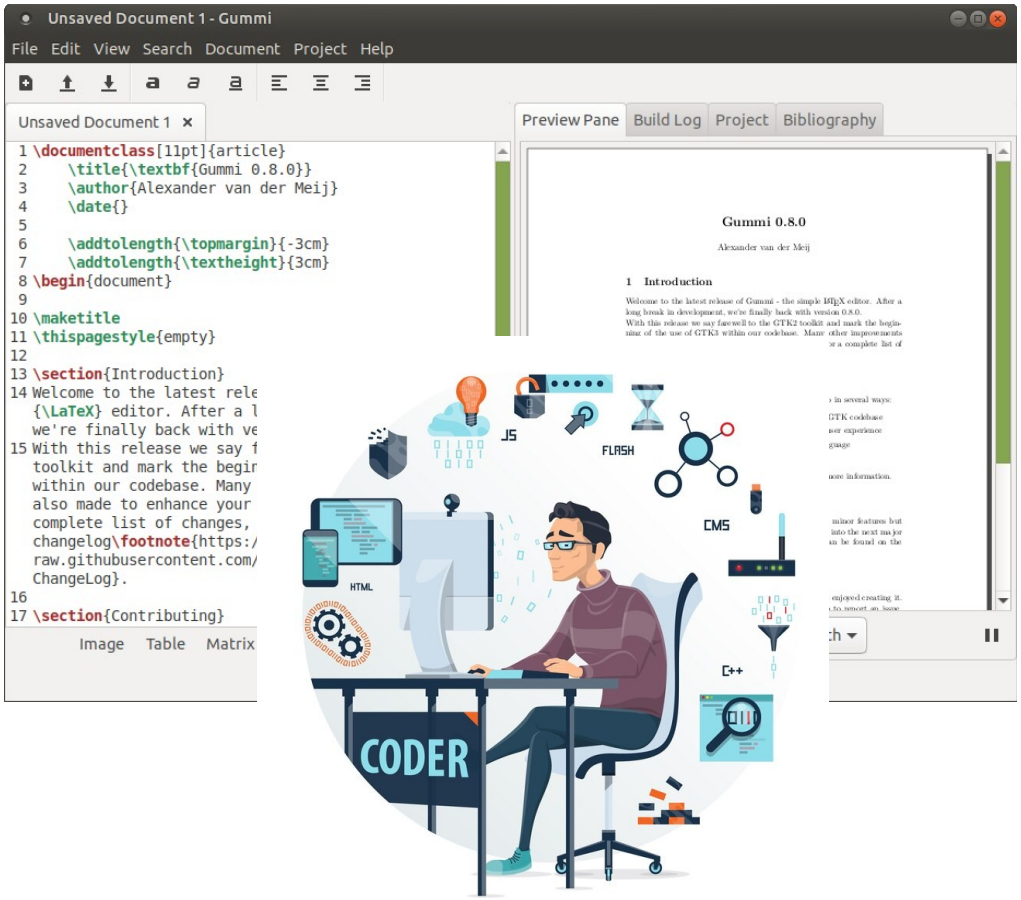
Herramientas Para la Edición de Textos Científicos en Linux



Antonio Carrillo Ledesma

Herramientas Para Edición de Textos Científicos en Linux

Antonio Carrillo Ledesma
Facultad de Ciencias, UNAM
http://academicos.fciencias.unam.mx/antoniocarrillo

La última versión de este trabajo se puede descargar de la página:
https://sites.google.com/ciencias.unam.mx/acl/en-desarrollo
http://132.248.181.216/acl/EnDesarrollo.html

2025, Versión 1.0α[1]

Índice

1 Introducción

Desde la aparición de la computadora se ha requerido un creciente número de especialistas en computación[1], así como personal que la programe. Y ante los retos que el vertiginoso y dinámico cambio informático que enfrenta el mundo globalizado y ante las exigencias de la sociedad de la información es aún más demandante la necesidad de especialistas en computación y en particular a su aplicación a la edición de textos científicos.

El papel actual de las tecnologías de la información y las comunicaciones (TIC) en la sociedad es muy importante porque ofrecen muchos servicios como: correo electrónico, búsqueda de información, banca en línea, descarga de música y vídeo, comercio electrónico, etc. Por esta razón las TIC han incursionado fácilmente en diversos ámbitos de la vida, entre ellos, el de la educación.

1.1 Edición de Textos Científicos

Si eres estudiante, tesista, profesor o investigador y necesitas escribir documentos científicos que contengan formulas matemáticas, requieran de un formato estandarizado, con índice, numeración de tablas, figuras, ecuaciones, por supuesto bibliografía, LATEX (véase [19]) es la solución.

LaTeX es un sistema de composición de textos, concebido para la creación de documentos escritos que presenten una alta calidad tipográfica. Por sus características y posibilidades, se utiliza especialmente y en gran medida en la generación de artículos y libros científicos que incluyen expresiones matemáticas, entre otros elementos. Es además un software libre distribuido bajo licencia GPL (véase [12]).

LaTeX está formado por un gran conjunto de macros de TeX, escrito por Leslie Lamport en 1984, con la intención de facilitar el uso del lenguaje de composición tipográfica TeX, creado por Donald Knuth. Es muy empleado para la composición de artículos académicos, tesis y libros técnicos, dado que la calidad tipográfica de los documentos realizados en LaTeX, se considera adecuada a las necesidades de una editorial científica de primera línea, muchas de las cuales ya lo emplean.

[1] El cuerpo de conocimiento de las ciencias de la computación es frecuentemente descrito como el estudio sistemático de los procesos algorítmicos que describen y transforman información: su teoría, análisis, diseño, eficiencia, implementación, algoritmos sistematizados y aplicación.

Resumidamente, las ventajas de LaTeX:

- Calidad Tipográfica: Los documentos generados tienen una apariencia profesional, ideal para textos académicos.
- Separación de Contenido y Formato: Permite centrarse en el contenido sin preocuparse por el diseño.
- Manejo de Referencias: Herramientas como BibTeX y biblatex facilitan la gestión de bibliografías.
- Ideal para Documentos Extensos: Maneja fácilmente libros, tesis y artículos largos.

Además de usar LaTeX para escribir documentos científicos, se requieren herramientas auxiliares que faciliten el editar, compilar, generar archivos gráficos y PDFs. En este trabajo mostraremos las diferentes herramientas en GNU/Linux que nos permiten trabajar en cada una de las diferentes partes que componen a un texto científico usando el ambiente de trabajo gráfico y de texto de Linux (véase [4]).

1.2 La Línea de Comandos

Todo creador comienza su viaje con un conjunto básico de herramientas de buena calidad. Un carpintero puede necesitar reglas, calibres, un par de sierras, unas buenas garlopas, escoplos finos, taladros, mazos y tornillos de banco. Estas herramientas se elegirán con mimo, estarán hechas paras durar, realizarán trabajos específicos sin mucho solapamiento con otras herramientas y, quizá lo más importante, se sentirá que las manos del carpintero en ciernes son el lugar al que pertenecen.

Comienza entonces el proceso de aprendizaje y adaptación. Cada herramienta tiene su propia personalidad y sus peculiaridades, y necesita su propio manejo especial. Cada una debe afilarse de una manera única o sostenerse de un modo concreto. Con el tiempo, cada una se desgastará en función de su uso, hasta que la empuñadura parezca un molde de las manos del carpintero y la superficie de corte se alinee a la perfección con el ángulo en el que sostiene la herramienta. En este punto, las herramientas se convierten en canales desde el cerebro del carpintero hasta el producto acabado; se han convertido en extensiones de sus manos. Con el tiempo,

el carpintero añadirá herramientas nuevas, como engalletadoras, sierras de inglete guiadas por láser, plantillas de cola de pato... todas ellas formas maravillosas de tecnología. Pero puede estar seguro de que el carpintero será más feliz con esas herramientas originales en la mano, sintiendo cómo canta la garlopa al deslizarse por la madera.

Las herramientas amplifican el talento. Cuanto mejores sean sus herramientas y mejor sepa cómo usarlas, más productivo podrá ser. Empiece con un conjunto básico de herramientas aplicables a nivel general. A medida que adquiera experiencia y vaya encontrándose requisitos especiales, añada más al conjunto básico. Al igual que el carpintero, cuente con añadir herramientas a su kit con regularidad. Esté siempre atento a la aparición de formas mejores de hacer las cosas. Si se encuentra en una situación en la que siente que sus herramientas actuales no sirven, tome nota de buscar algo diferente o más potente que le hubiese ayudado.

Deje que la necesidad impulse sus adquisiciones. Muchos creadores de textos científicos nuevos cometen el error de adoptar una sola herramienta potente, como un entorno de desarrollo integrado (IDE, por sus siglas en inglés) y nunca abandonan su acogedora interfaz. Eso es un verdadero error. Tiene que sentirse cómodo más allá de los límites impuestos por un IDE. La única manera de hacerlo es mantener las herramientas básicas afiladas y listas para usar.

En este texto, hablaremos acerca de la investigación de su propia caja de herramientas básica. Como ocurre con cualquier buena charla sobre herramientas, empezaremos echando un vistazo a nuestra materia prima, aquello a lo que vamos a dar forma. Para garantizar que nunca se pierde nada de nuestro valioso trabajo, deberíamos utilizar un sistema de "Control de versiones", incluso para cosas personales como recetas o notas. Y, puesto que Murphy era en realidad un optimista, al fin y al cabo, no se puede ser un gran creadores de textos científicos a menos que desarrolle una gran habilidad en la "Depuración". Necesitará algo de pegamento para unir toda la magia. Por último, vale más tinta pálida que memoria brillante. Mantenga un registro de sus pensamientos y su historial, en un cuadernos de bitácora.

Además, todo carpintero necesita una buena mesa de trabajo, sólida y fiable, para sujetar las piezas en las que trabaja a una altura conveniente mientras les da forma. La mesa de trabajo se convierte en el centro del taller, y el carpintero vuelve a ella una y otra vez mientras la pieza toma forma.

Para un creadores de textos científicos que manipula archivos de texto,

imagenes y datos, esa mesa de trabajo es el intérprete de comandos. Desde el Prompt de la línea de comandos, puede invocar su repertorio completo de herramientas, usando Pipes para combinarlos de maneras que sus desarrolladores originales jamás soñaron. Desde el intérprete de comandos, puede iniciar aplicaciones, depuradores, navegadores, editores y servicios. Puede buscar archivos, consultar el estado del sistema y filtrar salidas. Y, al programar el intérprete de comandos, puede crear comandos en macros complejas para actividades que realiza a menudo.

Para los creadores de textos científicos que han crecido con interfaces GUI y entornos de desarrollo integrados, esto podría parecer una posición extrema. Al fin y al cabo, ¿no se puede hacer todo igual de bien apuntando y haciendo clic?

La respuesta sencilla es "no". Las interfaces GUI son maravillosas y pueden ser más rápidas y convenientes para algunas operaciones sencillas. Mover archivos, leer y escribir correos electrónicos y crear y desarrollar un proyecto son cosas que podría interesarle hacer en un entorno gráfico, pero, si hace todo su trabajo usando GUI, está perdiéndose todas las capacidades de su entorno. No podrá automatizar tareas comunes ni aprovechar el máximo potencial de las herramientas a su disposición. Y no podrá combinar sus herramientas para crear macros personalizadas. Un beneficio de las GUI es *WYSIWYG* (what you see is what you get, lo que ves es lo que obtienes). La desventaja es *WYSIAYG* (what you see is all you get, lo que ves es todo lo que obtienes).

Los entornos GUI suelen estar limitados a las capacidades que sus diseñadores planearon. Si necesita ir más allá del modelo proporcionado por el diseñador, por lo general no tendrá mucha suerte, y la mayoría de las veces sí necesita ir más allá del modelo. Los programadores pragmáticos no solo escribimos código, desarrollamos modelos de objetos, escribimos documentación o automatizamos el proceso de construcción; hacemos todas esas cosas. El alcance de una herramienta cualquiera suele verse limitado a las tareas que se espera que realice esa herramienta. Por ejemplo, supongamos que necesita integrar un preprocesador de código (para revisar ortografía o algo de ese estilo) en su IDE. A menos que el diseñador del IDE proporcionase de forma explícita enganches para esa capacidad, no podrá hacerlo.

Familiarícese con el intérprete de comandos y verá cómo se dispara su productividad. Si no ha dedicado mucho tiempo a explorar las capacidades del intérprete de comandos en los sistemas que utiliza, esto podría parecer abrumador. No obstante, invierta algo de tiempo en familiarizarse con su

intérprete de comandos y pronto todo empezará a tener sentido. Juguetee con el intérprete de comandos y le sorprenderá cuánto le ayuda a ser más productivo.

Dedique tiempo a aprender a utilizar estas herramientas y, en algún momento, se sorprenderá al descubrir sus dedos moviéndose sobre el teclado, manipulando texto sin un pensamiento consciente. Las herramientas se habrán convertido en una extensión de sus manos.

La Línea de Comandos El sistema Linux básico es un entorno estándar para aplicaciones y programación de los usuarios -generalmente en modo de consola no gráfico-, pero no obliga a adoptar mecanismos estándar para controlar las funcionalidades disponibles como un todo (como Windows o Mac). A medida que Linux ha madurado, se ha hecho necesaria otra capa de funcionalidad encima del sistema Linux. Una distribución de GNU/Linux incluye todos los componentes estándar del sistema Linux -modo consola y gráfico-, más un conjunto de herramientas administrativas que simplifican la instalación inicial y desinstalación de paquetes del sistema.

GNU/Linux puede funcionar tanto en entorno gráfico como en modo consola (línea de comandos o *Shell*). La consola es común en distribuciones para servidores, mientras que la interfaz gráfica está orientada al usuario final del hogar como el empresarial. Así mismo, también existen los entornos de escritorio (GNOME, KDE, LXQt, LXDE, Xfce, Unity, MATE, Cinnamon, Pantheon, Deepin, Budgie, PIXEL, Enlightenment, Trinity, Moksha, Ukui, etc.), que son un conjunto de programas conformado por ventanas, íconos y muchas aplicaciones que facilitan el uso de la computadora.

La mayoría de los usuarios de ordenadores de hoy sólo están familiarizados con la interfaz gráfica de usuario o GUI (del inglés Graphical User Interface) y en los últimos años se han puesto muy de moda las WUI (del inglés Web User Interface) muy similar a las GUI, pero a las que se accede a través de un servidor Web.

Los vendedores y los expertos les han enseñado a los usuarios noveles que la interfaz de línea de comandos o CLI (del inglés Command Line Interface) y su complemento las TUI (del inglés Text User Interface) que juntas son interfaces de texto poco amigables, con menús y ayuda por pantalla es una cosa espantosa del pasado. Es una pena, porque una buena interfaz de línea de comandos es una maravillosa y práctica forma de comunicarse con el ordenador, muy parecida a lo que el lenguaje escrito es para los seres humanos.

Se ha dicho que "las interfaces gráficas de usuario hacen fáciles las tareas fáciles, mientras que las interfaces de línea de comandos hacen posibles las tareas difíciles" y eso es muy cierto aún hoy.

Cabe mencionar que Unix/Linux son los únicos sistemas operativos que quedan cuya interfaz gráfica (un montón de código llamado X Windows System[2]) está separado del sistema operativo[3], es decir, se puede ejecutar Unix/Linux en puro modo de línea de comandos si quieres, sin ventanas, iconos, ratones, etc., y seguir siendo Unix/Linux y capaz de hacer todo lo que se supone que hace Unix/Linux. Pero los demás sistemas operativos tienen sus GUI enmarañadas con las funciones del sistema operativo en tal grado que han de ejecutarse en modo GUI o no se ejecutarán, en estos casos no es posible pensar en las GUI como algo distinto del sistema operativo; ahora forman parte inalienable de los sistemas operativos a los que pertenecen -y son, con mucho, la parte mayor, más cara y difícil de crear-.

Dado que Linux fue desarrollado desde la familia de sistemas operativos Unix, comparte la misma rica herencia de herramientas de línea de comandos que Unix. Unix saltó a la fama a principios de los años ochenta -aunque fue desarrollado una década antes-, antes de que se extendiera la adopción de las interfaces gráficas de usuario, y por eso, se desarrolló una amplia interfaz de línea de comandos en su lugar. De hecho, una de las razones más potentes para que los primeros que utilizaron Linux lo eligieron sobre, digamos, Windows NT, era la poderosa interfaz de línea de comandos que hacía las "tareas difíciles posibles" y eso es lo que trataremos de mostrar a lo largo de este trabajo.

¿Por qué Aprender a Programar en la Línea de Comandos? hay muchas razones por las que deberías aprender sobre la línea de comandos de Linux/Unix. Algunas de estas son:

- Más control sobre tu máquina: tienes mucho poder y control con la línea de comando. Puede ejecutar comandos para cambiar permisos,

[2]Se trata de un potente sistema de ventanas basado en una arquitectura cliente/servidor. Una de sus ventajas de está arquitectura es que puede ser implementada tanto de manera distribuida (es decir, aplicaciones y servidor gráfico ejecutándose en máquinas diferentes) como local (todo el subsistema gráfico ejecutándose en el mismo equipo de cómputo).

[3]Existen distribuciones de GNU/Linux completas contenidas en solo unas decenas de megabytes. En ellas se usan las últimas versiones del Kernel de Linux y cuentan con los paquetes necesarios para hacer tareas especializadas.

ver archivos ocultos, interactuar con bases de datos, iniciar servidores y más.

- Es más rápido: puede completar tareas mucho más rápidamente con los comandos básicos de su caja de herramientas que con una interfaz gráfica de usuario (GUI). Solo tenga en cuenta que puede ser más lento mientras aprende la CLI.
- Automatice muchas tareas: puede acelerar su trabajo utilizando un solo comando para crear 10.000 archivos, cada uno con un nombre único. Con una GUI, este proceso es laborioso.
- Disponible en todas partes: las instrucciones que emita se ejecutarán automáticamente de manera similar en computadoras Linux y Mac. Y con algunos ajustes, también funcionarán en Windows.
- Requisito básico: necesitas utilizar la línea de comandos si desea mejorar sus conocimientos en cualquier campo tecnológico relacionado con la codificación, incluido el desarrollo, el análisis de datos, la ingeniería de desarrollo, la administración de sistemas, la seguridad, la ingeniería de aprendizaje automático y otros.

¿Qué es una Shell? un Shell es una interfaz de computadora para un sistema operativo. El Shell expone los servicios del sistema operativo a los usuarios u otros programas. El Shell toma sus comandos y se los da al sistema operativo para que pueda ejecutarlos.

Se llama Shell porque es la capa exterior que rodea el sistema operativo, ¡como la concha alrededor de una ostra!

¿Qué es la Terminal? una terminal es un programa que ejecuta un Shell. Aquí es donde ejecutamos la mayoría de nuestros comandos que le indican al sistema operativo qué hacer.

La terminal se instala de las siguientes maneras en diferentes sistemas operativos:

- Usuarios de alguna distribución de Linux: el Shell Bash está instalado de forma predeterminada
- Usuarios de Mac: la terminal se instala de forma predeterminada y puede ejecutar comandos similares de Linux en Unix

- Usuarios de Windows: descargue el Subsistema de Windows para Linux (WSL) o use git bash y ejecute todos los comandos de Linux desde allí.

Ventajas y Desventajas de Usar la Línea de Comandos Algunas de las ventajas y desventajas al usar la línea de comandos o terminal son las que citaremos a continuación:

Facilidad de Uso Aprender a usar la línea de comandos tiene una curva de aprendizaje más alta que usar interfaz gráfica. Usar la terminal requiere:

- Tener la capacidad y paciencia de memorizar comandos.
- Tener curiosidad y ganas de aprender.
- Ser paciente para leer la documentación de las páginas man.
- A ser posible tener nociones básicas de programación.
- Tener la suficiente paciencia para irse familiarizando con el uso de terminal.

En contraposición las aplicaciones con interfaz gráfica tienen una curva de aprendizaje mucho menor que las que se ejecutan en la línea de comandos. Esto es así por los siguientes motivos:

- Proporcionan un entorno visual agradable. Si una cosa es bonita por fuera atrae a los usuarios.
- Acostumbran a ser intuitivas. Las interfaces gráficas tienen iconos, colores e imágenes que normalmente hacen que la curva de aprendizaje de los usuarios sea mucho menor. En este caso es importante recordar que la mayoría de seres humanos tienden a memorizar mejor los conceptos gráficos que los que están en formato texto.

Por las razones citadas en este apartado los usuarios noveles prefieren usar interfaces gráficas antes que la línea de comandos. No obstante es importante remarcar lo siguiente. Es verdad que tardaremos más tiempo en aprender los comandos para usar la línea de comandos, pero una vez aprendidos podremos abrir una terminal en cualquier ordenador y trabajar de forma habitual. En cambio las GUI acostumbran a variar mucho entre distintas versiones de programas.

Automatización de Tareas mediante Scripts Por norma general las interfaces gráficas no permiten la automatización de tareas repetitivas. Un claro ejemplo de lo que acabo de comentar es el redimensionado de fotografías. Si usaramos un programa con interfaz gráfica y quisiéramos redimensionar 1000 fotografías de un directorio a un 50% es altamente probable que tengamos que repetir 1000 veces la misma operación.

En cambio si usamos la línea de comandos lo podríamos hacer en cuestión de segundos ejecutando un comando similar al siguiente:

```
$ for file in *.png; do convert $file -resize 50% resized-$file;
done
```

Notemos que, cuando se ejecutan comandos en GNU/Linux, ya sea uno a la vez en la línea de comandos o desde un Script de Bash, los comandos se ejecutan en secuencia (usando solo un core de nuestro procesador). El primer comando se ejecuta, seguido por el segundo, seguido por el tercero. Es cierto, el tiempo entre los comandos es tan minúsculo, que el ojo humano no se daría cuenta. Pero para algunos casos, puede que no sea el medio más eficiente para ejecutar comandos.

Con GNU Parallel[4] podemos hacer uso de todos los cores de nuestro procesador, por ejemplo, para cambiar el formato de los .jpg a .png podríamos hacer lo siguiente:

```
$ find .  -name "*.jpg" | parallel -I% –max-args 1 convert %
%.png
```

o

```
$ ls -l *.jpg | parallel convert '{}' '{.}.png'
```

Basándome en los últimos ejemplos recomendaría el uso de la línea de comandos a la totalidad de usuarios que tengan en mente la productividad personal. Tengamos en cuenta que toda tarea/operación realizada en la línea de comandos se podrá automatizar mediante Scripts. Cuantas más operaciones consigamos automatizar más tiempo ahorraremos. Por lo tanto una de las ventajas que proporciona la línea de comandos es la automatización de tareas.

[4]GNU Parallel se puede instalar en casi cualquier distribución de GNU/Linux. Dado que GNU Parallel se encuentra en el repositorio estándar, se instala mediante:

```
# apt install parallel
```

Consumo de Recursos y Rendimiento Las aplicaciones que se ejecutan en la línea de comandos consumen menos recursos y tienen un rendimiento superior. Esto es así porque las aplicaciones ejecutadas en la terminal:

- No requieren de un Driver gráfico avanzado.
- No usan iconos ni imágenes.
- No tienen que cargar ningún tipo de fuente.
- Tienen un consumo de memoria menor.
- Tienen un consumo de CPU menor.
- Requieren una capacidad de almacenamiento en disco duro menor. Las aplicaciones que se ejecutan en la terminal prácticamente no ocupan espacio en nuestro disco duro.
- Interactúan de forma mucho más directa con el sistema operativo.
- etc.

Otra de las ventajas de la línea de comandos será que el consumo de recursos será mucho menor que si usamos una interfaz gráfica. Consecuentemente el rendimiento y la velocidad con que se ejecutarán las tareas/operaciones será mejor en la terminal.

Velocidad de Ejecución y Uso de los Programas Con la línea de comandos pueden satisfacer el 100% de sus necesidades mediante el uso del teclado. En cambio con las interfaces gráficas tendrán que ir moviendo el ratón e ir seleccionado en las opciones pertinentes. Por lo tanto ejecutar acciones en la terminal será más rápido que con una interfaz gráfica. Esto es así porque en la mayoría de ocasiones es mucho más rápido usar el teclado que ir moviendo el ratón e ir clicando encima de los iconos y opciones pertinentes. No obstante para ejecutar tareas de forma rápida en la terminal tendremos que ser capaces de:

- Recordar comandos.
- Recordar los atajos de teclado para poder interactuar con las aplicaciones que ejecutamos en la terminal.

- Acceder de forma rápida a los comandos almacenados en el historial de nuestra Shell.
- Conocer trucos y los atajos de teclado para ejecutar de forma rápida comandos que se han ejecutado en el pasado.
- Tener conocimientos básicos de programación de Scripts. Los Scripts permiten ejecutar un conjunto de comandos para conseguir un fin de forma rápida y sencilla.

Por lo tanto, si tienen paciencia y dominan los 5 puntos que acabo de citar les aseguro que podrán realizar las tareas de forma sensiblemente más rápida en la terminal. No obstante requiere de un tiempo de aprendizaje y de persistencia.

Evitamos Distracciones Mientras usamos un Programa Las aplicaciones que se ejecutan en la terminal facilitan concentrarte en lo que estás haciendo. Las interfaces de terminal acostumbran a:

- Mostrar únicamente las opciones que son necesarias para proseguir al siguiente paso.
- Tener un número reducido de colores e información en pantalla. La interfaz de uso acostumbra a ser limpia.

En cambio las aplicaciones que se ejecutan mediante una interfaz gráfica acostumbran a ser todo lo contrario a lo que acabo de citar. Por lo tanto las aplicaciones que se ejecutan en la línea de comandos evitan distracciones mientras las estamos usando. También nos deberían permitir ser más productivos.

Interacción con el Sistema Operativo La línea de comandos permite interactuar de forma más directa y precisa con nuestro sistema operativo y saber en todo momento lo que está pasando. Por ejemplo si al ejecutar un programa en la terminal se produce un error nos saldrá un mensaje de error que nos dará alguna pista de lo que está pasando. En cambio si el programa se ejecuta a través de una interfaz gráfica no podremos ver el error de forma directa.

Además cuando lanzamos una aplicación en la terminal te da a entender que estás ejecutando un fichero binario junto a una serie de opciones que nosotros podemos modificar. Esta serie de opciones nos dan un mejor control de las acciones que realizamos en todo momento.

Nota de Usar la Terminal en GNU/Linux En ningún momento voy a decir que la línea de comandos es mejor que una interfaz gráfica o viceversa. Simplemente me limité a citar sus ventajas e inconvenientes y después que cada usuario elija lo que crea conveniente. No obstante, de antemano también digo que la terminal no es para todos los usuarios.

Hoy en día la realidad es que la gran mayoría de usuarios prefieren la interfaz gráfica porque de entrada es más amigable. No obstante si eres un usuario que requiere de muchas tareas repetitivas en uno o más equipos de cómputo, un programador o un administrador de sistemas te deberías plantear iniciarte poco a poco en el uso de la terminal.

1.3 Software Propietario y Libre

Con el constante aumento de la comercialización de las computadoras y su relativo bajo costo, las computadoras se han convertido en un objeto omnipresente, ya que estas se encuentran en las actividades cotidianas de millones de usuarios, en formas tan diversas como teléfonos celulares, tabletas, computadoras portátiles y de escritorio, etc.

Las computadoras por sí solas no resuelven los problemas para los que los usuarios las compran. El Software -sistema operativo y los programas de aplicaciones- son los que realmente generan las soluciones al interactuar uno o más paquetes informáticos con los datos del usuario. También, es común que al comprar una computadora, en el costo total, se integre el del sistema operativo, aplicaciones ofimáticas y de antivirus, sean estos usados por el usuario o no; y en la mayoría de los casos no es posible solicitar que no sean incluidos en el costo de la computadora.

Por otro lado, el Software comercial suele quedar obsoleto muy rápido, ya que constantemente se le agregan nuevas funcionalidades al mismo y estas en general son vendidas como versiones independientes de la adquirida originalmente. Esto obliga al usuario -si quiere hacer uso de ellas- a comprar las nuevas versiones del Software para satisfacer sus crecientes necesidades informáticas. Por lo anterior y dada la creciente complejidad de los paquetes

de cómputo y el alto costo de desarrollo de aplicaciones innovadoras, en muchos casos, el costo total del Software que comúnmente los usuarios instalan -y que no necesariamente usan las capacidades avanzadas del programa, por las cuales el Software tiene un alto costo comercial- en su computadora, suele ser más caro que el propio equipo en el que se ejecutan.

1.3.1 Software Propietario

En entornos comerciales, es posible por parte de la empresa, adquirir y mantener actualizado el Software necesario para sus actividades comerciales, pues el costo del mismo se traslada al consumidor final del bien o servicio que la empresa proporcione. En entornos educativos, de instituciones sin fines lucrativos e incluso, el sector gubernamental, no se cuenta con los recursos necesarios para adquirir y mantener actualizado el Software necesario para todas y cada una de las aplicaciones usadas en las computadoras, ya que en general, las licencias de uso del Software propietario son asignadas en forma individual a cada computadora y no es fácilmente transferible a otra computadora.

Dado que existe una gran demanda de programas de cómputo tanto de uso común como especializado por nuestras crecientes necesidades informáticas, y por la gran cantidad de recursos económicos involucrados, existe una gran cantidad de empresas que tratan de satisfacer dichas necesidades, para generar y comercializar, además de proveer la adecuada documentación y opciones de capacitación que permita a las empresas contratar recursos humanos capacitados.

Por otro lado, generalmente se deja la investigación y desarrollo de productos computacionales nuevos o innovadores a grandes empresas o Universidades -que cuenten con la infraestructura y el capital humano, que muchas veces es de alto riesgo- con la capacidad de analizar, diseñar y programar las herramientas que requieran para sus procesos de investigación, enseñanza o desarrollo.

Existe hoy en día, una gran cantidad de paquetes y sistemas operativos comerciales de Software propietario (véase 9.1) que mediante un pago oneroso, permiten a los usuarios de los mismos ser productivos en todas y cada una de las ramas comerciales que involucra nuestra vida globalizada, pero el licenciamiento del uso de los programas comerciales es en extremo restrictivo en su uso y más en su distribución.

1.3.2 Software Libre

El Software libre (véase sección 9.2) son programas de cómputo -sistema operativo, paquetes de uso común y especializados-, desarrollados por usuarios y para usuarios que, entre otras cosas, comparten el código fuente, el programa ejecutable y dan libertades para estudiar, adaptar y redistribuir a quien así lo requiera el programa y todos sus derivados.

El Software libre es desarrollado por una creciente y pujante comunidad de programadores y usuarios que tratan de poner la mayor cantidad de programas a disposición de todos los interesados, tal que, le permitan al usuario promedio sacar el mayor provecho de la computadora que use.

¿Qué es el Software Libre? La definición exacta y sus diversas variantes se plasman en 9, pero podemos entender el espíritu a través de los documentos de la fundación para el Software libre (véase [13], [2], [9], [10], [8] y [12]). El Software libre concierne a la libertad de los usuarios para ejecutar, copiar, distribuir, cambiar y mejorar el Software:

0. La libertad de usar el programa, con cualquier propósito.
1. La libertad de estudiar cómo funciona el programa y modificarlo, adaptándolo a tus necesidades.
2. La libertad de distribuir copias del programa, con lo cual puedes ayudar a tu prójimo.
3. La libertad de mejorar el programa y hacer públicas esas mejoras a los demás, de modo que toda la comunidad se beneficie.

La lista de proyectos de este tipo es realmente impresionante (véase [13], [2] y [1]). Algunos han conseguido un uso y alta calidad, por ejemplo el compilador GCC (véase [14]), el Kernel de Linux (véase [5]) y el sistema operativo Debian GNU/Linux (véase [6]) y Android (véase [7]). Mientras que otros proyectos han caído en el olvido, pero en la gran mayoría, se tiene copia del código fuente que permitiría a quienes estén interesados en dicho proyecto, el poder revivirlo y en su caso ampliarlo.

La característica más importante que aparece típicamente en un proyecto de este tipo, es que un conjunto de personas separadas a gran distancia, sean capaces, a través de la Web, de los e-mail y de foros, de aunar sus

esfuerzos para crear, mejorar, distribuir un producto, de forma que todos ellos se benefician unos de otros. Evidentemente, la mayor parte del peso recae en los desarrolladores, pero también es necesaria una difusión para que los usuarios documenten, encuentren errores, hagan foros de discusión, etc.

¿Por qué se Interesan los Autores, Alumnos y Profesores Universitarios en el Software Libre? Porque bajo el Software libre subyace la idea de compartir conocimiento y favorecer la existencia de nuevas ideas; y ¿qué es investigar y enseñar sino crear conocimiento y procurar que los alumnos aprendan e incluso vayan más allá de lo aprendido? Se comparte la idea, que el espíritu del Software libre es similar al que debería reinar en las instituciones educativas.

Concretando estas ideas, profesores e investigadores necesitan herramientas para la investigación y docencia, y estas deben de tener una calidad mínima y ser fácilmente distribuibles entre los alumnos. En muchos casos las compañías desarrolladoras y distribuidoras de programas de cómputo no han sabido ofrecer sus productos con la flexibilidad adecuada para las labores docentes o, en otros casos, los productos desarrollados no tienen la calidad esperada.

El Software libre es aún joven, pese a las decenas de miles de proyectos actuales (véase [1] y [3]) -en los que se trabaja constantemente en mejorar la parte computacional de los algoritmos involucrados en el proyecto, haciendo y puliendo interfaces gráficas, generando ayuda en línea así como la documentación necesaria para que usuarios noveles y avanzados usen la mayor cantidad de opciones programadas - existen muchas otras necesidades profesionales y de investigación que requieren el desarrollo innovador de programas de cómputo para automatizarlas y hacerlas eficientes. Esto queda plasmado en las decenas de proyectos que a diario son registrados en las páginas especializadas en busca de difusión y apoyo para su proyecto (véase [1] y [3]).

En los últimos años, muchos proyectos han pasado de ser simples programas en línea de comandos a complejas aplicaciones multiplataforma -se ejecutan en distintos sistemas operativos como son Windows, Linux, Unix, Mac OS, Android- con ambientes gráficos multimedia que en muchos casos han superado a sus contrapartes comerciales -por ejemplo los navegadores Web-. Para muestra de este maravilloso avance, tomemos el proyecto del sistema operativo Android, que actualmente ejecuta en millones de equipos -como celulares, tabletas, electrodomésticos, etc.- y en los cuales se pueden

descargar miles de aplicaciones y está soportado por una gran cantidad de usuarios y empresas comerciales como Google e IBM. Este ha logrado desplazar a muchos de sus competidores por sus múltiples bondades y bajo costo de desarrollo, al reusar miles de aplicaciones ya existentes que usan Software libre y permitir desarrollar otro tanto de aplicaciones bajo una plataforma que se ejecutará en los más diversos procesadores.

Además, el uso de Software libre y su posibilidad de ampliarlo y/o especializarlo según sea necesario, ha permitido crear de forma cada vez más rápida y confiable; y poner a disposición de un gran público programas de uso común, así como especializado que satisfagan las nuevas necesidades de los usuarios.

1.4 Agradecimientos

Este texto es una recopilación de múltiples fuentes, mi aportación -si es que puedo llamarla así- es plasmarlo en este documento, en el que trato de dar coherencia a mi visión de los temas desarrollados.

En la realización de este texto se han revisado -en la mayoría de los casos indicó la referencia, pero pude omitir varias de ellas, por lo cual pido una disculpa- múltiples páginas Web, artículos técnicos, libros, entre otros materiales bibliográficos, los más representativos y de libre acceso los pongo a su disposición en la siguiente liga:

Herramientas
http://132.248.181.216/Herramientas/

Además, la documentación y los diferentes ejemplos que se presentan en este trabajo, se encuentran disponibles en dicha liga, para que puedan ser copiados desde el navegador y ser usados. En aras de que el interesado pueda correr dichos ejemplos y afianzar sus conocimientos, además de que puedan ser usados en diferentes ámbitos a los presentados aquí.

2 Herramientas Administrativas y de Seguridad

Existen diversos sitios Web que están enfocados a explorar detalladamente cada distribución actual o antigua, a un nivel técnico acompañado de grandes y útiles análisis técnicos sobre los mismos, lo que facilita el aprendizaje puntual sobre qué distribución usar o empezar a usar sin tanta incertidumbre, algunos de estos lugares son:

- ArchiveOS https://archiveos.org
- Distro Chooser https://distrochooser.de/es/
- Distro Watch https://distrowatch.com
- Linux Distribution List https://lwn.net/Distributions/

¿Qué otros sabores de GNU/Linux hay?

https://upload.wikimedia.org/wikipedia/commons/1/1b/Linux_Distribution_Timeline.svg

Existen distintas distribuciones de GNU/Linux[5] para instalar, puedes conocer y descargar las diferentes distribuciones desde:

https://es.wikipedia.org/wiki/Anexo:Distribuciones_Linux
https://en.wikipedia.org/wiki/List_of_Linux_distributions

y ver cual es la que más te conviene:

https://en.wikipedia.org/wiki/Comparison_of_Linux_distributions

o probar alguna versión Live[6]:

[5] Una lista de las distribuciones de Linux y su árbol de vida puede verse en la página Web http://futurist.se/gldt/

[6] Linux es uno de los sistemas operativos pioneros en ejecutar de forma autónoma o sin instalar en la computadora, existen diferentes distribuciones Live -descargables para formato CD, DVD, USB- de sistemas operativos y múltiples aplicaciones almacenados en un medio extraíble, que pueden ejecutarse directamente en una computadora, estos se descargan de la Web generalmente en formato ISO.

https://livecdlist.com/

también las puedes correr como máquina virtual para VirtualBox:

https://www.osboxes.org/

o máquina virtual para QEMU/KVM:

https://docs.openstack.org/image-guide/obtain-images.html
https://github.com/palmercluff/qemu-images
https://bierbaumer.net/qemu/

por otro lado, existen diferentes servicios Web que permiten instalar, configurar y usar cientos de sistemas operativos Linux y Unix desde el navegador, una muestra de estos proyectos son:
Distrotest https://distrotest.net
JSLinux https://bellard.org/jslinux
OnWorks https://www.onworks.net

Entre las distintas distribuciones de GNU/Linux para instalar, una de las más ampliamente usadas es Debian GNU/Linux[7] y sus derivados como Ubuntu. La comunidad de Debian GNU/Linux te apoya para que lo obtengas, instales y de una vez por todas puedas usar GNU/Linux en tu computadora.

Tenemos una réplica en México de Debian GNU/Linux en:

http://ftp.mx.debian.org/debian/

de aquí puedes descargar las imágenes de instalación:

http://ftp.mx.debian.org/Replicas/debianInstall/

además de manuales de instalación y administración:

http://ftp.mx.debian.org/Replicas/debianInstall/DebianAdministracion/

con distintas versiones para todos los gustos:

[7] Algunas de las razones para instalar GNU/Linux Debian están detalladas en su página Web https://www.debian.org/intro/why_debian.es.html

https://www.debian.org/releases/index.es.html

también lo puedes correr como máquina virtual (VirtualBox):

https://www.osboxes.org/debian/

o en versiones Live:

https://www.debian.org/CD/live/

¿Por que usar Debian GNU/Linux?

https://www.debian.org/intro/why_debian.es.html

Arquitecturas soportadas (y sí, 32 bits seguirá soportada):

https://www.debian.org/releases/lenny/ia64/ch02s01.html.es

Entornos de escritorio soportados:

https://wiki.debian.org/es/DesktopEnvironment

Derivados de Debian:

https://upload.wikimedia.org/wikipedia/commons/6/69/DebianFamilyTree1210.svg

Administrador del Sistema Las labores del administrador de sistemas pueden incluir:

- Agregar, remover o actualizar información de cuentas, reinicio de contraseñas, etc.
- Análisis de registros de sistema e identificar potenciales inconvenientes con los equipos de cómputo de la red local o remota.
- Instalar actualizaciones de sistemas operativos, correcciones y cambios de configuración.
- Instalar y configurar nuevo Hardware y/o Software.
- Responder consultas técnicas y asistencia a usuarios.

- Es responsable de la seguridad (esta es inherente al cargo).
- Es responsable de documentar la configuración del sistema, ya sea para beneficio propio cuando tenga que tomar vacaciones, o para sus sucesores en el cargo.
- Solución de los problemas que reporten los usuarios.
- Afinación del desempeño del sistema.
- Asegurarse de que la infraestructura de red esté iniciada y funcionando (monitorización de servidores y redes).
- Configuración, adición y borrado de archivos del sistema.
- Supervisar de manera directa que los ambientes de desarrollo, pruebas y producción se sincronicen y funcionen sin inconveniente alguno.

En esta sección exploraremos algunas de estas actividades y cómo lograrlo usando Debian GNU/Linux. Además ponemos tu disposición una amplia bibliografía para aprender a usar, administrar y optimizar cada uno de los distintos aspectos de Linux en:

Sistemas Operativos

2.1 Cuentas de Usuario

El sistema operativo Linux, es un sistema multiusuario y multiplataforma, especialmente creado para trabajar estilo red. Es por esta razón, que cada usuario debe tener una cuenta para poder trabajar bajo GNU/Linux.

La tarea de crear cuentas de usuario es hecha por el administrador (Root ó Súper usuario), el cual tiene la capacidad de incluir nuevos usuarios al sistema y asignarle los permisos que crea convenientes.

Gestionar Usuarios en GNU/Linux Un usuario o cuenta de un sistema se identifica de forma única mediante un número denominado UID (número de identificación único). En GNU/Linux podremos encontrar básicamente tres tipos de usuarios:

Usuario root también llamado usuario administrador o superusuario, con UID 0 y que cuenta con privilegios de acceso y modificación sobre todo el sistema. Será el encargado de realizar configuraciones importantes, instalar programas, actualizaciones y todo lo relativo a la gestión de cuentas de usuarios y acceso total a todos los archivos y directorios con independencia de propietarios y permisos.

Usuarios especiales más que usuarios, son cuentas del sistema que se instalan con determinado Software o que ya vienen por defecto en el sistema. Ejemplo de ello son bin, mail, apache, sshd, etc. No son usuarios normales, pero tampoco llegarán a tener todos los permisos de root. Solamente tendrán activadas ciertas funciones de root en función del propósito para el que estén creados, lo anterior para proteger al sistema de posibles formas de vulnerar la seguridad. No tienen contraseñas pues son cuentas que no están diseñadas para iniciar sesiones con ellas -también se les conoce como cuentas de "no inicio de sesión" (nologin)-. Se crean (generalmente) automáticamente al momento de la instalación de GNU/Linux o de la aplicación y generalmente se les asigna un UID entre 1 y 100 (definido en /etc/login.defs).

Usuarios normales estos serán los que utilizamos en nuestro sistema para realizar las típicas tareas diarias. Contaremos con acceso a la terminal de comandos para tareas básicas, contaremos con un directorio de trabajo para cada uno en /home, así como una UID. Tienen solo privilegios completos en su directorio de trabajo o HOME. Por seguridad, es siempre mejor trabajar como un usuario normal en vez del usuario root, y cuando se requiera hacer uso de comandos solo de root, utilizar el comando su. En las distribuciones actuales de Linux se les asigna generalmente un UID superior a 1000.

2.1.1 El Usuario Administrador del Sistema

El usuario root en GNU/Linux es el usuario que tiene acceso administrativo al sistema. Los usuarios normales no tienen este acceso por razones de seguridad.

Cambiar de Usuario en Linux el comando su (Switch User) se utiliza para cambiar de usuario cuando estamos dentro de la consola de Linux, ejemplo:

```
$ su antonio
```

si delante del usuario ponemos - nos abrirá una nuevo Shell con las preferencias del usuario al que cambiemos, por ejemplo:

```
$ su - administracion
```

Por otro lado, si usamos el comando *su* sin usuario, nos permitirá ingresar como el usuario administrador del sistema *root* (por defecto), pidiendo el clave o *Password* de dicho usuario, ejemplo:

```
$ su
```

Uso de Sudo en múltiples sistemas derivados de Debian GNU/Linux no incluye el usuario root (por ejemplo en todos los derivados de Ubuntu). En su lugar, se da acceso administrativo a usuarios individuales, que pueden utilizar la aplicación *sudo* para realizar tareas administrativas. La primera cuenta de usuario que creó en su sistema durante la instalación tendrá, de forma predeterminada, acceso a *sudo*.

Cuando se necesite ejecutar una aplicación que requiere privilegios de administrador, *sudo* le pedirá que escriba su contraseña de usuario normal. Esto asegura que aplicaciones incontroladas no puedan dañar su sistema, y sirve como recordatorio de que está a punto de realizar acciones administrativas que requieren que tenga cuidado.

Para usar *sudo* en la línea de comandos, simplemente escriba *sudo* antes del comando que desea ejecutar, sudo le pedirá su contraseña[8]:

```
$ sudo apt update
```

[8] *Sudo* recordará su contraseña durante un periodo de tiempo (predeterminado a 15 minutos). Esta característica se diseñó para permitir a los usuarios realizar múltiples tareas administrativas sin tener que escribir su contraseña cada vez.

El Shell y el Prompt los programas, tanto los escritos por el usuario como los de sistemas, normalmente se ejecutan con un intérprete de órdenes. El intérprete de órdenes en Linux es un proceso de usuario como cualquier otro y recibe el nombre de Shell (concha o cáscara) porque rodea al núcleo del sistema operativo.

El Shell indica que está listo para aceptar otra orden exhibiendo una señal de espera (*Prompt*) y el usuario teclea una orden en una sola línea. En el Bourne Shell y sus derivados como Bash el *Prompt* que nos permite escribir los diferentes comandos, generalmente termina con el carácter:

- $ para usuario sin privilegios
- # para el administrador, conocido como *root*

Formas de Encontrar Información de Cuentas de Usuario y Detalles de Inicio de Sesión Hay varias formas de mostrar información acerca de los usuarios en un equipo con GNU/Linux, algunas de ellas son:

- id - Nos proporciona información del usuario, uso:

  $ id antonio

- groups - Indica a los grupos a los que pertenece el usuario, uso:

  $ groups antonio

- finger - Si está instalado, proporciona información sobre los usuarios del sistema, uso:

  $ finger antonio

- getent - Permite obtener entradas de la biblioteca NSS de una base de datos específica del sistema, uso:

  $ getent passwd antonio

- /etc/passwd - Nos permite buscar usuarios en el archivo, uso:

  $ grep -i antonio /etc/passwd

- lslogins - Muestra información de todas las cuentas del sistema, la bandera -u despliega la información de usuarios, uso:

  $ lslogins -u

- users - Despliega los nombre de usuario del sistema, uso:

  $ users

- who - Visualiza a los usuarios ingresados en el sistema, uso:

  $ who -u

- w - Visualiza a los usuarios ingresados en el sistema y desde donde lo hicieron, uso:

  $ w

- last - Muestra a todos los usuarios que han ingresados al sistema, uso:

  $ last

si queremos saber los usuario activos en este momento, podemos usar:

  $ last -ap now

- lastlog - Muestra los detalles de los recientes ingresos de todos los usuarios al sistema, uso:

  $ lastlog

- lastb - Muestra los intentos de inicio de sesión incorrectos, uso:

  # lastb

2.1.2 Creación, Modificación y Eliminación de Cuentas

El proceso de creación de cuentas de usuarios en GNU/Linux es sumamente sencillo, pero debe realizarse de manera metódica, para evitar algún error que nos obligue a crear la cuenta nuevamente. Existen dos formas de realizar esta tarea, una de ellas es completamente gráfica y muy intuitiva. El inconveniente de esta es que si el equipo es servidor, por lo general no tendremos entorno gráfico.

La otra forma, es mediante la terminal en línea de comandos. Es un proceso sencillo y muy eficiente, que se puede realizar desde cualquier equipo de la red, como *root* o mediante el comando *su*.

Se puede hacer a través de tres comandos:

- *adduser*: Crear un nuevo usuario con todos los parámetros predeterminados o actualizar la información del nuevo usuario predeterminado.
- *useradd*: Crear un nuevo usuario o actualizar la información predeterminada del nuevo usuario.
- *newusers*: Actualizar y crear nuevos usuarios en batch.

Y mientras se crea un nuevo usuario, se modificarán los cuatro archivos siguientes:

- */etc/passwd*: Los detalles del usuario se actualizarán en este archivo.
- */etc/shadow*: La información de la contraseña del usuario se actualizará en este archivo.
- */etc/group*: Los detalles del grupo del nuevo usuario se actualizarán en este archivo.
- */etc/gshadow*: La información de la contraseña del grupo del nuevo usuario se actualizará en este archivo.

Archivos de inicialización generales:

- /etc/profile Contiene la configuración del perfil de arranque del *login*. Se ejecuta cada vez que un usuario ingresa al sistema.

- /etc/bashrc Contiene funciones de configuración como el *umask*, *PS1* del Prompt. Se ejecuta cada vez que se invoca una Shell.
- /etc/motd Mensaje del día para todos los usuarios, que será mostrado al inicio de la sesión.
- /etc/default/useradd Configuración de los valores predeterminados al crear un usuario, como ser el directorio personal, el grupo principal.
- /etc/login.defs Configuración de los valores predeterminados al crear un usuario,como el número de usuario y valores de la contraseña.
- /etc/issue Contiene el banner que se mostrará en el momento del *login* local.
- /etc/issue.net Contiene el banner que se mostrará en el momento del *login* remoto, ejemplo por ssh.

Contraseñas las contraseñas o Passwords protegen la información que contienen los dispositivos y cuentas de los usuarios. No obstante, ante la cantidad de claves y combinaciones que cotidianamente se deben utilizar, la mayoría de las personas opta por contraseñas fáciles de recordar por la comodidad que esto implica, o bien, por la falta de conocimiento de lo fácil que puede ser para un ciberdelincuente obtenerlas. Para asegurar la efectividad de las contraseñas[9] y evitar el robo de éstas, es recomendable poner en práctica las siguientes acciones:

- Al generar las contraseñas de los dispositivos y cuentas se deben utilizar claves largas (mínimo 13 caracteres[10]) y únicas para cada caso, evitando utilizar la misma contraseña para diferentes dispositivos o cuentas.

[9] Para conocer la seguridad de una clave, podemos checarla en: https://howsecureismypassword.net/

[10] Solo para tener una idea del tiempo necesario para encontrar la clave de 13 caracteres usando fuerza bruta en un solo procesador: Sólo números (1 hr), mezclar letras mayúsculas y minúsculas (600 años), mezclar números, letras mayúsculas y minúsculas (6 mil años), mezclar números, símbolos, letras mayúsculas y minúsculas (5 mil años).

Si se aumentan más procesadores o múltiples máquinas los tiempos se acortarán de forma drástica.

- Se deben evitar las combinaciones sencillas como fechas de nacimiento, secuencias consecutivas, repeticiones de un mismo dígito o palabras simples como "password" o "contraseña".
- La mayor longitud de la contraseña, así como la incorporación de mayúsculas, minúsculas, números y símbolos (#$,.-_%&*!?), contribuyen a que ésta sea más segura y difícil de vulnerar.
- Se debe evitar escribir contraseñas en papeles o tener archivos con esa información que sean fácilmente accesibles para otros.
- Es importante no facilitar a nadie, aunque así lo solicite, por ningún medio, contraseñas y/o códigos para el inicio de sesión.
- Es recomendable cambiar con frecuencia las contraseñas a efecto de evitar accesos no autorizados.

Creando, Modificando y Borrando una Cuenta de Usuario Lo primero que debemos hacer, es ingresar como root desde la consola de GNU/Linux. Para hacer esto, una vez abierta la consola, tecleamos el siguiente comando:

```
$ su
```

seguidamente, la consola nos pedirá la contraseña, la ingresamos y activaremos los privilegios de root en el terminal. Notemos que el Prompt del sistema es el símbolo $, lo que nos indica que estamos ingresando como usuario estándar. Cuando activemos los privilegios de *su*, el Prompt cambiará al símbolo #.

Crear Cuenta de Usuario una vez activados los privilegios de root, procedemos a crear una nueva cuenta de usuario, para ello usamos:

```
# adduser antonio
```

nos pedirá la contraseña y su confirmación además de información sobre el usuario. De esta forma crea un nuevo usuario con el nombre de antonio, con todas las opciones por defecto, tales como ubicación de su directorio, duración de la cuenta de usuario, Shell a utilizar o grupos en los que va a ser incluido (el identificador de usuario UID y el identificador de grupo GID

serán iguales). Este comando acepta las mismas opciones que el comando *usermod*.

Para conocer la información de un usuario del sistema, usamos:

```
$ id antonio
```

Cambiar Contraseña Usuario en cualquier momento después de creada la cuenta, podemos cambiar la contraseña del usuario, usando:

```
# passwd antonio
```

luego, el sistema nos pedirá que ingresemos la contraseña dos veces para poder verificar si el procedimiento se ha realizado correctamente. Para conocer el estado del Password de mi cuenta puedo usar:

```
$ passwd -S antonio
```

si deseamos conocer el estado del Password de todas las cuentas del sistema usamos:

```
# passwd -Sa
```

El usuario root es el único que puede indicar el cambio o asignación de contraseñas de cualquier usuario. Usuarios normales pueden cambiar su contraseña en cualquier momento con tan solo invocar passwd sin argumentos, y podrá de esta manera cambiar la contraseña cuantas veces lo requiera.

