Introducción a los comandos en Linux

A pesar de que todo sistema Linux cuenta desde hace años con diversos entornos de escritorio (como *GNOME*, *KDE*, *MATE*, etc.) para una interacción tan fácil e intuitiva con el usuario como la de sistemas Windows, la *interfaz de línea de comandos* (o *CLI* en Inglés), proporcionada habitualmente por la *shell* **Bash** (al igual que en el sistema *macOS*) resulta muy útil y productiva para multitud de tareas cotidianas y además proporciona una interfaz universal entre distribuciones Linux (de escritorio, de servidor, para *sistemas embebidos*) con la que siempre puede contarse y que puede usarse de forma *local* o *remota* (esta última modalidad a través de *SSH*).

Comandos *internos*, *externos* y *alias*

Debemos partir de la idea de que una *shell* de línea de comandos como **Bash** no es más que un programa del sistema operativo que, al ejecutarse, típicamente dentro de un *emulador de terminal*, espera a que escribamos nuestras *órdenes* (mal traducidas como "*comandos*", pero es ya tarde para cambiarlo) para cumplirlas (ejecutarlas) si es posible (disponibilidad de la orden, privilegios del usuario, etc.) Los dos tipos principales de *órdenes* o *comandos* que entiende la *shell* son:

- **Comandos internos**: son órdenes que forman parte de una lista interna incorporada en el propio programa de la *shell* y que una vez reconocidos por esta son ejecutados directamente sin necesidad de contar con programas externos. Dos ejemplos pueden ser `cd` y `echo`.
- **Comandos externos**: se corresponden con distintos archivos ejecutables instalados en el sistema operativo a los que la *shell* invoca cuando el usuario introduce su nombre. Dos ejemplos pueden ser `ls` y `cp`.

Además, también existe un mecanismo (y comando interno) proporcionado por la *shell*, llamado **alias** que permite al usuario personalizar los comandos externos dándoles nombres alternativos, por ejemplo para acortar el nombre original o para incluir opciones específicas de uso. Esto se logra haciendo una sustitución de una *cadena de caracteres* por otra. Dos ejemplos habituales que suelen crearse de forma predeterminada en la mayoría de sistemas Linux:

`alias ll='ls -l'`

`alias ls='ls --color=auto'`

Para saber si una orden es *interna*, *externa* o se trata de un *alias*, podemos usar el comando (interno) `type` de la siguiente forma:

```
$ type cd
cd es una orden interna del shell
$ type cp
cp is /bin/cp
$ type ls
ls es un alias de `ls --color=auto'
```

Ayuda con los comandos en la propia *línea de comandos*

Además de accediendo a la enorme cantidad de documentación técnica disponible libremente en internet, podemos obtener ayuda para aprender el correcto uso de los comandos desde la propia terminal de varias formas:

- Ejecutando `help` seguido del nombre del *comando interno*: mostrará en la terminal una serie de líneas de texto sin formato donde se incluyen la finalidad, la sintaxis de uso y diversos aspectos a tener en cuenta sobre el comando consultado.
- Ejecutando el *comando externo* en cuestión seguido de la opción `--help`: mostrará en la terminal, usando una serie de líneas de texto sin formato, el modo de empleo, la finalidad del comando y las distintas opciones soportadas con su correspondiente descripción.
- Ejecutando `man` seguido del nombre del *comando externo*, donde se visualizarán en la terminal las distintas páginas del "*manual electrónico*" del comando, con un formato más atractivo y de estructura estandarizada, con secciones como *NOMBRE*, *SINOPSIS*, *DESCRIPCIÓN* y *VER TAMBIÉN*. Para examinar las distintas páginas se usa un programa *paginador* (típicamente `less`) que permite avanzar y retroceder página con las teclas correspondientes, buscar alguna cadena de texto tecleando `/` antes de la misma y salir pulsando la tecla `q`.

