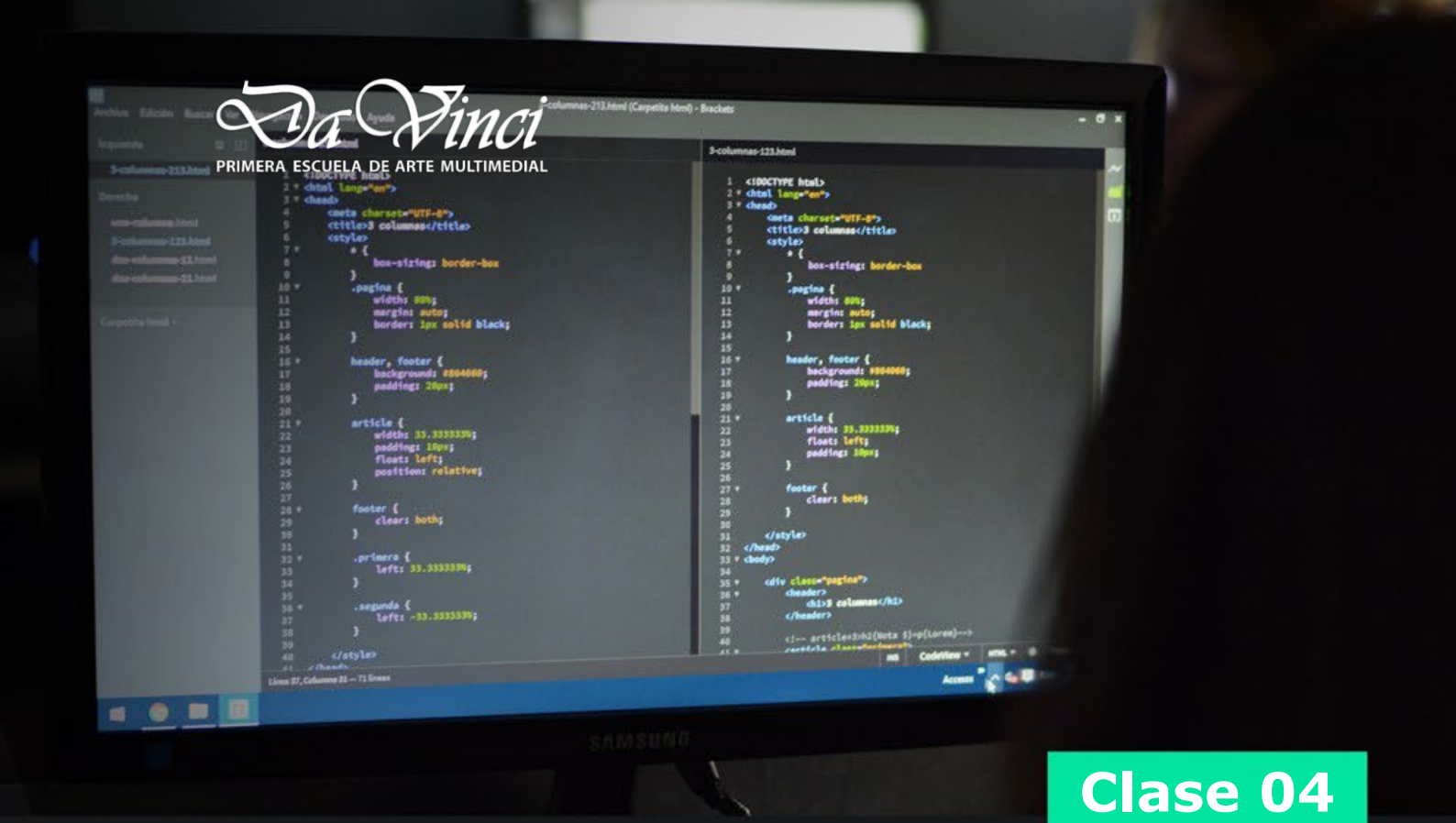

Clase 04

Diseño y Programación Web

Materia:

Sistemas Operativos

Docente contenidista: CARLASSARA, Fabrizio

Revisión: Coordinación

Contenido

Clase 4

¡Te damos la bienvenida a la materia **Sistemas Operativos**!

