**Sobre shell y el uso de sus comandos bases. (Holberton School)**

primer comando: **pwd**

**pwd** lo que hace es indiciarle la ruta de directorio en la que actualmente se encuentra para su terminal. *pwd* es una abreviatura de ' p rint w RABAJAR d irectorio'. Todo lo que hace es imprimir el directorio de trabajo actual del shell (decirle básicamente dónde se encuentra ubicado, ahora mismo, en la terminal y que es allí donde se ejecutará cualquier cambio que efectue en ella). Si por ejemplo desea crear una carpeta nueva desde la terminal, la carpeta se creará sobre la ruta de directorio o ubicación en la que actualmente se encuentre dicha terminal o shell. Bien es planteado en el siguiente texto:

*“Si intenta crear nuevos archivos o directorios, ver archivos existentes o incluso eliminarlos, el shell asumirá que los está buscando en el directorio de trabajo actual (ubicación), a menos que tome medidas para especificar lo contrario. Por lo tanto, es muy importante tener una idea del directorio en el que se encuentra el shell en un momento dado, después de todo, eliminar archivos del directorio incorrecto podría ser desastroso. Si alguna vez tiene alguna duda, el pwd comando le dirá exactamente cuál es el directorio de trabajo actual.”*

Ahora, si por ejemplo desea cambiar el directorio de trabajo (ubicación de la terminal o shell) puede hacerlo con otro comando, ese comando es: **cd,** una abreviatura de ' c hange d irectory' **(segundo comando: cd).**

Ejecute, por ejemplo: “cd /” y luego “pwd”. Su nueva ubicación para la terminal o shell sería: /

(El directorio "/", a menudo denominado directorio raíz, es la base de ese sistema de archivos unificado. A partir de ahí, todo lo demás se ramifica para formar un árbol de directorios y subdirectorios.)

*Advertencia: “aunque el directorio “/” a veces se denomina directorio raíz , la palabra “raíz” tiene otro significado.* ***root*** *es también el nombre que se ha utilizado para el superusuario desde los primeros días de Unix. El superusuario, como su nombre indica, tiene más poderes que un usuario normal, por lo que puede causar estragos fácilmente con un comando mal escrito. Por ahora, solo tiene que saber que la palabra "raíz" tiene múltiples significados en el mundo Linux, por lo que el contexto es importante.”*

Desde el directorio raíz, el siguiente comando lo moverá al directorio "inicio" (que es un subdirectorio inmediato de "/"):

cd home

pwd

Para subir al directorio principal, en este caso volver a "/", use la sintaxis especial de dos puntos ( ..) cuando cambie de directorio (observe el espacio entre cd y .., a diferencia de DOS, no puede escribir cd.. como un solo comando):

**cd .. (“..”: es la dirección de un directorio, el directorio que contiene al directorio actual).  
Qué sería entonces ¿ cd . ?: pues, sería la dirección de su directorio actual. OJO!**

pwd

También puede usar ..más de una vez si tiene que moverse hacia arriba a través de varios niveles de directorios principales:

cd ../..

pwd

**Caminos relativos y absolutos (relative path and absolute path)**

Los caminos relativos se refieren a las rutas o direcciones que, para llegar a ellas, depende de estar situados ya dentro de una ubicación especifica que las contenga (si esas nuevas rutas o direcciones deseadas no las contiene la ubicación actual, entonces no podremos llegar a ellas y la terminal nos emitirá un mensaje de error). Es decir, el lugar en el que termina depende de su directorio de trabajo actual (esa carpeta a la que desea llegar debe estar contenida en su ubicación o carpeta actual).

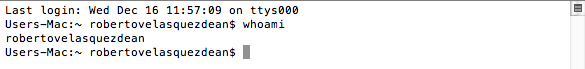
Pero hemos visto dos comandos que son absolutos. No importa cuál sea su directorio de trabajo actual (su ubicación actual), tendrán el mismo efecto. La primera es cuando se ejecuta ***cd*** solo para ir directamente a su directorio de inicio. El segundo es cuando solía ***cd /*** cambiar al directorio raíz. ***De hecho, cualquier ruta que comience con una barra inclinada es una ruta absoluta. Puede pensar en ello como decir "cambie al directorio raíz, luego siga la ruta desde allí".***

Hay otro atajo útil que funciona como una ruta absoluta. Como ha visto, usar "/" al comienzo de su ruta significa "comenzar desde el directorio raíz". ***Usar el carácter de tilde (”~”) al comienzo de su ruta significa de manera similar “comenzar desde mi directorio de inicio”. Personalmente, la ventaja practica que veo con el uso de la tilde “~” es la de ubicarme en una dirección especifica deseada de una manera más rapida y sencilla sin necesidad de llegar a esa dirección especifica carpeta por carpeta (”slash por slash”).***

Por ejemplo, si deseo llegar al escritorio directamente sin extenderme con una ruta larga del tipo:

***cd /home/USERNAME/Desktop***, se podría abreviar todo de la siguiente manera: ***cd ~/Desktop***

dato de interés: el ***whoami*** comando le recordará su nombre de usuario, en caso de que no esté seguro:

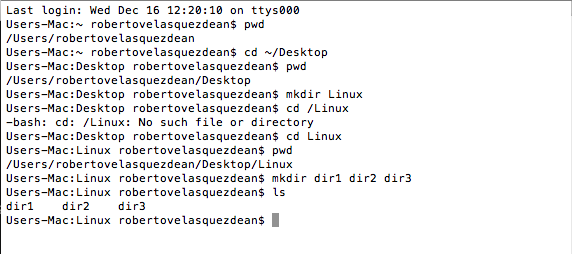


**Introducción a mkdir (tercer comando) a. sobre la creación de sub-directorios (carpetas)**

El comando mkdir lo que hace es crear sub-directorios o carpetas (cuantas desee) en una misma linea de código. Por ejemplo: ***mkdir dir1 dir2 dir3*** (he creado tres carpetas con los nombres de: dir1, dir2 y dir3)

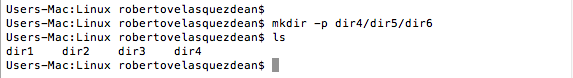
Hay algo un poco diferente en ese comando. Hasta ahora solo hemos visto comandos que funcionan por sí solos (cd, pwd) o que tienen un solo elemento después ( cd /, cd ~/Desktop). Pero esta vez hemos agregado tres cosas después del mkdir comando. Esas cosas se conocen como **parámetros o argumentos (en nuestro ejemplo; dir1, dir2 y dir3 serían parametros o argumentos)** , y diferentes comandos pueden aceptar diferentes números de argumentos. El mkdir comando espera al menos un argumento, mientras que el cd comando puede funcionar con cero o uno, pero no más.

Ahora, con el comando **ls (cuarto comando)** se nos mostrará las carpetas que tenemos ubicadas dentro del directorio de trabajo actual o ubicación actual de la terminal (nos hace un listado del contenido total del directorio actual, una lista de todos los archivos y carpetas que están en la ubicación actual) Si no se especifica un argumento para el parámetro directorio (si no se define con un ***cd*** la ubicación de la terminal), por defecto se listará el contenido del directorio donde nos encontremos ubicados. Pendiente con eso.

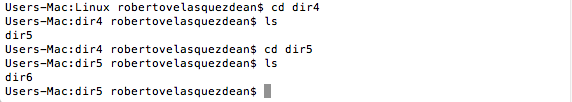


Observe que mkdir creó todas las carpetas en un directorio (nuestro directorio actual es una carpeta que creamos dentro del escritorio llamada “Linux”, es aquí donde se encuentran listadas las carpetas o sub-directorios “dir1”, “dir2” & “dir3”). Que no crean dir3 dentro dir2 dentro dir1 , o cualquier otra estructura anidada. ***Pero a veces es útil poder hacer exactamente eso, y mkdir tiene una manera, para nuestro ejemplo sería así***

***mkdir -p dir4/dir5/dir6***



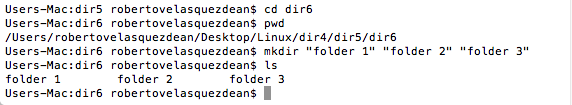
Esta vez verá que solo se ha agregado dir4 a la lista, porque dir5 está dentro y dir6 está dentro de dir5. Más adelante instalaremos una herramienta útil para visualizar la estructura, pero ya tienes el conocimiento suficiente para confirmarlo:



Ahora bien, si desea crear carpetas con espaciado dentro del nombre de ellas, tiene su manera y debe hacerlo bien (antes que el programa interprete que está intentado crear dos carpetas y no una con un espaciado en el nombre) Hay distintas formas para lograr esto, acá un par de ejemplos:

mkdir "folder 1"   
mkdir 'folder 2'   
mkdir folder\ 3   
mkdir "folder 4" "folder 5"   
mkdir -p "folder 6"/"folder 7"

Probemos creando 3 carpetas que se llamen: **folder 1, folder 2 & folder 3 dentro de dir6.**



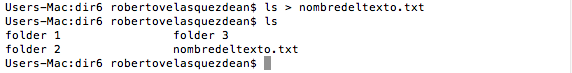
**Introducción a mkdir (tercer comando) b. sobre la creación de archivos.**

Nuestra carpeta de demostración está empezando a verse bastante llena de directorios, pero le faltan archivos. Si quisieramos, por ejemplo, crear un archivo del tipo .txt dentro de dir6 que contenga la salida del comando ls

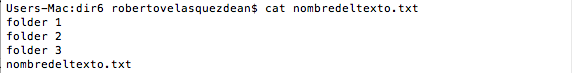


Sería así: **ls > nombredeltexto.txt (el contenido de este archivo .txt son cadenas de texto que dicen: folder 1, folder 2 & folder 3)**

Esta vez no hay nada impreso en la pantalla, porque la salida se redirige a nuestro archivo. Si solo lo ejecuta ls, debería ver que se ha creado el archivo nombredeltexto.txt .



Podemos usar el cat comando para ver su contenido (como lo prometimos, el contenido será: folder 1, folder 2 & folder 3):



De acuerdo, no es exactamente lo que se mostró en la pantalla anteriormente, pero contiene los mismos datos y está en un formato más útil para su procesamiento posterior. Veamos otro comando echo:

echo "This is a test"

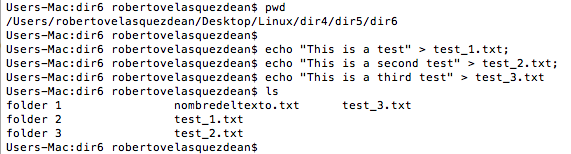
Sí, echo simplemente imprime sus argumentos nuevamente (de ahí el nombre). Pero combínelo con una redirección y tendrá una forma de crear fácilmente pequeños archivos de prueba:

**echo "This is a test" > test\_1.txt;**

**echo "This is a second test" > test\_2.txt;**

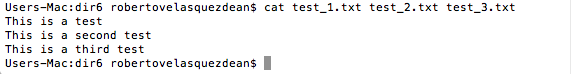
**echo "This is a third test" > test\_3.txt**

**ls**



Pero cat es más que un simple visor de archivos: su nombre proviene de 'con cat enate', que significa "enlazar". Si le pasa más de un nombre de archivo, cat se generará cada uno de ellos, uno tras otro, como un solo bloque de texto:

**cat test\_1.txt test\_2.txt test\_3.txt**



Cuando desee pasar varios nombres de archivo a un solo comando, existen algunos atajos útiles que pueden ahorrarle mucho escribir si los archivos tienen nombres similares. Se puede utilizar un signo de interrogación ("?") Para indicar "cualquier carácter" dentro del nombre del archivo. Se puede usar un asterisco (”\*”) para indicar “cero o más caracteres”. A veces se los denomina caracteres "comodines". Un par de ejemplos pueden ayudar, los siguientes comandos hacen lo mismo:

**cat test\_1.txt test\_2.txt test\_3.txt**

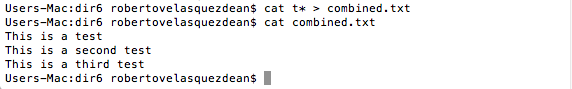
**cat test\_?.txt**

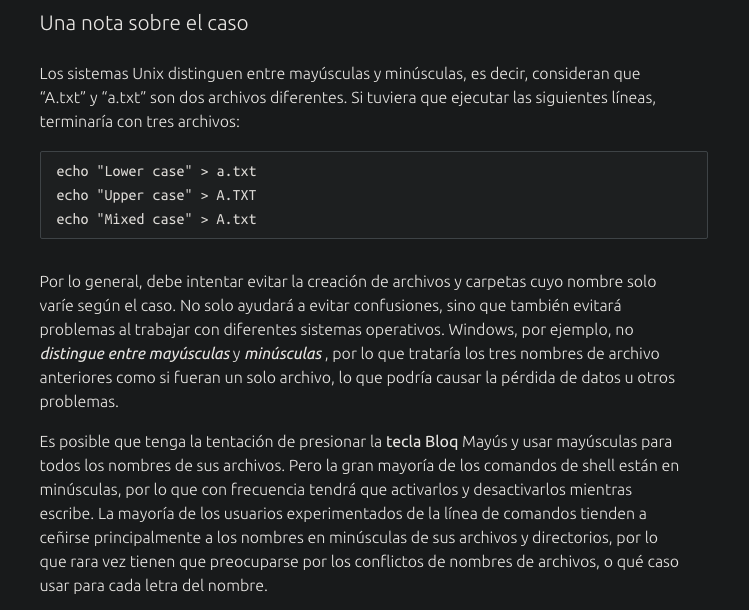
**cat test\_\***

Si observa el resultado de ls, notará que los únicos archivos o carpetas que comienzan con "t" son los tres archivos de prueba que acabamos de crear, por lo que incluso podría simplificar ese último comando aún más cat t\*, lo que significa "concatenar todos los archivos cuyos nombres comienzan con una t y van seguidos de cero o más caracteres ”. Usemos esta capacidad para unir todos nuestros archivos en un solo archivo nuevo y luego verlo:

**cat t\* > combined.txt** (creación de nuevo texto concatenando los textos que inician con t)

**cat combined.txt** (mostrando el contenido del nuevo texto ya concatenados en él los textos que inician con t)





***Concejos antes de continuar:*** *No se alarme si ejecuta un comando y aparece otro mensaje inmediatamente, ya que eso generalmente significa que el comando se realizó correctamente.*

*La importancia de las mayúsculas y minúsculas: Tenga mucho cuidado con las mayúsculas y minúsculas al escribir en la línea de comandos. Escribir en PWD en lugar de pwd producirá un error, pero a veces el caso incorrecto puede dar como resultado que un comando parezca ejecutarse, pero no haga lo que esperaba.*

***Otros comandos extras de Linux, un poco de Emacs y Apache***

***SSH:*** *Secure Shell (SSH) es una interfaz de comando y un protocolo basados en UNIX para obtener acceso seguro a una computadora remota (es decir, usted puede hacer uso de la computadora de alguien sin necesidad de tenerla fisicamente).* ***El acceso depende de un user y una contraseña, estos datos se meten en una terminal y se inicia sesión en su servidor.***

***uname*** *(abreviatura de unix name): es un programa de software en sistemas operativos de computadora Unix y similares a Unix que imprime el nombre, la versión y otros detalles sobre la máquina actual y el sistema operativo que se ejecuta en ella. Por ejemplo, Linux. (es el sistema operativo que está ejecutando mi servidor).*

***man:*** *Cuando no esté seguro de cómo usar un comando, o cuando necesite aprender más sobre las opciones disponibles, puede usar el comando man para mostrar el manual de un comando. Por ejemplo, si desea mostrar el manual del programa ls (que ya leyó para el Nivel 1) puede usar esta línea de comando: “****man ls****”. El comando también puede ser ejcutado así: “****man ls | more****”, lo que hace es: mostrar la información de asistencia por medio de saltos de linea al presionar “enter” o por medio de “página por página” al presionar la barra espaciadora.*

*El dato: hay otra forma de solicitar más información sobre un comando en especifico, esa forma se da ejecutando el siguiente código: “****ls --help****” o, con más detalles, así: “****ls --help |more****”*

***pwd:*** *significa "directorio de trabajo de impresión". Este comando genera la ruta completa del directorio en el que se encuentra actualmente, llamado directorio de trabajo (ubicación actual de la terminal).*

***ls:*** *Le permite listar archivos y directorios (carpetas) contenidos en un directorio. Si no especifica qué directorio listar, ls lista los archivos y directorios contenidos en su directorio de trabajo.*

***cd:*** *significa "cambiar directorio". cd le permite navegar por el sistema de archivos de Linux y lo mueve al directorio que especifique, cambiando su directorio de trabajo. Si no especifica un directorio,* ***“cd”*** *lo moverá de regreso directamente a su directorio de inicio (el mismo directorio en el que comienza cuando inicia sesión por primera vez). Tenga en cuenta que* ***~*** *siempre denota su directorio de inicio y* ***~/carpeta\_*** *denota un directorio llamado* ***carpeta\_****, que está dentro de* ***~****.*

***less:*** *es un comando de Linux que le permite ver archivos de texto (la información que tienen por dentro). Se diferencia de* ***cat*** *debido a que cat también puede concatenar la información de varios archivos entre sí (y no sólo visualizar su contenido por dentro). (no confundir con comando cd, esto no es una carpeta; no va a ver dentro de ella, ni a ir a su dirección. Sólo quiere leer la info que tiene el archivo por dentro. No tiene nada que ver con carpetas o directorios)*