Podemos usar el comando `whatis` para obtener la descripción de lo que hace un comando o utilidad externa, siempre que tenga instalado en el sistema su correspondiente manual electrónico. Ejemplo:

```
$ whatis grep
grep (1)             - print lines matching a pattern
```

Con la opción `-k` del comando `man` podemos buscar en todas las páginas de manual instaladas cierta *palabra clave* para encontrar los comandos relacionados con ella. Ejemplo:

```
$ man -k ftp
ftp (1)              - Internet file transfer program
netkit-ftp (1)       - Internet file transfer program
```

Comandos básicos

Comando	Finalidad
`cat`	Concatenar el contenido de archivos y volcarlo a la *salida estándar*
`cd`	Cambiar el *directorio de trabajo*. Si no se indica destino, cambia al *directorio personal*
`clear`	Limpiar la pantalla de la terminal
`cp`	Copiar archivos y directorios. La opción `-r` permite copia *recursiva*
`less`	Paginador (sustituto de `more`) para examinar el contenido de archivos de texto
`ls`	Listar el contenido de directorios. La opción `-l` lista detalles y `-a` lista archivos ocultos
`mkdir`	Crear directorios. La opción `-p` crea toda la ruta indicada
`mv`	Mover (o renombrar) archivos y directorios
`nano`	Editar el contenido de archivos de texto
`rm`	Eliminar archivos o directorios. La opción `-r` permite eliminación *recursiva*

Rutas *absolutas* y *relativas*

Todo archivo o directorio (carpeta) tiene una ubicación dentro del árbol de directorios, que puede indicarse mediante el *camino* o *ruta* (en inglés *path*) hasta el mismo. Esta ruta se expresa como la secuencia de directorios, separados por el carácter "**/**", que hay que atravesar desde el directorio raíz (que en Linux se llama "**/**") hasta el archivo o directorio en cuestión. Por ejemplo, la ruta del archivo `ManualDebian.pdf`, ubicado en la carpeta `Descargas` del usuario Torvalds, sería `/home/torvalds/Descargas/ManualDebian.pdf`. A este tipo de rutas que empiezan desde el directorio raíz se les llama *rutas absolutas* para distinguirlas de otro tipo puede usarse de forma alternativa en la línea de comandos cuando su longitud (y el tiempo para escribirlas) es menor. Se trata de las *rutas relativas*, que se expresan como el camino hasta el archivo o directorio pero partiendo del directorio actual o directorio de trabajo (en inglés *working directory*), que es donde está situado el usuario en ese momento en la *shell* de línea de comandos. Por ejemplo, si el usuario del ejemplo anterior estuviera situado en su directorio personal (`/home/torvalds`)y quisiera mover el archivo referido a su carpeta `Documentos` le sería más fácil y rápido teclear el comando `mv Descargas/ManualDebian.pdf Documentos` que la alternativa usando *rutas absolutas*: `mv /home/torvalds/Descargas/ManualDebian.pdf /home/torvalds/Documentos` por razones obvias.

Caracteres *comodín* y su expansión por parte de la *shell*

La *shell Bash* (típicamente ubicada en `/bin/bash`) reconoce ciertos caracteres como "especiales" y los maneja de forma diferente. Los *caracteres comodín* son una categoría de *caracteres especiales* que se usan mucho a la hora de trabajar con archivos y carpetas, ya que pueden sustituirse por cualquier carácter, igual que una carta comodín puede sustituirse por cualquiera de la baraja. Si queremos usar el comando `ls` para listar los detalles (permisos, tamaño, etc.) de todos los archivos del directorio `/etc` con extensión `.conf` escribiríamos la línea de comandos `$ ls -l /etc/*.conf`, usando el carácter `*` (llamado

asterisco) para indicarle a la *shell* que en esa posición puede ir cualquier cantidad y combinación de caracteres. Conviene señalar que el asterisco también se puede sustituir por el *carácter nulo*.