La semana pasada comenzamos a discutir cómo funciona el sistema de archivos de Linux y vimos algunos comandos básicos para empezar a trabajar.

El objetivo de esta semana es empezar a conocer comandos que nos dejen crear, modificar, copiar, mover y eliminar archivos y directorios. En esta clase vamos a aprender:

En esta clase vamos a ver los siguientes temas:

- Qué tipos de archivos son comunes en Linux.
- La diferencia entre rutas absolutas y relativas.
- Cómo movernos en el sistema de archivos de Linux.
- Cómo crear, mover, copiar, eliminar y editar archivos.
- Cómo usar vim para editar archivos.

Manejo de archivos y carpetas

En primer lugar, tenemos que reconocer que vamos a tener múltiples tipos de archivos en nuestro sistema operativo y cada tipo cumple un propósito particular. A lo largo de nuestra trayectoria usando Linux encontraremos:

- **Archivos de texto:** no tienen un formato en particular y pueden contener información de configuración, scripts de shell o documentos entre otras cosas.
- **Archivos ejecutables:** estos son programas que primero tienen que existir en el almacenamiento de la computadora para luego ser ejecutados y convertirse en procesos.
- **Archivos binarios:** estos tienen información común codificada en binario de la que luego hacen uso muchos programas.
- **Archivos de directorios:** cada vez que creamos una carpeta, un archivo con su nombre es creado en Linux para hacer referencia a ese directorio y ayudarnos a organizar lo que pongamos dentro.
- **Archivos de enlace simbólico:** estos son archivos que hacen referencia a otro, ya sea que contengan la misma información o que simplemente apuntan a otro archivo (como un atajo).
- **Archivos de dispositivos:** estos representan dispositivos del sistema como discos de almacenamiento o puertos seriales. Son los que se encuentran en el directorio /dev.
- **Archivos de tubería (pipes):** permiten direccionar la salida de un programa a la entrada de otro. Sirven para lograr comunicar procesos.
- **Archivos de sockets:** similar a las tuberías pero sirven para comunicar procesos en la red.

Nombres de archivos y extensiones

Los archivos son reconocidos por su nombre, que puede ser de hasta 255 caracteres y, típicamente, usan símbolos como "_" o "-" en su nombre y el "." al comienzo para indicar que son archivos ocultos. Otros caracteres no alfanuméricos se evitan porque tienen significados especiales para el shell y pueden producir errores o resultados inesperados. El uso del espacio para nombres que contengan múltiples palabras no es una práctica recomendada ya que para la shell el espacio es un separador de comandos y argumentos,

por eso recomendamos fuertemente que usen "_" o "-" en lugar de espacios.

En sistemas operativos como Windows, los nombres de archivos terminan con un punto y tres caracteres que identifican qué tipo de archivo es, a lo que denominamos extensión (doc.txt por ejemplo). En Linux, no siempre será el caso que todos los archivos tengan una extensión que delate inmediatamente qué tipo de archivo son pero si mencionamos abajo algunas extensiones comunes que consideramos importantes que podamos al menos tener de vista:

- **.bin:** binarios ejecutables (similar al .exe en Windows).
- **.c:** código fuente de programas en C.
- **.cc/.cpp:** código fuente de programas en C++.
- **.htm/.html:** archivos HTML (Hypertext Markup Language).
- **.txt:** archivos de texto.
- **.tar/.gz/.bz2/.xz/.Z:** archivos comprimidos o que tienen otros archivos dentro.
- **.conf/.cfg:** archivos de configuración que contienen texto.
- **.so:** archivos objeto compartidos por bibliotecas.
- **.o/.ko:** archivos objeto compilados.
- **.jpg/.jpeg/.png/.tiff/.xpm/.gif:** archivos binarios que contienen imágenes.
- **.sh:** archivos con scripts para que ejecute el shell.