***touch:*** *crea archivos nuevos, a diferencia de mkdir que lo que hace es crear carpetas. Adicionalmente, si desea editar sus archivos creados, la terminal posee un editor propio vinculado a ella; el cual, recibe por nombre y se ejecuta con el siguiente comando:* ***“nano”.*** *Entonces, pongamos un ejemplo, si desea editar un archivo que recibe por nombre: code.txt; para editarlo sería ejecutando su código más o menos así:* ***nano code.txt****. Lo particular e interesante de este editor es que ya estando dentro del archivo a editar (ya ejecutado el comando nano y abierto el texto editable code.txt) podemos visualizar en su parte inferior un sin número de comandos que nos orientan a darle una correcta manipulación a los cambios que queramos ejecutar dentro del archivo code.txt, para nuestro caso en cuestión. Con* ***nano*** *también podemos crear directamente dentro de él el archivo en cuestión: sólo ejecutando el comando* ***nano****; esto me genera un nuevo archivo vacio; en el cual,**podríamos escribir sobre él y luego guardar los cambios realizados al pasarle un nombre de archivo. Existe* ***“vim”*** *también.* ***Nano y Vim hacen lo mismo.***

***echo:*** *tiene funciones similares al comando touch, la diferencia radica en que “touch” crea archivos vacios y “echo” no. No necesariamente. Con echo se puede insertar, al menos, la primera linea de código que va dentro del archivo. La sintaxis para ejecutar el comando es más o menos así:* ***echo “aquí va el texto pequeño” >> file.ext*** *(nombre de archivo.extensión).*

*La forma más sencilla y habitual de utilizar el comando* ***cp*** *es de la siguiente: cp fichero1 fichero2. Esto lo que hace es una copia del archivo1 en archivo2 (carpeta o directorio destino), también podemos especificar una ruta distinta donde queramos que ubique el fichero 2. Si fichero2 no existía antes de ejecutar el comando, se crea una copia del fichero1, pero con el nombre dado en fichero2. Si ya existía, entonces se sobreescribe o se hace una copia de fichero1 y se guarda en fichero2. Ahora bien, preste atención a esto: si desea hacer una copia de su fichero1 en una carpeta o directorio fichero2; pero, adicionalmente, desea que esa copia reciba otro nombre (no el mismo nombre de fichero1), lo puede hacer de la siguiente forma: cp fichero1.txt fichero2/nuevonombrede\_fichero1.txt (este sería un buen ejemplo).* ***cp -r*** *para directorios.*

***mv:*** *es un comando que le permite mover y cambiar el nombre de archivos y directorios. La forma más sencilla y habitual de utilizar el comando mv es de la siguiente: mv fichero1.txt fichero2/fichero1.txt... Esto lo que hace es renombrar o mover el archivo1 en archivo2. La logica es la siguiente: Si archivo2 ya existia el comando mv va a interpretar entonces que desea desplazar el archivo1 a archivo2; sin embargo, si el archivo2 no existia el comando mv va a interpretar entonces que desea cambiar el nombre de archivo1 a archivo2. Hay una forma extensa de ejercutar este comando (en caso que desee desplazar un archivo): fichero1.txt /home/robertovelasquezdean/Documents/fichero2 (carpeta donde quiere contener a fichero1)/fichero1.txt (acá se pone toda la ruta completa desde home hasta llegar al directorio destino de su interés).*

***rm:*** *comando para eliminar archivos creados con comando touch.****mkdir:*** *crea carpetas, a diferencia de touch que crea archivos.*

***rmdir:*** *comando para eliminar carpetas creadas con mkdir. Tenga en cuenta algo: este comando sólo borra carpetas vacias; es decir, si la carpeta que desea borrar contiene sub-carpetas y/o archivos extras, debe deshacerse primero de todo eso para poder borrar la carpeta en cuestión. El comando “rm -r” o, en su manera recursiva y forzada, “rm -rf” ejecutado sí que puede borrar cualquier carpeta (tenga sub-carpetas y/o archivos dentro de ella o no). Ahora, un paso peligroso: si desea borrar todos sus archivos y carpetas de su disco duro podría ejecutar el siguiente comando: “rm -rf /”. Tenga cuidado.*

***Emacs:*** *es un editor de texto que puede utilizarse para editar cualquier archivo de programación (su código) o lenguaje de marcado. Por lo general es necesario instalar emacs en un servidor, en caso que esté trabajando con ellos. Para instalar Emacs en un servidor se utiliza el comando sudo apt-get (1ro. sudo apt-get update & 2do. sudo apt-get install Emacs).* ***Se guardan los cambios con Ctrl x + s, Ctrl g refresca el editor (para cuando se buguea). Para partir en dos ventanas el Emacs y visualizar así dos archivos: Ctrl x + 2, para regresar de nuevo a una sola ventana: Ctrl x + 1. Para minimizar la ventana del Emacs: Ctrl + z, para cerrar permanentemente el Emacs: Ctrl x + Ctrl c. En Emacs, la C significa Control (botón Ctrl) o comando y la M significa Meta key o Alt (botón alt).***

***Aclaremos algo sobre los editores:*** *Emacs es el único editor de texto que, al hacer un salto de linea sin contenido textual al final del archivo, no lo registra como una linea de código.*

*Para saber cuántas lineas de texto tiene un archivo, ejecute el siguiente comando:* ***wc -l “nombre del archivo.ext”.******Y con wc -c “nombre del archivo.ext” se sabe la cantidad de caracteres de dicho archivo.***

***Consejo: Trate de trabajar con entornos de desarrollo con "resaltado de sintaxis" (es decir, color).***

***Más detalles sobre Emacs (este apartado está extraido todo de Internet con ayuda de Holberton)***

***Comandos para manipular archivos***

*C-x C-f (****Ctrl x + f): abre archivos***

*find-file. Este es el comando principal que se utiliza para leer un archivo en un búfer para editarlo. De hecho, es bastante sutil. Cuando ejecuta este comando, le solicita el nombre del archivo (con la [finalización](https://www2.lib.uchicago.edu/keith/tcl-course/emacs-tutorial.html#completion) ). Luego comprueba si ya está editando ese archivo en algún búfer; si es así, simplemente cambia a ese búfer y en realidad no vuelve a leer el archivo desde el disco. Si no es así, se crea un nuevo búfer, se le asigna el nombre del archivo y se inicializa con una copia del archivo. En cualquier caso, la ventana actual se cambia para ver este búfer.*

*C-x C-s (****Ctrl x + s): guarda archivos***

*save-buffer. Este es el comando principal que se usa para guardar un archivo o, más exactamente, para escribir una copia del búfer actual en el disco, sobrescribir el archivo del búfer y manejar las versiones de respaldo.*

***Qué es un bufer***

*Un búfer es la estructura de datos interna que contiene el texto que realmente edita. Emacs puede tener cualquier cantidad de búferes activos en cualquier momento. La mayoría de los búferes, pero no todos, están asociados con un archivo. Los búferes tienen nombres; un búfer que se ha inicializado desde un archivo casi siempre recibe el nombre de ese archivo, y decimos que el búfer está visitando el archivo. Esto significa, en particular, que cuando guarda el búfer, se guarda en el archivo adecuado. En un momento dado, se selecciona exactamente un búfer : este es el búfer en el que se encuentra el cursor de hardware y aquí es donde los comandos que escribe entran en vigor (incluidos los comandos de autoinserción). Los búferes se pueden eliminar a voluntad; eliminar un búfer de ninguna manera elimina el archivo en el disco (aunque puede perder cualquier cambio de edición que haya realizado si no lo guarda primero).* ***Emacs almacena cada texto del archivo dentro de un objeto llamado «buffer»****. Al encontrar un archivo se crea un nuevo buffer dentro de Emacs. Para mirar la lista de los buffers que existen actualmente, teclee:* ***C-x C-b*** *(comando para imprimir: Lista de buffers).*

***Cuando tenga varios buffers, solo uno de ellos es «actual» en algún momento. Ese buffer es el que actualmente edita (sin embargo, puede tener varios abiertos). Si quiere editar otro buffer, necesita «cambiar» a él. Si quiere cambiar a un buffer que corresponde a un archivo, puede hacerlo visitando el archivo de nuevo con C-x C-f. Pero existe una manera más rápida: use el comando C-x b. En ese comando, necesita teclear el nombre de buffer.***

*>> Cree un archivo llamado «foo» tecleando C-x C-f foo <Return>. Después teclee C-x b TUTORIAL.es <Return> para regresar a este tutorial.*

***Comandos para manipular búferes***

***C-x b: cambio de bufers o creación de uno nuevo***

***switch-to-buffer. Solicita un nombre de búfer y cambia el búfer de la ventana actual a ese búfer. No cambia la configuración de su ventana. Este comando también creará un nuevo búfer vacío si escribe un nuevo nombre; este nuevo búfer no visitará ningún archivo, no importa cómo lo nombre.***

***C-x C-b*** *list-buffers. Aparece una nueva ventana que enumera todos sus búferes, dando para cada uno el nombre, modificado o no, tamaño en bytes, modo principal y el archivo que está visitando el búfer.*

***C-x k*** *kill-buffer. Solicita un nombre de búfer y elimina toda la estructura de datos para ese búfer de Emacs. Si se modifica el búfer, se le dará la oportunidad de guardarlo. Tenga en cuenta que esto de ninguna manera quita o elimina el archivo asociado, si lo hubiera.*

***C-x C-q*** *vc-toggle-read-only. Cree un búfer de solo lectura (de modo que los intentos de modificarlo se consideren errores) o conviértalo en lectura y escritura si fuera de solo lectura. Además, si los archivos están bajo control de versiones, lo buscará por usted.*

***Comandos para buscar hacia adelante o atrás***

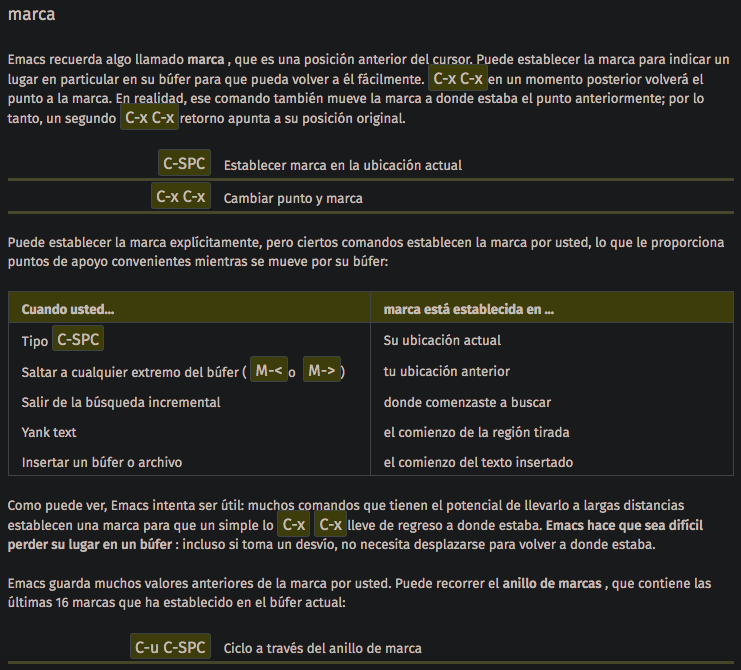
***Cómo buscar hacia adelante:***

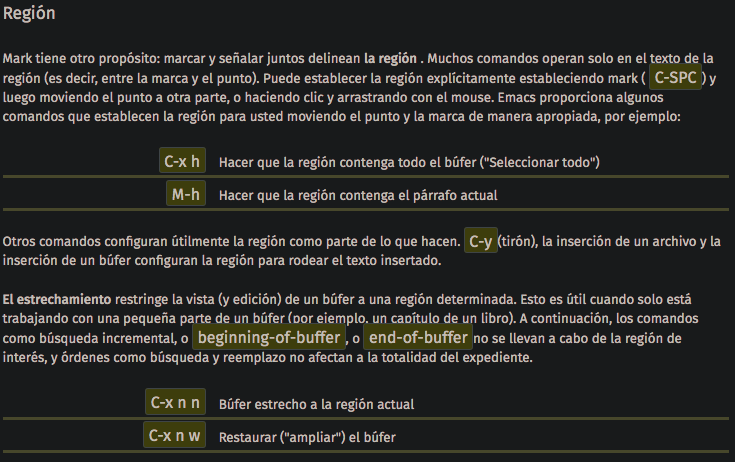
*Comando:* ***C-s***

***Cómo buscar hacia atrás***

*Comando:* ***C-r***

***Marca & Región en Emacs***





***Invocación de comandos por su nombre***

*Todos los comandos de Emacs tienen un nombre que ud. puede usar para ejecutarlos. Los comandos usados frecuentemente, o que deben ser rápidos de teclear, también están mapeados a teclas (secuencias cortas de caracteres) para usarlos más cómodamente. Ud. puede ejecutarlos por nombre si no recuerda las teclas. Otros comandos de Emacs que no necesitan ser rápidos no están mapeados a teclas; la única forma de ejecutarlos es por su nombre. @xref{Mapeo de teclas}, para la descripción de cómo mapear comandos a las teclas.*

*Por convención, un nombre de comando consta de una o más palabras, separadas por guiones; por ejemplo, auto-fill-mode o manual-entry. El uso de palabras en inglés hace al nombre del comando más fácil de recordar que una tecla formada con caracteres oscuros, aunque haya que teclear más caracteres.*

*La manera de ejecutar un comando por nombre es comenzar con M-x, teclear el nombre del comando, y terminarlo con RET. M-x usa el minibúfer para leer el nombre del comando. RET abandona el minibúfer y ejecuta el comando. La cadena `M-x' aparece al comienzo del minibúfer como un prompt para recordarle que introduzca el nombre de un comando para ejecutar. See section [El minibúfer](http://www.nongnu.org/emacs-man-es/index_36.html#SEC36), para información completa de las funcionalidades del minibúfer.*

*Puede usar completamiento para introducir el nombre del comando. Por ejemplo, puede invocar el comando forward-char por nombre tecleando ya sea:* ***M-x forward-char RET ó M-x forw TAB c RET***

***Comando deshacer, C-/.***

*Si hace un cambio al texto, y luego decide que fue un error, puede deshacer el cambio con el comando deshacer, C-/.*

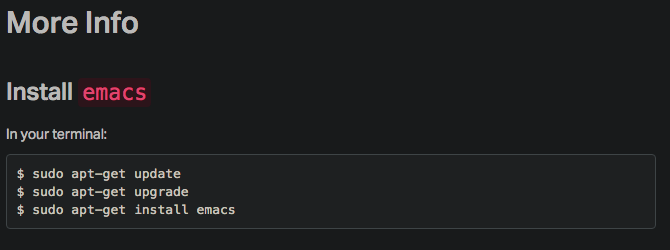
*Normalmente, C-/ deshace los cambios hechos por un comando; si repite varias veces seguidas C-/, cada repetición deshará un comando adicional.*

*Pero hay dos excepciones: los comandos que no cambian el texto no cuentan (esto incluye los comandos de movimiento del cursor y de desplazamiento), y los caracteres de autoinserción se manejan usualmente en grupos de hasta 20 caracteres. (Esto es para reducir el numero de C-/ que tenga que teclear para deshacer una inserción en el texto.)*

***>> Elimine esta línea con C-k, después teclee C-/ y debería reaparecer.***

*C-\_ es un comando alternativo para deshacer; funciona igual que C-/. En algunas terminales, al teclear C-/ en realidad envía C-\_ a Emacs. También existe la alternativa de usar C-x u, que funciona exactamente igual que C-/, pero es menos cómodo de teclear.*

*Un argumento numérico para C-/, C-\_ o C-x u actúa como un factor de repetición. Puede deshacer un texto borrado justo como puede deshacer el texto eliminado. La distinción entre eliminar algo y borrar algo afecta en si puede pegarlo con C-y; no hay diferencia alguna para deshacer.*



***Otros comandos adicionales de Emacs (Bonus):***

***¿Cuál es el comando para abrir un archivo desde Emacs?***

***R/*** C-x C-f (presione simultaneamente control, luego “x” y luego “f”; mantenga presionado)

***¿Cuál es el comando para guardar un archivo?***

***R/*** C-x C-s (presione simultaneamente control, luego “x” y luego “s”; mantenga presionado)

***¿Cuál es el comando para cortar una línea completa?***

***R/*** C-k (presione control y luego “k”; mantenga presionado)

***¿Cuál es el comando para pegar?***

***R/*** C-y (presione control y luego “y”; mantenga presionado)

***¿Cuál es el comando para buscar hacia adelante?***

***R/*** C-s (presione control y luego “s”; mantenga presionado)

***¿Cuál es el comando para deshacer?***

***R/*** C-x u (presione control y luego “x”; si desea suelte teclado, y luego, presione “u”)

***¿Cuál es el comando para salir de Emacs?***

***R/*** C-x C-c (presione simultaneamente control, luego “x” y luego “c”; mantenga presionado)

***Ahora hablaremos sobre el editor Vi (este apartado está extraido todo de Internet con ayuda de Holberton).***

***Empezemos por los comandos básicos más relevantes.***

***¿Cuál es el comando para insertar texto antes del cursor?***

***R/*** i

***¿Cuál es el comando para eliminar y cortar la línea actual?***

***R/*** dd

***¿Cuál es el comando para pegar las líneas en el búfer en el texto después de la línea actual?***

***R/*** p

***¿Cuál es el comando para deshacer lo que acaba de hacer?***

***R/*** u

***¿Cuál es el comando para salir de vi aunque no se hayan guardado los últimos cambios para esta llamada vi?***

***R/*** :q!<Return>

***¿Cuál es el comando para mover el cursor al inicio de la línea actual?***

***R/*** 0

***¿Cuál es el comando para mover el cursor al final de la línea?***

***R/*** $

***¿Cuál es el comando para mover el cursor a la línea 32?***

***R/***:32<Return>

***¿Cuál es el comando para eliminar la línea actual y anterior al mismo tiempo?***

***R/ dk***

***Dato adicional de caracter importante:*** *cuando ya haya terminado de trabajar con el editor de texto Vi y desee guardar cambios y salir, debe hacer lo siguiente: presionar “****esc****” para luego insertar el siguiente comando: “****:wq****”. De hecho, para ejecutar un comando, debe ejecutar primero el teclado “esc” justamente y luego sí proceder a escribir el comando.*