Otro carácter comodín muy usado es `?`, que la *shell* sustituye por cualquier carácter pero en esa posición específica, un solo carácter. Como ejemplo podríamos listar todos los archivos de audio con extensiones de cuatro caracteres (por ejemplo **.opus** y **.flac**) de la carpeta `Música` con la línea de comandos `ls -l Música/*.????`, mientras que con el comando `ls -l Música/*.m??` listaríamos los archivos, dentro de la misma carpeta, que tengan cualquier nombre seguido de un punto, una eme y después dos caracteres más, en cualquier combinación (pero excluyendo el *carácter nulo*). Este patrón listaría archivos **.mp3** y archivos **.m4a** pero no **.wma** ni **.ogg** (ni por supuesto archivos con extensiones de menos ni de más de tres caracteres).

Es importante aclarar que la *sustitución* o *expansión* de los caracteres *comodín* por los caracteres *literales* que encajen en el "patrón" la realiza la propia *shell* antes de pasarle la lista de los archivos coincidentes al comando correspondiente (en el ejemplo anterior `ls`), por lo que pueden usarse con cualquier comando que trabaje con múltiples archivos o directorios. Si la operación a realizar es delicada, como un borrado masivo de archivos cuyo nombre siga cierto patrón, podemos "previsualizar" la expansión que la *shell* hará de los comodines, y por tanto la lista final de archivos "seleccionados" usando el comando `echo`, por ejemplo `echo informe*.???` antes de borrarlos con el comando `rm informe*.???`.

Buscando e identificando archivos

Para buscar un archivo o directorio por todo el sistema de archivos podemos usar dos comandos:

- El programa `find` hace una búsqueda *recursiva* bajando por toda la "rama" elegida (si partimos de la raíz será por todo el árbol). Además permite buscar usando distintos criterios relacionados con el archivo o archivos a buscar, aunque típicamente será por nombre o parte del mismo. Un ejemplo:

  `$ find /etc -name '*.conf'` Buscará, partiendo del directorio `/etc` todos los archivos con cualquier nombre pero de extensión `.conf`. Si usamos comodines como `*` debemos protegerlos de la expansión de bash usando las comillas simples. Si quisiéramos "relajar" la búsqueda para que no se tenga en cuenta la diferencia entre mayúsculas y minúsculas podríamos usar `-iname`.

- El programa `locate` busca los archivos en una base de datos (que se actualiza diariamente) en lugar de recorriendo el sistema de archivos, por lo que es más rápido que `find`, aunque menos potente en cuanto a criterios de búsqueda y además podría no darnos resultados por no encontrarse actualizada su base de datos (que podríamos actualizar manualmente ejecutando `sudo updatedb`). Se usa básicamente indicando el nombre (o una parte del mismo) del archivo o directorio a buscar. Un ejemplo:

  `$ locate samba` Mostraría las *rutas completas* hasta archivos o directorios que contengan (en cualquier parte de la *ruta*, tanto en los nombres de directorios como en el nombre del archivo) la cadena de texto "samba".

Como en el mundo Linux no es obligatorio que los archivos tengan una extensión, aunque puede formar parte de su nombre, o símplemente podemos toparnos con archivos de extensión desconocida, puede sernos muy útil el comando `file`, ya que es capaz de identificar miles de tipos de archivos realizando una serie de pruebas. Por ejemplo, si en su día realizamos una descarga no del todo exitosa desde nuestro navegador web, que nos dejó un anodino archivo llamado `download` de unos 3 MB de tamaño, y no recordamos de qué tipo de archivo se trataba, simplemente ejecutaríamos:

```
$ file download
download: PDF document, version 1.4
```

Archivos comprimidos

En Linux existen comandos y utilidades para manejar todo tipo de archivos, incluyendo los archivos comprimidos, algunos de ellos bien conocidos en sistemas Windows y otros específicos del mundo Unix/Linux. Por supuesto, también hay aplicaciones y utilidades gráficas disponibles en cualquier *distribución* Linux y *entorno de escritorio* para gestionar este tipo de archivos, pero vamos a centrarnos en los programas de línea de comandos.