Por supuesto que hay muchas extensiones más de archivos, pero éstas son algunas de las que veremos frecuentemente mientras naveguemos entre los distintos directorios del sistema de archivos de Linux.

Rutas absolutas y relativas

Antes de comenzar a meternos con comandos de manejo de archivos, tenemos que entender un concepto más que es la idea de una ruta absoluta y relativa. Ya hemos desarrollado la semana pasada la idea del sistema de archivos de Linux y hemos dicho que la ruta no es más que una ubicación de un archivo dentro del sistema de archivos de Linux.

Nos va a ocurrir cuando ejecutemos comandos, que muchas veces vamos a tener que decirle al shell dónde ejecutarlo, la ubicación del archivo sobre el que queremos ejecutar el comando o incluso dónde dejar el resultado de ejecutar el comando dentro del sistema de

archivos. Esto nos lleva a una nueva discusión en cuanto a cómo escribir esas rutas y las diferencias y ventajas de cada una.

La primera forma es con una ruta absoluta. Estas especifican la ubicación completa de un archivo o directorio desde el directorio raíz del sistema. Comienza desde la raíz del sistema de archivos (/) y sigue la estructura de directorios hasta llegar al archivo o directorio deseado. Por ejemplo podemos decir: **/home/usuario/archivo.txt**: Esta es una ruta absoluta que especifica la ubicación exacta del archivo "archivo.txt" en el directorio "usuario" dentro del directorio "home".

Por otro lado, podemos usar rutas relativas. Estas especifican la ubicación de un archivo o directorio en relación con el directorio actual desde donde se está referenciando. No comienza desde la raíz del sistema de archivos, sino que se basa en la ubicación actual en el sistema de archivos. Por ejemplo: si estamos actualmente en el directorio /home/usuario, la ruta relativa **documentos/notas.txt** se refiere al archivo "notas.txt" dentro del directorio "documentos" en el directorio actual (/home/usuario).

Podemos ver entonces que las rutas absolutas siempre tienen de referencia base al raíz del sistema mientras que las relativas tienen de base el directorio actual. Lo bueno de las rutas absolutas por otro lado, es que un mismo comando va a poder ejecutarse desde cualquier lugar del sistema con ese tipo de rutas, a pesar de que a veces pueda resultar más extensa. En cambio, con las rutas relativas, si bien a veces son más cortas, tenemos que tener cuidado que cambia el significado de acuerdo al lugar donde estamos ejecutando el comando.

Comandos para manejo de archivos

Vamos finalmente a enumerar y ver cómo funcionan algunos comandos comunes para manejar archivos. Si bien hay muchos comandos posibles para usar, vamos a explicar los que más frecuentemente son necesarios.

Moverse entre directorios

Hemos visto anteriormente que cuando nos logueamos a nuestro sistema operativo, el shell termina estando ubicado en el el directorio creado para el usuario que se loguea (/root para el usuario root y /home/usuario para cualquier otro usuario. Esto lo pudimos comprobar con el comando pwd que nos indicaba la ruta donde nos encontramos. Esta ruta se ve resumida en el prompt de Linux con el símbolo "~" que nos indica que estamos en el directorio del usuario.

Sin embargo, vamos a necesitar movernos de ese directorio en algún momento para hacer otras operaciones. Uno de los comandos más elementales es el que nos permite cambiar de directorio y movernos en todo el sistema de archivos de Linux. Este es el comando **cd** que viene de change directory.

La forma de usar este comando es **cd [ruta]** por lo que simplemente necesitamos tener en claro a qué directorio queremos movernos para poder usarlo. Podemos usar tanto rutas absolutas como relativas, lo que nos quede más cómodo, sin embargo, es más habitual el uso de rutas relativas ya que uno se va moviendo muchas veces entre directorios muy próximos.