***Más a fondo sobre VI***

*El editor predeterminado que viene con el sistema operativo UNIX se llama vi (editor visual). [Los editores alternativos para entornos UNIX incluyen pico y emacs, un producto de GNU]. El editor vi de UNIX es un editor de pantalla completa y tiene dos modos de funcionamiento: el* ***modo “comando”*** *que hace que se realice una acción en el archivo, y el* ***modo “Insertar”*** *en el que el texto ingresado se inserta en el archivo.*

*En el modo de* ***comando,*** *cada carácter escrito es un comando que hace algo con el archivo de texto que se está editando; un carácter escrito en el modo* ***comando*** *puede incluso hacer que el editor* ***VI*** *ingrese al* ***modo de inserción (comando “i”)****. En el* ***modo de inserción (insertar)****, cada carácter escrito se agrega al texto del archivo; presionar la tecla* ***<Esc> (Escape)*** *desactiva el* ***modo Insertar*** *para poder retornar de nuevo al* ***modo******Comando y viceversa.***

*Si bien hay varios comandos Vi, solo un puñado de ellos suele ser suficiente para los usuarios principiantes de Vi. Para ayudar a estos usuarios, esta página web contiene una muestra de comandos Vi básicos.* ***Los comandos más básicos y útiles están marcados con un asterisco (\* o estrella) en las tablas siguientes. Con la práctica, estos comandos deberían volverse automáticos.***

*NOTA: Tanto UNIX como vi distinguen entre mayúsculas y minúsculas. Asegúrese de no utilizar una letra mayúscula en lugar de una letra minúscula; los resultados no serán los esperados.*

***Para iniciar vi***

*Para usar vi en un archivo, escriba:* ***vi “nombre\_del\_archivo.ext”.*** *Si el archivo llamado “nombre\_del\_archivo.ext” existe, se mostrará la primera página (o pantalla) del archivo; si el archivo no existe, se crea un archivo y una pantalla vacía en los que puede ingresar texto.*

*\** ***vi -r “nombre\_del\_archivo.ext”*** *le permite recuperar un archivo en caso que se hubiera cerrado de forma inesperada la terminal o el sistema hubiera fallado.*

***Para salir de vi***

*Normalmente, el archivo nuevo o modificado se guarda al salir de vi. Sin embargo, también es posible salir de vi sin guardar el archivo.*

***Nota:*** *El cursor se mueve a la parte inferior de la pantalla cada vez que se escriben dos puntos (:). Este tipo de comando se completa presionando la tecla* ***<Return> (o <Enter>).***

***\* :x + enter = sale de vi, escribiendo el archivo modificado en el archivo nombrado en la invocación original.***

***:wq + enter = sale de vi, escribiendo el archivo modificado en el archivo nombrado en la invocación original.***

***\* :q! + enter = sale de vi aunque los últimos cambios no se hayan guardado para esta llamada vi.***

***Mover el cursor***

***\* 0 (cero) = mueve el cursor al inicio de la línea actual (la que tiene el cursor).***

***\* $ = mueve el cursor al final de la línea actual. (la que tiene el cursor).***

***\* :n (cualquier número) + enter = mueve el cursor a la línea n.***

***\* :$ + enter = mueven el cursor a la última línea del archivo.***

***Nota: el símbolo ^*** *antes de una letra significa que la tecla <Ctrl> debe mantenerse presionada mientras se presiona la tecla de la letra.*

***Manipulación de pantalla***

*Los siguientes comandos permiten que la pantalla (o ventana) del editor vi suba o baje varias líneas y se actualice.*

***^ f = avanza una pantalla.***

***^ b = retrocede una pantalla.***

***^ u = sube (atrás) media pantalla.***

***^ d = mover hacia abajo (adelante) media pantalla.***

***Agregar, cambiar y eliminar texto***

***\* u = deshace lo que acaba de hacer.***

*Los siguientes comandos le permiten insertar y agregar texto. Cada uno de estos comandos pone el editor vi en modo de inserción; por lo tanto, se debe presionar la tecla <Esc> para terminar la entrada de texto y poner el editor vi nuevamente en modo comando.*

***\* i = Inserto texto antes del cursor.***

***I = Inserto texto al comienzo de la línea actual.***

***\* a = Inserto texto después del cursor. (hace lo mismo que el comando “i” prácticamente: permitirte anotar sobre la hoja)***

***\* o = Insertar texto en una nueva línea debajo de la línea actual del cursor.***

***\* O = Insertar texto en una nueva línea encima de la linea actual del cursor.***

***Cambiar texto***

*Los siguientes comandos le permiten modificar texto.*

***\* r = reemplaza un solo carácter debajo del cursor, esto lo puede hacer cuantas veces desee con cada caracter*** *(no se necesita <Esc>; por lo tanto, mientras no ejecute otro comando, el editor va a interpretar que, por cada vez que presione la tecla “r” sobre cualquier caracter, entonces, usted desea reemplazar ese caracter por el caracter que “teclee” luego de presionada la tecla “r”).*

***R = reemplaza caracteres (no sólo uno, es basicamente sobre-escribir sobre lo ya escrito).*** *Parte desde la posición actual del cursor hasta que presione <Esc>.*

***C = elimina los caracteres de la linea actual hasta donde se situa parcialmente el cursor sobre el texto, la eliminación va de derecha a izquierda. (hasta donde se situa su cursor y dentro de la misma linea) (ejecutable una sóla vez por cada llamado de comando)***

***cc = elimina toda la línea actual. (ejecutable una sóla vez por cada llamado de comando)***

***Eliminar texto***

*Los siguientes comandos le permiten eliminar texto.*

***\* x = elimina un solo carácter debajo del cursor, esto lo puede hacer cuantas veces desee con cada caracter*** *(no se necesita <Esc>; por lo tanto, mientras no ejecute otro comando, el editor va a interpretar que, por cada vez que presione la tecla “x” sobre cualquier caracter, entonces, usted desearía eliminar dicho caracter). La anterior lógica es aplicable para todos los comandos siguientes dentro de esta misma sesión de “eliminar texto”.*

***Nx = eliminar N caracteres, comenzando con el carácter debajo del cursor. (Con este comando, si presiona 5 y luego “x”, quiere decir que se borrarán los 5 siguientes caracteres que estén situados por debajo del cursor).***

***dw = borra la palabra que comienza con el carácter debajo del cursor (idealmente situese en la primera letra de dicha palabra para que el comando pueda borrar toda la palabra completa).***

***dNw = eliminar N palabras que comienzan con el carácter debajo del cursor; por ejemplo, d5w elimina 5 palabras. (idealmente situese en la primera letra de dicha palabra para que el comando pueda borrar toda la palabra completa).***

***D = elimina los caracteres de la linea actual hasta donde se situa parcialmente el cursor sobre el texto, la eliminación va de derecha a izquierda. (hasta donde se situa su cursor y dentro de la misma linea). (ejecutable cuantas veces desee hasta que no ejecute otro comando)***

***\* dd = elimina toda la línea actual. (ejecutable cuantas veces desee hasta que no ejecute otro comando)***

***Cortar y pegar texto***

*Los siguientes comandos le permiten copiar y pegar texto.*

***yy = copia la línea actual en el búfer.***

***Nyy = copia las siguientes N líneas, incluida la línea actual, en el búfer.***

***p = pega la (s) linea (s) que copió en el texto después de la línea actual.***

***Determinación de números de línea***

*A veces es útil poder determinar el número de línea de la línea actual o el número total de líneas en el archivo que se está editando.*

***:.= = devuelve el número de línea de la línea actual en la parte inferior de la pantalla.***

***:= = devuelve el número total de líneas en la parte inferior de la pantalla.***

***^g = proporciona el número de línea actual, junto con el número total de líneas, en el archivo en la parte inferior de la pantalla.***

***Guardar y leer archivos***

*Estos comandos le permiten ingresar y generar archivos que no sean el archivo con nombre con el que está trabajando actualmente.*

***:r “nombre\_de\_archivo” + enter = lo que hace es importar, dentro del archivo actual, otro archivo llamado:  
“nombre\_de\_archivo” y lo incorpora inmediatamente después de la línea actual (la línea actual donde está el cursor).***

***:w + enter = proporciona la información del número total de lineas del archivo junto con el total de sus caracteres.***

***:w “nombre\_de\_archivo” + enter = hace una copia exacta del archivo actual y la guarda en un nuevo archivo llamado: “nombre\_de\_archivo”.***

***:12,35w “nombre\_de\_archivo” + enter = escribe el contenido de las líneas enumeradas de la 12 a la 35 en un nuevo archivo llamado “nombre\_de\_archivo”.***

***:w! “nombre\_de\_archivo” + enter = hace una copia exacta del archivo actual y la pega sobre un archivo pre-existente llamado: “nombre\_de\_archivo”.***

*Para mayor info sobre más comandos relacionados a vi visite esta página: https://www.cs.colostate.edu/helpdocs/vi.html*

***Apache:*** *es un servidor web, es basicamente lo que nos permite el acceso a nuestra pagina web en el internet. En este momento, si escribe su dirección IP en la barra de direcciones de su navegador, obtendrá un error. La razón por la que muestra un error es que primero debe instalar un servidor web en su máquina, ese servidor web podría ser Apache. Y se instala dentro de su terminal así: apt-get install apache2, pero antes debe abrirse un servidor privado con SSH. Luego, sí puede escribir su dirección IP en la barra de direcciones de su navegador y su pagina web aparecerá magicamente.*

***Más detalles a fondo sobre los comandos de Linux.***

***Sobre el comando Clear:***

***Clear*** *en Linux es un comando que, al ser ejecutado, su única función es despejar completamente la pantalla: dejar su lienzo completamente limpio (vacio). Sólo es necesario escribir “clear” y presionar Enter. Es preciso aclarar que: “Clear” sólo limpia la pantalla, no tiene funciones para alterar o modificar el directorio donde, actualmente, se encuentra ubicado. El comando “cd” es, de hecho, el único que puede cambiar la ubicación de su directorio actual.*

*Algo que no hemos dicho sobre el comando Cd es que, si sólo se escribe* ***“cd”*** *como comando y se ejecuta así, entonces la terminal te dirige a la ruta inicial (home - home folder). Lo mismo hace el comando* ***“cd ~”*** *al ser ejecutado.*

***De interés general:*** *Cuidado cuando pretenda escribir entre mayúsculas y minúsculas, Linux diferencia entre ambas. Tiene que respetar las mayúsculas y minúsculas de las rutas a las q desea ir (y, por extensión, las mayúsculas y minúsculas de cualquier cosa).*

***Otro detalle:*** *Linux tiene la capacidad o inteligencia de autocompletar las palabras antes de ser escritas manualmente por usted; esto en caso que Linux interprete que, lo que intenta escribir, sólo se parece a una ruta, archivo o carpeta con dicho inicio de escritura. Por ejemplo, si desea ir a una carpeta o directorio en específico, él puede interpretar que la carpeta o directorio que pretende buscar es cierta carpeta o directorio a partir del inicio de la escritura que has hecho de forma manual; dicho lo anterior, entonces, Linux podría autocomplementar su búsqueda si presiona la tecla “Tab”.*

***Sobre el comando Ls:***

*Ya sabemos que “ls”, al ser ejecutado, lo que hace es mostrarnos en una lista todos los archivos y carpetas que posee nuestra ruta actual: hace un listado de lo contenido actualmente en su ruta (donde te encuentras ubicado) en términos de archivos y carpetas. Pues bien, si usted tipea:* ***“ls******-l”,*** *estariamos ejecutando un comando con una especificación más a fondo de la función inicial del comando “ls”: esta especificación agrega un detalle en sus funciones y es que no sólo nos alista las carpetas y archivos contenidos en nuestra ruta actual; sino que también nos clasifica, línea por línea de código, qué tipo de elemento es cada item contenido en la ruta actual o directorio (si se trata de una carpeta o de un archivo como tal).*

*La dinámica es más o menos así: toma elemento por elemento y los clasifica por separado en una línea de código, luego en el carácter inicial de cada línea de código (de cada elemento) podemos visualizar si se trata de un archivo o de una carpeta. Cómo lo vemos? Pues así:*

*Si el carácter o letra inicial del elemento (en su línea específica de código) empieza con la letra: "d", pues, estamos ante un directorio (los directorios son como carpetas también, se tiene la facultad de poder navegar entre ellas con el comando Cd).*

*Si el caracter empieza con: "-" estamos ante un archivo (el cual no se puede navegar con Cd, sino que podría verse su contenido interior por medio de la ejecución del comando Less).*

*Luego, los caracteres que siguen después de esa letra o carácter inicial tienen que ver con los permisos que tiene cada usuario según su clasificación: propietario, grupo o usuario público. Pues bien, la primera sesión de caracteres corresponde al owner (propietario de la consola), luego hay una separación por medio de un guión (-) para distinguir los permisos del “propietario” con los del “grupo”. Así mismo sucede, cuando se quiere hacer la separación entre “grupo” y “usuario público”.*

*Expliquemos ahora, entonces, cómo se especifican los permisos:*

***r: "read", quiere decir que el usuario en cuestión tiene derecho a leer (lectura)***

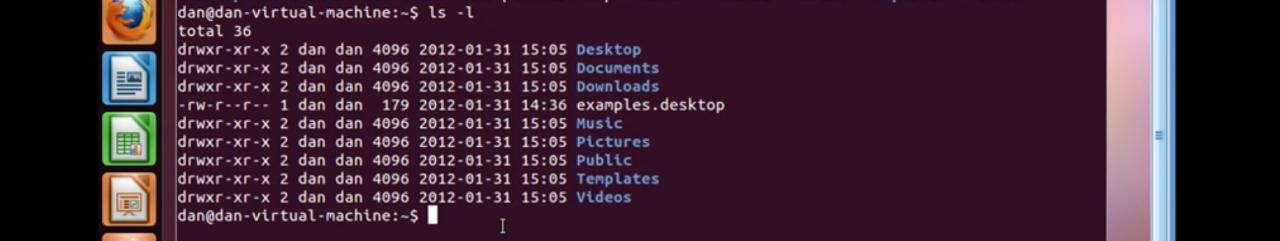
***w: "write", quiere decir que el usuario en cuestión tiene derecho a editar o escribir.***

***x: "execute", quiere decir que el usuario en cuestión tiene derecho a ejecutar dicho elemento.***

*Posteriormente, se aclara el nombre del propietario de la consola y, justo al lado, el nombre que recibe el grupo.*

*Ya casi para finalizar se aclara el tamaño de la carpeta o archivo y, por último, la fecha última en la que fue modificado dicho elemento en cuestión.*

*La organización de la información se ve más o menos así, dentro de la terminal, al ser ejecutado el comando: “ls -l”:*



*Ahora, el comando ls -la al ser ejecutado hace lo mismo que ls -l: sólo que "****ls -la****" va más allá y también muestra los archivos, o contenido en sí, ocultos.*

*Los archivos o elementos ocultos se caracterizan por empezar su nombre o etiqueta con un punto, tal que así, por ejemplo: ".cache". El punto, recuerde, lo tienen al frente del nombre del archivo o elemento: justo en el inicio.*

***“ls -lh”****, muestra la misma información que ls -la; pero por medio de y con las unidades de tamaño (bytes, kilobytes, gigabytes, terabytes, etc).*

*Ahora bien, si queremos ver los archivos (ocultos o no) de un directorio en especifico, también podríamos hacerlo. Supongamos que deseamos ver los archivos ocultos del directorio “etc”. Sería así el comando a ejecutar: “****ls -la /etc****”. En caso tal que queramos ver la misma información que solicitamos en el anterior directorio (etc), pero en el directorio principal que está conteniendo a nuestro directorio actual, sería así:* ***“ls -la ..”****.**Sin embargo, puede pasar que no se alisten todos los archivos ocultos o no de dicho directorio; entonces, en caso que desee ver absolutamente todo (de la A a la Z) del directorio “etc” puede ejecutar más bien el siguiente comando: “ls -la /etc |more”. De la misma manera se hace con cualquier otro directorio en caso que desee visualizar, en su totalidad, sus archivos ocultos o no; adicionalmente vaya presionando la tecla “espacio” para ir desplazandose por el listado y poder observar así todos los archivos del directorio en cuestión. En caso que desee salir del listado sólo presione la tecla “Q” de su teclado. (o bien puede ejecutar también el comando “clear”). Otra manera de ejecutar este comando: “ls -la /etc |less”.*

*TALLER COMPLEJO SOBRE COMANDOS SHELL (Introducción a fondo de Shell)*

*12. Escriba un script que imprima el tipo de archivo llamado*iamafile*. El archivo* iamafile *estará en el directorio*/tmp *cuando ejecutemos su script.*

file /tmp/iamafile

*13. Cree un enlace simbólico a* /bin/ls*, llamado* \_\_ls\_\_*. El enlace simbólico debe crearse en el directorio de trabajo actual.*

ln -s /bin/ls \_ls\_

*14. Cree un script que copie todos los archivos HTML del directorio de trabajo actual al padre del directorio de trabajo, pero solo copie los archivos que no existían en el padre del directorio de trabajo o que eran más recientes que las versiones en el padre del directorio de trabajo. directorio de trabajo. Puede considerar que todos los archivos HTML tienen la extensión* .html

cp \*.html ..