Para descomprimir un archivo **.zip** disponemos del comando `unzip`, que en su forma más sencilla (y habitual) se ejecutaría de la siguiente forma: `$ unzip archivo_comprimido.zip` para extraer el contenido del archivo comprimido al directorio actual. Si lo que queremos es comprimir una serie de ficheros e incluirlos todos en un único archivo **.zip** para, por ejemplo, mandarlo por e-mail a un compañero, haríamos lo siguiente: `$ zip *.mp3 canciones.zip` (estando posicionados previamente en el directorio donde se encuentran, claro).

Sin embargo, en sistemas Linux, la compresión más usada la proporciona el programa `gzip`, que usa el algoritmo LZ77 para reducir el tamaño de los ficheros procesados, los cuales sustituye uno a uno por una copia comprimida a la que se añade la extensión **.gz** , en lugar de "empaquetarlos" todos juntos en un archivo comprimido.

Para "empaquetar" o "archivar" múltiples ficheros en un solo archivo se utiliza el venerable comando `tar`, usado durante décadas en el mundo Unix para hacer copias de seguridad. Si además se quiere comprimir el archivo obtenido para que ocupe menos espacio en disco, se usa la opción z que lo pasará por el compresor `gzip` introducido más arriba. Las dos acciones típicas suelen ser:

- Extraer el contenido de un archivo de extensión **.tar.gz** (o la equivalente **.tgz**): `$ tar xzvf archivo.gz` teniendo en cuenta que la **x** significa *extraer*, la **z** comprimir con *gzip*, la **v** significa *verboso*, "abundante en palabras" (que muestre información) y la **f** indica al programa que a continuación tiene el nombre del archivo (*file*) a extraer.
- Archivar (y comprimir) una serie de ficheros en un *archivo tar*: `$ tar czvf archivo.gz fichero1 fichero2` teniendo en cuenta que la **c** significa *crear*, la **z** comprimir con *gzip*, la **v** significa *verboso*, "abundante en palabras" (que muestre información) y la **f** indica al programa que a continuación tiene el nombre del archivo (*file*) a crear. Es **importante** observar que el programa **tar** puede archivar también directorios, y que lo hace de forma *recursiva*, es decir, manteniendo la estructura original de la rama de directorios. Como ejemplo, podríamos archivar nuestra colección de música MP3 descargada, con las distintas carpetas y subcarpetas conteniendo dicho tipo de archivos, creando con **tar** el archivo **MúsicaMP3.tgz** de la siguiente forma: `$ tar czvf MúsicaMP3.tgz Descargas/MP3`

Tuberías y *filtros*

Las **tuberías** de comandos consisten en combinar múltiples comandos, de forma que la *salida* (la información que muestra) de cada uno se canaliza hacia la *entrada* del siguiente. El símbolo utilizado es "**|**", llamado *pipe* o "tubería", y cuando se escribe entre los nombres de comandos o programas en una sola línea, la *shell* (Bash en nuestro caso) interpreta que debe conectarlos para formar una *tubería*:

`$ cat /etc/passwd | grep operador | cut -d: -f7`

En el ejemplo anterior, el comando `cat` lee el contenido del archivo de texto `/etc/passwd` que contiene la información sobre las cuentas de usuario, pero en lugar de volcarlo a la pantalla, el uso de la tubería hace que todo ese texto se le pase al siguiente comando, `grep` que se utiliza para *filtrar* líneas de texto, en este caso aquellas en que aparezca la palabra "operador" (un supuesto nombre de usuario). La línea resultante, en lugar de aparecer por pantalla, vuelve a "entubarse" con otro comando, `cut`, especializado en *filtrar* columnas, en este caso la séptima (*field 7*) usando como *delimitador* de las mismas el caracter ":".

