Trabajo Autónomo

1. Como actividad autónoma, se debe tener ya instalado y configurado el sistema operativo en el virtual box, que se realizo en la clase anterior.
2. Abrir la terminal de Linux y ejecutar los comandos relacionados acontinuación.

Para interactuar con el shell se utilizan comandos con los que se inician los **programas de líneas de comandos**. Para cualquier acción que se quiera ejecutar en el terminal se escribe una orden con el siguiente esquema básico:

BEFEHL [OPTIONEN] [ARGUMENTE]

Para iniciar un programa en el terminal se necesita su nombre (comando) y la posibilidad de invocar ciertas funciones a través de las “**opciones”**. Por último, si un programa espera “**argumentos**” (en forma de archivos o rutas de directorio, p. ej.), estos suelen introducirse tras las opciones elegidas.

A continuación, te mostramos cuáles son los **comandos de Linux más habituales** y los programas que se inician con ellos.

Comandos básicos de Linux

En esta categoría se incluyen todas las órdenes fundamentales que se utilizan para **administrar el terminal**, con las que se puede limpiar la ventana de la consola, recuperar comandos anteriores desde la historia o finalizar la sesión.

|  |  |
| --- | --- |
| **Comando** | **Descripción** |
| **clear** | **Limpia la ventana del terminal**  Con el comando *clear* se borran todos los comandos de la sesión.  *clear*  El usuario recibe un terminal vacío con el prompt a la espera de órdenes. Los comandos introducidos anteriormente se guardan en el scrollback buffer.  También se puede limpiar la ventana de la consola con el atajo de teclado [CTRL] + [L] |
| **exit** | **Finaliza la sesión en el terminal**  Con el programa*exit* finaliza la sesión actual y se cierra la consola.  *exit*  También se puede utilizar el atajo de teclado [CTRL] + [D]. |
| **help** | **Muestra la lista de todos los comandos**  El comando *help* se utiliza para mostrar una lista de todas las órdenes del sistema.  *help*  Si introduces *help* en combinación con un comando shell obtienes una breve descripción del comando en cuestión.  *help COMANDO* |
| **history** | **Muestra las entradas del historial**  En Bash, los últimos 500 comandos introducidos en el terminal se guardan en el historial. Esta función sirve de ayuda al introducir órdenes en la consola y permite ejecutar de nuevo un comando introducido con anterioridad seleccionándolo con ayuda de las flechas del teclado y confirmándolo con la tecla Enter.  También se puede examinar el historial en función de palabras clave con el atajo de teclado [CTRL] + [R]. Utilizando la orden sin opciones ni argumentos se obtiene la lista completa de comandos numerada.  *history*  Para filtrar la salida se combina *history* con el programa *grep* (ver opciones de búsqueda) y una palabra clave utilizando la barra vertical.  *history | grep PALABRA CLAVE* |

Páginas de ayuda

¿No sabes cuáles son los pasos siguientes? A este respecto, Linux ofrece en el terminal **páginas de ayuda y de documentación** como las Unix Man Pages (Manual de Unix) y las páginas de información de GNU*,*las cuales contienen una exhaustiva descripción de todos los programas, comandos, archivos de configuración, formatos de archivo y funciones del kernel. Con *whatis* y *apropos* en la categoría de páginas de ayuda, cuentas con programas de líneas de comando que permiten examinar las páginas del manual del sistema operativo por palabras clave.

|  |  |  |
| --- | --- | --- |
| **Comando** | | **Descripción** |
| **apropos** | **Examina el manual**  Con *apropos* se examinan los títulos y las descripciones de las páginas del manual de tu sistema operativo por palabras clave. El programa de líneas de comandos muestra en el terminal todas las correspondencias incluyendo una breve descripción.  Sigue el siguiente esquema:  *apropos [OPCIONES] PALABRA CLAVE*  Este comando soporta diferentes opciones. Con la opción *-e(--exact*) delimitas la búsqueda a correspondencias exactas, pero también puedes utilizar comodines (*-w '\*TÉRMINO DE BÚSQUEDA'*) y expresiones habituales (*-r*). | |
| **info** | **Abre la documentación en línea de GNU**  El comando *info* abre las páginas die información de GNU sobre un tema determinado. Normalmente equivalen al manual que se puede invocar con *man*, aunque a diferencia de este, contienen enlaces que facilitan la navegación en el manual.  Para abrir la documentación de lectura de GNU utiliza esta sintaxis:  *info [OPCIÓN] TEMA*  Si no indicas una opción ni un tema accedes al menú principal del manual de GNU. | |
| **man** | **Abre el manual**  Con *man* se abren las páginas del manual (man pages) de tu distribución de Linux en la consola.  Para ello se sigue el siguiente esquema:  *man [OPCIÓN] TEMA*  Las man pages de Linux se distribuyen en 10 secciones temáticas:      (1) Comandos de usuario      (2) Llamadas al sistema      (3) Biblioteca de funciones del lenguaje de programación C      (4) Formatos de archivo      (5) Archivos de configuración      (6) Juegos      (7) Miscelánea      (8) Comandos para administrar el sistema      (9) Funciones del kernel      (n) Comandos nuevos  Si, por ejemplo, quieres abrir el manual para un comando específico, se combina *man* con el nombre del comando:  *man clear*  O se concreta la búsqueda con el número de la sección correspondiente:  *man 1 clear*  En ambos casos se abre la página del manual por el comando *clear*. Para cerrarlo y volver a la consola utiliza la tecla [Q].  Si quieres explorar las páginas del manual de Linux por palabras clave dispones del comando *apropos*. | |
| **pinfo** | **Abre la documentación en línea al estilo Lynx**  Con *pinfo* cuentas con una versión del programa *info* que sigue el modelo del explorador de líneas de comandos Lynx y muestra las páginas de información con los enlaces destacados con colores.  Para utilizar *pinfo* se sigue el mismo esquema sintáctico que con *info*:  *pinfo [OPCIONES] TEMA* | |
| **whatis** | **Busca en el manual por palabras clave**  El programa *whatis* permite buscar por palabras clave en el manual. Para examinar el manual del sistema operativo en busca de una equivalencia exacta se llama al programa con el término de búsqueda. Si se encuentra una correspondencia, *whatis* responde con una breve descripción en el terminal.  *whatis [OPCIONES] PALABRA CLAVE*  *whatis* (*-w '\*PALABRA CLAVE'*) también soporta comodines y expresiones regulares (*-r*). | |

Operaciones en el directorio

Algunos comandos básicos de Linux permiten llevar a cabo operaciones en los directorios del sistema, como puede ser crear ficheros, borrarlos y gestionarlos, así como navegar por el árbol del directorio. Entre los comandos más importantes de esta categoría se incluyen *cd*, *ls*, *mkdir* o *rmdir*.

|  |  |
| --- | --- |
| **Comando** | **Descripción** |
| **cd** | **Navega por el árbol de ficheros**  El comando *cd* es la abreviatura de *change directory* y se utiliza para navegar por el directorio.  La sintaxis de esta orden sigue el esquema:  *cd [OPCIÓN] DIRECTORIO*  Si no se indica ningún directorio concreto, *cd* cambia automáticamente al directorio principal del usuario.  Si se acompaña de un guion (-), *cd* abre el directorio precedente. |
| **chroot** | **Ejecuta el programa en un nuevo directorio raíz**  La orden*chroot* (abreviatura de *change root*) se utiliza para ejecutar un comando en un directorio raíz diferente para, por ejemplo, aislar programas peligrosos del sistema de archivos restante, lo que se denomina *chroot jail*.  Este comando requiere derechos raíz y se orienta por este esquema:  *chroot DIRECTORIO COMANDO* |
| **ls** | **Muestra el contenido del directorio como una lista**  La orden*ls* equivale a *list* y se utiliza para mostrar el contenido de un fichero (los nombres de todos sus archivos y carpetas).  Esta es la sintaxis de la orden:  *ls [OPCIONES] DIRECTORIO*  Si no se añade a *ls* ningún directorio, el comando enumera el contenido del directorio en curso.  Con ayuda de diferentes opciones se puede definir qué información se ha de mostrar y cómo. |
| **mkdir** | **Crea un directorio**  El comando *mkdir* corresponde a *make directory*y permite a los usuarios de Linux crear directorios desde cero.  Para crear un directorio en el fichero en curso escribe la siguiente sintaxis:  *mkdir [OPCIÓN] DIRECTORIO*  Si lo que se necesita es crear varios ficheros a la vez, se escribe uno detrás de otro sin signos de puntuación y con espacio intermedio:  *mkdir [OPCIÓN] DIRECTORIO1 DIRECTORIO2*  Si se quiere crear un directorio nuevo en otro fichero diferente al actual se ha de indicar la ruta absoluta o relativa al fichero:  mkdir /home/user/Desktop/test  *mkdir/Desktop/test*  En ambos ejemplos se crea el directorio *test* en el directorio *Desktop*. |
| **mkdirhier** | **Crea una jerarquía en el directorio**  Con *mkdirhier* puedes crear jerarquías completas con un único comando:  *mkdirhier [OPCIÓN] /home/user/directorio1/directorio2/directorio3*  Si *directorio1* y *directorio2* ya existieran, entonces *mkdirhier*solo crearía *directorio3*. En caso contrario, se crean los tres. |
| **pwd** | **Muestra el nombre del directorio**  Con *pwd* (abreviatura de *print working directory*) la consola muestra el nombre del directorio de trabajo (en curso).  La sintaxis del comando es:  *pwd [OPCIONES]* |
| **rmdir** | **Borra el directorio**  Si hiciera falta borrar un determinado directorio, se utiliza el comando *rmdir* (*remove directory*) según la siguiente sintaxis:  *rmdir [OPCIÓN] DIRECTORIO*  Con *rmdir,*sin embargo,solo se pueden borrar directorios vacíos. Si se quiere eliminar un fichero con todas sus carpetas y subcarpetas se utiliza el comando*rm* (*remove*) con la opción *-r*.  **Cuidado:***rmdir*no solicita confirmar el borrado. Los directorios seleccionados con el comando se borran definitivamente. |
| **tree** | **Presenta los directorios en forma de árbol**  Mientras que *ls* muestra el contenido de los directorios como lista, el comando *tree,*siguiendo este esquema sintáctico*,*muestra la jerarquía completa del directorio en forma de árbol:  *tree [OPCIONES] [DIRECTORIO]* |

