:

- `/etc/skel`: directorio plantilla de los nuevos directorios personales. Suele incluir ficheros `.bashrc`, `.bash_logout` y `.profile`.
- `/etc/profile`: script que se ejecuta al iniciar un shell con inicio de sesión.
- `/etc/bash.bashrc`: script que se ejecuta al iniciar un shell Bash interactivo.

Además de los anteriores scripts globales, cada usuario tiene los siguientes scripts:

- `~/.bashrc`: script que se ejecuta justo después de `/etc/bash.bashrc`.
- `~/.bash_logout`: script que se ejecuta al cerrar un shell Bash interactivo.
- `~/.profile`: script que se ejecuta justo después de `/etc/profile`.

Es común que cada usuario personalice estos ficheros, especialmente `.bashrc`, para definir variables y alias personalizados (como `PS1`, la variable que define el prompt).

Variables

Una variable es un identificador que almacena una cadena de caracteres. Son empleadas por programas, comandos o el propio shell, almacenando valores de configuración.

Una variable (de shell) se define mediante:

```
NOMBRE=VALOR
```

Si el valor contiene espacios o caracteres especiales, debe entrecomillarse.

Para usar el valor de la variable, se le antepone un `$` delante (`$NOMBRE`). Por ejemplo:

```
$ echo $HOME
```

Existen dos tipos de variables:

- **Variables de shell**: solo son visibles por el shell actual, no por subshells.
- **Variables de entorno**: son visibles por la shell actual y sus subshells. Se definen mediante:

  ```
  export NOMBRE=VALOR
  ```

Variables

Algunas variables predefinidas:

- `SHELL`: intérprete del shell actual (`/bin/sh`, `/bin/bash`, etc.)
- `USER`: usuario actual.
- `PWD`: directorio actual.
- `PATH`: conjunto de rutas de directorios donde buscar los comandos.
- `HOME`: directorio personal del usuario actual.
- `HOSTNAME`: nombre del equipo.
- `PS1`: prompt.

Alias

Los alias permiten ejecutar comandos de manera personalizada, ahorrando tiempo en la escritura.

Para definir un alias:

```
alias NOMBRE='COMANDO'
```

Para ejecutar un alias, solo hay que escribir su nombre, como un comando cualquiera, con o sin argumentos.

Para eliminar un alias:

```
unalias NOMBRE
```

Por ejemplo:

```
$ alias grep='grep --color=auto'
$ grep sys /etc/passwd
```

Gestión de procesos en Linux

Gestión de procesos

En un SO multitarea siempre hay más procesos que núcleos capaces de ejecutarlos, por ello cada SO tiene un **planificador de procesos**.

Los procesos pueden pasar por distintos **estados**. El planificador se encarga de establecer el estado de cada proceso y de modificarlo, atendiendo a un **algoritmo de planificación**, asignando un tiempo de ejecución a aquellos procesos en ejecución.