Este mecanismo de *conexión de programas*, permite combinar comandos y programas existentes, como si fueran *piezas de lego*, para realizar nuevas tareas, y resolver problemas concretos. Además de usar estas *tuberías* en la línea de comandos, si la guardamos en un archivo para usos posteriores tendremos la base de un típico *script*.

grep, cut, sort

Redireccionamiento de entrada y salida

Una alternativa que nos ofrece una *shell* como Bash es la de desviar los datos de un comando, en vez de a otro comando como en el caso de las *tuberías*, a un archivo. A esto se le llama **redirección de la salida estándar** y para usarla se utiliza el símbolo "**>**" situado entre el nombre del programa y el del archivo de destino:

```
$ mount > discos_montados.txt
```

Si el archivo ya existe, se sobreescribirá su contenido con el nuevo generado por el comando. Si no deseamos esto, sino que preferimos que se añada al final, podemos duplicar el símbolo "**>**" de la siguiente forma:

```
$ mount >> discos_montados.txt
```

Un archivo especial que suele usarse en *scripts* para deshacernos de los datos de salida generados por un comando es `/dev/null` conocido como el *sumidero de bits*, porque todos los datos que se le envían, se pierden.

También podría desviarse el contenido de un archivo hacia la entrada de datos de un comando utilizando el símbolo opuesto "**<**" (ver ejemplo de *bucle while* más abajo). A esto se le llama **redirección de la entrada estándar**.

Información básica del sistema

La información sobre la cantidad de **memoria RAM libre** es básica en cualquier sistema operativo, y podemos obtenerla desde la terminal de cualquier sistema Linux ejecutando: `$ free -h` donde la opción ***h*** le indica al programa que debe mostrar la información en formato "para humanos", es decir, usando los múltiplos del *byte* más adecuados, como *megas* y *gigas*...

Otro recurso preciado en un sistema operativo es el **espacio en disco**, y el comando que permite obtener información sobre los distintos sistemas de archivos en uso y su espacio libre y ocupado sería: `$ df -h` (**df** de *disk free*) donde se usa de nuevo la opción **h** para usar las unidades de medida más convenientes.

Un comando complementario a `df` es el comando `du` (*disk usage* o "uso de disco") que permite saber el espacio ocupado por los distintos archivos dentro de cierta carpeta. Por ejemplo: `$ du -h Descargas` nos mostraría el desglose de la ocupación, archivo a archivo y descendiendo por las subcarpetas existentes, del contenido de la carpeta `Descargas`. Si solo nos interesa saber la suma total: `$ du -hs Descargas`

Para saber la ocupación de CPU por parte de los procesos (programas en ejecución) en un determinado momento, tenemos el programa intectivo `top` que listará el *ranking* de los procesos con más consumo de CPU, actualizando y reordenando la lista cada pocos segundos para que aparezcan arriba los procesos más costosos. Además nos muestra toda una serie de datos sobre el funcionamiento del sistema que pueden resultar de utilidad a administradores de sistemas y usuarios avanzados. Para salir debemos pulsar la tecla `q` .

Gestión de particiones y sistemas de archivos

Para particionar discos duros mediante el esquema clásico basado en el MBR o mediante el nuevo estándar GPT podemos usar la utilidad `fdisk`, aunque si disponemos de un entorno gráfico, como es normal en *Ubuntu Desktop* o *Linux Mint* lo mas recomendable es usar *GParted*. En cualquier caso, puede ser útil ejecutar desde la terminal `$ sudo fdisk -l` en ciertas ocasiones para listar los dispositivos de almacenamiento (discos duros o SSDs internos, discos y pendrives externos, etc.) con las particiones que contienen. Típicamente la nomenclatura Linux es del tipo `/dev/sda1` donde **sd** se refiere a disco SATA, la letra **a** al primer disco físico conectado en el sistema y el número **1** a la primera partición del disco.