Operaciones en archivos

Los comandos Linux de esta categoría permiten llevar a cabo diversas operaciones desde el terminal que atañen a los archivos y, utilizando comandos básicos como *cp*, *mv* y *rm,* se pueden copiar, desplazar, renombrar o borrar archivos del sistema.

|  |  |
| --- | --- |
| **Comando** | **Descripción** |
| **basename** | **Muestra el nombre del archivo**  Al comando *basename*se le indica una ruta a un archivo y devuelve su nombre sin la ruta.  La sintaxis del comando se compone de:  *basename [OPCIONES] Ruta/al/archivo [SUFIJO]*  Si en el terminal escribes, por ejemplo,*$ basename /home/user/imagen.jpg*, obtienes la siguiente respuesta:  *imagen.jpg*  Si se añade el sufijo de nuevo, el terminal no lo devuelve, como vemos aquí:  Entrada:*$ basename /home/user/imagen.jpg .jpg*  Salida: *imagen*  El comando puede ser ampliado a varios archivos con diferentes opciones. |
| **cat** | **Agrupa contenido de varios archivos**  El comando *cat* (de *concatenate*) nace como herramienta para enlazar archivos y puede emplearse como pager para mostrar el contenido de los archivos en el terminal.  Escribe *cat* con la siguiente sintaxis para leer y mostrar un archivo en la salida estándar:  *cat OPCIONES ARCHIVOS*  Si incluyes varios archivos, sepáralos mediantes espacios:  *cat OPCIONES ARCHIVO1 ARCHIVO2*  Para concatenar el contenido de varios archivos se utilizan los operadores de redirección >, < y |. Si utilizas el operador “mayor que“ (>), se aúna el contenido de dos archivos en un tercero:  *cat archivo\_1.txt archivo\_2.txt > archivo\_3.txt* |
| **cmp** | **Compara archivos a nivel de byte**  *cmp*forma parte del paquete *diff* y, como este, también se utiliza para comparar el contenido de varios archivos aunque, a diferencia de como sucede en *diff,*aquí el cotejo tiene lugar byte por byte, lo que lo hace especialmente idóneo para archivos binarios*.*  Invoca a*cmp* según la siguiente sintaxis:  *cmp [OPCIONES] ARCHIVO1 ARCHIVO2*  Si *cmp*encuentra diferencias, entrega el número de bytes y el de la línea del primer archivo que difiera. |
| **comm** | **Compara archivos clasificados por líneas**  Utiliza el comando *comm* para comparar por líneas archivos previamente ordenados (con *sort*, p. ej.).  La invocación de este comando se fundamenta en la siguiente estructura sintáctica:  *comm [OPCIONES] ARCHIVO1 ARCHIVO2*  Sin opciones adicionales, el programa genera una salida en tres columnas: la primera contiene todas las líneas que solo aparecen en ARCHIVO1; la segunda, todas las líneas que solo aparecen en *ARCHIVO2,*y la tercera, todas las líneas que aparecen en ambos archivos.  El comando *comm* soporta tres opciones:      -1 = ocultar líneas únicas del *ARCHIVO1*      -2 = ocultar líneas únicas del *ARCHIVO2*      -3 = ocultar todas las líneas que aparecen en ambos archivos |
| **cp** | **Copia archivos o directorios**  La orden*cp* (de *copy*) se utiliza para copiar archivos o ficheros y sigue la siguiente sintaxis:  cp *[OPCIONES] ORIGEN DESTINO*  El elemento ORIGEN es el que se ha de copiar y DESTINO se define a un archivo o un directorio donde se ha de alojar el elemento copiado. Si se define como destino a un archivo que ya existe, el archivo origen reescribe su contenido. También se puede crear un archivo de destino nuevo. Si se han de copiar varios archivos, entonces el destino ha de ser un directorio, del mismo modo que si se copia un directorio.  Copiar un archivo de origen en un archivo de destino en el directorio actual:  *cp [OPCIONES] ORIGEN DESTINO*  Ejemplo: *cp archivo.txt archivo\_copia.txt*  Copiar un archivo del directorio actual en un directorio de destino:  *cp [OPCIONES] ARCHIVO\_ORIGEN DIRECTORIO\_DESTINO*  Ejemplo: *cp archivo.txt home/user/documentos/2017*  Copiar varios archivos en un directorio:  *cp [OPCIONES] ARCHIVO\_ORIGEN1 ARCHIVO\_ORIGEN2 DIRECTORIO\_DESTINO*  Ejemplo: *cp archivo.txt archivo.odt home/user/documentos/2017*  Copiar un directorio desde el directorio actual en otro directorio diferente:  *cp  DIRECTORIO\_ORIGEN DIRECTORIO\_DESTINO*  Ejemplo: *cp directorio1 home/user/documentos/2017*  Si se tienen que copiar todos los directorios con el contenido completo se deben incluir en el copiado todos los subdirectorios con la opción -R. |
| **cut** | **Extrae el contenido de los archivos**  La orden*cut* permite extraer el contenido de las líneas de texto de un archivo (log o CSV) por columnas.  La sintaxis de este comando es:  *cut [OPCIONES] ARCHIVO*  Para indicar la posición exacta del fragmento que se tiene que extraer se utilizan las opciones *-b*(posición de byte),*-c*(posición del carácter), *-d*(carácter de separación) y *-f*(campo). |
| **diff** | **Compara archivos o directorios**  Con el programa *diff* se cotejan dos archivos. También permite averiguar si dos directorios contienen los mismos archivos. Se le invoca con esta estructura:  *diff [OPCIONES] ARCHIVO1 ARCHIVO2* |
| **dirname** | **Muestra la ruta al archivo**  *dirname* es lo contrario que *basename*: el comando permite extraer la parte de la ruta a un fichero sin mostrar su nombre.  Su sintaxis es:  *dirname [OPCIONES]*  Introduce *$ dirname /home/user/imagen.jpg*en el terminalpara obtener la siguiente salida:  */home/user* |
| **file** | **Muestra el tipo de archivo**  Con *file* se puede obtener información sobre el tipo o formato de un archivo.  Sigue el siguiente esquema:  *file [OPCIONES] ARCHIVO* |
| **ln** | **Crea un enlace a un archivo o un directorio**  El programa *ln* (*link*) crea un vínculo (un archivo especial) que lleva a un archivo o un directorio. Con ello se genera otra entrada a este archivo que permite acceder a él a través de una ruta diferente. Para invocar *ln* el comando ha de incluir, como mínimo, la ruta al archivo de origen.  *ln [OPCIONES] ruta/al/archivo de origen*  En este caso, el enlace se crea en el directorio actual de trabajo sin cambiar de nombre. Otra alternativa consiste en indicar una ruta de destino y dar un nombre diferente al vínculo:  *ln [OPCIONES] ruta/al/archivo de origen ruta/al/enlace*  *ln* crea por defecto los llamados enlaces duros o hardlinks, que no son adecuados para crear vínculos a los directorios y tampoco se pueden utilizar más allá de las fronteras de partición. Es por esto que el comando suele usarse en combinación con la opción -s (*--symbolic*), con la cual se pueden generar enlaces simbólicos, incluso más allá de los límites del sistema de archivos. Los enlaces simbólicos se refieren siempre a la ruta “real” a un archivo y dependen de esta. |
| **lsof** | **Muestra los archivos abiertos en el terminal**  *lsof* equivale a *list open files* (muestra los archivos abiertos), un programa de apoyo que muestra en el terminal información sobre los archivos abiertos ordenados en función del PID (ID de proceso).  Para invocarlo se sigue el siguiente esquema:  *lsof [OPCIONES]*  Dado que todos los sistemas tipo UNIX como Linux lo gestionan todo en archivos *(«Everything is a file»*), la lista que devuelve *lsof* es muy larga. Por eso suelen usarse opciones que concretan el resultado. |
| **md5sum** | **Calcula sumas de control**  Con ayuda del comando *md5sum*se pueden calcular y comprobar sumas de control MD5*.* |