*15. Cree un script que mueva todos los archivos que comienzan con una letra mayúscula al directorio* /tmp/u*. Puede asumir que el directorio* /tmp/u *existirá cuando ejecutemos su script*

mv [[:upper:]]\* /tmp/u

*16. Cree una secuencia de comandos que elimine todos los archivos del directorio de trabajo actual que terminan con el carácter* ~*.*

rm \*[~]

*17. Cree un script que cree los directorios* welcome/*,* welcome/to/ *y* welcome/to/holberton *en el directorio actual. Solo se le permite usar dos espacios en su script, no más.*

mkdir -p welcome/to/holberton/

*18. Escriba un comando que enumere todos los archivos y directorios del directorio actual, separados por comas (*,*).*

*Los nombres de directorio deben terminar con una barra (*/*)*

*Se deben enumerar los archivos y directorios que comienzan con un punto (*.*)*

*La lista debe estar ordenada alfabéticamente, excepto los directorios* . *y* ..*que debería aparecer al principio*

*Solo se utilizan dígitos y letras para ordenar; Los dígitos deben ir primero*

*Puede asumir que todos los archivos con los que probaremos tendrán al menos una letra o un dígito*

*La lista debe terminar con una nueva línea.*

ls -amp (o ls -mpa, ls -pam...)

***Introducción a Shell***

***¿Qué es “Shell”?***

*En pocas palabras, el shell es un programa que toma comandos del teclado y se los da al sistema operativo para que los ejecute. En los viejos tiempos, era la única interfaz de usuario disponible en un sistema similar a Unix como Linux. Hoy en día, tenemos interfaces gráficas de usuario (GUI) además de interfaces de línea de comandos (CLI) como el shell.*

*En la mayoría de los sistemas Linux, un programa llamado bash (que significa Bourne Again SHell, una versión mejorada del programa shell original de Unix, sh, escrito por Steve Bourne) actúa como programa shell. Además de bash, hay otros programas de shell disponibles para sistemas Linux. Estos incluyen: ksh, tcsh y zsh.*

***¿Qué es una "Terminal"?***

*Es un programa llamado emulador de terminal. Este es un programa que abre una ventana y le permite interactuar con el shell. Hay un montón de emuladores de terminal diferentes que podemos usar. Algunas distribuciones de Linux instalan varios. Estos pueden incluir gnome-terminal, konsole, xterm, rxvt, kvt, nxterm y eterm.*

*Si bien hay varios emuladores de terminal diferentes, todos hacen lo mismo. Nos dan acceso a una sesión de shell.*

***No estás operando como root, ¿verdad?***

***Si el último carácter de su indicador de shell es # en lugar de $, está operando como superusuario****. Esto significa que tiene privilegios administrativos. Esto puede ser peligroso, ya que puede eliminar o sobrescribir cualquier archivo del sistema. A menos que necesite absolutamente privilegios administrativos, no opere como superusuario.*

***datos:***

*El primer directorio del sistema de archivos se denomina directorio raíz* ***(the root directory).***

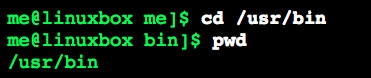
*El directorio en el que nos encontramos se llama directorio de trabajo* ***(working directory).***

*Cuando iniciamos sesión por primera vez en nuestro sistema Linux, el directorio de trabajo se establece en nuestro directorio de inicio* ***(home directory)****. Aquí es donde colocamos nuestros archivos. En la mayoría de los sistemas, el directorio de inicio se llamará / home / user\_name, pero puede ser cualquier cosa según los caprichos del administrador del sistema.*

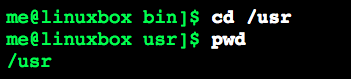
*Un nombre de ruta es la ruta que tomamos a lo largo de las ramas del árbol para llegar al directorio que queremos. Los nombres de ruta se pueden especificar de dos formas diferentes; nombres de ruta absolutos o nombres de ruta relativos* ***(absolute pathnames or relative pathnames)*** *Veamos primero con nombres de ruta absolutos.*

*Donde una ruta absoluta comienza desde el directorio raíz y conduce a su destino, una ruta relativa comienza desde el directorio de trabajo. Para hacer esto, usa un par de notaciones especiales para representar posiciones relativas en el árbol del sistema de archivos. Estas notaciones especiales son "." (punto) y ".." (punto punto). Los* ***"."*** *la notación se refiere al directorio de trabajo en sí y la notación* ***".."*** *se refiere al directorio principal del directorio de trabajo* ***(the working directory's parent directory)****. Así es como funciona. Pongamos un ejemplo:*

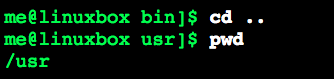
*Cambiemos el directorio de trabajo a* ***/usr/bin*** *nuevamente:*



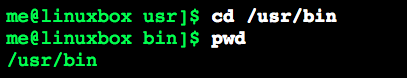
*Bien, ahora digamos que queremos cambiar el directorio de trabajo al padre de* ***/usr/ bin****, que es* ***/usr****. Podríamos hacerlo de dos formas diferentes.* ***Primero, con un nombre de ruta absoluto:***



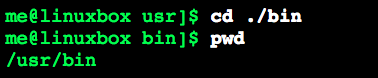
***O, con un nombre de ruta relativo:***



*Dos métodos diferentes con resultados idénticos. ¿Cuál deberíamos usar? ¡El que requiere menos mecanografía! Asimismo, podemos cambiar el directorio de trabajo de* ***/usr*** *a* ***/usr/bin*** *de dos formas diferentes.* ***Primero usando un nombre de ruta absoluto:***



***O, con un nombre de ruta relativo:***



*Con todo y eso hay una forma más sencilla de hacer lo anterior:* ***cd bin***

***Algunos atajos:***

*Si escribimos* ***cd,*** *seguido de nada, cd cambiará el directorio de trabajo a nuestro directorio de inicio (home directory).*

*Un atajo relacionado es escribir* ***cd ~ nombre\_usuario****. En este caso,* ***cd*** *cambiará el directorio de trabajo al directorio de inicio del usuario especificado.*

*Escribir* ***cd-*** *cambia el directorio de trabajo al anterior.*

***Algunos concejos:***

***Los nombres de archivo en Linux, como Unix, distinguen entre mayúsculas y minúsculas.*** *Los nombres de archivo "Archivo1" y "archivo1" se refieren a archivos diferentes.*

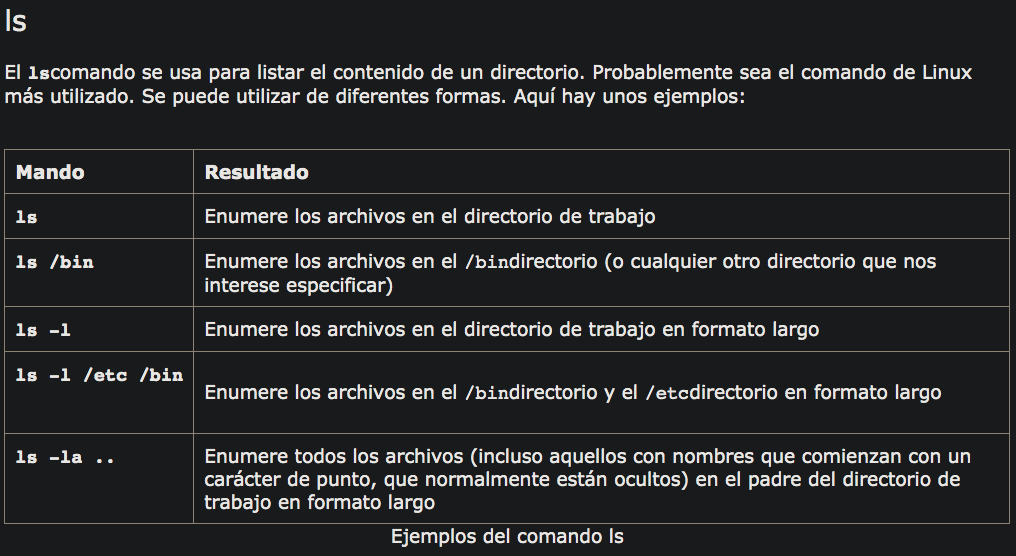
***Linux no tiene el concepto de una "extensión de archivo" como los sistemas Windows.*** *Puede nombrar los archivos como desee.* ***Sin embargo, aunque Linux en sí no se preocupa por las extensiones de archivo, muchos programas de aplicación sí lo hacen.***

*Aunque Linux admite nombres de archivo largos que pueden contener espacios incrustados y caracteres de puntuación,* ***limite los caracteres de puntuación al punto, guión y subrayado****.* ***Lo más importante es que no inserte espacios en los nombres de los archivos.*** *Si desea representar espacios entre palabras en un nombre de archivo, utilice caracteres de subrayado. Te lo agradecerás más tarde.*

***Otros comandos...***

***ls (lista de archivos y directorios)   
less (ver archivos de texto)   
file (clasifica el contenido de un archivo)***

*El comando* ***ls*** *se usa para listar el contenido de un directorio. Probablemente sea el comando de Linux más utilizado. Se puede utilizar de diferentes formas. Aquí hay unos ejemplos:*



*Estos ejemplos también señalan un concepto importante sobre los comandos. La mayoría de los comandos funcionan así:*

***comando -opciones argumentos,***

*donde* ***comando*** *es el nombre del comando,* ***-options*** *es uno o más ajustes al comportamiento del comando y* ***argumentos*** *es una o más "cosas" sobre las que opera el comando.* ***En el caso de ls, vemos que ls es el nombre del comando, y que puede tener una o más opciones, como -a y -l, y puede operar en uno o más archivos o directorios (argumentos).***

***Practiquemos un poco lo anterior (comando -opciones argumentos)...***

***Quick question:*** *Write a script that lists all files (even ones with names beginning with a period character, which are normally hidden) in* ***the current directory*** *and* ***the parent of the working directory*** *and the* ***/boot*** *directory (in this order), in long format.*

***Prueba Rapida:*** *Escriba un script que enumere todos los archivos (incluso los que tienen nombres que comienzan con un carácter de punto, que normalmente están ocultos) en el* ***directorio actual****, en e****l padre del directorio de trabajo*** *y en el directorio* ***“/ boot”*** *(en este orden), en formato largo.*

*R/* ***ls -la . .. /boot***

***Quick question:*** *Display current directory contents by the next form:*

*-Long format, -with user and group IDs displayed numerically And - hidden files (starting with .)*

***Prueba Rapida:*** *Muestra el contenido actual del directorio.*

*-Formato largo, -con ID de usuario y grupo mostrados numéricamente, - archivos ocultos (comienzan con .).*

*R/* ***ls -anl***

***“n”*** *vendría siendo un elemento más del comando “ls” que nos permite conocer justamente el “ID de usuario y grupo mostrados numéricamente”.* ***Estos elementos puede alternalos en cualquier orden y llamar los que usted desee, eso no importa. Puede ejecutar, por ejemplo, este mismo comando de la siguiente forma: “ls -nla” “ls -lan” “ls -aln”...***

***En caso que le pidan hacer un listado con unas caracteristicas especificas y no recuerda que elemento las muestra, recuerde que puede hacer uso del comando “man”. Para nuestro caso, “man” le alistará todos los elementos de los que se compone el comando “ls”, ejecute tal que así: “man ls”. Esto último es aplicable para todos los comandos de la Shell.***

***Comando File***

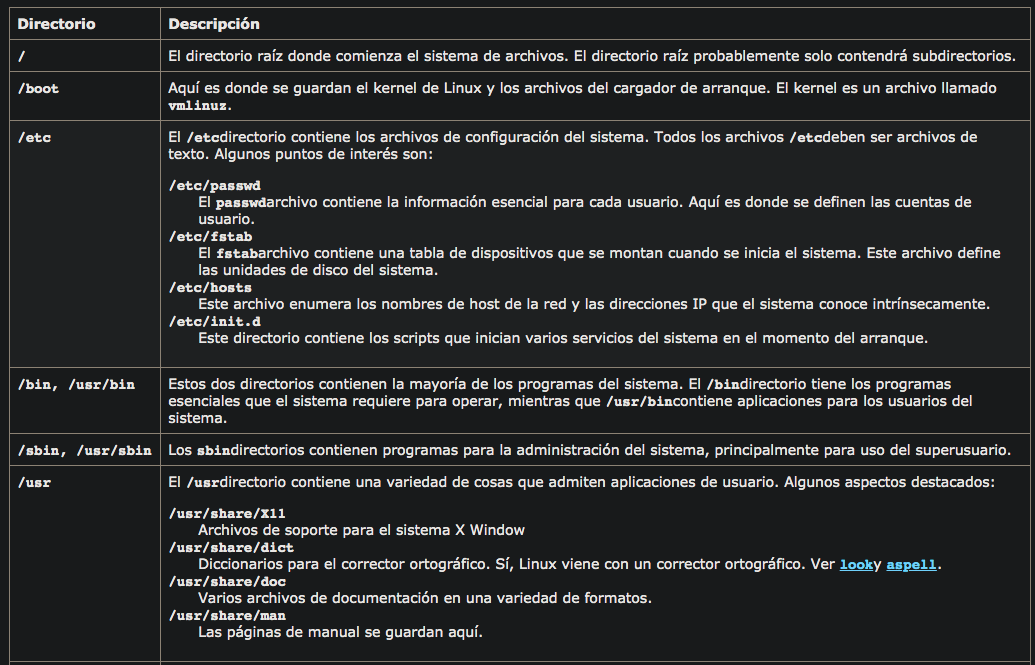
*Mientras deambulamos por nuestro sistema Linux, es útil determinar qué tipo de datos contiene un archivo antes de intentar verlo. Aquí es donde entra el comando* ***File.******File examinará un archivo y nos dirá qué tipo de archivo es****. No hace falta que el archivo tenga una extensión para que File determine su tipo. Para usar el programa de archivos, simplemente escribimos:*

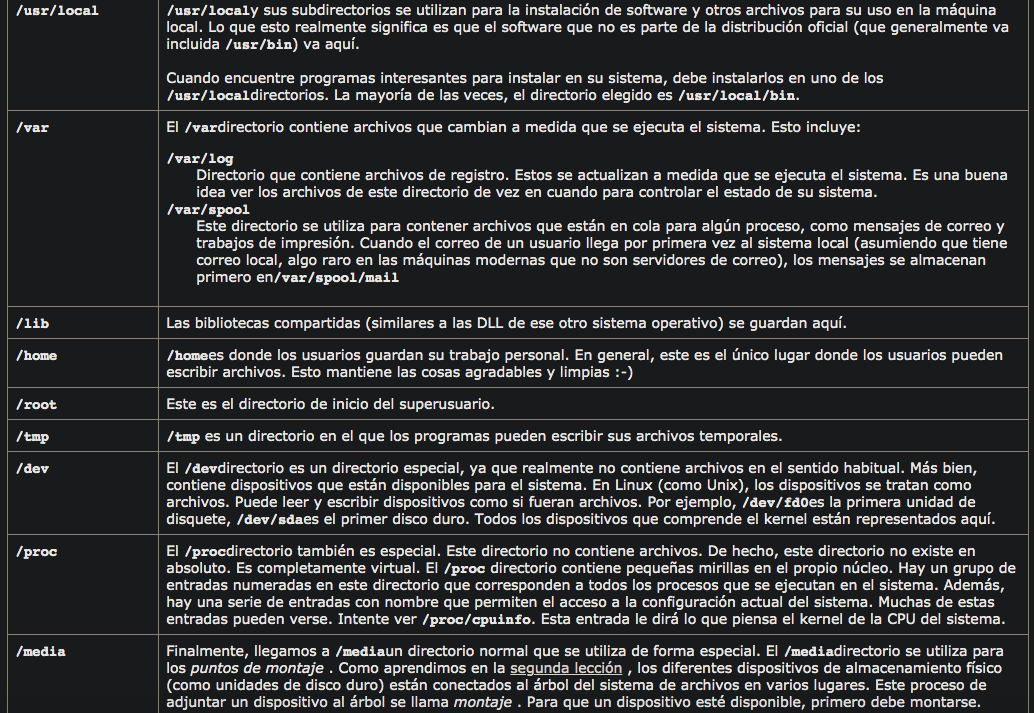
***File “nombre\_del\_archivo”.***

*El comando File principalmente puede identificar carpetas, archivos y datos; sin embargo, hay toda una gama de variedades que puede identificar.*