Podemos forzar el cambio de clave de acceso de un usuario en su próximo ingreso al sistema, mediante:

```
# passwd -r usuario
```

Bloquear y Desbloquear un Usuario otra tarea que se puede realizar al gestionar usuarios en GNU/Linux es bloquear y desbloquear una cuenta de usuario. Para bloquear una cuenta, necesitamos invocar passwd con la opción *-l*.

```
# passwd -l antonio
```

la opción -u con passwd desbloquea una cuenta:

```
# passwd -u antonio
```

Borrar Cuenta de un Usuario también, tendremos la capacidad de eliminar cuentas de usuario con el comando:

```
# userdel antonio
```

sin opciones elimina la cuenta del usuario de */etc/passwd* y de */etc/shadow*, pero no elimina su directorio de trabajo ni archivos contenidos en el mismo, esta es la mejor opción, ya que elimina la cuenta pero no la información de la misma.

```
# userdel -r antonio
```

al igual que lo anterior elimina la cuenta totalmente, pero con la opción *-r* además elimina su directorio de trabajo y archivos y directorios contenidos en el mismo, así como su buzón de correo, si es que estuvieran configuradas las opciones de correo. La cuenta no se podrá eliminar si el usuario está en el sistema al momento de ejecutar el comando.

```
# userdel -f antonio
```

la opción -f es igual que la opción *-r*, elimina todo lo del usuario, cuenta, directorios y archivos del usuario, pero además lo hace sin importar si el usuario está actualmente en el sistema trabajando. Es una opción muy radical, además de que podría causar inestabilidad en el sistema, así que hay que usarla solo en casos muy extremos.

Crear Grupos de Trabajo[11] podemos crear grupos completos de usuario, para ello hacemos:

```
# groupadd nombre_grupo
```

y podemos cambiar el nombre de un grupo mediante:

```
# groupmod -n grupo_nuevo grupo_antiguo
```

si necesitamos crear un grupo del sistema, usamos:

```
# groupadd -r nombre_grupo
```

[11] Es posible crear, manipular y borrar grupos mediante el comando *usermod*.

Agregar y Remover usuarios a un Grupo para conocer a los grupos que pertenece un usuario, usamos:

```
$ groups usuario
```

podemos agregar un usuario a un grupo, mediante:

```
# gpasswd -a usuario nombre_grupo
```

y podemos borrar un usuario de un grupo, usando:

```
# gpasswd -d usuario nombre_grupo
```

podemos conocer los grupos a los que pertenece un usuario, usando:

```
$ groups usuario
```

si queremos conocer la lista de todos los grupos existentes, usamos:

```
$ getent group
```

Borrar Grupos de Trabajo podemos borrar grupos completos de usuarios, mediante:

```
# groudel nombre_grupo
```

Access Control List también están disponibles los comandos *getfacl* y *setfacl* pertenecientes a *Access Control List* (ACLs) que es un método más flexible para establecer permisos a usuarios y grupos sobre archivos y directorios. Por ejemplo podemos ver los permisos y grupos a los que pertenece un archivo o directorio mediante:

```
$ getfacl archivo
```

y cambiarlos usando:

```
$ setfacl -m g:invitado:7 archivo
```

Archivos de configuración Los usuarios normales y root en sus directorios de inicio tienen varios archivos que comienzan con "." es decir están ocultos. Varían mucho dependiendo de la distribución de Linux que se tenga, pero seguramente se encontrarán los siguientes o similares:

```
# ls -la

drwx– 2 alumno alumno 4096 jul 9 09:54 .
drwxr-xr-x 7 root root 4096 jul 9 09:54 ..
-rw-r–r– 1 alumno alumno 24 jul 9 09:54 .bash_logout
-rw-r–r– 1 alumno alumno 191 jul 9 09:54 .bash_profile
-rw-r–r– 1 alumno alumno 124 jul 9 09:54 .bashrc
```

.bash_profile aquí podremos indicar alias, variables, configuración del entorno, etc. que deseamos iniciar al principio de la sesión.

.bash_logout aquí podremos indicar acciones, programas, Scripts, etc., que deseemos ejecutar al salirnos de la sesión.

.bashrc es igual que .bash_profile, se ejecuta al principio de la sesión, tradicionalmente en este archivo se indican los programas o Scripts a ejecutar, a diferencia de *.bash_profile* que configura el entorno.

Si deseamos configurar archivos de inicio o de salida de la sesión gráfica entonces, en este caso, hay que buscar en el menú del ambiente gráfico algún programa gráfico que permita manipular que programas se deben arrancar al iniciar la sesión en modo gráfico. En la mayoría de las distribuciones existe un programa llamado "sesiones" o "sessions", generalmente está ubicado dentro del menú de preferencias. En este programa es posible establecer programas o Scripts que arranquen junto con el ambiente gráfico, sería equivalente a manipular 'bashrc'.

Además GNU/Linux permite que el usuario decida que tipo de entorno Xwindow a utilizar, ya sea algún entorno de escritorio como *KDE* o *Gnome* o algún manejador de ventanas como *Xfce* o *Twm*. Dentro del Home del usuario, se creará un directorio o archivo escondido "." , por ejemplo '.gnome'

o '.kde' donde vendrá la configuración personalizada del usuario para ese entorno. Dentro de este directorio suele haber varios directorios y archivos de configuración. Estos son sumamente variados dependiendo de la distribución y del entorno. No es recomendable modificar manualmente (aunque es perfectamente posible) estos archivos, es mucho más sencillo modificar vía la interfaz gráfica, que permiten cambiar el fondo, protector de pantalla, estilos de ventanas, tamaños de letras, etc.

/etc/passwd Cualquiera que sea el tipo de usuario, todas las cuentas se encuentran definidas en el archivo de configuración 'passwd', ubicado dentro del directorio /etc. Este archivo es de texto tipo ASCII, se crea al momento de la instalación con el usuario root y las cuentas especiales, más las cuentas de usuarios normales que se hayan indicado al momento de la instalación.

El archivo */etc/passwd* contiene una línea para cada usuario del sistema, y puede ser visualizada usando:

```
$ cat /etc/passwd
```

o podemos usar el comando *column* para mostrarla algo más legible:

```
$ column -s ":" -t /etc/passwd
```

Cada línea contiene algo similar a:

```
root:x:0:0:root:/root:/bin/Bash
antonio:x:501:500:Antonio Carrillo:/home/antonio:/bin/Bash
```

La información de cada usuario está dividida en 7 campos delimitados cada uno por ':' dos puntos.

/etc/passwd

- Campo 1 Es el nombre del usuario, identificador de inicio de sesión (login). Tiene que ser único.
- Campo 2 La 'x' indica la contraseña cifrada del usuario, además también indica que se está haciendo uso del archivo /etc/shadow, si no se hace uso de este archivo, este campo se vería algo así como: 'ghy675gjuXCc12r5gt78uuu6R'.

- Campo 3 Número de identificación del usuario (UID). Tiene que ser único. 0 para root, generalmente las cuentas o usuarios especiales se numeran del 1 al 100 y las de usuario normal del 1000 en adelante.
- Campo 4 Numeración de identificación del grupo (GID). El que aparece es el número de grupo principal del usuario, pero puede pertenecer a otros, esto se configura en /etc/groups.
- Campo 5 Comentarios o el nombre completo del usuario.
- Campo 6 Directorio de trabajo (Home) donde se sitúa al usuario después del inicio de sesión.
- Campo 7 Shell que va a utilizar el usuario de forma predeterminada.

Podemos ver las características de la cuenta usando:

```
# chage -l antonio
```

y

```
$ id antonio
```

/etc/shadow Anteriormente (en sistemas Unix) las contraseñas cifradas se almacenaban en el mismo */etc/passwd*. El problema es que 'passwd' es un archivo que puede ser leído por cualquier usuario del sistema, aunque solo puede ser modificado por root. Con cualquier computadora potente de hoy en día, un buen programa de descifrado de contraseñas y paciencia es posible "crackear" contraseñas débiles (por eso la conveniencia de cambiar periódicamente la contraseña de root y de otras cuentas importantes). El archivo 'shadow', resuelve el problema ya que solo puede ser leído por *root*. Considérese a 'shadow' como una extensión de 'passwd' ya que no solo almacena la contraseña cifrada, sino que tiene otros campos de control de contraseñas.

El archivo /etc/shadow contiene una línea para cada usuario, similar a las siguientes:

```
root:ghy675gjuXCc12r5gt78uuu6R:10568:0:99999:7:7:-1::
antonio:rfgf886DG778sDFFDRRu78asd:10568:0:-1:9:-1:-1::
```

La información de cada usuario está dividida en 9 campos delimitados cada uno por ':' dos puntos.

/etc/shadow

- Campo 1 Nombre de la cuenta del usuario.
- Campo 2 Contraseña cifrada, un '*' indica cuenta de 'nologin'.
- Campo 3 Días transcurridos desde el 1/ene/1970 hasta la fecha en que la contraseña fue cambiada por última vez.
- Campo 4 Número de días que deben transcurrir hasta que la contraseña se pueda volver a cambiar.
- Campo 5 Número de días tras los cuales hay que cambiar la contraseña. (-1 significa nunca). A partir de este dato se obtiene la fecha de expiración de la contraseña.
- Campo 6 Número de días antes de la expiración de la contraseña en que se le avisará al usuario al inicio de la sesión.
- Campo 7 Días después de la expiración en que la contraseña se inhabilitará, si es que no se cambió.
- Campo 8 Fecha de caducidad de la cuenta. Se expresa en días transcurridos desde el 1/Enero/1970 (epoch).
- Campo 9 Reservado.

/etc/group Este archivo guarda la relación de los grupos a los que pertenecen los usuarios del sistema, contiene una línea para cada usuario con tres o cuatro campos por usuario:

```
root:x:0:root
test:x:501:
antonio:x:502:ventas,supervisores,produccion
alumno:x:503:ventas,pedro
```

- Campo 1 indica el usuario.
- Campo 2 'x' indica la contraseña del grupo, que no existe, si hubiera se mostraría un 'hash' cifrado.

- Campo 3 es el Group ID (GID) o identificación del grupo.
- Campo 4 es opcional e indica la lista de grupos a los que pertenece el usuario

Actualmente al crear al usuario con useradd se crea también automáticamente su grupo principal de trabajo GID, con el mismo nombre del usuario. Es decir, si se añade el usuario 'antonio' también se crea el /etc/group el grupo 'antonio'.

Podemos ver las características de la cuenta usando:

```
# chage -l antonio
```

y

```
$ id antonio
```

/etc/login.defs En el archivo de configuración */etc/login.defs* están definidas las variables que controlan los aspectos de la creación de usuarios y de los campos de shadow usadas por defecto. Algunos de los aspectos que controlan estas variables son:

- Número máximo de días que una contraseña es válida PASS_MAX_DAYS
- El número mínimo de caracteres en la contraseña PASS_MIN_LEN
- Valor mínimo para usuarios normales cuando se usa useradd UID_MIN
- El valor umask por defecto UMASK
- Si el comando useradd debe crear el directorio home por defecto CREATE_HOME

Basta con leer este archivo para conocer el resto de las variables que son autodeScriptivas y ajustarlas al gusto.

Comandos de administración y control de usuarios Existen varios comandos más que se usan muy poco en la administración de usuarios, que sin embargo permiten administrar aún más a detalle a tus usuarios de Linux. Algunos de estos comandos permiten hacer lo mismo que los comandos previamente vistos, solo que de otra manera 'chage', 'id', ' gpasswd', 'groupmod', 'groups', 'shadow', 'chfn', 'groupmod', 'grpck', 'pwck', 'vigr', 'vipw'; y otros como 'chpasswd' y 'newusers' resultan muy útiles y prácticos cuando de dar de alta a múltiples usuarios se trata y si se necesita trabajar con control de accesos los comando 'setfacl' y 'getfacl' son una mejor opción.

2.2 Información del Equipo y Sistema Operativo

Antes de proceder a realizar actividades administrativas es bueno saber lo más posible de nuestro sistema operativo, para ello existen múltiples comando que nos proporcionan esta información, a saber:

Algunos comando usados para conocer y manipular el Hardware:

- lscpu - Información de procesador, uso:

  $ lscpu

- lshw - Lista de Hardware en Linux, uso:

  $ lshw

- hwinfo - Información del Hardware en Linux, uso:

  # hwinfo

- dmidecode - Lista de Hardware de Linux, uso:

  # dmidecode

- hardinfo - Información del Hardware en Linux, uso:

  # hardinfo

- cpuid - Información del CPU, uso:

  $ cpuinfo

- cpufreq-info - Información sobre las frecuencias soportadas por el equipo, uso:

  # cpufreq-info

- lspci - Lista todos los dispositivos PCI, uso:

  $ lscpi

  por ejemplo, podemos obtener la cantidad de RAM de la tarjeta de video:

  lspci | grep -i "Controlador VGA" | cut -d' ' -f1 | xargs lspci -v -s | grep ' prefetchable'

- lspcmcia - Información de PCMCIA, uso:

  # lspcmcia

- lsscsi - Listar dispositivos SCSI, uso:

  $ lsscsi

- lsusb - Lista de los buses USB y detalles del dispositivo, uso:

  $ lsusb

- lsof - Información de todos los archivos abiertos y sus procesos, uso:

  $ lsof

- swapon - Nos muestra la el tamaño de la partición de intercambio (Swap), uso:

  $ swapon

- inxi - Script completo para Bash con múltiple información, uso:

  $ inxi -F

- i7z - Información en tiempo real sobre cada core, uso:

$ i7z

- stress - Permite generar carga para medir rendimiento del equipo –cpu (cpu), –vm (memoria), –io (entrada/salida), uso:

  $ stress –cpu 2

- sysbench - Permite conocer el rendimiento de un sistema con Linux

  $ sysbench cpu –cpu-max-prime=20000 run

- nproc - Imprime el número de cores del equipo, uso:

  $ nproc

- free - Da información de la memoria RAM y Swap, uso:

  $ free

- lsipc - Muestra la información de los inter-procesos de comunicación, uso:

  $ lsipc

- lslocks - Muestra la información de todos los bloqueos de archivos activos, uso:

  $ lslocks

- lsmem - Enumera los rangos de memoria disponibles, uso:

$ lsmem

- lsmod - Lista de los módulos cargados en el Kernel, uso:

  $ lsmode

- tuptime - Reporte histórico de uso del sistema en tiempo real, uso:

  $ tuptime

- modprobe - Adiciona o remueve módulos del Kernel, uso:

  # modprobe vmhgfs
  # modprobe -r vmhgfs

- sensors - Lee información del CPU como la temperatura, voltaje y velocidad de los ventiladores[12], uso:

[12] Para instalarlo usamos:

```
$ sensors
```

Algunos comando usados para conocer y manipular la red:

- ip - Información sobre los dispositivos de red y su configuración, uso:

```
$ ip address
```

- macchanger - Permite cambiar la MAC Addres de nuestra tarjeta de red, uso:

```
# macchange -r wlp3s0
```

- nmcli - Información sobre los dispositivos de red y su configuración, uso:

```
$ nmcli
```

- ping - Envia ICMP ECHO_REQUEST a servidores en red, uso:

```
$ ping 192.168.1.1
```

- whois - Proporciona información de un dominio, uso:

```
$ whois www.google.com
```

- nslookup - Nos da información de DNS de una dirección IP, uso:

```
$ nslookup www.google.com
```

- iperf - Permite hacer mediciones de rendimiento en red en tiempo real, uso:

```
$ iperf -s
```

```
# apt install lm_sensors
```

lo configuramos mediante:

```
# sensors-detect
```

- iperf3 - Permite hacer mediciones de rendimiento en red en tiempo real, uso:

  $ iperf3 -s -f K

- ifstat - Imprime las estadísticas de las interfaces de red, uso:

  $ ifstat

- nc - Herramienta versátil y potente de red, uso:

  $ nc -v -n 127.0.0.1 1-100

- ss - Nos muestra las estadísticas de los Sockets TCP/UDP entre otros, uso:

  $ ss

Algunos comando usados para conocer discos y sus particiones:

- df - espacio en disco de los sistemas de archivos, uso:

  $ df -h

- lsblk - lista de dispositivos de bloque, uso:

  $ lsblk

- fdisk - permite manipular la tabla de particiones de un disco, uso:

  # fdisk /dev/sda

- fsck[13] - permite detectar y corregir errores de los discos, uso:

  # fsck -ay /dev/sda1
  # fsck.ext4 -py /dev/sda1

[13] Existen comandos específicos para diferentes tipos de sistemas de archivos, por ejemplo: fsck.cramfs, fsck,exfat, fsck.ext2, fsck.ext3, fsck.ext4, fsck.fat, fsck.minix, fsck.msdos, fsck.vfat.

- badblocks - permite buscar errores en un dispositivo, uso:

  # badblocks -w -s -o error.log /dev/sda

- parted - permite manipular la tabla de particiones de un disco, uso:

  # parted /dev/sda1

- gparted - permite manipular la tabla de particiones de un disco, uso:

  # gparted

- fio - Permite hacer pruebas de dispositivos de entrada/salida, uso:

  $ fio –name=global –rw=randread –size=128m –name=job1 –name=job2

- mount - Permite montar y desmontar y ver sistema de archivo montados, uso:

  $ mount

- hdparm - Información de disco duro, uso:

  # hdparam -I /dev/sda2

Algunos comando para conocer datos del sistema operativo:

- uname, Imprime detalles de la máquina y del sistema operativo que se esta ejecutando, uso:

  $ uname -a

- lsb_release, Imprime información del sistema operativo, uso:

  $ lsb_release -a

- hostnamectl, Imprime información del equipo y del sistema operativo, uso:

$ hostnamectl

- /etc/os-release, Información del sistema operativo, uso:

  $ cat /etc/os-release

- /etc/issue, Información del sistema operativo, uso:

  $ cat /etc/issue

Archivos del directorio */proc/*, contienen información accesible usando el comando *cat* -muchos de los comando de Linux toman su información de este lugar-*:*

- */proc/* - Información de los procesos corriendo en el sistema
- */proc/cpuinfo* - Información de CPU
- */proc/meminfo* - Información de la memoria
- */proc/modules* - Información de los módulos cargados al Kernel
- */proc/filesystems* - Información de los sistemas de archivos soportados
- */proc/crypto* - Información sobre los sistemas de cifrado disponibles
- otras opciones son: brcm_monitor0, keys, loadavg, mtrr, softirqs, version, buddyinfo, devices, interrupts, key-users, locks, pagetypeinfo, stat, vmallocinfo, cgroups, diskstats, iomem, kmsg, partitions, swaps, vmstat, cmdline, dma, ioports, kpagecgroups, misc, sched_debug, sysrq-trigger, zoneinfo, consoles, execdomains, kallsyms, kpagecount, schedstat, timer_list, fb, kcore, kpageflags, mounts, slabinfo, uptime, etc.

Comandos que nos proporcionan informaciòn sobre dependencias y bibliotecas son:

- ldd - Nos muestra las dependencias de objetos compartidos de un comando, uso:

  $ ldd /bin/ls

- ltrace - Un rastreador de llamadas a biblioteca, uso:

  $ ltrace ls

- hexdump - Despliega el contenido de un archivo en ASCII, decimal, hexadecimal o octal, uso:

  $ hexdump -c /bin/ls

- strings - Imprime las cadenas de caracteres imprimibles en el archivo, uso:

  $ strings /bin/ls

- readelf - Muestra la información del formato de archivo ejecutable y enlazable (ELF), uso:

  $ readelf -h /bin/ls

- objdump - Muestra información de un archivo objeto, uso:

  $ objdump -d /bin/ls

- strace - Nos muestra el sistema de rastreo de llamadas y señales, uso:

  $ strace -f /bin/ls

- nm - Lista los símbolos de los archivos de objeto, usó:

  $ nm archivoobjeto

- gdb - El depurador de GNU, uso:

  $ gdb -q ./ejecutable

Comandos que nos permiten apagar o reinicializar el sistema:

- shutdown - permite apagar o reinicializar el equipo, usó:

```
# shutdown -h now
# shutdown -r now
# shudown +5 "Guarda tus archivos, el equipo se apagará en
5 minutos"
```

- halt - Detiene todas las funciones del equipo, usó:

```
#halt
```

- poweroff - apaga el equipo, usó:

```
# poweroff
```

- reboot - reinicializa el equipo, usó:

```
# reboot
```

2.3 Instalar, Actualizar y Borrar Paquetes

Un paquete de Software en un Sistema Operativo GNU/Linux es generalmente un archivo comprimido que posee una estructura interna predefinida que facilita y permite que el mismo sea manipulado por herramientas de gestión de Software (*Synaptic, Pacman, Yum, Entropy, ZYpp,* etc.) para lograr su compilación y/o instalación, actualización y/o eliminación sobre el sistema operativo, de forma cómoda, segura, estable y centralizada.

Un paquete es compilable si su instalación se basa en su código fuente directamente (Ejm. **.tar.gz*) o instalable si lo hace en binarios compilados ya para una determinada arquitectura o plataforma (Ejm. **.deb*). La mayoría de los paquetes vienen con su documentación incluida, sus Scripts de pre y post instalación, sus archivos de configuración inicial, sus archivos de recursos, y sus binarios o el código fuente con todo lo necesario si está destinado a ser compilado.

La mayoría de los formatos de paquete vienen con sus correspondientes herramientas de gestión de Software, los más conocidos son los *.deb* creado para la distribución Debian GNU/Linux y todas sus distribuciones derivadas, y los *.rpm* creados por *Red Hat* para su propia distribución y derivadas como *Fedora* y *Open SUSE*. También existen los paquetes compilables *.ebuilds* de *Gentoo*.

El que un paquete haya sido creado para una distribución en particular no implica que sólo se pueda usar en esa distribución o derivadas, ya que basta con tener herramientas especializadas en cualquier otra distribución para la gestión de estos formatos para poder usarlos. Entre esas Herramientas tenemos: *dpkg*, *apt-get*, *aptitude*, *rpm*, *emerge*, *alíen*, entre otros.

Cada distribución mantiene almacenada su paquetería en repositorios, tanto en medios como *CDs / DVDs* como en servidores remotos, lo que permite actualizar e instalar por red todo o parte del sistema operativo desde una localización segura y confiable (repositorios oficiales) para no tener que andar buscando servidores desconocidos (e inseguros) a menos que fuese estrictamente necesarios.

Cada distribución suele aportar sus propios paquetes (parches) de seguridad y de mejoras (actualizaciones), para así lograr poner a disposición de sus comunidades de usuarios gran cantidad de Software perfectamente funcional e integrado en el sistema operativo. Y en cuanto a las dependencias entre cada paquete, las mismas suelen gestionarse de forma automática para evitar posibles problemas a los usuarios menos expertos.

¿Compilar o Instalar? lo bueno de compilar frente al instalar se puede decir que lo principal es la posibilidad de especificar opciones de compilación para tu sistema y Software usado que permiten aprovechar mejor los recursos y ajustarse a las preferencias del usuario/administrador, y lo malo lo lento y complicado que puede tornarse este proceso. Ya que por lo general, el instalar un paquete (*.deb* o *.rpm*) es muy rápido y sencillo, pero suele no conseguirse bien actualizado o ajustado a la distribución de nuestro uso o recursos de nuestro equipo de cómputo.

La Retrocompatibilidad es un enorme dolor de cabeza, tomar Software hecho para Linux de hace 10 o 5 años y ejecutarlo en una distribución moderna[14]. Cualquier cosa de mínima complejidad o que use una GUI, simplemente no funciona. Mientras la retrocompatibilidad en Windows es simplemente increíble. En Linux somos dependientes de los repo-sitorios en línea, y cuando una aplicación depende de ciertas librerías que empiezan a de-

[14]Siempre estamos en posibilidad de usar una Máquina Virtual que nos permite usar un programa desarrollado hace años o décadas en su entorno original, corriendo en un equipo moderno con un sistema operativo de última generación con todas sus actualizaciones de seguridad pertinentes.

saparecer de esos repositorios, nos encontramos en una pesadilla. Y mientras más viejo el Software, peor.

Si tengo un Software ahora y quiero ejecutarlo dentro de cinco o diez años en el futuro ¿Por qué no debería ser capaz de hacerlo?, Parte de la belleza del Open Source es que el código fuente está disponible, por lo que es más fácil mantener vivo el Software, de modo que no muera cuando alguien deja de mantenerlo. Excepto que mantener el Software en Linux se está convirtiendo en un desafío tan grande que daría igual que fuese privativo. Porque no vamos a ser capaces de hacerlo funcionar en un tiempo razonable, a menos que seas un desarrollador, e incluso entonces te va a costar muchos dolores de cabeza, y vas a dejar algo sin funcionar en el camino.

Instalar, Actualizar y Borrar Paquetes existen distintos comandos para hacer la instalación, actualización y borrado de paquetes en Debian GNU/Linux, el más usado actualmente es *apt* (Advanced Packaging Tool lanzado en 2014, herramienta avanzada de empaquetado que es una interfase del paquete *apt-get lanzado en 1998*), es un programa de gestión de paquetes *.deb* creado por el proyecto Debian, *apt* simplifica en gran medida la instalación, actualización y eliminación de programas en GNU/Linux. Existen también programas que proporcionan estos mismos servicios como: *apt-get*, *aptitude*, *tasksel* y *dpkg*.

2.3.1 apt-get

Diariamente se actualizan decenas de paquetes en los servidores de Debian GNU/Linux, por ello es necesario hacer el mantenimiento a los paquetes del sistema cada vez que usemos el equipo de cómputo y tengamos acceso a internet, para ello debemos ser el usuario administrador *root*, mediante el comando[15]:

```
$ su
```

[15]En el Bourne Shell y sus derivados como BASH el *Prompt* que nos permite escribir los diferentes comandos, generalmente termina con el carácter:

- $ para usuario sin privilegios
- # para el administrador, conocido como *root*

ya siendo *root*, podemos solicitar la descarga de las actualizaciones disponibles, usando[16]:

```
# apt-get update
```

para conocer qué paquetes instalados en el sistema están listos para ser actualizados, usamos:

```
# apt-get check
```

después, se solicita la descarga, actualización y en su caso borrado de los paquetes, mediante la descarga de los *.deb*, usando:

```
# apt-get upgrade
```

al final, se solicita el borrado de los archivos .deb de los paquetes descargados (Cache de descarga), mediante:

```
# apt-get clean
```

Algunas veces, si se interrumpe el proceso de instalación, es posible corregir este proceso mediante:

```
# dpkg –configure -a
```

Si algún paquete instalado automáticamente ya no es requerido porque se borraron sus dependencias podemos solicitar su desinstalación mediante:

```
# apt-get autoremove
```

Si algún paquete ya no es requerido por tener ya instalada una nueva versión del mismo, podemos solicitar su desinstalación mediante:

```
# apt-get autoclean
```

El comando *apt-get* soporta las siguientes opciones:

[16] Si se trabaja en algunas distribuciones derivadas de Debian, es necesario usar sudo antes del comando apt-get, por ejemplo:
sudo apt-get update

Actualizar Listas:

```
# apt-get update
```

Revisar actualización de listas:

```
# apt-get check
```

Instalar paquete:

```
# apt-get install paquete
# apt-get install '*nombre*'
# apt-get install paquete=<version>
```

si sólo queremos simular la instalación usamos:

```
# apt-get -s install paquete
```

instala el paquete indicado solo actualizando con una nueva versión, usamos:

```
# apt-get install paquete –only-upgrade
```

instala el paquete indicado sin actualizar la versión instalada, usamos:

```
# apt-get install paquete –no-upgrade
```

para instalar evitando que se agreguen paquetes no necesarios, usamos:

```
# apt-get install –no-install-recommends paquete
```

Reinstalar paquete:

```
# apt-get install –reinstall paquete
```

Actualizar Distro:

```
# apt-get upgrade
# apt-get dist-upgrade
# apt-get full-upgrade
```

Actualizar paquete:

```
# apt-get upgrade paquete
```

Actualizar paquetes usando dselect:

```
# apt-get dselect-upgrade
```

Eliminar paquetes:

```
# apt-get remove
# apt-get autoremove
```

Purgar paquetes:

```
# apt-get purge paquete
```

si sólo queremos simular la operación para conocer sus efectos usamos:

```
# apt-get -s purge paquete
```

Conocer paquete:

```
# apt-cache show paquete
# apt-cache showpkg paquete
# apr-cache policy paquete
# apt-cache depends paquete
# apt-cache rdepends –installed –recurse paquete
```

Listar paquetes:

```
# apt-cache search paquete
# apt-cache pkgnames paquete
# apt-cache search –name-only paquete
```

Listar dependencias de un paquete:

```
# apt-cache depends paquete
```

Listar paquetes instalados:

```
# apt-cache pkgnames –generate
# apt-show-versions
```

Validar dependencias incumplidas de un paquete:

```
# apt-cache unmet paquete
```

Configurar dependencias de un paquete:

```
# apt-get build-dep paquete
```

Descargar paquetes:

```
# apt-get source paquete
```

Corregir problemas post-instalación de paquetes:

```
# apt-get install -f
```

Forzar ejecución de orden de comando:

```
# apt-get comando -y
```

Eliminar descargas de paquetes:

```
# apt-get clean
```

Eliminar paquetes obsoletos y sin usos:

```
# apt-get autoclean
```

Trabajar con las claves GPG de un repositorio

```
# apt-key list
# apt-key del <clave>
```

Otros importantes:

```
# apt-file update
# apt-file search paquete
# apt-file list paquete
```

2.3.2 apt

Este comando es una interfase del paquete *apt-get* que viene ya instalado por omisión en el sistema, cuya sintaxis básica es:

```
# apt [opciones] comando
# apt [opciones] comando paquete
```

para usarlo debemos ser el usuario administrador *root*, mediante el comando[17]:

```
$ su
```

ya siendo *root*, podemos solicitar la descarga de las actualizaciones disponibles, usando[18]:

```
# apt update && apt upgrade && apt clean
```

Por ejemplo, si deseamos eliminar el paquete htop e instalar el paquete vim en un solo paso, usamos:

```
# apt purge htop vim+
```

u otra forma de hacer lo mismo:

```
# apt install htop- vim
```

los sufijos + (agregar) y - (eliminar) de los nombres de los paquetes pueden anular y cambiar los interruptores de instalación/eliminación (es una función de ahorro de tiempo y escritura).

Algunas veces, si se interrumpe el proceso de instalación, es posible corregir este proceso mediante:

[17] En el Bourne Shell y sus derivados como BASH el *Prompt* que nos permite escribir los diferentes comandos, generalmente termina con el carácter:

- $ para usuario sin privilegios
- # para el administrador, conocido como *root*

[18] Si se trabaja en algunas distribuciones derivadas de Debian, es necesario usar sudo antes del comando apt, por ejemplo:
sudo apt update

```
# apt –fix-broken install
# dpkg –configure -a
```

para completar las instalaciones pendientes usar

```
# apt -f upgrade
```

En algunas ocasiones es necesario saber que pasaría si instalamos o borramos un determinado paquete, para estos casos existe la opción –simulate, que se usa:

```
# apt install vlc –simulate
# apt purge vlc –simulate
```

El comando *apt* soporta las siguientes opciones:

Editar la información de /etc/apt/sourcelist

```
# apt edit-sources
```

Actualizar:

```
# apt update
```

Conocer los paquetes susceptibles a ser actualizados:

```
# apt list –upgradable
```

Buscar paquetes:

```
$ apt search php
$ apt search mysql-5.?
$ apt search mysql-server-5.?
$ apt search httpd*
$ apt search ^apache
$ apt search ^nginx
$ apt search ^nginx$
```

Instalar paquete:

```
# apt install paquete
# apt -s install paquete
# apt install paquete –no-upgrade
# apt install paquete –only-upgrade
```

Reinstalar paquete:

```
# apt reinstall paquete
# apt install –reinstall paquete
```

Actualizar la distribución:

```
# apt upgrade
# apt full-upgrade
# apt dist-upgrade
```

Actualizar paquete:

```
# apt upgrade paquete
```

Eliminar paquete:

```
# apt remove paquete
# apt autoremove
```

Purgar paquete:

```
# apt purge paquete
# apt -s purge paquete
```

Conocer paquete:

```
# apt show paquete
# apt showsrc paquete
# apt changelog paquete
# apt depends paquete
# apt rdepends paquete
# apt rdepends –installed –recurse paquete
# apt hold paquete
# apt unhold paquete
# apt policy
```

Listar paquetes:

```
# apt list paquete
# apt list –installed
# apt list –upgradeable
# apt list –all-versions paquete
```

Listar paquetes instalados / actualizables:

```
# apt list –installed
# apt list –upgradeable
```

Corregir problemas post-instalación de paquetes:

```
# apt install -f
```

Forzar ejecución de orden de comando:

```
# apt comando -y
```

Eliminar descargas de paquetes:

```
# apt clean
```

Eliminar paquetes obsoletos y sin usos:

```
# apt autoclean
```

Otros importantes:

```
# apt edit-sources
# apt build-dep paquete
# apt source paquete
# apt download paquete
# apt-mark
# apt-mark hold paquete
# apt-mark unhold paquete
```

Cuando se hace una búsqueda, el comando *apt* toma información sobre el estado de la aplicación en su sistema además de lo que arroja la búsqueda de apt-cache, al inicio de la salida hay algunas banderas. Aquí hay una lista de banderas que verá en su terminal y lo que significan:

A- se instala automáticamente, tal vez como parte de un meta-paquete más grande o la instalación del sistema operativo en sí.

B- El paquete está marcado como roto y debe reinstalarse.

H- Medio instalado. El paquete debe terminar de instalarse.

c- El paquete fue eliminado, pero su "fantasma" persiste en forma de archivos de configuración. Puede resolver esto usando *apt-get purge* o *apt purge*, seguido del nombre del paquete con esta bandera.

p- El paquete se purgó o nunca se instaló.

v- El paquete es usado genéricamente por otros para proporcionar una función. Por ejemplo, Firefox proporciona capacidades de navegación que pueden ser utilizadas por otras aplicaciones, lo que lo convierte en un paquete virtual.

i- Este paquete está instalado en su sistema.

h- Hay una retención en este paquete, lo que evita que se actualicen a versiones más recientes.

2.4 Instalación de los Paquetes más Usados

Existen distintas distribuciones de Linux[19] para instalar, una de las más ampliamente usadas es Debian GNU/Linux y sus distribuciones derivadas como Ubuntu.

Debian GNU/Linux es una distribución sólida como una roca con más de 64,419 paquetes disponibles en sus repositorios oficiales. Es adecuado para servidores, estaciones de trabajo, dispositivos móviles y sistemas integrados. Además tiene un sistema de instalación simple y limpio que permite instalarlo con poco esfuerzo y soporta las siguientes arquitecturas:

- 32-bit PC (i386) y 64-bit PC (amd64)
- 64-bit ARM (arm64)
- ARM EABI (armel)

[19] Una lista de las distribuciones de Linux y su árbol de vida puede verse en la página Web http://futurist.se/gldt/

- ARMv7 (EABI hard float ABI, armhf)
- little-endian MIPS (mipsel)
- 64-bit little-endian MIPS (mips64el)
- 64-bit little-endian PowerPC (ppc64el)
- IBM System z (s390x)

Entre los diferentes entornos de escritorio que pueden ser seleccionados al momento de instalar son:

- GNOME
- KDE Plasma
- LXDE
- LXQt
- MATE
- Xfce

Versiones de Debian GNU/Linux siempre mantiene al menos tres versiones en mantenimiento activo: "estable", "en pruebas" e "inestable" ("stable", "testing" y "unstable"). Estas permiten a cada tipo de usuario actualizar la distribución con paquetes estables, en pruebas o inestables pero que proporcionan características de última generación.

Estable

- La distribución "estable" contiene la publicación oficial más reciente de Debian GNU/Linux.
- Esta es la versión de producción de Debian GNU/Linux , cuyo uso recomendamos principalmente.
- La versión "estable" actual de Debian GNU/Linux es la 12, llamada "bookworm". Fue publicada originalmente el 10 de junio del 2023.

En pruebas

- La distribución "en pruebas" ("testing") contiene paquetes que aún no han sido aceptados en la rama "estable", pero están a la espera de ello. La principal ventaja de usar esta distribución es que tiene versiones más recientes del Software.
- Vea las Preguntas frecuentes de Debian GNU/Linux si desea más información sobre qué es "pruebas" y cómo se convierte en "estable".
- La distribución actual de "en pruebas" es "trixie".

Inestable

- La distribución "inestable" es donde tiene lugar el desarrollo activo de Debian GNU/Linux. Generalmente, esta distribución es la que usan los desa-rrolladores y aquellos que quieren estar a la última. Es recomendable que las personas que usan "inestable" se suscriban a la lista de correo debian-devel-announce para recibir notificaciones de los cambios importantes, por ejemplo sobre actualizaciones que pueden fallar.
- La distribución "inestable" siempre se llama "sid".

Hay muchas razones por las que Debian GNU/Linux es distinta al resto de distribuciones Linux del mercado. Es distinta en primer lugar por contar con tres documentos que determinan su evolución: su Contrato Social, su Constitución y su guía de desarrollo de Software Libre.

En esos documentos se establecen los principios de una distribución que "se mantendrá 100% libre" –tuvieron problemas con proyectos como Firefox por esa filosofía– y que "no esconderá los problemas", mientras que en su Constitución se define el funcionamiento interno de una comunidad en el que las decisiones las toman diversos tipos de miembros, tales como los desarrolladores o los líderes del proyecto.

Esa parte algo más organizativa deja clara la seriedad de un proyecto que siempre ha sido conservador en su aproximación al software: en Debian GNU/Linux (sobre todo en la rama Stable) uno no puede contar habitualmente con lo último de lo último, pero de lo que sí puede estar seguro es de que la distribución es especialmente estable y fiable.

Esa actitud algo más conservadora probablemente acabó provocando que usuarios independientes, grupos de usuarios y empresas y organismos desarrolladores de Software acabaran tomando Debian GNU/Linux como base para sus propias distribuciones derivadas.

Es así como se creó Ubuntu, la que probablemente es la distribución GNU/Linux y que debutó en 2004 para tratar de popularizar el uso de unas distribuciones con fama de complejas para el usuario novel.

De hecho en su logotipo se leyó durante mucho tiempo la frase "Linux for human beings" ("Linux para seres humanos"), y buena parte de su oferta inicial estaba dirigida a lograr facilitar el uso de Linux a todo tipo de usuarios, no a los tradicionales usuarios de Linux que habitualmente contaban con ciertos conocimientos técnicos.

En Distrowatch donde desde hace años se mantiene un seguimiento de la aparición (y desaparición) de distribuciones Linux, se contabilizan en septiembre del 2020 a 391 distribuciones derivadas de Debian GNU/Linux, con más de 121 de ellas aún activas. Debian GNU/Linux mantiene su propio censo al respecto, por supuesto, pero es que además invita a todo el que quiera a crear su propia derivada –de hecho cuentan con unas versiones especiales llamadas Debian Blends– con una serie de guías para lograrlo.

En realidad aquí cuentan no solo las distribuciones derivadas directas, sino también aquellas "derivadas de derivadas", y es que por ejemplo un gran número de distribuciones no parten ya de Debian GNU/Linux, sino directamente de Ubuntu para luego ofrecer ciertas características diferenciales que sus creadores estiman interesantes para cierto nicho de usuarios.

Debian GNU/Linux es Grande por Muchas Cosas muchas han sido las características que han llevado a Debian GNU/Linux donde está ahora, pero sin duda una de ellas es su gestor de paquetes, Advanced Package Tool o APT, un conjunto de comandos que desde 1998 hicieron que la instalación, actualización y eli-minación de nuevos paquetes software fuera tan potente, eficiente y sencilla para los usuarios.

También es legendaria su atención a todo tipo de arquitecturas, descartadas a menudo por grandes desarrolladores comerciales como Apple, Microsoft o Google, pero que tienen en Debian GNU/Linux a su mejor aliada.

De hecho es uno de los proyectos software que más arquitecturas soporta, con versiones tanto oficiales como no oficiales. Aquí tenemos soporte tanto para plataformas muy extendidas (las x86-64 e IA-32 que se utilizan en proce-

sadores de Intel y AMD, así como MIPS y distintas versiones de ARM) como para otras mucho menos populares como DEC Alpha, HP PA RISC, Intel Itanium, Motorola 68k, PowerPC, SPARC o RISC-V.

Debian GNU/Linux en particular (como Linux en general) es el mejor ejemplo de esa ubicuidad y versatilidad de un sistema operativo que permite por ejemplo que sea protagonista en todo ese segmento de mini-pcs orientados al segmento Maker y de la educación que nos ha conquistado con proyectos como las Raspbery Pi o Arduino.

Su presencia es también notable en un segmento radicalmente distinto. De lo "pequeño" que proponen las Raspberry Pi Debian GNU/Linux también es referente en el segmento de los servidores: según W3Techs la presencia de sistemas Unix en servidores Web es del 68,1% frente al 31,9% de servidores Windows.

De esos servidores basados en distintas versiones de Unix y sobre todo -las estadísticas aquí son imprecisas- distribuciones Linux, Debian GNU/Linux es protagonista con un 23,8% del total, sólo por debajo del 35,3% de una Ubuntu que desde hace años se ha ido dando cuenta de que ese segmento era importante para su parte comercial.

Ciclo de Vida de las Versiones Debian GNU/Linux anuncia su nueva versión estable de manera regular. Los usuarios pueden esperar unos 3 años de soporte completo para cada versión, y 2 años de soporte extra «LTS».

Índice de Versiones la siguiente versión de Debian 13 GNU/Linux se llama "trixie" -se espera para mediados o finales del 2026-, Debian 12 "bookworm" -la actual estable-, Debian 11 ("bullseye"), Debian 10 ("buster"), Debian 9 ("stretch"), Debian 8 ("jessie"), Debian 7 ("wheezy"), Debian 6.0 ("squeeze"), Debian 5.0 ("lenny"), Debian 4.0 ("etch"), Debian 3.1 ("sarge"), Debian 3.0 ("woody"), Debian 2.2 ("potato"), Debian 2.1 ("slink"), Debian 2.0 ("hamm").

Cómo Descargar una Distribución Linux La mayoría de las distribuciones Linux están disponibles bajo un formato llamado imagen *.iso* -no hay que pensar que el término imagen se refiere a un gráfico-. Estamos hablando de un archivo informático que utiliza un estándar que permite garantizar que la copia descargada es igual que el original almacenado en el servidor.

Esto es muy importante. Al poder hacer una verificación de integridad podemos confirmar que la distribución Linux fue descargada correctamente y que lo que vamos a instalar es una copia exacta de la publicada por los desarrolladores. Cualquier problema durante la descarga podría terminar en una instalación incompleta o fallida. Además, la verificación permite evitar sustituciones maliciosas.

¿Cómo se Hace la Verificación? todas las distribuciones permiten verificar la integridad de la imagen descargada comprobando su valor de Hash. Cada archivo contiene datos únicos, y cuando se le aplica un cierto algoritmo llamado «función de Hash criptográfico», se obtiene un valor de Hash que sólo es válido para ese archivo en su estado actual. Los algoritmos Hash más populares son MD5 (Compute and Check MD5 Message Digest), SHA (Secure Hash Algorithm) y BLAKE2 y las distribuciones incluyen el resultado correcto en algún lugar cerca del enlace de descarga para que puedas hacer la comprobación.

Generar la suma de verificación con md5sum, shasum y openssl, ejemplos:

```
$ md5sum debian-testing-amd64-netinst.iso
$ shasum debian-testing-amd64-netinst.iso
$ shasum -a256 debian-testing-amd64-netinst.iso
$ shasum -a512 debian-testing-amd64-netinst.iso
$ b2sum debian-testing-amd64-netinst.iso
$ openssl dgst -sha256 debian-testing-amd64-netinst.iso
```

Instalación de Debian GNU/Linux Debian[20] GNU/Linux se puede descargar de múltiples ligas (sin y con Firmware no libre integrado en el *.ISO*), una de ellas es:

https://www.debian.org/distrib/

Réplicas de Debian GNU/Linux en el Mundo En el mundo hay decenas de réplicas de Debian GNU/Linux, para acceder a alguna de ellas (en este apartado supondremos que se trabaja con la versión estable de Debian GNU/Linux), hay que modificar el archivo */etc/apt/sources.list*[21], mediante:

[20] Algunas de las razones para instalar GNU/Linux Debian están detalladas en su página Web https://www.debian.org/intro/why_debian.es.html

[21] En los inicios de cada línea, los *deb* se refiere a paquetes binarios y *deb-src* a paquetes de código fuente, solo necesarios para labores de desarrollo.

#[22] nano /etc/apt/sources.list

o

apt edit-sources

para agregar la réplica más cercana a nosotros (supongamos qu es: ftp.us.debian.org), usamos:

deb http://ftp.us.debian.org/debian/ buster main
deb-src http://ftp.us.debian.org/debian/ buster main

Una vez actualizado el archivo */etc/apt/sources.list,* hay que actualizar las definiciones de paquetes disponibles en Debian GNU/Linux, usando:

apt update
apt upgrade

Una vez que *apt* ha terminado de instalar los paquetes, necesitamos solicitar que borre los archivos descargados para la actualización, usando:

apt clean

Paquetes contrib y non-free Por omisión en Debian GNU/Linux sólo se instalan paquetes libres, pero hay otro tipo de paquetes útiles que no son libres o que tienen licencia distinta a la usada por Debian GNU/Linux, para poder tener acceso a ellos hay que modificar el archivo */etc/apt/sources.list*, mediante:

#[23] nano /etc/apt/sources.list

[22] En el Bourne Shell y sus derivados como BASH el *Prompt* que nos permite escribir los diferentes comandos, generalmente termina con el carácter:

- $ para usuario sin privilegios
- # para el administrador, conocido como *root*

[23] En el Bourne Shell y sus derivados como BASH el *Prompt* que nos permite escribir los diferentes comandos, generalmente termina con el carácter:

- $ para usuario sin privilegios
- # para el administrador, conocido como *root*

o

```
# apt edit-sources
```

en nuestro caso el contenido de */etc/apt/sources.list*, es:

```
deb http://ftp.us.debian.org/debian/ buster \
main contrib non-free
deb-src http://ftp.us.debian.org/debian/ buster \
main contrib non-free



deb http://security.debian.org/debian-security \
buster/updates main contrib non-free
deb-src http://security.debian.org/debian-security \
buster/updates main contrib non-free
```

esto permite tener acceso a paquetes libres (main) y no libres o con licencias distintas a Debian GNU/Linux GPL de Linux (contrib y non-free).

Una vez actualizado el archivo */etc/apt/sources.list,* hay que actualizar las definiciones de paquetes disponibles en Debian GNU/Linux, usando:

```
# apt update
# apt upgrade
```

Una vez que *apt* ha terminado de instalar los paquetes, necesitamos solicitar que borre los archivos descargados para la actualización, usando:

```
# apt clean
```

Ahora el sistema está listo para poder instalar los paquetes que el usuario de la máquina requiere (hay más de 59,000 disponibles en la versión 11).

Conocer la Réplica con Mejor Velocidad de Descarga si necesitamos conocer cuales son las diez réplicas de Debian GNU/Linux que nos de la mejor velocidad de descarga podemos usar *netselect-apt*, se instala mediante:

```
# apt-get install netselect-apt
```

para luego correrlo usando:

```
# netselect-apt
```

el cual generará un archivo *source.list* en el Home del usuario *root* con la configuración óptima además genera una salida indicando las velocidades de acceso.

Rama de Desarrollo de Backports Algunas veces es necesario instalar en la versión estable algún paquete de la versión testing, pero que no rompa la estabilidad del sistema, para ello se desarrolló Backports[24]. Para usarlo, lo primero es agregar en el archivo */etc/apt/sources.list* las líneas:

```
deb http://deb.debian.org/debian/ bullseye-backports main
deb-src http://deb.debian.org/debian/ bullseye-backports main
```

también existen para los paquetes contrib y non-free:

```
deb http://deb.debian.org/debian/ bullseye-backports main \
contrib non-free
deb-src http://deb.debian.org/debian/ bullseye-backports main
\
contrib non-free
```

después de concluir la edición, es necesario ejecutar:

```
# apt update
```

Para instalar algún paquete, usar:

```
# apt -t bullseye-backports install nombreDelPaquete
```

Pese a no ser una opción recomendada, podemos actualizar todos los paquetes, usando:

```
# apt -t bullseye-backports update
# apt -t bullseye-backports upgrade
```

[24] https://backports.debian.org

Rama Experimental Además de la posibilidad de usar Backports, es posible también usar los repositorios de Debian experimental, pero esto debe ser la última opción para la instalación de paquetes, para ello debemos agregar en el archivo */etc/apt/sources.list* las líneas:

```
deb http://deb.debian.org/debian/ experimental main
deb-src http://deb.debian.org/debian/ experimental main
```

también existen para los paquetes contrib y non-free:

```
deb http://deb.debian.org/debian/ experimental main \
contrib non-free
deb-src http://deb.debian.org/debian/ experimental main \
contrib non-free
```

después de concluir la edición, es necesario ejecutar:

```
# apt update
```

Para instalar algún paquete, usar:

```
# apt -t experimental install nombreDelPaquete
```

2.4.1 Paquetes para Diferentes Necesidades

Metapaquetes Son una solución para grupos de personas con necesidades específicas, no solamente proporciona colecciones manejables (metapaquetes) de paquetes de programas específicos, sino que también facilita la instalación y configuración para el propósito convenido. Cubren los intereses de distintos grupos de personas, su objetivo común es simplificar la instalación y administración de ordenadores para una audiencia determinada[25].