| **mv** | **Mueve archivos o directorios**  El programa *mv* (*move*) copia un archivo o un directorio y borra el original. Si esto tiene lugar dentro del mismo directorio, *mv* puede utilizarse también para cambiar el nombre de los archivos.  Para ello se sigue el siguiente esquema:  *mv [OPCIÓN] ORIGEN DESTINO*  Ejemplos de su aplicación:  Trasladar un archivo a otro directorio:  *mv [OPCIONES] ARCHIVO\_ORIGEN DIRECTORIO\_DESTINO*  Ejemplo: *mv archivo1.txt home/user/documentos/2017*  Trasladar varios archivos de origen a un directorio de destino:  *mv [OPCIONES] ARCHIVO\_ORIGEN1 ARCHIVO\_ORIGEN2 DIRECTORIO\_DESTINO*  Ejemplo: *mv archivo1.txt archivo2.txt home/user/documentos/2017*  Trasladar un subdirectorio del fichero actual a un directorio de destino:  *mv [OPCIONES] DIRECTORIO\_ANTIGUO DIRECTORIO\_NUEVO*  Ejemplo: *mv directorio1 home/user/documentos/2017*  Cambiar el nombre de un archivo en el directorio de trabajo:  mv *[OPCIONES] ARCHIVO\_ANTIGUO ARCHIVO\_NUEVO*  Ejemplo: *mv archivo1.txt archivo2.txt*  Cambiar el nombre de un subdirectorio en el directorio actual:  *mv [OPCIONES] DIRECTORIO\_ANTIGUO DIRECTORIO\_NUEVO*  Ejemplo: *mv directorio1 directorio2* |
| **paste** | **Agrupa el contenido de archivos en columnas**  De forma parecida a *cat*, el programa *paste* también muestra el contenido de los archivos en la salida estándar, pero mientras que *cat* los muestra de forma encadenada, *paste* los agrupa en columnas.  El esquema básico de esta función es:  *paste [OPCIONES] ARCHIVO1 ARCHIVO2*  En el modo estándar, los archivos indicados se agrupan de tal forma que todas las líneas cuyo número coincide se emiten en una misma línea en la salida. Así, cada línea de la salida contiene el contenido de todos los archivos de entrada.  Con la opción *-d* se puede configurar individualmente el tipo de separador que se usa. Por defecto suele utilizarse la tabulación.  Con la opción *-s* (de serial) se activa otro modo diferente en el cual todas las líneas del primer archivo de entrada se pegan en la primera línea de la salida. Los datos del resto de archivos de entrada se pegan a continuación en líneas de salida separadas. De esta forma, cada línea de la salida contiene únicamente el contenido de un archivo de entrada. |
| **rename** | **Cambia el nombre de archivos**  *rename*permite cambiar el nombre de archivos y carpetas con ayuda de expresiones regulares compatibles con perl (Regex). Al contrario de *mv*, *rename* se presta así para modificar los nombres de varios archivos parcial o totalmente.  La invocación de *rename* se realiza siguiendo el siguiente esquema:  *rename [OPCIONES] 'EXPRESIÓN\_REGULAR' ARCHIVOS*  En el caso de las sustituciones, las expresiones regulares siguen la siguiente sintaxis:  *s/PATRONDEBUSQUEDA/SUSTITUCIÓN/MODIFICADOR*  En el siguiente ejemplo todas las terminaciones *.html* se sustituyen por *.xhtml*:  *rename 's/\.html$/.xhtml/' \*.html* |
| **rm** | **Borra archivo o directorio**  El programa *rm* (*remove*) borra archivos o directorios de forma irreversible. Para ello se sigue la siguiente estructura:  *rm [OPCIONES] ARCHIVO*  o  *rm [OPCIONES] DIRECTORIO*  Para eliminar un directorio junto a todos sus subdirectorios se utiliza *rm* con la opción *-R* (*--recursive*):  *rm -R DIRECTORIO*  Cuando se ordena eliminar varios archivos o ficheros, estos se separan por espacios:  *rm [OPCIONES] ARCHIVO1 ARCHIVO2* |
| **shred** | **“Tritura” archivos**  *shred*es un comando que permite borrar archivos de forma permanente porque los reescribe de modo que, incluso contando con los mejores medios forenses, no se pueden restablecer.  Este comando se invoca siguiendo este esquema básico:  *shred [OPCIONES] ARCHIVO*  Utilizándolo con las siguientes opciones se eliminan archivos concretos:  *shred -fuz ARCHIVO*  La opción *-f* impulsa el proceso de eliminación, *-z* reescribe el contenido del archivo con ceros (la configuración estándar son datos aleatorios) y *-u* elimina el archivo triturado del sistema de forma similar al comando *rm*. |
| **sort** | **Ordena listas de archivos y salidas de programa**  Con el comando *sort* se ordenan las listas de archivos y las salidas de programa en líneas numérica y alfabéticamente.  Se invoca a *sort* siguiendo este esquema:  *sort [OPCIONES] ARCHIVO*  Este método se puede complementar con opciones como *-n* (ordenar numéricamente), *-R*(ordenar de forma aleatoria) o *-r* (ordenar en orden inverso). |
| **split** | **Divide archivos**  El comando *split* se utiliza para subdividir archivos. Sigue el siguiente esquema sintáctico:  *split [OPCIONES] [INPUT [PREFIJO]]*  INPUT equivale al archivo que se ha de subdividir y PREFIJO hace las veces de prefijo del nombre del nuevo subarchivo. A la hora de cambiar su nombre se sigue el siguiente esquema:  *PREFIJOaa, PREFIJOab, PREFIJOac …*  Si no se define ningún prefijo, *split* recurre al estándar *x*.  Con la opción *-b* (bytes) se puede definir el tamaño de las partes en que se divide el archivo. Esto puede hacerse en bytes (b), kilobytes (k) o megabytes (m).  Ejemplo:  *split -b 95m archivo.tgz split-archivo.tgz.*  Aquí*split* divide el archivo *archivo.tgz*en archivos de 95 megabytes y los nombra como sigue:  *split-archivo.tar.aa*  *split-archivo.tar.ab*  *split-archivo.tar.ac*  Los archivos que se han subdividido utilizando *split* pueden reunirse de nuevo con el comando *cat*.  *cat split-archivo.tar.\* > archivo.tar* |
| **stat** | **Muestra fecha y hora**  Con ayuda del comando *stat* (*status*) se puede obtener la fecha y la hora de los accesos y los cambios relativos a determinados archivos o directorios.  Este comando sigue esta sintaxis:  *stat [OPCIONES] ARCHIVO*  El formato de salida se puede configurar con opciones. |
| **touch** | **Cambia fecha y hora**  Para modificar la fecha y la hora de los accesos o los cambios que han tenido lugar en un archivo se utiliza el comando *touch.*Si se utiliza en un archivo que no existe, este se crea automáticamente, lo que hace que este comando también se utilice para generar archivos vacíos.  Para invocar a *touch*se tiene que seguir este esquema:  *touch [OPCIONES] ARCHIVO*  Si de lo que se trata es de cambiar la fecha y la hora de un archivo por una fecha determinada se utiliza la opción *-t* incluyendo la fecha en el formato *[AA]MMDDhhmm[.ss].*  Ejemplo*:*  *touch -t 1703231037 archivo.txt*  La fecha y la hora del acceso y los cambios respecto a este archivo se han fijado en el 23 de marzo de 2017 a las 10:37 horas. Si se prefiere, el cambio se puede limitar a la fecha de acceso o de modificación con las opciones *-a* y *-m*. Si se utiliza sin la opción *-t* se sella automáticamente con la fecha y la hora actual. |
| **uniq** | **Elimina duplicados en listas de archivos y salidas de programa**  El comando*uniq* suele usarse junto con *sort* para limpiar líneas dobles en los archivos ordenados.  En el siguiente ejemplo se enlaza el comando *sort* con el comando *uniq* con la barra vertical para seleccionar en primer lugar un archivo y a continuación para emitirlo sin líneas duplicadas:  *sort archivo.txt | uniq* |