Quizás al momento de usar este comando, queda claro que uno puede moverse dentro del sistema de archivos yendo cada vez más adentro del mismo, pero puede surgir la pregunta, ¿cómo puedo volver los directorios hacia atrás?

Si buscamos el comando ls en la clase de la semana pasada, vamos a ver que hay una opción -a que nos mostraría archivos ocultos dentro del directorio. Si entonces ejecutamos el comando ls -a vamos a ver que, además de los archivos normales que podemos ver en el directorio, vamos a ver dos archivos llamados **.** y **..** que están ocultos.

Estos dos archivos hacen referencia al directorio actual y al directorio anterior respectivamente.

Si queremos entonces movernos al directorio anterior al que estamos, podemos simplemente escribir cd .. y volvemos un directorio atrás.

Otra cosa típica que podemos necesitar es volver al directorio del usuario en el que estamos. Si bien podemos usar la ruta absoluta al directorio, es más sencillo aprovechar el carácter que mencionamos más arriba y escribir cd ~.

Mostrar contenido de archivos de texto

Los archivos de texto son los más comunes que nos van a interesar mostrar permanentemente. Muchos de estos van a tener scripts, programas o configuración que tendremos que verificar. Hay varios comandos que pueden servirnos para ese propósito, a continuación vamos a mencionar algunos que incluyen:

- **cat:** es el comando más común para ver contenido de texto y su nombre viene de concatenación. La forma de usarlo es **cat [ruta a archivo]**. Por ejemplo, si quisiéramos ver el contenido de un doc.txt ubicado en el directorio del usuario root, podríamos decir: cat /root/doc.txt o cat ~/doc.txt.
- **tac:** funciona exactamente igual que el cat, pero muestra el texto de la última línea a la primera. Esto es especialmente útil para archivos que permanentemente tienen texto agregado al final del mismo como los log de algunos servicios de Linux. Esto implica que suelen ser archivos muy extensos y donde lo más reciente se encuentra al final.
- **head:** nos muestra una cantidad determinada de líneas de texto del archivo, por defecto, solo muestra las primeras 10 líneas del archivo. El uso del comando es: **head -[cantidad opcional] [ruta a archivo]** por lo que es similar a cat pero además, nos deja escribir la cantidad de líneas que queremos mostrar. Por ejemplo, si quisiéramos ver las primeras 5 líneas del archivo doc.txt escribimos head -5 doc.txt.
- **tail:** así como existe el tac como contraparte del cat, existe el **tail** como contraparte del head. Este comando funciona de la misma forma que el comando head y con la misma posibilidad de elegir la cantidad de líneas pero ahora comenzando por el final del archivo en vez del final.

- **more:** funciona como cat pero es especialmente útil para archivos de texto demasiado extensos. Al ejecutarlo, nos muestra un bloque de texto y nos permite seguir corriendo comandos. Por ejemplo, con la barra espaciadora, nos muestra la siguiente página de texto y con el enter nos muestra sólo la siguiente línea. Esto es útil si queremos navegar en un texto extenso buscando algo especial. Una vez que queremos terminar el comando more, solo hace falta apretar la tecla q. El uso de este comando es **more [ruta a archivo]**.
- **less:** funciona exactamente igual que el comando more y nos sirve para lo mismo, pero además, nos permite usar las flechas para poder subir y bajar en el texto.

Todos estos comandos, especialmente los últimos dos tienen más opciones y funcionalidades pero no es el objetivo tratarlos a fondo en esta clase.

Mostrar contenido de archivos binarios

Si bien los comandos anteriores son buenos para mostrar contenido de archivos de texto, a veces necesitamos ver un poco el contenido de algún archivo binario. Se nos podría ocurrir usar los mismos comandos para ver el contenido pero en seguida vamos a frustrarnos al ver una cantidad de caracteres extraños, esto es debido a que los archivos binarios tienen una cantidad de caracteres que no suelen ser imprimibles.