*también puede ejecutar:* ***File \*, para saber todos los tipos de archivos que tiene en su directorio actual.*** *Es hora de hacer nuestro recorrido. La siguiente tabla enumera algunos lugares interesantes para explorar. Esta no es de ninguna manera una lista completa, pero debería resultar una aventura interesante. Para cada uno de los directorios enumerados a continuación, haga lo siguiente:*







***ln -s:*** *Es el comando que realmente crea los enlaces simbolicos (ayudadose del elemento* ***-s****). Toca aclararlo porque en la anterior imagen no se especifica* ***(ln*** *sólo crea enlaces, más no necesariamente simbolicos. Ya se dará cuenta de esto). Cuando se le presente un caso similar, debe revisar todos los elementos por los que se compone un comando (repito); eso se puede saber ejecutando un comando* ***“man”*** *sobre el comando de interés, tal que así:* ***“man ln”.***

***Ejercicio de ejemplo:***

*\* Cree un enlace simbólico a* ***/bin/ls,*** *llamado* ***\_\_ls\_\_.*** *El enlace simbólico debe crearse en el directorio de trabajo actual.*

*R/* ***ln -s /bin/ls \_\_ls\_\_***

*Esta forma de especificar, desde un directorio padre* ***(/bin)****, la creación de un enlace simbolico en una carpeta contenida por /bin* ***(una sub-carpeta)****; de tal manera que, en la linea de código se vea reflejada: un comando, su elemento, ~~la dirección del directorio destino~~ y el argumento, no es igual de posible para crear archivos o directorios. Lo normal es que la linea de código de todo comando de la Shell, al ser ejecutado, se vea de la siguiente forma:* ***comando -opciones argumentos (recuerdelo).***

*Hay, principalmente, dos tipos de uso para ejecutar el comando* ***ln.*** *Uno para crear enlaces físicos propiamente hablando* ***(ln)*** *y, segundo, para crear enlaces simbolicos* ***(ln -s). Veamos las diferencias.***

*Se le llama* ***Hard-link*** *a los enlaces creados con el comando* ***“ln”;*** *es decir, a los enlaces físicos.* ***ln*** *puede apuntar solamente a archivos (crea el enlace físico sobre ellos). Esa copia de información, el enlace resultante (reflejada en un archivo) se guarda en un* ***nuevo nodo*** *distinto al del archivo de origen; es decir, en un nuevo espacio de memoria. Esto qué significa? significa que, cualquier tipo de manipulación sobre el archivo de origen creado, si bien sí se reflejaría sobre el enlace físico (archivo apuntado), no se podría eliminar el enlace físico eliminando solamente al archivo de origen que lo crea (porque* ***no están*** *en un mismo espacio de memoria).*

***Ej. de creación de un “hard-link”:***

***ln text.txt copia\_text.txt,****donde* ***text.txt*** *es el archivo de origen y* ***copia\_text.txt*** *sería el enlace físico. (el archivo original* ***text.txt*** *apunta “copia” su información en un enlace físico llamado:* ***copia\_text.txt****)*

*Ahora, se le llama* ***Symbolic-link (o soft-link)*** *a los enlaces creados con el comando* ***“ln -s”;*** *es decir, a los enlaces simbolicos.* ***ln -s*** *puede apuntar tanto a carpetas como hacia archivos. Esa copia de información, el enlace resultante (reflejado en un archivo o directorio) se guarda en el* ***mismo nodo*** *del archivo de origen; es decir, acá no hay un nuevo espacio de memoria. Esto qué significa? significa que, cualquier tipo de manipulación sobre el archivo de origen creado se verá reflejada en su totalidad sobre el enlace resultante (guardado en un archivo o directorio apuntado), incluso si se borra el archivo de origen se borrará también el enlace simbólico (porque* ***están*** *en un mismo espacio de memoria).*

***Ej. de creación de un “symbolic-link”:***

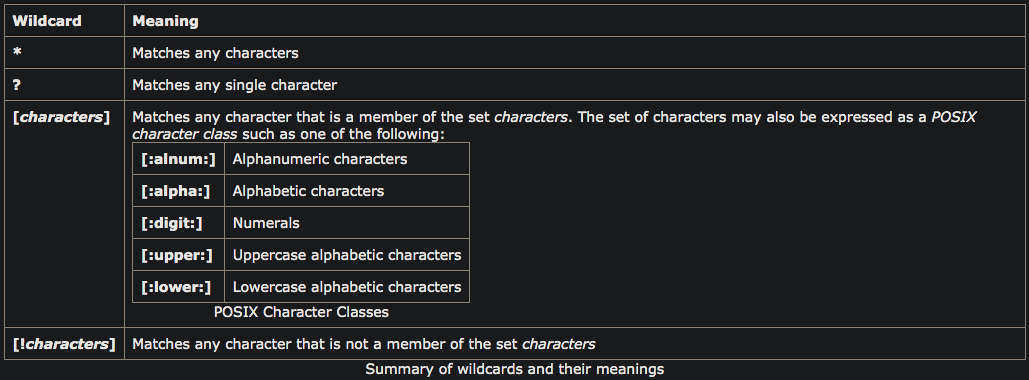
***ln -s text.txt copia\_text.txt,****ó****ln -s directory copia\_directory,****donde* ***directory*** *es el archivo de origen y* ***copia\_directory*** *sería el enlace simbólico. (el archivo original* ***directory*** *apunta “copia” su información en un enlace simbólico llamado:* ***copia\_directory****)*

***Otros comandos y sus comodines (Wildcards)...***

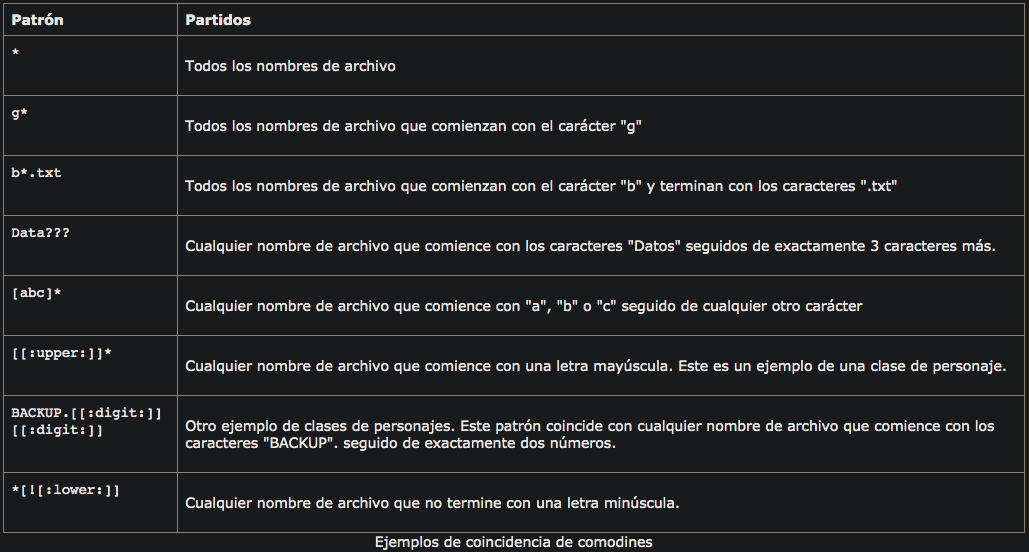
***cp - copiar archivos y directorios  
mv - mover o cambiar el nombre de archivos y directorios  
rm - eliminar archivos y directorios  
mkdir - crear directorios***

***Comodines (Wildcards)***

*Antes de comenzar con nuestros comandos, primero veremos una función de shell que hace que estos comandos sean tan poderosos. Dado que el shell usa tanto nombres de archivos, proporciona caracteres especiales para ayudarlo a especificar rápidamente grupos de nombres de archivos.* ***Estos caracteres especiales se denominan comodines****.* ***Los comodines le permiten seleccionar nombres de archivos (files) basados en patrones de caracteres. La siguiente tabla enumera los comodines y lo que seleccionan:***



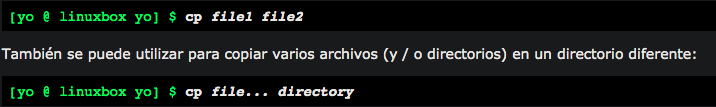
***Usando comodines, es posible construir criterios de selección muy sofisticados para nombres de archivos (files)****. A continuación, se muestran algunos ejemplos de patrones y sus coincidencias:*



***Podemos usar comodines con cualquier comando que acepte argumentos de nombre de archivo (files).***

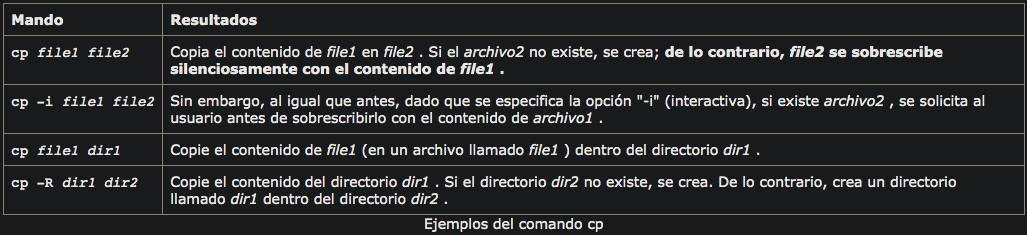
***Cp***

*El programa cp copia archivos y directorios. En su forma más simple, copia un solo archivo:*



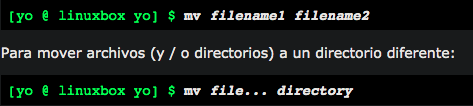
*Una nota sobre la notación: ... significa que un elemento se puede repetir una o más veces.*

***Otros ejemplos útiles de cp y sus opciones incluyen:***

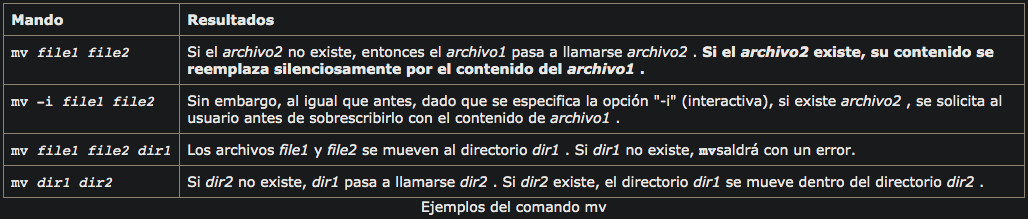


***Mv***

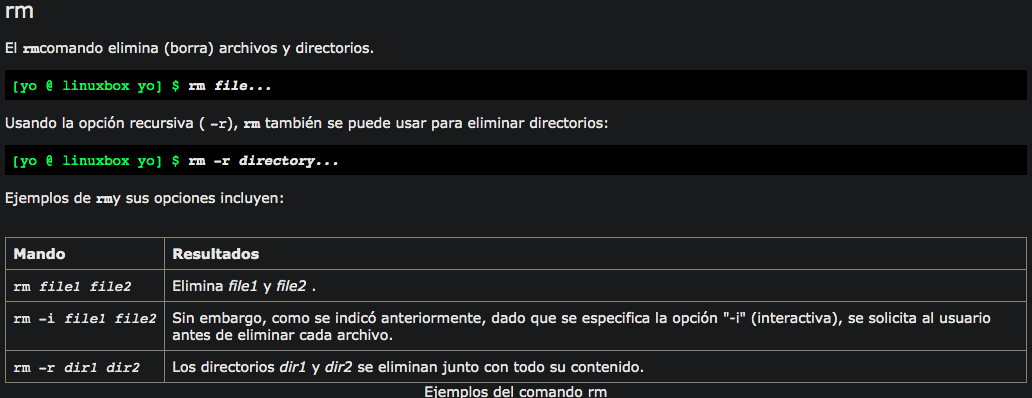
*El comando mv mueve o cambia el nombre de archivos y directorios dependiendo de cómo se use. Moverá uno o más archivos a un directorio diferente o cambiará el nombre de un archivo o directorio. Para cambiar el nombre de un archivo, se usa así:*



***Ejemplos de mv y sus opciones incluyen:***



***Rm***



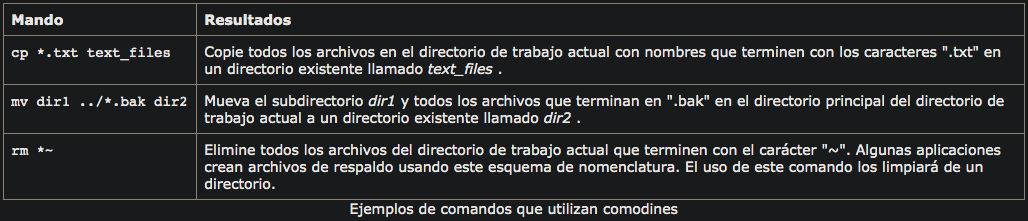
***Mkdir***

*El comando mkdir se usa para crear directorios. Para usarlo, simplemente escriba:*



***Usar comandos con comodines***

***Dado que los comandos que hemos cubierto aquí aceptan múltiples nombres de archivos y directorios como argumentos, puede usar comodines para especificarlos. Aquí están algunos ejemplos:***



***El arterisco (\*) si se da cuenta tiene la función de ser un totalizador; se refiere, llama, invoca y afecta sobre todos los objetos de una carpeta determinada y/o bajo unos parametros de identificación establecidos. De no especificarse nada, esta función afectará sobre el directorio de trabajo actual y sobre la totalidad de los objetos que hay dentro de él.***

***Pongamos en práctica todo lo anterior con el taller sobre comandos Shell*** *(Introducción a fondo de Shell) que está justo antes de este apartado****.***

***Trabajar con comandos***

*Hasta ahora, hemos visto varios comandos y sus misteriosas opciones y argumentos. En esta lección, intentaremos eliminar algo de ese misterio. Introduciremos los siguientes comandos.*

***type - Mostrar información sobre el tipo de comando  
which - Localizar un comando  
help - Mostrar página de referencia para shell incorporado  
man - Mostrar una referencia de comando en línea***

***¿Qué son los "comandos"? Los comandos pueden ser de 4 tipos diferentes:***

***Un programa ejecutable*** *como todos esos archivos que vimos en / usr / bin. Dentro de esta categoría, los programas pueden ser binarios compilados como programas escritos en C y C ++, o programas escritos en lenguajes de scripting como shell, Perl, Python, Ruby, etc.*

***Un comando integrado en el propio shell.*** *bash proporciona una serie de comandos llamados internos de shell. El comando cd, por ejemplo, es un shell incorporado.*

***Una función de shell. Estos son scripts de shell en miniatura incorporados al entorno.*** *Cubriremos la configuración del* ***entorno y la escritura de funciones de shell en lecciones posteriores, pero por ahora, tenga en cuenta que existen.***

***Un alias.*** *Comandos que podemos definir nosotros mismos, construidos a partir de otros comandos. Esto se cubrirá en una lección posterior.*

***Identificación de comandos***

*A menudo es útil saber exactamente cuál de los cuatro tipos de comandos se está utilizando y Linux proporciona un par de formas de averiguarlo.*

***Type*** *es un shell incorporado que muestra el tipo de comando que ejecutará el shell, dado un nombre de comando en particular. Funciona así:*

***“Type comando”,*** *donde "comando" es el nombre del comando que queremos examinar.*

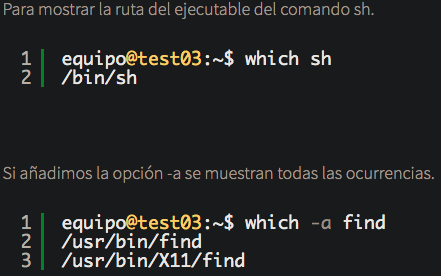
***Which***

*A veces, hay más de una versión de un programa ejecutable instalada en un sistema. Si bien esto no es muy común en los sistemas de escritorio, no es inusual en servidores grandes. Para determinar la ubicación exacta de un ejecutable dado, se usa el comando* ***which:***



*El comando which de linux nos muestra dónde se encuentra un ejecutable.*

*El modo de empleo es: which [-a] “nombre\_del\_ejecutable”*



*Al igual, puede especificar varios programas en el mismo comando. Por ejemplo:*

***Which*** *firefox gimp banshee;*

*esto devolverá los siguientes resultados:*

*/usr/bin/firefox  
/usr/bin/gimp  
/usr/bin/banshee*

***Entonces,por qué usaria el comando which?***

*Imagine que sabe que se ha instalado un programa, pero que por alguna razón no se ejecutará. Es muy probable que se deba a que la carpeta en la que se ha instalado el programa no está en la ruta.*

*Usando el comando* ***which*** *puede localizar dónde está el programa y navegar hasta la carpeta en la que el programa debe ejecutarlo.*

***which*** *sólo funciona para programas ejecutables, no incorporados ni alias que son sustitutos de programas ejecutables reales.*

***Obtener documentación de comandos***

*Con este conocimiento de lo que es un comando, ahora podemos buscar la documentación disponible para cada tipo de comando.*