Gestión de los permisos

Con Linux se pueden definir los **derechos de acceso y de posesión de archivos y ficheros** por parte de los usuarios fácilmente. Los comandos más importantes en cuanto a la gestión de permisos son chown y chmod. Los grupos se administran con la orden chgrp.

|  |  |
| --- | --- |
| **Comando** | **Descripción** |
| **chattr** | **Gestiona los atributos de los archivos**  Con el programa *chattr* (abreviatura de *change attribute*) puedes asignar atributos a archivos o directorios. Diversos sistemas de archivos nativos de Linux (ext2, ext3, ext4, XFS, ReiserFS, JFS y OCFS2) soportan la configuración de atributos.  Utiliza *chattr* según la siguiente sintaxis para asignar un atributo:  *chattr [OPCIONES] +ATRIBUTO ARCHIVO*  Con el siguiente esquema se pueden eliminar los atributos asignados:  *chattr [OPCIONES] -ATRIBUTO ARCHIVO*  Añadiendo el atributo *-i* (de inmutable) se blinda un archivo o un directorio frente a cambios:  *chattr +i archivo.txt*  En la página del manual dedicada a *chattr* encuentras otros atributos y posibles opciones. |
| **chgrp** | **Administra los grupos de archivos y directorios**  El comando *chgrp* significa *change group* y se usa para cambiar el grupo de un archivo o un directorio, aunque se necesita estar en posesión de permisos de administrador para poder aplicarlo a un archivo o un fichero determinado. Además, solo se puede trabajar con los grupos a los cuales se pertenece.  *chgrp* se utiliza siguiendo la siguiente estructura:  *chgrp [OPCIONES] GRUPO ARCHIVO*  o  *chgrp [OPCIONES] GRUPO DIRECTORIO*  La opción *-R* incluye en la operación a las subcarpetas y los archivos contenidos en un directorio. |
| **chmod** | **Gestiona los permisos de acceso**  La llamada al sistema *chmod* (*change mode*) sirve para administrar los permisos en sistemas de archivos unixoides (ext2, ext3, ext4, reiser, xfs).  La sintaxis general de *chmod* es:  *chmod [OPCIONES] MODO ARCHIVO*  o  *chmod [OPCIONES] MODO DIRECTORIO*  *MODO* corresponde a la forma de asignar los permisos que se aplica. Para saber cómo se crean y qué hay que tener en cuenta, lee nuestro artículo sobre el [reparto de permisos de acceso](https://www.ionos.es/digitalguide/servidores/know-how/asignacion-de-permisos-de-acceso-con-chmod/) con *chmod*.  Con ayuda de la opción *-R* se pueden ampliar los derechos de forma recursiva a las subcarpetas y a los archivos contenidos en un fichero. |
| **chown** | **Administra derechos de propietario**  Normalmente, el creador de un archivo o un directorio se convierte automáticamente en su propietario (*owner*). La orden *chown* se deriva de *change owner* y permite configurar la propiedad de archivos y directorios.  Este comando se utiliza basándose en la siguiente composición:  *chown [OPCIONES] [USUARIO][:[GRUPO]] ARCHIVO*  o  *chown [OPCIONES] [USUARIO][:[ GRUPO]] DIRECTORIO*  Con objeto de definir derechos de propietario para usuarios o grupos se dispone de cuatro combinaciones posibles:  1) El propietario y el grupo de un fichero se definen según las indicaciones:  *chown [OPCIONES] nombre\_propietario:grupos\_archivos.txt*  2)El grupo se modifica en función de las indicaciones pero el usuario permanece invariable:  *chown [OPCIONES] :grupos\_archivos.txt*  3) Se define al propietario pero el grupo permanece invariable:  *chown [OPCIONES] propietario\_archivos.txt*  4) Se vuelve a definir al propietario y el grupo se define por el grupo estándar del usuario activo:  *chown [OPCIONES] nombre\_propietario: archivo.txt*  Los cambios se pueden extender a los subdirectorios con ayuda de la opción *-R*. |
| **lsattr** | **Muestra los atributos del archivo**  Si quieres ver qué atributos se definieron para un archivo o un directorio, puedes usar el comando *lsattr*(abreviatura de *list attributes*) según el esquema que sigue:  *lsattr [OPCIONES] ARCHIVO/DIRECTORIO* |

Opciones de búsqueda

Linux también dispone de diversos comandos para explorar el sistema desde el terminal.

|  |  |
| --- | --- |
| **Comando** | **Descripción** |
| **find** | **Explora el sistema de archivos**  *find*es un programa de líneas de comandos cuya función es buscar archivos en el sistema.  Se utiliza en base al siguiente esquema sintáctico:  *find [OPCIONES] [RUTA] [EXPRESIÓN\_DE\_BÚSQUEDA ][ACCIÓN]*  El directorio que se indica en *RUTA* define el inicio de la búsqueda, de tal forma que se exploran tanto el directorio como sus subdirectorios. Si no se indica ninguno, *find* comienza a explorar desde el directorio en el que se está en ese momento (directorio actual).  Las opciones indicadas en *EXPRESIÓN DE BÚSQUEDA* permiten definir los criterios de búsqueda y las acciones. La acción predeterminada es *-print*, que muestra el resultado de la búsqueda en la salida estándar (normalmente la consola).  Los criterios de búsqueda más habituales son el nombre del archivo *(-name NOMBRE DEL ARCHIVO[SUFIJO]*), un nombre de usuario *(-user NOMBRE DE USUARIO),*el tamaño del archivo*(*-*size n[cwbkMG]),*el momento del acceso en días*(-atime [+-]n*) o el momento de la modificación en días (*-mtime [+-]n*).  En la búsqueda de nombres de archivo se pueden utilizar metacaracteres (comodines) como \*. Si se escriben entre comillas se evita que el shell los interprete.  Ejemplo:  *find /tmp -name "\*.odt" -mtime -3 -size +20k*  Como directorio inicial se ha definido */tmp*. El comando *find* entrega en la salida estándar todos aquellos archivos que terminan en *.odt*, pesan más de 20 k y fueron modificados por última vez hace menos de tres días.  En la página del manual dedicada al comando de búsqueda *find* encuentras opciones adicionales. |
| **grep** | **Explora archivos de texto**  Con la orden *grep* (*global regular expression print*) se pueden explorar archivos de texto tales como archivos de registro. Como patrón de búsqueda se pueden utilizar secuencias de caracteres o expresiones regulares.  Utiliza *grep* según esta sintaxis:  *grep [OPCIONES] PATRÓN\_DE\_BÚSQUEDA [ARCHIVOS]*  Si *grep* encuentra una cadena que se corresponde con el patrón dado, el terminal muestra el número de línea y especifica el nombre del archivo.  Por regla general, *grep* se aplica a todos los archivos del directorio de trabajo actual. La opción *-r* permite la búsqueda recursiva en los subdirectorios. |
| **locate** | **Explora el índice de archivos**  El programa *locate* también permite buscar archivos desde el terminal, pero, al contrario que *find*, no se basa en el directorio de archivos, sino en una base de datos creada especialmente para ello y actualizada regularmente. Con ello *locate* entrega sus resultados mucho más rápido que *find*.  Para buscar un determinado archivo en la base de datos se invoca a *locate* con esta frase:  *locate PATRÓN\_DE\_BÚSQUEDA*  El patrón puede contener metacaracteres que actúan de comodín (\*). Si los escribes entre comillas el shell no los interpreta como metacaracteres de shell.  En el ejemplo siguiente *locate* muestra todos los archivos que terminan en *.png*:  *locate "\*.png"*  El comando*locate* distingue mayúsculas y minúsculas. Con la opción *-i* se evita esta diferenciación.  El archivo /var/lib/locatedb actúa de base de datos para la indexación de los archivos. Este archivo contiene una lista de todos los archivos del sistema en un momento determinado y se actualiza regularmente con el comando *updatedb*. |
| **tre-agrep** | **Búsqueda aproximada en archivos de texto**  También *tre-agrep* (*approximate grep*, grep aproximado) se utiliza para buscar cadenas de caracteres en los archivos de texto en base a patrones pero, a diferencia de *grep*, no solo entrega coincidencias exactas, sino también aquellas que contienen pequeños errores tipográficos. El programa se apoya para ello en la biblioteca TRE.  Su sintaxis coincide con la del comando *grep*:  *tre-agrep [OPCIONES] PATRÓN ARCHIVOS*  Con las opciones se puede definir un número máximo de errores. En el ejemplo siguiente se indica el número máximo de errores que se toleran:  *tre-agrep -1 'Linux' test.txt*  Aquí*tre-agrep* muestra todas las líneas del archivo *test.txt* que contiene palabras que coinciden con el patrón (Linux) o se diferencian de este como máximo en una letra (Linus, por ejemplo). |
| **updatedb** | **Actualiza el índice de archivos**  La búsqueda con *locate* solo es fiable si el archivo */var/lib/locatedb*se mantiene constantemente actualizado. El comando *updatedb* permite actualizar la base de datos manualmente. Ten en cuenta que para ello necesitas derechos de administrador:  *Updatedb* |
| **whereis** | **Busca páginas binarias, del código fuente o del manual de programas**  Con el comando *whereis* se pueden localizar los archivos binarios, de código fuente o del manual de ciertos programas. Su sintaxis general consiste en:  *whereis [OPCIONES] PROGRAMA*  *Ejemplo:*  *whereis firefox*  *whereis* entrega entonces en la salida estándar la ruta a los archivos encontrados separados por espacios:  *firefox: /usr/bin/firefox/ usr/lib/firefox/ usr/share/man/man1/firefox.1.gz*  Con opciones se puede delimitar la búsqueda a ciertos tipos de archivo o directorio. |
| **which** | **Encuentra los archivos binarios de los programas**  Para indagar el paradero del archivo binario de un programa se utiliza el comando *which*, el cual muestra la ruta al archivo. Este programa se invoca con la siguiente sintaxis:  *which [OPCIONES] PROGRAMA*  Ejemplo:  *which firefox*  Salida:  */usr/bin/firefox*  En el modo estándar, *which* responde con el primer archivo que encuentra. Si se desea, la opción *-a* muestra todos los archivos que cumplen con el criterio de búsqueda. |

Información sobre los usuarios

Utiliza los programas de esta categoría para solicitar **información detallada sobre los usuarios registrados en el sistema**, así como sobre sus grupos y procesos.