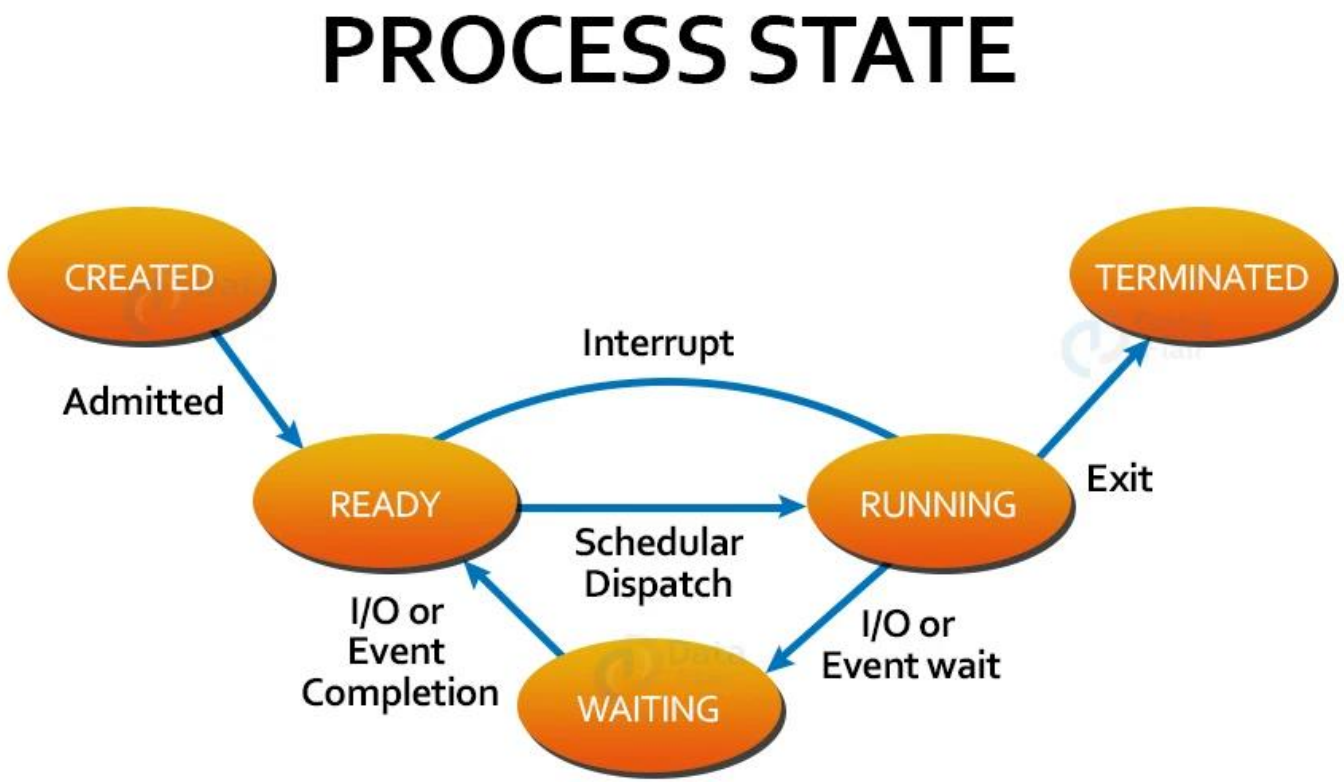
Pasado ese tiempo, genera una interrupción de reloj, el proceso se queda en una **cola de procesos** listos para ser ejecutados, el SO recupera el control y después toma el siguiente proceso de la cola, según su algoritmo de planificación.

Con esto consigue mantener la ilusión de **multitarea** y evita que procesos que necesiten mucho tiempo de CPU se apodere de la CPU.

Estados de un proceso

Un proceso puede pasar por varios estados. Algunos son:

- **Nuevo**: se crean las estructuras de datos para la gestión del nuevo proceso, la más importante es el bloque de control de proceso (PCB), y se le asigna un espacio en memoria.
- **Listo**: el proceso está preparado para pasar a ejecución. En este estado están los procesos que:
  - Se acaban de crear.
  - Han vuelto a la cola de listos al resolverse una operación de E/S.
  - Han terminado el tiempo asignado de CPU.
  - Han recibido la señan SIGCONT después de estar detenidos.
- **En ejecución**: el proceso se encuentra ejecutándose en la CPU.
- **Bloqueado**: un proceso se encuentra a la espera de un evento o una operación de E/S.
- **Detenido**: el proceso ha recibido una señal para que se detenga (SIGSTOP/SIGTSTP).
- **Zombie**: el proceso ha terminado de ejecutarse, pero su proceso padre no lo sabe.
- **Terminado**: el proceso ha terminado de ejecutarse y se liberan sus recursos.