Para "*formatear*", es decir, para crear un *sistema de archivos* vacío en cierta partición, se utilizan los comandos `mkfs`, dependiendo del tipo de sistema de archivos a crear, por ejemplo `mkfs.ext4`, `mkfs.ntfs` o `mkfs.fat`, indicando la partición donde crearlo y ejecutando con `sudo` para ganar los privilegios necesarios: `$ sudo mkfs.fat /dev/sdc1` serviría para formatear la partición principal (y normalmente única) de un *pendrive*.

Para acceder al contenido de los *sistemas de archivos* disponibles en los *dispositivos de almacenamiento* presentes en el equipo, hay que "*montarlos*", es decir, conectarlos en un "*punto de montaje*", que será normalmente un directorio vacío donde se conectará el árbol de directorios del sistema de archivos recién montado. Aunque los entornos de escritorio se encargan de montar automáticamente los discos y pendrives detectados, podría ser necesario hacerlo manualmente con el comando `mount` de la siguiente forma: `$ sudo mount /dev/sdc1 /mnt` donde primero se indica la partición a montar y finalmente el directorio donde hacerlo, en este caso un punto de montaje estándar presente en cualquier sistema Linux.

Si se ejecuta el comando `mount` "a secas", se listan los sistemas de archivos montados actualmente.

Si alguna vez necesitamos comprobar los errores de un sistema de archivos, por ejemplo por haberlo extraído sin desmontarlo previamente (se usaría el comando `umount`) o por un apagón brusco del equipo, el comando a tene en cuenta es `fsck`, al que se le indica el sistema de archivos a comprobar y alguna opción que indique qué hacer con los errores detectados (si los encontrara), típicamente que los repare automáticamente con la opción `-a`: `$ sudo fsck -a /dev/sdb1`.

Gestión de software

La forma más fácil y al mismo tiempo estándar, disponible en distribuciones derivadas de Debian como Ubuntu, para actualizar, instalar y desinstalar *software* desde la línea de comandos, usando el sistema ***APT*** de gestión de paquetes, es usar el comando `apt` con las distintas acciones a realizar:

Comando	Finalidad
`apt update`	Actualizar la base de datos local de software disponible
`apt upgrade`	Actualizar el software instalado si hay nuevas versiones
`apt search <nombre_programa>`	Buscar el paquete correspondiente a un programa
`apt install <nombre_paquete>`	Instalar el paquete indicado
`apt remove <nombre_paquete>`	Desinstalar el paquete indicado

Si se descarga manualmente un paquete en forma de archivo **.deb** se puede instalar desde la línea de comandos con la utilidad (de más bajo nivel que `apt`) `dpkg`: `$ sudo dpkg -i archivo_paquete.deb`

Gestión de usuarios y grupos

La forma más cómoda de crear nuevas cuentas de usuario o añadir una cuenta de usuario existente a un grupo es usando el script interactivo `adduser`:

`$ sudo adduser nuevousuario` preguntará en pantalla los datos del nuevo usuario (incluyendo su contraseña que se guardará cifrada en `/etc/shadow`) y creará la cuenta de usuario (en `/etc/passwd`), su *grupo personal* (en `/etc/group`) y su *directorio personal* (en `/home/nuevousuario`).

`$ sudo adduser nuevousuario grupo` podría usarse para añadir al usuario al grupo indicado. Por ejemplo podríamos añadir a un usuario al grupo *sudo* para que pueda usar el comando `sudo` para elevar privilegios en el sistema.

Además existen los comandos *no interactivos* (hay que indicarles opciones de uso, lo que dificulta su uso pero los habilita para su utilización en *scripts*) `useradd`, `usermod`, `userdel` para, además de crear cuentas de usuario, modificarlas y borrarlas.