|  |  |
| --- | --- |
| **Comando** | **Descripción** |
| **finger** | **Muestra información sobre los usuarios**  El comando *finger*muestra información sobre los usuarios y para ello se combina con el nombre del usuario en cuestión:  *finger [OPCIONES] [USUARIO]*  El terminal entrega entonces la siguiente información sobre la cuenta de este usuario: nombre de inicio de sesión y nombre real, el momento del registro, el tiempo transcurrido desde la última actividad (idle time), el directorio principal del usuario, el shell por defecto, su localización (office number), correo y número de teléfono (si consta) y el contenido de los archivos *.plan*, *.project*, *.pgpkey* y *.forward*(si existen).  Utiliza *finger* sin indicar un nombre de usuario para solicitar información sobre tu propia cuenta. |
| **groups** | **Muestra información sobre grupos**  Con *groups* el terminal enumera los grupos a los que pertenece una determinada cuenta de usuario.  Utiliza el comando siguiendo este esquema:  *groups [OPCIONES] [USUARIO]*  Cuando no se indica usuario, *groups* localiza todos los grupos a los que pertenece tu cuenta. |
| **id** | **Solicita los ID de usuarios y grupos**  El comando *id* entrega los códigos de los usuarios y grupos de la cuenta de usuario seleccionada.  *id [OPCIONES] [USUARIO]*  Si no indicas ningún nombre de usuario, el programa muestra tu propio ID. Añadiendo opciones se puede limitar la búsqueda. |
| **last** | **Muestra información sobre el último usuario activo**  Utiliza el comando *last* según el esquema que indicamos abajo para obtener una lista de los últimos usuarios que han iniciado una sesión, junto con la fecha y la hora de su inicio y final de sesión:  *last [OPCIONES]*  Para ello, la consola solicita información del archivo *wtmp* en */var/log/wtmp.*  Si la búsqueda se refiere a una cuenta específica, se añade el nombre del usuario al comando.  *last [OPCIONES] [USUARIO]* |
| **w** | **Muestra información sobre usuarios activos y sus procesos**  En este caso, *w* devuelve una lista de todos los usuarios con sesión abierta y todos los procesos que están ejecutando.  Si añades un nombre de usuario, la búsqueda se limita a este usuario:  *w [OPCIONES] [USUARIO]*  De nuevo puedes personalizar la exhaustividad y el formato de la salida con opciones. |
| **who** | **Muestra información detallada sobre los usuarios en activo**  La orden *who* muestra información sobre los usuarios activos en ese momento en el sistema si se utiliza según esta sintaxis:  *who [OPCIÓN] [ARCHIVO]*  *who* es compatible con diversas opciones que permiten concretar cuánta información se obtiene como respuesta (terminal, tiempo que han estado conectados, nombre del host desde el que se han conectado, etc.).  Por defecto, *who* extrae los datos sobre el usuario actual del archivo */var/run/utmp*, pero opcionalmente también se pueden indicar estos archivos a continuación como fuente de la información:  Si son las sesiones anteriores las que te interesan, combina *who* con el archivo */var/log/wtmp*:  *who [OPCIÓN] /var/log/wtmp*  Si son los inicios de sesión fallidos, escribe entonces *who* con el archivo */var/log/btmp*:  *who [OPCIÓN] /var/log/btmp* |
| **whoami** | **Muestra el nombre de usuario propio**  Utiliza el comando *whoami* para obtener tu propio nombre de usuario.  *whoami [OPCIONES]* |

Gestión de cuentas de usuario

Linux ofrece toda una serie de programas con los cuales se pueden crear, eliminar y gestionar cuentas de usuario y grupos a través del terminal. Estos aparecen en el siguiente listado general de **comandos de Linux para la administración de cuentas de usuario**, categoría en la que también se incluyen las órdenes que permiten abrir programas con los derechos de otros usuarios, incluidos los de administrador.

|  |  |
| --- | --- |
| **Comando** | **Descripción** |
| **adduser** | **Crea una cuenta de usuario**  La opción más sencilla a la hora de crear una cuenta de usuario es la que ofrece el programa de líneas de comandos *adduser*, que requiere derechos de administrador y se utiliza así:  *adduser [OPCIONES] USUARIO*  Si utilizas *adduser* sin opciones se crean automáticamente, además de una cuenta de usuario, su ID, su directorio por defecto y un grupo con su mismo nombre, como en este ejemplo:  *adduser test*  Salida estándar en el terminal:  *Adding user 'test' (1001) …*  *Adding new group 'test' (1001) …*  *Adding new user 'test' (1001) with group 'test' …*  *Creating home directory '/home/test' …*  *Copying files from '/etc/skel' …*  A continuación, aparece un diálogo interactivo en el cual se pueden definir la contraseña y otros datos sobre el usuario, como su nombre real, el número de oficina, el número de teléfono, etc.  Mediante opciones se puede ajustar o cancelar este proceso automático.  El guion de Perl *adduser* se basa en la utilidad de bajo nivel *useradd* y ofrece las mismas funciones aunque en una forma más amigable para el usuario. |
| **chfn** | **Configura la información ampliada del usuario**  El comando *chfn*, equivalente a *change finger*, permite editar la información adicional de una cuenta de usuario, en la que se incluyen datos como el nombre real del usuario, el número de oficina y los números de teléfono privado y de trabajo.  La sintaxis general de *chfn*sigue esta estructura:  *chfn [OPCIÓN "NUEVO VALOR"] [USUARIO]*  Este comando requiere derechos de administrador.  Con las opciones *-f* (nombre real), *-r* (número de oficina), *-w* (número oficial) y *-h* (número privado) se define qué información del usuario se sustituye por un nuevo valor.  En el siguiente ejemplo se sustituye el número de oficina antiguo *peter23*por el valor nuevo*122:*  *chfn -r "122" peter23* |
| **chsh** | **Cambia el shell por defecto**  El comando *chsh* (*change shell*) cambia el login shell de un usuario. Para ello se sigue el siguiente esquema:  *chsh [OPCIONES] USUARIO*  Para configurar el shell estándar de un usuario, *chsh* se complementa con la opción *-s*, que espera la ruta al nuevo shell (p. ej., */usr/bin/fish).*  Los usuarios sin derechos de administrador solo pueden configurar su propio shell. Para modificar el de otros usuarios se ha de ejecutar el comando con derechos raíz, como aquí:  *sudo chsh -s /usr/bin/fish peter23*  Para el usuario *peter23*se ha definido el shell*fish* (*friendly interactive shell*) como estándar.  El cambio solo entra en vigor cuando el usuario cierra y abre sesión nuevamente. |
| **deluser** | **Cierra la cuenta de usuario**  El programa de líneas de comandos *deluser* borra todas las entradas de una cuenta de usuario de los archivos de cuentas del sistema. *deluser* solo se puede utilizar con derechos admin y se orienta por el siguiente esquema:  *deluser [OPCIONES] USUARIO*  Ejemplo:  *deluser peter23*  Con esto se elimina la cuenta de usuario *peter23*.  Si se tiene la intención de borrar todos los archivos del usuario en el directorio principal, entonces se utiliza el comando junto con la opción *--remove-home*, y si se trata de eliminar todos los archivos del usuario del sistema, se utiliza la opción *--remove-all-files*, así:  *deluser --remove-all-files peter23*  Puede ser que antes de borrar los archivos se quieran guardar en otro directorio: en este caso se utiliza la opción  *--backup-to* indicando el directorio que se vaya a utilizar:  *deluser --backup-to /ruta/al/directorio peter23*  *deluser* es un script de Perl con las funciones del programa de bajo nivel *userdel* en una forma más fácil de usar. |
| **groupadd** | **Crea grupos de usuarios**  El programa *groupadd* se utiliza para crear grupos de usuarios. Requiere derechos de administrador y se invoca siguiendo esta sintaxis:  *sudo groupadd [OPCIONES] GRUPO*  Cada grupo que se crea obtiene un identificador de grupo único (GID). Los ID de grupo entre 0 y 99 están reservados para grupos del sistema. Para definir el GID de un nuevo grupo de usuarios se utiliza el comando con la opción *-g* (GID).  En el ejemplo que sigue se crea el grupo *users* con el GID *1425*:  *groupadd -g 1425 users*  Para crear un grupo de sistema se añade la opción *-r* (root). |
| **delgroup** | **Borra grupos de usuarios**  El equivalente a *delete group*, *delgroup*, se utiliza para borrar grupos de usuarios y requiere derechos root.  Sigue esta sintaxis:  *delgroup [OPCIONES] GRUPO*  La siguiente orden tiene como resultado que el grupo *users* se borre del sistema:  *delgroup users*  De forma parecida a como ocurre con el programa *deluser*, también en este caso estamos ante un guion de Perl que pone a disposición las funciones del programa de bajo nivel *groupdel* de una forma más manejable. |
| **groupmod** | **Configura grupos de usuarios**  Con *groupmod* se puede configurar tanto el nombre como el GID de un grupo de usuarios. El comando se utiliza como administrador siguiendo el siguiente esquema:  *groupmod OPCIONES GRUPO*  Si utilizas *groupmod* con la opción *-g* puedes cambiar el GID y con la opción *-n*, el nombre del grupo.  Ejemplos:  *groupmod -g 1800 users*  En este ejemplo, el GID del grupo *users* se define como *1800*.  *groupmod -n all users*  El grupo *users* pasa a llamarse *all*. |
| **newgrp** | **Cambia el grupo de usuarios**  *newgrp* (de *new group*) es un comando que permite a los usuarios cambiar el ID de su grupo sin tener que cerrar e iniciar sesión de nuevo.  La sintaxis general de este comando es:  *newgrp [-] [GRUPO]*  Si *newgrp* se acompaña del parámetro opcional *[-]*, tras el cambio de grupo tiene lugar un reinicio del entorno del usuario, como si se abriera una nueva sesión.  Los usuarios que utilizan el comando sin indicar ningún grupo se ven desplazados al grupo estándar que se ha configurado en */etc/passwd.*  En principio, un usuario ha de ser miembro del grupo al cual quiere mudarse, con la excepción de los grupos protegidos por contraseña, donde se tiene que introducir la contraseña en el terminal. |
| **passwd** | **Cambia la contraseña del usuario**  Para cambiar la contraseña de un usuario o para definir intervalos de bloqueo o de cambios utiliza el programa de la línea de comandos *passwd* con la siguiente sintaxis:  *passwd [OPCIONES] USUARIO*  Para cambiar la contraseña de otro usuario necesitas derechos de administrador.  Escribiendo el comando*passwd*sin nombre de usuario puedes cambiar tu propia contraseña.  *passwd*  Si de lo que se trata es de bloquear la contraseña de otro usuario, se utiliza el comando con la opción*-l*(*lock*)*.*  *passwd -l USUARIO*  Otras opciones permiten definir una fecha de caducidad para contraseñas *(-x),*así como intervalos de aviso*(-w)*y de inactividad*(-i).*  *passwd -x MAX\_DÍAS -w AVISO\_DÍAS -i INACTIVO\_DÍAS USUARIO*  En el siguiente ejemplo se define que el usuario *peter24* debe renovar su contraseña cada 30 días y cinco días antes de que se agote el plazo recibe una alerta. Si pasados los 30 días el usuario no ha renovado su contraseña, esta caduca y la cuenta de *peter24* se bloquea al cabo de tres días:   *passwd -x 30 -w 5 -i 3 peter24* |
| **sudo** | **Ejecuta programas con los permisos de otros usuarios**  La instrucción *sudo* (*substitute user do*) puede anteponerse a una llamada al sistema para ejecutarla con los derechos de un usuario diferente de forma segura. Generalmente se requiere la contraseña del usuario que realiza la invocación.  Cuando se introduce el comando sin indicar ningún nombre de usuario, se utiliza el superusuario root como usuario.  *sudo LLAMADA*  En el archivo */etc/sudoers,*los administradores tienen la posibilidad de definir quién puede utilizar *sudo* y qué llamadas se permiten. Para poder utilizar el comando *sudo,* el usuario ha de pertenecer al grupo *sudo*.  Para seleccionar un usuario diferente se utiliza *sudo* con la opción *-u* y el nombre de usuario deseado.  *sudo -u USUARIO LLAMADA*  Un cambio de usuario así solo es posible cuando está autorizado en /etc/sudoers*.*  Si de lo que se trata es de trasladarse definitivamente al shell root para ejecutar comandos con derechos de administrador se utiliza sudo con la opción *-i*.  *sudo -i*  El comando *sudo* es útil porque permite a los usuarios ejecutar comandos previamente definidos como usuarios raíz sin que sea necesario revelar la contraseña root. |
| **su** | **Trabajar con los permisos de otro usuario**  El comando *su* también permite cambiar de usuario temporalmente para realizar llamadas al sistema con los permisos de otro usuario, pero a diferencia de *sudo*, en este caso las órdenes no se ejecutan directamente, sino que es necesario cambiar de identidad porque *su* no solicita la contraseña del usuario que invoca, sino la del usuario cuyos derechos se quieren suplantar. Para invocar a programas como administrador, el usuario necesita así la contraseña root del sistema. Además, al contrario que *sudo*, no es posible limitar a *su* a un conjunto de llamadas al sistema previamente definido por el administrador.  La sintaxis general del comando obedece a este orden:  *su [OPCIONES] [USUARIO]*  Una solicitud sin nombre de usuario selecciona a *root* como usuario. |
| **usermod** | **Configura las cuentas de usuario**  El comando *usermod* ofrece la posibilidad de editar cuentas ya creadas. Utiliza *usermod* con derechos raíz según el siguiente esquema:  *usermod [OPCIONES] USUARIO*  Con ayuda de las opciones defines cuáles son los cambios que se han de realizar:  - Cambiar el nombre del usuario (*-l NUEVO\_NOMBRE*):  *usermod -l peter24 peter23*  El usuario *peter23* se llama ahora *peter24*.  - Crear un nuevo directorio principal (*-d DIRECTORIO*) y mudar los archivos antiguos (*-m*):  *usermod -d /ruta/al/directorio/peter24 -m peter24*  Todos los archivos del hasta ahora directorio principal se encuentran en el nuevo directorio principal.  - Bloquear a un usuario (*-L*):  *usermod -L peter24*  La contraseña de *peter24* ha sido bloqueada.  - Añadir usuarios a un grupo (*-a*) y mantener a los demás miembros (*-G*):  *usermod -aG users peter24*  Peter se ha añadido al grupo *users*. |