Estados de un proceso

Modos de ejecución

El procesador puede ejecutar instrucciones en dos modos:

- **Modo usuario**: normalmente es empleado por los programas de usuario y las actividades no críticas del SO. El SO puede definir varios modos de usuario, diferenciados por privilegios.
- **Modo *kernel***: es empleado por el *kernel* del SO para ejecutar las instrucciones contenidas en él. Es este modo se puede obtener el control total del procesador y del sistema. En este modo se ejecutan tareas como la gestión de procesos, la gestión de memoria, la gestión de interrupciones, etc.
  - Si un programa ejecutándose en modo usuario desea realizar una operación crítica, debe solicitar dicha operación al SO, que este la acepte y la ejecute en modo *kernel*.

Modos de ejecución

Por otro lado, la ejecución de los procesos puede realizarse de varios modos:

- **Por lotes o batch**: se lanza un conjunto de tareas para realizar por el sistema y este ejecuta todas ellas, una detrás de otra, sin intervención del usuario.
  - Por ejemplo, la automatización de las copias de seguridad en un sistema, la instalación de actualizaciones automáticas, etc.
- **Interactivo**: solicitan constantemente la intervención del usuario para su continuidad.
  - Por ejemplo: un programa de diseño CAD, una hoja de cálculo, el comando bc en modo interactivo, etc.

Prioridad de un proceso

El algoritmo de planificación de Linux que determina el orden de ejecución de los procesos de la cola de procesos listos para ejecutar emplea una mezcla de algoritmos Round Robin, FIFO, prioridades, etc.

La prioridad real de cada proceso se calcula mediante varios factores, pero se puede alterar parcialmente mediante un índice llamado ***nice*** ("amable").

El valor *nice* oscila entre `-20` (máxima prioridad) y `19` (mínima prioridad), y puede ser modificada por el superusuario (`root`) o por el propietario del proceso (por defecto, solo se le permite disminuir la prioridad, no aumentarla).

El comando `nice` nos permite ver el valor *nice* por defecto (normalmente, `0`). También permite ejecutar un comando con un valor nice determinado:

```
$ nice -n 19 ./tarea_pesada.sh /srv/data/big.data
```

Información de un proceso

A un proceso se le asocia información como:

- Identificador del proceso (`pid`)
- Identificador del proceso padre (`ppid`), el proceso que lo creó.
- Usuario real (`ruser`/`ruid`) y grupo real (`rgroup`/`rgid`): el usuario (y su grupo principal) que lo lanzó.
- Usuario efectivo (`euser`/`euid`) y grupo efectivo (`egroup`/`euid`): usuario y grupo que determinan los privilegios que tiene el proceso.
  - Los usuario y grupo efectivos casi siempre coinciden con los usuario y grupo reales, pero si se ejecuta un ejecutable con alguno de los bits Set-UID/Set-GID activos, el usuario/grupo efectivo será el usuario/grupo propietario del fichero.
- Estado (`stat`)
- Valor nice (`nice`) o prioridad (`pri`)
- Ficheros abiertos
- % de CPU usada (`%cpu`) y de memoria usada (`%mem`), o tamaño de memoria usada en KB (`rss`)
- Fecha/hora de lanzamiento (`start`) y tiempo de CPU consumido (`time`).
- Comando con el que se lanzó el proceso (`comm`) o comando completo (`args`)
- Terminal asociada al proceso (`tty`)

Ver información de procesos

Los siguientes comandos nos permiten ver información sobre los procesos y recursos a tiempo real:

- `top`
- `htop`: muestra información parecida a `top` pero de forma más amigable para el usuario y con colores.

El comando `ps` es una herramienta muy potente para listar procesos del sistema:

- Sin opciones, `ps` lista los procesos que cuelgan de la sesión actual. Como mínimo, intérprete shell y ps.
- Con la opción `-e`, muestra todos los procesos del sistema.
- Con un argumento numérico, muestra la información de un proceso con ese PID.
- Con `-U`/`-G`/`-u`/`-g`, muestra los procesos de un usuario/grupo real/efectivo.
- Con `-o <columnas>`, muestra solo las columnas especificadas (ver diapositiva anterior). Se pueden poner varias separadas por comas y sin espacios.
- Con -H, muestra los procesos hijos bajo los procesos padres como una especie de árbol.