Gestión de permisos y propietarios de archivos

En sistemas Linux, los permisos de los archivos, que pueden listarse usando `ls -l`, se pueden modificar con el comando `chmod`, usando una notación *numérica* o una notación *simbólica*. La simbólica es la más fácil de usar y consiste en indicar los *permisos* (**r**: lectura, **w**: escritura y **x**: ejecución) a *dar* (con **+**) o *quitar* (con **-**) al *usuario propietario* (**u** de *user*), al *grupo propietario* (**g** de *group*) y al resto de usuarios del sistema (**o** de *others*). Esta forma de uso es *relativa* a los permisos actuales del archivo, "tocando" (para dar o quitar) permisos concretos, lo que dejaría al resto como estuvieran. También se puede usar de forma *absoluta* usando el signo "**=**" para indicar exáctamente cuáles deben ser los permisos de cada conjunto de usuarios, dando los permisos mencionados y quitando los quen o aparecen. Algunos ejemplos:

Comando	Finalidad
`chmod +x backup.sh`	Da perm. de ejecución a los tres conjuntos de usuarios (al no concretar)
`chmod g-w,o-r f1`	Quita perm. de escritura al grupo prop. y el de lectura a otros
`chmod u+w,go-r f1`	Da perm. escritura a propietario y quita lectura a grupo y otros
`chmod u=rw,g=r,o= f1`	Da a propietario **r** y **w**, al grupo solo **r** y al resto nada

Es justo aclarar que solo el propietario de un archivo, o el *superusuario* puede cambiar sus permisos.

Para cambiar el *propietario* y/o *grupo propietario* de un archivo, podemos usar el comando `chown`, del siguiente modo: `$ sudo chown -R www-data:www-data /var/www/html`. Solo el *superusuario* puede cambiar la propiedad de un archivo, y en el ejemplo se usa además la opción `-R` para cambiar *recursivamente* la propiedad de todos los archivos a partir del directorio indicado.

Gestión de procesos

El comando `ps` sirve para listar los procesos en ejecución. Para que liste todos los procesos (de todos los usuarios) y muestre información sobre los mismos suele usarse con opciones: `$ ps -ef` .

A veces puede ser necesario "matar" un proceso, y para ello se usa el comando `kill`, indicando el *PID* (*identificador del proceso*) y la *señal* a enviarle. Si no se especifica ninguna señal, se manda la de *terminación*, que podría ser ignorada por el proceso por distintos motivos, incluyendo que este se encuentre "*colgado*". En ese caso podemos enviar la señal 9 o SIGKILL, que fuerza la "muerte" del mismo. Ejemplos:

`$ kill 1234` le mandaría la señal de finalización al proceso con PID 1234.

`$ kill -9 4321` le mandaría la señal de "muerte forzosa" al proceso con PID 4321.

Para listar el *ranking* de los procesos que más CPU están consumiendo en un determinado momento, así como diversa información del funcionamiento del sistema (tiempo encendido, total de usuarios conectados, memoria libre, total de procesos, etc.) puede usarse el programa `top`, que irá actualizando la información cada pocos segundos hasta que salgamos pulsando `q`.

Gestión de servicios

Los servicios son *procesos* (programas en ejecución) que corren en *segundo plano* y que prestan algún servicio a otros procesos locales o remotos. Tienen especial importancia en servidores, aunque podemos encontrarlos en menor número en sistemas de escritorio, y desde hace un tiempo son gestionados por el proceso **systemd**, al que podemos controlar con el comando `systemctl` indicando las acciones a realizar:

`systemctl -t service` lista en pantalla los servicios disponibles, indicando su estado de ejecución.

`systemctl start|stop|restart <servicio>` con la acción correspondiente, inicia, detiene o reinicia respectivamente el servicio indicado al final.

`systemctl enable|disable <servicio>` con la acción correspondiente, habilita (configura para ejecutarse en el arranque del sistema) o deshabilita (no se ejecutará) el servicio indicado.

Configuración y diagnóstico de red

El comando `ifconfig` sin más opciones lista las *interfaces de red* con su configuración actual, incluyendo la *dirección IP* y la *máscara de subred*. Se puede usar para cambiar temporalmente (en memoria RAM) la configuración de las interfaces para hacer pruebas o conectar momentáneamente a otras redes, de la siguiente forma: `$ sudo ifconfig eth0 192.168.2.10`. Hay que tener en cuenta que los cambios permanentes en la configuración de red se hacen editando el archivo `/etc/network/interfaces` o, si se usa *NetworkManager* en el sistema (lo normal con entorno de escritorio), con sus pantallas/archivos de configuración correspondientes.