Por eso, hay algunos comandos especiales para tratar con archivos binarios y su contenido. Vamos a mencionar dos en particular:

- **strings:** es un comando que sólo muestra los caracteres imprimibles dentro de un archivo binario. Esto puede ayudarnos a rescatar cualquier texto que nos da una idea de qué hace ese archivo binario. Por ejemplo, podemos escribir strings /bin/echo y veremos sólo el contenido legible del archivo binario que puede darnos alguna noción de en qué consiste.
- **hexdump:** este comando tiene un uso mucho más específico pero vale la pena mencionarlo. Su objetivo es mostrarnos el contenido del archivo binario completo en forma hexadecimal y, si brindamos la opción -C, también en carácter ASCII. Esto tiene sus usos para aquellos que quieran investigar a fondo anomalías en archivos binarios.

Crear, mover, copiar y eliminar archivos

En esta sección vamos a tratar muchos comandos que en general sirven para manipular archivos. Vamos a intentar separarlos lo mejor posible para que tengan coherencia entre sí y sea más llevadero entenderlos.

Crear archivos

Hasta ahora hemos visto formas de visualizar archivos pero, ¿qué pasa si queremos crear algún archivo nosotros? Como siempre, existen varias formas para eso, ahora les presentamos algunas:

- **touch:** este comando es el encargado de crear archivos. Su uso es sencillo, solamente tenemos que escribir **touch [nombre del archivo]** y va a aparecer en donde lo indiquemos. En el argumento que corresponde al nombre del archivo, podemos especificar la ruta absoluta o relativa donde queremos que termine apareciendo el archivo. Por ejemplo, touch /root/test.txt va a crear el archivo test.txt dentro del directorio del usuario root. Si el archivo ya existía, no se sobreescribe. También podemos crear múltiples archivos con el mismo comando solamente escribiendo touch [archivo1] [archivo2] y así sucesivamente.
- **echo:** en realidad el objetivo de este comando es imprimir un mensaje por el shell. El uso es **echo "[texto para imprimir]"** y veremos que va a aparecer ese mismo mensaje en el shell. Sin embargo, podemos aprovecharlo para que, en vez de mandar ese mensaje al shell, lo mande a un archivo nuevo. La forma de usarlo sería **echo "[texto]" > [nombre del archivo]** y el resultado será que aparecerá el texto que indicamos en un archivo que va a ser creado con el mismo comando. Por ejemplo, echo "Hola mundo!" > /root/test.txt va a hacer aparecer el mensaje "Hola mundo!" en un nuevo archivo llamado test.txt en el directorio del usuario root. El símbolo ">" tiene algunas implicaciones que discutiremos en clases posteriores.
- **cat:** otra vez, si bien el objetivo de este comando no es el de crear archivos, podemos hacer algunas modificaciones para que nos ayude a crear uno. Si escribimos **cat > [nombre de archivo]** el shell nos dejará escribir texto libremente. Una vez que hayamos terminado de escribir, apretamos Ctrl + D y el archivo va a ser creado con el texto que indicamos.
- **mkdir:** en el caso particular de directorios, usamos el comando **mkdir [nombre]** para crear un directorio vacío en la ruta que

especifiquemos. Es el análogo de touch para directorios. Por ejemplo, si queremos crear un directorio llamado dir en el directorio del usuario root podemos escribir mkdir /root/dir.

Existen algunas formas más de crear archivos, pero vamos a dejarlas para la sección siguiente ya que merecen bastante más atención.

Mover archivos

Veamos ahora el caso de mover archivos. Este es el comando **mv**. Su uso es un poquito más delicado que los comandos que venimos viendo hasta ahora. La sintaxis para usarlo es **mv [opciones] [origen] [destino]** donde origen y destino son la ruta del archivo que queremos mover y la ruta a dónde queremos moverlo respectivamente.