Por ejemplo los hay para:

- Ciencia (science-)
- LaTeX (texlive-)
- Astronomía (astro-)

[25] https://www.debian.org/blends/

- Química (debichem-)
- Sistema de Información Geográfica (gis-)
- Medicina (med-)
- Multimedia (multimedia-)
- Educación (education-)
- Junior (junior-)
- Juegos (games-)

para conocer todos los paquetes que inician (por ejemplo textlive), usar:

$ apt search textlive

para saber qué hace cada paquete usar:

$ apt show texlive

para instalar, usar:

apt install texlive

A continuación daremos un listado paquetes[26] agrupados según su uso, muchos de ellos hacen cosas parecidas o se complementan, para saber qué hace cada paquete usar:

$ apt show paquete

para conocer todos los paquetes que inician (por ejemplo textlive), usar:

$ apt search textlive

para instalar, usar:

apt install paquete(s)

[26] https://packages.debian.org/stable/

Drivers para nuestro Hardware

drive

Firmware libre y no libre

firmware-linux firmware-linux-free firmware-linux-nonfree \

firmware-misc-nonfree

Actualización del Microcódigo

amd64-microcode intel-microcode

Linux Vendor Firmware Service (LVFS) un demonio para actualizar Firmware

fwupd

```
# fwupdmgr get-devices
# fwupdmgr refresh
# fwupdmgr refresh –force
# fwupdmgr get-updates
# fwupdmgr update
# fwupdmgr report-history
# fwupdmgr clear-history
```

en caso de haber descargado el archivo .CAB del Firmware del proveedor, lo instalamos con:

```
# fwupdmgr install archivo.cab
```

Manejo de máquinas virtuales QEMU/KVM

qemu-kvm qemu qemu-utils xtightvncviewer

Aplicaciones para conocer el Hardware de un equipo

procinfo lshw hwinfo inxi cpuid hardinfo cpufrequtils i7z \

dmidecode cpufreq-info lspcmcia lsscsi gparted

Aplicaciones para administración

rcconf powertop cpulimit timelimit

Comandos usuales, nueva implementación

dfc gcp ripgrep ack ncdu most dog htop tldr fd jq bat \
exa pydf fd-find zoxide

Sincronización de archivo y directorios

rsync grsync rclone luckybackup syncthing

Descarga de archivos URL

wget uget curl lftp woof httpie youtube-dl
persepolis

Acelerador de descargas de archivos URL

aria2 axel

Intérpretes de órdenes

csh tcsh ksh fish zsh tsch dash dsh busybox

Manejo de archivo sobre CLI

mc ranger vifm nnn lfm wcm ytree

Navegadores de red

chimera2 chromium conkeror dillo edbrowse epiphany-browser \
iceweasel konqueror midori netrik netsurf netsurf-fb arora hv3 \
surf uzbl tfirefox-esr qupzilla netsurf-gtk falkon

Actualizar el navegador por omisión

```
# update-alternatives –config x-www-browser
```

Servidor de SSH

openssh-server
dropbear

Clientes de SSH

openssh-client filezilla gftp mosh

Respaldo y recuperación de datos

timeshift bacula dump

Manejo de documentos e imágenes

science-typesetting texlive-full texlive-science \
texlive-extra-utils texlive-latex-base \
texlive-latex-recommended texlive-publishers \
texlive-bibtex-extra texstudio pandoc texmaker \
inkscape kile lyx gummi texstudio latexila kexi \
latexila texworks libreoffice calligra abiword \
gnumeric

htmldoc converseen djview4 okular gv evince diffpdf \
mupdf pdf-presenter-console evince xpdf okular atril pdfcrack \
qpdf pdfsam pdfshuffler pdfmod pdfposter pdfchain pdf2djvu \
catdoc chktex cxref cxref-doc latex2rtf antiword a2ps \
unoconv bookletimposer qpdfview rst2pdf xchm \
archmage qpdfview-ps-plugin qpdfview qpdfview-djvu-plugin \
kchmviewer pdfarranger zathura zathura-djvu mupdf \
pdftk poppler-utils pdf2svg imagemagick pdfgrep \

ispanish wspanish texlive-lang-spanish myspell-es \
translate-Shell myspell-en-us

Microsoft Fonts:

ttf-mscorefonts-installer fonts-arkpandora

Manejo de archivos DJVU

djview djview4 okular evince zathura zathura-djvu \
qpdfview qpdfview-djvu-plugin

Edición y manejo de archivos Epub

mupdf calibre fbreader sigil

Manejo de notas

rednotebook cherrytree zim focuswriter tomboy basket \
elpa-org feathernotes xpad gnote nvpy
remind

Tomar notas, dibujar y convertir a PDF

xournal cherrytree

Tipografías

^ttf-
^fonts-*
^xfonts-*

para conocer todos los paquetes asociados, usar:

$ apt search ^ttf-

Microsoft Fonts:

ttf-mscorefonts-installer fonts-arkpandora

Tipografía para programadores

fonts-hack-ttf fonts-powerline fonts-ubuntu-console \
fonts-inconsolata fonts-fantasque-sans fonts-firacode

Editores de gráficos

gpaint inkscape imagemagick dia xfig scribus blender \
calibre kdenlive kazam pinta krita pencil2d graphviz \
calligra feh ufraw showfoto rawtherapee mypaint fotoxx \
kolourpaint darktable pencil2d photoflare openshot \
textdraw converseen
gimp gim-data gimp-data-extra gimp-cbmplugs gimp-dcraw \
gimp-dds gimp-gap gimp-gluas gimp-gmic gimp-gutenprint \
gimp-lensfun gimp-texturize gimp-ufraw gimp-plugin-registry

Editores de vídeo blender pitivi shotcut lives lives-plugins

Editores de vídeo no lineales

kdenlive openshot flowblade

Paquetes para generar videos a partir de imágenes

openshot imagination photofilmstrip kdenlive openshot

Paquetes para tomar vídeos o imágenes del escritorio

simplescreenrecorder gtk-recordmydesktop byzanz kazam \
recordmydesktop vlc vokoscreen obs-studio peek ksnip \
screengrab flameshot imagemagick gnome-screenshot \
gtk-vector-screenshot deepin-screenshot kde-spectacle \
grim scrot shutter gimp

Digitalización de imágenes y documentos

gscan2pdf simple-scan skanlite xsane

Multimedia y conversión de formatos de audio y video

vlc amarok mplayer xbmc ffmpeg mpg123 clementine audacity \
libxine2-bin libxine2-ffmpeg libxine2-x libxine2-plugins ffmpeg \
libavcodec-extra gstreamer1.0-fluendo-mp3 vorbis-tools \
gstreamer1.0-plugins-ugly gstreamer1.0-plugins-bad \
gstreamer1.0-pulseaudio kodi pitivi ffmpegthumbnailer \
libvpx5 handbreak lmms handbrake soundconverter \
converseen libav-tools

Reproducir DVDs

libdvd-pkg

Manejo de escáner

gscan2pdf xsane simple-scan eikazo

Reconocimiento Óptico de Caracteres OCR

tesseract-ocr tesseract-ocr-spa tesseract-ocr-eng gocr ocrad

Búsqueda de archivos duplicados

fslint fdupes jdupes findimagedupes duff rdfind hardlink

Utilerias de compactación y descompactación

arj bzip2 clzip cpio dtrx gzip kgb lbzip2 lhasa lrzip \
lzip lziprecover lzop ncompress p7zip-full p7zip-rar pax \
pbzip2 pigz pixz plzip rar rarcrack tarlz unace unar \
unace-nonfree unp unrar unrar-free unzip xz-utils zip \
zpaq zstd zutils
ark file-roller nemo-fileroller xarchiver engrampa
cabextract kgb xdms archivemount

Sistema de Información Geográfica

gis-*

para conocer todos los paquetes asociados, usar:

```
$ apt search ^gis-
```

Respaldo de información

deja-dup grsync timeshift bacua duplicity kopano-backup \
amanda-client amanda-server backupninja luckybackup \
backintime-gnome backintime-kde backintime-qt4 kup-backup \
backup-manager backup2l backupchecker backuppc kbackup \
rsync rdiff-backup rsbackup vbackup zbackup borgbackup \
rsnapshot rear rclone bup restic syncthing

Reproductores de música

cmus moc mpg123 mp3blaster

Sintetizador de Texto a Voz

speak-ng

Gadgets/Widgets

screenlets screenlets-pack-all

2.4.2 Manejadores de Paquetes Multiplataforma

La principal diferencia entre los formatos de paquetes nativos y los formatos de paquetes independientes es que los primeros comparten dependencias con otros programas instalados en el sistema operativo. Es decir que si el programa Y necesita la dependencia 1 y esa dependencia fue instalada por el programa X que también la necesita, esa dependencia no volverá a instalarse.

Los programas empaquetados con formatos independientes incluyen todas las dependencias que necesitan para su funcionamiento. Es decir que

la dependencia 1 se instalará cada vez que se instale un programa que la necesite.

La segunda diferencia es que los formatos de paquetes tradicionales deben ser construidos con las especificaciones de cada distribución. Es por eso que por más que Ubuntu sea una distribución derivada de Debian GNU/Linux, las diferencias son lo suficientemente importantes como para que los repositorios de la primera no puedan ser usados en la segunda.

La tercera diferencia es que cualquier modificación a una dependencia de los paquetes tradicionales puede afectar el funcionamiento de todos los demás que la necesitan. En cambio, las modificaciones a un programa en un formato independiente, no afectará al resto del sistema.

Dependiendo de las particularidades de cada distribución, es posible instalar las aplicaciones en formatos independientes desde un gestor de paquetes y automatizar su actualización con el gestor encargado de las mismas.

Flatpak, Snap, AppImage, seguro que son nombres con los que estás más que familiarizado. Los paquetes universales han irrumpido en el mundo Linux para poder funcionar en cualquier distribución y así quitar el problema de la fragmentación en cuanto a paquetes. Sin embargo, aún no son mayoría, aunque poco a poco va creciendo el número de Software que se empaqueta en estos tipos de paquetes.

¿Qué es Flatpak? Flatpak es un tipo de paquete universal y para la virtualización de aplicaciones para entornos GNU/Linux. Proporciona una sandbox aislada de procesos conocida como Bubblewrap o envoltorio burbuja. En él los usuarios pueden ejecutar las aplicaciones aisladas del resto del sistema, para mayor seguridad.

Lennart Pöttering fue el programador que lo propuso en 2013, y publicó un artículo al respecto un año más tarde para finalmente desarrollar la idea y formar parte del proyecto freedesktop.org., bajo el nombre de xdg-app, que es lo mismo que Flatpak. Y su popularidad desde el lanzamiento fue en aumento, actualmente cuenta con soporte en más de 20 distribuciones de las más populares.

¿Qué es Snap? Mientras que Flatpak tuvo sus orígenes en la comunidad de desarrollo de Fedora/Red Hat, Snap lo tuvo en Canonical, la empresa que desarrolló este tipo de gestión de paquetería tan peculiar. Un tipo de paquete universal que ya aceptan gran cantidad de distribuciones y

apps empaquetadas en él. En este caso, los paquetes se ejecutan dentro de AppArmor, aunque se pueden ejecutar fuera de la sandbox.

Por cierto, hay que reconocer que existen otros paquetes como los AppImage, que cada vez cobra más y más importancia por su sencilla instalación, o mejor dicho, no instalación. Solo descargas y ejecutas el paquete y listo, como una especie de versión portable. Además, en el sitio oficial AppImage Hub podrás encontrar multitud de herramientas empaquetadas en este formato binario. En cuanto a la aseguridad, se pueden ejecutar dentro de la caja de arena o dentro de AppArmor, Bublewrap o Firejail.

En Ubuntu, el Centro de Software permite instalar tanto programas en formatos tradicionales como Snap, dándole preferencia a estos últimos. Por más que existe un plugin que permite que el Centro de Software de GNOME (del cuál se deriva el de Ubuntu) no funciona con esta distribución. En el caso de Ubuntu Studio, es posible activar la opción de usar paquetes Snap mientras que KDE Neon y Manjaro pueden funcionar con ambos formatos.

Las dependencias de los paquetes Flatpak y Snap que nadie habla
En Linux hay muchas maneras de instalar un mismo Software, pero uno de los reclamos que podemos encontrar es que al instalar se incluye Software principal y dependencias en un mismo paquete, lo que les hace funcionar desde un principio, son más limpios y esas cosas, pero esto es una verdad a medias.

Supongamos que no usamos ningún paquete Flatpak y queremos instalar sólo uno porque lo necesitamos. Por ejemplo la aplicación llamada *Immagini* con la que podremos crear *AppImages*, ese tipo de aplicación portable que se puede ejecutar, en teoría, en cualquier distribución Linux si la arquitectura es compatible. *Immagini* tiene un peso aproximado de 22.4 MB, pero para poder instalarla necesitamos... 1,325 MB de espacio. ¿Cómo?

Dependencias compartidas, pero dependencias al fin y al cabo
Por ejemplo, cuando queremos instalar un programa que convierte archivos multimedia a otros formatos, si no lo tenemos ya es probable que nos descargue *FFmpeg* e *ImageMagick*, cada uno con unas cuantas dependencias más. Estas sí son dependencias al uso, pero las que se instalan junto a un paquete Flatpak o snap son lo necesario para que se pueda ejecutar ese programa en nuestra plataforma. Si la aplicación está escrita en GTK o tiene componentes de GNOME, nos instalará la plataforma de GNOME y sus traducciones.

Cuando instalemos otro programa GTK/GNOME, esto ya lo tendremos, por lo que no será necesario y el peso de la aplicación ya será el que vemos en las tiendas de Software. En el caso de los paquetes Snap tenemos un poco lo mismo.

Vigilando las dependencias de los Flatpak y Snap Controlar los paquetes Flatpak que tenemos de más es más sencillo, puesto que hay varios comandos para eliminar lo que no está siendo usado. Los datos y Cache de la aplicación suelen estar almacenados en ~/.var/app, y se pueden eliminar perfectamente a mano porque está dentro de nuestra carpeta personal y sin protección, algo así como lo que hay dentro de .config. Si queremos eliminarlo con el terminal, tendremos que usar este comando:

$ flatpak uninstall –delete-data

Para eliminar las dependencias de un paquete, que para usar el nombre correcto deberíamos decir «runtimes», el comando sería:

$ flatpak uninstall –unused

Si lo que queremos es eliminarlo todo, deberemos escribir:

$ flatpak uninstall –all

El último está diseñado como medio para restablecer todo lo relacionado a Flatpak. Se podrá volver a instalar un paquete Flatpak, pero empezaremos de cero. Es para hacer una limpieza general.

En cuanto a los paquetes snap, no conozco nada parecido. Cuando instalamos una aplicación, ésta aparece dentro de la carpeta snap. Si eliminamos el paquete, su contenido desaparece, pero sus archivos de configuración no, y pueden estar en .config, .caché o en otra carpeta. Los runtimes o dependencias, junto a los paquetes, suelen estar en /var/snap/ o /var/lib/snapd, pero hay que tener cuidado con lo que tocamos aquí. Mi recomendación sería tirar de tienda de Software, y si tiene un apartado para ello, ir a la pestaña de snaps instalados. Si vemos algo que sabemos que no estamos usando, eliminarlo desde ahí.

También podemos escribir snap list, encontrar lo que sepamos que no estamos usando y eliminarlo con snap remove "paquete".

Aunque hay que saber que existen, y en ocasiones al ver lo que puede ocupar una aplicación al instalarla, no todo es malo. Las aplicaciones de Linux pesan muy poco, y eso es gracias a que existe Software y dependencias que se comparten con otros programas. Esto es perfectamente aplicable a los paquetes Flatpak y Snap: si no existieran estas dependencias, cada nuevo paquete que las necesitara debería incluirlas en él mismo, por lo que las aplicaciones podrían ser muy pesadas. Tal y como están las cosas, las únicas pesadas serán las primeras; las siguientes ya no tendrán que descargar nada extra.

Snap es el más nuevo de los formatos independientes ya que su desarrollo comenzó en el 2014. Está pensado no solo para ser usado en distribuciones Linux de escritorio si no también para Internet de las cosas, dispositivos móviles y servidores. Aunque es posible crear tiendas de aplicaciones independientes, en este momento solo existe una operada por Canonical, Snapcraft.

Aunque Snapcraft tiene un surtido de las aplicaciones más populares de código abierto, su fuerte son los programas desarrollados por empresas desarrolladores de software privativo y prestadoras de servicios en la nube.

Para instalar Snap. usamos:

```
# apt install snapd
```

Para buscar un paquete usamos:

```
$ snap find libreoffice
```

Para instalar un paquete usamos:

```
$ snap install <snap_name>
```

Para listar los paquetes instalados usamos

```
$ snap list
```

Para actualizar un paquete previamente instalado usamos:

```
$ snap refresh <snap_name>
```

Para desinstalar un paquete usamos:

```
$ snap remove <snap_name>
```

Flatpak aunque oficialmente Flatpak se lanzó en el 2015, es la continuidad de otro proyecto de formato universal conocido como xdg-app. Este proyecto nació con el objetivo de poder ejecutar aplicaciones en una caja de arena virtual segura, que no requiera privilegios de root ni suponga una amenaza de seguridad para el sistema.

Flatpak está enfocado en las distribuciones de escritorio también utiliza el concepto de tienda de aplicaciones siendo Flathub la más conocida. El punto fuerte es que suele tener las versiones más actualizadas de las principales aplicaciones de código abierto.

Para instalar Flatpak, usamos:

```
# apt install flatpak
```

para activar su integración con Gnome, usamos:

```
# apt install gnome-software-plugin-flatpak
```

Para buscar algún paquete, usamos:

```
$ flatpak search aplicación
```

Para instalar paquetes usamos:

```
$ flatpak install <remotes> <aplicacion>
```

por ejemplo:

```
$ flatpak install flathub org.libreoffice.LibreOffice
```

Algunos desarrolladores tienen su propio repositorio, podemos usarlo mediante:

```
$ flatpak install –from \
https://flathub.org/repo/appstream/com.spotify.Client.flatpakref
```

Si hemos descargado un paquete de la red, podemos instalarlo usando:

```
$ flatpak install <ApplicationID>.flatpakref
```

Supongamos que hemos descargado *net.poedit.Poedit.flatpakref*, lo instalamos con:

```
$ flatpak install net.poedit.Poedit.flatpakref
```

Si al instalar un paquete nos manda un error, para tratar de corregirlo podemos usar:

```
$ flatpak update -v
```

Podemos correr un Flatpak mediante:

```
$ flatpak run <ApplicationID>
```

si hemos instalado spotify, lo podemos correr usando:

```
$ flatpak run com.spotify.Client
```

Para listar los Flatpak instalados usamos:

```
$ flatpak list
```

Para desinstalar un Flatpak, usamos:

```
$ flatpak uninstall <ApplicationID>
```

por ejemplo:

```
$ flatpak uninstall com.spotify.Client
```

Para actualizar todas las aplicaciones Flatpak instaladas usamos:

```
$ flatpak update
```

Podemos remover las aplicaciones Flatpak no usadas mediante:

```
$ flatpak uninstall –unused
```

Appimage es el más antiguo de los formatos de paquetes independientes ya que se lanzó por primera vez en 2004. Fue el primer formato en seguir el paradigma de «Una aplicación-un archivo». Eso significa que cada vez que descargamos un archivo Appimage estamos descargando la aplicación y todo lo que necesita para funcionar. Si queremos usar la aplicación solo debemos darle permisos de ejecución y hacer doble clic sobre el icono que la identifica.

Appimage no utiliza el sistema de tienda de aplicaciones, pero, hay una página web en la que podemos encontrar una lista de todos los títulos disponibles.

Cargo es el gestor de paquetes predeterminado del lenguaje de programación Rust, el cual se usa para descargar dependencias de los paquetes Rust creados para compilar exitosamente los mismos, haciéndolos distribuibles y facilitando su carga a Crátes (crates.io), el registro de paquetes de la comunidad Rust. Para instalarlo usamos:

```
# apt install cargo
```

para auto actualizar el paquete usamos:

```
$ cargo update
```

para instalar algún paquete usamos:

```
$ cargo install paquete
```

para desinstalar un paquete usamos:

```
$ cargo uninstall paquete
```

Alien es un convertidor de paquetes de línea de comandos que convierte entre diferentes formatos de paquetes de Linux como Red Hat *rpm*, Debian *deb*, Stampede *slp*, Slackware *tgz* y Solaris *pkg*, etc. Actualmente, Alien admite los siguientes formatos de paquete:

- Linux Standard Base (LSB)
- LSB-compliant .rpm packages
- .deb

- Stampede (.slp)
- Solaris (.pkg)
- Slackware (.tgz, .txz, .tbz, .tlz)

El programa *alien* viene al rescate cuando un paquete específico o una versión específica de un paquete no está disponible para su distribución de Linux. Podemos convertir fácilmente dicho paquete al formato de paquete preferido utilizando Alien e instalarlo en su sistema.

Alien no es sólo un convertidor de paquetes, también puede instalar los paquetes generados automáticamente después de la conversión de paquetes. Incluso puede tener la opción de convertir los Scripts que deben ejecutarse cuando se instala el paquete (es recomendable examinar las secuencias de comandos detenidamente y comprobar qué hacen estas secuencias de comandos antes de utilizar esta opción).

Para instalar el paquete usamos:

```
# apt install alien
```

La sintaxis general para convertir paquetes de Linux usando Alien de un formato a otro es:

```
$ alien [–to-deb] [–to-rpm] [–to-tgz] [–to-slp] [opciones] archivo [...]
```

Para convertir un paquete *.rpm* en un paquete *.deb*, simplemente ejecutamos alien como usuario root o sudo:

```
# alien –to-deb /path/to/file.rpm
```

Del mismo modo, para convertir el archivo .deb a .rpm, ejecutamos:

```
# alien –to-rpm /path/to/file.deb
```

Aquí está la lista de opciones para convertir paquetes de Linux a diferentes formatos:

- -d, –to-deb - Crea paquetes Debian (este es el predeterminado)
- -r, –to-rpm: crea paquetes rpm

- -l, –to-lsb: crea un paquete LSB
- -t, –to-tgz - Crea paquetes tgz
- –to-slp - Crea paquetes slp
- -p, –to-pkg: crea paquetes pkg de Solaris

2.4.3 Windows en Linux

Uno de los problemas más comunes al pasar de Windows a Linux es no poder usar los programas a los que estamos acostumbrados. Es cierto que cada vez hay más software disponible para Linux, y que los programas más comunes (como Chrome, Spotify o VLC) tienen sus respectivas versiones en este sistema. Sin embargo, hay otros programas que no tienen versión para Linux, como puede ocurrir con Office, o con Photoshop. En ese caso, o bien tendremos que buscar alternativas (que las existen, como LibreOffice y GIMP), o recurrir a una herramienta que nos va a permitir ejecutar cualquier programa o juego de Windows en Linux: Wine.

Wine nació inicialmente como un proyecto que buscaba crear un emulador de Windows. Su acrónimo era inicialmente «WINdows Emulator», aunque viendo su evolución, y la forma de funcionar, este acrónimo fue actualizado por «Wine Is Not an Emulator«. Y es que en realidad no es un emulador, sino que este programa está formado por un cargador de programas binarios junto a un conjunto de herramientas de desarrollo que permiten portar en tiempo real el código de las aplicaciones de Windows a Unix. Además, trae por defecto una gran cantidad de bibliotecas y librerías de manera que no tengamos problemas de dependencias.

Principales Características este programa es capaz de ejecutar sin problemas cualquier programa diseñado para cualquier versión de Windows, desde la 3.x hasta Windows 10. Eso sí, solo es compatible con programas Win32 (tanto de 32 bits como de 64 bits), por lo que no vamos a poder ejecutar las apps UWP de la Microsoft Store, al menos por ahora.

Entre toda la variedad de librerías, bibliotecas y recursos, podemos encontrarnos con prácticamente todas las bibliotecas de interrupciones para programas, lo que permite hacer llamadas INT en tiempo real. De esta manera, los programas no saben que se están ejecutando en un sistema operativo que no es Windows, simplemente se ejecutan. Y lo hacen igual que en

él. Si algún programa, o juego, tiene dependencias especiales (por ejemplo, una DLL concreta) podemos añadirla fácilmente a Wine. Todas las librerías se encuentran dentro del directorio «~/.wine/drive_c/windows/system32», que equivale al directorio System32 de Windows.

Por supuesto, Wine tiene soporte para una gran cantidad de recursos gráficos. Los programas se pueden dibujar tanto en una interfaz gráfica X11 (el escritorio) como desde cualquier terminal X. Es compatible con las tecnologías OpenGL, DirectX y cuenta con soporte total para GDI (y parcial para GDI32). También permite y gestiona varias ventanas a la vez (del mismo programa, o de diferentes) y es compatible con los temas msstyle de Windows.

También es compatible con los controladores de sonido de Windows, y tiene acceso a los puertos del PC, al Winsock TCP/IP y hasta a los escáneres.

¿Qué Programas y Juegos Puedo Ejecutar con Wine? por desgracia, a pesar de tener una gran compatibilidad, Wine no es capaz de ejecutar el 100% de los programas y juegos de Windows en Linux. Y algunos, aunque se pueden ejecutar, no funcionan del todo bien. Para saber si un programa se puede ejecutar, o no, en Linux, podemos buscarlo en la red del proyecto. Allí vamos a encontrar una gran base de datos que nos va a permitir saber si un programa funciona va a funcionar, si no lo hace, o qué tal lo hace.

Además de poder buscar manualmente cualquier programa o juego, también vamos a encontrar una lista con los Top-10 que mejor funcionan. Los juegos «Platino» son los que funcionan de manera idéntica en Windows que en Linux, los «Oro» los que funcionan bien, pero requieren de alguna configuración especial y los «Plata», los que funcionan, pero tienen pequeños problemas. Los programas o juegos «Bronce» o «Basura» son los que no funcionan.

Saca Todo el Partido a Wine con Estos Programas Wine, al final, es la parte más importante para poder usar programas de Windows en Linux. Sin embargo, su configuración, sobre todo para los programas que no tienen una clasificación de platino, puede ser algo tediosa. Por suerte, existen programas que, aunque se basan igualmente en Wine, nos ayudan a configurar cada uno de estos programas de manera automática para que nosotros no tengamos que hacer nada más.

La instalación y configuración de los programas y juegos de Windows para

usarlos en Linux es lo peor. Si no tenemos muchos conocimientos podemos perder mucho tiempo y, además, no conseguiremos que todo funcione del todo bien. Aquí es donde entra en juego *PlayOnLinux*. Este programa, gratuito y de código abierto, busca ayudarnos con la instalación y configuración de los programas y juegos para hacerlos funcionar en este sistema operativo.

PlayOnLinux nos ofrece una completa base de datos de programas con sus correspondientes configuraciones óptimas de manera que nosotros solo tengamos que seleccionar el programa que queremos, cargarle su instalador y dejar que este complete el proceso de puesta en marcha. Nada más. Cuando acabe la instalación ya podremos abrir el programa o juego y empezar a usarlo.

Podemos descargar esta herramienta de forma totalmente gratuita desde su página Web, o desde una termina:

```
# apt install playonlinux
```

Descargar e Instalar Wine hay muchas formas de instalar Wine en Linux. Sus desarrolladores tienen binarios específicos para cada distribución, así como unos completos repositorios desde los que vamos a poder descargar y actualizar el programa desde terminal:

```
# apt install wine
```

WINE no es una aplicación que se inicia por sí sola. Es un backend que se invoca cuando se inicia una aplicación de Windows. Lo más probable es que su primera interacción con WINE ocurra cuando inicie el instalador de una aplicación de Windows.

Ejecutar e Instalar Programas Windows una vez instalado, Wine se ejecutará al hacer doble clic sobre cualquier archivo .EXE. Además, te permitirá instalar programas, como si estuvieras en Windows y pondrá los accesos directos en el menú principal bajo la categoría «Wine».

A pesar de lo que mucha gente cree, Wine sirve no sólo para correr aplicaciones «sencillas» de Windows, sino incluso juegos complejos. Es más, está demostrado que terribles juegazos como Sim 3, Half Life 2, Command & Conquer 3, Star Wars: Jedi Knight, o importantes suites como Microsoft Office funcionan a la perfección.

Bottles Es una aplicación alternativa para la fácil ejecución de Software de Windows sobre GNU/Linux facilitando la gestión de Wine usando una especie de contenedores llamados Botellas, el paquete se puede descargar como .AppImage y FlatPak que permite manejar correctamente la instalación de dependencias y compatibilidad del Software, además mantiene las aplicaciones seguras dentro del contenedor.

2.4.4 macOS en Linux

cumple una función similar a la de Wine con los programas de Windows, solo que no tiene ningún complejo en definirse como un emulador. Lo que hace es actuar como un traductor que permite ejecutar los programas de macOS usando los recursos de Linux. El nombre Darling (Querida) es la primera parte del nombre del núcleo de macOS (Darwin) y las primeras 3 letras de Linux. Supongo que la G final es para construir una palabra fácil de memorizar.

Hay que decir que a los desarrolladores de Darling la cosa les resulta más fácil que a los de Wine. No tienen que hacer ingeniería inversa ni reinventar nada dado que se basan en las partes de Darwin que están bajo licencias abiertas. El propio Darling se distribuye bajo la licencia GPL.

Iniciando Darling el programa no tiene interfaz gráfica. Lo ponemos en marcha desde la terminal con el comando:

```
$ darling shell
```

Al escribirlo, Darling creará un directorio raíz virtual o se conectará con uno existente. Además cargará los módulos del núcleo y construirá el sistema de archivos virtual donde ejecutaremos los programas.

Desde la línea de comandos podemos acceder a dos tipos de sistemas de archivos: el tradicional de macOS que incluye los directorios de nivel superior como */Applications*, */Users* y */System* entre otros. Por otro lado, el del sistema operativo anfitrión lo hallamos en una partición denominada */Volumes/SystemRoot*

Podemos verificar el núcleo con el siguiente comando:

```
$ uname
```

y averiguar la versión de macOS con:

```
$ sw_vers
```

salimos de la terminal con

```
$ exit
```

y apagamos el contenedor con:

```
$ darling shutdown
```

Instalación de Programas Si estás usando Linux en arranque dual con macOS y quieres ejecutar alguno de los programas que tienes instalado en la partición de Mac, puedes hacerlo con el comando:

```
$ /Volumes/SystemRoot/run/media/usuario/Macintosh HD
/Applications/nombre_app.app)
```

Muchos programas para macOS se distribuyen en formato .dmg. Para instalarlos en Darling hacemos:

```
Darling [~]$ hdiutil attach Downloads/aplicación.dmg /Vol-
umes/aplicacion
Darling [~]$ cp -r /Volumes/aplicación/aplicación.app /Ap-
plications/
```

En el caso de aplicaciones almacenadas en archivos comprimidos, lo descomprimimos y copiamos en la carpeta */Applications*. Lo mismo con aplicaciones previamente descargadas de la tienda de aplicaciones.

Por último nos quedan las aplicaciones *.pkg*, el formato de paquete nativo de macOS. Este formato implica ejecutar Scripts durante la instalación. Para poder usarlos debemos hacer:

```
Darling [~]$ installer -pkg aplicación.pkg -target /
```

Podemos desinstalar los programas con:

```
Darling [~]$ uninstaller nombre_del_paquete
```

Debemos entender que si bien Darling funciona muy bien con aplicaciones para la línea de comandos, solo tiene funcionalidades muy limitadas para las que necesitan una interfaz gráfica.

Instalación de Darling Si utilizas Debian o derivados, la instalación de Darling no tiene mayor problema. Solo tienes que escribir los comandos:

```
# apt install gdebi
# gdebi darling-dkms_X.X.X.testing_amd64.deb
# gdebi darling_X.X.X.testing_amd64.deb
```

reemplaza las X por el número de versión de los paquetes que descargarás o bien se puede descargar los archivos fuentes del proyecto para su compilación e instalación.

2.4.5 Debian GNU/Linux User Repository

En el 2021 se lanzó el repositorio DUR (Debian User Repository) que se posiciona como un análogo del repositorio AUR (Arch User Repository) para GNU/Linux Debian, permitiendo a los desarrolladores de terceros distribuir sus paquetes sin incluirlos en los principales repositorios de la distribución. Al igual que con el AUR, los metadatos y las instrucciones de construcción del paquete en el DUR se definen utilizando el formato PKGBUILD.

2.4.6 Debian Janitor Paquetes Desde Repositorio Git

Debian Janitor es un sistema automatizado que permite tomar fuentes de repositorios Git y generar paquetes Debian. El objetivo general es reducir el esfuerzo manual que se necesita para mantener un paquete, lo que permite a los usuarios probar versiones posteriores más nuevas de los paquetes sin la participación del responsable de mantenimiento.

3 Procesamiento de Textos Científicos

Existe una gran cantidad de usos para los programas de edición de texto, pero en las carreras de Ciencias e Ingenierías, la edición de textos con tipografía científica es común. Es por ello que la gran mayoría de los procesadores de textos más usados no proporcionan las herramientas necesarias para incluir en el texto fórmulas y/o notación matemática. En caso de proveer dichas herramientas, muchas de ellas son de uso tedioso, pues están diseñadas para uso ocasional.

Para subsanar este hecho, existen herramientas y editores hechos a ex profeso, para permitir la edición de textos científicos en los cuales numerar ecuaciones, usar tipografía matemática, manipular bibliografía y referencias cruzadas es una tarea sencilla de realizar.

Existe una gran variedad de paquetes para la edición de textos científicos -los cuales existen tanto en las plataformas de Windows, Linux, Mac-, entre los que destacan:

- Editor de ecuaciones integrado en Word en Microsoft Office (véase [15])
- MathType para Word en Microsoft Office para Windows (véase [20])
- Scientific WorkPlace LaTeXpara Windows (véase [21])
- Gummi LaTeX(véase [22])
- Kile LaTeX(véase [23])
- LED LaTeX(véase [24])
- LyX LaTeX(véase [25])
- Texmaker LaTeX(véase [26])
- TeXnicCenter LaTeX(véase [27])
- TextPad LaTeX(véase [28])
- TeXstudio LaTeX(véase [29])
- WinEdt LaTeX(véase [30])
- Formula de Libre Office (véase [17])

- Math de OpenOffice (véase [16])
- Formula de Calligra (véase [18])

Salvo para los productos de Microsoft Office, el resto de los paquetes tienen una curva de aprendizaje de media a alta, pero en contraste permiten desarrollar textos y gráficos con tipografía científica de alta calidad.

Instalación de Procesadores de Texto e IDEs Existen diversas versiones de paquetes para procesar texto en Linux, para instalar las más comunes en Debian GNU/Linux es necesario hacer:

```
# apt install science-typesetting texlive-science texstudio \
pandoc texmaker inkscape kile gummi texstudio \
texlive-latex-basetexlive-latex-recommended latexila lyx \
texlive-extra-utils texlive-full latexila libreoffice calligra \
abiword evince gnumeric kexi xpdf okular medit texworks \
libreoffice imagemagick djview4 okular gv zathura diffpdf \
mupdf pdf-presenter-console evince atril pdfcrack qpdf \
pdfsam pdfshuffler pdfmod pdfposter pdfchain pdf2djvu \
catdoc chktex cxref cxref-doc latex2rtf antiword \
unoconv a2ps rst2pdf xchm archmage htmldoc \
qpdfview-ps-plugin qpdfview kchmviewer pdfgrep \
qpdfview-djvu-plugin pdfarranger ispanish \
wspanish myspell-es myspell-en-us texlive-lang-spanish \
translate-Shell pdftk poppler-utils pdf2svg bookletimposer \
converseen
```

También podemos instalar más de 400 distintos tipos de Fonts:

```
# apt install ^fonts-* ^ttf-* ^xfonts-*
```

y los Fonts de Microsoft:

```
# apt install ttf-mscorefonts-installer fonts-arkpandora
```

para verificar las fuentes tipográficas instaladas (el paquete es fontconfig y viene instalado por omisión) en la máquina usamos:

```
$ fc-list
$ fc-list : family
$ fc-list : family style
```

Aprender a Trabajar en LaTeX En la red existen múltiples sitios especializados y una amplia bibliografía para aprender a programar cada uno de los distintos aspectos de LaTeX, nosotros hemos seleccionado diversos textos que ponemos a su disposición en:

Latex

3.1 Trabajando con LaTeX

LaTeX es un poderoso paquete que puede hacer múltiples operaciones por nosotros, a continuación haremos un viaje por sus capacidades, pero sólo es una muestra ellas, para más información revise el manual y ejemplos en línea. Supongamos que necesitamos hacer el siguiente texto:

Pequeño Gran Teorema

Alrededor de 1630 el jurista y matemático Pier de Fermat estudió la Aritmética de Diofanto, en una traducción latina de Claude Bachet (1581-1638), publicada en 1621. El problema ocho del libro II de la Aritmética propone descomponer un número al cuadrado en suma de dos números al cuadrado y Fermat anotó en el margen la imposibilidad de hacer algo similar para los cubos, las cuartas potencias, o cualquier potencia más alta, afirmando tener una prueba notable de este hecho que no poda escribir en los márgenes estrechos del libro (Hanc marginis exiguitas non caperet).

Así, esta conjetura nació y en los siglos que se sucedieron tras su muerte, otros matemáticos fueron abordando y solucionando cada uno de los problemas que Fermat había garabateado... Hasta que sólo quedó uno por resolver:

"No existe ningún número entero positivo mayor que 2 que satisfaga la ecuación $a^n + b^n = c^n$, donde n es dicho número"

Esta conjetura fue bautizada con el nombre de "el último teorema de Fermat", precisamente porque era la última proposición de este autor que nadie había podido refutar o verificar.

Pero el pasado 15 de marzo de 1994 se entregó el premio Abel, el premio Nobel de matemáticas, a Andrew Wiles por haber confirmado una conjetura matemática cuya validez no había podido ser demostrada desde que se propuso en 1642.

Figura 1: Ejemplo a desarrollar en LaTeX.

Ahora tenemos que tomar la decisión de con que haremos la edición del texto, en Linux hay varias opciones que podemos usar, la primera es usar un

editor de texto plano[27] en la terminal de línea de comandos como *nano*, *vi o micro*; otra opción es usar un editor gráfico de texto plano[28] como *scite* o *kate*; o los editores especializados para LaTeX [29] como *Gummi* o *Texmaker*.

Entonces es necesario teclear el siguiente texto que lo generará:

```
\documentclass[letterpaper,12pt]{article}

\pagestyle{empty}

\begin{document}

\section*{Peque\~no Gran Teorema}

Alrededor de 1630 el jurista y matem\'atico Pier de Fermat estudi\'o
la Arit-m\'etica de Diofanto, en una traducci\'on latina de Claude
Bachet (1581-1638), publicada en 1621. El problema ocho del libro II
de la Aritm\'etica propone descomponer un n\'umero al cuadrado en
suma de dos n\'umeros al cuadrado y Fermat anot\'o en el margen
la imposibilidad de hacer algo similar para los cubos, las cuartas po-
tencias, o cualquier potencia m\'as alta, afirmando tener una prueba
notable de este hecho que no pod\´ia escribir en los m\'argenes estre-
chos del libro (Hanc marginis exiguitas non caperet).

As\'i, esta conjetura naci\'o y en los siglos que se sucedieron tras su
muerte, otros matem\'aticos fueron abordando y solucionando cada
uno de los problemas que Fermat hab\'ia garabateado... Hasta que
s\'olo qued\'o uno por resolver:

"No existe ning\'un n\'umero entero positivo mayor que 2 que sat-
isfaga la ecuaci\'on $a^{n}+b^{n}=c^{n}$, donde $n$ es dicho n\'umero"

Esta conjetura fue bautizada con el nombre de "el \'ultimo teorema
de Fermat", precisamente porque era la \'ultima proposici\'on de este
autor que nadie hab\'ia podido refutar o verificar.
```

[27] Otras opciones son: Diakonos, Jet, JOE, LE, Mined, Nano, Nice Editor, Pico, SETEdit, Vim, FTE

[28] Otras opciones son: Gedit, JEdit, Nedit, Medit, Kscope, Editra, Kate, Kwrite, Leafpad, Mousepad, Anjunta, Tea, Pluma, GVim, Emacs, Atom, Geany, Glade, Notepadqq, Scribes, Sublime Text

[29] Otras opciones son: Scientific WorkPlace, Kile, LED, LyX, TeXnicCenter, TextPad, TeXstudio, WinEdt, Formula de Libre Office, Math de OpenOffice, Formula de Calligra

```
Pero el pasado 15 de marzo de 1994 se entreg\'o el premio Abel, el
premio Nobel de matem\'aticas, a Andrew Wiles por haber confir-
mado una conjetura matem\'atica cuya validez no hab\'ia podido ser
demostrada desde que se propuso en 1642.

\end{document}
```

3.2 Compilando un Documento

Ahora hecho el documento, con nombre *teorema.tex* necesitamos compilar y generar el archivo de salida que usaremos para visualizar el resultado. Primero debemos tener todo lo necesario para ello, en Debian GNU/Linux se instala LaTeX mediante:

```
# apt install science-typesetting texlive-science
```

Si usamos editores para LaTeX especializados, ellos generan la salida gráfica en formato PDF[30] o DVI[31], sin requerir otra cosa que solicitarla. Si trabajamos en la terminal, necesitamos compilar y generar el archivo de visualización, para ello tenemos algunas opciones:

- Si la salida la necesitamos en DVI, podemos compilar usando:

  ```
  $ latex teorema.tex
  ```

  y visualizamos mediante:

  ```
  $ xdvi teorema.dvi
  ```

- Si la salida la necesitamos en PDF, podemos compilar usando:

  ```
  $ texi2pdf teorema.tex
  ```

  y visualizamos mediante:

[30] El formato de documento portatil (Portable Document Format PDF) es un formato de almacenamiento de documentos digitales independientes de plataformas de Software o Hardware, este formato es de tipo compuesto (imagen vectorial, mapa de bits y texto).

[31] EL formato DVI (DeVice Independent) es un formato de archivo utilizado como salida por el programa de tipografía TeX. Por lo general el fichero DVI es utilizado como entrada por un postprocesador para generar archivos PostScript o PDF.

xpdf teorema.pdf

El archivo *teorema.tex* y su salida *teorema.pdf* se puede descargar de:

Ejemplitos

Por último, si necesitamos trabajar con LaTeX y tenemos acceso a un equipo de cómputo al que no podemos instalar paquetería, existen diferentes servicios Web que permiten editar, compilar y ejecutar código LaTeX, esto en aras de que el usuario cuente con algún sistema de acceso a red y un navegador pueda trabajar sin hacer instalación alguna en su equipo de cómputo, tableta o teléfono celular. Por ejemplo, en la dirección: https://es.sharelatex.com/

3.3 Estructura del Documento

Un documento en LaTeX tiene dos partes principales: el preámbulo y el cuerpo del documento.

El preámbulo es iniciado por la instrucción *documentclass*, mientras que el cuerpo del documento esta delimitado por los comandos *begin{document}* y *end{document}*.

El esqueleto vacío de un documento en LaTeX se ve así:

```
\documentclass{article}
% pre'ambulo
\begin{document}
% cuerpo del documento
\end{document}
```

Comandos Como puedes empezar a observar, los comandos en LaTeX inician con una diagonal invertida: *comando*, mientras que los comentarios (texto que no aparecerá en el documento final y sólo sirve para agregar notas dentro del código) se escriben después de un signo de porcentaje: *% comentario*. Algunos comandos tienen parámetros obligatorios que se escriben entre llaves *{..}*. Algunos otros llevan también parámetros opcionales que van entre corchetes *[..]*.

Por ejemplo el parámetro *article* en *documentclass* indica a LaTeX que el documento se trata de un artículo y utilizará entonces el formato adecuado.

Otras opciones pueden ser *book, report, letter y slides* que sirven para hacer libros, reportes, cartas, y diapositivas respectivamente.

Algunos parámetros opcionales para *documentclass* son *11pt* y *12pt* que especifican un tipo de letra más grande (el normal es de *10pt*), *twocolumn* que escribe el texto a dos columnas y *twoside* que ajusta los márgenes del documento para imprimir a dos caras. Por ejemplo, para escribir un reporte con letra tamaño 12pt y a dos columnas entonces se escribe el comando

```
\documentclass[12pt,twocolumn]{report}
```

Preámbulo En el preámbulo se pueden incluir instrucciones para activar paquetes que agregan funciones adicionales a LaTeX, así como datos generales sobre el documento que estas escribiendo. Un preámbulo típico podría verse así:

```
\documentclass{article}
\usepackage{lmodern}
\usepackage[T1]{fontenc}
\usepackage[spanish,activeacute]{babel}
\usepackage{mathtools}
\title{Ejemplo de \LaTeX{}}
\author{Nombre del autor}
\date{29 de diciembre de 2019}
```

Los dos primeros paquetes, *lmodern* y *fontenc*, se utilizan para mejorar el soporte de caracteres especiales en la fuente (tipo de letra) que se usará en tu documento. Por ejemplo para que puedas copiar y pegar texto correctamente desde el documento PDF que produzcas al final.

El siguiente paquete incluido es *babel* con la opción *spanish* que traduce algunas de las etiquetas usadas por LaTeX, y agrega opciones especiales para redactar documentos en español. Si no incluyes este paquete, o cambias *spanish* por *english*, LaTeX supondrá que estas escribiendo en inglés.

El último paquete incluido es *mathtools* que agrega algunos comandos y funciones especiales para facilitar la escritura de fórmulas y ecuaciones matemáticas.

Hay muchos otros paquetes que puedes incluir y que agregan funciones adicionales a tu documento, pero estos son los básicos que siempre es una buena idea incluir. Algunos otros paquetes típicos son: *hyperref*, que permite

incluir ligas en tu documento, *biblatex*, para administrar tu bibliografía, o *tikz*, para crear todo tipo de ilustraciones.

Finalmente los campos *title,* *author* y *date* especifican los datos que irán en el encabezado del documento. Normalmente, de hecho, no es necesario incluir el comando *date* pues LaTeX usará en su lugar la fecha actual cuando generes tu documento.

Cuerpo del Documento En el cuerpo del documento es donde escribes todo el texto que quieras que aparezca en el documento final. Usualmente se inicia con el comando *maketitle* que se encarga de escribir los datos del título con la información que indicaste en el preámbulo.

Todo el texto normal se escribe tal cual[32]. Si quieres decir "Hola" simplemente escribe: Hola.

Acentos y Signos Especiales Como podrás ver en los ejemplos anteriores, los acentos no se pueden escribir de manera directa dentro el código. Cuando necesites escribir una letra con acento como la "á" deberás escribir 'a y en lugar de "ñ" escribe 'n. Para las mayúsculas funciona lo mismo sólo utiliza la letra mayúscula adecuada.

La opción *activeacute* de *babel* te permite usar este método "corto" para escribir acentos. Esta opción, sin embargo, no funciona en el preámbulo. Para poner acentos en el preámbulo debes usar la forma larga en la que escribes \'*a* para obtener la letra acentuada "á", \'*e* para la letra "é", y \~*n* para la "ñ".

Otros símbolos que requieren atención son: ¡ y ¿ para producir "¡" y "¿", así como las comillas ‘sencillas’ y “dobles” que producen 'sencillas' y "dobles".

Fórmulas Matemáticas La primera forma de escribir formulas matemáticas es el modo en línea que inserta un símbolo o una fórmula sencilla dentro

[32] a) Si dejas varios espacios en blanco entre palabras, LaTeX los toma como si fueran uno solo.

b) No es necesario dejar espacios al inicio de un párrafo para indicar una sangría, LaTeX ignora estos espacios y ajusta las sangrías adecuadas de manera automática.

c) Para separar dos párrafos simplemente deja una línea en blanco entre un párrafo y el siguiente, el simple fin de línea no hace la separación.

d) Varias líneas en blanco juntas valen lo mismo que una sola.

de la redacción de un párrafo. Este modo se obtiene encerrando entre los signos: *$..$* el contenido matemático, un ejemplo[33]:

```
... si $x = 0$ entonces $y^{2} = 4p + 7$, pero si damos otro
valor a $x$ no s'e que pase ...
```

El otro modo para insertar texto matemático es en una fórmula destacada. Este modo es para ecuaciones más grandes que, por ejemplo si incluye sumatorias o límites, no se verían bien incrustadas dentro de un párrafo. Una fórmula destacada lo que hace es abrir un espacio amplio en medio del párrafo y centrar la ecuación en la página[34]. Una forma de lograr esto es usando los comandos *\begin{equation}* y *\end{equation}* o, si no te interesa ir numerando las ecuaciones, la variante *equation**.

```
Y despu'es de experimentar mucho con diferentes t'ecnicas
resulta que la ecuaci'on
\begin{equation}
w = \sum_{i=1}^{n} (x_{i}+y_{i})^{2}
\end{equation}
es muy importante.
... y como sabemos que
\begin{equation*}
\lim_{x \to 0} (x^{2} + 2x + 4) = 4
\end{equation*}
se concluye que...
```

Una gran familia de comandos que puedes utilizar son las letras griegas. Así como *\pi* puedes encontrar *\alpha, \lambda*, etc. Para obtener las letras griegas mayúsculas capitaliza la primera letra, por ejemplo en *\Omega* o *\Pi*.

Otra familia de comandos corresponden a nombres de funciones[35] como *\sin, \log, \lim*, etc. La guía completa de todos los símbolos que puedes

[33] Una de las primeras cosas que notarás es que las letras en el entorno matemático aparecen en itálicas y que puedes escribir exponentes cómo en *y^{2}*. Ojo, sin embargo, que nunca debes de usar el entorno matemático para escribir palabras en cursivas.

[34] Nota que, en el código de LaTeX, no hay separación entre la ecuación y el texto del párrafo. Esto es porque que la ecuación es parte de la redacción del párrafo.

[35] Observa que no se obtiene el resultado correcto en la tipografía si escribes únicamente *sin*; eso es s por i por n, y no la función 'seno' que obtienes con *$\sin$*.

utilizar en LaTeX es un libro que se llama The Comprehensive LaTeX Symbol List de Scott Pakin. Algunos editores, como TeXnicCenter para Windows, tienen barras con botones para escribir los comandos dando Click sobre el símbolo o construcción que necesites.

Estructura del Documento Parte de la ideología de LaTeX es que el autor de los documentos no debe preocuparse por el formato o la apariencia que tendrá el documento impreso en papel, ya que eso es tarea de LaTeX (o de un diseñador de formatos). El autor debe preocuparse sólo por el contenido y la estructura de su documento. Siguiendo esta ideología, esta guía no muestra comandos para manipular el formato del texto. Si en algún lugar ya aprendiste esos comandos lo mejor es que (cuando escribes el cuerpo de un documento) te olvides de que existen.

Por ejemplo, un comando importante es *\emph{..}* que te permite agregar énfasis a palabras u oraciones. Normalmente el resultado es que el texto aparece en itálicas. Sin embargo, no debes pensar en *\emph{..}* como un comando para poner itálicas (¡eso es pensar en formato!) sino como un comando para agregar énfasis (¡eso es pensar en contenido!). Diferentes estilos de documentos podrían incluso agregar énfasis usando diferentes formatos, por ejemplo subrayando o escribiendo en rojo. Un autor decide qué enfatizar, y es tarea del editor el decidir cómo hacerlo.

Otra familia importante de comandos te permite poner títulos y dividir tu documento en secciones. Una de las ventajas importantes de estos comandos es que cosas como el índice y tablas de contenido se hacen de forma automática al utilizarlos:

```
\part{..}
\chapter{..}
\section{..}
\subsection{..}
\subsubsection{..}
```

Los comandos pueden variar según el estilo de documento usado. Por ejemplo un artículo (*article*) suele dividirse comenzando por *\section{..}*, mientras que un libro (*book*) puede incluir *\part{..}* o *\chapter{..}*.

Para conseguir que aparezca el índice en tu documento usa el comando *\tableofcontents*, por ejemplo después de *\maketitle*, y compila dos o tres veces.

Insertar Figuras en LaTeX suele ser una de las principales causas de dolores de cabeza para quienes nos enfrentamos a esta tarea por primera vez. Y la situación se complica con la diversidad e incompatibilidad que existe entre formatos para almacenar gráficos. Además, pareciera que LaTeX no es muy amigable con formatos tipo Web (JPEG, GIF) a los que podríamos estar más acostumbrados.

Antes de comenzar, y para evitar posibles confusiones, es necesario ponernos de acuerdo con los términos, y lo que significan:

- Un gráfico es cualquier dibujo, ilustración, imagen, diagrama, fotografía, gráfica de puntos o líneas, histograma, diagrama de sectores, etc.; que podrías querer insertar en tu documento para ilustrar o clarificar alguna idea. La mayor parte de esta guía trata sobre cómo preparar los gráficos para incluirlos en tu documento.

- Una figura es la forma que normalmente se utiliza para insertar un gráfico dentro de un documento. Las figuras están compuestas por un gráfico y un título (que no es parte del gráfico), así como de una numeración que indica la secuencia de figuras dentro del documento (Figura 1, Figura 2, ...).

Ya que tienes listo el gráfico en un formato adecuado, agrega en el preámbulo de tu documento principal (antes de *\begin{document}*) la instrucción:

```
\usepackage{graphicx}
```

Este paquete, permite incluir gráficos externos en tu documento. Luego, en algún lugar cercano en donde quieras que se coloque tu figura agrega el siguiente código:

```
\begin{figure}
\centering
\includegraphics{grafico}
\caption{Mi Figura}
\label{fig:ejemplo}
\end{figure}
```

El comando *\includegraphics{..}* indica el nombre del archivo que contiene el gráfico que quieres insertar. Observa que no es necesario incluir la extensión del archivo (JPEG, PNG o PDF), LaTeX buscará y utilizará el archivo apropiado.

Tamaño de la figura Si la figura resulta demasiado grande o pequeña, puedes agregar opciones al comando para incluir el gráfico. Por ejemplo:

```
\includegraphics[width=0.7\textwidth]{migrafico}
```

esta opción ajusta el ancho de la figura al 70% del ancho del texto que cabe en una página. Puedes, por supuesto, modificar el valor 0.7 por cualquier según tus necesidades. Hay muchas más opciones que provee el comando \includegraphics, si tienes curiosidad puedes leer la guía Using Imported Graphics in LaTeX2e.

Posición de la figura Una de las tareas de LaTeX es encontrar el lugar más adecuado para colocar tu figura dentro del documento. Esto suele ocasionar sorpresa y un poco de incomodidad sobre todo en usuarios principiantes, "¡pero yo quiero mi figura aquí! ¿por qué LaTeX la pone allá?". La mejor solución realmente es relajarse, cambiar de actitud y dejar que LaTeX haga su trabajo. Evita usar redacciones del tipo:

```
... como en la siguiente figura:
```

y utiliza en su lugar las bondades de LaTeX:

```
... como en la Figura~\ref{fig:ejemplo}.
```

Esto hace que la referencia no dependa del lugar donde aparezca la figura y, finalmente, todo se ve mucho más elegante.

La opción *draft* para borradores Un problema común que nos puede ocurrir es que la gráfica no aparece y, en su lugar, sólo vemos una caja con el nombre del archivo del gráfico. Esto ocurre porque se tiene activada la opción *draft*, ya sea como opción del paquete en *\usepackage[draft]{graphicx}* o como una opción global para todo el documento en \documentclass. Esta opción puede ser útil para visualizar documentos más rápidamente si es que tienes demasiados gráficos. Sin embargo recuerda quitar esta opción, o cambiarla al final, si quieres que aparezcan los gráficos en el documento.

3.4 Hacer Presentaciones

Para hacer presentaciones donde se requiera mos-trar expresiones matemáticas es preferible usar Beamer (véase [31]), el cual es una clase de LaTeX para la creación de presentaciones. Este funciona con pdflatex, dvips, LyX entre otros. Primero debemos tener todo lo necesario para poder trabajar con Beamer, en Debian GNU/Linux se instala LaTeX mediante:

```
# apt install -Rlatex-beamer texlive-full
```

Entonces tecleamos el siguiente texto que lo generará una presentación:

```
\documentclass{beamer}
\mode<presentation> {
 \usetheme{Darmstadt}
 \setbeamercovered{transparent}
}
\usepackage[spanish]{babel}
\usepackage[utf8]{inputenc}
\usepackage{pdfpages}
\usepackage{alltt}
\usepackage{verbatim}
\usepackage{hyperref}
\title{M\'aquinas Virtuales y sus M\'ultiples Usos}
\author[Karla y Antonio]{Karla Gonz\ ´alez y Antonio Carrillo —
{\tt aula@ciencias.unam.mx}}
\institute[FC - UNAM;]
 {Facultad de Ciencias, UNAM}
\date[Intersemestral]
 {Curso intersemestral 2020\\
 11 de enero, 2020}
\AtBeginSection[]
```

```
{
 \begin{frame}<beamer>
 \frametitle{Temas}
 \tableofcontents[currentsection]
 \end{frame}
}
\begin{document}
\begin{frame}
 \titlepage
\end{frame}
\section[¿Qu\'e es]{¿Qu\'e es la virtualizaci\'on?}
\begin{frame}
 \frametitle{¿Qu\'e significa {\it virtualizar} en el c\'omputo?}
 \begin{itemize}
 \item Proveer de algo que no est\'a all\'i, aunque parece estarlo
 \item Ofrecer y mantener una ilusi\'on, un truco de magia
 \end{itemize}
 \begin{center}
 La {\em virtualizaci\'on} es, en t\'erminos generales, es ofrecer
 recursos que no existen en realidad — Y mantener la ilusi\'on, tan
 bien como sea posible.
 \end{center}
\end{frame}
\section[Gracias]{Gracias}
\begin{frame}
 \frametitle{Gracias}
 \begin{center}
 Gracias por su atenci\'on
```

```
\end{center}
\end{frame}
\end{document}
```

Si trabajamos en la terminal, necesitamos compilar y generar el archivo de visualización, para ello tenemos algunas opciones.