Gestión del sistema

En esta categoría desciframos los comandos básicos de Linux para la administración del sistema. Con ellos puedes, por ejemplo, reiniciar o apagar el sistema desde la consola e incluso programar estas acciones.

 Nota

la mayor parte de comandos de gestión del sistema requiere permisos de administrador.

|  |  |
| --- | --- |
| **Comando** | **Descripción** |
| **logger** | **Crea registros log**  Con el programa de líneas de comandos *logger* se genera una línea en el fichero de log del sistema. Utiliza para la siguiente sintaxis:  *logger "MENSAJE"*  El fichero de log del sistema se encuentra en*/var/log/syslog*. |
| **reboot** | **Reinicia el sistema**  *reboot* es un comando que permite reiniciar el sistema, para lo cual se necesita disponer de permisos root.  *reboot [OPCIONES]* |
| **rtcwake** | **Inicia y apaga el sistema automáticamente**  En el caso de *rtcwake* se trata de una orden que permite programar el inicio, la suspensión y el apagado del ordenador. Se escribe así:  *rtcwake [OPCIONES] [MODO] [Tiempo]*  En *MODO* selecciona el modo (*-m*) en que el sistema ha de entrar por un determinado periodo (*-s* para tiempo en segundos). Opcionalmente también puedes “despertar” al sistema en un punto definido (*-t* para tiempo Unix).  Ejemplo 1:  *rtcwake -m standby -s 300*  En este ejemplo el sistema entrará en modo standby durante 5 minutos (300 segundos).  Ejemplo 2:  *rtcwake -m off -t 1490997660*  En el segundo ejemplo el sistema se apagará y se “despertará” según la fecha Unix 1490997660. Esta equivale al 1 de abril de 2017 a las 00:01:00 horas.  El tiempo Unix contabiliza todos los segundos que han pasado desde el 1 de enero de 1970 a las 00:00 horas. Dada la dificultad de comprender esta unidad temporal se recomienda utilizar el comando *date* (ver abajo) como traductor.  *rtcwake -m off -t $(date -d '20170401 00:01' +%s)* |
| **shutdown** | **Apaga el sistema**  Los usuarios con permisos de administrador pueden utilizar el comando *shutdown* para apagar el sistema.  Su base la constituye la siguiente sintaxis:  *shutdown [OPCIONES] [TIEMPO] [MENSAJE]*  Cuando se quiere apagar el sistema, existe la posibilidad de definir el momento exacto. Para ello se puede introducir una hora exacta (hh:mm) o una cuenta atrás (+m).  Otros usuarios activos en el sistema reciben un aviso de apagado que puede complementarse con un mensaje opcional.  En este ejemplo el sistema se va a apagar en 10 minutos:  *shutdown +10*  Si se añade al comando la opción *-r*, el sistema se reinicia después de apagarse:  *shutdown -r +10* |

Información del sistema

Esta categoría recoge todos los programas de líneas de comandos con los cuales se **solicitan información y mensajes de estado** del sistema para hacerse una idea global de su estado general.