Primer y segundo plano

Cuando un usuario ejecuta un comando en un *shell*, este crea un *subshell* que ejecuta dicho comando. El usuario ha de esperar para recuperar el control del intérprete de comandos hasta que el programa termina, volviendo a mostrar el *prompt*.

Los comandos que se ejecutan de esta forma, es decir, bloqueando el proceso padre hasta que terminan, se ejecutan en **primer plano** (*foreground*).

Para evitar que el usuario (o el siguiente comando) tenga que esperar a la terminación de una tarea, existe la ejecución en **segundo plano** (*background*).

Un comando terminado en **&** se ejecutará en segundo plano, por lo que devolverá el control al usuario inmediatamente aunque el proceso abierto tarde en ejecutarse.

Trabajos

En Linux las tareas en segundo plano se denominan trabajos. Para ver los trabajos de la sesión actual:

```
jobs
```

Los trabajos tienen un PID, como cualquier proceso, y también un identificador de trabajo (por ejemplo, `[2]`), al que se puede hacer referencia en los siguientes comandos anteponiendo un `%` (por ejemplo, `%2`).

Para mandar un trabajo a primer plano:

```
fg <trabajo>
```

Para reanudar un trabajo detenido:

```
bg <trabajo>
```

Interrumpir un proceso

Cuando hay un proceso ejecutándose en primer plano y deseamos retomar el control, existen dos atajos del teclado que pueden ayudarnos:

- Ctrl+C: envía la señal SIGINT al proceso en primer plano actual. La mayoría de las veces el programa terminará, aunque algunos programas deciden ignorar esta señal.
- Ctrl+Z: envía la señal SIGTSTP al proceso en primer plano actual. La mayoría de las veces el programa se detendrá (y pasará a ser un trabajo detenido), aunque algunos programas deciden ignorar esta señal.

Señales

Para controlar la ejecución de un proceso por parte del sistema operativo o del usuario, existe el envío de **señales**. Se pueden ver todas las señales mediante `kill -l`. Las más utilizadas son:

- **SIGINT** (2): señal que usa Ctrl+C para interrumpir a un proceso. El proceso puede ignorarla.
- **SIGKILL** (9): señal para matar un proceso. No se puede ignorar.
- **SIGTERM** (15): señal para matar un proceso. El proceso puede ignorarla.
- **SIGCONT** (18): señal para reanudar un proceso detenido.
- **SIGSTOP** (19): señal para detener un proceso. No se puede ignorar.
- **SIGTSTP** (20): señal que usa Ctrl+Z para detener un proceso. El proceso puede ignorarla.

El comando `kill` se usa para mandar una señal a un proceso (por defecto envía SIGTERM). Se le puede especificar el tipo de señal mediante su nombre con SIG, sin SIG o su código:

```
$ kill -STOP 41019
$ kill -9 %1
```

Automatización de tareas

Existen principalmente dos formas de ejecutar tareas planificadas en Linux:

- Cron: ejecuta tareas periódicas. Se edita con el comando `crontab -e`. Cada usuario tiene un fichero crontab que ejecuta tareas en su nombre. Se le especifican líneas de la siguiente forma:

  ```
  m h dom mon dow command
  ```

  - Minutos, horas, día del mes, mes, día de la semana (0 domingo-6 sábado), comando (admite redirecciones y tuberías). Se pueden poner valores concretos (3), cualquier valor (*), rangos (1-7), múltiples valores (1,3,5-8), uno de cada X (4/3 es uno de cada 3 empezando por 4), etc.
- At: ejecuta tareas en una fecha y hora determinadas:
  - Para añadir una tarea, se usa el comando `at` con una hora y opcionalmente una fecha, que por defecto leerá los comandos de la entrada estándar:

    ```
    $ echo "logger '¡¡¡FELIZ AÑO 2025!!!'" | at 00:00:00 2025-01-01
    ```
  - Para ver las tareas en cola:

    ```
    $ atq
    ```
  - Para eliminar una tarea según su número de tarea:

    ```
    $ atrm <N>
    ```