El comando `ping` es básico para diagnosticar problemas de conectividad de red, pues permite enviar paquetes hacia un *host* de destino que deberá devolvérnoslos (si no se lo impide algún *cortafuegos*) indicando que está activo. Además, por el tiempo de retorno de los paquetes, podremos hacernos una idea de la "calidad" de la conexión. La comprobación puede ser con *hosts* internos de nuestra red (por ejemplo, para ver si responde el *router* o está caído) o con *hosts* externos que sabemos suelen estar *online*. Por ejemplo, si ejecutamos `$ ping 8.8.8.8` estamos comprobando que podemos llegar hasta el servidor DNS de Google, lo cual nos indicará que "tenemos salida a internet". Si en lugar de una IP usamos un nombre de dominio así `$ ping google.com` estaremos probando además de la conectividad a nivel IP, el correcto funcionamiento de la *resolución de nombres*, o lo que es lo mismo, que los servidores DNS que nos permiten traducir nombres de *hosts* en *direcciones IP*, están haciendo su trabajo.

Si necesitamos profundizar en el diagnóstico de nuestra conexión, en lugar del clásico `traceroute` que no suele estar preinstalado en muchas distribuciones Linux, podemos usar `mtr`, que trazará la ruta seguida por los paquetes de prueba hacia el *host* de destino. Esto permitiría encontrar en qué punto (*router*) se queda "atascado" el tráfico.

Programación de tareas

El equivalente del *Programador de tareas* de los sistemas Windows lo encontramos en forma del servicio `cron`, que se encarga de revisar periódicamente las tareas programadas para su ejecución en el archivo `/etc/crontab`, o de forma más fácil, de los *scripts* contenidos en los directorios `/etc/cron.hourly`, `/etc/cron.daily`, `/etc/cron.weekly` que se lanzarán automáticamente con la frecuencia establecida en el propio nombre de cada directorio.

Además de ese servicio para programar tareas a nivel del sistema operativo (solo el *superusuario* puede utilizarlo), cada uno de los usuarios dispone de su propia *tabla cron*, que puede listar con el comando `crontab -l` y editar (con el editor `nano` de forma predeterminada) usando el comando `crontab -e`. Para documentarnos sobre la sintaxis a utilizar en dichos archivos de configuración podemos consultar su propia página de manual con el comando `$ manual 5 crontab`.

Conexión y copia remota con SSH

En cualquier sistema Linux suele contarse con el cliente OpenSSH, que podemos ejecutar de la siguiente forma: `$ ssh usuario@dirección_o_nombre_host`. Esto nos permite iniciar sesión con cuentas de usuario disponibles en sistemas remotos de forma segura (el protocolo *SSH* usa cifrado para ello) a través de nuestra red local o incluso de internet. En el otro extremo simplemente deberá estar instalado el servidor OpenSSH, lo que puede hacerse mediante el comando `$ sudo apt install openssh-server`.

Para copiar archivos de forma segura entre nuestro equipo y uno remoto que tenga instalado y activo el servicio *SSH* podemos usar el comando `scp` de la forma siguiente:

`$ scp archivo.ext usuario@dirección_o_nombre_host:~/Descargas` donde se indica el archivo o archivos a copiar (pueden usarse comodines) y el directorio de destino en el host remoto, en elste caso el directorio `Descargas` del usuario "*usuario*".

Apagado y reinicio del sistema

Aunque con el comando `shutdown` y sus distintas opciones se pueden gestionar todas las situaciones de apagado y reinicio del sistema es más fácil de recordar y usar que podemos apagar un equipo con `poweroff` y podemos reiniciarlo con `reboot`.