|  |  |
| --- | --- |
| **Comando** | **Descripción** |
| **date** | **Muestra la hora del sistema**  El comando *date* te permite conocer la fecha y la hora del sistema:  *date [OPCIONES] [FORMATO DE SALIDA]*  Si al invocar un programa necesitas trabajar con una fecha concreta (ver *rtcwake*), la opción *-d* (fecha) te permite definirla. El comando soporta también otras opciones para cambiar el formato de fecha y hora.  Si utilizas, por ejemplo, la opción *+%s*obtienes una fecha en formato Unix(segundos que han pasado desde la fecha 1970-01-01 00:00:00 UTC).  Entrada:  *date -d '20170427 11:29' +%s*  Salida:  *1493285340*  *1493285340* en tiempo Unix equivale al 27.04.2017 - 11:29:00 horas. |
| **df** | **Muestra el espacio libre en el disco duro**  Escribiendo el comando *df*(*disk free*) en el terminal en base al siguiente esquema obtienes como respuesta el espacio libre en disco para cada partición.  *df [OPCIONES]*  Cuando se utiliza el comando combinado con un determinado archivo, el sistema responde con el espacio libre en disco en la partición donde se aloja el archivo.  *df [OPCIONES] [ARCHIVO]*  La opción *-l* (local) delimita *df* a sistemas de archivos locales. También soporta opciones con las cuales configurar el formato de salida: la opción *-h* (*human readable*) traduce la información a un formato que las personas pueden leer, como 3K 124M 1G. |
| **dmesg** | **Lista el buffer de mensajes del núcleo**  El programa *dmesg* (*display message*) muestra los mensajes de diagnóstico del buffer del kernel y permite localizar errores en el hardware o el driver.  Utiliza *dmesg* con esta sintaxis:  *dmesg [OPCIONES]*  La salida *dmesg* contiene todos mensajes generados en el arranque del sistema y es consecuentemente largo. Esto hace que se acompañe a menudo de un pager como *more*, *less* o *tail*.  Ejemplo:  *dmesg | tail*  La salida *dmesg* se redirige al pager *tail* con ayuda del operador |: con ello se muestran únicamente las últimas 10 notificaciones.  Combinado con el comando *grep* se pueden buscar ciertos mensajes. |
| **du** | **Muestra el espacio ocupado en disco**  Si lo que quieres es averiguar cuánto espacio ocupan todos los directorios en el disco duro, utiliza el comando *du* (*disk usage*) según este esquema:  *du [OPCIONES] [DIRECTORIO]*  Puedes escoger si quieres indicar un determinado directorio o no. Con la opción *–h,* el terminal muestra el espacio ocupado en disco en un formato legible. |
| **free** | **Muestra la carga de la memoria RAM**  El comando *free* muestra la ocupación de la memoria con esta sintaxis:  *free [OPCIONES]*  Como salida obtienes dos conceptos: *Mem* (*Memory*) y *Swap*.  *Mem* consiste en la memoria física del sistema. Si está agotada, Linux desplaza porciones de los datos almacenados en la RAM hacia el disco duro. En este caso se habla de *Swap-Space.*  *free* también soporta la opción *–h,* con la que obtienes la información en un formato legible. |
| **hostname** | **Muestra el nombre del host**  Utiliza el comando *hostname* con esta sintaxis para que el terminal muestre el nombre DNS del sistema.  *hostname [OPCIONES]* |
| **uname** | **Muestra información del núcleo**  El comando *uname* significa *unix name* y se utiliza para solicitar información sobre el kernel.  El comando soporta diversas opciones con las cuales filtrar la salida de la información:  *uname [OPCIONES]* |
| **uptime** | **Muestra el tiempo de ejecución del sistema**  Para averiguar cuánto tiempo lleva el sistema ejecutándose desde el último arranque sin ser reiniciado, utiliza el comando *uptime* con esta sintaxis:  *uptime* |
| **vmstat** | **Muestra las estadísticas sobre la memoria virtual**  Con ayuda de la herramienta de control *vmstat* se puede solicitar información sobre la memoria virtual, acerca de procesos de escritura y lectura en el disco duro, así como sobre la actividad del procesador central.  Introduce *vmstat* con este esquema para obtener los valores medios desde el último arranque del sistema:  *vmstat [OPCIONES]*  La orden *vmstat* también ofrece un modo de monitorización continuo con el cual se pueden solicitar los valores dentro de un intervalo de tiempo en segundos tan a menudo como sea necesario.  *vmstat [OPCIONES] [INTERVALO [REPETICIONES]]*  Ejemplo:  *vmstat 4 8*  La petición tiene lugar en 8 vueltas cada 4 segundos.  Para cancelar la petición en ejecución se utiliza el atajo de teclado [CTRL] + [C]. |

Información acerca del hardware

Los comandos Linux englobados en esta categoría entregan **datos detallados sobre los componentes de hardware** sobre los que se ejecuta el sistema.

|  |  |
| --- | --- |
| **Comando** | **Descripción** |
| **lscpu** | **Muestra información sobre el procesador**  Utiliza *lscpu* (*list cpu*) de esta forma para obtener datos sobre la CPU en la consola:  *lscpu [OPCIONES]*  En el manual de tu sistema operativo puedes consultar todas las opciones que puedes utilizar con el comando. |
| **lshw** | **Muestra información sobre el hardware**  En este caso *lshw* equivale a *list hardware* y facilita información sobre los componentes de hardware. Estos datos afectan a la CPU, a los módulos de la memoria y a dispositivos como tarjetas de sonido, de imagen o unidades de disco conectadas a interfaces PCI, USB o IDE.  El comando *lshw* se utiliza siguiendo esta sintaxis:  *lshw [OPCIONES]*  También este comando soporta diversas opciones con las cuales ajustar no solo el formato de salida (-*html*, -*xml*, -*short*, -*businfo*), sino también el detalle de la información de salida (*-sanitize* se usa para ocultar información sensible, p. ej.). |
| **lspci** | **Muestra información sobre dispositivos PCI**  Escribiendo *lspci* (*list pci*) según esta estructura se obtienen datos exhaustivos sobre los dispositivos PCI.  *lspci [OPCIONES]*  Para consultar otras opciones dirígete al manual de tu sistema operativo. |
| **lsusb** | **Muestra información sobre dispositivos USB**  Si quieres obtener información detallada sobre los dispositivos USB, entonces introduce el comando *lsusb* (*list usb*) en el terminal de esta forma:  *lsusb [OPCIONES]*  En el manual del sistema operativo encuentras también las opciones compatibles con este comando |

Gestión de procesos

En Linux se denominan procesos a las instancias de un programa en ejecución. Los siguientes comandos pertenecen al repertorio estándar de la gestión de procesos y permiten supervisarlos cómodamente desde el terminal e intervenir si es necesario.