Si la salida la necesitamos en DVI, podemos compilar usando:

```
$ latex Presentacion01.tex
```

y visualizamos mediante:

```
$ xdvi Presentacion01.dvi
```

Si la salida la necesitamos en PDF, podemos compilar usando:

```
$ texi2pdf Presentacion01.tex
```

y visualizamos mediante:

```
xpdf Presentacion01.pdf
```

El archivo *Presentacion01.tex* y su salida *Presentacion01.pdf* se puede descargar de:

Ejemplitos

El esqueleto de la presentación Con LaTeX se puede hacer fácilmente una presentación[36] usando la clase *{beamer}*. Cada transparencia se define entre los comandos *\begin{frame}* y *\end{frame}*, por ejemplo:

[36] Unos pequeños consejos sobre presentaciones:

- Una presentación tiene que ser sencilla y clara. Con una (o dos) figuras por transparencia. Con poco texto y letra grande, (4 o 5 líneas es ya suficiente), o mejor, con solo los puntos y palabras clave. Nada de un montón de texto pequeñito y todo apretado, como si fuera un libro. Recuerda que una presentación "es para que el público vea, no para que tú leas".
- Una presentación tiene que ser agradable y elegante. Una presentación en colores es más visible que una en blanco y negro. Pero que se aprecie el buen gusto que tienes.

```
\documentclass{beamer}
\usepackage[spanish]{babel}
\usepackage[latin1]{inputenc}
\begin{document}
\begin{frame}
 \frametitle{Marsupiales}
 Canguro, Koala, Wombat...
\end{frame}
\end{document}
```

Colores y formato LaTeX tiene varios formatos y combinaciones de colores ya definidos. Para elegir un formato, se utilizan los siguientes comandos:

- *\usetheme{Warsaw}* Define el formato.
- *\usecolortheme{crane}* Define la combinación de colores.
- *\useoutertheme{shadow}* Define el encabezado y pie de página. Puedes elegir entre: *{infolines}, {miniframes}, {shadow}, {sidebar}, {smoothbars}, {smoothtree}, {split}, {tree}...*
- \useinnertheme{rectangles} Define el formato de los puntos. Puedes elegir entre: *{circles}, {inmargin}, {rectangles}, {rouded}...*

- También es bueno que una presentación sea corta. Tratar de meter demasiada información, solo consigue que el público se pierda, aburra y desconecte.
- Recuerda que tienes que practicar. "La práctica te da confianza. La confianza te da profesionalidad."
- El tono de voz y los gestos son también muy importantes. Un tono fijo, monótono, constante... es aburrido y hace que la gente se desconecte y se duerma. Los buenos oradores, cambian la fuerza y el tono de voz. Hacen pausas, se mueven, gesticulan. Y consiguen que sus presentaciones sean dinámicos y atrayentes.
- Por último, sonríe, y mira al público. :-)

Ejemplo:

```
\usetheme{Warsaw}
\usecolortheme{crane}
\useoutertheme{shadow}
\useinnertheme{rectangles}
\begin{document}
\title[Animales]{Animales de todo tipo}
\subtitle{Dando nombres a los animales}
\author[Adan, Eva, Serpiente]{
A. Ad\´an$^{1}$ \and E. Eva$^{2}$ \and S. Serpiente$^{3}$$}
\institute[EDEN \& HELL]{
 $^{1-2}$
 Universidad de Ed\´en\\
 Al lado del manzano, Para\´iso
 \and
 $^{3}$
 Universidad del Infierno\\
 Inframundo, 666, Tierra
 \and
 \texttt{\{$^{1}$eva, $^{2}$adan\}@paraiso.com, $^{3}$serpiente@infierno.com}
}
\date{\today}
\begin{document}
\frame{\titlepage}
\end{document}
```

Índice de secciones. LaTeX permite crear fácilmente el índice de nuestra presentación, con el comando *tableofcontents, ejemplo*:

```
\begin{frame}
 \frametitle{\´Indice}
 \tableofcontents
\end{frame}
\section{Animales}
\subsection{Marsupiales}
\begin{frame}
 \frametitle{Marsupiales}
 Canguro, Koala, Wombat...
\end{frame}
\subsection{Marinos}
\section{Plantas}
\subsection{Flores}
\subsection{Árboles}
```

También se puede que, cada vez que pasamos de sección y subsección, nos vuelva a aparecer el índice, marcando el punto por el que nos llevamos. Para ello, bastaría añadir el siguiente código, que hay que poner antes de *begin{document}*, ejemplo:

```
\AtBeginSection{
\begin{frame}
 \frametitle{\´Indice}
 \tableofcontents[currentsection]
\end{frame}
}
\AtBeginSubsection{
\begin{frame}
```

```
 \frametitle{\´Indice}
 \tableofcontents[currentsection,currentsubsection]
\end{frame}
}
\begin{document}
```

Si nuestro índice es muy largo, quizás sea mejor ponerlo en dos columnas. Para ello, necesitamos cargar el paquete *{multicol}*, y utilizar el código siguiente. Además, también recomiendo definir un encabezado que ocupe poco espacio, como *{tree}*, ejemplo:

```
\usepackage{multicol}
\useoutertheme{tree}
\AtBeginSection{
\begin{frame}
 \frametitle{\´Indice}
 \begin{multicols}{2}
 \tableofcontents[currentsection]
 \end{multicols}
\end{frame}
}
\AtBeginSubsection{
\begin{frame}
 \frametitle{\´Indice}
 \begin{multicols}{2}
 \tableofcontents[currentsection,currentsubsection]
 \end{multicols}
\end{frame}
}
\begin{document}
```

```
\begin{frame}
 \frametitle{Índice}
 \begin{multicols}{2}
 \tableofcontents
 \end{multicols}
\end{frame}
```

Cajas Cuando queremos agrupar ideas o palabras clave, es muy útil el comando *{block}*, como en el siguiente ejemplo:

```
\begin{frame}
 \frametitle{Marsupiales}
 \begin{block}{Marsupiales en Australia}
 Koala, Canguro, Wombat...
 \end{block}
 \begin{block}{Marsupiales fuera de Australia}
 Oposum, Zarig\ ¨uella...
 \end{block}
\end{frame}
```

Dos Columnas De manera similar a como hicimos con el índice, podemos dividir una transparencia en varias columnas, utilizando el siguiente código. Nótese que, con el comando *\column{x}*, "x" significa la anchura de cada columna, por ejemplo:

```
\begin{frame}
 \frametitle{Animales}
 \begin{columns}[t]
 \column{0.5\textwidth}
```

```
Algunos mam\ ´iferos marinos:

Ballena, Narval, Cachalote...
\column{0.5\textwidth}
Y algunos más:
Morsa, Le\ ´on marino, Foca...
\end{columns}
\end{frame}
```

Animaciones Si queremos que varios puntos, vayan apareciendo de manera secuencial, según vayamos haciendo Click con el ratón, podemos usar el siguiente código. Donde $<a->$ siginfica que el texto aparecerá desde el Click número $"a"$, hasta el último, por ejemplo:

```
\begin{frame}
\frametitle{Flores}
\begin{itemize}
\item<1->{Rosa}
\item<2->{Azucena}
\item<3->{Margarita}
\end{itemize}
\end{frame}
```

Otra posible animación, consiste en, cuando hagamos Click con el ratón, que un cierto texto cambie de color. Pero sin que aparezca ni desaparezca nada. Para ello, utilizamos el siguiente código:

```
\begin{frame}
\frametitle{Preguntas}
\begin{itemize}
\item \alert<1>{?‘Cuáles ponen huevos?}
```

```
\item \alert<3>{?‘Cuáles tienen veneno?}
\end{itemize}
Armadillo, \alert<2,4>{Ornitorrinco}, \alert<2>{Equidna}, Pangolín, Erizo.
\end{frame}
```

El archivo *Presentacion02.tex* y su salida *Presentacion02.pdf* se puede descargar de:

Ejemplitos

4 Procesamiento de Imágenes, Vídeos y Archivos PDFs

En GNU/Linux contamos con múltiples paquetes para manipular archivos de imágenes, la gran mayoría de ellos trabajan en ambiente gráfico, pero también hay los que trabajan en línea de comandos que permiten procesar una o múltiples imágenes desde la línea de comandos.

Podemos usar al comando *file* para que nos muestre la información básica (tipo de imagen, resolución, etc.) de cualquier archivo de imagen, usando:

```
$ file archivoImagen
```

4.1 Procesamiento de Imágenes y Vídeos

¿Y en caso de borrar accidentalmente un archivo de imagen o vídeo ... lo puedo recuperar?, la respuesta es afirmativa para algunos casos, para ello, instalamos:

```
$ apt install recoverjpeg
```

esta herramienta sólo funciona con archivos JPEG, pero en el paquete se incluye otra herramienta llamada *recovermov*, que se utiliza para recuperar archivos .MOV y trabaja en una gran variedad de sistema de archivos. Para usarla hacemos:

recoverjpeg [ruta]

la ruta puede ser algo como: */dev/sda3*, los archivos recuperados serán dejados en la carpeta donde se ejecuto el comando. Una vez recuperados los archivos, puede haber archivos duplicados, los podemos eliminar usando:

```
$ remove-duplicates
```

Al tomar fotografías o vídeos es recomendable por seguridad desactivar el guardado de datos *Exif* (Exchangeable Image File) también conocidos como metadatos, ya que estos contienen información sobre la cámara, sobre la fotografía y sobre su origen como ubicación por GPS, etc. En GNU/Linux podemos instalar el paquete *ExifTool* que permite conocer y borrar los datos *Exif* en fotografías. Para instalarlo usamos:

```
# apt install libimage-exiftool-perl
```

Para visualizar los datos Exif, usamos:

```
$ exiftool imagen.gif
```

Para borrar todos los datos Exif, usamos:

```
$ exiftool -all=imagen.gif
```

Pngcheck verifica la integridad de los archivos PNG, JNG y MNG (verificando los CRC internos de 32 bits, también conocidos como sumas de comprobación, y descomprimiendo los datos de la imagen), opcionalmente puede volcar casi toda la información a nivel de fragmentos en la imagen legible por humanos. Por ejemplo, se puede utilizar para imprimir las estadísticas básicas sobre una imagen (dimensión, profundidad de bits, etc.); para enumerar la información de color y transparencia en su paleta (suponiendo que renga una), o para extraer las anotaciones de texto incrustadas. Para usarlo, lo instalamos mediante:

```
# apt install pngcheck
```

Para conocer revisar la integridad y datos de una imagen, usamos:

```
$ pngcheck -cvt imagen.png
```

si queremos trabajar con múltiples imágenes simultáneamente, usamos:

```
$ pngcheck -c *.png
```

también podemos usar el programa *parallel* para hacer las revisiones usando todos los procesadores del equipo:

```
$ ls *.png | parallel –nice 19 –bar –will-cite "pngcheck -q {}"
```

Imagemagick algunos consideran a este paquete como la "navaja suiza" para manipular imágenes desde la línea de comandos. Para instalarlo usamos:

```
# apt install imagemagick
```

Entre los múltiples comando que contiene este paquete iniciamos con *conver*, entre sus múltiples opciones es la de reducir el tamaño de las imágenes, la calidad o transformar entre diferentes formatos. La forma más sencilla de usarlo para pasar un archivo de PNG a JPG sin cambiar tamaño ni calidad es:

```
$ convert antiguo.png nuevo.jpg
```

o, usando:

```
$ convert archivo.{svg,png}
```

para juntar dos imágenes en donde ponemos una junto a la otra, usamos:

```
$ convert +append a.png b.png Resultado.png
```

para juntar dos imágenes en donde ponemos una arriba y la otra abajo, usamos:

```
$ convert -append a.png b.png Resultado.png
```

y si queremos reducir el tamaño (alto y ancho) a la mitad, respetando el aspecto:

```
$ convert -scale 50% antiguo.png nuevo.jpg
```

también podemos reducirle el tamaño y también la calidad al 80% (con esto conseguimos un archivo mucho más pequeño):

```
$ convert -scale 50% -quality 80% antiguo.png nuevo.jpg
```

o simplemente podemos pedir que la imagen sea reducida a un determinado porcentaje:

```
$ convert -resize 50% original.jpg modificada.jpg
```

Otra cosa común es querer transformar una imagen al formato PDF, por ejemplo:

```
$ convert foto.jpg archivo.pdf
```

y si son múltiples archivos usamos:

```
$ convert *.jpg archivo.pdf
```

también podemos rotar la imagen (digamos 90 grados) y añadir compresión, usando:

```
$ convert -rotate 90 foto.jpg + compress archivo.pdf
```

Podemos convertir múltiples imagenes al mismo tiempo usado:

```
$ find .  -name "*.jpg" -exec convert \{} -verbose -resize
450x250\> \{} \;
```

Otro comando es *mogrify* que permite cambiar el tamaño de una imagen, desenfocar, recortar, eliminar, difuminar, dibujar, voltear, unir, volver a muestrear y mucho más. Por omisión el comando sobre escribe el archivo de imagen original, mientras convert no. Por ejemplo para modificar el tamaño de la foto al 25%, usamos:

```
$ mogrify -resize 25% archivo.jpeg
```

o podemos indicar el tamaño, por ejemplo 100x67, usando:

```
$ mogrify -resize 100x67 archivo.jpeg
```

podemos cambiar el formato de jpeg a png, usando:

```
$ mogrify -format png archivo.jpeg
```

podemos trabajar con multiples archivos, por ejemplo cambiando todos los archivos *.jpeg a png y reduciendo su tamaño al 50%, usando:

```
$ mogrify -resize 50% -format png *.jpeg
```

para rotar una imagen digamos 180 grados, usamos:

```
$ mogrify -rotate "+180" archivo.jpeg
```

Otro comando más es montage, su uso original es generar tablas de miniaturas de imágenes, es decir, hacer referencia con miniaturas a grandes colecciones de imágenes, especialmente fotos. Y aunque se puede usar para este propósito, también permite hacer mucho más, por ejemplo para hacer Collage de fotos.

Para este ejemplo, suponemos que tenemos cuatro imágenes (sin importar el tipo), si lo que buscamos es crear un montaje básico a partir de estas imágenes, en la terminal solo tendremos que ejecutar:

```
$ montage img1.png img2.png img3.png img4.png salida.png
```

Si todas las imágenes son del mismo tipo, también podemos utilizar el siguiente comando para realizar el montaje con todas las imágenes situadas en el mismo directorio:

```
$ montage *.png salida.png
```

También podemos establecer el tamaño y el espacio entre las imágenes, para ello, la herramienta que nos ocupa cuenta con una opción llamada -*geometry*. Esta nos va a ser de ayuda a la hora de establecer el tamaño de la miniatura y el espacio entre cada imagen. La configuración predeterminada para esto es de 120x120>+4+3. Si en un montaje nos interesa establecer un espacio de 2 píxeles entre las imágenes, el comando a ejecutar sería :

```
$ montage -geometry +2+2 *.png salida.png
```

Esto es útil solo cuando buscamos crear una imagen compuesta a partir de imágenes del mismo tamaño. Cosa que no ocurre con frecuencia, en caso de que nuestras imágenes tengan diferentes tamaños, es posible cambiar el tamaño de todas ellas al mismo tiempo:

```
$ montage -geometry 90x90+2+2 *.png salida.png
```

este comando va a reducir las imágenes dadas para que quepan en un cuadro de 90x90 píxeles de tamaño.

Para crear un montaje con efecto Polaroid con nuestras imágenes solo tendremos que ejecutar:

```
$ montage +polaroid *.png salida.png
```

también podremos dar un efecto Polaroid y hacer que las imágenes se superpongan, utilizando el comando:

```
$ montage -geometry 100x100-10-2 +polaroid *.png salida.png
```

Otra opción disponible será la de *-set label*. Con ella podemos indicarle a la herramienta *montage* que establezca etiquetas para cada imagen en miniatura. Este comando etiquetará las imágenes en miniatura con sus nombres de origen:

```
$ montage -set label '%f' *.png salida.png
```

si te interesa poder establecer una etiqueta personalizada para cada imagen, el comando a utilizar sería algo como:

```
$ montage -label Ejemplo1 img1.png -label Ejemplo2 img2.png
-label Ejemplo3 img3.png -label Ejemplo4 img4.png salida.png
```

además, también se puede establecer un título al montaje que acabamos de realizar. Solo tendremos que añadir la opción -title de la siguiente forma:

```
$ montage -label Ejemplo1 img1.png -label Ejemplo2 img2.png
-label Ejemplo3 img3.png -label Ejemplo4 img4.png -title 'Ejem-
plo para este texto' salida.png
```

Otra característica interesante de la herramienta de *montage* es la posibilidad de concatenar imágenes sin espacios entre ellas, para ello hacemos:

```
$ montage -mode Concatenate *.png salida.png
```

Hay muchas otras opciones de este paquete, solo nos quedamos con lo más básico que ofrece esta herramienta, se pueden consultar todas las opciones disponibles en las páginas de proyecto.

WebP es un formato de imagen de código abierto creado por Google hace ya varios años, para mejorar la gestión de imágenes al hacerlas más ligeras conservando una mejor calidad, con la finalidad de que sean más rápidas su carga y visualización sobre los sitios Web. Es un formato de imagen moderno que proporciona una compresión superior sin pérdidas y con pérdidas para las imágenes en la Web.

Las imágenes sin pérdida de WebP son 26% más pequeñas en tamaño comparadas con las PNG. Las imágenes con pérdida de WebP son 25-34% más pequeñas que las imágenes JPEG comparables con un índice de calidad SSIM equivalente. WebP sin pérdidas soporta transparencia (también conocido como canal alfa) a un costo de sólo 22% de Bytes adicionales. Para los casos en que la compresión RGB con pérdida es aceptable, la WebP con pérdida también soporta la transparencia, típicamente proporcionando tamaños de archivo tres veces más pequeños comparados con PNG. Para instalar el paquete WebP usamos:

```
# apt install webp
```

Y para usarlo escribimos:

```
$ cwebp -q 60 imagen.png -o imagen.webp
```

donde el modificador -q define la calidad de salida y -o especifica el archivo de salida. Otras herramientas del paquete WebP son;

- anim_diff - herramienta para mostrar la diferencia entre imágenes de animación
- anim_dump - herramienta para volcar la diferencia entre imágenes de animación.
- cwebp - herramienta de codificador Webp.
- dwebp - herramienta decodificadora Webp.
- gif2webp - herramienta para convertir imágenes GIF a Webp.
- img2webp - herramientas para convertir una secuencia de imágenes en un archivo Webp animado.
- vwebp - visor de archivos Webp.

- webpinfo - se usa para ver información sobre un archivo de imagen Webp.
- webpmux - herramienta de muxing Webp.

Si tenemos varios archivos y queremos aplicar la misma transformación a todos a la vez, podemos crearnos un bucle. En el siguiente ejemplo vemos cómo ir tomando uno por uno cada archivo PNG y pasándolo a JPG:

```
for img in *.png; do
    filename=${img%.*}
    convert "$filename.png" "$filename.jpg"
done
```

La siguiente orden hace algo similar pero al mismo tiempo va reduciendo la calidad de la imagen de salida:

```
$ find . -name "*png" | xargs -l -i basename -s ".png" "{}" |
xargs -l -i convert -quality 85% "{}.png" "{}.jpg"
```

Por último, dos comandos bastante útiles. El primero nos da información detallada del formato, dimensiones, paleta de color, etc de un archivo de imagen cualquiera:

```
identify imagen.jpg
```

Otras formas de usarlo, por ejemplo convirtiendo de PNG a JPG::

```
$ ls -1 *.png | xargs -n 1 bash -c 'convert "$0" "${0%.png}.jpg"'
```

y de JPG a PNG:

```
$ ls -1 *.jpg | xargs -n 1 bash -c 'convert "$0" "${0%.jpg}.png"'
```

Por otro lado, podemos poner textos a una imagen (para ser usada en un meme), mediante:

```
$ convert imagen.png -font impact -fill white -pointsize 84
-stroke black -strokewidth 3
-gravity north -annotate +0+20 'Texto superior'
-gravity south -annotate +0+20 'Texto inferior' resultado.png
```

Si necesitamos tomar capturas de pantalla, podemos usar el paquete *imagemagick*, mediante el comando *import*, por ejemplo para tomar captura de toda la pantalla, usamos:

```
$ import -window root imagen.png
```

y si solo queremos solo una parte de la pantalla, usamos:

```
$ import imagen.png
```

Otra opción es el programa es *scrot*, para instalarlo usamos:
apt install scrot
Si necesitamos tomar captura de toda la pantalla, usamos:

```
$ scrot imagen.png
```

y si solo queremos solo una parte de la pantalla, usamos:

```
$ scrot -s imagen.png
```

También es posible convertir una imagen a formato *ASCII*, para ello tenemos que instalar el paquete *jp2a*, mediante:

```
# apt install jp2a
```

y para convertir un archivo.*jpg* a *ASCII*, usamos:

```
$ jp2a archivo.jpg
```

o si lo necesitamos, indicamos los archivos a convertir:

```
$ jp2a archivo1.jpg archivo2.jpg archivo3.jpg
```

podemos indicar el tamaño de la salida, usando:

```
$ jp2a –size=50x30 archivo.jpg
```

o podemos imprimir las imágenes en formato con fondo Light/Dark, mediante:

```
$ jp2a –background=light archivo.jpg
$ jp2a –background=dark archivo,jpg
```

o podemos invertir la imagen, usando:

```
$ jp2a archivo.jpg –invert
```

Dado que *jp2a* solo trabaja con archivos de formato *.jpg*, podemos usar *convert* para poder convertir cualquier archivo de imagen, ejemplo:

```
$ convert archivo.png jpg:- | jp2a -
```

GNU Parallel permite a un usuario construir y ejecutar comandos de Shell desde la entrada estándar en paralelo. Por ejemplo convirtiendo de PNG a JPG:

```
$ parallel convert '{}' '{.}.jpg' ::: *.png
```

y de JPG a PNG:

```
$ parallel convert '{}' '{.}.png' ::: *.jpg
```

o mediante, por ejemplo de de PNG a JPG:

```
$ ls -1 *.png | parallel convert '{}' '{.}.jpg'
```

y de JPG a PNG:

```
$ ls -1 *.jpg | parallel convert '{}' '{.}.png'
```

Reconocimiento Óptico de Caracteres (OCR) es la capacidad de mirar y encontrar palabras en una imagen y luego extraerlas como texto editable. Esta sencilla tarea para los humanos es difícil de realizar para las computadoras. Podemos hacer esto desde la línea de comandos de GNU/Linux, una opción es instalar *tesseract-ocr*, mediante:

```
# apt install tesseract-ocr
```

y para usarlo, tomamos cualquier imagen con texto y escribimos:

```
$ tesseract archivo.png archivoTxt
```

podemos indicar el idioma y la resolución para un mejor reconocimiento, mediante:

```
$ tesseract -l eng –dpi 300 archivo.png archivoTxt
```

en caso de tener un archivo *.pdf*, podemos hacer:

```
$ pdftoppm -png turing.pdf turing
$ tesseract -l eng –dpi 300 turing.png turing
```

FFMPEG es una completa solución multi-plataforma para grabar, convertir y hacer Streaming de audio y vídeo a otras fuentes. Aunque a menudo se confunde a FFMPEG con un codec, en realidad es un Framework, un conjunto de Codecs y herramientas con las que lleva a cabo todas las tareas de grabar, convertir y hacer Streaming.

A grandes rasgos, este Framework multimedia está formado por varios componentes:

- ffmpeg: la herramienta para usar el framework desde línea de comandos.
- ffserver: servidor para hacer streaming multimedia.
- ffplay: reproductor multimedia integrado.
- libavcodec: biblioteca con todos los codecs de audio y vídeo de FFMPEG.

- libavformat: biblioteca que contiene los multiplexadores y demultiplexadores.
- libavutil: biblioteca de apoyo.
- libpostproc: biblioteca que se encarga del postproceso de vídeo.
- libswscale: la biblioteca de escalado de vídeo.

El cual se instala usando:

```
# apt install ffmpeg
```

para conocer los Codecs disponibles usamos:

```
$ ffmpeg -codecs
```

y para listar los formatos disponibles usamos:

```
$ ffmpeg -formats
```

Por ejemplo, muestra los datos del video:

```
$ ffmpeg -i output.mp4
```

cuando ejecuta el comando anterior, ffmpeg muestra los datos del video, pero el encabezado dificulta ver esto, para ignorar los datos iniciales, ejecute:

```
$ ffmpeg -i output.mp4 -hide_banner
```

Para conocer la resolución de un video usamos:

```
$ ffprobe -v error -select_streams v:0 -show_entries stream=width,height
-of csv=s=x:p=0 video.mp4
```

Para cambiar la resolución a digamos 1920:1080 usamos:

```
$ ffmpeg -i videoOriginal.mp4 -vf scale=1980:1080 videoFi-
nal.mp4
```

Podemos usarlo para convertir audio usando:

$ ffmpeg -i archivoEntrada.wav salida.mp3

Podemos convertir un video *.av* a *.mp4* usando:

$ ffmpeg -i video.av -code cpy video.mp4

Bajar la resolución de un video (por ejemplo mediante H264 y mp3), usando:

$ ffmpeg -i entrada.mp4 -vcodev h264 -acodec mp3 salida.mp4

Convertir video *.mp4* a *gif* mediante:

$ ffmpeg -i Archivo.mp4 archivo.gif

Remover el audio de un video mediante:

$ ffmpeg -i entrada.mp4 -c copy -an salids.mp4

Averiguar cuántos FPS tiene tu video

$ ffmpeg -i output.mp4 2>&1 | egrep -o '[0-9]+ fps'

Insertar ZOOM en su video

$ fmpeg -i input.mp4 -vf "zoompan=z='if(lte(mod(time,10),10),2,1)':d=1:x=iw/2-(iw/zoom/2):y=ih/2-(ih/zoom/2):fps=30" output.mp4

Optimizando tu video (disminuir el tamaño sin perder calidad de imagen):

$ ffmpeg -i input.mp4 -vcodec libx264 -crf 28 output.mp4

Cambiar el tamaño de la resolución de video (dejándolo en la resolución: 1280x720):

$ ffmpeg -i input.mp4 -vf scale=1280:720 -preset slow -crf 18 output.mp4

Cambiar el tamaño de la anchura y la altura será proporcional (especificado para un ancho de 1280):

```
$ ffmpeg -i input.mp4 -vf scale=1280:-1 output.mp4
```

Cambiar el tamaño de la altura y el ancho será proporcional (especificado para altura de 720):

```
$ ffmpeg -i input.mp4 -vf scale=-1:720 output.mp4
```

Rotar un video (parece la pantalla de un celular, el ancho se convierte en alto y viceversa):

```
$ ffmpeg -i input.mp4 -vf "transpose=clock" output.mp4
```

Rotando 180° (si el video está "al revés" lo invertirás):

```
$ ffmpeg -i input.mp4 -vf "transpose=2,transpose=2" output.mp4
```

Extraer cuadros de un video:

```
$ ffmpeg -y -ss 00:00 -i input.mp4 -t 10 "frames/filename%05d.jpg"
```

Extraer fotogramas solo de los 10 segundos iniciales

```
$ ffmpeg -y -ss 00:00 -i input.mp4 -t 10 "frames/filename%05d.jpg"
```

por ejemplo extraemos 30 imágenes por segundo desde el primer segundo hasta el segundo 6 -es decir, durante 5 segundos- y las guardará siguiendo el siguiente patrón imagen001.jpg, imagen 002.jpg, etc., ejemplo:

```
$ ffmpeg -i video.mp4 -ss 00:00:01 -t 5 - r 30 -f image2 imagen%03d.jpg
```

y si ahora tenemos los fotogramas y queremos formar un video, podemos hacerlo mediante:

```
$ ffmpeg -framerate 10 -i imagen-%03d.jpg video.mp4
```

Agregar subtítulos al video:

```
$ ffmpeg -i input.mp4 -i subtitle.srt -c copy -c:s mov_text outfile.mp4
```

ejemplo del archivo: *subtitle.srt*

```
1
00:00:00,000 –> 00:00:02,827
- Terminal Raíz - Sistemas
Operacional, C++ y Desarrollo.
2
00:00:02,827 –> 00:00:06,383
Ejemplos de diferentes usos
de ffmpeg para ayudarte
3
00:00:06,383 –> 00:00:09,427
No olvides leer también
los enlaces de abajo.
```

Ver un video:

```
$ ffplay video.mp4
```

Escuchar música:

```
$ ffplay music.mp3
```

Se tienen múltiples opciones en el paquete, por ejemplo: para controlar la calidad de audio, control del Bitrate, extraer parte de un audio, unir dos o más audios, recortar un audio, agregar etiquetas en los audios, añadir un portal al audio, manipulación del volumen del audio entre otras cosas.

Descargar Vídeos de youtube En la gran mayoría de las distribuciones se tiene acceso al comando youtube-dl para descargar videos de www.youtube.com, en caso de no ser así, lo podemos instalar usando:

```
# apt install youtube-dl
```

para descargar video o Playlist, usamos:

```
$ youtube-dl [URL]
```

en caso desearle cambiar el nombre usamos:

$ youtube-dl -o 'nombre' [URL]

si se desea descargar múltiples vídeos podemos usar:

$ youtube-dl [URL1] [URL2]
$ youtube-dl -a archivoDireciones.txt

si deseamos descargar un solo video de una Playlist, usamos:

$ youtube-dl –playlist-items [número1, número2, etc] [URL]

si necesitamos descargar el video con un formato especifico, usamos:

$ youtube-dl –format mp4 [URL]

si sólo deseamos descargar el audio de un video, usamos:

$ youtube-dl -x [URL]

por omisión se descarga con formato *Ogg*, pero podemos descargarlo usando *mp3*, mediante:

$ youtube-dl -x –audio-format mp3 [URL]

también podemos descargar los vídeos con su descripción, metadatos, anotaciones, subtítulos y Thumbnail, usando:

$ youtube-dl –write-description –write-info-json –write-annotations –write-sub –write-thumbnails [URL]

Para conocer todos los formatos disponibles de video o Playlist, usamos:

$ youtube-dl –list-formats [URL]

o

$ youtube-dl -F [URL]

para indicar que se descargue con algún formato especifico, usamos:

$ youtube-dl f [númeroFormato] [URL]

y para solicitar que se descargue con la mejor resolución posible, usamos:

$ youtube-dl -f best [URL]

otras opciones son: *worst, bestvideo, worstvideo, bestaudio, worstaudio*
Por omisión, el comando *youtube-dl* reanudará automáticamente la descarga donde la dejamos. Si no deseamos reanudar la descarga, podemos forzar la reanudación de los archivos parcialmente descargados, usando:

$ youtube-dl -c [URL]

4.2 Procesamiento de Archivos PDFs

Existen en GNU/Linux múltiples opciones para manipular archivos *PDF*, entre los que destacan por su poder y versatilidad desde la línea de comandos son:

libreoffice entre las muchas cosas que podemos hacer con libreoffice es tratar de convertir un archivo PDF a uno con formato Odt, Doc o Docx, en todos los casos la conversión dependerá de la complejidad del PDF, para el primer caso usamos:

```
$ libreoffice –infilter="writer_pdf_import" –convert_to odt
archivo.pdf
```

para el segundo caso:

```
$ libreoffice –infilter="writer_pdf_import" –convert_to doc
archivo.pdf
```

para el último caso:

```
$ libreoffice –infilter="writer_pdf_import" –convert_to docx
archivo.pdf
```

Pero también podemos convertir un archivo .doc a imagen:

```
$ soffice –convert-to jpg "archivo.doc"
```

o un archivo .txt a imagen:

```
$ soffice –convert-to jpg "archivo.txt"
```

pdftk es un potente programa hecho con Java que permite eliminar o añadir páginas de un PDF, juntar páginas de distintos documentos PDF en uno solo, todo con sencillos comandos. Útil, rápido y muy potente. Para instalarlo basta invocar:

```
# apt-get install pdftk
```

Entre las distintas opciones de uso del comando esta el poder unir, separar, cifrar, reparar, rotar, entre otras cosas, para mostrar sus capacidades veamos algunos ejemplos:

- Para unir dos documentos diferentes podemos ejecutar lo siguiente:

  ```
  $ pdftk archivo1.pdf archivo2.pdf cat output salida.pdf
  ```

- También podemos unirlos utilizando etiquetas:

  ```
  $ pdftk A=archivo1.pdf B=archivo2.pdf cat A B output salida.pdf
  ```

- Y por supuesto podemos usar comodines:

  ```
  $ pdftk *.pdf cat output salida.pdf
  ```

- Para separar páginas de varios documentos y crear un documento nuevo con estas hacemos lo siguiente:

  ```
  $ pdftk A=uno.pdf B=dos.pdf cat A1-7 B1-5 output salida.pdf
  ```

- Otro ejemplo con un solo documento:

  ```
  $ pdftk A=archivo1.pdf cat A1-12 A14-end output salida.pdf
  ```

- Eliminar las páginas de la 1 a la 4 y de 21 en adelante de un fichero PDF:

  ```
  $ pdftk fichero.pdf cat 1-4 21-end output ficheros_eliminados.pdf
  ```

- Extraer páginas de un fichero PDF (por ejemplo, extraer páginas 3, 4 y 5 en un nuevo PDF):

$ pdftk fichero.pdf cat 3-5 output fichero_final.pdf[/code]

- Adjuntar archivos a páginas de un fichero PDF:

  $ pdftk fichero.pdf attach_files adjunto1 adjunto2 output fichero_final.pdf

- Extraer los adjuntos de un fichero PDF:

  $ pdftk fichero.pdf unpack_files output directorio/de/salida

- Añadir una marca de agua a un fichero PDF:

  $ pdftk fichero.pdf background watermark.pdf output fichero_final.pdf[/code]

- Para cifrar con una clave de 128 bits (opción por defecto) y restringir todos los permisos (opción por defecto):

  $ pdftk archivo.pdf output archivo_cifrado.pdf owner_pw foopass

- Para cifrar igual que el caso anterior pero asignando una contraseña "miclv" que permite abrir el archivo de salida:

  $ pdftk archivo.pdf output archivo_cifrado.pdf owner_pw foo user_pw miclv

- Igual que el caso anterior pero con permiso de impresión:

  $ pdftk archivo.pdf output archivo_cifrado.pdf owner_pw foo user_pw miclv allow printing

- Para descifrar:

  $ pdftk asegurado.pdf input_pw foopass output inseguro.pdf

- Para repara un archivo pdf:

  $ pdftk corrupto.pdf output arreglado.pdf

- Para descomprimir un archivo pdf para su posterior edición en algún editor de texto:

$ pdftk midoc.pdf output midoc_desc.pdf uncompress

- Para separar cada una de las páginas del documento:

  $ pdftk in.pdf burst

- Para generar un reporte del documento (resulta útil cuando se necesita organizar un índice de un conjunto de un PDF):

  $ pdftk archivo.pdf dump_data output reporte.txt

- Multistamp:

  $ pdftk fondo.pdf multistamp stamp.pdf output salida.pdf

- Stamp:

  $ pdftk fondo.pdf stamp stamp.pdf output salida.pdf

- Giro[37] de todo el fichero 90^o:

  $ pdftk archivo.pdf cat 1-endeast output salida.pdf

- También puedes ponerlo como:

  $ pdftk archivo.pdf cat 1-endE output salida.pdf

- Giro de todo el fichero 180^o:

  $ pdftk archivo.pdf cat 1-endsouth output salida.pdf

- Giro de la página 2 90^o:

  $ pdftk archivo.pdf cat 2east output salida.pdf

[37] Otras opciones son: north: 0, east: 90, south: 180, west: 270, left: -90, right: +90, down: +180

poppler-utils la herramienta pdfinfo, parte del paquete "poppler-utils"[38], para instalarlo usamos:

```
# apt install poppler-utils
```

pdfinfo permite volcar por pantalla la información o metadatos de un archivo PDF pasado como parámetro:

```
$ pdfinfo archivo.pdf
```

Podemos combinar archivos en uno solo, para realizar esta acción, los archivos que se van a combinar deben estar en el mismo directorio donde se ejecuta pdfunife. Usando los archivos que he nombrado anteriormente, el comando a utilizar sería el siguiente:

```
$ pdfunite archivo1.pdf archivo2.pdf archivoCombinado
```

Podemos convertir PDFs a Portable Pixmap (.ppm) u otros formatos (PNG, JPEG, JPEGCMYK, JPEGOPT, TIFF) usando el comando *pdftoppm*, por ejemplo el archivo.pdf a imagen-1.png, imagen-2.png, etc., mediante:

```
$ pdftoppm archivo.pdf imagenes -png
```

o podemos indicar las páginas (first y last):

```
$ pdftoppm -f 10 -l 15 archivo.pdf imagenes -png
```

y en caso necesario podemos indicar la resolución:

```
$ pdftoppm -rx 300 -ry 300 archivo.pdf imagenes -png
```

para convertir en escala de gris usamos:

```
$ pdftoppm -gray archivo.pdf imagenes
```

para convertir en monocromo usamos:

[38] Otros comandos del paquete son: pdfattach, pdfdetach, pdffonts, pdfimages, pdfinfo, pdfseparate, pdfsig, pdftocairo, pdftohtml, pdftoppm, pdftops, pdftotext, pdfunite.

```
$ pdftoppm -mono archivo.pdf imagenes
```

También podemos transforma archivos pdf en archivos de texto (txt).

```
$ pdftotext -layout archivo.pdf archivo.txt
```

la opción -layout intenta mantener (en la medida de lo posible) el formato original del texto.

Podemos usar el comando pdftohtml para convierte un *PDF* en formato html (genera su salida en el directorio de trabajo actual), mediente:

```
$ pdftohtml archivo.pdf archivo.html
```

si queremos que el resultado sea más fiel al original podemos usar la opción -c. En tal caso cada página aparecerá como un archivo separado. Si el archivo original tiene muchas páginas, puede ser conveniente crear una carpeta para almacenar el resultado.

```
$ pdftohtml -c archivo.pdf carpeta_creada/archivo.html
```

Podemos convertir archivos *PDF* a archivos de texto simple. Básicamente lo que hace es extraer los datos de texto de los archivos *PDF*. En ella vamos a encontrar muchas opciones disponibles, incluida la capacidad de especificar el rango de páginas para convertir, la posibilidad de mantener el diseño físico original del texto lo mejor posible, establecer finales de línea e incluso trabajar con archivos *PDF* protegidos con una contraseña.

```
$ pdftotext -layout pdf-entrada.pdf pdf-salida.txt
```

si no nos interesa convertir todo el archivo *PDF*, y queremos acotar un rango de páginas del PDF a convertir en texto habrá que utilizar la opción -f (primera página para convertir) y -l (última página para convertir) seguida cada una de las opciones con el número de la página. El comando a utilizar sería algo como lo siguiente:

```
$ pdftotext -layout -f P -l U pdf-entrada.pdf
```

Por otro lado, podemos extraer todas las imágenes de una archivo PDF, y guardarlas como archivos de tipo Portable Pixmap (PPM) Portable Bitmap (PBM) o archivos JPEG. La sintaxis de esta herramienta es:

$ pdfimages archivo.pdf imagen

donde archivo.pdf es el fichero del que quieres extraer las imágenes e imagen será el nombre que tendrá la imagen que se extraiga, en caso de ser varias, se irán nombrando de la forma: imagen-000.(extensión), imagen-001.(extensión), etc. La extensión por defecto será .ppm, si se trata de imágenes en color, o pbm si son en grises. Si queremos que en lugar de estos formatos queremos que lo guarde en jpg, tendremos que utilizar la opción "-j", de esta manera, las imágenes que estén en formato DCT, las extraerá en formato jpeg, y el resto en los formatos anteriores según sea en escala de grises o no:

$ pdfimages -j archivo.pdf imagen

Por otro lado, si no queremos extraer imágenes de todo el documento, sino solo de las páginas de la 8 a la 15, por ejemplo, tendremos que utilizar el siguiente comando:

$ pdfimages -f pagina-inicial -l pagina-final archivo.pdf imagen

También, puede suceder que el documento esté protegido, o bien con algunas restricciones, para lo que tendremos que proporcionar la contraseña de propietario:

$ pdfimages -opw contraseña_de_propietario archivo.pdf imagen

O bien, si el documento está protegido con una contraseña de usuario

$ pdfimages -upw contraseña_de_usuario archivo.pdf imagen

Otra medida de urgencia podría ser la de realizar una conversión intermedia a otro formato y posteriormente volver a regenerar el documento PDF. Para ello, podríamos utilizar los comandos pdf2ps y ps2pdf utilizando el formato intermedio PostScript:

```
$ pdf2ps original.pdf intermedio.ps
$ ps2pdf intermedio.ps optimizado.pdf
```

De la misma forma, también podríamos hacer el mismo proceso utilizando el formato intermedio DJVU y los comandos pdf2djvu y djvu2pdf:

```
$ pdf2djvu original.pdf intermedio.djvu
$ djvu2pdf intermedio.djvu optimizado.pdf
```

Eso sí, ten en cuenta que el formato DJVU está pensado para imágenes escaneadas, por lo que no es apto para todo tipo de documentos PDF. En ambos casos es necesario el intérprete GHostScript.

didjvu si se tiene un archivo DJVU y se necesita convertir a PDF una de las mejores opciones es didjvu, se instala usando:

```
# apt install didjvu
```

este puede hacer la conversión usando:

```
$ didjvu -format=pdf -quality=100 -verbose archivo.djvu archivo.pdf
```

además permite hacer separación de contenido de archivos DJVU de primer y segundo plano.

pdf2svg transforma los archivos pdf en gráficos vectoriales (svg), para instalarlo hacer:

```
# apt intsall pdf2svg
```

y para usarlo:

```
$ pdf2svg archivo.pdf archivo.svg
```

Convierte una página PDF en un archivo SVG. Si tenemos un documento PDF con varias páginas, y queremos transformarlas todas, escribiremos algo como esto:

```
$ pdf2svg archivo.pdf archivo_página%d.svg all
```

Con all le decimos que las transforme todas. Con %d hacemos que las numere.

ImageMagic: convert si existe una herramienta mágica con la que nunca dejas de sorprenderte es con ImageMagick. Aunque está destinada para tareas con formatos gráficos, es posible utilizarla para reducir el tamaño de un archivo PDF.

Para instalarlo usar:

```
# apt install imagemagick
```

ImageMagick es capaz de trabajar junto a GhostScript para reducir el tamaño de las imágenes que contiene. Para ello, utilizaremos la herramienta convert con los parámetros -compress y -quality.

```
$ convert original.pdf -compress jpeg speaker_jpeg.pdf
```

Con el comando convert -list compress puedes obtener una lista de las opciones de compresión que tienes a tu disposición (BZip, JPEG, LZW, Zip, Lossless...) para comprimir las imágenes del documento y mediante el parámetro -quality establecer la calidad de las imágenes (100 mayor calidad).

Podemos transfórmalas en imágenes usando:

```
$ convert archivo.pdf archivo.png
```

Podemos covertir imagenes de un formato a otro[39], usando:

```
$ convert archivo.{svg,png}
```

o reducir una imagen a un determinado porcentaje:

```
$ convert -resize 50% original.jpg modificada.jpg
```

También podemos transformar múltiples imágenes en un archivo *pdf* mediante:

```
$ convert imagen{1...8}.png archivo.pdf
```

es probable que se deban cambiar las políticas de ImageMagick para hacer que funcione.

Es posible generar una imágen con el texto indicado, usando:

[39] Soporta los formatos: SVG, BMP, TIFF, PNG, JPG, GIF, WEBP, etc.

```
$ convert -size 1000x600 -define gradient:radii=1000,500 \
radial-gradient:#884b88-#010101 -font Impact -pointsize 72
\
-fill white -gravity center -interline-spacing 50 -annotate 0,0 \
"¡Herramientas computacionales en Linux!" Archivo.png
```

y es posible crear una paleta de colores animada con nombres usando:

```
colors=( $( convert -list color | awk '/srgb/{print "xc:"$1}' )
) ; \
montage -pointsize 12 -fill black -label "%f" -geometry \
100x100+1+1 ${colors[@]} -tile 5x5 colors.gif ; convert \
-delay 80 colors.gif colors.gif ; animate colors.gif
```

pdfgrep es una utilidad de línea de comandos para buscar texto en archivos PDF de forma simple y funcional ahorrándonos tiempo de acceder a cada archivo y buscar el texto con herramientas propias de PDF. Para instalar usamos:

```
# apt install pdfgrep
```

Algunas de sus características son:

- Compatible con Grep, podremos ejecutar muchos parámetros de grep como -r, -i, -n o -c.
- Capacidad de buscar texto en múltiples archivos PDF
- Colores destacados, esta opción de color de GNU Grep es compatible y está habilitada por defecto.
- Admite el uso de expresiones regulares.

Iniciaremos con una búsqueda simple, por ejemplo, buscaremos la palabra Solvetic en el archivo Solvetic.pdf, para ello ejecutamos lo siguiente:

```
$ pdfgrep Solvetic Solvetic.pdf
```

Las opciones generales que nos ofrece pdfgrep son:

-i, –ignore-case, Ignora las distinciones de los casos tanto en el origen como en los archivos de entrada.

-F, –fixed-strings, Interpreta PATTERN como una lista de cadenas fijas separadas por líneas nuevas.

–Cache, Usa una Caché para el texto renderizado con el fin de acelerar la operación en archivos de gran tamaño.

-P, –perl-regexp, Interpreta PATTERN como una expresión regular compatible con Perl (PCRE).

-H, –with-filename, Imprime el nombre del archivo para cada coincidencia.

-h, –no-nombre de archivo, Suprime el prefijo del nombre de archivo en la salida.

-n, –page-number, Prefija cada coincidencia con el número de la página donde se encontró el termino buscado.

-c, –count, Suprime la salida normal y, en su lugar, imprime el número de coincidencias para cada archivo de entrada.

-p, –Conteo de páginas, Imprime el número de coincidencias por página. Implica -n.

–color, Permite resaltar nombres de archivos, números de página y texto coincidente con diferentes secuencias para mostrarlos en color en la terminal, algunas de sus opciones son Siempre, nuca o automático.

-o, –only-matching, Imprime solo la parte coincidente de una línea sin ningún contexto circundante.

-r, –recursive, Nos permite buscar de forma recursiva todos los archivos (restringidos por –include y –exclude) debajo de cada directorio, siguiendo los enlaces simbólicos solo si están en la línea de comando.

-R, –de referencia-recursiva, Igual que -r, pero sigue todos los enlaces simbólicos.

lowriter uso de la CLI de LibreOffice 'Lowriter' para la conversión de PDF, para instalarlo usamos:

```
# apt instal libreoffice
```

Para realizar la conversión, no tendremos más que seguir la siguiente sintaxis y usar el comando para convertir un solo archivo .doc, ubicado en nuestro directorio actual de trabajo:

```
$ lowriter –convert-to pdf Ejemplo1.doc
```

Si lo que quieres es convertir un archivo .docx, la orden a utilizar es prácticamente la misma:

```
$ lowriter –convert-to pdf Ejemplo2.docx
```

Como puede ver en las anteriores capturas de pantalla, cuando listé el contenido de mi carpeta actual a través del comando ls, también se puede ver los archivos pdf recién creados.

Si nos interesa convertir un grupo de archivos a .pdf no tendremos más que utilizar la siguiente sintaxis. Esta nos servirá para convertir por lotes todos los archivos .doc o .docx a pdf ubicados en nuestro directorio actual:

```
$ lowriter –convert-to pdf *.doc
```

Si los archivos a convertir son .docx, el comando a utilizar será el siguiente:

```
$ lowriter –convert-to pdf *.docx
```

ghostscript para realizar la conversión-optimización necesitamos (en el caso de que no se tenga instalada):

```
apt install ghostscript
```

y que el archivo que queremos optimizar tenga de nombre original.pdf (se puede cambiar pero también necesitas cambiarlo en el comando siguiente), metemos la siguiente línea en nuestra terminal de comandos:

```
gs -sDEVICE=pdfwrite -dCompatibilityLevel=1.4 -dNOPAUSE
-dQUIET -dBATCH -sOutputFile=optimizado.pdf original.pdf
```

Una vez terminado el proceso se genera un archivo llamado optimizado.pdf que tendrá un peso inferior al original:

Pero y si ¿aún sigue siendo muy grande? tenemos otro comando que aún lo reduce más:

```
gs -sDEVICE=pdfwrite -dCompatibilityLevel=1.4 -dPDFSETTINGS=/screen
-dNOPAUSE -dQUIET -dBATCH -sOutputFile=optimizado.pdf
original.pdf
```

Otras opciones de dPDFSETTINGS son:

- /screen Baja calidad (72 dpi) menor tamaño
- /ebook Mediana calidad (150 dpi) moderado tamaño
- /printer Alta calidad (300 dpi) gran tamaño
- /prepress (omisión) Alta calidad (300 dpi) preservando el color
- /default Casi identico a screen, pero con calidad ligeramente superior

Con estos sencillos pasos conseguirás bajar el peso del *PDF*, esta claro donde más comprime son las imágenes así que hay que revisarlas antes, para verificar que la calidad es la adecuada.

qpdf es una herramienta de la línea de comandos que permite trabajar con archivos *pdf*. Esta herramienta permite crear archivos optimizados para la *Web*, así como cifrar y descifrar archivos. También te permite convertir archivos con objetos comprimidos en archivos sin objetos comprimidos, así como soporta un modo que te permite editar el contenido del archivo *pdf* en un editor de texto.

Para instalarla usamos:

```
# apt install qpdf
```

Si necesitamos separar las páginas de un pdf (que generará los archivos salida-01.pdf, salida-02.pdf, etc.), usamos:

$ qpdf –split-pages original.pdf salida.pdf

Si queremos extraer las primeras diez hojas de un documento, solo tenemos que ejecutar el siguiente comando:

$ qpdf –empty –pages ejemplo.pdf 1-10 – salida.pdf

donde, ejemplo.pdf, es el archivo de entrada, 1-10, las páginas que copiamos del fichero de entrada, salida.pdf, el fichero de salida.

También podemos reordenar las páginas de un documento en orden inverso:

$ qpdf –empty –pages ejemplo.pdf z-1 – salida.pdf

donde, z, representa la última página.

Podemos unir las páginas de un documento a las del otro:

$ qpdf –empty –pages documento1.pdf 1-z documento2.pdf 1-z – salida.pdf

También nos permite seleccionar páginas sueltas o rangos tanto de un documento como de varios:

$ qpdf –empty –pages documento1.pdf 1,5-7,10-z documento2.pdf z-8,5,4,2 – salida.pdf

Además podemos rotar las páginas, por ejemplo 180 grados usando:

$ qpdf archivo.pdf salida.pdf –rotate=+180

Otra interesante opción que ofrece *qpdf* es la posibilidad de cifrar y descifrar documentos *pdf*. De cualquier forma, para cifrar un archivo pdf tendrás que utilizar la etiqueta -flag, de forma que la sintaxis es algo como:

$ qpdf entrada.pdf –encrypt passwd_usr passwd_own longitud_passwd [restricciones] – salida.pdf

o

$ qpdf –linearize –encrypt "" "MiContraseña" 128 –print=full –modify=none –extract=n –use-aes=y – Guia_Limpia.pdf Guia_Protegida.pdf

La longitud de la contraseña, puede tomar uno de los siguientes valores, 40, 128 ó 256. Si no se indican restricciones es permisivo por defecto. Si la longitud de la contraseña es 40, se pueden utilizar las siguientes restricciones:

-print=[yn], indica si se puede imprimir

-modify=[yn], indica si se puede modificar

-extract=[yn], indica si se puede extraer texto o imágenes

-annotate=[yn], indica si se pueden añadir comentarios, cumplimentar un formulario y firmar

Si la longitud es de 128, las restricciones podrán ser las siguientes:

-accessibility=[yn], indica si se es accesible

-extract=[yn], indica si se puede extraer texto o imágenes

-print=print-opt, controla el acceso a la impresión, pudiendo ser alguna de las siguientes opciones:

full, permite la impresión
low, permite la impresión solo a baja resolución
none, no te permite imprimir

-modify=modify-opt, determina si se puede modificar el documento, siguiendo criterios similares a los de impresión:

all, permite cualquier modificación
annotate, permite anotaciones y cumplimentar un formulario
form, permite rellenar un formulario y firmar
none, no permite ninguna modificación

Si tenemos un pdf cifrado y queremos conocer el tipo de cifrado usamos:

$ qpdf –password='passwd' –show-encryption test.pdf

y para quitar la clave usamos:

$ qpdf –password='passwd' –decrypt test.pdf salida.pdf

pdfcrack esta aplicación permite obtener la clave de un pdf usando fuerza bruta, se puede instalar con:

```
# apt install pdfcrack
```

Una vez instalado, ejecutaremos la herramienta[40] con el comando:

```
$ pdfcrack -f nombre_del_archivo.pdf
```

El proceso puede ser muy largo, y más teniendo en cuenta que pdfcrack sólo usa un procesador. Eso sí, puede usar el 100% del mismo. Para acelerar el proceso, podemos añadirle caracteres para probar, para lo que usaremos la opción -c[41]. Esto le dará un punto de partida y puede ser útil si siempre usamos un patrón. Por ejemplo, el siguiente comando sería si yo usara contraseñas con la palabra "coche" y números:

```
$ pdfcrack -f nombre_del_archivo.pdf -c coche1234
```

Si, por la razón que sea, queremos parar el proceso podemos hacerlo con Ctrl+c. En el momento lo pulsemos, *pdfcrack* intentará salvar el estado del proceso, lo que significa que podremos seguirlo más adelante. El archivo del proceso suele guardarse con el nombre "savedstat.sav" en nuestra carpeta personal y para seguirlo usaremos la opción -1. El comando quedaría así:

```
$ pdfcrack -f nombre_del_archivo.pdf -1 savedstate.sav
```

Otra opción que podemos configurarle es el número mínimo (-n=valor) y máximo (-m=valor) de caracteres:

```
$ pdfcrack -f nombre_del_archivo.pdf -m=20 -n=12 -c 100690
```

Con el comando anterior le habremos dicho que:

[40] Por omisión, usara estos caracteres para encontrar el password:
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
Podemos tener guardado en un archivo los caracteres a usar y usar -w para indicar que estos se lean de dicho archivo.