En cuanto a opciones, vamos a destacar la opción -**f** para obviar cualquier tipo de pregunta y forzar la operación. Tenemos que tener precaución a la hora de usar este comando con esa opción ya que podemos sobreescribir archivos sin darnos cuenta y perder información.

Si queremos mover directorios enteros, es posible con este comando, ya que los directorios no dejan de ser archivos para Linux. Un ejemplo de esto podría ser: mv dir /root/dir que movería el directorio dir de la ruta donde estamos parados al directorio del usuario root. Recordemos que el contenido entero del directorio va a ser movido.

Por otro lado, un caso especial de este comando tiene su uso para renombrar archivos o directorios. Podemos escribir mv file file_2 que resulta en mover el archivo file al mismo directorio pero cambiándole el nombre a file_2.

Este comando también nos permite mover muchos archivos en un solo comando. Lo único que tenemos que tener en cuenta es que el último argumento del comando es el destino de todos los archivos que especifiquemos que queremos mover.

Copiar archivos

Ahora que entendemos cómo mover archivos, podemos discutir cómo copiarlos. La lógica es prácticamente igual que para el comando que vimos anteriormente pero ahora nuestro comando es **cp**. La sintaxis

para poder usar este comando es **cp [opciones] [origen] [destino]**.

En cuanto a origen y destino, lo mismo que estuvimos discutiendo para el comando mv es válido aquí. Lo mismo es válido para el caso de las opciones que discutimos pero, vamos a agregar una opción más que es **-r** o **-R** y que implica una copia recursiva. Esta opción es necesaria si queremos copiar directorios con todo su contenido ya que no funciona como el comando mv en ese sentido. Si queremos copiar un directorio entonces tenemos que escribir cp -r dir dir_2 que va a copiar dir con todo su contenido como dir_2.

En cuanto a copiar múltiples archivos, lo mismo que se dijo sobre el comando mv es válido para este comando.

Eliminar archivos

Por último en esta tanda de comandos, nos toca ver el caso de eliminar archivos, algo bastante común de hacer si nos hemos equivocado mientras estábamos trabajando.

El comando que vamos a analizar es el **rm**. Su sintaxis es muy sencilla, solamente tenemos que escribir **rm [opciones] [archivo]**. El argumento archivo es el archivo o ruta al archivo que queremos eliminar y, como con los comandos anteriores, podemos indicar varios archivos para eliminar en el mismo comando.

En cuanto a las opciones tenemos **-f** como en los casos de mv y cp y funciona de la misma manera, forzando la eliminación sin preguntar respectivamente.

Si nos interesa eliminar directorios, usamos la opción **-r** que, como en el comando cp, significa que se eliminarán los archivos recursivamente. Por ejemplo, si queremos eliminar el directorio dir con todo su contenido, escribimos en el shell: rm -r dir. Si no queremos que nos pregunte por una confirmación con cada archivo, podemos usar como opción -rf que eliminará todo el directorio sin consultar.

Como alternativa, si tenemos un directorio que está vacío, podemos usar el comando **rmdir**. La forma de usarlo es **rmdir [directorio]** pero sólo va a funcionar si el directorio que especificamos no tiene nada dentro.

Editar archivos

Hasta ahora, venimos tratando como crear, mover, copiar y borrar archivos, pero todavía nos falta algo esencial: ¡aprender a editar archivos!

Linux tiene nativamente editores de texto que nos van a permitir editar archivos a gusto. El que vamos a mencionar particularmente en esta sección es **vim** que, si bien es un poco complejo al comienzo, viene en cualquier distribución de Linux que podamos encontrar.

Para abrir un archivo con vim, solo hace falta escribir **vi [archivo]** y veremos abrirse un editor de texto con el contenido del archivo. Si el archivo no existiera, vamos a ver el editor vacío y podremos crearlo cuando lo guardemos.