|  |  |
| --- | --- |
| **Comando** | **Descripción** |
| **chrt** | **Solicita y ajusta atributos de tiempo real**  Con *chrt* estamos ante un programa para el control avanzado de procesos que permite averiguar y configurar los atributos de tiempo real (reglas de scheduling o planificación de procesos y prioridades) de procesos activos o ejecutar comandos y sus argumentos con determinados atributos de tiempo real. Se escribe con esta sintaxis:  *chrt [OPCIONES] [PRIORIDAD] PID/COMANDO [ARGUMENTOS]*  Para descifrar los atributos de tiempo real de procesos determinados utiliza *chrt* sin indicar ninguna prioridad junto con la opción *-p*:  *chrt -p PID*  Ejemplo:  *chrt -p 1234*  *Chrt*arroja aquí los atributos de tiempo real del proceso*1234.*  Cuando el comando se escribe según el siguiente esquema se ejecuta un comando y sus argumentos con una cierta prioridad de tiempo real.  *chrt [OPCIONES] PRIORIDAD COMANDO [ARGUMENTOS]*  Ejemplo:  *chrt 99 firefox*  El programa Firefox se arranca con una prioridad de tiempo real de 99.  Si, en cambio, tiene que modificarse la prioridad de tiempo real de procesos activos, se utiliza la sintaxis que sigue:  *chrt -p PRIORIDAD PID*  Ejemplo:  *chrt -p 20 1234*  La prioridad de tiempo real del proceso 1234 se reduce a 20.  La orden *chrt* también permite cambiar o definir las reglas de scheduling de procesos activos o reiniciados utilizando diversas opciones.  *chrt* utiliza la regla de planificación SCHED\_RR (Round Robin, explícitamente con la opción *-r*) como valor estándar. Esto quiere decir que todos los procesos en espera (ready) reciben porciones de tiempo de CPU por parte del planificador de forma consecutiva. Esto recibe el nombre de “time slicing” e indica cuánto tiempo ha de correr un proceso hasta que deba dar paso a otro proceso. El tamaño de esta porción de tiempo asignada a un proceso depende de su prioridad. Linux ofrece 140 niveles de prioridad para procesos, en donde 0 equivale a la más alta y 139 a la más baja. Los niveles de prioridad del 1 al 99 se reservan a procesos con la denominada prioridad de tiempo real. Los procesos de usuario suelen ser ejecutados con un nivel del 100 al 139, lo que equivale a un valor nice de -20 a +19 (ver *nice*).  Junto a esta estrategia de scheduling, Linux también conoce con SCHED\_FIFO (*-f*) otra regla para procesos en tiempo real. Como SCHED\_RR, esta también funciona como un algoritmo “First In/First OUT” (FIFO), el cual no utiliza el time slicing, sino que los procesos que se inician con SCHED\_FIFO permanecen activos tanto tiempo como sea necesario hasta que finalicen o sean desplazados por un proceso con una prioridad de tiempo real más alta. Los procesos interrumpidos se añaden con esto al final de la cola. |
| **ionice** | **Asigna clases en operaciones de E/S**  El comando *ionice* se utiliza para influir sobre la prioridad de un proceso que utiliza la interfaz I/O del kernel.  La sintaxis general del comando es:  *ionice [OPCIONES] COMANDO*  Para poder invocar a *ionice* es necesario contar con permisos de administrador.  El comando distingue tres clases de planificación y estas se transmiten con la opción *-c* y el valor, que puede ir del 1 al 3:  1 = Real Time (tiempo real): la operación de E/S se ejecuta de inmediato.  2 = Best Effort (mejor esfuerzo): la operación de E/S se ejecuta tan pronto como sea posible.  3 = Idle (reposo): la operación de E/S solo se ejecuta cuando no haya ningún otro proceso que requiera tiempo de E/S.  El PID de un proceso activo se transfiere con la opción *-p*PID.  Ejemplo:  *ionice -c2 -p1234*  El proceso con el PID *1234* obtiene la clase 2 de prioridad (mejor esfuerzo). |
| **kill** | **Interrumpe y finaliza un proceso con PID**  *kill* es un programa de la consola para cancelar y cerrar procesos. El comando se invoca según el siguiente esquema con una señal y el ID del proceso seleccionado:  *kill [OPCIONES] [-SEÑAL] PID*  Las señales más habituales son:  *TERM*: impele a un proceso a que finalice por sí mismo (estándar)  *KILL*: fuerza la clausura de un proceso (por parte del sistema)  *STOP*: interrumpe un proceso  *CONT*: permite continuar un proceso interrumpido  La siguiente llamada envía una señal al proceso *1234* pidiéndole que finalice por sí mismo. Como no se envía ninguna señal *kill,* envía la estándar *TERM*.  *kill 1234*  Es mejor ofrecer siempre a los procesos la posibilidad de finalizar por sí mismos. Solo se recomienda forzar la situación con *KILL* cuando el proceso no reacciona como debiera.  *kill -KILL 1234*  Si solo quisieras pausar temporalmente el proceso *1234,* utiliza la siguiente llamada para interrumpirlo o dejarlo continuar:  *kill -STOP 1234*  *kill -CONT 1234*  Utiliza el comando *kill* con la opción *-l* (*--list*) para obtener un listado de todas las posibles señales que puedes enviar vía *kill*. |
| **killall** | **Interrumpe y cierra procesos usando el nombre**  En combinación con un término clave, *killall* se utiliza para finalizar procesos cuyo nombre coincide exactamente con él (se cotejan los 15 primeros caracteres).  *killall [OPCIONES] [-SEÑAL] [PROCESO]*  La opción *-e(--exact*) permite extender la comparación a todos los caracteres del nombre del proceso. |
| **nice** | **Define las prioridades de los procesos**  El comando *nice* asigna a un proceso durante el arranque un valor nice entre -20 y +19 que determina la forma como se distribuye la capacidad disponible del microprocesador. La franja mencionada equivale a los niveles de prioridad en Linux de 100 a 139. Un proceso con un valor nice de -20 tiene, de esta forma, una prioridad más elevada que un proceso con un valor nice de 19.  Su sintaxis reza así:  *nice [OPCIÓN] [COMANDO]*  Sin ninguna otra información, todos los procesos arrancan con un valor nice de 0. Con la opción *-n* se definen las prioridades de los procesos. Hay que tener en cuenta que las prioridades negativas solo se otorgan con derechos root.  En este ejemplo que sigue el editor *nano* se arranca con una prioridad de 4:  *nice -n 4 nano* |
| **nohup** | **Separa un proceso de la sesión**  Normalmente, cuando un usuario cierra una sesión en la consola con *exit*, por ejemplo, todos los procesos dependientes finalizan automáticamente. El comando *nohup* (*no hangup*) disuelve a un proceso de la sesión en curso de forma que puede seguir activo aun cuando el usuario ha cerrado su sesión. La señal *HUP* (*hangup*), que generalmente conduce a la interrupción de un proceso automáticamente, se ve bloqueada por *nohup*.  Este comando sigue este esquema:  *nohup COMANDO* |
| **pgrep** | **Busca el PID mediante un término de búsqueda**  El programa de la consola *pgrep* compara la lista de procesos activos con un término clave y, en caso de coincidencia, entrega los respectivos PID.  La sintaxis básica de tal llamada consiste en:  *pgrep [OPCIONES] Término*  Por defecto, *pgrep* devuelve los PID de todos los procesos que contienen la palabra clave.  Ejemplo:  *pgrep ssh*  Con esta llamada, la consola devuelve una lista de todos los procesos que contienen *ssh* en su nombre.  Si la búsqueda se ha de delimitar a todas las coincidencias exactas, se utiliza la opción *-x*.  Ejemplo:  *pgrep -x sshd*  Con este ejemplo solo se muestran los procesos que se llaman exactamente *sshd*.  Si además del PID también quieres solicitar el nombre del proceso, combina *pgrep* con la opción *-l*.  De forma similar a *grep*, *pgrep* soporta palabras clave basadas en expresiones regulares. |
| **pidof** | **Muestra los ID de los procesos**  En el caso del programa *pidof,* se obtiene de la consola los números de identificación de todos los procesos de un programa. Solicita los PID con *pidof* siguiendo este esquema:  *pidof [OPCIONES] PROGRAMA*  Con la siguiente entrada se muestran los ID de todos los procesos activos del programa *nano* en el terminal.  *pidof nano*  Cuando solo se quiere obtener el ID del primer proceso, se combina *pidof* con la opción *-s* (de *single shot*). |
| **pkill** | **Finaliza y cierra procesos con palabra clave**  Como *kill*, *pkill* también envía una señal a un proceso determinado, aunque el direccionamiento no tiene lugar solamente con el PID, sino con un término clave que se compara con el nombre de los procesos activos y puede ser formulado como expresión regular.  También *pkill* envía la señal estándar *TERM* si no se define cualquier otra señal. La sintaxis general del comando reza:  *pkill [OPCIONES] [-SEÑAL] [TÉRMINO]*  Gracias a diversas opciones, podemos limitar el comando a los procesos de un determinado usuario (*-U UID*), a los procesos hijo de un proceso padre (*-P PID*) o a los procesos recientes (*-n*) o a los más antiguos (*-o*).  Mientras que *pkill* se dirige a todos los procesos cuyo nombre contiene la palabra clave, *killall* se refiere solo a los procesos que coinciden exactamente con él. |
| **ps** | **Muestra una lista de todos los procesos activos**  El comando *ps* entrega una lista de todos los procesos activos:  *ps [OPCIONES]*  Si necesitas una salida más detallada utiliza *ps* con las opciones *-f* (detallado) o *-F* (muy detallado). En el manual del sistema operativo encuentras otras posibles opciones. |
| **pstree** | **Muestra los procesos activos como estructura en árbol**  Utiliza *pstree* para obtener una lista en forma de árbol de todos los procesos en activo en la actualidad. Sigue la siguiente sintaxis:  *pstree [OPCIONES]*  Utilizando diversas opciones puedes definir el formato y la amplitud de la salida. |
| **renice** | **Configura las prioridades de procesos activos**  El comando renice permite modificar la prioridad de un proceso activo. Se invoca con esta sintaxis:  *renice PRIORIDAD [OPCIONES]*  El direccionamiento tiene lugar con ayuda de opciones por medio del ID de proceso (*-p PID*), los ID de grupos (*-g GID*) o un nombre de usuario (*-u USUARIO*).  Ejemplos:  *renice 12 -p 1234*  Al proceso con el ID *1234* se le asigna una prioridad de *12*.  *renice 3 -g 3456*  Todos los procesos del grupo con el GID *3456* obtienen un valor de prioridad de *3*.  *sudo renice -6 -u peter24*  En este último caso, todos los procesos activos del usuario *peter24* reciben una prioridad de *-6.*  Sin opción, *renice* adquiere el valor por defecto *-p* e interpreta la secuencia consiguiente como ID del proceso. |
| **sleep** | **Retrasa la ejecución actual de un proceso**  El comando sleep se utiliza para interrumpir por un periodo determinado de tiempo la sesión actual en la consola. La sintaxis general del comando reza:  *sleep CIFRA[SUFIJO]*  Si invocas a *sleep* sin sufijo, la cifra indicada se interpreta como franja en segundos (s), pero también se cuenta con la posibilidad de interrumpir la sesión durante unos minutos (m), unas horas (h) o incluso durante días.  La siguiente llamada interrumpe la sesión durante 4 minutos:  *sleep 4m*  Este comando puede utilizarse, por ejemplo, para retrasar la ejecución de un comando ulterior:  *sleep 1h && reboot*  En el ejemplo, el sistema espera una hora y, a continuación, ejecuta el comando *reboot*, que reinicia el sistema. |
| **taskset** | **Asignar procesos a determinados procesadores**  En el caso de *taskset* nos encontramos ante un comando para el control ampliado de procesos que se utiliza en sistemas con procesadores multinúcleo para asignar procesos o programas a los núcleos. Esta orden se realiza con permisos de administrador y utiliza uno de los siguientes esquemas:  *taskset [OPCIONES] MÁSCARA ORDEN*  *taskset [OPCIONES] -p PID*  La asignación de procesos y órdenes a un núcleo de la CPU tiene lugar utilizando una máscara de bits hexadecimal:      0x00000001 = núcleo #0      0x00000003 = núcleos #0 y #1      0xFFFFFFFF = todos los núcleos (#0 a #31)  El resultado de este método de reparto con máscara de bits es poco intuitivo, motivo que lleva a utilizar *taskset* con la opción *-c* (*--cpu-list*), que permite la asignación numérica de los núcleos (0, 5 7, 9-11).  La siguiente orden le indica al proceso *1234* que ha de utilizar los núcleos *1* y *2*:  *taskset -p 1234 -c 1,2* |
| **top** | **Listado dinámico de los procesos en ejecución**  Con el comando *top* obtienes una lista dinámica de todos los procesos activos. Para ello utiliza este esquema:  *top [OPCIONES]*  La salida de la información sobre los procesos se puede ajustar con ayuda de diversas opciones. El comando soporta, además, el uso de las siguientes teclas para ordenar la salida:      [P] = ordena la salida en función del uso de CPU      [M] = ordena la salida según el uso de memoria      [N] = ordena la salida numéricamente por PID      [A] = ordena la salida por edad      [T] = ordena la salida por tiempo utilizado de CPU      [U USUARIO o UID] = filtra la salida por el usuario  Utiliza la tecla [H] para acudir a las páginas de ayuda y [Q] para abandonar la vista de los procesos. |

Pager