***Help***

***bash*** *tiene una función de ayuda integrada disponible para cada una de las funciones integradas de shell. Para usarlo, escriba "ayuda" seguido del nombre del shell incorporado. Opcionalmente, podemos agregar la opción -m para cambiar el formato de la salida.*

***Shell, Permissions***

***Introducción:*** *Saber a qué grupos pertenece un usuario. En sistemas operativos Linux, los usuarios pueden pertenecer a mas de un grupo. ¿Cómo saber a qué grupos pertenece un usuario? Veamos dos formas para saber a que grupos pertenece un usuario:*

*1) Comando* ***groups:*** *Ejecutando el comando groups, nos aparecerán los grupos a los que pertenece el usuario actual.*

*Si ejecutamos* ***groups*** *seguido de un nombre de usuario, veremos los grupos a los que pertenece el usuario indicado. Ejemplo:*

*[root@LINUX1 ~]#* ***groups root******root****: root bin daemon sys adm disk wheel.*

*2) Comando* ***id:*** *El comando id, nos muestra información acerca de la identidad del usuario.*

*Si ejecutamos el comando* ***id,*** *con los parámetros* ***-Gn,*** *conseguiremos obtener la lista de grupos a los que pertenece el usuario.*

*[root@LINUX1 ~]#* ***id -Gn root******root:*** *bin daemon sys adm disk wheel*

***Elementos usados: El parámetro -G, muestra todos los grupos a los que pertenece; mientras que el parámetro -n muestra el nombre en vez del número de grupo.***

***Permisos***

*Los sistemas operativos similares a Unix, como Linux, se diferencian de otros sistemas informáticos en que no solo son multitarea sino también multiusuario .*

*¿Qué significa esto exactamente? Significa que más de un usuario puede operar la computadora al mismo tiempo. Si bien una computadora de escritorio o portátil solo tiene un teclado y un monitor, aún puede ser utilizada por más de un usuario. Por ejemplo, si la computadora está conectada a una red oa Internet, los usuarios remotos pueden iniciar sesión a través de ssh(shell seguro) y operar la computadora. De hecho, los usuarios remotos pueden ejecutar aplicaciones gráficas y mostrar el resultado en una computadora remota. El sistema X Window lo admite.*

*La computadora admitiría muchos usuarios al mismo tiempo.Para que esto fuera práctico, se tuvo que idear un método para proteger a los usuarios unos de otros. Después de todo, no queremos que las acciones de un usuario bloqueen la computadora, ni permitiríamos que un usuario interfiera con los archivos que pertenecen a otro usuario.*

*Esta lección cubrirá los siguientes comandos:*

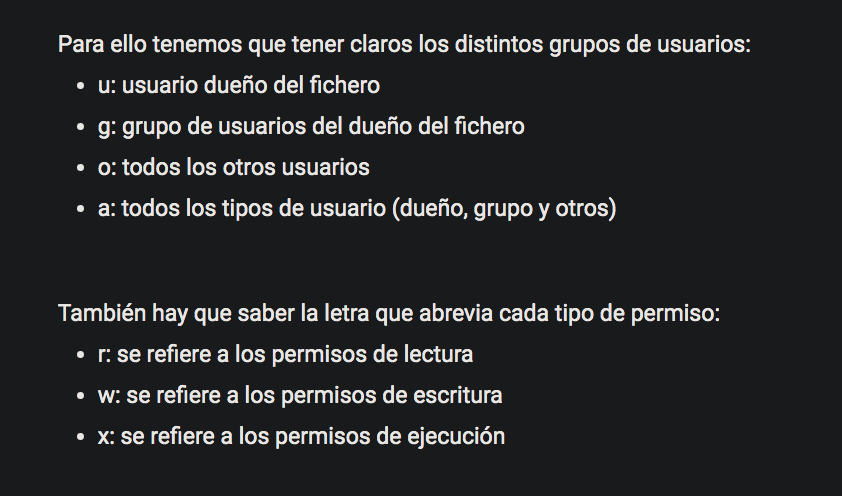
***chmod - modificar los derechos de acceso a archivos   
su - convertirse temporalmente en superusuario  
sudo - convertirse temporalmente en superusuario  
chown - cambiar la propiedad del archivo  
chgrp - cambiar la propiedad del grupo de un archivo***

***Permisos de archivos***

*En un sistema Linux, a cada archivo y directorio se le asignan derechos de acceso para el propietario del archivo, los miembros de un grupo de usuarios relacionados y todos los demás. Se pueden asignar derechos para leer un archivo, escribir un archivo y ejecutar un archivo (es decir, ejecutar el archivo como un programa).*

***Para ver la configuración de permisos para un archivo, podemos usar el”ls” comando. Como ejemplo, veremos el bashprograma que se encuentra en el /bin directorio:***

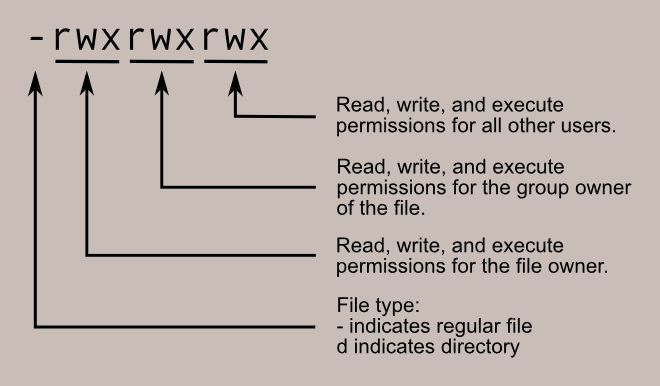




***Retomemos bin/directorio.***

***El archivo "/bin/bash" es propiedad del usuario "root".  
El archivo es propiedad del grupo "root".  
El superusuario tiene derecho a leer, escribir y ejecutar este archivo.  
Los miembros del grupo "root" también pueden leer y ejecutar este archivo.  
Todos los demás pueden leer y ejecutar este archivo.***

*En el diagrama siguiente, vemos cómo se interpreta la primera parte del listado. Consiste en un carácter que indica el tipo de archivo, seguido de tres conjuntos de tres caracteres que transmiten el permiso de lectura, escritura y ejecución para el propietario, el grupo y todos los demás.*



***chmod***

*El comando* ***chmod*** *se usa para cambiar los permisos de un archivo o directorio. Para usarlo, especificamos la configuración de permisos deseada y el archivo o archivos que deseamos modificar. Hay dos formas de especificar los permisos. En esta lección nos centraremos en uno de estos, llamado método de notación octal. Sin embargo, vamos a dejar una pequeña introducción de la otra forma primero. Partamos de este par de ejemplos:*

***chmod u+x \**** *= dar permiso al user de ejecutar todos los archivos contenidos en el directorio actual ("chmod u+x . ": hace lo mismo).*

***chmod u+x \*.c*** *= dar permiso al user de ejecutar todos los archivos tipo”.c" contenidos en el directorio actual.*

***chmod u+x file =*** *dar permiso al usuario de ejecutar solamente un archivo en cuestión, llamado en el ejemplo como: "file".*

***chmod +x .*** *= le da el permiso a todos los usuarios de ejecutar todos los archivos contenidos en el directorio actual.*

***chmod +x .c*** *= le da el permiso a todos los usuarios (user, grupo y otros) de ejecutar todos los archivos "tipo .c" contenidos en el directorio actual.*

***chmod +x file*** *= le da el permiso a todos los usuarios (user, grupo y otros) de ejecutar puntualmente un archivo en cuestión, llamado en el ejemplo como: "file".*

***Ejercicio rápido:*** *Escriba un script que agregue permiso de ejecución al propietario y al propietario del grupo, y permiso de lectura a otros usuarios, al archivo* ***hello.***

*R/* chmod ug+x,o+r hello

*Ahora, si representamos cada uno de los tres conjuntos de permisos (propietario, grupo y otros) como un solo dígito, tenemos una forma bastante conveniente de expresar las posibles configuraciones de permisos. Por ejemplo, si quisiéramos establecer en “some\_file” permiso de lectura y escritura para el propietario, pero quisiéramos mantener el archivo privado para otros, haríamos lo siguiente:*

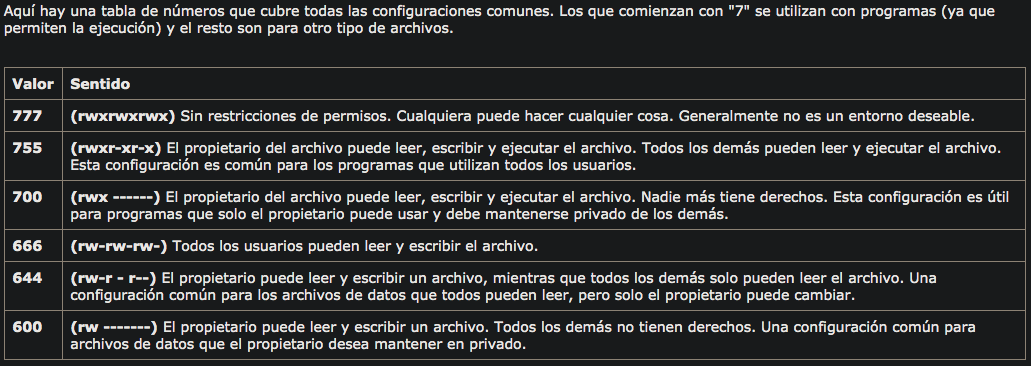


*Veamos ahora el siguiente ejemplo:* ***chmod 754 mi-archivo.txt***

*Aquí los dígitos 7, 5 y 4 representan individualmente los permisos para el usuario, grrupo y otros, en ese orden. Cada dígito es una combinación de los números 4, 2, 1 y 0:*

***4 - significa “leer”  
2 - significa “escribir”  
1 - significa “ejecutar”, y  
0 - significa “sin permiso”.***

*Entonces, qué podemos concluir de “chmod 754 mi-archivo.txt”? 7 es la combinación de permisos 4 + 2 + 1 (leer, escribir y ejecutar), 5 es 4 + 0 + 1 (leer, no escribir y ejecutar), y 4 es 4 + 0 + 0 (leer, no escribir y no ejecutar).*

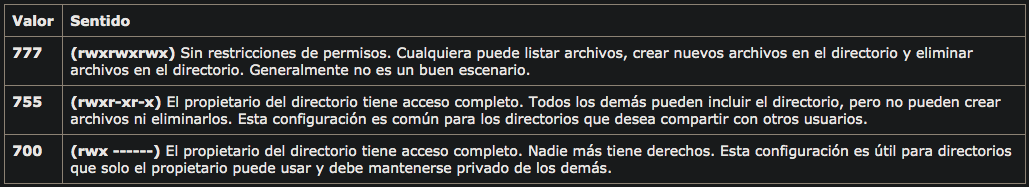


***Permisos de directorio***

*El comando chmod también se puede utilizar para controlar los permisos de acceso a los directorios. Nuevamente, podemos usar la notación octal para establecer permisos, pero el significado de los atributos r, w y x es diferente:*

***r: permite que se enumere el contenido del directorio si también se establece el atributo x.  
w: permite crear, eliminar o renombrar archivos dentro del directorio si el atributo x también está configurado.  
x: permite ingresar a un directorio (es decir cd dir).***

*Aquí hay algunas configuraciones útiles para directorios:*



***Cambiar la propiedad del archivo: chown***

*Podemos cambiar el propietario de un archivo usando el comando* ***chown.*** *Acá ejemplo con un archivo llamado“some\_file”: Supongamos que queremos cambiar el propietario de "yo" a "usted". Podríamos:*



*Tenga en cuenta que para cambiar el propietario de un archivo, debemos tener privilegios de superusuario. Para hacer esto, nuestro ejemplo empleó el sudo comando para ejecutar chown. Este comando funciona de la misma manera en directorios que en archivos.* ***Ahora veremos varias diversas formas de uso, que son bastantes, que se le podría dar a este comando.***

*Como ya se mencionó al principio, el comando* ***chown*** *le permite cambiar el propietario y el grupo del archivo a través de la línea de comando. A continuación se muestra la sintaxis genérica del comando:*

***chown [OPTION]... [OWNER][:[GROUP]] FILE...***

***1. ¿Cómo cambiar el propietario de un archivo?***

*Considere el siguiente ejemplo:*

*- Aquí, el propietario del archivo es 'himanshu' y el grupo al que pertenece también es 'himanshu'. Ahora, para cambiar el propietario a, por ejemplo 'root', use el siguiente comando:*

***chown root file1***

***2. ¿Cómo cambiar el grupo de un archivo?***

*Cambiar un grupo es similar a cambiar el propietario. La única diferencia está en la sintaxis del comando, que es la siguiente:*

***chown :[group-name] [file-name]***

*Entonces, suponga que el requisito es cambiar el grupo de 'file1' a 'root'. Entonces el comando sería:*

***chown :root file1***

***3. ¿Cómo cambiar tanto el propietario como el grupo de un archivo?***

*Para cambiar tanto el propietario como el grupo de archivos, utilice la siguiente sintaxis:*

***chown [new-owner]:[new-group] [file-name]***

*Entonces, en nuestro caso, para cambiar el propietario y el grupo existentes de 'root' a 'himanshu', usaremos el siguiente comando:*

***chown himanshu:himanshu file1***

***4. ¿Cómo cambiar el propietario o el grupo (o ambos) después de verificar el propietario/grupo existente?***

*Puede haber situaciones en las que desee verificar primero el propietario grupo existente de un archivo antes de realizar cualquier cambio. Entonces, para esos casos, puede usar la opción de línea de comando* ***--from****. Esta opción requiere que menciones el nombre del propietario/grupo que deseas verificar.*

***chown --from=[curr-own]:[curr-group] [new-owner]:[new-group] [filename]****; por ejemplo,*

***chown --from=root:himanshu himanshu:root file1***

*El comando anterior* ***verificará*** *si el propietario existente es 'root' y el grupo es 'himanshu'.* ***En caso afirmativo, el propietario se cambiará a 'himanshu' y el grupo se convertirá en 'root'.***

***5. ¿Cómo elegir la información del propietario/grupo de un archivo de referencia?***

*También puede haber situaciones en las que desee que Chown recopile información relacionada con el propietario y el grupo de un archivo, en lugar de ingresarla manualmente en la línea de comando. Para esos casos, puede usar la opción de línea de comandos* ***--reference*** *. Esta opción requiere que ingrese el nombre del archivo de referencia.*

***chown --reference=[ref-file-name] [filename]****; por ejemplo,*

***chown --reference=file2 file1***

*Entonces, el comando anterior* ***copiará*** *la información del propietario y del grupo del archivo2 al archivo1.*

***6. ¿Cómo hacer que chown opere en archivos y directorios de forma recursiva?***

*Para hacer que el comando chown opere recursivamente en archivos y directorios, use la opción de línea de comandos -R****.***

***chown -R [new-owner]:[new-group] [directory-name-or-path]***

*Para aquellos que no lo saben,* ***recursivo*** *significa que la operación se realizará para todos los archivos en el directorio dado, así como para los archivos y directorios dentro de todos los subdirectorios.*

***7. ¿Cómo hacer que Chown suprima los mensajes de error?***

*Puede haber ocasiones en las que el comando chown que ejecuta dé un error. Por ejemplo, el siguiente comando cuando se ejecuta en mi sistema:*

***chown --from=himanshu:himanshu himanshu:root file4****,**dio el siguiente error:*

*“chown: cannot access 'file4': No such file or directory”.*

*Ahora suponga que el requisito es que la herramienta no muestre tales errores.* ***Entonces esto puede ser posible usando la opción de línea de comando -f .***

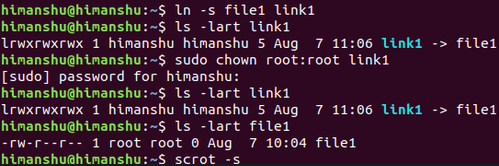


*La captura de pantalla mencionada anteriormente confirma que la opción de línea de* ***comandos -f*** *suprime los errores / advertencias.*

***8. ¿Cómo funciona chown con enlaces simbólicos?***

***De forma predeterminada, si intenta cambiar la propiedad del usuario y del grupo de un enlace simbólico, no habrá ningún cambio. En cambio, el archivo al que se vincula obtendrá estos cambios.***

*Por ejemplo, la captura de pantalla a continuación muestra que creé un enlace simbólico 'link1' cuyas propiedades de usuario y grupo están configuradas como 'himanshu'. Luego ejecuté el comando chown para cambiar de usuario y grupo a 'root'.* ***Pero el comando no tuvo ningún efecto en el archivo de enlace simbólico; en cambio, es 'file1' (al que apunta el enlace simbólico) cuyas propiedades de usuario y grupo cambiaron.***



*Sin embargo, si lo desea, puede anular este comportamiento predeterminado usando la* ***opción -h. Con la opción -h se puede cambiar la propiedad de un enlace simbólico encontrado y no la del archivo o directorio al que apunta el enlace simbolico.*** *Veamoslo con un ejemplo: escriba un script que cambie el propietario y el propietario del grupo del archivo* ***\_hello*** *a* ***betty y holberton*** *respectivamente.*

***R/ chown -h betty:holberton \_hello***

***9. ¿Cómo cambiar el propietario y el grupo de un directorio?***

*Al igual que la forma en que lo hace con los archivos. A continuación se muestra un ejemplo:*