El editor vim es un editor bimodal ya que tiene dos modos de funcionamiento: el de modo de comandos y el modo de inserción. El editor se abre por defecto en el modo de comandos, donde vamos a tener que presionar algunas teclas para cambiar de modo, guardar el archivo, descartar cambios y salir entre otras cosas.

Para entrar en el modo de inserción que es donde vamos a escribir texto y modificar el archivo, apretamos la tecla **i** en el modo de comandos y vamos a ver que abajo de todo vamos a ver el texto -- INSERT -- que confirma que el editor espera que escribamos.

Cuando estemos satisfechos con el texto que escribimos en el archivo, podemos salir del modo de inserción con la tecla de escape y volvemos al modo de comandos.

Abajo dejamos un resumen de los comandos más comunes que podemos usar en este modo.

Comando	Descripción
i	Entra en modo de inserción poniendo el cursor antes del carácter donde estamos parados.
a	Entra en modo de inserción poniendo el cursor después del carácter donde estamos parados.
o	Entra en modo de inserción poniendo el cursor en una nueva línea debajo de donde estamos parados.
I	Entra en modo de inserción poniendo el cursor al principio de la línea donde estamos parados.
A	Entra en modo de inserción poniendo el cursor al final de la línea donde estamos parados.
O	Entra en modo de inserción poniendo el cursor en una nueva línea arriba de donde estamos parados.
Esc	Sale del modo de inserción y vuelve al modo de comandos.
:q	Sale de vim si no se hicieron cambios.
:q!	Sale de vim sin guardar los cambios.
:wq	Guarda los cambios y sale de vim.
:w file	Guarda el documento escrito en un archivo llamado file.
:r file	Escribe el contenido del documento file en el documento abierto por vim debajo de la línea donde nos encontramos.

Vim puede ser un editor un poco complicado al comienzo, pero sólo requiere algo de práctica para poder al menos editar algunos archivos, una vez que la tengamos, puede ser un editor de texto muy poderoso.

Hemos llegado así al final de esta clase

Esperamos que puedan poner en práctica estos comandos que estuvimos discutiendo. Recuerden que siempre pueden consultar más información de los comandos con el comando man. Esta semana hemos visto:

- Qué tipos de archivos son comunes en Linux.
- La diferencia entre rutas absolutas y relativas.
- Cómo movernos en el sistema de archivos de Linux.
- Cómo crear, mover, copiar, eliminar y editar archivos.
- Cómo usar vim para editar archivos.

Te esperamos en la **clase en vivo** de esta semana.
No olvides realizar el **desafío semanal**.

¡Hasta la próxima clase!

Bibliografía

Eckert, J. W. (2020). Linux+ and LPIC-1: Guide to Linux Certification. Cengage.

Para ampliar la información

It's FOSS. (s.f.). Ruta absoluta vs. relativa en Linux: ¿Cuál es la diferencia?. Recuperado de https://itsfoss.com/es/ruta-absoluta-relativa-linux/

Gitbook. (s.f). Título del capítulo: cd - Comandos Linux. En Comandos Linux. Recuperado de https://didweb.gitbooks.io/comandos-linux/content/chapter1/directorios/cd.html

Gitbook. (s.f). Título del capítulo: mkdir - Comandos Linux. En Comandos Linux. Recuperado de https://didweb.gitbooks.io/comandos-linux/content/chapter1/directorios/mkdir.html

Gitbook. (s.f). Título del capítulo: rm y rmdir - Comandos Linux. En Comandos Linux. Recuperado de https://didweb.gitbooks.io/comandos-linux/content/chapter1/descargas-de-archivos/rm.html

Gitbook. (s.f). Título del capítulo: cp - Comandos Linux. En Comandos Linux. Recuperado de https://didweb.gitbooks.io/comandos-linux/content/chapter1/descargas-de-archivos/cp.html

Gitbook. (s.f). Título del capítulo: mv - Comandos Linux. En Comandos Linux. Recuperado de https://didweb.gitbooks.io/comandos-linux/content/chapter1/descargas-de-archivos/mv.html