[41] Podemos usar toda la colección de caracteres posibles, usando algo como:

```
-c 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789.-
_@#~%$&'
```

La contraseña tiene un mínimo de 12 caracteres.

El máximo que debe combinar son 20 caracteres.

La contraseña tiene en algún punto los caracteres "100690" (no es necesario que estén ordenados).

Otras opciones disponibles son:

-w: para abrir un archivo de texto en donde le configuremos varias palabras. Esto es lo que se conoce como un diccionario. Muchas herramientas que usan la fuerza bruta para descifrar contraseñas cuentan con un diccionario u opción para añadírselo.

-o: para que trabaje con la contraseña de un dueño.

-p: proporciona la contraseña de un usuario para facilitar a obtener la contraseña de un dueño.

-s: la permutación está limitada a cambiar la primera palabra por mayúsculas.

-b: nos mostrará unos benchmarks para ver el rendimiento de pdfcrack durante el proceso.

ps2pdf este paquete es parte de ghostscript, para instalarlo usar:

```
apt install ghostscript
```

permite convertir archivos *.ps* a *.pdf*, mediante:

```
$ ps2pdf archivo.ps
```

y lo podemos usar para generar *pdf* desde la línea de comandos, ejemplo:

```
$ man -t top | ps2pdf - top.pdf
```

pdf2ps este paquete es parte de ghostscript, para instalarlo usar:

```
apt install ghostscript
```

permite convertir archivos *.pdf* a *.ps*, mediante:

```
$ pdf2ps archivo.pdf
```

Podemos usar este comando para bajar la calidad de un *pdf* mediante:

```
$ pdf2ps -dLanguageLevel=3 archivo.pdf
$ ps2pdf -dPDFSETTINGS=/ebook archivo.ps
```

htmldoc permite convertir documentos HTML a PDF y PS, para instalarlo usar:

```
# apt install htmldoc
```

y convertimos mediante:

```
$ htmldoc –webpage -f nombre.pdf nombre.html
```

wkhtmltopdf permite convertir páginas Web a PDF, para instalarlo usar:

```
# apt install wkhtmltopdf
```

y convertimos mediante:

```
$ wkhtmltopdf www.debian.org debian.pdf
```

podemos pedir que lo genere en tonos de gris mediante:

```
$ wkhtmltopdf -g www.debian.org debian.pdf
```

o que excluya las imágenes mediante:

```
$ wkhtmltopdf –no-images www.debian.org debian.pdf
```

y si lo necesitamos podemos convertir la página Web a una imagen mediante:

```
$ wkhtmltopdf www.debian.org debian.png
```

enscript convertir archivos ASCII a *.ps* o *.pdf*, para instalarlo usar:

```
# apt install enscript ghostscript
```

Supongamos que necesitamos convertir el archivo salida.txt a formato PDF. Pero además deseamos que la fuente sea "Courier" en tamaño "7", los márgenes sean de apenas 1 punto, y el formato de página o tamaño de papel sea "A4".

Primero se debe convertir el archivo .txt a PS utilizando enscript con las siguientes opciones:

```
$ enscript -M A4 –margins=1:1:1:1 -f Courier7 salida.txt -p
salida.ps
```

luego simplemente convertir el archivo PS generado por enscript a PDF utilizando ps2pdf:

```
$ ps2pdf salida.ps
```

Las opciones más comunes de enscript son:

- -a: rango de páginas a imprimir, por ejemplo: "1-10".
- -b: encabezado de página en formato de texto, por ejemplo "$n %W Página $% de $=’".
- -f: fuente y tamaño a utilizar, por ejemplo "Times-Roman12".
- -M: tamaño de papel, por ejemplo: "A4"
- -p: nombre del archivo de salida o "-" para volcar por salida estándar.
- -r: modo "landscape".
- –margins: márgenes (en puntos).
- –footer: pie de página (funciona de manera similar a -b).

Y decenas de opciones más, incluso permite generar un índice o tabla de contenidos (–toc).

pandoc es posible convertir los archivos MD (MarkDown) comunes en Github o GitLab a PDF[42], es necesario instalar el paquete mediante:

```
# apt install pandoc
```

y lo usamos mediante:

```
$ pandoc archivo.md –pdf-engine=xelatex -o archivo.pdf
```

o

```
$ pandoc archivo.md -o archivo.pdf
```

Podemos convertir un archivo MarkDown a .docx, usando:

```
$ pandoc archivo.md -o archivo.docx
```

o convertir un archivo .docx a MarkDown, usando:

```
$ pandoc archivo.docx -o archivo.md
```

Podemos convertir un archivo MarkDown a .html, usando:

```
$ pandoc archivo.md -o archivo.html
```

o convertir un archivo .html a MarkDown, usando:

```
$ pandoc archivo.html -o archivo.md
```

esta herramienta soporta una amplia variedad de conversiones, para más detalle ver su página Web.

[42] Se necesita tener instalado LaTeX, podemos instalarlo usando:

```
# apt install texlive
```

o

```
# apt install texlive-full
```

Otras Herramientas para Trabajar con PDF existen múltiples herramientas para trabajar archivos PDF, a continuación describimos algunas:

- diffpdf.- Muestra las diferencias de dos versiones de un mismo documento
- pdfarranger.- Permite juntar, separar y reordenar páginas de uno o más PDF
- pdfchain.- Interfase gráfica para PDF Tool Kit
- pdfposter.- Permite escalar y colocar en mosaico imágenes/páginas para imprimir en formato largo
- pdf-presenter-console.- Herramienta de presentación multimonitor
- pdfcrack.- Permite encontrar la clave de protección de un PDF mediante fuerza bruta

4.3 Desde la Nube

Existen diferentes servicios Web[43] que permiten editar, compilar y generar el archivo PDF o DVI desde el navegador, esto en aras de que los estudiantes y profesores que cuenten con algún sistema de acceso a red y un navegador puedan trabajar en LaTeX sin hacer instalación alguna en su equipo de cómputo, tableta o teléfono celular.

Algunos ejemplos de estos servicio son:

- https://es.sharelatex.com/
- https://papeeria.com/
- https://www.overleaf.com/

[43] Cuando se trabaja desde la Web es recomendable usar el modo Privado o Incógnito para no guardar el historial de navegación, información introducida en los formularios y borrar al cerrar el navegador los datos de los sitios visitados. Pero recuerda que los sitios Web que visitamos sí guardan información de nuestra visita, nuestro proveedor de Internet también guarda constancia de nuestra visita y si descargamos algo, esto no se borra al igual que el historial de descargas, además de las marcas de páginas o favoritos se conservarán al cerrar al navegador.

- https://www.authorea.com/
- https://latexbase.com/
- https://www.codecogs.com/latex/eqneditor.php

Detexify permite conocer el código de LaTeX de un símbolo o mediante el dibujo de un símbolo nos muestra diferentes opciones y su respectiva codificación en LaTeX, se puede consultar en:
https://detexify.kirelabs.org/classify.html

Tex Match es una aplicación Snapd que puede ser instalada para diversas distribuciones de Linux que permite conocer el código de LaTeX de más de 1000 símbolos o mediante el dibujo de un símbolo nos muestra diferentes opciones y su respectiva codificación en LaTeX, se puede consultar en:
https://snapcraft.io/tex-match

Mathpix es una aplicación (Linux, MacOS, Windows) de escáner para crear documentos digitales que contienen ecuaciones matemáticas con la mínima cantidad de esfuerzo. Digitaliza texto escrito a mano o impreso y copia los resultados al portapapeles que se puede pegar en editores LaTeX, se puede consultar en:
https://mathpix.com

Mathcha es un editor en línea que permite escribir y compartir textos e imágenes de mátemáticas, se puede consultar en:
https://www.mathcha.io

5 Generación de Imágenes y Gráficación de Datos

La visualización gráfica de datos constituye una disciplina propia dentro del universo de las Ciencias e Ingenierías y en particular en la ciencia de datos. Esta práctica ha marcado hitos importantes a lo largo de la historia en la analítica de datos. Desde el gráfico más sencillo hasta el más complejo y refinado, todos ofrecen alto valor tanto al analista, durante su proceso de ciencia de datos, como al usuario final, al cual estamos comunicando una historia basada en datos.

¿Por qué es tan importante la visualización gráfica de los datos? En la ciencia de datos existen muchos tipos diferentes de datos para analizar. Una forma de clasificación de los datos atiende al nivel de estructuración lógica que éstos tienen. Por ejemplo, se entiende que los datos en formatos similares a hojas de cálculo -aquellos datos que se estructuran en forma de filas y columnas- son datos con una estructura bien definida - o datos estructurados- Sin embargo, aquellos datos como los 140 caracteres de un Feed de Twitter se consideran datos sin estructura - o desestructurados-.

En medio de estos dos extremos se encuentra toda una gama de grises, que va desde los ficheros delimitados por caracteres especiales (comas, puntos y comas, espacios, etc.) hasta las imágenes o los videos de Youtube. Es evidente que las imágenes y los vídeos solamente cobran sentido humano una vez representadas visualmente. De nada serviría (para un humano) que presentaremos una imagen como una matriz de números que representan una combinación de colores RGB (Red, Green, Blue). En el caso de los datos estructurados, su representación gráfica es necesaria en todas las etapas del proceso de análisis, desde la etapa exploratoria, hasta la presentación final de resultados.

La visualización gráfica de los datos tiene un papel fundamental en todos los estadios del análisis de datos. Existen múltiples aproximaciones sobre cómo realizar un proceso de análisis de datos de forma correcta y completa.

La visualización de datos está en el núcleo del proceso. Es una herramienta básica para el analista o científico de datos que -mediante un proceso iterativo- va transformando y componiendo un modelo lógico de los datos. Apoyándose en la visualización, el analista va descubriendo los secretos enterrados en los datos. La visualización permite de forma rápida:

- Descartar aquellos datos poco representativos o erróneos.
- Identificar aquellas variables que dependen unas de otras y por lo tanto contienen información redundante
- Realizar cortes a los datos para poder observarlos desde diferentes perspectivas.
- Finalmente, comprobar que aquellos modelos, tendencias, predicciones y agrupaciones que hemos aplicado sobre los datos, nos devuelven el resultado esperado.

Herramientas para el análisis visual de datos Tan importante es la visualización gráfica de los datos en todos los ámbitos de la ciencia, ingeniería, negocios, banca, medio ambiente, etc. que existen multitud de herramientas para diseñar, desarrollar y comunicar la visualización gráfica de los datos. Estas herramientas cubren un amplio espectro del público objetivo, desde desarrolladores de Software, hasta científicos de datos pasando por periodistas y profesionales de la comunicación.

Para desarrolladores de Software, existen cientos de librerías y paquetes de Software que contienen miles de tipos de visualizaciones. Los desarrolladores tan solo tienen que cargar estas librerías en sus respectivos entorno de trabajo de programación y parametrizar el tipo de gráfico que deseen generar. El desarrollador tan solo ha de indicar los datos de origen que desea representar, el tipo de gráfico (líneas, barras, etc.) y la parametrización de dicho gráfico (escalas, colores, etiquetas, etc.).

El científico de datos acostumbra a trabajar con un entorno de trabajo de análisis concreto que, normalmente, incluye todos los componentes, entre ellos su motor de análisis visual de los datos. Los entornos más populares, hoy en día, para la ciencia de datos son R y Python, y ambos incluyen librerías nativas para la analítica visual. Quizás la librería más popular y potente en R sea ggplot2, mientras que en Python, matplotlib y Plotly son de las más populares.

Para comunicadores profesionales o personal no técnico de las distintas áreas de negocio (Marketing, Recursos Humanos, Producción, etc.) que necesita tomar decisiones basadas en datos, existen herramientas - que no son únicamente herramientas de Visual Analytics - con funcionalidades para generar representaciones gráficas de los datos. Herramientas modernas de Business Intelligence de autoservicio como MS Excel, MS Power BI, Qlik, Tableau,

etc. son estupendas herramientas para comunicar los datos sin necesidad de disponer de competencias en programación o codificación.

En definitiva, las herramientas de visualización permiten a todos estos profesionales acceder a los datos de una manera más ágil y sencilla. En un universo donde la cantidad de datos útiles a analizar no deja de crecer, cada vez son más necesarias este tipo de herramientas, que facilitan la obtención de valor procedente de los datos y, con ello, la toma decisiones relativas al presente y al futuro de nuestro negocio o actividad. Se ponen a disposición los fuentes de los gráficos mostrados a continuación y otros más en la liga: Graficos

5.1 Python

El lenguaje Python permite realizar una amplia variedad de visualizaciones de datos en 2D, 3D y animaciones. La misma se realiza mediante la ayuda de módulos (como math o numpy) que están destinados a brindar herramientas de visualización.

En la lista que sigue se presentan algunos de los módulos más usados. El objetivo no es aprenderlos, sino saber que existen varias opciones y disminuir un poco la confusión que se puede producir al principio al buscar ayuda y bibliografía:

- Matplotlib
- Seaborn
- Plotly
- Bokeh
- Altair
- Pygal
- Pandas
- Plotnine

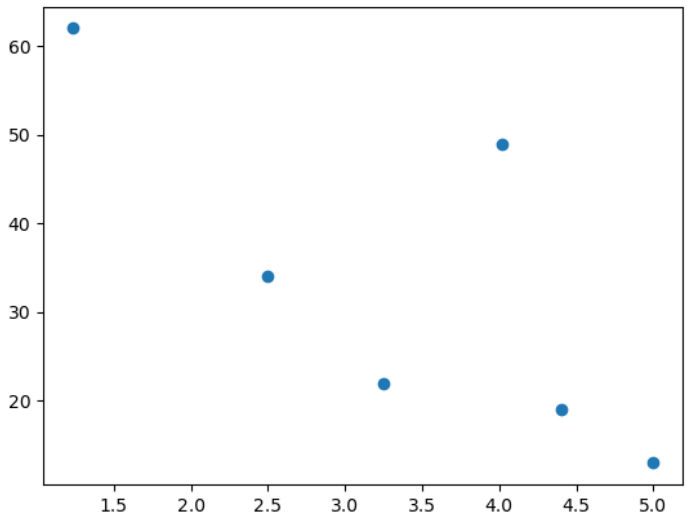
La elección de una de estas herramientas para hacer un gráfico depende del contexto, para qué se quiere hacer el gráfico, dónde se va a mostrar y a partir

de qué datos. Así, por ejemplo Plotly, Bokeh y Altair devuelven gráficos en HTML para ser mostrados en páginas Web, Pygal genera gráficos vectoriales y Pandas grafica datos guardados en cierto tipo especial de estructura.

Por ejemplo:

```
import matplotlib.pyplot as plt
precio = [2.50, 1.23, 4.02, 3.25, 5.00, 4.40]
ventas_por_dia = [34, 62, 49, 22, 13, 19]
plt.scatter(precio, ventas_por_dia)
plt.savefig('Grafica.pdf')
plt.savefig('Grafica.png')
plt.show()
```

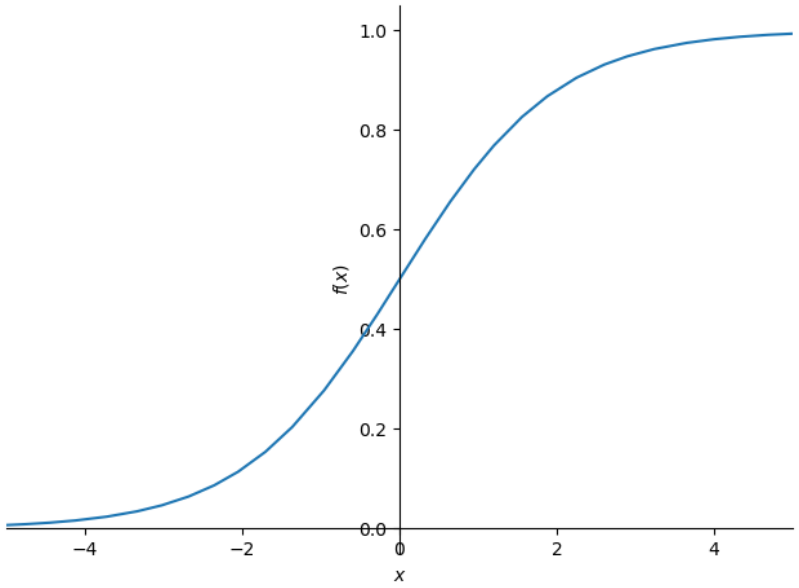
genera el siguiente gráfico (además lo graba en formato PDF y PNG):



Otro ejemplo:

```
import sympy
graf = sympy.plot("1/(1+exp(-x))", xlim=(-5,5));
graf.save('Grafica.pdf')
graf.save('Grafica.png')
```

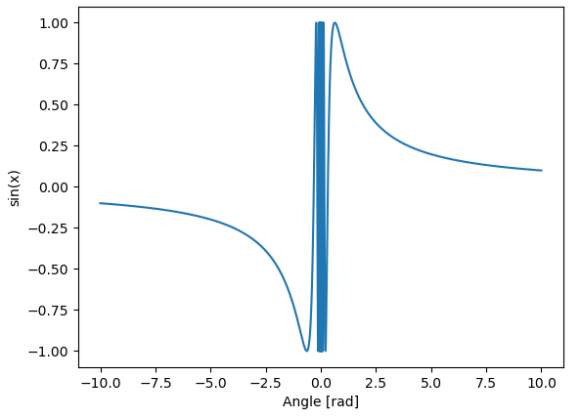
genera el siguiente gráfico (además lo graba en formato PDF y PNG):



Otro ejmplo:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-10, 10, 100000)
y = np.sin(1 / x)
# se graban los datos
np.savetxt('valores.txt'.(x,y))
# se leen los datos
xx,yy = np.loadtxt('valores.txt')
# se grafica
plt.figure()
plt.plot(xx, yy)
plt.xlabel("Angle [rad]")
plt.ylabel("sin(x)")
plt.axis("tight")
plt.show()
```

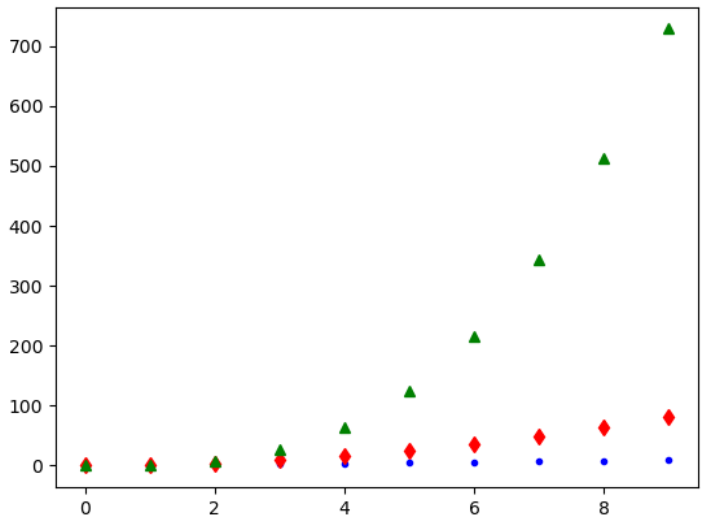
genera el siguiente gráfico (además de grabar y leer los valores de las variables x e y):



Otro ejemplo:

```
from pylab import *
import matplotlib.pyplot as plt
x = arange(10.) # array de floats, de 0.0 a 9.0
x2 = x**2              # definimos el array x2
x3 = x**3              # definimos el array x3
# dibujamos tres curvas en el mismo gráfico y figura
plot(x, x, 'b.', x, x2, 'rd', x, x3, 'g^')
show()
```

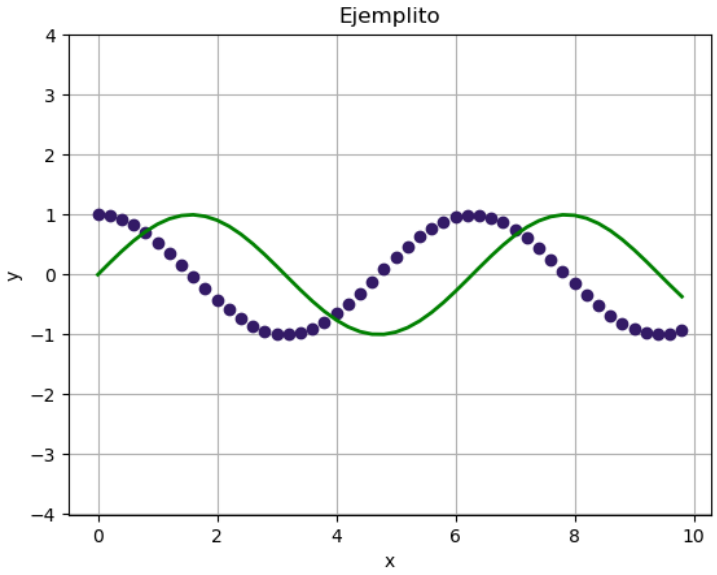
genera el siguiente gráfico:



Otro ejemplo:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(0,10,0.2)
y1 = np.cos(x)
y2 = np.sin(x)
plt.plot(x,y1,'o',linewidth=3,color=(0.2,0.1,0.4))
plt.plot(x,y2,'-',linewidth=2,color='g')
plt.grid()
plt.axis('equal')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Ejemplito')
plt.show()
```

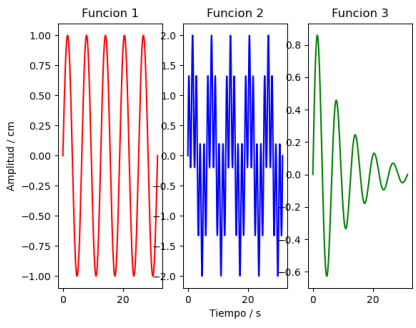
genera el siguiente gráfico:



Otro ejemplo:

```
from pylab import *
import matplotlib.pyplot as plt
x = linspace(0, 10*pi, 800) # array de valores a representar
def f1(x):
    return sin(x)
def f2(x):
    return sin(x) + sin(5.0*x)
def f3(x):
    return sin(x) * exp(-x/10.)
subplot(131)
p1, = plot(x,f1(x),'r-')
ylabel('Amplitud / cm')
title('Funcion 1')
subplot(132)
p2, = plot(x,f2(x),'b-')
xlabel('Tiempo / s')
title('Funcion 2')
subplot(133)
p3, = plot(x, f3(x),'g-')
title('Funcion 3')
show()
```

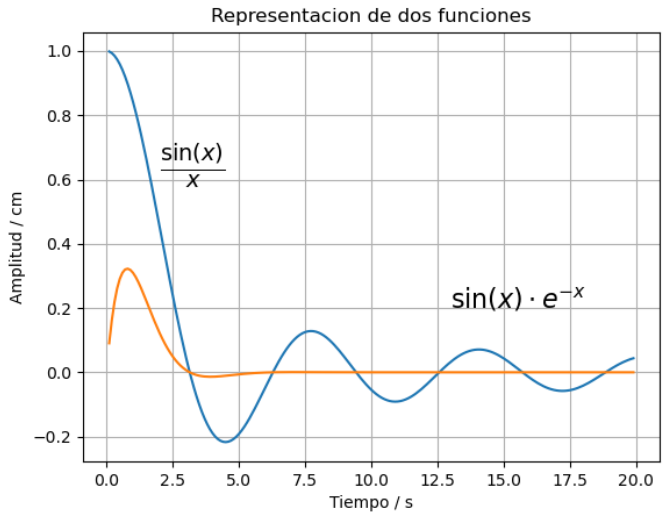
genera el siguiente gráfico:



Otro ejemplo:

```
from pylab import *
import matplotlib.pyplot as plt
t = arange(0.1, 20, 0.1)
y1 = sin(t)/t
y2 = sin(t)*exp(-t)
p1, p2 = plot(t, y1, t, y2)
texto1 = text(2, 0.6, r'$\frac{\sin(x)}{x}$', fontsize=20)
texto2 = text(13, 0.2, r'$\sin(x) \cdot e^{-x}$', fontsize=16)
grid()
title('Representacion de dos funciones')
xlabel('Tiempo / s')
ylabel('Amplitud / cm')
show()
```

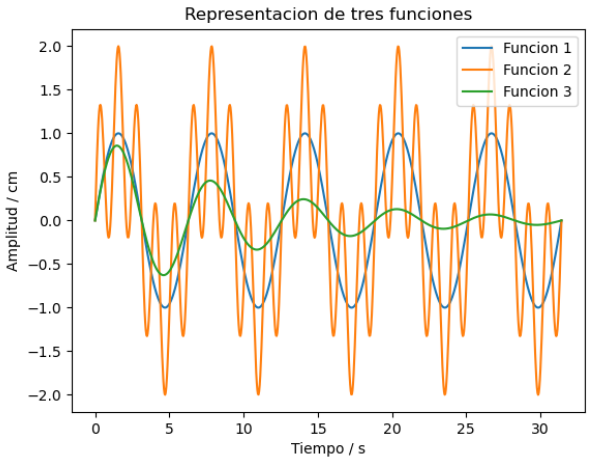
genera el siguiente gráfico:



Otro ejemplo:

```
from pylab import *
import matplotlib.pyplot as plt
def f1(x):
    return sin(x)
def f2(x):
    return sin(x) + sin(5.0*x)
def f3(x):
    return sin(x) * exp(-x/10.)
x = linspace(0, 10*pi, 800)
p1, p2, p3 = plot(x, f1(x), x, f2(x), x, f3(x))
legend(('Funcion 1', 'Funcion 2', 'Funcion 3'),
prop = {'size': 10}, loc='upper right')
xlabel('Tiempo / s')
ylabel('Amplitud / cm')
title('Representacion de tres funciones')
figure(figsize=(12, 5))
show()
```

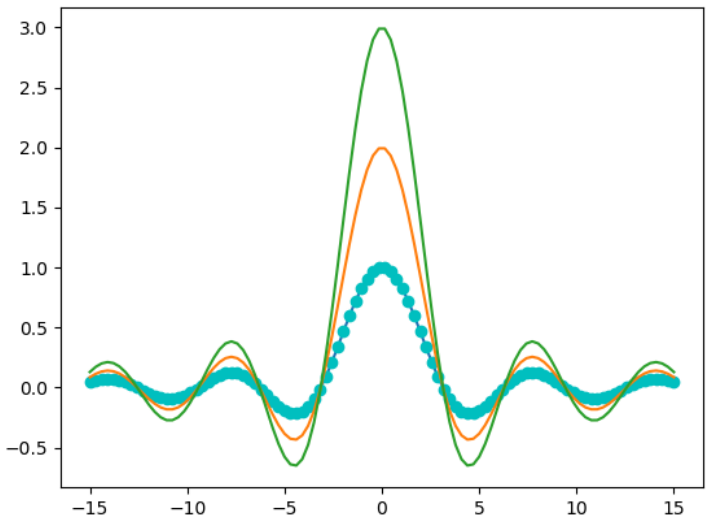
genera el siguiente gráfico:



Otro ejemplo:

```
import pylab
import numpy
x = numpy.linspace(-15, 15, 100)
y = numpy.sin(x) / x
pylab.plot(x, y)
pylab.plot(x, y, "co")
pylab.plot(x, 2 * y, x, 3 * y)
pylab.show()
```

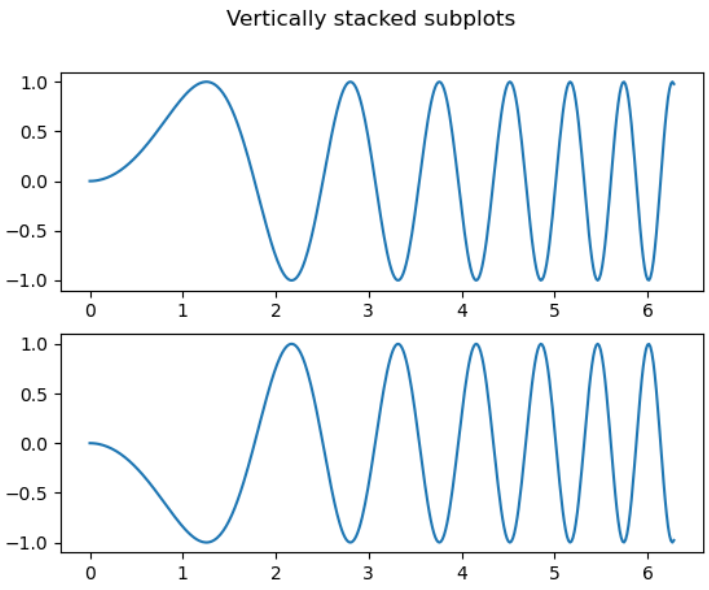
genera el siguiente gráfico:



Otro ejemplo:

```
import matplotlib.pyplot as plt
import numpy as np
def multiple_plots():
    # Some example data to display
    x = np.linspace(0, 2 * np.pi, 400)
    y = np.sin(x ** 2)
    fig, axs =plt.subplots(2)
    fig.suptitle('Vertically stacked subplots')
    axs[0].plot(x, y)
    axs[1].plot(x, -y)
    plt.show( )
multiple_plots()
```

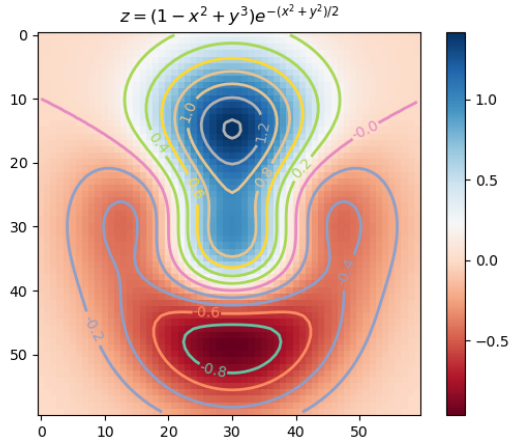
genera el siguiente gráfico:



Otro ejemplo:

```
from numpy import exp, arange
from pylab import meshgrid, cm, imshow, contour, clabel,
colorbar, axis, title, show
def z_func(x, y):
  return (1 - (x ** 2 + y ** 3)) * exp(-(x ** 2 + y ** 2) / 2)
x = arange(-3.0, 3.0, 0.1)
y = arange(-3.0, 3.0, 0.1)
X, Y = meshgrid(x, y)
Z = z_func(X, Y)
im = imshow(Z, cmap=cm.RdBu)
cset = contour(Z, arange(-1, 1.5, 0.2), linewidths=2, cmap=cm.Set2)
clabel(cset, inline=True, fmt="%1.1f", fontsize=10)
colorbar(im)
title("$z=(1-x^2+y^3) e^{-(x^2+y^2)/2}$")
show()
```

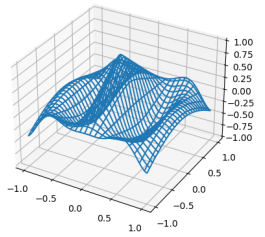
genera el siguiente gráfico:



Otro ejemplo:

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import numpy as np
import time
def generate(X, Y, phi):
    R = 1 - np.sqrt(X ** 2 + Y ** 2)
    return np.cos(2 * np.pi * X + phi) * R
fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")
xs = np.linspace(-1, 1, 50)
ys = np.linspace(-1, 1, 50)
X, Y = np.meshgrid(xs, ys)
ax.set_zlim(-1, 1)
wframe = None
tstart = time.time()
for phi in np.linspace(0, 180.0 / np.pi, 100):
    if wframe:
    ax.collections.remove(wframe)
    Z = generate(X, Y, phi)
    wframe = ax.plot_wireframe(X, Y, Z, rstride=2, cstride=2)
    plt.pause(0.001)
```

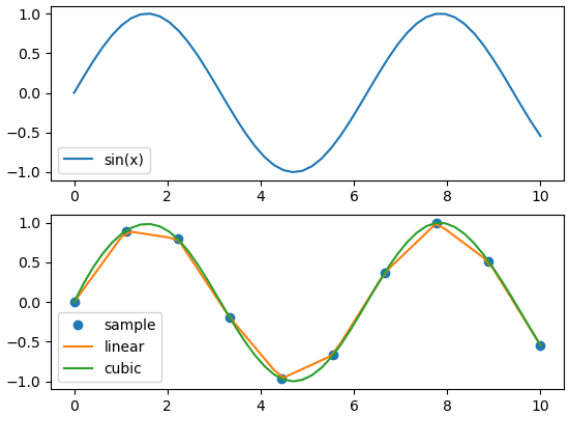
genera el siguiente animación:



Otro ejemplo:

```
import scipy.interpolate as sp
import numpy
import pylab
xx = numpy.linspace(0, 10, 50)
yy = numpy.sin(xx)
x = numpy.linspace(0, 10, 10)
y = numpy.sin(x)
fl = sp.interp1d(x, y,kind='linear')
fc = sp.interp1d(x, y,kind='cubic')
xnew = numpy.linspace(0, 10, 50)
pylab.subplot(211)
pylab.plot(xx, yy)
pylab.legend(['sin(x)'], loc='best')
pylab.subplot(212)
pylab.plot(x, y, 'o', xnew, fl(xnew), xnew, fc(xnew))
pylab.legend(['sample', 'linear', 'cubic'], loc='lower left')
pylab.show()
```

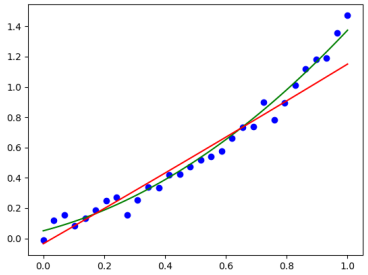
genera el siguiente gráfico:



Otro ejemplo:

```
from pylab import *
from numpy import *
from numpy.random import normal
from scipy.optimize import fmin
# parametric function, x independent variable, c parameters
fp = lambda c, x: c[0]+c[1]*x+c[2]*x*x
real_p = rand(3)
# error function to minimize
e = lambda p, x, y: (abs((fp(p,x)-y))).sum()
# generating data with noise
n = 30
x = linspace(0,1,n)
y = fp(real_p,x) + normal(0,0.05,n)
# fitting the data with fmin
p0 = rand(3) # initial parameter value
p = fmin(e, p0, args=(x,y))
print ('estimater parameters: ', p)
print ('real parameters: ', real_p)
xx = linspace(0,1,n*3)
plot(x,y,'bo', xx,fp(real_p,xx),'g', xx, fp(p,xx),'r')
show()
```

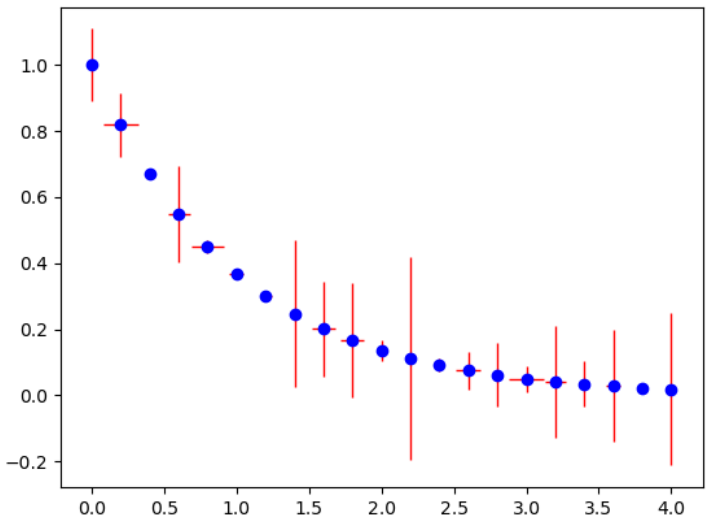
genera el siguiente gráfico:



Otro ejemplo:

```
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as npr
x=np.linspace(0,4,21)
y=np.exp(-x)
xe=np.abs(0.08*npr.randn(len(x)))
ye=np.abs(0.1*npr.randn(len(y)))
plt.errorbar(x,y,fmt='bo',lw=2,xerr=xe,yerr=ye,ecolor='r',elinewidth=1)
```

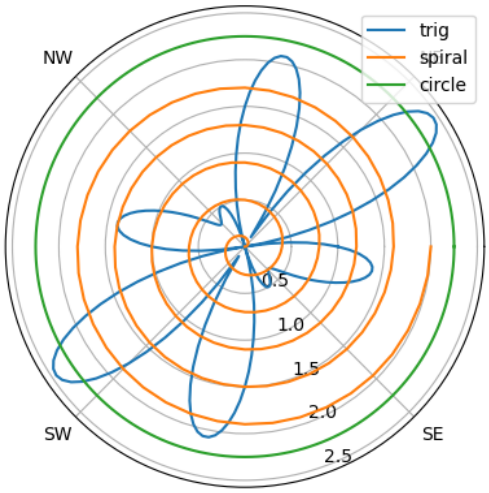
genera el siguiente gráfico:



Otro ejemplo:

```
import numpy as np
import matplotlib.pyplot as plt
theta=np.linspace(0,2*np.pi,201)
r1=np.abs(np.cos(5.0*theta) - 1.5*np.sin(3.0*theta))
r2=theta/np.pi
r3=2.25*np.ones_like(theta)
plt.polar(theta, r1,label='trig')
plt.polar(5*theta, r2,label='spiral')
plt.polar(theta, r3,label='circle')
plt.thetagrids(np.arange(45,360,90), ('NE','NW','SW','SE'))
plt.rgrids((0.5,1.0,1.5,2.0,2.5),angle=290)
plt.legend(loc='best')
plt.show()
```

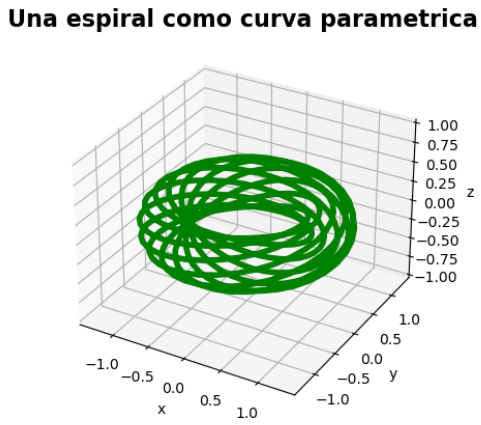
genera el siguiente gráfico:



Otro ejemplo:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
theta=np.linspace(0,2*np.pi,401)
a, m, n = 0.3, 11, 9
x=(1+a*np.cos(n*theta))*np.cos(m*theta)
y=(1+a*np.cos(n*theta))*np.sin(m*theta)
z=a*np.sin(n*theta)
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.plot(x,y,z,'g',linewidth=4)
ax.set_zlim3d(-1.0,1.0)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.set_title('Una espiral como curva parametrica', weight='bold',size=16)
plt.show()
```

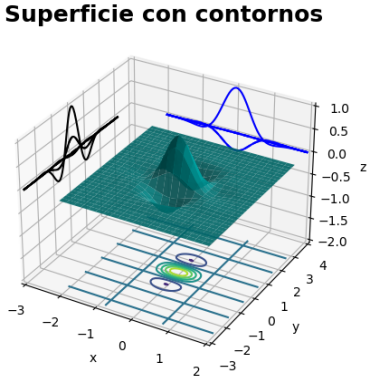
genera el siguiente gráfico:



Otro ejemplo:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
xx, yy=np.mgrid[-2:2:81j, -3:3:91j]
zz=np.exp(-2*xx**2-yy**2)*np.cos(2*xx)*np.cos(3*yy)
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.plot_surface(xx,yy,zz,rstride=4,cstride=3,color='c',alpha=0.9)
ax.contour3D(xx,yy,zz,zdir='x',offset=-3.0,colors='black')
ax.contour3D(xx,yy,zz,zdir='y',offset=4.0,colors='blue')
ax.contour3D(xx,yy,zz,zdir='z',offset=-2.0)
ax.set_xlim3d(-3.0,2.0)
ax.set_ylim3d(-3.0,4.0)
ax.set_zlim3d(-2.0,1.0)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.set_title('Superficie con contornos',weight='bold',size=18)
plt.show()
```

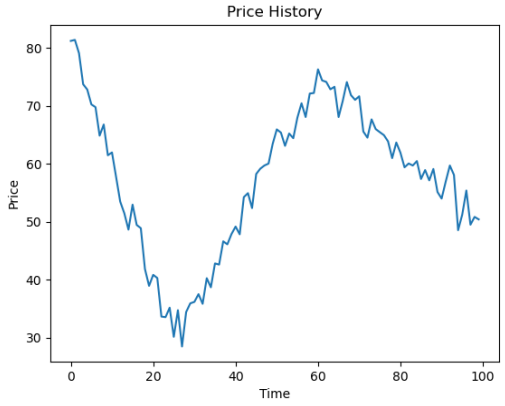
genera el siguiente gráfico:



Otro ejemplo:

```
import numpy as np
import matplotlib.pyplot as plt
prices = np.full(100, fill_value=np.nan)
prices[[0, 25, 60, -1]] = [80.0, 30.0, 75.0, 50.0]
x = np.arange(len(prices))
is_valid = ~np.isnan(prices)
prices = np.interp(x=x, xp=x[is_valid], fp=prices[is_valid])
prices += np.random.randn(len(prices)) * 2
mn = np.argmin(prices)
mx = mn + np.argmax(prices[mn:])
kwargs = {"markersize": 12, "linestyle": ""}
fig, ax = plt.subplots()
ax.plot(prices)
ax.set_title("Price History")
ax.set_xlabel("Time")
ax.set_ylabel("Price")
ax.plot(mn, prices[mn], color="green", **kwargs)
ax.plot(mx, prices[mx], color="red", **kwargs)
plt.show()
```

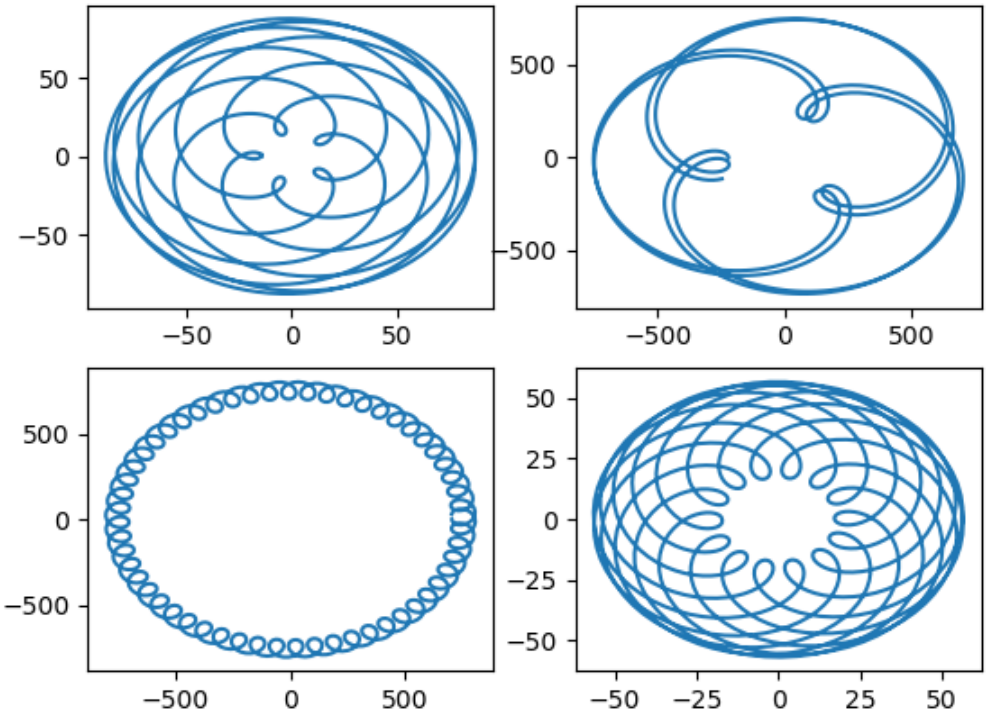
genera el siguiente gráfico:



Otro ejemplo:

```
import matplotlib.pyplot as plot
from numpy import linspace, sin, cos, pi
period = {
    'mercury' : 87.96926,
    'venus' : 224.7008,
    'earth' : 365.25636,
    'mars' : 686.97959,
    'ceres' : 1680.22107,
    'jupiter' : 4332.8201,
    'saturn' : 10755.699,
    'uranus' : 20687.153,
    'neptune' : 60190.03
}
dist = lambda T : T**(2/3) # Kepler
def plot_orbit(ax, planet1, planet2, periods=10):
    T1 = period[planet1]
    T2 = period[planet2]
    d1 = dist(T1)
    d2 = dist(T2)
    theta = linspace(0, 2*pi*periods, 1000)
    x = d1*cos(T2*theta/T1) - d2*cos(theta)
    y = d1*sin(T2*theta/T1) - d2*sin(theta)
    ax.plot(x, y)
fig=plot.figure()
ax = fig.add_subplot(2,2,1)
plot_orbit(ax, "venus", "earth", 8)
ax = fig.add_subplot(2,2,2)
plot_orbit(ax, "jupiter", "saturn", 4)
ax = fig.add_subplot(2,2,3)
plot_orbit(ax, "uranus", "earth", 57)
ax = fig.add_subplot(2,2,4)
plot_orbit(ax, "mercury", "venus", 9)
plot.show()
```

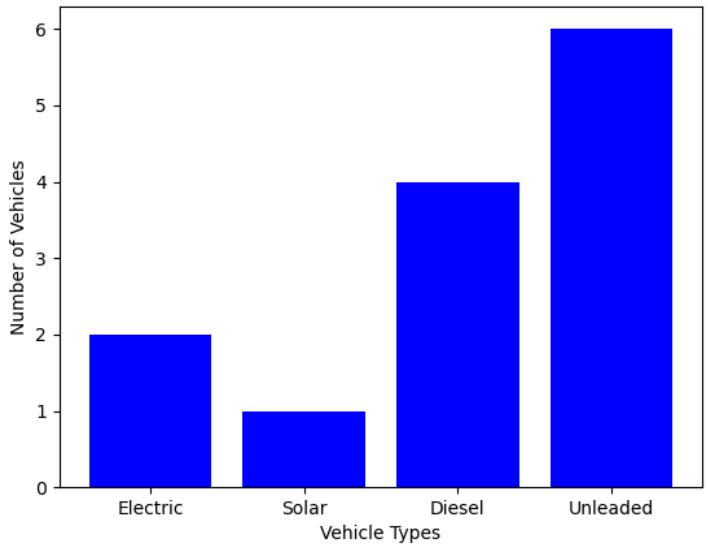
genera el siguiente gráfico:


50
0
−50
−50
0
50
500
0
−500
−500
0
500
500
0
−500
−500
0
500
50
25
0
−25
−50
−50
−25
0
25
50

Otro ejemplo:

```
import matplotlib.pyplot as plt
def bar_chart(numbers, labels, pos):
    plt.bar(pos, numbers, color="blue")
    plt.xticks(ticks=pos, labels=labels)
    plt.xlabel("Vehicle Types")
    plt.ylabel("Number of Vehicles")
    plt.show()
numbers = [2, 1, 4, 6]
labels = ["Electric", "Solar", "Diesel", "Unleaded"]
pos = list(range(4))
bar_chart(numbers, labels, pos)
```

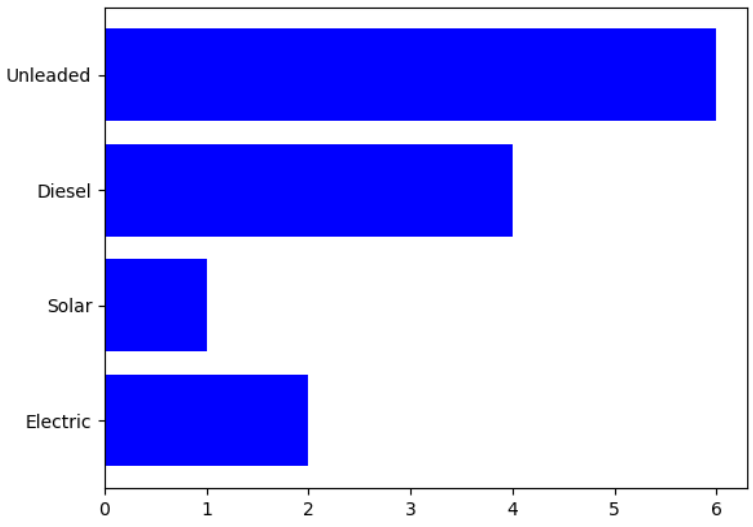
genera el siguiente gráfico:



Otro ejemplo:

```
import matplotlib.pyplot as plt
def bar_charth(numbers, labels, pos):
    plt.barh(pos, numbers, color="blue")
    plt.yticks(ticks=pos, labels=labels)
    plt.show()
numbers = [2, 1, 4, 6]
labels = ["Electric", "Solar", "Diesel", "Unleaded"]
pos = list(range(4))
bar_charth(numbers, labels, pos)
```

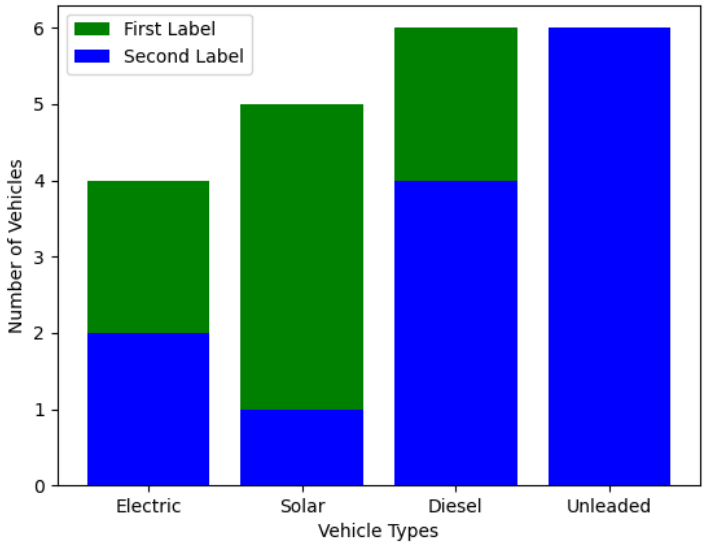
genera el siguiente gráfico:



Otro ejemplo:

```
import matplotlib.pyplot as plt
def bar_chart(numbers, labels, pos):
    plt.bar(pos, [4, 5, 6, 3], color="green")
    plt.bar(pos, numbers, color="blue")
    plt.xticks(ticks=pos, labels=labels)
    plt.xlabel("Vehicle Types")
    plt.ylabel("Number of Vehicles")
    plt.legend(["First Label", "Second Label"], loc="upper
left")
    plt.show()
numbers = [2, 1, 4, 6]
labels = ["Electric", "Solar", "Diesel", "Unleaded"]
pos = list(range(4))
bar_chart(numbers, labels, pos)
```

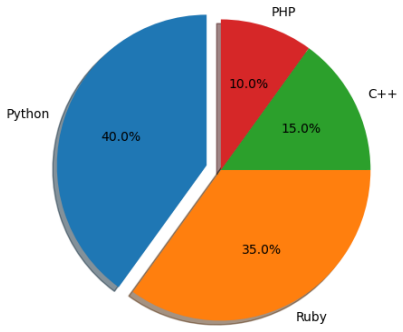
genera el siguiente gráfico:



Otro ejemplo:

```
import matplotlib.pyplot as plt
def pie_chart():
    numbers = [40, 35, 15, 10]
    labels = ["Python", "Ruby", "C++", "PHP"]
# Explode the first slice (Python)
    explode = (0.1, 0, 0, 0)
    fig1, ax1 = plt.subplots()
    ax1.pie(
        numbers,
        explode=explode,
        labels=labels,
        shadow=True,
        startangle=90,
        autopct="%1.1f%%",
    )
    ax1.axis("equal")
    plt.show()
pie_chart()
```

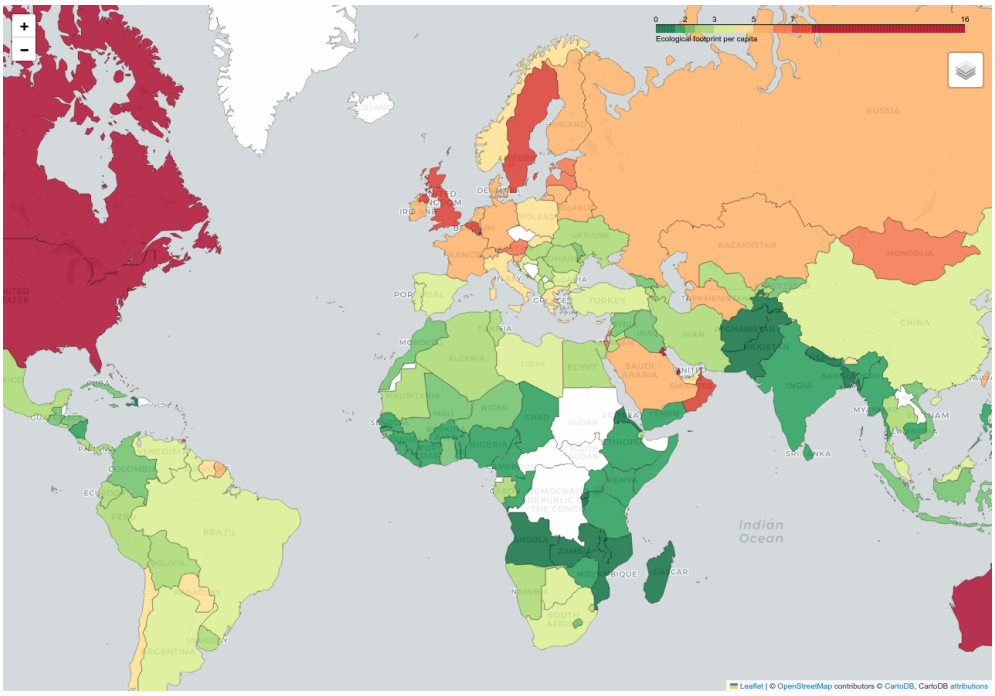
genera el siguiente gráfico:



Otro ejemplo:

```
import folium
import pandas as pd
eco_footprints = pd.read_csv("footprint.csv")
max_eco_footprint = eco_footprints["Ecological footprint"].max()
political_countries_url = (
    "http://geojson.xyz/naturalearth-3.3.0/ne_50m_admin_0_countries.geojson"
)
m = folium.Map(location=(30, 10), zoom_start=3, tiles="cartodb
positron")
folium.Choropleth(
    geo_data=political_countries_url,
    data=eco_footprints,
    columns=("Country/region", "Ecological footprint"),
    key_on="feature.properties.name",
    bins=(0, 1, 1.5, 2, 3, 4, 5, 6, 7, 8, max_eco_footprint),
    fill_color="RdYlGn_r",
    fill_opacity=0.8,
    line_opacity=0.3,
    nan_fill_color="white",
    legend_name="Ecological footprint per capita",
    name="Countries by ecological footprint per capita",
).add_to(m)
folium.LayerControl().add_to(m)
m.save("M7.html")
```

genera el siguiente gráfico:


0
2
3
5
7
16
Ecological footprint per capita
RUSSIA
KAZAKHSTAN
MONGOLIA
CHINA
INDIA
TURKEY
IRAN
SAUDI ARABIA
EGYPT
LIBYA
ALGERIA
SUDAN
NIGERIA
BRAZIL
Indian Ocean
Leaflet | © OpenStreetMap contributors © CartoDB, CartoDB attributions

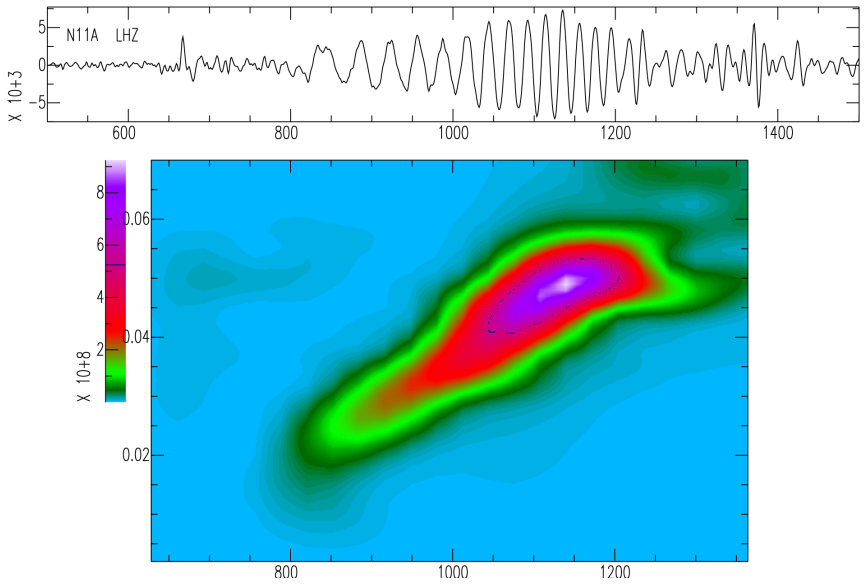
Existe una gran cantidad herramientas para generar gráficos 2D, 3D y videos que se pueden usar desde lenguajes de programación, en particular para Python hay una gran variedad, entre las utilidades destacan:

- https://mpmath.org/
- https://plotly.com/python/
- https://github.com/szabolcsdombi/zengl
- https://towardsdatascience.com/interactive-animated-visualization-db91d1c858ad
- https://docs.enthought.com/mayavi/mayavi/
- https://docs.bokeh.org/en/latest/index.html
- https://matplotlib.org/
- https://pyopengl.sourceforge.net/
- https://pandas.pydata.org/pandas-docs/stable/user_guide/visualization.html
- https://pythonplot.com/
- http://www.pyqtgraph.org/
- http://seaborn.pydata.org/
- https://github.com/sepandhaghighi/samila
- https://github.com/paulcbogdan/NiChord
- https://pypi.org/project/Shapely/
- https://tech.marksblogg.com/pretty-maps-in-python.html
- https://www.manim.community/
- https://plotnine.readthedocs.io/en/stable/
- https://www.pythonguis.com/tutorials/pyqt6-plotting-pyqtgraph/
- https://pbpython.com/visualization-tools-1.html

- https://github.com/moshi4/pyCirclize
- https://mathspp.com/blog/til/xkcd-plots
- https://github.com/ResidentMario/geoplot

5.2 SAC

SAC (Código de análisis sísmico) es un programa interactivo de propósito general diseñado para el estudio de señales secuenciales, especialmente datos de series temporales. Se ha puesto énfasis en las herramientas de análisis utilizadas por los sismólogos investigadores en el estudio detallado de eventos sísmicos. Las capacidades de análisis incluyen operaciones aritméticas generales, transformadas de Fourier, tres técnicas de estimación espectral, filtrado IIR y FIR, apilamiento de señales, diezmado, interpolación, correlación y selección de fase sísmica. SAC también contiene una amplia capacidad gráfica.

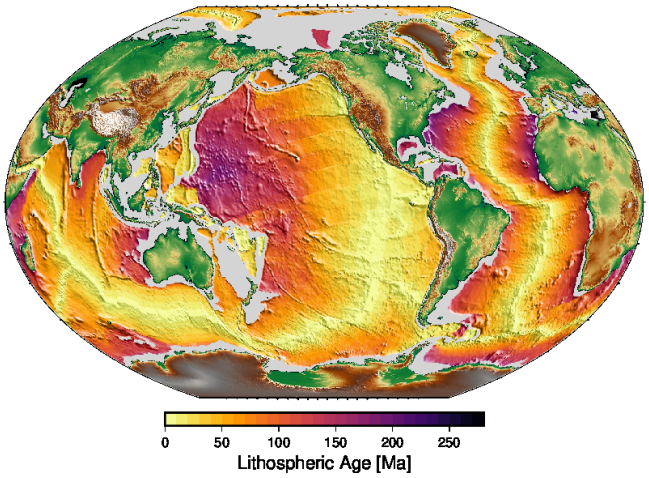


SAC fue desarrollado en el Laboratorio Nacional Lawrence Livermore y tiene derechos de autor de la Universidad de California. Actualmente es desarrollado y mantenido por un pequeño grupo de desarrolladores que trabajan en cooperación con IRIS. Las versiones binarias están disponibles para Intel Mac y Linux, pero SAC se puede construir a partir del código fuente para otros sistemas operativos de computadora. El código fuente está escrito en C.

5.3 Sistema de Información Geográfica

Un sistema de información geográfica (SIG) es un marco para recopilar, gestionar y analizar datos. Adaptado en la ciencia de la geografía, SIG integra muchos tipos de datos. Analiza la ubicación espacial y organiza capas de información en visualizaciones utilizando mapas y escenas en 3D. Con esta capacidad única, SIG revela conocimientos más profundos sobre los datos, como patrones, relaciones y situaciones, lo que ayuda a los usuarios a tomar decisiones más inteligentes.

Uno de los problemas más complejos de resolver cuando se trabaja con SIG, es la obtención de salidas cartográficas de suficiente calidad. No se trata de obtener mapas como los que producen los organismos cartográficos, sino tan sólo de disponer de un sistema lo suficientemente flexible como para crear un mapa que une los objetivos de representación y simplificación, así como los criterios estéticos, que se esperan de un mapa que va a publicarse, como tal o dentro de un trabajo científico.



Las salidas gráficas de un SIG se caracterizan por su inmediatez y por ser una representación para el usuario de los modelos de datos que utiliza el programa, asumiendo a este usuario los suficientes conocimientos de SIG, del área de estudio y de las variables representadas, como para que baste un representación cruda. Sin embargo un mapa es para uso de terceros a los que no debemos suponer ningún conocimiento a priori y que, por tanto, van a necesitar ayuda para interpretar el mapa (escalas, leyendas, etc.). Con-

scientes de este problema, los desarrolladores de SIG han implementado una serie de herramientas de maquetación de mapas.

- Los sistema de información geográfica debierán permitir:
- Crear, compartir y utilizar mapas inteligentes
- Compilar información geográfica
- Crear y administrar bases de datos geográficas
- Resolver problemas con el análisis espacial
- Crear aplicaciones basadas en mapas
- Dar a conocer y compartir información mediante la geografía y la visualización

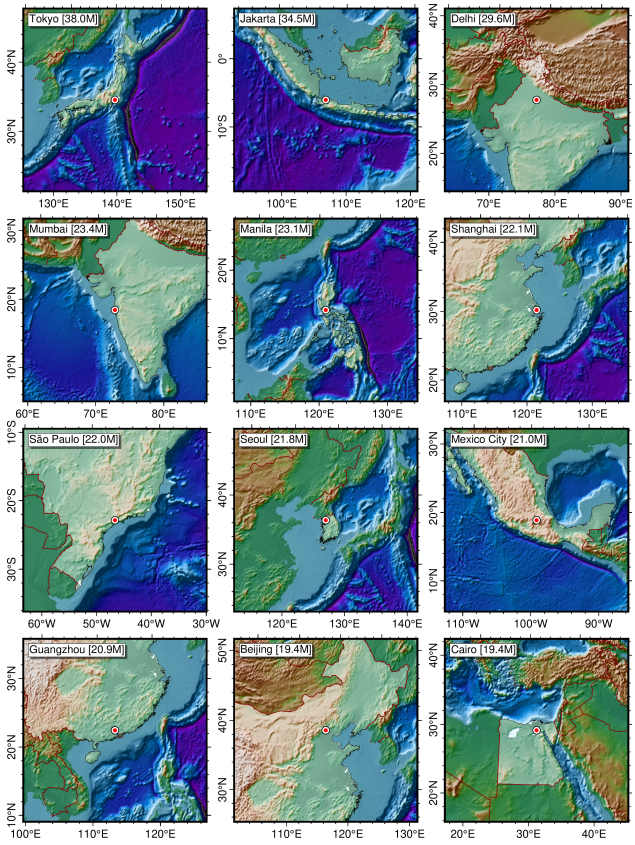
Existe una gran cantidad herramientas para que se pueden usar desde lenguajes de programación, entre las utilidades destacan:

- https://geopy.readthedocs.io/en/stable/

GMT GMT (Generic Mapping Tools) pone a disposición del usuario un conjunto de módulos orientados a la producción de cartografía a partir de datos codificados según los modelos lógicos (Raster, vectorial, lista de puntos) habituales en los SIG. Los datos raster se almacenan por defecto en formato propio (GMT/ netCDF) pero también se admiten ficheros binarios con datos en coma flotante, enteros, bytes o bits. Los datos vectoriales se almacenan como simples pares de coordenadas, no se incluye por tanto información topológica pero de hecho no tiene sentido incluirla si el resultado que espera obtenerse es un mapa para imprimir en papel.

El manejo de estos módulos en linea de comandos y la posibilidad de combinarlos (entre si y con otras herramientas LINUX/UNIX) así como el elevado número de opciones de cada uno de ellos, convierte GMT en un entorno de maquetación de mapas extremadamente flexible. La contrapartida de esta flexibilidad es un lenguaje que puede llegar a ser bastante complejo y hermético.

Además de los módulos estrictamente destinados a la generación de mapas, GMT dispone de módulos de álgebra de mapas y de producción de gráficos más generales (histogramas, etc.). GMT tiene capacidad de trabajar con lenguajes como: MATLAB/Octave, Julia o Python.



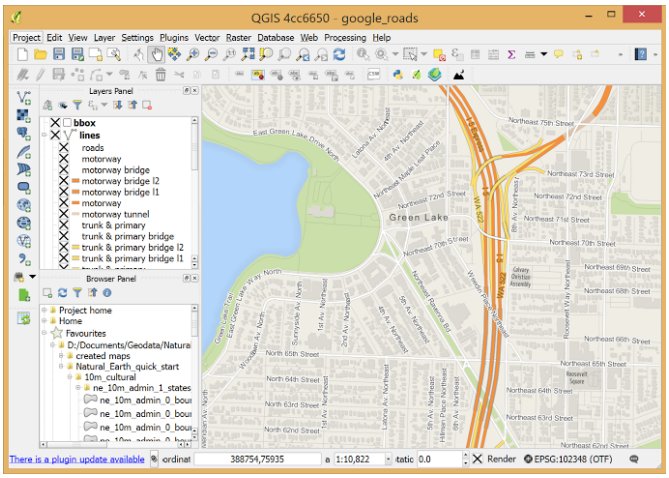
instalación en linux:

```
# apt install gmt gmt-gshhg-full
```

QGIS QGIS es un Sistema de Información Geográfica (SIG) de Código Abierto licenciado bajo GNU - General Public License . QGIS es un proyecto oficial de Open Source Geospatial Foundation (OSGeo). Corre sobre Linux, Unix, Mac OSX, Windows y Android y soporta numerosos formatos y funcionalidades de datos vector, datos ráster y bases de datos.

instalación en linux:

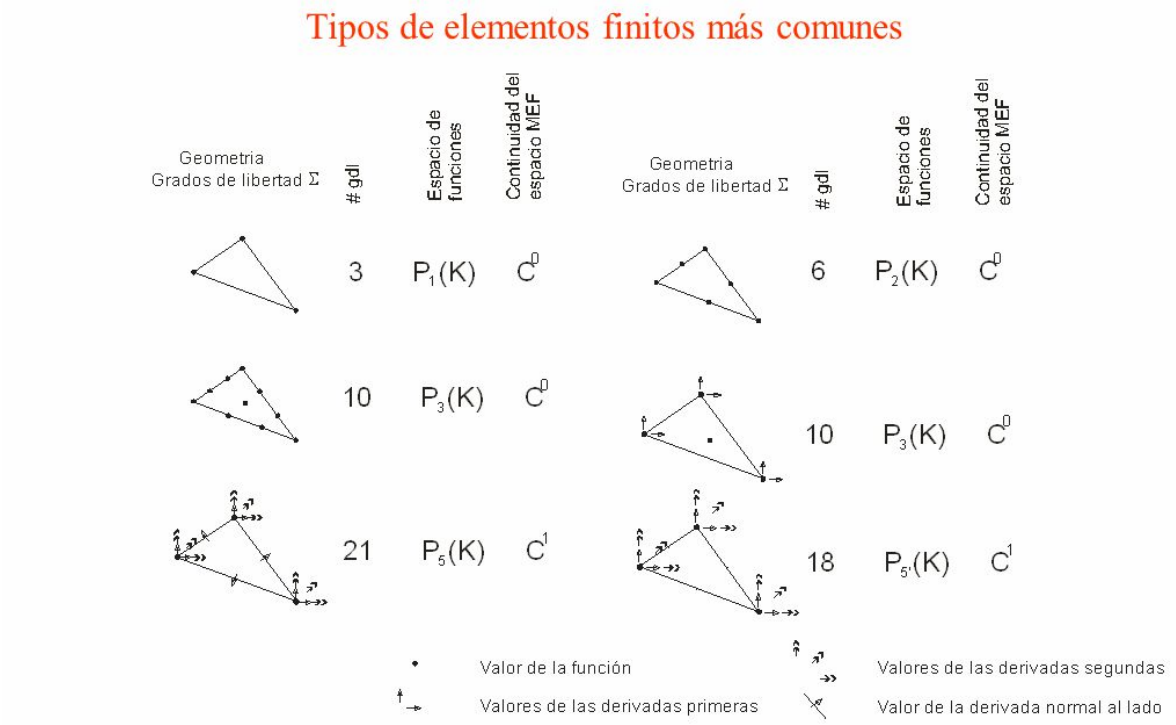
```
# apt install qgis
```



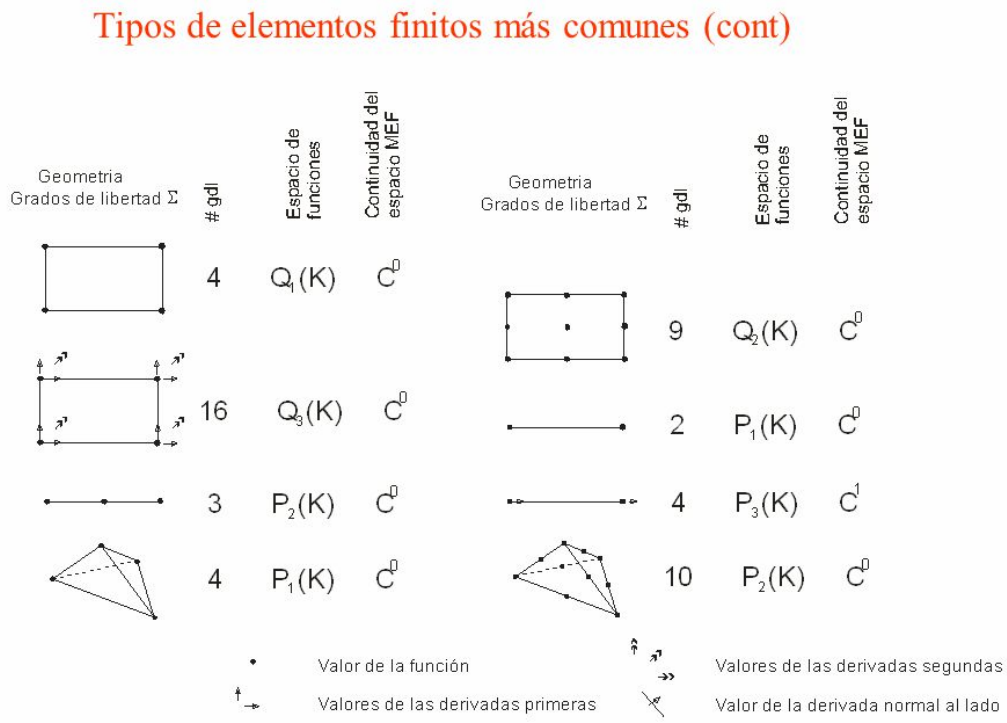
ArcGIS ArcGIS es un completo sistema que permite recopilar, organizar, administrar, analizar, compartir y distribuir información geográfica. Es una plataforma para crear y utilizar sistemas de información geográfica (SIG), ArcGIS es utilizada por personas de todo el mundo para poner el conocimiento geográfico al servicio de los sectores del gobierno, la empresa, la ciencia, la educación y los medios. ArcGIS permite publicar la información geográfica para que esté accesible para cualquier usuario. El sistema está disponible en cualquier lugar a través de navegadores Web, dispositivos móviles como Smartphones y equipos de escritorio.

5.4 Generar Mallas con GMSH

GMSH es un generador para mallar tanto sus superficies como sus volúmenes para Elementos Finitos de código abierto con un motor integrado tipo CAD y un post-procesador. Su diseño tiene como fin proveer una herramienta rápida, ligera y amigable con un editor paramétrico y un gran capacidad de visualización avanzada. GMSH está organizado en cuatro módulos: geometría, malla, motor de cálculo, y post-proceso. La introducción de datos e instrucciones a cualquiera de los módulos puede hacerse ya sea de manera interactiva usando la interfaz gráfica, mediante archivos de texto ASCII empleando el lenguaje propio de GMSH ó a través de la interfaz de programación de aplicaciones C, C++, Python, Julia, Fortran, etc.



Mallado. Se define como mallado la discretización de una línea, superficie o volúmen en porciones de tamaño finito. Las porciones, además de tener un tamaño característico varias veces menor que el espacio discretizado, serán entidades geométricas elementales como los triángulos o cuadriláteros en dos dimensiones o los tetraedros o prismas en tres dimensiones. Esta discretización en estructuras más elementales es esencial para la resolución de ecuaciones en derivadas parciales en dominios arbitrarios por el método de volúmenes finitos (FVM) o el método de elementos finitos (FEM).



El proceso de mallado más habitual es el de discretizar sucesivamente entidades de mayor dimension como si de un problema de contorno se tratara. Por ejemplo, si queremos mallar un cubo primero definiremos los puntos de control en cada uno de sus doce aristas. Luego discretizaremos en porciones elementales cada uno de sus seis lados y finalmente discretizaremos el volúmen. Para entender mejor este proceso, y con anterioridad a entender la sintaxis de los archivos GMSH presentamos este ejemplo de mallado de un cubo con tetraedros. El primer paso es definir los puntos básicos de la geometría.

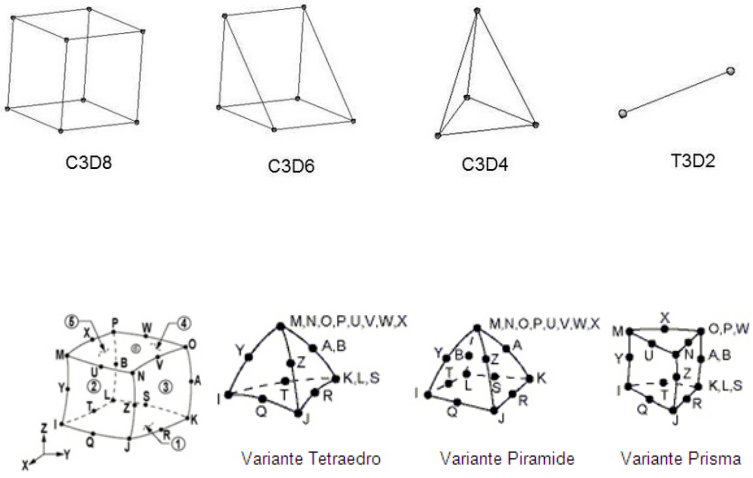


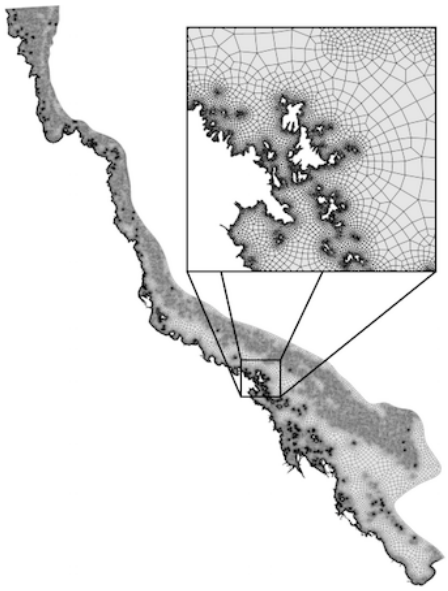
Fig. 3. Geometría del elemento finito sólido estructural

Mallas estructuradas Una malla estructurada es una malla que puede ser transformada, mediante una transformación biyectiva, a una cuadrícula. Esto significa que el problema podría ser resuelto en una malla uniforme y rectangular y devuelto a la geometría original sólo conociendo el jacobiano de la transformación porque este jacobiano se puede invertir.

Mallas no estructuradas Las mallas no estructuradas parten de distribuciones de puntos que cumplen una serie de propiedades dadas, la mayoría de ellas relacionadas con una distribución lo más uniforme posible dentro del contorno. Son adecuadas para geometrías irregulares donde es muy difícil generar una malla estructurada.

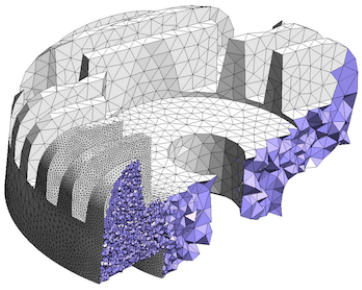
GMSH utiliza un algoritmo de creación de malla basado en la triangulación Delaunay. Dado un contorno se demuestra que sólo existe una triangulación óptima en la que, para cada triángulo formado por tres puntos de la malla, no exista ningún punto dentro de la circunferencia que pasa por los tres puntos.

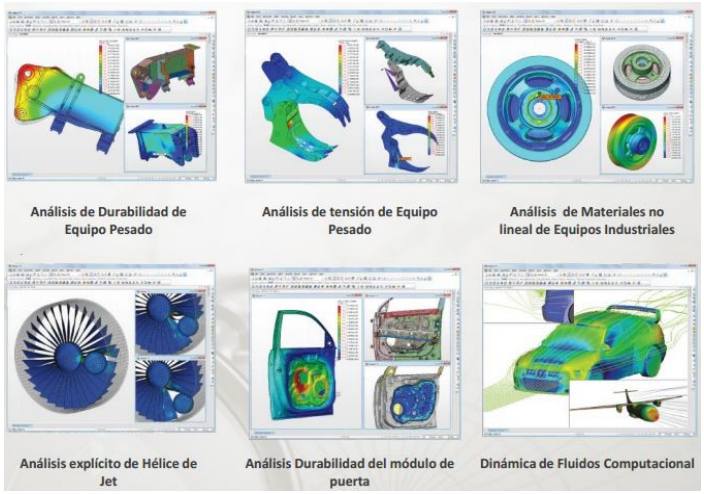
Si bien esta propiedad cierra el problema de la triangulación dados los puntos de la malla, en la mayoría de los casos sólo dispondremos del contorno. GMSH es capaz de encontrar una distribución no estructurada de puntos cuya triangulación resultante tiene suficiente calidad.



Discretización y calidad de malla. Una de las diferencias esenciales entre las mallas estructuradas y no estructuradas es el proceso de refinado. En algunos problemas la malla utilizada no produce suficiente precisión y es necesario rehacerla para aumentarla. En las mallas estructuradas el aumento de precisión no suele suponer un problema más allá de generar demasiados grados de libertad. Sin embargo en las mallas no estructuradas algunos algoritmos de refinado automático pueden funcionar mal.

Uno de los algoritmos más utilizados es el de partir cada uno de los triángulos o tetraedros en dos o más elementos. Este algoritmo recursivo puede refinar puntos arbitrarios en el espacio produciendo una pérdida de





precisión el introducir gradientes espúreos debidos a una mala definición de la geometría.

GMSH se puede usar en conjunto con FreeFem++ que es un software libre que sirve para la resolución numérica de problemas 2D y 3D utilizando el método de elementos finitos. ¿Qué problemas puedes resolver? De mecánica de fluidos, difusión de calor, elasticidad, electromagnetismo, etc. Todos aquellos que estén modelizados matemáticamente a través de una ecuación en derivadas parciales (EDP), por ejemplo:

https://www.um.es/freefem/ff++/pmwiki.php?n=Main.Elasticidad3D

Existe una gran cantidad herramientas para generar mallas en 2D, 3D que se pueden usar desde lenguajes de programación, entre las utilidades destacan:

- https://www.salome-platform.org/
- https://www.paraview.org/
- https://www.meshlab.net/
- https://wias-berlin.de/software/index.jsp?id=TetGen&lang=1
- http://alice.loria.fr/software/geogram/doc/html/index.html

5.5 Adquisición de Datos

En la ciencia y en particular en las ciencias de la tierra la adquisición de datos o adquisición de señales es una importante rama técnica que consiste en la toma de muestras del mundo real (sistema analógico) para generar datos que puedan ser manipulados por una computadora u otros dispositivos electrónicos (sistema digital). Consiste en tomar un conjunto de señales físicas, convertirlas en tensiones eléctricas y digitalizarlas de manera que puedan ser procesadas por una computadora. Se requiere una etapa de acondicionamiento, que adecua la señal a niveles compatibles con el elemento que hace la transformación a señal digital. El elemento que hace dicha transformación es el módulo de digitalización o tarjeta de adquisición de datos (DAQ).

Proceso de Adquisición de Datos Algunas definiciones son:

- Dato: Representación simbólica (numérica, alfabética...), atributo o característica de un valor. No tiene sentido en sí mismo, pero convenientemente tratado (procesado), se puede utilizar en la relación de cálculos o toma de decisiones.
- Adquisición: Recogida de un conjunto de variables físicas, conversión en voltaje y digitalización de manera que se puedan procesar en un ordenador.
- Sistema: Conjunto organizado de dispositivos que interactúan entre sí ofreciendo prestaciones más completas y de más alto nivel.
- Bit de resolución: Número de bits que el convertidor analógico a digital (ADC) utiliza para representar una señal.

- Rango: Valores máximo y mínimo entre los que el sensor, instrumento o dispositivo funcionan bajo unas especificaciones.

Una vez que las señales eléctricas se transformaron en digitales, se envían a través del bus de datos a la memoria de la computadora. Una vez los datos están en memoria pueden procesarse con una aplicación adecuada, archivarlas en el disco, visualizarlas en la pantalla, etc...

Los componentes de los sistemas de adquisición de datos, poseen sensores adecuados que convierten cualquier parámetro de medición de una señal eléctrica[44], que se adquiere por el Hardware de adquisición de datos. Los datos adquiridos se visualizan, analizan, y almacenan en una computadora, ya sea utilizando el proveedor de Software suministrado u otro Software desarrollado para ese caso particular.

Los controles y visualizaciones se pueden desarrollar utilizando varios lenguajes de programación de propósito general como VisualBASIC, C, C++, Python, Fortran, Java, Lisp, Pascal. Los lenguajes especializados de programación utilizados para la adquisición de datos incluyen EPICS, utilizada en la construcción de grandes sistemas de adquisición de datos, LabVIEW, que ofrece un entorno gráfico de programación optimizado para la adquisición de datos, y MATLAB. Estos entornos de adquisición proporcionan un lenguaje de programación además de bibliotecas y herramientas para la adquisición de datos y posterior análisis.

De la misma manera que se toma una señal eléctrica y se transforma en una digital para enviarla a la computadora, se puede también tomar una señal digital o binaria y convertirla en una eléctrica. En este caso el elemento que hace la transformación es una tarjeta o módulo de Adquisición de

[44]Teorema de Nyquist: Al muestrear una señal, la frecuencia de muestreo debe ser mayor que dos veces el ancho de banda de la señal de entrada, para poder reconstruir la señal original de forma exacta a partir de sus muestras. En caso contrario, aparecerá el fenómeno del Aliasing que se produce al infra-muestrear. Si la señal sufre Aliasing, es imposible recuperar el original.

Velocidad de muestreo recomendada

> -2*frecuencia mayor (medida de frecuencia) -10*frecuencia mayor (detalle de la forma de onda)

Si se muestrea al doble de su frecuencia se la puede reconstruir exactamente, lo que no quiere decir que si triplico la frecuencia voy a tener una mejor señal muestreada, para nada.. ya verá el lector alguna aplicación en donde las frecuencias altas resultarán en un problema, por ende no crea que aumentando las pulsaciones va a mejorar la señal, puesto que al menos de forma teórica el teorema no enuncia ni demuestra eso.

Datos de salida, o tarjeta de control. La señal dentro de la memoria de la computadora la genera un programa adecuado a las aplicaciones que quiere el usuario y, luego de procesarla, es recibida por mecanismos que ejecutan movimientos mecánicos, a través de servomecanismos, que también son del tipo transductores.

Un sistema típico de adquisición utiliza sensores, transductores, amplificadores, convertidores analógico - digital (A/D) y digital - analógico (D/A), para procesar información acerca de un sistema físico de forma digitalizada.

En este caso DAQ funciona como transductor de señales análogas a digitales, propiciando así el procesamiento de datos a través de sus funciones de entradas.

¿Cómo se adquieren los datos? La adquisición de datos se inicia con el fenómeno físico o la propiedad física de un objeto (objeto de la investigación) que se desea medir. Esta propiedad física o fenómeno podría ser el cambio de temperatura o la temperatura de una habitación, la intensidad o intensidad del cambio de una fuente de luz, la presión dentro de una cámara, la fuerza aplicada a un objeto, o muchas otras cosas. Un eficaz sistema de adquisición de datos pueden medir todos estas diferentes propiedades o fenómenos.

Un sensor es un dispositivo que convierte una propiedad física o fenómeno en una señal eléctrica correspondiente medible, tal como tensión, corriente, el cambio en los valores de resistencia o condensador, etc. La capacidad de un sistema de adquisición de datos para medir los distintos fenómenos depende de los transductores para convertir las señales de los fenómenos físicos mensurables en la adquisición de datos por Hardware. El término transductores es sinónimo de sensores en sistemas de DAQ. Hay transductores específicos para diferentes aplicaciones, como la medición de la temperatura, la presión, o flujo de fluidos. DAQ también despliega diversas técnicas de acondicionamiento de Señales para modificar adecuadamente diferentes señales eléctricas en tensión, que luego pueden ser digitalizados usando CED.

Las señales pueden ser digitales (también llamada señales de la lógica) o analógicas en función del transductor utilizado.

El acondicionamiento de señales suele ser necesario si la señal desde el transductor no es adecuado para la DAQ Hardware que se utiliza. La señal puede ser amplificada o desamplificada, o puede requerir de filtrado, o un cierre patronal, en el amplificador se incluye para realizar demodulación. Varios otros ejemplos de acondicionamiento de señales podría ser el puente

de conclusión, la prestación actual de tensión o excitación al sensor, el aislamiento, linealización, etc. Este pretratamiento del la señal normalmente lo realiza un pequeño módulo acoplado al transductor.

DAQ Hardware son por lo general las interfaces entre la señal y una computadora. Podría ser en forma de módulos que pueden ser conectados a la computadora de los puertos (paralelo, serie, USB, etc...) o ranuras de las tarjetas conectadas a (PCI, ISA) en la placa madre. Por lo general, el espacio en la parte posterior de una tarjeta PCI es demasiado pequeño para todas las conexiones necesarias, de modo que una ruptura de caja externa es obligatorio. Las tarjetas DAQ a menudo contienen múltiples componentes (multiplexores, ADC, DAC, TTL-IO, temporizadores de alta velocidad, memoria RAM). Estos son accesibles a través de un bus por un microcontrolador, que puede ejecutar pequeños programas. El controlador es más flexible que una unidad lógica dura cableada, pero más barato que una CPU de modo que es correcto para bloquear con simples bucles de preguntas.

El driver (Software) habitualmente viene con el Hardware DAQ o de otros proveedores, y permite que el sistema operativo pueda reconocer el Hardware DAQ y dar así a los programas acceso a las señales de lectura por el Hardware DAQ. Un buen driver ofrece un alto y bajo nivel de acceso.

Tiempo de conversión Es el tiempo que tarda en realizar una medida el convertidor en concreto, y dependerá de la tecnología de medida empleada. Evidentemente nos da una cota máxima de la frecuencia de la señal a medir.

Este tiempo se mide como el transcurrido desde que el convertidor recibe una señal de inicio de "conversión" normalmente llamada Start of Conversión (SOC) hasta que en la salida aparece un dato válido. Para que tengamos constancia de un dato válido tenemos dos caminos:

- Esperar el tiempo de conversión máximo que aparece en la hoja de características.
- Esperar a que el convertidor nos envíe una señal de fin de conversión.

Si no se respeta el tiempo de conversión, en la salida tendremos un valor, que dependiendo de la constitución del convertidor será:

- Un valor aleatorio, como consecuencia de la conversión en curso
- El resultado de la última conversión

La etapa de acondicionamiento de la señal

Con más detalle, en una etapa de acondicionamiento podemos encontrar estas etapas, aunque no todas están siempre presentes:

- Amplificación: Es el tipo más común de acondicionamiento. Para conseguir la mayor precisión posible la señal de entrada debe ser amplificada de modo que su máximo nivel coincida con la máxima tensión que el convertidor pueda leer.

- Aislamiento: Otra aplicación habitual en el acondicionamiento de la señal es el aislamiento eléctrico entre el transductor y la computadora, para proteger al mismo de transitorios de alta tensión que puedan dañarlo. Un motivo adicional para usar aislamiento es el garantizar que las lecturas del convertidor no son afectadas por diferencias en el potencial de masa o por tensiones en modo común.

- Multiplexado: El multiplexado es la conmutación de las entradas del convertidor, de modo que con un solo convertidor podemos medir los datos de diferentes canales de entrada. Puesto que el mismo convertidor está midiendo diferentes canales, su frecuencia máxima de conversión será la original dividida por el número de canales muestreados. Se aconseja que los multiplexores se utilicen antes del conversor y después del condicionamiento del señal, ya que de esta manera no molestará a los aislantes que podamos tener.

- Filtrado: El fin del filtro es eliminar las señales no deseadas de la señal que estamos observando. Por ejemplo, en las señales cuasi-continuas, (como la temperatura) se usa un filtro de ruido de unos 4 Hz, que eliminará interferencias, incluidos los 50/60 Hz de la red eléctrica. Las señales alternas, tales como la vibración, necesitan un tipo distinto de filtro, conocido como filtro Antialiasing, que es un filtro pasabajo pero con un corte muy brusco, que elimina totalmente las señales de mayor frecuencia que la máxima a medir, ya que se si no se eliminasen aparecerían superpuestas a la señal medida, con el consiguiente error.

- Excitación: La etapa de acondicionamiento de señal a veces genera excitación para algunos transductores, como por ejemplos las galgas "extesométricas", "termistores" o "RTD", que necesitan de la misma, bien por su constitución interna, (como el termistor, que es una resistencia variable con la temperatura) o bien por la configuración en

que se conectan (como el caso de las galgas, que se suelen montar en un puente de Wheatstone).

- Linealización: Muchos transductores, como los termopares, presentan una respuesta no lineal ante cambios lineales en los parámetros que están siendo medidos. Aunque la linealización puede realizarse mediante métodos numéricos en el sistema de adquisición de datos, suele ser una buena idea el hacer esta corrección mediante circuitería externa.

Algunos ejemplos de adquisición de datos y los programas respectivos lo podemos encontrar en:

- https://marceluda.github.io/python-para-fisicos/tuto/labo2/05_instrumentacion/

6 Herramientas en Línea de Comandos

En esta sección mostraremos el uso de varios comandos útiles que se pueden usar desde la línea de comandos

6.1 Historia de Comandos

Cada vez que se entra en la terminal y se trabaja, esta es guardada en la historia de comandos, podemos acceder al historial usando las flechas para moverse entre los comandos tecleados o las teclas *Ctrl-r* para buscar mediante una cadena al comando tecleado.

El comando que nos permite ver la historia de comandos teclados es *history*, podemos usarlo mediante:

```
$ history
```

esto nos permite ver los comandos tecleados (según la configuración de *history* en el sistema guarda los últimos mil comandos) y podemos reejecutar alguno usando ! y el número de comando en el historial, por ejemplo:

```
$ !20
```

en caso de que necesitemos ejecutar alguno de los últimos comandos ejecutados, podemos usar !-N, por ejemplo el penúltimo usamos:

```
$ history !-2
```

también podemos borrar un determinado comando del historial usando su número en history:

```
$ history -d 20
```

y podemos borrar la historia mediante:

```
$ history -c
```

pero esto no impide que se siga grabando si continuamos trabajando en la terminal, para borrarla y que no guarde nada de lo que hagamos, usamos:

```
$ set +o history
$ history -cw
```

o forzar el borrado del historial y salir de sesión, mediante:

```
$ cat /dev/null > ~/.bash_history && history -wc && exit
```

Por otro lado, podemos cambiar el formato de visualización para conocer la fecha y hora del comando tecleado, mediante:

```
$export HISTTIMEFORMAT="%d/%m/%y %T "
```

otra opción es:

```
$export HISTTIMEFORMAT="%h/%d  - %H:%M:%S "
```

y ahora podemos ver el historial en el formato solicitado usando:

```
$ history
```

si queremos que sea de forma permanente, debemos agregarlo en ~/.bashrc. Además podemos configurar en ~/.bashrc algunos otros aspectos como:

- cambiar el tamaño máximo del archivo de history, mediante:

  ```
  HISTFILESIZE=50000
  ```

- el número de comando a recordar, usando:

  ```
  HISTSIZE=10000
  ```

  si lo ponemos en cero, se deshabilita el guardado de la historial:

  ```
  HISTSIZE=0
  ```

- decirle a history que no guarde duplicados, usando:

  ```
  HISTCONTROL=ignoredups
  ```

- que no guarde los comandos que inician con espacios, usando:

HISTCONTROL=ignorespace

- que no guarde duplicados e ignore los que inician con espacio:

  HISTCONTROL=ignoreboth

- cambiar el nombre del archivo en que se guarda la historia, que por omisión es ~/.bash_history, usando:

  HISTFILE=nombre

- además podemos ignorar ciertos patrones o comandos de la historia, para ello usamos:

  HISTIGNORE="history"

de este modo *history* no aparecerá, pero si algo como «*history* | *less*» y similares, así que podemos usar un comodín para evitarlo:

HISTIGNORE="history*"

o podemos añadir más comando separándolos por «:», por ejemplo:

HISTIGNORE="history*:echo*:ps*"

- con la configuración por defecto de *history*, usar varias sesiones de forma simultánea supone un problema. Por un lado, el histórico de comandos se guarda cuando finalizamos la sesión, es decir, al hacer un «*exit*» en Bash. Por defecto, al guardar el histórico de sesión, los contenidos del archivo se sobreescriben así que es posible que los comandos ejecutados en una de las sesiones no queden almacenados en el histórico. Para solucionar este problema, podemos decir que añada los comandos en el archivo *history* en lugar de sobreescribirlos:

  shopt -s histappend

- además, podemos especificar que los comandos se almacenen en el archivo de *history* al momento de ser ejecutados en lugar de al finalizar la sesión. De este modo, sesiones simultáneas podrán visualizar en el histórico sus respectivas ejecuciones de comandos[45]:

[45] Debemos verificar antes que la variable de entorno PROMPT_COMMAND no tiene ningún valor asignado, así no lo perderemos al establecer esta configuración. En caso de que lo tenga, podemos concatenar ambos valores separados por «;».

PROMPT_COMMAND="history -a"

Si queremos ver los cambios se apliquen inmediatamente, usamos:

$ source ~/.bashrc

Si deseamos conocer cuáles son los comandos usados y cuantas veces los hemos usado, escribimos:

$ history | awk '{print $2}' | sort | uniq -c | sort -nr

6.2 Alias a Comandos

Me gustaría crear un alias para el comando rm para tener un mensaje de confirmación antes de ejecutar este comando. Entonces podemos crear un alias como este:

$ alias rm='rm -i'

este es un alias temporal y dura hasta que cierras la terminal. Para guardar el alias de forma permanente, es necesario editar el archivo ~/.bashrc y agregar mi alias allí.

La estructura de alias es la siguiente:

alias name=value

alias name="command"

alias name="command arg1 arg2"

alias name="/path/to/Script"

alias name="/path/to/Script.pl arg1"

y podemos eliminarlos mediante:

unalias aliasname

Alias útiles

```
alias ls="ls –color=auto"
alias ll="ls -la"
alias l.="ls -d .* –color=auto"
alias la="ls -Ah"
alias l="ls -Cfh"
alias dir="dir –color=auto"
alias vdir="vdir -color=auto"
alias cd..="cd .."
alias ..="cd .."
alias ...="cd ../../../"
alias ....="cd ../../../../"
alias .....="cd ../../../../"
alias .4="cd ../../../../"
alias .5="cd ../../../../.."
alias grep="grep –color=auto"
alias egrep="egrep –color=auto"
alias fgrep="fgrep -color=auto"
alias diff="colordiff"
alias bc="bc -l"
alias mkdir="mkdir -pv"
alias df="df -H"
alias du="du -ch"
```

Confirmación de la acción

```
alias rm="rm -I –preserve-root"
alias mv="mv -i"
alias cp="cp -i"
alias ln="ln -i"
alias chown=2chown –preserve-root"
alias chmod="chmod –preserve-root"
alias chgrp="chgrp –preserve-root"
```

También podemos crear algunos comando nuevos, como este, que es una combinación de cd y ls, escribiendo en ~/.bashrc, lo siguiente:

```
cs() {
    cd "$@" && ls -a;
}
```

Alias a directorios también podemos hacer aliar a directorios como por ejemplo:

```
alias desk="cd ~/Desktop"
```

pero podemos hacer una función en ~/.bashrc. que guarde los directorios que más usamos, en el archivo ~/.directoriosGuardados, mediante:

```
# Permite guardar una trayectoria del árbol de directorios
guarda() {
    printf "$(pwd)\n" >> ~/.directoriosGuardados;
}
```

y en donde nos interese guardar la trayectoria usamos:

```
$ guarda
```

y con la función goto, nos permita movernos entre las diferentes trayectorias guardadas:

```
    # Permite cambiar entre las trayectorias guardadas por la
función: guarda
    function goto
    {
        local foo=$(sort ~/.directoriosGuardados | nl -w1)
        local REPLY
        echo -e "\n0\tSalir\n$foo\n"
        read -p "Introduzca el numero de directorio: "
        if [[ $REPLY =~^[0-"$(echo "$foo" | wc -l)"]$ ]]; then
            [[ $REPLY = "0" ]] && return
            cd "$(grep ^$REPLY <<<$foo | cut -f2)" && echo
"Ahora en: $(pwd)"
        else
            echo "No existe tal numero"
        fi
    }
```

de esta forma, podremos guardar múltiples trayectorias y cuando deseemos un cambio a una nueva trayectoria usamos:

```
$ goto
```

que nos mostrará las diferentes opciones guardadas y seleccionar usando el número que le corresponda o 0 para salir.

6.3 Ayuda de Comandos y Tipo de Archivos

Muchos comandos que podemos llegar a utilizar se pueden clasificar en categorías de acuerdo a su origen. Algunos están incorporados en el Shell, mientras otros provienen de un determinado paquete que hayamos instalado. También existe la posibilidad de que un comando sea en realidad un *alias* de otro comando con sus opciones.

type permite identificar el comando o comandos pasados como parámetro indicando la trayectoria si es comando externo o si es comando interno al Shell. Uso del comando type:

$ type [*opciones*] comando o comandos

Algunas opciones del comando *type* son:

- -P muestra la trayectoria completa del comando
- -p retorna el nombre del archivo en disco al cual pertenece o nada si no hay archivo
- -a muestra la mayor información posible del comando
- -t retorna el tipo de comando, no la trayectoria

Por ejemplo para el comando creado en la función *cs* de la sección anterior

$ type cs

desplegará:

```
cs is a function
cs() {
      cd "$@" && ls -a;
}
```

man muestra la documentación completa de todos los comandos, su sintaxis es:

$ man [*opciones*] [sección] comando

para conocer todas las páginas disponibles del sistema, usamos:

$ man -k .

por ejemplo, para *clear*:

$ man clear

para una versión corta de la ayuda, usamos:

$ man -f clear

también podemos buscar ayuda de algo que tiene que ver con alguna palabra, por ejemplo algo con "sound", usamos:

$ man -k sound

además podemos usar grep en conjunción con man para una ayuda rápida sobre alguna opción del comando buscado, por ejemplo:

$ man ls | grep – –g

El comando *man* tiene diversas secciones donde se puede solicitar que busque la información de un comando para evitar ambigüedades, estas son:

1. *bin*: Binarios esenciales para el funcionamiento del sistema
2. *sys*: Llamadas al sistema
3. *lib*: Funciones de las bibliotecas
4. *dev*: Archivos de dispositivos
5. *etc*: Archivos de configuración
6. *games*: Juegos

7. *mis*: Miscelánea
8. *sbin*: Binarios esenciales para administración y mantenimiento del sistema
9. *boot*: Información del Kernel del sistema

por ejemplo para conocer *man* en la sección 7, usamos:

$ man 7 man

También es común el uso de *–help* en el comando que necesitamos conocer su uso, por ejemplo:

$ clear –help

help proporciona ayuda de los comandos, con frecuencia puede sustituir al comando *man*. Por ejemplo, para conocer la lista de comandos que soporta:

$ help

info proporciona ayuda de los comandos al igual que *man* y *help*, su uso es similar:

$ info mkdir

pinfo navegador de visualización de información de un comando del sistema, ejemplo:

$ pinfo mkdir

whatis proporciona una ayuda breve de lo que hacen los comandos, sin mostrar opciones del comando, ejemplo:

$ whatis ls

apropos busca en las páginas del manual para la palabra clave o expresión regular que le pasemos como parámetro, ejemplo:

$ apropos chmod

whereis localiza el archivo binario, sus fuentes y las páginas de manual del comando, ejemplo:

```
$ whereis info
```

which sirve para averiguar dónde se encuentra instalado uno o más comandos y para ello busca en los directorios del sistema, ejemplo:

```
$ which chmod ls
```

si se quiere saber todos los lugares en donde esta el comando usamos:

```
$ which -a sync
```

otra opción es:

```
$ command -v sync
```

file determina el tipo de archivo y muestra el resultado. No hace falta que el archivo tenga una extensión para que *file* determine su tipo, pues la aplicación ejecuta una serie de pruebas sobre el mismo para tratar de clasificarlo, ejemplo:

```
$ file un_archivo_de_texto.txt
```

Si se tiene un archivo que contiene una lista de trayectorias a diferentes archivos, podemos pedirle a file que nos de información de dichos archivos, usando:

```
$ file -f listaArchivos.txt
```

Si se tiene un archivo compactado, podemos pedirle a file que nos indique con que fue compactado, usando:

```
$ file -z archivoCompactado
```

6.4 Redireccionando la Entrada y Salida Estándar

Cuando se abre un archivo en Linux, a cada archivo se le asignará un número entero y esta información se almacena en el Kernel. De esta forma el Kernel sabe qué archivos se abren y qué proceso los ha abierto. El número entero asignado es lo que llamamos un descriptor de archivo (en breve FD).

Los procesos pueden abrir archivos a discreción, pero la mayor parte de los procesos esperan a que estén abiertos tres descriptores de archivos (nume-rados 0, 1 y 2) cuando inician. Estos descriptores se conocen como entrada estándar (FD 0 Standar Input "Keyboard"), salida estándar (FD 1 Standard Output "Display Terminal") y error estándar (FD 2 Standard Error "Display Terminal"). Es común que los tres estén abiertos en la terminal del usuario. Así, el programa puede leer lo que el usuario teclea leyendo la entrada estándar, y puede enviar salidas a la pantalla del usuario escribiendo en la salida estándar. El descriptor de archivo de error estándar también esta abierto para escritura, y se usa para los mensajes de error. Podemos ver estos descriptores usando:

```
$ ls -al /dev/std*
```

Standard input la entrada estándar, en inglés *standard input* (mejor conocido como *stdin*) es el mecanismo por el cual un usuario le indica a los programas la información que estos deben procesar. Por omisión, el teclado es la entrada estándar. La entrada estándar representa los datos que necesita una aplicación para funcionar, como por ejemplo un archivo de datos o información ingresada desde la terminal y es representado en la terminal como el tipo 0.

Standard output la salida estándar, en inglés *standard output* (mejor conocido como *stdout*) es el método por el cual el programa puede comunicarse con el usuario. Por omisión, la salida estándar es la pantalla donde se ejecutaron las instrucciones. La salida estándar es la vía que utilizan las aplicaciones para mostrarte información, allí podemos ver el progreso o simplemente los mensajes que la aplicación quiera darte en determinado momento y es representado en la terminal como el tipo 1.

Standard error por último existe un flujo conocido como error estándar, en inglés *standard error output* (mejor conocido como *stderr*) que es

utilizado por las instrucciones para desplegar mensajes de error que surjan durante el transcurso de su ejecución. Al igual que *stdout*, el error estándar será la pantalla donde se procesaron las instrucciones. El error estándar es la forma en que los programas te informan sobre los problemas que pueden encontrarse al momento de la ejecución y es representado en la terminal como el tipo 2.

Operadores de redirección a modo de resumen, indicamos las posibles formas de direccionamiento y los símbolos que se utilizan para lograrlo:

- $>$ Redirecciona stdout hacía un archivo, crea archivo si no existe, sobreescribe si existe.
- $>>$ Redirecciona stdout hacía un archivo, crea archivo si no existe, concatena si existe.
- $<$ Redirecciona stdin desde un archivo. El contenido de un archivo es la entrada del comando.
- 2$>$ Redirecciona stderr hacía un archivo, crea archivo si no existe, sobreescribe si existe.
- 2$>>$ Redirecciona stderr hacía un archivo, crea archivo si no existe, concatena si existe.
- $p>\&q$ Fusiona la salida del flujo p con el flujo q $(p, q \in [0, 1, 2])$.
- $p<\&q$ Fusiona la entrada del flujo p con el flujo q $(p, q \in [0, 1, 2])$.

Otros redireccionamientos que no utilizan descritopres

- $<<$ Conocido como HERE-DOCUMENT o HereDoc
- $<<<$ Conocido como HERE-STRING

Todos estos tipos son representados físicamente como archivos en el sistema, todo en GNU/Linux son archivos. Así, una redirección consiste en trasladar la información de un tipo a otro, por ejemplo de la salida estándar a la entrada estándar o del error estándar a la salida estándar. Esto lo logramos usando el símbolo >. Por ejemplo, para redireccionar la salida de un comando y volcarla a un archivo bastaría con ejecutar:

```
$ ls -la > archivo.txt
```

Sin embargo, cada vez que ejecutemos el comando, el contenido del archivo *archivo.txt* será reemplazado por la salida del comando ls. Si queremos agregar la salida del comando al archivo, en lugar de reemplazarla, entonces ejecutamos:

```
$ ls -la >> archivo.txt
```

Utilizar el comando touch para crear un archivo vació, es una práctica común que también puede realizarse con el operador de redireccionamiento >, mediante:

```
$ > documento1
```

Lo interesante es que, además de la salida estándar, también podemos redireccionar el error estándar y la entrada estándar. Si queremos forzar a que un programa nos imprima en pantalla los errores que consiga durante su ejecución podemos redireccionar el error estándar hacia la salida estándar. Eso lo logramos ejecutando:

```
$ programa 2>&1
```

¿Recuerdan que líneas arriba se comentó que GNU/Linux identifica a cada tipo con un número? Bueno, aquí es donde esos números cobran sentido. El tipo 2 es el error estándar y el tipo 1 es la salida estándar. En los ejemplos previos no tuvimos la necesidad de especificar el tipo 1 porque la terminal lo asume pero pudimos expresarlos explícitamente de la siguiente manera:

```
$ ls -la 1> archivo.txt
$ ls -la 1>> archivo.txt
```

Podemos, por ejemplo, contar las líneas que tiene un archivo redireccionando la entrada estándar de wc hacía un archivo de texto. Así:

```
$ wc < archivo.txt
```

También podemos redireccionar la entrada y salida en el mismo comando, por ejemplo:

```
$ sort < lista_desordenada.txt > lista_ordenada.txt
```

o podemos redireccionar la salida de errores y la salida al mismo archivo, por ejemplo:

```
$ comando > todo.txt 2>&1
```

la otra opción (menos común) es utilizar 1>&2, que indicaría redirecciona la salida del comando stdout hacía donde stderr apunte.