***chown root:root ./test-dir/***

***10. ¿Cómo hacer que Chown muestre los detalles de la operación en la salida?***

***Si desea que el comando chown muestre detalles de la operación que realiza, use la opción de línea de comandos -v.*** *Por ejemplo, este comando:*

***sudo chown himanshu:himanshu link1 -v,***

*produce la siguiente salida:* ***“changed ownership of 'link1' from root:root to himanshu:himanshu”***

***Ahora, hay otra opción -c que también actúa como -v, solo que no muestra ningún detalle cuando no se cambia nada.***

***Un par de ejercicios de práctica:***

*A. Cambie el propietario al perfil de usuario "sam" para el archivo "personal.file".*

***R/ chown sam personal.file***

*B. Cambie de forma recurrente el propietario al perfil de usuario "larry" para el subdirectorio "moe.dir" y todos los archivos y subdirectorios debajo de este directorio.*

***R/ chown -R larry moe.dir***

*C. Cambie el propietario al identificador de usuario "500" para el archivo "su archivo".*

***R/ chown 500 your.file***

*D. Cambie el propietario al perfil de usuario "sam" y el grupo al perfil de grupo "abbey" para el archivo "memo.txt".*

***R/ chown sam:abbey memo.txt***

***Muchos de estos elementos son perfectamente aplicables, bajo la misma lógica, al comando chmod. Analicelo!***

***De gran interés: la sintaxis para quitar un permiso a chmod (o cualquier comando que manipule sobre los usuarios en general) es similar a cuando se les da el permiso en cuestión; sólo que, en vez de poner “+”, se coloca: “-”.***

***Cambiar la propiedad del grupo***

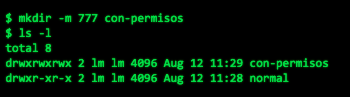
*La propiedad del grupo de un archivo o directorio se puede cambiar con* ***chgrp.*** *Este comando se usa así:*



*En el ejemplo anterior, cambiamos la propiedad del grupo “some\_file” a su grupo anterior "new\_group". Debemos ser el propietario del archivo o directorio para realizar un* ***chgrp.***

***Bonus track: crear un directorio y asignarle permisos.***

*Cuando creas un directorio con el comando linux* ***mkdir*** *puedes asignar los permisos al directorio en la misma sentencia y para eso existe la opción* ***-m.******Esta opción te permite asignar permisos de la misma forma en la que se usa el comando linux chmod.*** *Por ejemplo, si queremos crear un directorio con permisos en linux de 777 podemos usar la siguiente sentencia:*



*En este ejemplo se crea el directorio «con-permisos» y se le asignan los permisos de lectura, escritura y acceso para el propietario, el grupo y otros. El linux command mkdir es muy útil, pues te ahorra tiempo si sabes como usarlo.*

***Shell, I/O Redirections and filters***

*En un principio se tiene por sabido que Shell, por defecto, tiene una serie de entradas y salidas por la ejecución de sus comandos, y que de ahí derivan los términos técnicos de: entrada estándar* ***(Standart******Input)*** *y salida estándar* ***(Standart******Output);*** *respectivamente, la entrada suele ser el teclado (lo que se “meta” por tecla) y la salida normalmente es la pantalla (el resultado que se nos arroja sobre ella). Sin embargo, adicionalmente a este par de “Standarts” que usted conoce, hay otro:* ***“Standart******error”,*** *su función es la de generar los mensajes de error que se producen al ejecutar de forma incorrecta algún comando. Los tres tipos de Standart tienen un número asociado y de identificación a ellos. En su orden,*

***Standart******Input = 0<; (teclado)  
Standart Output = 1>; (pantalla)  
Standart Error = 2>; (pantalla)***

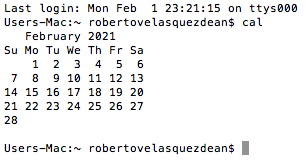
***Introducción a redirecciones en Standart Output (entrada estándar) y Standart Input (salida estándar)***

*En muchos casos es útil que, en vez de utilizar las sálidas o las entradas por defecto, las podamos re-direccionar mejor a un destino propio a nuestra elección.* ***Si reemplazamos la entrada por tecla, entonces tendríamos que re-direccionar nuestro mecanismo de entrada a una entrada que no sea con tecla*** *(he aquí el misterio resuelto sobre las re-direcciones en entrada: cualquiera que sea lo que haya cambiado su entrada por defecto, la cuál es por teclado, se vuelve en una re-dirección de entrada y debe conocer cómo se ejecuta su sintaxis en una línea de código);* ***lo propio para las salidas, si reemplazamos la salida por pantalla, tendríamos que elegir también una salida que no sea una salida por pantalla. Todo lo que me dé una salida que no sea por pantalla, muy probablemente será una re-dirección de salida.*** *En ambos, para lograr ese cometido, se maneja una sintaxis propia.*

***Sintaxis de Standart Output: comando > fichero*** *(esto es la representación más básica de una re-dirección de salida: el comando ejecutado ya no se reflejará en pantalla como salida, sino que se guardará más bien en un fichero)*

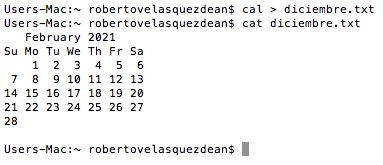
*Veamos cómo se ve con un ejemplo: ejecutemos el comando* ***cal (cal imprime un calendario; por defecto, el del mes actual)****.*

*Naturalmente si hubieramos ejecutado el comando* ***“cal”*** *como de costumbre (a secas), se hubiera visto reflejado la entrada estándar por defecto que es por* ***teclado*** *y, además, el comando “cal” estaría generando también su salida en la salida estandar por defecto que es por* ***pantalla.*** *Tal que así:*



*Ahora bien, si cambiamos esas salidas y entradas por defecto, estaríamos justamente haciendo una redirección de salida y entrada, respectivamente.* ***Por ejemplo, que en vez de una salida por pantalla, el comando “cal” nos guarde el resultado en un fichero. Veamos:***

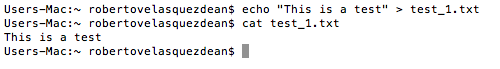
***cal > diciembre.txt*** *(esto efectivamente cambia la salida por pantalla por guardar el resultado en un fichero; así hemos re-direccionado entonces nuestra salida, la cual por defecto es por pantalla). Tal que así:*

***nota:*** *cat* ***muestra (display)***

***Observe que para visualizar el resultado de “cal” tuve que llamar, por medio del comando cat, el fichero “diciembre.txt” que fue donde efectivamente se guardó la salida del comando cal, lo que reemplazó la salida por pantalla.***

*Recuerda cuando introducimos el comando “echo” en las primeras hojas de este documento? Pues bien, lo habíamos ejecutado de la siguiente forma:* ***echo "This is a test" > test\_1.txt;***

*Esa linea de código ejecutada anteriormente es una muestra más de una re-dirección de salida aunque usted, en ese entonces, no lo supiera. Lo que está haciendo el comando* ***“echo”*** *es re-direccionar su salida al fichero* ***“test\_1.txt”*** *en vez de pasarla por pantalla. Entonces, para poder visualizar por pantalla la declaración de echo tendría que llamar, acá también, al fichero (test\_1.txt) por medio del comando* ***cat.*** *Tal que así:*



*Si no se hubiera re-direccionado la salida con el comando echo, echo hubiera reflejado la salida por pantalla también (como de costumbre, como lo hace por defecto). Tal que así:*

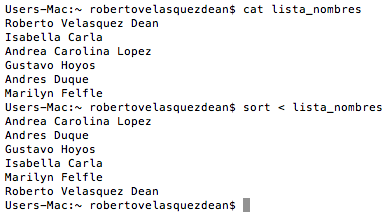
***nota:*** *echo* ***imprime (print)***

***Ahora, vamos a introducir a fondo el tema de: redirección de entrada.***

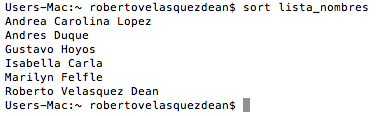
***Sintaxis de Standart Input: comando < fichero*** *(esto es la representación más básica de una re-dirección de entrada: no se ejecuta propiamente un comando por medio del teclado como entrada, sino que un fichero toma la función de un comando como tal (para que el comando afecte sobre él).*

*Veamos cómo se ve con un ejemplo: ejecutemos el comando* ***sort****.* ***Sort,*** *por si lo desconoce, es un comando que imprime en pantalla las líneas de un archivo ordenadas alfabéticamente. Para ejecutarlo basta con:* ***sort archivo.txt***

*Supongamos, por ejemplo, que tenemos un listado desordenado de nombres en un archivo llamado “lista\_nombres” y queremos que una re-dirección de entrada nos permita organizar dicha lista de forma ordenada alfabeticamente. Cómo lo haríamos bajo una re-dirección de entrada?*



*Sin embargo, esto mismo podría hacerse sin necesidad de re-direccionar la entrada, tal que así:*



*Entonces, por ejemplo ¿en qué caso particular es util o necesario hacer una re-dirección de entrada?*

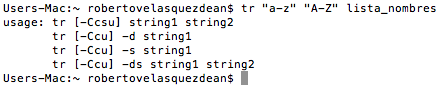
*Trabajemos, por ejemplo, con el comando* ***tr.***

***tr,*** *por si no lo conoce, es un comando que transforma un texto de un formato a otro. Por poner un ejemplo, podemos ejecutar a* ***tr*** *para que nos traduzca o nos transforme un texto de minusculas a mayusculas, tal que así:*

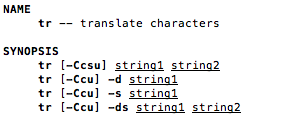


***nota:*** *En caso de que ya desee* ***parar******la ejecución*** *de este comando debe presionar “****Control + z****” de manera simultanea.*

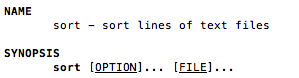
*Esto mismo que acabamos de hacer se podría ejecutar en caso que quisieramos pasarle la función del comando* ***tr*** *a un fichero o archivo de texto? Es decir,* ***¿el comando tr permite que uno le pase un argumento o “file”? Probemos.***



***Mmm parece que no funciona...*** *Salgamos de esa duda revisando los límites del comando tr ejecuntando un comando* ***man.***

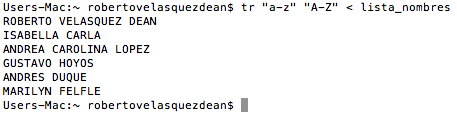


*Logra ver alguna palabra “file” dentro de la “synopsis” de tr? Ahora revisemos el comando* ***sort.***



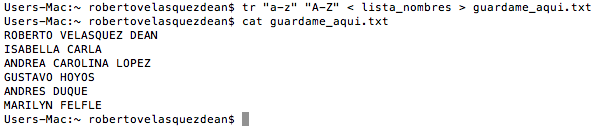
*Se da cuenta de la diferenica? Sort sí permite, tal como lo vimos en su ejemplo, manipular directamente sobre un fichero (”file”, se puede ver en “synopsis”); en cambio,* ***tr no!*** *he ahí la importancia de hacer re-direcciones de entrada;* ***que algunos comandos de nuestro interés no podran manipular directamente sobre los ficheros si no se asisten con una re-dirección de entrada: y tr es el caso.***

*Pasemos ahora sí, con una re-dirección de entrada, los parametros definidos anteriormente en tr para que puedan afectar sobre un fichero o archivo de texto (en nuestro caso, para que afecte sobre el archivo “lista\_nombres”); tal que así:*

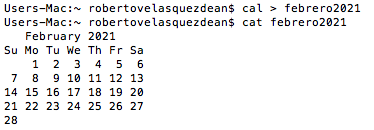


***Listo, funciona!***

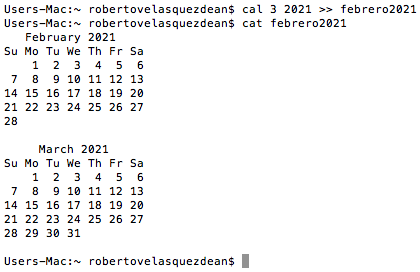
***Ahora, podríamos hacer una redirección de entrada y una redirección de salida a la vez? Sí, claro. Lo podemos probar en el mismo ejemplo.*** *Por lo general se emplea cuando, además de hacer una redirección de entrada, no queremos que nuestro resultado se muestre por pantalla. Tal que así:*



*Esto sería suficiente para entender lo puntualmente necesario sobre redirecciones de entrada y de salida.* ***Sin embargo, también podemos hacer una doble redirección.*** *Retomemos por ejemplo nuestro primer caso, el calendario; pero ahora vamos a situarnos sobre nuestro mes actual (en el momento en que escribo estas lineas estamos en Febrero del año 2021)*



*Si quisieramos ahora colocar, por ejemplo, el mes de marzo del 2021 sobre el mismo fichero; de tal manera que, el fichero guarde toda la información anual del calendario mes a mes, no podríamos redireccionar la salida solamente con esto: “>”, no. Esto (>) lo que hará es “machacarme” el contenido actual, sobreescribir sobre lo ya escrito en dicho fichero y reemplazarlo. Para que la nueva información sea registrada sin anularme la previa, debe hacer una doble-redirección de salida; la cual, se simboliza así:* ***“>>”****. Probemos.* ***Vamos a colocar el mes de marzo del 2021 (cal 3 2021)\* sobre el mismo fichero sin borrar el mes de febrero. Tal que así:***



*Y así sucecivamente puede seguir agregando más información sobre el mismo fichero sin borrar la información ya establecida. Todo con la herramienta de dobre-redirección. Si desea pruebe agregando ahora el mes de abril del año 2021.*

***\**** *le pasamos al comando* ***cal*** *los paramentos de* ***“mes”*** *y* ***“año”,*** *respectivamente.*

***Ahora vamos a introducir las redirecciones del “Standart Error”.***

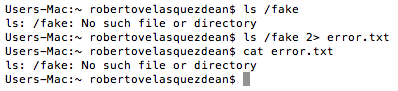
*Los mensajes de error que nos muestran por pantalla también pueden ser redireccionados, esto se puede con una redirección del tipo “Standart Error”.*

***Sintaxis de Standart Error: comando argumento 2> fichero*** *(esto es la representación más básica de una re-dirección de salida del tipo “error”: el comando ejecutado ya no se reflejará en pantalla como salida, sino que se guardará más bien en un fichero).*

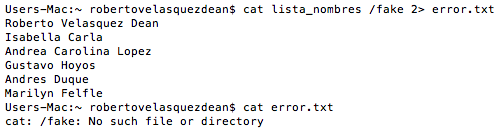
*Veamos cómo se ve con un ejemplo: supongamos que queremos hacer un listado de un directorio inexistente (un inminente error, cierto?) veamos:*



*Si queremos redireccionar este error a un fichero, pues lo que podríamos hacer es invocar un “Standart error”, (y su salida se verá reflejada propiamente en ese fichero y no en la pantalla) tal que así:*



*Ahora bien, esta herramienta puede distinguir entre lo que sería un error a lo que no; es decir, “Standart error” puede guardar en un fichero aquello que esté mal de nuestra línea de código sin dejar de ejecutar lo que sí estaría bien. Veamoslo.*

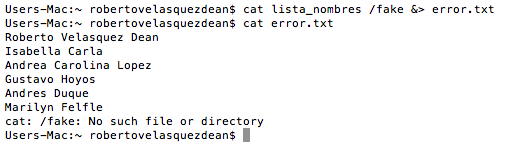


*Mire que en un principio sí me mostró el primer argumento (lista\_nombres) que está correcto, más no el segundo (/fake) que está incorrecto; este último argumento lo redireccionamos a un fichero llamado “error.txt”, el cuál luego en una segunda linea de códigos decidimos mirarlo con el comando cat (cat error.txt).*

***Trabajando mutuamente entre “Standart Output” & “Standart Error”.***

***Si por ejemplo deseamos guardar una salida correcta y una salida con error en el mismo fichero...***

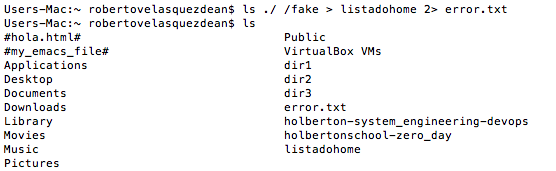
***Sintaxis: comando argumento1 argumento2 &> fichero*** *(uno de estos argumentos es correcto y el otro no, sin embargo ambos registrarían su información en el mismo fichero). Ej.:*



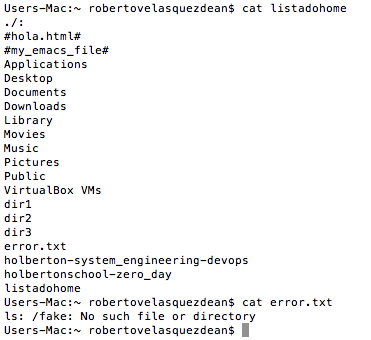
*Observe que ambos argumentos, uno con salida correcta y otro con error, fueron guardados en un mismo fichero; el fichero llamado “error.txt”.*

***Si por ejemplo deseamos guardar una salida correcta y una salida con error en ficheros diferentes...***

***Sintaxis: comando argumento1 argumento2 > fichero1 2> fichero2*** *(donde el argumento correcto se guardaría en “fichero1” y el argumento con errores se guardaría en “fichero2”). Ej.: Probemos ejecutando un comando* ***ls*** *sobre el directorio actual y sobre un directorio inexistente, tal que así:*



*Efectivamente se crearon dos ficheros; uno para guardar la salida correcta* ***(> listadohome)*** *y otro para guardar la salida con error* ***(2> error.txt)****. Comprobemos esto:*



***Introducción a Filtros.***

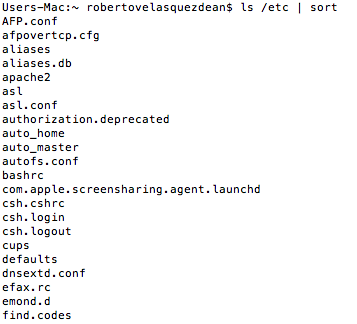
*Los “pipes” o barras tienen la función de filtrar de forma más detallada la información que se pide sobre un comando; lo hace a través de enviar el resultado de un comando inicial a un comando segundario y éste, a su vez, filtra mucho más la información al enviar su resultado a un tercer comando; es decir, se re-direcciona la salida de un comando sobre la entrada de otro para ser más precisos en la busqueda de una información especifica de nuestro interés. Esto en relación a: ser más puntuales en la información que queremos sobre ciertas especificaciones de directorio o archivo.*

*Por ejemplo, podemos especificar que, en la visualización de un archivo x por medio del comando* ***cat,*** *podamos ver dicha información pero con la condición de que se nos presente de forma paginada (pasando la info. con “enter” o con “space”), y eso lo hace el comando:* ***more****, tal que así:*



*Es decir se han especificado, incluso más, los resultados de la información que estamos buscando al ejecutar más comandos en su misma línea de códigos.*

*Bajo esa misma lógica también podríamos, por ejemplo, hacer un listado de la carpeta /etc con comando* ***ls****; pero, no sólo eso, sino que haga la lista de sus archivos y sub-directorios y los ordene también de forma alfabetica con un comando* ***sort****; tal que así:*



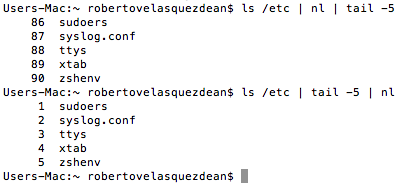
*Y si deseo el mismo listado anterior, pero ahora con una especificación más... lo deseo paginado. Se podría? Pues, sí. Tal que así:*



*Sinopsis de more:* ***more archivo.txt***

*Sinopsis de sort:* ***sort archivo.txt***

*Tenga en cuenta algo, los comandos se ejecutan según su orden; y cada comando posterior ejecutará su función a partir de los comandos previamente ya ejecutados. Es decir, el segundo comando afecta sobre la línea de código a partir del efecto que dejó la función del primer comando: es de esta forma cómo opera la lógica que hay detrás de los filtros con tuberias (pipelines). Observe bien con un ejemplo lo que trato de plantear (de paso daremos introducción a dos comandos nuevos:* ***nl*** *y* ***tail):***



***nl*** *(no confundir con* ***ln****) es un comando que indica a qué número de linea pertenece cada linea de un archivo enlistado (enumera las lineas por las que se compone un archivo). En cuanto a* ***tail****, es la anti-tesis de* ***head.*** *Expliquemos ambos.*

***Tail*** *muestra en pantalla las últimas líneas de un archivo. Por defecto siempre muestra 10, pero podemos indicarle un número diferente de líneas a visualizar usando el parámetro* ***-n****; por ejemplo: “tail -n archivo.txt”* ***(donde tail es el comando, -n es la opción y archivo.txt es el argumento). Head****, hace justamente lo contrario: muestra las primeras líneas de un archivo. Al igual que tail, muestra por defecto las 10 primeras líneas pero podemos indicarle un número diferente usando el parámetro -n también: “head -n archivo.txt”.*

*Explicado esto, entonces ya puede lograr ver la diferenecia en la manera en cómo se ordenan los comandos haciendo uso de filtros? Saque su propia conclusión.*

***Otros comandos de interés:***

***Wc*** *imprime en pantalla la cantidad de saltos de línea, palabras y bytes totales que contenga un archivo. Para usarlo con un archivo cualquiera ejecutamos:* ***wc archivo.txt***

***Cmp*** *compara el contenido de dos archivos y devuelve 0 si los archivos son idénticos ó 1 si los archivos tienen diferencias. En caso de error devuelve:* ***-1****. Para ejecutarlo basta con:* ***cmp -s archivo1 archivo2.*** *Cmp también puede mostrar algo de información sobre las diferencias pero para un reporte más detallado tenemos el siguiente comando.*

***Diff,*** *al igual que cmp, compara el contenido de dos archivos pero en lugar de devolver un valor imprime en pantalla un resumen detallado línea a línea de las diferencias. Ejecutarlo es tan simple como:* ***diff archivo1.txt archivo2.txt***

*comando curioso* ***Write:*** *ejecutando comando “write” literalmente puedes chatear con otro usuario, cual sea de tu dispositivo, dentro de la terminal. Sólo ejecutas esto: write “nombre\_de\_usuario” y listo. Ya puedes dejarle un mensaje a otro usuario desde la consola. También es usado a veces para imprimir texto, se usa casi que de la misma forma que la función “printf” o “puts”. Pruebela*

***otros...***

***Id*** *nos permite conocer el usuario actual del dispositivo.*

***adduser*** *nos permite agregar otro usuario y grupo al dispositivo, así se ve:* ***adduser “nombre\_del\_nuevo\_usuario”****; luego, se le pedirá que inserte las contraseñas correspondientes a ese nuevo usuario que estás creando. Listo, se crea, pero, debe autocompletar con un nombre completo, los demás datos son opcionales. (ejecute con sudo antes de crear nuevo usuario).*

***useradd*** *va más allá, porque incluso, además de registrarte un nuevo usuario puedes, por medio de opciones, especificar a que grupo va a pertenecer de entrada dicho usuario nuevo (-g) o cuál sería su directorio de trabajo (-d), entre otros.   
El listado de opciones más comunes:*

***-g “nombre\_del\_grupo”:*** *nombre del grupo principal al que debe permanecer el usuario.*

*-****m:*** *crea el directorio home del usuario si no existe, con el mismo nombre del usuario. Esta opción es muy importante porque crea además la estructura de carpetas necesaria (descargas, documentos, escritorio, etc.). Sin éstas, el sistema no nos dejará “logearnos” desde el entorno gráfico.*

***-d “nombre\_del\_directorio”:*** *le indicamos el directorio de trabajo del usuario.*

***-s “nombre\_del\_interprete”:*** *intérprete de comandos que utilizará el usuario, normalmente /bin/bash*

*Acá su sinopsis:* ***useradd opciones “nombre\_del\_nuevo\_usuario”,****& acá la estructura más recomendable de usuario:* ***useradd -g “nombre\_del\_grupo” -m “nombre\_del\_nuevo\_usuario”.***

*Si se da cuenta, esta forma de crear un usuario, inicialmente no le solicita crearle una contraseña también; por lo que tendrá problemas para “loguearse” en caso que no configure posteriormente una contraseña para este user. La solución a este problema está en la “nota de interés” del apartado de: passwd.*

***groupadd*** *nos permite crear un grupo. Sinopsis:* ***groupadd “nombre\_del\_nuevo\_grupo”***

***passwd*** *nos permite cambir la contraseña del usuario actual. Sinopsis:* ***passwd***

***nota de interés:*** *sin embargo, si desea darle o cambiarle la contraseña a un usuario diferente al suyo (al actual); debe, antes que nada, contar con los permisos de superusuario; es decir, saber la contraseña del dueño del dispositivo prácticamente. Hace un “sudo su” y luego ejecuta lo siguiente:* ***passwd “nombre\_del\_usuario\_o\_nuevo\_usuario”.*** *(generalmente esta forma de invocar el comando es usada para asignarle contraseña a un usuario creado con el comando “useradd”... o para cambiarsela... mientras no sea usada para malas intenciones).*

***usermod:*** *nos permite modificar las caracteristicas de un usuario ya creado. Funciona igual que “useradd”, pero sobre un usuario ya creado.* ***Sinopsis: usermod -g “nuevo\_grupo\_destino” “nombre\_de\_usuario”. (ejemplo para cambiar grupo)***

***userdel:*** *nos permite eliminar un usuario.* ***Sinopsis: userdel “nombre\_de\_usuario”.***

***groupdel:*** *nos permite eliminar un grupo.* ***Sinopsis: groupdel “nombre\_de\_grupo”. Aclaración:*** *se puede eliminar grupos a los usuarios siempre y cuando el grupo a eliminar no sea el grupo primario del usuario en cuestión.*

***su - “user”:*** *nos permite cambiar de usuario:* ***Sinopsis: su - “nombre\_de\_usuario”.***

***PARA MÁS INFO, visite: https://www.neoguias.com/crear-usuario-linux-comando-useradd/***

***Comando Alias***

*El comando alias hace posible ejecutar cualquier comando o grupo de comandos (incluidas las opciones, los argumentos y la redirección ) ingresando una cadena preestablecida (es decir, una secuencia de caracteres ).*

*Es decir, permite a un usuario crear nombres simples o abreviaturas (incluso que constan de un solo carácter) para comandos independientemente de cuán complejos sean los comandos originales y luego usarlos de la misma manera que se usan los comandos ordinarios.*

*Un comando es una instrucción dada por un usuario para decirle a una computadora que haga algo. Los comandos generalmente se emiten escribiéndolos en la línea de comandos (es decir, una interfaz de usuario de texto completo) y luego presionando la tecla ENTER, que los pasa al shell . Un shell es un programa que proporciona la interfaz de usuario tradicional de solo texto para sistemas operativos similares a Unix . Su función principal es leer comandos y luego ejecutarlos (es decir, ejecutarlos).*

*El comando alias está integrado en varios shells, incluidos ash , bash (el shell predeterminado en la mayoría de los sistemas Linux ), csh y ksh . Es una de las varias formas de personalizar el shell (otra es establecer variables ambientales ). Los alias son reconocidos solo por el shell en el que se crean, y se aplican solo para el usuario que los crea, a menos que ese usuario sea el usuario raíz (es decir, administrativo), que puede crear alias para cualquier usuario.*

***Listado y creación de alias***

*La sintaxis general del comando alias varía un poco según el shell. En el caso del shell bash es*

***alias [-p] [name ="comando"]***

***Cuando se usa sin argumentos y con o sin la opción -p , alias proporciona una lista de alias que están vigentes para el usuario actual.***

*Es probable que algunos de los alias enumerados sean alias de todo el sistema que se aplican a todos los usuarios y se crean automáticamente para cada nuevo usuario de un shell en particular. Los alias para cualquier otro shell se pueden ver cambiando primero a ese shell y luego usando el comando alias como se indicó anteriormente.*

*nombre es el nombre del nuevo alias y valor es el comando (s) que inicia. El nombre de alias y el texto de reemplazo pueden contener cualquier entrada de shell válida, excepto el signo igual (=).*

*Los comandos, incluidas las opciones, los argumentos y los operadores de redireccionamiento, están todos encerrados dentro de un solo par de comillas, que pueden ser comillas simples o dobles. No se permiten espacios antes o después del signo igual. Se puede crear cualquier número de alias simultáneamente encerrando el nombre en cada par de nombre-valor entre comillas.*

*Como ejemplo trivial de creación de alias, el alias p podría crearse para el comando pwd de uso común , que muestra la ubicación actual del usuario en la estructura del directorio (y que es una abreviatura del directorio de trabajo actual ), escribiendo el siguiente comando y luego presionando la tecla ENTER:*

*alias p="pwd"*

*Luego, para mostrar la ubicación actual, en lugar de escribir pwd , el usuario solo tendría que escribir la letra py presionar la tecla ENTER, es decir,*

*p*

*Se puede crear un alias con el mismo nombre que el nombre principal de un comando (es decir, un comando sin opciones ni argumentos). En tal caso, es el alias el que se llama (es decir, se activa) primero cuando se usa el nombre, en lugar del comando con el mismo nombre. Por ejemplo, se podría crear un alias llamado ls para el comando ls -al de la siguiente manera:*

*alias ls="ls -al"*

*ls es un comando de uso común que, por defecto, enumera los nombres de los archivos y directorios dentro del directorio actual (es decir, el directorio en el que el usuario está trabajando actualmente). La opción -a le indica a ls que también muestre los archivos y directorios ocultos , y la opción -l le indica que proporcione información detallada sobre cada archivo y subdirectorio.*

*Dicho alias se puede deshabilitar temporalmente y se puede llamar al comando principal precediéndolo directamente (es decir, sin espacios entre ellos) con una barra invertida, es decir,*

*\ls*

*No importa si se utilizan comillas simples o dobles al crear un alias. Puede ser un poco más fácil usar comillas simples porque esto evita la necesidad de usar simultáneamente la tecla Mayús. Por lo tanto, el ejemplo anterior podría haberse escrito como*

*alias ls='ls -al'*

*Este ejemplo podría simplificarse aún más, nuevamente sin ningún efecto adverso en el rendimiento, utilizando un solo carácter en lugar de dos caracteres para el nombre de alias, por ejemplo:*

*alias l='ls -al'*

*Además de las opciones, los argumentos también se pueden incluir en los valores de alias. Por ejemplo, para que el alias ls siempre muestre el contenido del directorio / etc , se podría reescribir como:*

*alias l='ls -al /etc'*

*Se pueden incluir varios comandos en el mismo alias insertándolos dentro del mismo par de comillas y separándolos con punto y coma. Por ejemplo, el alias pl podría crearse para iniciar primero pwd y luego lanzar inmediatamente ls:*

*alias pl='pwd; ls'*

*Incluso se pueden crear alias para llamar a otros alias. Por ejemplo, si el alias ls como se mostró anteriormente ya se había creado, entonces pl lo lanzará en el ejemplo anterior; de lo contrario, lanzará el comando ls convencional. Si los alias p y l que se muestran en los ejemplos anteriores ya se han creado, entonces el ejemplo anterior podría escribirse alternativamente como*

*alias pl='p; l'*

*El siguiente es un ejemplo de cómo crear dos alias separados simultáneamente, en contraste con la creación de un solo alias que lanza dos comandos separados:*

*alias p="pwd"; l="ls -al"*

*Como ejemplo de un alias para una serie de comandos vinculados por una tubería (representada por una línea vertical), el alias dir se puede crear para generar una lista de los nombres e información sobre todos los subdirectorios en el directorio actual:*

*alias dir="ls -al | grep ^d"*

*Aquí ls -al obtiene una lista de todos los archivos y directorios del directorio actual. Su salida es enviada por la tubería al filtro grep , que luego busca líneas que comienzan con la letra d (ya que todos los directorios tienen una línea devuelta por ls -al que comienza con d). El signo de intercalación (es decir, el carácter angular que apunta hacia arriba) antes de d le dice a grep que busque solo las líneas que comienzan con esa letra.*

*No solo se pueden usar opciones y argumentos en los comandos que un alias puede sustituir, sino que también se pueden usar con un alias que ya se ha creado. Como ejemplo trivial, suponiendo que se crea el alias l para el comando ls:*

*alias l="ls -a"*

*Entonces, el alias l podría usarse con cualquier argumento con el que pueda usarse el comando ls. Por ejemplo, para listar los archivos y directorios en el directorio / etc:*

*l /etc*

*El alias l también se podría usar con cualquier opción con la que se pueda usar el comando ls. Por ejemplo:*

*l -l /etc*

*El comando alias es inusual porque solo tiene una opción. Esa opción, -p , le dice que muestre una lista de los alias del usuario actual en el shell actual. Esto puede ser útil si se usa al crear un alias, pero, por supuesto, es redundante cuando el comando alias se usa sin argumentos.*

***Usos de los alias***

*Hay varios tipos de usos para los alias. Incluyen:*

*(1) Reducir la cantidad de escritura necesaria para los comandos o grupos de comandos que son largos y / o tediosos de escribir. Estos comandos podrían incluir la apertura de un archivo que se usa con frecuencia para estudiar o editar.*

*Por ejemplo, si un usuario accede a menudo al archivo de configuración del servidor web Apache , que es /etc/httpd/conf/httpd.conf en Red Hat Linux 9, y usa el editor de texto gedit para leerlo, dicho usuario puede escribir cada vez:*

*gedit /etc/httpd/conf/httpd.conf*

*Sin embargo, esto podría volverse tedioso rápidamente. Sería mucho más fácil hacer este comando en un alias y darle un nombre corto, tal vez incluso una sola letra, como un :*

*alias a="gedit /etc/httpd/conf/httpd.conf"*

*Luego, siempre que el usuario quiera abrir el archivo de configuración de Apache usando gedit, todo lo que es necesario es escribir el siguiente comando de una sola letra y presionar la tecla ENTER:*

*“a”*

*Tenga en cuenta que la ubicación del archivo de configuración de Apache puede ser diferente en diferentes sistemas. Además, los usuarios normales probablemente solo podrán abrirlo para leerlo, y solo el usuario root podrá editarlo (a menos que se modifique la configuración de permisos).*

***PARA MÁS INFO,*** *<http://www.linfo.org/alias.html>*