```
$ comando 2> errores.txt 1>&2
```

o redireccionar la salida y salida de error a distintos archivos, por ejemplo::

```
$ comando > salida.txt 2> errores.txt
```

Otra opción para guardar la salida de un comando es usar *logsave*, por ejemplo

```
$ logsave salida.txt ls -al
```

si necesitamos adicionar otras salidas usamos:

```
$ logsave -a salida.txt date
```

siendo posible redireccionar el error, usando:

```
$ logsave salida.txt ls -al /dev/null 2>&1
```

Utilizando << y <<< Es cuando un bloque de texto puede ser redireccionado a un comando o archivo de una manera interactiva. El Here Document funciona indicando un DELIMITADOR que no es más que una palabra o cadena cualquiera que cierra el bloque de texto que se desea redireccionar. Veámoslo con ejemplos:

```
$ wc << fin

> uno dos tres ⟵ de manera interactiva el usuario teclea el
bloque de texto, el delimitador es la palabra "fin""
> cuatro cinco
> seis
> fin
3 6 31
```

El comando wc recibe (stdin) un bloque de texto teclado por el usuario. Indicando el fin del bloque de texto con el delimitador fin. Cuando se recibe este, entonces wc ejecuta su función, al recibir por completo su entrada a examinar, en este caso 3 líneas, 6 palabras, 31 caracteres.

```
$ cat << TERMINA > documento1

uno dos
tres
cuatro cinco
TERMINA

$
$ cat documento1

uno dos
tres
cuatro cinco
```

En este caso el comando *cat* recibe el redireccionamiento de hereDoc con del delimitador TERMINA, cuando encuentra el delimitador deja de recibir entradas y direcciona lo ingresado al archivo "documento1". Si se quisiera agregar o concatenar a "documento1" se utilizará cat << TERMINA >> documento1

La diferencia entre << HERE DOCUMENT y <<< HERE STRING es que en HERE STRING no se pasa un delimitador, sino que se pasa una cadena que es interpretada por el comando al que se le redirecciona como un argumento(s). Este argumento puede ser una variable de Shell que puede expandirse. Es decir, HERE STRING se compone de una cadena o string de posibles argumentos o variables:

```
$ bc <<< 5*5
25
$ sed 's/hola/Hola/' <<< "hola mundo"
Hola mundo
```

Redirección Mediante Pipe las tuberías (pipe) unen la salida estándar de un comando con la entrada estándar de otro, es decir, la salida de un comando se emplea como entrada del siguiente. Para ello se emplea el símbolo pipe "|", su sintaxis es:

```
$ comando1 | comando2 | comando3 | ... | comandoN
```

El orden de los comandos en una tubería es crucial, ya que determina el flujo de datos. Cada comando procesa los datos que recibe del comando anterior y pasa su salida al siguiente comando de la cadena. La utilización de tuberías evita la generación constante de archivos intermedios reduciendo el tiempo de procesamiento.

El siguiente ejemplo lista todos los procesos en ejecución en el sistema mediante el comando "ps -ef". Utilizando el pipe, el resultado viaja como entrada hacia el comando "grep" que se quedará solo con aquellas líneas donde aparezca la palabra "antonio". Este nuevo resultado del comando "grep" se envía como entrada del comando "sort" que se encargará de ordenar el resultado y mostrarlo por la pantalla de la terminal. Es decir que el último comando es el que hace uso de la salida estándar.

$ ps -ef | grep antonio| sort

En este otro ejemplo, el comando "cat" abre el archivo "nombres.dat" y su contenido lo envía, utilizando el pipe, como entrada del comando "cut" el cual recortará las líneas entre los caracteres 10 y 80. Este nuevo resultado, se envía al comando "sort" como entrada de información y haciendo uso de la opción "-r" lo ordenará en orden inverso al orden por defecto y luego muestra el resultado por la pantalla de la terminal.

$ cat nombres.dat | cut -c10-80 | sort -r

Las tuberías se pueden combinar con la redirección de entrada/salida para crear potentes flujos de trabajo de procesamiento de datos. Los operadores > y < le permiten redirigir la salida de un comando a un archivo o recibir información de un archivo, respectivamente.

Redireccionar la salida a un archivo

$ comando1 | comando2 > salida.txt

Tomar la entrada de un archivo

$ comando1 < entrada.txt | comando2

También puede redirigir el error estándar (stderr) 2> si necesita separar los mensajes de error de la salida normal.

$ comando1 2> errores.txt | comando2

A continuación se muestra un ejemplo que combina tuberias, redirección y procesamiento de texto para extraer y formatear información específica de un archivo de registro:

$ grep "error" log.txt | awk '{print $3, $5}' | sort | uniq > Errores_unicos.txt

Este comando:

- Filtra líneas que contienen "error" del log.txt uso grep
- Canaliza la salida a awk, que imprime los campos (columnas) tercero y quinto de cada línea.
- Ordena la salida usando sort
- Elimina líneas duplicadas con uniq
- Redirige la salida final a Errores_unicos.txt

Redirección Mediante el Comando tee existe un comando que permite desviar una copia o bifurcación de la salida de un comando hacia un archivo sin alterar la entrada del siguiente en una tubería. Se trata del comando "tee"

Para entenderlo mejor, en el siguiente ejemplo, se almacena en en el archivo "conectados.dat" el resultado del comando "who" y, sin que este resultado sufra ninguna alteración por parte del comando "tee", se pasa mediante un segundo pipe hacia el comando "wc", quien se encargará de contar las líneas del resultado que produjo el comando "tee".

```
$ who | tee conectados.dat | wc -l
```

otros ejemplos son:

```
$ ps auxww | tee salida.log
$ ls -alh | tee archivo.txt | grep python
$ ls -alh | grep python | tee archivo.txt
$ ls -alh | grep python | tee archivo1.txt archivo2.txt archivo3.txt
$ ls -alh 2>&1 | tee archivo.txt
```

Redirección hacia el dispositivo nulo como se comentó, la salida estándar está asociada al descriptor de archivos 1 y la salida de errores está asociada al descriptor de archivos 2. Si bien para ambas salidas se utiliza el mismo dispositivo físico que es la pantalla de la terminal, ambos se gestionan de manera independiente. Son dos archivos diferentes.

Debes tener en claro que un comando puede generar una salida positiva, que sería aquello que se espera que haga, y eso va a parar a la salida estándar. Pero también puede generar un mensaje de error, y eso va a parar a la salida de errores. En ambos casos, el efecto es verlo reflejado en la pantalla de la terminal, pero internamente están en dos archivos independientes.

Otra cosa a tener en claro es que cuando utilizas las redirecciones de ">" (símbolo mayor) o "|" (pipe), lo que viaja a través de ellas es el resultado de la ejecución de un comando que hubiera ido normalmente a la salida estándar. Los mensajes de errores no viajan a través de las redirecciones a menos que fusiones las salidas.

Pero algo interesante para contar, es que en ocasiones podría no interesarte que los mensajes de errores se muestren en la pantalla de la terminal por lo que es posible reasignar el descriptor de archivos 2 de la salida de errores

estándar hacia un dispositivo que no exista físicamente. A este dispositivo se lo conoce como "dispositivo nulo" y su descriptor está en el directorio /dev.

En el siguiente ejemplo, con el comando "cat" intentaremos abrir el archivo "noexiste.txt" el cual no existe físicamente. En condiciones normales, verías en la pantalla de la terminal el mensaje que indica que "cat" no puede abrir el archivo "noexiste.txt". Este mensaje es un mensaje de error y por consiguiente fue a parar a la salida de errores estándar, en consecuencia lo ves en la pantalla de la terminal.

Pero, mediante la redirección "2>/dev/null", le indicamos al intérprete de comandos que está haciendo la ejecución del comando que todo lo que vaya a parar al descriptor de archivos 2, sea redireccionado hacia el dispositivo nulo "*/dev/nul*l". Y dado que este dispositivo no existe, entonces el mensaje de error se pierde y no se muestra ni almacena en ningún lado.

```
$ cat noexiste.txt 2>/dev/null
```

También podemos hacer algo muy común en la administración de sistemas, descartar el error estándar de un proceso. Para eso ejecutamos:

```
$ programa 2> /dev/null
```

O incluso descartar su salida estándar:

```
$ programa > /dev/null
```

En GNU/Linux, */dev/null* es un archivo especial al que se envía cualquier información que quiera ser descartada. Aunque al principio no lo parezca, el uso del dispositivo nulo es muy útil.

6.5 Metacarácter o Shell Globbing

los metacaracteres o Shell Globbing son caracteres que tienen un significado especial en la línea de comandos, para usar estos caracteres como caracteres ordinarios en la línea de comandos, debes marcarlos de manera especial para que el Shell no los interprete. Por ejemplo:

- La diagonal inversa (\) siempre sirve como carácter de escape para uno o más caracteres que le suceden, lo cual le da a esos caracteres un significado especial. Si un carácter es un metacaracter, el significado

especial es el carácter mismo. Por ejemplo \\ usualmente significa una solo \ y \$ el signo de pesos. En estos casos, la diagonal inversa le quita su significado a los caracteres.

- Cuando un texto se encierra entre comillas (" "), la mayoría de los de los metacaracteres que estén en dicho texto son tratados como caracteres normales, excepto por $, que generalmente indica sustituciones a realizar.

- Otra forma de citar muy similar a la anterior, pero más fuerte es usar comillas sencillas (' ') ya que ni siquiera el carácter $ es interpretado

También existen otros metacaracteres que son comodines que el sistema permite usar para especificar los nombres de archivos que satisfacen el filtro especificado a la hora de buscar, eliminar o filtrar nombres de archivo, estos metacaracteres son: *, ?, [ ] y [^][46].

- ***** Se utiliza para reemplazar cero o más caracteres. Puede ser sustituido por cualquier cadena de caracteres, ejemplos:

  Muestra el contenido de las carpetas que contengan archivos de extensión *txt*:

  ```
  $ ls *.txt
  ```

  Lista todos los archivos que se llamen *archivo* sin importar su extensión:

  ```
  $ ls archivo.*
  ```

  Muestra todos los archivos con extensión *jpg* y que su nombre tenga al final "*chivo*":

  ```
  $ ls *chivo.jpg
  ```

  Muestra todos los archivos que inicien con a y tengan la extensión *png*:

  ```
  $ ls a*.png
  ```

[46] Véase también el uso de las secuencias (véase 6.5).

- **?** Sustituye un carácter cualquiera, ejemplos:

  Muestra todos los archivos empiecen con letras o números pero que luego de ellos tengan los valores "*b4ts.txt*":

  $ ls ?b4ts.txt

  Muestra todos los archivos que inicien con *ab*, siga cualquier letra, número o carácter y finalice con *ts.txt*:

  $ ls ab?ts.txt

  Muestra todos los archivos de tres letras que en medio tenga una letra *i*:

  $ ls ?i?

  Muestra todos los archivos de cuatro letras cualesquiera con extensión *txt*:

  $ ls ????.txt

- [ ] Se usa para definir rangos o conjuntos de caracteres a localizar, para definir los rangos se debe usar el guión -, si son varios caracteres se separan por coma, ejemplos:

  Selecciona el rango de letras mayúsculas del alfabeto [:upper:] o más específicamente [A-Z], muestra todos los archivos que inicien con una letra mayúscula:

  $ ls [A-Z]*

  Selecciona el rango de letras minúsculas del alfabeto [:lower:] o más específicamente [a-z], muestra todos los archivos que inicien con una letra minúscula:

  $ ls [a-z]*

  Selecciona el rango de letras mayúsculas y minúsculas del alfabeto [:alpha:] o más específicamente [a-zA-Z], muestra todos los archivos que inicien con una letra minúscula o mayúscula:

```
$ ls [a-zA-Z]*
```

Selecciona el rango de dígitos [:digit:] o más específicamente [0-9], muestra todos los archivos que inicien con un dígito:

```
$ ls [0-9]*
```

Selecciona el rango de letras mayúsculas, minúsculas y dígitos del alfabeto [:alnum:] o más específicamente [a-zA-Z0-9], muestra todos los archivos que inicien con una letra mayúscula, minúscula o dígito:

```
$ ls [:alnum:]*
```

Selecciona el rango de caracteres de puntuación del alfabeto [:puncht:] o más específicamente " ! # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] _ ´ { | } ~ ", muestra todos los archivos que inicien con un carácter de control:

```
$ ls [:puncht:]*
```

Muestra todos los archivos que comiencen por *z* o *v* sin importar la extensión:

```
$ ls [zv]*
```

Muestra todos los archivos que comiencen por z o v y terminen con la extensión *.txt*:

```
$ ls [zv]*.txt´
```

Lista todos los archivos de cualquier extensión que tengan los rangos establecidos entre los corchetes:

```
$ ls archivo[12].*
```

Muestra la lista de todos los archivos que cumplan con el rango de "a-f" sin importar la extensión o el nombre:

```
$ ls [a-f]*
```

Muestra la lista de todos los archivos que inicien con cualquier cosa, pero que terminen con una letra mayúscula:

$ ls *[A-Z]

Muestra la lista de todos los archivos que inicien con una letra minúscula, tenga después una letra mayúscula, continué con cualquier carácter, después tenga una letra a, b, c-f, z y siga con cualquier cantidad de caracteres:

$ ls [a-z][A-Z]?[a,b,c-f,z]*

- [^] Este caso es contrario al anterior, este representa que se busque algo exceptuando lo que se encuentra entre los corchetes, también trabaja con rangos.

  Muestra los archivos que no empiecen con una letra minúscula pero que tengan extensión *.txt*:

  $ ls [^a-z]*.txt

- [!] Este caso igual al anterior, este representa que se busque algo exceptuando lo que se encuentra entre los corchetes, también trabaja con rangos.

  Muestra los archivos que no empiecen con una letra minúscula pero que tengan extensión *.txt*:

  $ ls [!a-z]*.txt

Secuencias Como parte del BASH podemos usar el generador de secuencias, como por ejemplo:

$ echo {1..10}

que muestra la secuencia de los números de 1 a 10, si ahora probamos:

$ echo {1..10..2}

visualizará los números $1, 3, 5, 7, 9$. Es decir, iniciará con el primer número en la secuencia y terminará en el segundo de la secuencia con incrementos del último número de la secuencia. También podemos hacerlo en orden inverso mediante:

```
$ echo {10..1..2}
```

que nos entregará los números $10, 8, 6, 4, 2$. También podemos usar relleno con ceros, por ejemplo:

```
$ echo {000..121..2}
```

el cual imprimirá los números de 0 a 121 con saltos de dos en dos, como: 000 002 004 006 ... 050 052 054 ... 116 118 120. Y si necesitamos números negativos, podemos usar algo como:

```
$ echo {3..-4}
```

también podemos usarlos para trabajar secuencias con letras, por ejemplo:

```
$ echo {a..z}
```

imprimirá las letras a hasta la z, o este otro ejemplo:

```
$ echo {n..z} {a..m}
```

imprimirá las letras n a la z, y a continuación de a a la m.

Con este generador de secuencias numéricas podemos aplicar a comandos, como por ejemplo:

```
$ mkdir {2009..2019}_Facturas
```

que creará los directorios de 2009_Facturas, hasta 2019_Facturas. También lo podemos usar para borrar archivos, como por ejemplo:

```
$ rm cuadros_{043..61..3}
```

Este generador de secuencias podemos usarlo también en modo texto, por ejemplo:

```
$ touch archivo_{a..z}.txt
```

creará el archivo *archivo_ a.txt* hasta *archivo_ z.txt*. También es posible usar algo como {Z..a} pero generará caracteres no alfanuméricos. Otros usos son:

```
$ touch {blahg, splurg, mmmf}_file.txt
```

creará los archivos *blahg_file.txt, splurg_file.txt* y *mmmf_file.txt*.
Si queremos hacer:

```
$ cp -v file1.txt file1.txt.bak
```

lo podemos lograr, usando:

```
$ cp -v file1.txt{,.bak}
```

Otros ejemplos:

```
$ cp archivo{,.bak}
$ ls {????.txt, *.php}
$ ls {*.txt. *.php}
$ mv proyecto/{nuevo/viejo}/dir/otrodir
$ rm d{01..20}/archivo
$ touch {a,b,c}.{rs,cpp}
$ git add {main,x{1,2}}.rs
$ ls -l ~/{Downloads,Pictures}/*.{pdf,png}
$ mkdir -p ~/test/{etc/x1,lib,usr/{x2,x3},bin,tmp/{Y1,Y2,Y3/z},opt,var}
```

6.6 Nombres de Archivos y Directorios con Caracteres Especiales

Una de las preguntas más obvias aquí es: ¿quién crea/trata con archivos/nombres de directorios que tienen un (#), un punto y coma (;), un guión (-) o cualquier otro carácter especial?

Estoy de acuerdo con usted en que dichos nombres de archivos no son comunes, pero su Shell no debería fallar o darse por vencido cuando tenga que lidiar con dichos nombres. También hablando técnicamente, todo, ya sea una carpeta, un controlador o cualquier otra cosa, se trata como un archivo en Linux.

Nombre de Archivo que Inicia con un Guión en Linux, los nombres de archivos que comienzan con un guión ("-") a veces pueden causar problemas al trabajar con ellos porque el guión inicial puede malinterpretarse como una opción o ser marcado por las utilidades de línea de comandos.

Crear Archivo que Inicia con un Guión si deseamos cree un archivo que comience con un guión (-), digamos -abx.txt usando el comando:

$ touch -abc.txt

el sistema nos mandará un error. El motivo del error es que el Shell interpreta cualquier cosa después de un guión (-) como una opción y, obviamente, no existe tal opción, de ahí el error. Para resolver tal error, tenemos que decirle al Shell que no interprete nada después del carácter especial (aquí guión), como una opción.

Hay dos formas de resolver este error como:

```
$ touch - -    -abc.txt
$ touch ./-abc.txt
```

Podemos verificar que el archivo así creado mediante las dos formas anteriores ejecutando los comandos ls o ls -l para obtener un listado largo.

Editar Archivo que Inicia con un Guión para editar un archivo con un nombre de archivo con guiones, podemos utilizar varios editores de texto disponibles. Sin pérdida de generalidad, aquí hay un ejemplo usando el editor de texto nano:

```
$ nano - -   -abc.txt
$ nano ./-abc.txt
```

Cambiar Nombre Archivo que Inicia con un Guión para cambiar el nombre de un archivo que inicia con guiones, puede usar el comando mv como se muestra. Por ejemplo, para cambiar el nombre de un archivo llamado "-abc.txt" a "-a.txt", usaría:

```
$ mv - -   -abc.txt -a.txt
$ mv ./-abc.txt ./-a.txt
```

Eliminar Archivo que Inicia con un Guión para eliminar un archivo con un nombre de archivo discontinuo, puede usar el comando rm como se muestra.

```
$rm - -   -abc.txt
$ rm ./-abc.txt
```

Si tiene muchos archivos en una carpeta cuyo nombre inicia con guión y desea eliminarlos todos a la vez, podemos hacer lo siguiente:

```
$ rm ./-*
```

Se sigue la misma regla comentada anteriormente para cualquier número de guiones en el nombre del archivo y su aparición. A saber, -a-b-c.txt, ab-c.txt, abc-.txt, etc.

Se sigue la misma regla que se analizó anteriormente para el nombre de la carpeta que tiene cualquier número de guiones y su aparición, excepto el hecho de que para eliminar la carpeta debe usar 'rm -rf' como:

```
$rm -rf - -   -abc
$ rm -rf ./-abc
```

Nombre de Archivo con "#" en Linux, el carácter "#" no está restringido ni reservado para ningún propósito específico en los nombres de archivos. Puede utilizar el carácter "#" como cualquier otro carácter alfanumérico en un nombre de archivo. Sin embargo, vale la pena mencionar que algunos caracteres especiales, incluido el "#", tienen significados especiales en ciertos contextos o cuando se usan con comandos o utilidades específicas.

Por ejemplo, el carácter "#" se usa comúnmente en Scripts de Shell para indicar comentarios. Si está escribiendo un Script de Shell y desea incluir el carácter "#" en un nombre de archivo, es recomendable utilizar caracteres de escape o comillas correctamente para evitar cualquier interpretación no deseada como comentario.

Crear un Archivo con "#" si deseamos cree un archivo que comience con un guión (#), digamos #abx.txt usando el comando:

```
$ touch #abc.txt
```

el sistema nos mandará un error. El motivo del error anterior es que Shell interpreta #abc.txt como un comentario y, por lo tanto, lo ignora. Entonces, el comando touch se pasó sin ningún operando de archivo y, por lo tanto, surge el error.

Para resolver dicho error, puede pedirle al Shell que no interprete # como un comentario.

```
$ touch ./#abc.txt
$ touch '#abc.txt'
```

y podemos verificar el archivo recién creado con ls.

Crear archivo Archivo con "#" ahora cree un archivo cuyo nombre contenga # en cualquier lugar excepto al principio.

```
$ touch ./a#bc.txt
$ touch ./abc#.txt
$ touch 'a#bc.txt'
$ touch 'abc#.txt'
```

¿Qué sucede cuando crea dos archivos (digamos a y #bc) a la vez?

```
$ touch un.txt #bc.txt
```

Obviamente, en el ejemplo anterior solo se creó el archivo 'a' y el archivo '#bc' se ignoró. Para ejecutar la situación anterior con éxito podemos hacer:

```
$ touch a.txt ./#bc.txt
$ touch un.txt '#bc.txt'
```

finalmente, podemos verificar el archivo que acaba de ser creado usando el comando ls.

Cambiar el Nombre o Copiar el Archivo con "#" para cambiar el nombre de un archivo con "#", puede usar el comando mv como se muestra:

```
$ mv ./#bc.txt ./#cd.txt
$ mv '#bc.txt' '#cd.txt'
```

Para copiar un archivo con "#", puede usar el comando cp como se muestra:

```
$ cp ./#cd.txt ./#de.txt
$ cp '#cd.txt' '#de.txt'
```

Editar o eliminar Archivo con "#" para editar un archivo con "#", puede usar el editor nano o vim como se muestra:

```
$ vi ./#cd.txt
$ vi '#cd.txt'
```

Para eliminar un archivo con "#", puede usar el comando rm como se muestra:

```
$ rm ./#bc.txt
$ rm '#bc.txt'
```

Para eliminar todos los archivos que tienen (#) en el nombre del archivo, puede usar:

```
$ rm ./#*
```

Nombre de Archivo con ";" en caso de que no lo sepas, el punto y coma actúa como separador de comandos en BASH y quizás también en otro Shell. El punto y coma le permite ejecutar varios comandos a la vez y actúa como separador. Cree un archivo que tenga un punto y coma:

```
$ toque ./';abc.txt'
$ toque ';abc.txt'
```

Nota: Hemos incluido el nombre del archivo entre comillas simples ' '. Le dice a BASH que; es parte del nombre del archivo y no un separador de comandos. El resto de la acción (es decir, copiar, mover, eliminar) en el archivo y la carpeta que tiene un punto y coma en su nombre se puede realizar directamente encerrando el nombre entre comillas simples.

Caracteres especiales en Nombres de Archivos en Linux, los nombres de archivos pueden contener la mayoría de los caracteres especiales, incluidos espacios, puntos, guiones, guiones bajos e incluso símbolos como @,!, $ y %.

Aquí hay una lista de caracteres especiales que debe usar con precaución o considerar utilizar como escape cuando los use en nombres de archivos:

Signo (+) en el Nombre del Archivo para crear un archivo con signo (+), podemos usar el comando táctil junto con el nombre del archivo entre comillas simples:

```
$ touch '+tony.txt'
```

Signo ($) en Nombre de Archivo para crear un archivo con el signo de ($) , podemos usar el comando táctil junto con el nombre del archivo entre comillas simples:

```
$ touch '$tony.txt'
```

Porcentaje (%) en Nombre de Archivo para crear un archivo de signo (%), podemos utilizar:

```
$ touch '%tony.txt'
```

Asterisco (*) en el Nombre del Archivo para crear un archivo con signo (*) , podemos utilizar.

```
$ touch '*tony.txt'
```

Nota: Cuando tenga que eliminar un archivo que comienza con *, nunca utilice los siguientes comandos para eliminar dichos archivos.

```
$ rm *
$rm -rf *
```

En su lugar, utilizar:

```
$ rm ./*.txt
```

Signo de (!) en el Nombre del Archivo simplemente incluya el nombre del archivo entre comillas simples y el resto de las cosas serán iguales:

```
$ touch '!tony.txt'
```

Signo de (@) en Nombre de Archivo nada adicional, trate un nombre de archivo que tenga el signo @ como un archivo normal:

```
$ touch '@tony.txt'
```

Signo de (^) en Nombre de Archivo no se requiere atención adicional; use un archivo que tenga ^en el nombre del archivo como un archivo normal.

```
$ touch '^tony.txt'
```

Signo (&) en Nombre de Archivo el nombre del archivo debe estar entre comillas simples y estará listo para comenzar:

```
$ touch '&tony.txt'
```

Paréntesis () en Nombre de Archivo si el nombre del archivo tiene paréntesis, debe encerrarlo entre comillas simples:

```
$ touch '(tony.txt)'
```

Signo de llaves ({}) en nombre de archivo no se necesita cuidado adicional. Simplemente trátelo como un archivo más:

```
$ touch {tony.txt}
```

Signo de (<>) en el nombre de archivo un nombre de archivo que tenga (<>) debe estar entre comillas simples:

```
$ touch '<tony.txt>'
```

Signo de ([]) en Nombre de Archivo trate los nombres de archivos que tienen corchetes como archivos normales y no necesitará prestarles especial atención:

```
$ touch [tony.txt]
```

Signo (_) en Nombre de Archivo son muy comunes y no requieren nada extra. Simplemente haga lo que habría hecho con un archivo: normal.

```
$ touch _tony.txt
```

Signo (=) en Nombre de Archivo Tener un signo igual no cambia nada, puedes usarlo como un archivo normal:

```
$ touch =tony.txt
```

Signo (\) en Nombre de Archivo la barra invertida le dice al Shell que ignore el siguiente carácter. Debe encerrar el nombre del archivo entre comillas simples, como hicimos en el caso del punto y coma. El resto de las cosas son sencillas:

```
$ touch '\tony.txt'
```

Signo (/) en Nombre de Archivo no puede crear un archivo cuyo nombre incluya una barra diagonal (/), hará que el sistema de archivos tenga un error. No hay forma de escapar de una barra diagonal.

Signo (?) en Nombre del Archivo nuevamente, un ejemplo en el que no es necesario hacer ningún intento especial. Un nombre de archivo que tiene un signo de interrogación se puede tratar de la forma más general:

```
$ touch '?tony.txt'
```

Signo (.) en Nombre del Archivo los archivos que comienzan con un punto (.) son muy especiales en Linux y se llaman archivos de puntos. Son archivos ocultos generalmente archivos de configuración o de sistema. Debe utilizar el modificador '-a' o '-A' con el comando ls para ver dichos archivos.

Crear, editar, cambiar el nombre y eliminar dichos archivos es sencillo.

```
$ touch '.tony.txt'
```

Nota: En Linux, puede tener tantos puntos (.) como necesite en un nombre de archivo. A diferencia de otros puntos del sistema en el archivo, los nombres no significan separar nombres y extensiones. Puede crear un archivo que tenga varios puntos como:

```
$ touch 1.2.3.4.5.6.7.8.9.10.txt
```

Signo (,) en Nombre de Archivo puedes tener una coma en un nombre de archivo, tantos como quieras y no necesitas nada adicional. Simplemente hágalo de la manera normal, como el nombre de archivo simple:

```
$ touch ',tony.txt'
$ touch ',tony,.txt'
```

Signo (:) en Nombre de Archivo puede tener dos puntos en un nombre de archivo, tantos como desee y no necesita nada adicional. Simplemente hágalo de la manera normal, como el nombre de archivo simple:

```
$ touch :12.txt
$ touch ':tony:.txt'
```

Signo (" ") en Nombre de Archivo para tener comillas en el nombre del archivo, debemos utilizar la regla de intercambio. Es decir, si necesita tener una comilla simple en el nombre del archivo, encierre el nombre del archivo entre comillas dobles y si necesita tener una comilla doble en el nombre del archivo, enciérrela con una comilla simple.

```
$ touch "15'.txt"
$ touch 'tony".txt'
```

Signo (~) en Nombre de Archivo Algunos editores como emacs, crean un archivo de copia de seguridad del archivo que se está editando. El archivo de copia de seguridad tiene el nombre del archivo original más una tilde al final del nombre del archivo. Puedes tener un archivo cuyo nombre incluya una tilde, en cualquier ubicación simplemente como:

```
$ touch '~tony.txt'
$ touch 'anusha~.txt'
```

Espacio en Blanco en el Nombre del Archivo cree un archivo con el nombre que tenga espacios entre caracteres/palabras, diga "hola, mi nombre es tony.txt". No es una buena idea tener nombres de archivos con espacios y si tiene que distinguir un nombre legible, debe usar un guión bajo o un guión. Sin embargo, si tiene que crear un archivo de este tipo, debe utilizar una barra invertida que ignore el siguiente carácter. Para crear el archivo anterior tenemos que hacerlo de esta manera:

```
$ touch hola\ mi\ nombre\ es\ tony.txt
```

He intentado cubrir todos los escenarios con los que te puedas encontrar. La mayoría de las implementaciones anteriores son explícitamente para BASH Shell y es posible que no funcionen en otros Shells.

6.7 Permisos de Archivos y Directorios

GNU/Linux, al ser un sistema diseñado fundamentalmente para trabajo en red, la seguridad de la información que almacenemos en nuestros equipos (y no se diga en los servidores) es fundamental, ya que muchos usuarios tendrán o podrán tener acceso a parte de los recursos de Software (tanto aplicaciones como información) y Hardware que están gestionados en estos equipos de cómputo. ¿Ahora podemos ver por qué la necesidad de un sistema de permisos?

En GNU/Linux, los permisos o derechos que los usuarios pueden tener sobre determinados archivos contenidos en él se establecen en tres niveles claramente diferenciados. Estos tres niveles son los siguientes:

- Permisos del propietario.

- Permisos del grupo.
- Permisos del resto de usuarios (o también llamados "los otros").

Para tener claros estos conceptos, en los sistemas en red siempre existe la figura del administrador, superusuario o root. Este administrador es el encargado de crear y dar de baja a usuarios, así como también, de establecer los privilegios que cada uno de ellos tendrá en el sistema. Estos privilegios se establecen tanto para el directorio de trabajo (Home) de cada usuario como para los directorios y archivos a los que el administrador decida que el usuario pueda acceder.

Permisos del propietario el propietario (user ID, UID) es aquel usuario que genera o crea un archivo/carpeta dentro de su directorio de trabajo, o en algún otro directorio sobre el que tenga derechos. Cada usuario tiene la potestad de crear, por defecto, los archivos que quiera dentro de su directorio de trabajo. En principio, él y solamente él será el que tenga acceso a la información contenida en los archivos y directorios que hay en su directorio trabajo o Home -bueno, no es del todo cierto esto, ya que el usuario *root* siempre tiene acceso a todos los archivos y directorios del sistema-.

Permisos del grupo lo más normal es que cada usuario pertenezca a un grupo de trabajo (group ID, GID). De esta forma, cuando se gestiona un grupo, se gestionan todos los usuarios que pertenecen a éste. Es decir, es más fácil integrar varios usuarios en un grupo al que se le conceden determinados privilegios en el sistema, que asignar los privilegios de forma independiente a cada usuario.

Permisos del resto de usuarios por último, también los privilegios de los archivos contenidos en cualquier directorio, pueden tenerlos otros usuarios que no pertenezcan al grupo de trabajo en el que está integrado el archivo en cuestión. Es decir, a los usuarios que no pertenecen al grupo de trabajo en el que está el archivo, pero que pertenecen a otros grupos de trabajo, se les denomina resto de usuarios del sistema.

¿cómo puedo identificar todo esto? sencillo, abre una terminal y realiza lo siguiente:

```
$ id
```

el comando *id* nos muestra nuestro UID, GID y los grupos a los que pertenecemos, podemos usar:

```
$ id antonio
```

para que nos muestra el UID, GID y los grupos a los que pertenece el usuario antonio.

Para conocer los permisos de nuestros archivos, podemos usar:

```
$ ls -l
```

entregará una salida como esta:

```
-rwxrwxr– 1 antonio ventas 9090 sep 9 14:10 presentacion
-rw-rw-r– 1 antonio antonio 2825990 sep 7 16:36 reporte1
drwxr-xr-x 2 antonio antonio 4096 ago 27 11:41 videos
```

Veamos por partes el listado, tomando como ejemplo la primera línea. La primera columna (-rwxrwxr–) es el tipo de archivo y sus permisos, la siguiente columna (1) es el número de enlaces al archivo, la tercera columna (antonio) representa al propietario del archivo, la cuarta columna (ventas) representa al grupo al que pertenece al archivo y las siguientes son el tamaño, la fecha y hora de última modificación y por último el nombre del archivo o directorio.

El primer carácter al extremo izquierdo, representa el tipo de archivo, los posibles valores para esta posición son los siguientes:

- \- Archivo
- d Directorio
- b Archivo de bloques especiales (Archivos especiales de dispositivo)
- c Archivo de caracteres especiales (Dispositivo tty, impresora...)
- l Archivo de vínculo o enlace (soft/symbolic link)

- p Archivo especial de cauce (pipe o tubería)

Los siguientes 9 restantes, representan los permisos del archivo y deben verse en grupos de 3 y representan:

- - Sin permiso
- r Permiso de lectura
- w Permiso de escritura
- x Permiso de ejecución

Los tres primeros representan los permisos para el propietario del archivo. Los tres siguientes son los permisos para el grupo del archivo y los tres últimos son los permisos para el resto del mundo u otros.

Las nueve posiciones de permisos son en realidad un bit que o está encendido (mostrado con su letra correspondiente) o está apagado (mostrado con un guión -), así que, por ejemplo, permisos como rwxrw-r–, indicaría que los permisos del propietario (rwx) puede leer, escribir y ejecutar el archivo, el grupo (o sea los usuarios que estén en mismo grupo del archivo) (rw-) podrá leer y escribir pero no ejecutar el archivo, y cualquier otro usuario del sistema (r–), solo podrá leer el archivo, ya que los otros dos bits de lectura y ejecución no se encuentran encendidos o activados.

Permisos para archivos:

- Lectura: permite, fundamentalmente, visualizar el contenido del archivo.
- Escritura: permite modificar el contenido del archivo.
- Ejecución: permite ejecutar el archivo como si de un programa ejecutable se tratase.

Permisos para directorios:

- Lectura: Permite saber qué archivos y directorios contiene el directorio que tiene este permiso.
- Escritura: permite crear archivos en el directorio, bien sean archivos ordinarios o nuevos directorios. Se pueden borrar directorios, copiar archivos en el directorio, mover, cambiar el nombre, etc.

- Ejecución: permite situarse sobre el directorio para poder examinar su contenido, copiar archivos de o hacia él. Si además se dispone de los permisos de escritura y lectura, se podrán realizar todas las operaciones posibles sobre archivos y directorios.

Observación 1 *Si no se dispone del permiso de ejecución, no podremos acceder a dicho directorio (aunque utilicemos el comando "cd"), ya que esta acción será denegada. También permite delimitar el uso de un directorio como parte de una ruta (como cuando pasamos la ruta de un archivo que se encuentra en dicho directorio como referencia. Supongamos que queremos copiar el archivo "X.ogg" el cual se encuentra en la carpeta "/home/antonio/Z" -para lo cual la carpeta "Z" no tiene permiso de ejecución-, haríamos lo siguiente:*

$ cp /home/antonio/Z/X.ogg /home/antonio/Y/

obteniendo con esto un mensaje de error diciéndonos que no tenemos los permisos suficientes para acceder al archivo. Si el permiso de ejecución de un directorio está desactivado, se podrá ver su contenido (si se cuenta con permiso de lectura), pero no se podrá acceder a ninguno de los objetos contenidos en él, pues para ello este directorio es parte del camino necesario para resolver la ubicación de sus objetos.

Permisos en formato numérico octal La combinación de valores de cada grupo de los usuarios forma un número octal, el bit x es 20 es decir 1, el bit w es 21 es decir 2, el bit r es 22 es decir 4, tenemos entonces:

- r = 4
- w = 2
- x = 1

La combinación de bits encendidos o apagados en cada grupo da ocho posibles combinaciones de valores, es decir la suma de los bits encendidos:

$---$	$= 0$	no se tiene ningún permiso
$--x$	$= 1$	solo permiso de ejecución
$-w-$	$= 2$	solo permiso de escritura
$-wx$	$= 3$	permisos de escritura y ejecución
$r--$	$= 4$	solo permiso de lectura
$r-x$	$= 5$	permisos de lectura y ejecución
$rw-$	$= 6$	permisos de lectura y escritura
rwx	$= 7$	todos los permisos establecidos

Cuando se combinan los permisos del usuario, grupo y otros, se obtienen un número de tres cifras que conforman los permisos del archivo o del directorio. Esto es más fácil visualizarlo con algunos ejemplos:

Estableciendo Permisos con el Comando chmod habiendo entendido lo anterior, es ahora fácil cambiar los permisos de cualquier archivo o directorio, usando el comando chmod (change mode), cuya sintaxis es la siguiente:

chmod [opciones] permisos archivo[s], algunos ejemplos:

```
$ chmod 755 reporte1
$ chmod 511 respaldo.sh
$ chmod 700 julio*
$ chmod 644 *
```

Los ejemplos anterior establecen los permisos correspondientes que el usuario propietario desea establecer, el tercer ejemplo (chmod 700 julio*) cambiará los permisos a todos los archivos que empiecen con julio (julio01, julio02, julio_respaldo, etc.) debido al carácter '*' que es parte de las expresiones regulares que el Shell acepta, e indica lo que sea. El último ejemplo por lo tanto cambiará los permisos a los archivos dentro del directorio actual.

Una opción común cuando se desea cambiar todo un árbol de directorios, es decir, varios directorios anidados y sus archivos correspondientes, es usar la opción -R, de recursividad:

```
$ chmod -R 755 respaldos/*
```

Esto cambiará los permisos a 755 (rwxr-xr-x) del directorio respaldos y de todos los subdirectorios y archivos que estén contenidos dentro de este.

Estableciendo Permisos en Modo Simbólico Otra manera popular de establecer los permisos de un archivo o directorio es a través de identificadores del bit (r,w, o x) de los permisos, como ya se vio anteriormente, pero ahora identificando además lo siguiente:

- al usuario con la letra u
- al grupo con la letra g

- a otros usuarios con la letra o
- y cuando nos referimos a todos (usuario, grupo, otros) con la letra a (all, todos en inglés)
- el signo + para establecer el permiso
- el signo - para eliminar o quitar el permiso

La sintaxis es muy simple:

```
$ chmod augo[+|-]rwx[,...] archivo[s]
```

así por ejemplo, si queremos que otros tengan permiso de escritura sería chmod o+w archivo, todos los usuarios con permisos de ejecución chmod a+x archivo.

Si quieres prevenirte de modificar un archivo importante, simplemente quita el permiso de escritura en tu «archivo» con el comando chmod

```
$ chmod -w tuArchivo
```

si quieres hacer un Script ejecutable, escribe

```
$ chmod +x tuScript
```

si quieres remover o agregar todos los atributos a la vez

```
$ chmod -rwx archivo
$ chmod +rwx archivo
```

también puedes usar el signo = (igual) para establecer los permisos en una combinación exacta, este comando remueve los permisos de escritura y ejecución dejando solo el de lectura

```
$ chmod =r archivo
```

En este modo de establecer permisos, solo hay que tomar en cuenta que partiendo de los permisos ya establecidos se agregan o se quitan a los ya existentes

Cambiando Propietario y Grupo Volviendo a mostrar el listado al inicio de esta sección:

```
$> ls -l

-rwxrwxr– 1 antonio ventas 9090 sep 9 14:10 presentacion
-rw-rw-r– 1 antonio antonio 2825990 sep 7 16:36 reporte1
drwxr-xr-x 2 antonio antonio 4096 ago 27 11:41 videos
```

Vemos en la tercera y cuarta columna al usuario propietario del archivo y al grupo al que pertenece, es posible cambiar estos valores a través de los comandos chown (Change Owner, cambiar propietario) y chgrp (Change Group, cambiar grupo). La sintaxis es muy sencilla:

```
$ chown usuario archivo[s]
```

y

```
$ chgrp grupo archivo[s].
```

por ejemplo:

```
# ls -l presentacion

-rwxrwxr– 1 antonio ventas 9090 sep 9 14:10 presentacion

# chown juan presentacion
# ls -l presentacion

-rwxrwxr– 1 juan ventas 9090 sep 9 14:10 presentacion

# chgrp gerentes presentacion
# ls -l presentacion

-rwxrwxr– 1 juan gerentes 9090 sep 9 14:10 presentacion
```

Solo el usuario root puede cambiar usuarios y grupos a su voluntad sobre cualquier usuario, queda claro que habiendo ingresado al sistema como usuario normal, solo podrá hacer cambios de grupos, y eso solo a los que pertenezca.

Una manera rápida para el usuario *root* de cambiar usuario y grupo al mismo tiempo, es con el mismo comando chown de la siguiente manera:

```
# chown juan.gerentes presentacion
```

o en vez de punto, con : puntos

```
# chown juan:gerentes presentacion
```

así, cambiará el usuario.grupo en una sola instrucción.

Además, al igual que con chmod, también es posible utilizar la opción -R para recursividad.

Bits SUID, SGID y de Persistencia (Sticky Bit) Aún hay otro tipo de permisos que hay que considerar. Se trata del Bit de permisos *SUID* (Set User ID), el Bit de permisos *SGID* (Set Group ID) y el Bit de permisos de persistencia (Sticky Bit). Para entender los dos primeros el *SUID* y el *SGID* veamos los permisos para un comando de uso común a todos los usuarios, que es el comando passwd, que como se sabe sirve para cambiar la contraseña del usuario, y puede ser invocado por cualquier usuario para cambiar su propia contraseña, si vemos sus permisos observaremos un nuevo tipo de permiso:

```
# ls -l /usr/bin/passwd
```

```
-r-s–x–x 1 root root 21944 feb 12 2020 /usr/bin/passwd
```

SUID en vez de la 'x' en el grupo del usuario encontramos ahora una 's' (suid). passwd es un comando propiedad de root, pero sin embargo debe de poder ser ejecutado por otros usuarios, no solo por root. Es aquí donde interviene el Bit SUID, donde al activarlo obliga al archivo ejecutable binario a ejecutarse como si lo hubiera lanzado el usuario propietario y no realmente quien lo lanzó o ejecutó. Es decir, es poder invocar un comando propiedad de otro usuario (generalmente de root) como si uno fuera el propietario.

SGID el Bit SGID funciona exactamente igual que el anterior solo que aplica al grupo del archivo. Es decir si el usuario pertenece al grupo 'ventas' y existe un binario llamado 'reporte' que su grupo es 'ventas' y tiene el Bit SGID activado, entonces el usuario que pertenezca al grupo 'ventas' podrá ejecutarlo. También se muestra como una 's' en vez del Bit 'x' en los permisos del grupo.

STICKY BIT (Bit de persistencia) este Bit se aplica para directorios como en el caso de /tmp y se indica con una 't':

```
# ls -ld /tmp

drwxrwxrwt 24 root root 4096 feb 25 18:14 /tmp
```

Puede apreciarse la 't' en vez de la 'x' en los permisos de otros. Lo que hace el Bit de persistencia en directorios compartidos por varios usuarios, es que el sólo el propietario del archivo pueda eliminarlo del directorio. Es decir cualquier otro usuario va a poder leer el contenido de un archivo o ejecutarlo si fuera un binario, pero sólo el propietario original podrá eliminarlo o modificarlo. Si no se tuviera el Sticky Bit activado, entonces en estas carpetas públicas, cualquiera podría eliminar o modificar los archivos de cualquier otro usuario.

Estableciendo Permisos Especiales Para cambiar este tipo de Bit se utiliza el mismo comando *chmod* pero agregando un número octal (1 al 7) extra al principio de los permisos, ejemplo:

```
# ls -l /usr/prog

-r-x–x–x 24 root root 4096 sep 25 18:14 prog

# chmod 4511 /usr/prog
# ls -l /usr/prog

-r-s–x–x 24 root root 4096 sep 25 18:14 prog
```

Nótese que el valor extra es el '4' y los demás permisos se dejan como se quieran los permisos para el archivo. Es decir, los permisos originales en este ejemplo eran 511 (r-x–x–x), y al cambiarlos a 4511, se cambió el Bit SUID reemplazando el Bit 'x' del usuario por 's'.

Los posibles valores serían los siguientes:

- - - - - - - - - = 0 Predeterminado, sin permisos especiales. No se requiere indicar.
- - - - - - - - t = 1 Bit de persistencia, Sticky Bit
- - - - - s - - - = 2 Bit Sgid de grupo
- - - - - s - - t = 3 Bit Sgid y Sticky

- - s - - - - - -	= 4	Bit Suid
- - s - - - - - t	= 5	Bit Suid y Sticky
- - s - - s - - -	= 6	Bit Suid y Sgid
- - s - - s - - t	= 7	Bit Suid, Sgid y Sticky

Observación 2 *Algo sumamente delicado y que se tiene que tomar muy en cuenta es lo que decidas establecer con permisos de Bit SUID y SGID, ya que recuerda que al establecerlos de esta manera, cualquier usuario podrá ejecutarlos como si fueran el propietario original de ese programa. Y esto puede tener consecuencias de seguridad severas en tu sistema. Siempre considera y reconsidera si conviene que un usuario normal ejecute aplicaciones propias de root a través del cambio de Bits SUID o SGID. Mejores alternativas pueden ser los comandos sudo y su.*

Permisos preestablecidos con umask El comando umask establece un permiso de archivo y directorio para todos los archivos y directorios que se creen, para conocer su valor podemos teclear:

```
$ umask
```

y nos regresa su valor, por ejemplo: 0022

En formato simbólico usamos la opción -S:

```
$ umask -S
```

y nos regresa su valor, por ejemplo: u=rwx,g=rx,o=rx

lo anterior indica que un directorio se creará con los permisos 755 y los archivos con los permisos 644. Esto se logra restando de 777 el valor de umask (777-022) y (666-022) respectivamente. El primer valor de umask corresponde para valores de Sticky Bit, *GUID* o *SUID*, que por omisión es 0.

Para establecer el valor de la máscara, simplemente se usa el mismo comando *umask* seguido del valor de máscara que se desee:

```
$ umask 0002
```

para dejarlo fijo en la sesión, entonces conviene agregarlo a *.bash_rc* o *.bash_profile* de nuestro directorio de inicio. Algunos ejemplos son:

umask	*Permisos*	*en Archivos*	*Permisos*	*en Directorios*
000	666	(rw-rw-rw-)	777	(rwxrwxrwx)
002	664	(rw-rw-r- -)	775	(rwxrwxr-x)
022	644	(rw-r- -r- -)	755	(rwxr-xr-x)
027	640	(rw-r- - - - -)	750	(rwxr-x- - -)
077	600	(rw- - - - - - -)	700	(rwx- - - - - -)
277	400	(r- - - - - - - -)	500	(r-x- - - - - -)

Archivos y Fechas En sistemas similares a UNIX como GNU/Linux, todo se considera un archivo, y toda la información sobre un archivo (metadatos o atributos del archivo como la hora de creación, la última modificación, etc) excepto en contenido del archivo real se almacena en un inodo y Linux identifica todos y cada uno de los archivos por su número de inodo que no sea el nombre de archivo legible por humanos.

Además, el programa *stat* de GNU/Linux es una utilidad útil para mostrar archivos o el estado del sistema de archivos. Muestra información como el número de inodo, la hora de creación del archivo, la última modificación de datos, el último acceso, el último cambio de estado y mucho más. Podemos combinar *stat* y *debugfs* para obtener toda la información de un archivo, mediante:

```
$ stat archivo
```

no regresa el inodo del archivo en cuestión (supongamos 12), y usando *df* encontramos la partición en la que reside el archivo (supongamos /dev/sda1), combinando esto entonces:

```
# debugfs -R 'stat <12>' /dev/sda2
```

nos mostrará toda la información de dicho archivo.

6.8 Procesos en Primer y Segundo Plano

La ejecución de una orden se efectúa con una o más llamadas al sistema operativo. Por lo regular, el Shell ejecuta una pausa cuando se le pide ejecutar una orden, queda en espera a que ésta termine se de ejecutar. Existe una sintaxis sencilla (un signo & al final de la línea de órdenes) para indicar que el Shell no debe esperar hasta que termine de ejecutarse la orden. Una orden

que deja de ejecutándose de esta manera mientras el Shell sigue interpretando órdenes subsecuentes es una orden en segundo plano (Background), o que se ejecuta en segundo plano. Los procesos para los cuales el Shell sí espera se ejecutan en primer plano (Foreground).

El Shell del sistema GNU/Linux ofrece un recurso llamado control de trabajos (y visualizarlos con los comandos *ps* o *top*) implementado especialmente en el núcleo. El control de trabajos permite transferir procesos entre el primer y segundo plano. Los procesos pueden detenerse y reiniciarse según diversas condiciones, como que un trabajo en segundo plano requiera entradas desde la terminal del usuario. Este esquema hace posible la mayor parte del control de procesos que proporcionan las interfaces de ventanas, pero no requiere Hardware especial. Cada ventana se trata como una terminal, y permite a múltiples procesos estar en primer plano (uno por ventana) en cualquier momento. Desde luego pueden haber procesos de segundo plano en cualquiera de las ventanas.

En GNU/Linux podemos ejecutar procesos en primer plano (Foreground) o bien en segundo plano (Background). Un programa en primer plano lanzado desde un terminal monopoliza dicho terminal, por lo que en principio, no podremos ejecutar ningún otro programa a la vez (veremos más adelante como se puede hacer). Por el contrario un programa en segundo plano una vez iniciado, deja de monopolizar el terminal desde el que se lanzó, y este nos vuelve a mostrar el Prompt[47].

El comando *bg* que es la abreviatura de la palabra background por tanto recibe como parámetro un número de proceso en estado *Stopped* y hace que ésta reanude su ejecución pero en segundo plano es decir, sin bloquear el teclado para el Shell.

De igual forma que hemos pasado un proceso a ejecutar a un segundo plano, el Shell nos permite recuperar la ejecución en primer plano de un proceso que estaba en segundo plano. Para ello se utiliza el comando *fg* (abreviatura de la palabra Foreground). Como parámetro recibe al igual que *bg* el número de un proceso prefijado con %.

¿Como podemos lanzar otro programa desde un terminal con otro pro-

[47] En el Bourne Shell y sus derivados como BASH el *Prompt* que nos permite escribir los diferentes comandos, generalmente termina con el carácter:

- $ para usuario sin privilegios
- # para el administrador, conocido como *root*

grama en ejecución en Foreground?

Pulsamos *CTRL-z* con lo que pausamos el programa en ejecución y Foreground, ojo lo pausamos con lo cual dejará de funcionar, y ya podremos lanzar otro programa, por ejemplo *ls* o podemos hacer una prueba lanzamos *gimp* y comprobamos que podemos operar con él, luego pulsamos *CTRL-z* y vemos cómo dejamos de poder trabajar con *gimp*).

Ahora queremos volver a poner en funcionamiento a *gimp* y así poder volver a utilizar gimp o si queremos devolverlo a Foreground escribiremos *fg*, o si queremos devolverlo a background escribiremos *bg* (esta sería la opción más lógica).

En el caso de que tengamos más de un programa detenido deberemos indicarle tanto a *fg* como a *bg* el identificador de tarea sobre el que actuarán, este ID podemos obtenerlo con el comando *jobs*.

Tenemos a la mano muchas herramientas en el Shell que nos permiten trabajar con los procesos, saber sus PID, sus estamos, sus nombre, sus usuarios, la cantidad de recursos que consumen. Podemos mencionar algunas:

- ps
- pstree
- pmap
- top
- kill
- jobs
- disown

Para interrumpir un proceso que se está ejecutando en segundo plano podemos hacerlo de dos formas. Mediante el comando *fg* la pasamos a ejecutar al primer plano y luego con *Ctrl-c*. Pero también podemos conseguir el mismo efecto sin utilizar el comando *fg* sino simplemente con el comando *kill* seguido del número de proceso prefijado con el símbolo % que nos da el comando *jobs* y podemos usar *disown* para que el comando continúe activo después de cerrar la terminal.

Nohup y & Cuando se cierra una sesión en GNU/Linux, el sistema manda una señal para matar todos los procesos que esté lanzando nuestro usuario (Señal *SIGHUP*). Por lo tanto, aunque les pongamos el & al final, morirán. Para evitar que esto suceda, hay que ejecutar el Script o comando poniendo el comando *nohup* delante y lo que va a hacer es ignorar la señal de *SIGHUP* y redirigir toda la información correspondiente a un fichero llamado nohup.out

```
$ nohup ./Script.sh &
```

esta opción es muy útil cuando no hemos de interactuar con el proceso, puesto que al quedar el fichero de log, se puede ver el resultado con toda la información al respecto.

Los Scripts son indispensable en Linux y en algunas ocasiones puede ser necesario ejecutarlo en segundo plano, ya sea porque tarde mucho en finalizar o porque el programa tiene que ejecutarse de forma indefinida y al mismo tiempo se quieren analizar sus entradas/salidas en tiempo real, o cuando, en el caso de conexiones remotas, por el motivo que sea, se pueda producir una desconexión.

Una buena práctica sería redireccionar stdin, stdout y stderr. Básicamente, por dos razones: rastrear la salida de nuestro Script en caso de producirse algún error, y evitar problemas al terminar nuestra sesión *ssh*, si es que la ejecutamos en un servidor remoto, por ejemplo:

```
$ nohup ./Script.sh > foo.out 2> foo.err < /dev/null &
```

esta opción es muy útil cuando necesitamos ejecutar un proceso largo y no nos interesa saber nada de él hasta que finalice. Podemos desconectarnos irnos, regresar al día siguiente y analizar el resultado del proceso, como alternativa podemos también usar:

```
$ nohup ./Script.sh > salida.txt 2>&1 &
```

Limitar el Tiempo de Ejecución Algunas veces es necesario limitar el tiempo máximo de ejecución de un comando o Script, para ello disponemos de *timeout*, su formato es el siguiente:

```
$ timeout [opciones] duración comando [argumentos]
```

la duración la podemos especificar en segundos (s), minutos (m), horas (h) o días (d), ejemplos:

```
$ timeout 8s ping 127.0.0.1
$ timeout 2h comando argumentos
```

Podemos darle un período de gracia antes de forzar su terminación, ejemplo:

```
$ timeout -k=5 2m comando argumentos
$ timeout -k=5 -s SIGKILL 2m comando argumentos
```

en este caso se envía una señal *KILL* si el comando todavía se está ejecutando tanto tiempo después de que se envió la señal inicial.

También podemos pedir que corra en segundo plano el comando, mediante:

```
$ timeout –foreground 30s ssh -t usuario@servidor top
```

Limitar uso de CPU a un programa Algunas veces es necesario limitar el uso de la CPU (expresado en porcentaje de uso) a un proceso o programa para ello usamos a *cpulimit*, para instalarlo usamos:

```
# apt install cpulimit
```

Si queremos limitar el uso del CPU de un programa en ejecución usamos:

```
# cpulimit -l 50 -e apache2
```

Si conocemos el PID de nuestro proceso, podemos usarlo para limitar el uso del CPU (digamos al 30%), usando:

```
$ cpulimit -l 30 -p 4546 &
```

o bien, podemos lanzar el proceso limitando el uso del CPU, mediante:

```
$ cpulimit -l 30 ./miprograma &
```

6.9 GNU Parallel

¿Alguna vez tuviste la extraña sensación de que tu computadora no es tan rápida como debería ser? Solía sentirme así, y luego encontré GNU Parallel. GNU Parallel es una herramienta de Shell que permite la ejecución de trabajos en paralelo.

Un trabajo puede ser un único comando o entrada de un archivo que contiene elementos tales como una lista de comandos, una lista de archivos, una lista de Hosts, una lista de usuarios, una lista de URL o una lista de tablas. GNU Parallel también puede tomar información de un comando con pipes (|).

Si te encuentras en una situación en la que necesitas ejecutar varios comandos al mismo tiempo (por ejemplo, en los servidores Linux de un centro de datos), ¿qué haces? GNU Parallel es una alternativa interesante que debes conocer.

Cuando se ejecutan comandos en Linux, ya sea uno a la vez en la línea de comandos o desde un Script de Bash, los comandos se ejecutan en secuencia. El primer comando se ejecuta, seguido por el segundo, seguido por el tercero. Es cierto, el tiempo entre los comandos es tan minúsculo, que el ojo humano no se daría cuenta. Pero para algunos casos, puede que no sea el medio más eficiente para ejecutar comandos.

GNU Parallel se puede instalar en casi cualquier distribución de Linux. Dado que GNU Parallel se encuentra en el repositorio estándar, se instala mediante:

```
# apt install parallel
```

Por ejemplo, para cambiar el formato de los .jpg a .png podríamos hacer lo siguiente:

```
$ find .  -name "*.jpg" | parallel -I% –max-args 1 convert %
%.png
```

o

```
$ ls -l *.jpg | parallel convert '{}' '{.}.png'
```

Con eso conseguimos que los comandos *find* o *ls* busquen todos los ficheros .jpg en el directorio actual con cualquier nombre y le pase todos los resultados

a *parallel* mediante la pipe, que luego transmitirá de uno a uno al comando convert para convertirlos a png. Es decir, va a realizar convert nombre1.jpg nombre1.png, convert nombre2.jpg nombre2.png, y así sucesivamente...

O directamente usamos *parallel* para ello:

```
$ parallel convert '{}' '{.}.png' ::: *.jpg
```

ahora, utilizaremos zenity para presentar una barra de progreso de manera gráfica. ¡Porque, que sea terminal no significa que no sea visual!

```
$ find -maxdepth 1 -name '*.jpg' | parallel –bar convert {}
convertidas/{/.}.png 2> >(zenity –progress –auto-kill)
```

7 Trabajando en Línea de Comandos

La mayoría de los usuarios de ordenadores de hoy sólo están familiarizados con la interfaz gráfica de usuario o GUI (del inglés Graphical User Interface) y los vendedores y los expertos les han enseñado que la interfaz de línea de comandos o CLI (del inglés Command Line Interface) es una cosa espantosa del pasado. Es una pena, porque una buena interfaz de línea de comandos es una maravillosa y expresiva forma de comunicarse con el ordenador, muy parecida a lo que el lenguaje escrito es para los seres humanos. Se ha dicho que "las interfaces gráficas de usuario hacen fáciles las tareas fáciles, mientras que las interfaces de línea de comandos hacen posibles las tareas difíciles" y eso es muy cierto aún hoy.

Dado que Linux fue desarrollado desde la familia de sistemas operativos Unix, comparte la misma rica herencia de herramientas de línea de comandos que Unix. Unix saltó a la fama en los primeros años ochenta -aunque fue desarrollado una década antes-, antes de que se extendiera la adopción de las interfaces gráficas de usuario, y por eso, se desarrolló una amplia interfaz de línea de comandos en su lugar. De hecho, una de las razones más potentes para que los primeros que utilizaron Linux lo eligieran sobre, digamos, Windows NT, era la poderosa interfaz de línea de comandos que hacía las "tareas difíciles posibles".

Emuladores de Terminal cuando utilizamos una interfaz gráfica de usuario, necesitamos otro programa llamado emulador de terminal para interactuar con el Shell. Si buscamos en nuestros menús de escritorio, probablemente encontraremos uno. KDE usa Konsole y GNOME usa Gnome-terminal, aunque es probable que se llame simplemente "Terminal" en nuestro menú. Hay muchos otros emuladores de terminal disponibles para Linux, pero todos hacen básicamente lo mismo; nos dan acceso al Shell.

Trabajar con la línea de comandos no es una tarea tan desalentadora como muchos pudieran pensar. No se requieren conocimientos especiales para usar la línea de comandos, pues es un programa como otro cualquiera. La mayoría de las acciones realizadas en Linux pueden llevarse a cabo usando la línea de comandos. Aunque existen herramientas gráficas para la mayoría de programas, a veces esto no es suficiente. Entonces es cuando la línea de comandos cobra su utilidad.

La terminal es llamada a menudo la línea de comandos, el "Prompt", o el "Shell". Antes, éste era el único método por el cual el usuario interactuaba con lel ordenador; sin embargo, muchos usuarios de Linux encuentran más rápido el uso de la terminal que un método gráfico e incluso hoy día tiene algunas ventajas. Aquí aprenderemos cómo usar la terminal.

El Intérprete de Órdenes de Consola o Shell es un programa informático, cuya función consiste en interpretar órdenes, y un lenguaje de consola. El más conocido es Bash[48]. Es una Shell de Unix compatible con POSIX y el intérprete de comandos por defecto en la mayoría de las distribuciones GNU/Linux, además de Mac OS. También se ha llevado a otros sistemas como Windows y Android.

Algunas alternativas a Bash son:

- SH (Bourne Shell) fue desarrollado por Stephen Bourne y es un Shell que se encuentra dentro de la jerarquía de archivos de Unix en /bin/sh.
- TCSH/CSH (C Shell) ha sido desarrollado para proporcionar una interfaz de usuario. Gracias a este Shell podremos ejecutar comandos y ejecutar múltiples programas desde la consola del sistema.
- KSH (Korn Shell) su desarrollo principal fue la interpretación de órdenes a través de línea de comandos.
- Fish (Friendly Interactive Shell) no está basado en sh y posee una sintaxis de línea de comandos única que está diseñada para ser más amigable con los usuarios que están iniciando en el mundo Shell.
- ZSH (Z Shell) ha sido un Shell diseñado en 1990 influenciado por Bash, Ksh y Tcsh. Zsh es un Shell popular gracias a sus características de desempeño y funcionalidades a la hora de ejecutar comandos.
- TSCH (TS Shell) es una versión mejorada de CSH , la cual ofrece múltiples usos ya que es un lenguaje de comandos que puede ser usado tanto como un Shell de inicio de sesión interactivo como un procesador de comandos Shell.

[48] Su nombre es un acrónimo de Bourne-again Shell ("Shell Bourne otra vez"), haciendo un juego de palabras (Born-again significa "nacido de nuevo") sobre la Bourne Shell (sh), que fue uno de los primeros intérpretes importantes de Unix.

- DaSH (Debian Almquist Shell) es un intérprete de comandos de Unix compatible con el estándar sh de POSIX, mucho más ligero y rápido que otros como Bash pero con menos características.
- Busybox, integra más de doscientas utilidades en un solo ejecutable, usa Almquist Shell y es ideal para dispositivos Linux integrados.
- DSH (Distributed Shell) es un pequeño programa que nos permitirá ejecutar un comando en varias máquinas de una sola vez.

Podemos conocer que interprete de comandos usamos, para ello escribimos:

```
$ ps -p $$
```

Si deseamos cambiar de intérprete a por ejemplo *zsh*, podemos instalarlo usando:

```
# apt install zsh
```

y debemos indicarle al sistema que queremos utilizar *zsh* como Shell por defecto usando:

```
chsh -s $(which zsh)
```

La sintaxis de órdenes de Bash es un superconjunto de instrucciones basadas en la sintaxis del intérprete Bourne. La especificación definitiva de la sintaxis de órdenes de Bash, puede encontrarse en el Bash Reference Manual distribuido por el proyecto GNU. Esta sección destaca algunas de sus características únicas.

7.1 Herramientas de Programación y Edición de Archivos Fuente

En Linux existe una gran variedad de herramientas para programación y edición de archivos fuente, ya que este sistema operativo fue hecho por programadores y para programadores, por ello entre las miles de herramientas, tenemos algunas que son ampliamente usadas, entre las que destacan:

Editores de Terminal

- Diakonos
- Jet
- Joe
- LE
- Mined
- Nano
- Nice Editor (NE)
- Pico
- Setedit
- Vim
- Fte

Editores Sencillos con Interfaz Gráfica

- Gedit
- SciTE
- JEdit
- NEdit
- MEdit
- KScope
- Editra
- Kate
- KWrite

- Leafpad
- Mousepad
- Anjunta
- TEA
- Pluma
- GVim
- Emacs

Editores Avanzados con Interfaz Gráfica

- Atom
- Bluefish
- BlueGriffon
- Brackets
- Geany
- Glade
- Google Web Designer
- KompoZer
- Light Table
- Notepadqq
- Scribes
- Sublime Text

Entornos de Programación Integrado (IDEs)

- Aptana
- Arduino IDE
- Android Studio
- CodeLite
- Code::Blocks
- Eclipse
- Gambas
- JetBrains Suite
- NetBeans
- Ninja-IDE
- Python IDLE
- PyDev
- Postman
- Qt Creator
- Simply Fortran
- Visual Studio Code
- Wing Python IDE
- Spyder
- PyCharm
- Jupyter
- Eric

Kit de Desarrollo de Software

- .Net Core SDK
- Android SDK
- Java JDK

Comparadores de texto y fuentes

- KDiff3
- Meld
- Diffuse
- DirDiff
- kompare
- Numdiff
- colordiff
- wdiff
- xxdiff
- tkdiff
- Ndiff

Otras Herramientas

- Alleyoop
- C2HTML
- Java2HTML
- Code2HTML
- c2html

- AutoDia
- txt2html
- html2text

Programas para control de versiones que permite desarrollo colaborativo de Software:

- Git https://git-scm.com/
- Mercurial https://www.mercurial-scm.org/
- Subversion https://subversion.apache.org/
- Perforce
- Bazaar
- CVS
- LibreSource
- Monotone
- SmartGit
- GitKraken
- Git Cola

Generadores automaticos de documentación que generan salida en PDF, HTML y XML para lenguajes como C++ y Java:

- Doxygen http://www.doxygen.org/
- JavaDoc

Formateador de código fuente para C, C++, Java y C#

- Astyle http://astyle.sourceforge.net

Lenguaje Unificado de Modelado (Unified Modeling Language)[49] forja un lenguaje de modelado visual común semántica y sintácticamente rico para la arquitectura, el diseño y la implementación de sistemas de Software complejos tanto en estructura como en comportamiento:

- UML https://www.uml.org/

En este apartado, solo tocaremos las más usadas, pero abunda la documentación de estas y otras importantes herramientas en línea de comandos.

7.1.1 ¿Qué es eso de ASCII, ISO-8859-1 y UTF-8?

Los tres estándares representan el esfuerzo informático por brindar un sistema de codificación que permita representar los caracteres que se usan en todos los idiomas. El primer esfuerzo lo hizo *ASCII* y fue para el idioma inglés (128 caracteres), luego ante su insuficiencia para representar otros caracteres como los latinos por ejemplo, nace *ISO-8859-1* (también llamado *LATIN-1* ó *ASCII* extendido) pero debido a que no podía representar caracteres de otros idiomas aparece el estándar Unicode (del cual es parte *UTF-8*).

Un buen detalle a saber es que mientras *ISO-8859-1* usa un byte para representar un carácter, no pasa lo mismo con *UTF-8* que puede usar hasta 4 bytes ya que es de longitud variable. Esto hace que una base de datos en *UTF-8* sea un poco más grande que una en ISO-8859-1. Esto sucede porque -por ejemplo- mientras *ISO-8859-1* usa un byte para representar la letra ñ, *UTF-8* usa dos bytes. Hay un tema más y es que muchas veces cuando vamos a migrar información nos encontramos con caracteres *ISO-8859-1* (los correspondientes a los números 147, 148, 149, 150, 151 y 133) que no pueden verse en un editor *UNIX/LINUX* pero si en un navegador *HTML*.

Unicode es un set de caracteres universal, es decir, un estándar en el que se definen todos los caracteres necesarios para la escritura de la mayoría de los idiomas hablados en la actualidad que se usan en la computadora. Su objetivo es ser, y, en gran medida, ya lo ha logrado, un superconjunto de

[49] Otras opciones son: UML Diagram Generation, Code Generation, Document Generation and Reporting, Scaling, Database Schema Generation, Entity Relationship Diagrams, Data Flow Datagrams, StarUML BOUML, EclipseUML, UML Modeller, Papyrus, Nclass, PlantUML, UMLet, NetBeansIDE, Open ModelSphere, gModeler, RISE, Oracle jdeveloper, Oracle SQL Developer, Dia, Kivio, ArgoUML, Xfig, etc.

todos los sets de caracteres que se hayan codificado. El texto que aparece en la computadora o en la Web se compone de caracteres. Los caracteres representan letras del abecedario, signos de puntuación y otros símbolos.

En el pasado, distintas organizaciones han recopilado diferentes sets de caracteres y han creado codificaciones específicas para ellos. Un set puede abarcar tan sólo los idiomas de Europa occidental con base en el latín (sin incluir países de la Unión Europea como Bulgaria o Grecia), otro set puede contemplar un idioma específico del Lejano Oriente (como el japonés), y otros pueden ser parte de distintos sets diseñados especialmente para representar otro idioma de algún lugar del mundo.

Lamentablemente, no es posible garantizar que su aplicación particular pueda soportar todas las codificaciones, ni que una determinada codificación pueda soportar todos sus requerimientos para la representación de un cierto idioma. Además, generalmente resulta imposible combinar distintas codificaciones en la misma página Web o en una base de datos, por lo que siempre es muy difícil soportar páginas plurilingües si se aplican enfoques "antiguos" cuando se trata de tareas de codificación.

El Consorcio Unicode proporciona un único y extenso set de caracteres que pretende incluir todos los caracteres necesarios para cualquier sistema de escritura del mundo, incluyendo sistemas ancestrales (como el cuneiforme, el gótico y los jeroglíficos egipcios). Hoy resulta fundamental para la arquitectura de la Web y de los sistemas operativos, y las principales aplicaciones y navegadores Web incluyen soporte para este elemento. En el Estándar Unicode también se describen las propiedades y algoritmos necesarios para trabajar con caracteres. Este enfoque facilita mucho el trabajo con sistemas o páginas plurilingües y responde mucho mejor a las necesidades del usuario que la mayoría de los sistemas de codificación tradicionales.

Sets de caracteres, sets de caracteres codificados y codificaciones un set de caracteres o repertorio comprende el grupo de caracteres que se utilizarían para una finalidad específica, ya sea los necesarios para el soporte de los idiomas de Europa Occidental en la computadora.

Un set de caracteres codificados es un grupo de caracteres en el que se ha asignado un número exclusivo a cada carácter. Las unidades de un set de caracteres codificados se conocen como puntos de código. El valor de un punto de código representa la ubicación de un carácter en el set de caracteres codificados. Por ejemplo, el punto de código para la letra *á* en el set de

caracteres codificados Unicode es *225* en notación decimal, o *E1* en notación hexadecimal.

La codificación de caracteres refleja la manera en la que el set de caracteres codificados se convierte a bytes para su procesamiento en la computadora. Además, en Unicode existen distintas formas de codificar el mismo carácter. Por ejemplo, la letra *á* se puede representar mediante dos bytes en una codificación y con cuatro bytes, en otra. Los formatos de codificación que se pueden usar con Unicode se denominan *UTF-8, UTF-16 y UTF-32.*

Por todo lo anterior, al programar es necesario tener en cuenta la codificación usada por el editor o IDE que se use para ello y que no todos los editores soportan las mismas codificaciones[50], además puede haber problemas de portabilidad en los archivos entre distintos sistemas operativos. En el código fuente (las instrucciones del programa) no se suele usar caracteres distintos al *ASCII*, pero en las cadenas de visualización o en la documentación es común el uso de caracteres acentuados, es aquí donde hay que tomar una decisión sobre el usar o no dichos caracteres, siempre y cuando el compilador los soporte.

Si siempre se usa el mismo editor y la misma plataforma de desarrollo, no hay razón para no usar caracteres extendidos como los acentos. Pero si se usarán múltiples sistemas operativos y no hay garantía de usar editores que soporten dichos caracteres, entonces existe la posibilidad de perder dichos caracteres o bien pueden generar errores al compilar los archivos por no ser soportados. Por ello una opción para evitar problemas es sólo usar caracteres *ASCII* o tener el cuidado de usar editores que no pierdan dichos caracteres.

En Linux, para verificar la codificación de un archivo se utiliza el comando *file –i* o *–mime*, este comando permite mostrar en pantalla el tipo de archivo y su codificación, usando:

```
$ file -i Car.java
```

El comando *iconv* es utilizado para realizar esta tarea de convertir el código de un texto a otro. La lógica para aplicar correctamente el commando *iconv* es la siguiente:

```
$ iconv options -f from-encoding -t to-encoding inputfile(s) -o
outputfile
```

[50] Dado que los archivos fuente se intercambian entre usuarios y es común el uso de diferentes sistemas operativos, la conversiones de los caracteres entre diferentes formatos puede ser causa de problemas de codificación, perdiéndose dichos caracteres en la conversión.

así, *-f* o *–from-code* significa la entrada de la codificación, y *–t* o *–to-encoding* especifica la salida de la misma. Para enumerar todos los juegos de caracteres podemos usar *-l* o *–list*:

```
$ iconv -l
```

Con todo esto en mente podemos proceder a explicar la codificación de *UTF-8* a *ASCII*. Primero hay que comenzar con conocer las codificaciones de los caracteres en el archivo y luego poder ver el contenido del mismo. Así, se podrán convertir todos los archivos a la codificación *ASCII*. Todo después de haber utilizado el comando *iconv*, para poder verificar lo que contiene la salida del archivo. Para eso hay que hacer lo siguiente:

```
$ file -i input.file
$ cat input.file
$ iconv -f ISO-8859-1 -t UTF-8//TRANSLIT input.file -o
out.file
$ cat out.file
$ file -i out.file
```

Cabe destacar que, si el comando *//IGNORE* se añade a *to-encoding*, los caracteres no pueden ser convertidos y un error se mostrará luego de la conversión. También, si el comando *//TRANSLIT* es añadido a *to-encoding* como en el ejemplo dado *(ASCII//TRANSLIT)*, los caracteres convertidos son transliterados, si es posible, como necesarios.

Esto implicaría que en este evento los caracteres no pueden ser representados como se desea, aunque pueden haber aproximaciones del mismo, inclusive dos. Por lo que, si un carácter no puede ser transliterado, no será reconocido como un objetivo para reemplazo y se mostrará la marca (?) en la salida del archivo.

Algunas veces es necesario convertir el archivo de UTF-8 a ASCII y lo hacemos mediante:

```
$ iconv -f UTF-8 -t ISO-8859-1 prog.c -o progMod.c
```

o mediante:

```
$ iconv -f UTF-8 -t ASCII//TRASLIT prog.c -o progMod.c
```

7.1.2 Uso de Espacios o Tabuladores en Fuentes

En muchos lenguajes de programación la identación es una forma opcional de hacer legible el código para el programador, en otros (como Python) es imprescindible (ya que de ella dependerá su estructura).

¿Qué es la identación? En un lenguaje informático, la identación es lo que a la sangría al lenguaje humano escrito (a nivel formal). Así como para el lenguaje formal, cuando uno redacta una carta, debe respetar ciertas sangrías, los lenguajes informáticos, pueden requerir una identación.

No todos los lenguajes de programación necesitan una identación, aunque sí se estila implementarla, a fin de otorgar mayor legibilidad al código fuente. Una identación depende del lenguaje y estilo de codificación usado, por ejemplo en C, C++ y Java se acostumbra usar tres espacios en blanco. En el caso de Python se suele usar 4 espacios en blanco o un tabulador, que indicará que las instrucciones identadas forman parte de una misma estructura de control.

Los programadores siempre han debatido entre el uso de espacios y tabulaciones para estructurar su código. Los espacios y las tabulaciones son utilizados por los programadores para estructurar el código de una forma determinada. La primera línea de código (sin espacio o tabulación) inicia un "bloque" de contenido. Si las sucesivas líneas de código forman parte de ese mismo bloque (encerrado entre corchetes) o forman nuevos subbloques, estas se van desplazando hacia la derecha para indicar esa subordinación. En caso de formar un bloque completamente nuevo, se mantiene en la misma posición que la línea inmediatamente anterior.

A nivel funcional, la diferencia entre el uso de espacios o tabulaciones es nula. Cuando el código pasa por el compilador antes de ser ejecutado, la máquina interpreta de igual forma ambos formatos. No obstante, sí existen diferencias técnicas que marcan la diferencia entre el uso de tabulaciones y espacios:

- Precisión. Una tabulación no es más que un conjunto de espacios agrupados. Por norma general, este conjunto suele ser de 8 caracteres, pero puede variar. ¿Qué quiere decir esto? Que cuando un mismo fichero de código se abre en dos máquinas diferentes, la apariencia del código puede ser diferente. En cambio, el uso de espacios no conlleva este problema: un espacio siempre ocupa el mismo "espacio" —valga la

redundancia— y asegura que el código se visualiza de la misma forma en todas las máquinas.

- Comodidad. En el caso de las tabulaciones, basta con pulsar la tecla de tabulación una única vez para estructurar correctamente el código. En el caso de los espacios, es necesario pulsar varias veces la misma tecla para lograr la estructura deseada.

- Almacenamiento. El uso de tabulaciones también reduce el tamaño del fichero final, mientras que el uso de espacios lo aumenta. Lo mismo sucedería con el uso de espacios en lugar de saltos de línea.

Entonces, ¿cuál es la más correcta? La realidad es que todo depende de las preferencias personales. Si necesitas optimizar el tamaño de los ficheros al máximo, el uso de espacios se convierte en un sacrilegio. Si, en cambio, tu código debe lucir exactamente igual en múltiples máquinas, el uso de espacios puede ser más conveniente para lograr esa homogeneidad.

Por suerte, existen múltiples editores en la actualidad que trabajan y facilitan la transición entre ambos sistemas. Asimismo, los equipos de desarrollo de software establecen en sus guidelines el uso de espacios o tabulaciones. De esta forma, se evitan conflictos entre los programadores de un mismo proyecto y se alcanza esa homogeneidad tan deseada.

El comando *expand* y *unexpand* (que vienen instalados en los paquetes *GNU Core*) permite convertir tabuladores en espacios y viceversa, según nuestras necesidades o gustos. Estos comandos sacan el resultado de stdin o de los archivos nombrados en la línea de comando. Utilizando la opción *-t* se pueden establecer una o más posiciones de tabulador.

Para ver si se usan espacios o tabuladores en un archivo fuente podemos usar el comando *cat* con la opción *-T* que nos mostrará los caracteres tabulador como *^I*, ejemplo:

```
$ cat -T archivo
```

Para convertir los espacios en tabuladores (un tabulador igual a 8 espacios) usamos:

```
$ unexpand progEsp.c
```

o redireccionando la salida usando:

```
$ unexpand progEsp.c > progTab.c
```

Para convertir los tabuladores en espacios (1 tabulador por ejemplo 4 caracteres) usamos:

```
$ expand -t 4 progTab.c
```

o redireccionando la salida usando:

```
$ expand -t 4 progTab.c > progEsp.c
```

También es posible buscar todos los archivos (digamos *.cpp) y cambiar los tabuladores por 4 espacios (para ello usamos el comando *sponge* que está contenido en el paquete *moreutils*), mediante:

```
$ find . -name '*.cpp' -type f -exec bash -c \
'expand -t 4 "$0" | sponge "$0"' {} \;
```

7.1.3 Comparar Contenido de Fuentes

Cuando se programa es común generar distintas versiones de un mismo archivo, en GNU/Linux se tiene varias herramientas para comparar y combinar cambios. En la línea de comandos el comando *diff* permite ver los cambios entre dos versiones de un archivo y el comando *merge* sirve para combinar cambios. Por otro lado *sdiff* nos permite ver las diferencias entre dos archivos y de forma interactiva combinar cambios.

Pese a que son poderosos estos comandos, en forma gráfica se puede obtener todo su potencial. Algunas de estas opciones son:

```
# apt install kdiff3 meld diffuse dirdiff kompare wdiff  \
numdiff colordiff xxdiff tkdiff ndiff
```

Estos permiten comparar dos o tres versiones de un archivo simultáneamente, y hacerlo con el contenido de una o más carpetas. Cada uno tiene la capacidad de mostrar los cambios y si se requiere hacer la combinación de ellos.

meld nos muestra gráficamente las diferencias entre dos archivos o también, entre todos los archivos de dos directorios utilizando distintos colores, y nos permite editar estos archivos desde el propio programa, actualizando dinámicamente las diferencias. El programa incluye filtros y distintas ayudas para hacer la edición más sencilla, como flechas al lado de los cambios para aplicar cambio en cualquiera de los archivos con un simple clic. Este programa se puede utilizar como un sencillo cliente de control de cambios para Git, CVS, Subversion, etc.

kdiff3 nos muestra gráficamente las diferencias entre tres archivos utilizando distintos colores, y nos permite editar estos archivos desde el propio programa, actualizando dinámicamente las diferencias. El programa incluye filtros y distintas ayudas para hacer la edición más sencilla, como flechas al lado de los cambios para aplicar cambio en cualquiera de los archivos con un simple clic.

7.2 Buscar Archivos

La búsqueda de archivos de GNU/Linux se puede hacer con por lo menos dos comandos diferentes para buscar archivos: find (que traduce encontrar) y locate (que traduce ubicar).

Usando el comando find El comando más común utilizado para encontrar y filtrar archivos es a través del comando *find*. El diseño básico de este comando es el siguiente:

find <directorio inicio> <opciones> <termino búsqueda>

El argumento <*directorio inicio*> es el punto de origen de donde deseas iniciar la búsqueda. Esto es útil si tienes una idea aproximada de dónde podría estar ubicado el archivo deseado, ya que hace más específica la búsqueda. La mayoría de las veces, sin embargo, querrás buscar el archivo en todo el sistema. Puedes hacer esto reemplazando tu ruta con una barra " / ", que es el símbolo del directorio raíz. A veces es posible que quieras iniciar la búsqueda desde el directorio de trabajo actual, es decir, el directorio donde está abierto el terminal. Esto se puede hacer con el argumento punto " . ". Para averiguar tu directorio actual, usa el comando pwd. Finalmente, para

comenzar la búsqueda de archivos desde tu carpeta de inicio, usa el símbolo *" ~"*.

El segundo argumento es el filtro que deseas usar para buscar tu archivo. Este podría ser el nombre, tipo, fecha de creación o de modificación del archivo, etc. El tercer argumento es una continuación del segundo, donde se especificará el término de búsqueda relevante.

Búsqueda por Nombre por supuesto, el método más común y obvio para buscar un archivo es usar su nombre. Para ejecutar una consulta de búsqueda simple usando el nombre del archivo, usa el comando find de la siguiente manera:

```
$ find . -name "archivo"
```

En el comando anterior, usamos la opción *-name* y buscamos un archivo llamado *archivo*. Ten presente que comenzamos la búsqueda en nuestro directorio actual y nos dará todas las coincidencias que encuentre. Pero si sólo deseamos la primera coincidencia podemos usar:

```
$ find . -name "archivo" -print -quit
```

Una cosa importante para recordar cuando se utiliza el estándar es el argumento *-name*, que busca términos distinguiendo entre mayúsculas y minúsculas en GNU/Linux. Por lo tanto, si conoces el nombre del archivo, pero no estás seguro de su las mayúsculas y minúsculas, usa el comando find de esta manera:

```
$ find . -iname "archivo"
```

o bien para buscar todos los archivos con extensión .zip, .tar, .tar.gz, entre otros, usamos:

```
$ find . -type f \( -name "*.zip" -o -name "*.tar*" \)
```

Otra forma de utilizar la opción name es buscar todos los archivos sin una palabra clave en el nombre. En GNU/Linux, puedes hacer esto de dos maneras. El primer método implica el uso de la palabra clave *-not* de la siguiente manera:

```
$ find . -not -name "archivo"
```

También podemos usar *" ! "*[51], aunque debe estar precedido por el identificador de escape para que GNU/Linux sepa que esta es parte del comando de búsqueda y no una independiente.

```
$ find . \! -name "archivo"
```

También puede buscar varios archivos con un formato común como *.txt*, lo cual podría ser útil en algunos casos:

```
$ find . -name "*.txt"
```

esto listará todos los archivos de texto comenzando con la carpeta actual.

Algunas veces es necesario acotar la profundidad de la búsqueda y limitarla, digamos al actual directorio, por ejemplo:

```
$ find . -maxdepth 1 archivo
```

Si deseas buscar un determinado archivo por nombre y eliminarlo, usamos el argumento *-delete* después del nombre del archivo:

```
find . -name "archivo" -delete
```

otra forma es usar:

```
$ find . -name "archivo" | xargs rm -f
```

también podemos tomar la salida del comando y hacer algo con ella, como:

```
$ find . -name 'log*' | xargs wc
```

Si al realizar una búsqueda se generan errores (por no tener acceso a archivos o directorios por ejemplo), estos pueden ser redireccionados, mediante:

```
$ find / -type f 2> /dev/null
```

[51] Otras opciones son -not (!), -or (-o), y -and (-a) aunque este último está implícito cuando se enumeran dos requisitos.

Búsqueda por Tipo para la mayoría de los usuarios, basta con saber cómo encontrar archivos por sus nombres. Sin embargo, siempre es útil conocer todas las herramientas que se ofrecen para aprovechar GNU/Linux al máximo.

Aquí es donde entra en juego el argumento -type. GNU/Linux ofrece a los usuarios las siguientes opciones para buscar archivos por tipo:

- f - Archivo normal
- d - Directorio o carpeta
- l - Enlace simbólico
- c - Dispositivos de caracteres
- b - Dispositivos de bloque

Un ejemplo simple del uso del tipo de archivo para la búsqueda se puede ver a continuación:

```
$ find / -type d
```

Esto mostrará una lista de todos los directorios presentes en el sistema de archivos, al haber comenzado la búsqueda desde nuestro directorio raíz con el símbolo de barra inclinada. Para encontrar las ligas simbólicas que hay desde nuestra actual trayectoria, usamos:

```
$ find . -type l
```

También puedes concatenar las opciones -type y -name para hacer tus búsquedas aún más específicas:

```
$ find / -type f -name "archivo"
```

esto buscará archivos llamados *archivo*, excluyendo directorios o enlaces.

Otro ejemplo es buscar archivos, directorios y enlaces simbólicos, mediante:

```
$ find /tmp -type f,d,l
```

o

```
$ find /tmp \( -type f -o -type d -o -type l \)
```

Para busca todas las ligas simbólicas y saber a donde apuntan, usamos:

```
$ find . -type l -print | xargs ls -ld | awk '{print $9 $10 $11}'
```

y si queremos conocer las ligas rotas, usamos:

```
$ find . -xtype l
```

Algunas veces es necesario acotar la profundidad de la búsqueda y limitarla, digamos al actual directorio, por ejemplo:

```
$ find . -maxdepth 1 -type f -newer archivo
```

Búsqueda por Fecha si quieres buscar archivos en función de su fecha de acceso y las registros de fecha de modificación, GNU/Linux te ofrece las herramientas para hacerlo. Hay 3 registros de tiempo de los cuales Linux realiza seguimiento en los archivos:

- Tiempo de acceso (-atime) - Fecha más reciente en que el archivo fue leído o escrito.
- Tiempo de modificación (-mtime) - Fecha más reciente en que se modificó el archivo.
- Hora de cambio (-ctime) - Fecha más reciente en que se actualizaron los metadatos del archivo.

Esta opción debe usarse con un número. Este número especifica el número de días desde que se accedió, modificó o cambió el archivo. La forma más sencilla de buscar archivos por tiempo es:

```
$ find / -atime 1
```

Esto encontrará todos los archivos a los que se accedió hace un día desde el momento actual. Esto significa que se listarán todos los archivos que fueron leídos y/o escritos desde el día anterior.

Podemos hacer que nuestras consultas sean más precisas con los signos más (+) y menos (-) precediendo al número de días. Por ejemplo:

```
$ find / -mtime +2
```

Esto listará todos los archivos que tienen un tiempo de modificación de más de dos días.

Para buscar todos los archivos cuyos metadatos de *inodo* se actualizaron hace menos de un día, ejecuta lo siguiente:

```
$find / -ctime -1
```

Finalmente, hay algunos argumentos adicionales además de estos 3 que también están relacionados con las búsquedas por fecha. El argumento -*<time-deScriptor>min* busca cuántos minutos han pasado. Se puede usar así:

```
$find / -mmin -1
```

Esto buscará archivos que se modificaron hace menos de un minuto. Además, el argumento *-newer* se puede usar para comparar la antigüedad de dos archivos y encontrar el más reciente.

Para buscar archivos **.log* de más de 90 días, podemos usar:

```
$ find / -name "*.log" -type f -mtime +90 -ls
```

y los podemos borrar usando:

```
$ find /app/logs/ -name "*.log" -type f -mtime +90 -delete
```

Podemos buscar archivos no vacíos de más de 4 días en el directorio actual sin extensión *.gz* y comprimirlos:

```
$ find . -mtime +4 ! -name '*.gz' ! -empty -execdir gzip -v9
{} +
```

Búsqueda por tamaño al igual que GNU/Linux te brinda la opción de buscar archivos basados en registros de tiempo, también te permite hacer lo mismo con los tamaños. La sintaxis básica para buscar archivos por tamaño es:

$ find <directorio inicio > -size <tamaño> <unidad tamaño>

Puedes especificar las siguientes unidades de tamaño:

- c - Bytes
- k - kiloBytes
- M - MegaBytes
- G - GigaBytes
- b - Trozos de 512 Bytes

Un ejemplo simple de cómo usar el comando find para los tamaños de archivo es el siguiente:

$ find / -size 10M

Esto buscará en tu sistema archivos que tengan exactamente 10 MegaBytes de tamaño. Al igual que cuando buscaste en función del tiempo, puedes filtrar aún más tus búsquedas con los signos más y menos:

$ find / -size +5G

El comando anterior listará todos los archivos de tu disco que tengan más de 5 GigaBytes de tamaño. De manera similar, puede lograr esto con el signo menos para indicar "menor que" en tus consultas.

Además podemos pedir que la búsqueda nos entregue los datos de los archivos encontrados usando el comando *ls -l*, por ejemplo:

$ find . -size +1000c -exec ls -l {} \;

o si queremos buscar y borrar, usamos:

$ find / -type f -mtime +30 -sise +1M -exec rm -rf {} \;

o bien:

```
$ find / type f -mtime +30 -size +1M -delete
```

Por último, podemos buscar ficheros vacíos, mediante:

```
$ find . -type f -empty
$ find . -size 0c
```

o directorios vacíos, usamos:

```
$ find . -type d -empty
```

Algunos ejemplos útiles son:

```
$ find . -printf '%s %p\n' | sort -nr | head -10
$ find . -type d -printf '%s %p\n' | sort -nr | head -10
$ find . -type f -printf '%s %p\n' | sort -nr | head -10
$ find .  -iname "*.mp4" -printf '%s %p\n' | sort -nr | head
-10
$ find . -type d -empty -print0 | xargs -0 rmdir
$ find . -type f -empty -print0 | xargs -0 rm
```

Búsqueda por Propiedad La jerarquía de privilegios de Linux también se puede utilizar al buscar archivos. GNU/Linux te da la capacidad de especificar tus búsquedas según la propiedad del archivo, así como los permisos otorgados a diferentes usuarios.

Para buscar archivos de un determinado propietario, se debe ejecutar el siguiente comando:

```
$ find / -user usr
```

Esto devolverá una lista de todos los archivos que posee el usuario llamado *usr*. Similar a los nombres de usuario, también podemos buscar archivos a través de nombres de grupo:

```
$ find / -group otro
```

Esto buscará aquellos archivos que son propiedad del grupo llamado *otro*.

Búsqueda por permisos Los usuarios que desean buscar archivos basados en los permisos de los archivos[52] pueden hacerlo usando la opción *-perm* del comando *find*. Por ejemplo:

```
$ find / -perm 644
```

En GNU/Linux, 644 corresponde a permisos de lectura y escritura. Lo que significa que este comando buscará todos los archivos que solo tienen permisos de lectura y escritura. Otras opciones son:

```
$ find . -perm 1551
$ find . -perm /u=r
$ find . -perm /a=x
```

o si deseamos buscar los que no tienen esos permisos, para ello usamos *-not*, por ejemplo:

```
$ find . -not -perm 777
```

También podemos buscar ficheros/directorios que tengan activos los bits *SUID*, *SGID* y *Sticky Bit* que también podrían representar problemas de seguridad. Pero para que busque cualquiera sin importar el resto de los permisos, debemos usar / para obviar los permisos estándar, para ello usamos respectivamente:

```
$ find . -perm /4000
$ find . -perm /2000
$ find . -perm /1000
```

[52]

Octal	Binario	Modo Archivo
0	000	- - -
1	001	- - x
2	010	- w -
3	011	- w x
4	100	r - -
5	101	r w -
6	110	r w -
7	111	r w x

El siguiente ejemplo busca archivos que sean de lectura para todos (-perm -444 o -perm -a+r) y que tenga al menos un conjunto de bits de escritura (-perm /222 o -perm /a+w) pero no son ejecutables para cualquiera (\! -perm /111 o \! -perm /a+x):

```
$ find . -perm -444 -perm /222 \! -perm /111
```

o

```
$ find . -perm -a+r -perm /a+w \! -perm /a+x
```

Por último, podemos cambiar todos los permisos digamos 744 encontrados a 644, usando:

```
$ find . -perm -744 -exec chmod -R 644 {} \;
```

Búsqueda por inodo el comando *ln* permite crear enlaces a los archivos, tanto duros (Hard Links) como simbólicos *-s* (Soft Links). Todos los archivos que apuntan a un mismo enlace duro, comparten el *inodo* -este es un registro en el disco que contiene la información del archivo como su propietario, tamaño, fecha de creación y ubicación-.

Para conocer el *inodo* de un archivo, usamos:

```
$ ls -li archivo
```

y para conocer todos los archivos que apuntan a un mismo enlace duro cuyo *inodo* conozcamos, usamos:.

```
$ find / -inum <inodo>
```

y para conocer a todos los archivos que comparten inodo, usamos:

```
$ find . -type f -printf '%10i %p\n' | sort | uniq -w 11 -d -D |
less
```

Búsqueda usando expresiones regulares esto nos ofrece una potencia considerable. Por ejemplo si deseamos buscar un nombre como:

archivo01_01.txt, archivo02_03.txt, etc.

podemos utilizar:

```
$ find . -regex './archivo0[1-2]_0[1-3].*'
```

Otras opciones útiles Además de todos estos métodos de búsqueda de archivos, hay otras opciones útiles que uno debería recordar. Por ejemplo, para buscar archivos y carpetas vacíos en el sistema, usamos:

```
$ find / -empty
```

del mismo modo, para buscar todos los ejecutables guardados en tu disco, utiliza la opción *-exec:*

```
$ find / -exec
```

para buscar archivos leíbles, usamos:

```
$ find / -read
```

Hay veces que necesitamos quitar espacios en el nombre de los archivo, por ejemplo:

```
$ find . -name "* *mp3" -exec rename 's/\ /-/g' {} \;
```

remplazará en todos los archivos **.mp3* los espacios en el nombre.
Si necesitamos buscar en todos nuestros archivos *.java* aquellos que contengan por ejemplo la palabra *interface*, podemos usar:

```
$ find . -name "*.java" - type f -exec grep -l interface {} \;
```

Si necesitamos buscar archivos y filtrarlos por nombre de archivo, además de canalizar el resultado a algún otro comando (por ejemplo *wc* para contar palabras y usamos *tr* para sustituir el carácter nulo por nuevas líneas), el comando queda como:

```
$ find . -name ".md" | grep "notas/2020" | \
tr '\n' '\0' | xargs -0 wc -w
```

El siguiente ejemplo copia el contenido de la actual trayectoria a /dest-dir, pero omite archivos y directorios llamados .snapshot (y cualquier cosa dentro de ellos). También omite archivos o directorios cuyo nombre termina en ~, pero no sus contenidos.

```
    $ find . -name .snapshot -prune -o \( \! -name '*~' -print0 \)
\
    | cpio -pmd0 /dest-dir
```

la construcción -prune -o \(\! -name '*~' -print0 \)| es bastante común. La idea aquí es que la expresión antes de -prune coincide con las cosas que se deben podar, sin embargo, el -prune acción en si misma se devuelve verdadero, por lo que el siguiente -o asegura que el lado derecho es evaluado solo para aquellos directorios que no fueron podados. La expresión en el lado derecho de -o está entre paréntesis solo por claridad. Hace hincapié en que la acción -print0 tiene lugar solo para las cosas que no tenían -prune aplicado a ellos. Otro ejemplo es:

```
$ find repo/ \( -exec test -d '{}'/.svn \; -or \
-exec test -d {}/.git \; -or -exec test -d {}/CVS \; \) \
-print -prune
```

en este ejemplo -prune evita el descenso innecesario a los directorios que ya han sido descubiertos.

Los siguientes ejemplos permiten copiar los archivos (por ejemplo *.mp3) pero preservan la estructura de directorios que existía en su lugar de origen, veamos:

```
$ find . -name '*.mp3' -exec cp –parents \{\} ~/OtroDirectorio \;
$ find . -name '*.mp3' | cpio -pdm ~/OtrDirectorio
$ find ~/miDirectorio -name '*.mp3' -exec cp –parents \{\} ~/OtrDirectorio \;
$ rsync -a -m –include '*/' –include '*.mp3' –exclude '*' ~/miDirectorio/ ~/OtrDirectorio
$ rsync -a –prune-empty-dirs –include '*/' –include '*.mp3' –exclude '*' ~/miDirectorio/ ~/OtrDirectorio
```

el primer y segundo ejemplo busca a partir de la actual trayectoria y los copia en *~/OtroDirectorio*, en los demás ejemplos, inicia la búsqueda en *~/midirectorio/* y los copia en *~/OtroDirectorio*.

Para eliminar múltiples archivos del tipo que buscamos, tenemos dos opciones:

```
$ find . -name "archivo" -exec rm -rf {} \;
```

o

```
$ find . -type f -name "archivo" -exec rm -rf {} \;
```

la diferencia entre ambos comandos indicados arriba es que el primero localiza y borra archivos y directorios, y el segundo sólo archivos:

Usando el comando Locate En este punto, podrías cuestionar la necesidad de tener una alternativa para el comando *find*. Después de todo, hace todo lo necesario para buscar archivos. Sin embargo, locate es una alternativa útil, ya que es más rápido que find para buscar en todo el sistema.

Por defecto, GNU/Linux no viene con el comando locate preinstalado. Para obtener el paquete, ejecuta los siguientes comandos en tu terminal:

```
# apt install mlocate
```

Ahora que has adquirido locate, puedes usarlo para buscar archivos así:

```
$ locate archivo
```

Puedes usar el argumento *-b* para una búsqueda más específica. Solo buscará el "nombre base" del archivo, enumerando efectivamente solo los archivos que tienen el término de búsqueda en lugar de devolver los directorios que conducen a los archivos.

```
$ locate -b archivo
```

Otros argumentos disponibles incluyen:

-e muestra entradas de archivos existentes en el momento en que se ejecuta el comando locate.

-q inhabilita la visualización de errores encontrados en el proceso de búsqueda.

-c muestra la cantidad de archivos que coinciden, en lugar de los nombres de los archivos

El comando locate estándar a veces puede mostrar archivos que han sido eliminados. Esto se debe a que el comando locate busca en la base de datos principal del sistema operativo GNU/Linux. Si esa base de datos no se actualiza, incluso los archivos eliminados pueden aparecer en los resultados de búsqueda. Puedes actualizar manualmente la base de datos ejecutando lo siguiente:

```
# updatedb
```

7.3 Empaquetadores, Compresores y Descompresores

La herramienta de empaquetado por excelencia en Linux y Unix es *tar.* Simplemente se trata de meter o guardar todos los archivos o directorios que necesitemos en un mismo archivo. Una forma sencilla de transportar archivos de un sitio a otro, ya sea en nuestra máquina o entre diferentes máquinas. Además *tar*, no solo te permite empaquetar esos archivos y directorios, sino que además te da la posibilidad de comprimirlos para de esta manera que ocupen menos espacio, y sean más fáciles de transportar.

Existen otros programas para manejar de manera óptima y fácil la agrupación y la compactación de un conjunto de archivos, como *gzip, bz2, xz, zip, lha, arj, rar, etc.* La mayoría de los programas para comprimir y descomprimir son secuenciales (solo usan un core), pero hay otros que trabajan en paralelo, que permiten usar dos o más cores logrando un alto desempeño como: pigz, plzip, pbzip2, lrzip, lbzip2, pixz.

Para instalar los programas más comunes en Debian GNU/Linux usar:

```
# apt install arj bzip2 clzip cpio dtrx gzip kgb \
lbzip2 lhasa lrzip lzip lziprecover lzop ncompress \
p7zip-full p7zip-rar pax pbzip2 pigz pixz plzip \
rar rarcrack tarlz unace unace-nonfree unar unp \
unrar unzip xz-utils zip zpaq zstd zutils \
archivemount
```

Podemos conocer los programas que disponemos en nuestro equipo para compresión usando:

```
$ apropos compress
```

Lo Básico Empaquetar y Desempaquetar Empecemos por lo más básico, que es empaquetar y desempaquetar usando el comando *tar.* Lo distingo de comprimir y descomprimir, porque esto lo veremos más adelante, ya que se trata de dos conceptos distintos.

Para que sea más sencillo, creamos un conjunto de archivos con los que trabajaré a continuación. Para ello ejecuta lo siguiente:

```
$ touch archivo{1..20}.txt
```

Empaquetando Archivos se han creado 20 archivos que vamos a empaquetar a continuación. Para ello, ejecutamos lo siguiente[53]:

$ tar -cvf empaquetado.tar *.txt

ahora tenemos todos los archivos en un mismo paquete.

¿Que son esas opciones que he puesto? -c es para crear un archivo
-v muestra el progreso del comando
-f para indicar el archivo que se va a crear en este caso[54]

Desempaquetando Archivos ahora que ya tenemos claro como empaquetar archivos, es necesario que se puedan desempaquetar. Es tan importante una operación como otra. Así, para desempaquetar, tenemos que utilizar la opción *-x*, de extraer. Así, para desempaquetar, ejecuta la siguiente instrucción:

$ tar -xvf empaquetado.tar

Extraer un Archivo de un Paquete para extraer un o algunos archivos del paquete ejecutamos:

$ tar -xvf empaquetado.tar archivo1 archivo2

Extraer un grupo de Archivos de un Paquete si necesitamos extraer un grupo de archivos podemos usar comodines, por ejemplo:

$ tar -xvf empaquetado.tar –wildcards '*.php'

[53] Esto también se puede poner por separado, como:
$ tar -c -v -f empaquetado.tar *.txt

[54] Tienes que tener en cuenta que -f tiene que preceder al nombre del archivo que vas a crear, ya sea que lo utilices por separado o junto con otras opciones. Quiero decir, que -cvf empaquetado.tar funcionará, pero en cambio -fcv empaquetado.tar arrojará un error, esto es algo que debemos tener en cuenta.

Listando el Contenido para conocer el contenido del archivo *.tar* usamos:

$ tar -tvf empaquetado.tar

Si no indicamos la opción -v solo mostrará los archivos empaquetados. Utilizando esta opción, además muestra otra información como el propietario, los permisos o la fecha.

Verificar un Archivo Empaquetado para verificar el contenido de un archivo empaquetado, usamos:

$ tar -tvfW empaquetado.tar

Quitando un Archivo del Paquete Una vez creado un determinado paquete, es posible que necesitemos quitar un archivo, es decir, que el archivo no pertenezca a ese nuevo paquete. Para hacer esto, debemos tener en cuenta que el paquete no puede estar comprimido. En caso de que esté comprimido no funcionará. Así, para borrar un archivo de un paquete ejecutamos:,

$ tar -f empaquetado.tar –delete archivo12.txt

Añadir un Archivo a un Paquete otra situación común que nos encontramos con más o menos frecuencia es cuando ya creamos el paquete, y dejamos un archivo fuera. En este caso, hay varias opciones, desde empaquetar de nuevo, a simplemente añadir ese archivo al paquete. Así, por ejemplo, si lo que queremos es simplemente añadir el archivo al paquete recurrimos a esta opción:

$ tar -f empaquetado.tar –append archivo12.txt

pero, ¿y si el archivo ya existe y simplemente lo queremos modificar? En este caso, usamos la opción:

$ tar -f empaquetado.tar –update archivo12.md

No solo esto, sino que también podemos comparar si un archivo que tenemos empaquetado es distinto del que no está empaquetado. Lo cierto es que lo vamos a comparar por fecha y por tamaño, únicamente, pero por lo menos ya sabemos si se ha modificado. Para hacer esto, ejecutamos:

$ tar -f empaquetado.tar –diff file12.txt

Esto devolverá un resultado como:

archivo12.txt: La fecha de modificación es distinta

archivo12.txt: El tamaño es distinto

Excluir Archivos o Directorios es común al respaldar excluir algunos archivos o directorios de nuestro respaldo, para ello podemos usar la opción *–exclude* en el que podemos indicar archivo que deseamos excluir del empaquetado, por ejemplo:

$ tar -cvf empaquetado.tar *.txt –exclude='archivo.txt'

o podemos indicar el directorio a excluir:

$ tar -cvf empaquetado.tar *.txt –exclude=carpeta/ignoraEsta

Cifrar y Descifrar es posible cifrar[55] el contenido que deseamos empaquetar usando OpenSSL para generar un archivo con extensión *.tar.gz*, por ejemplo, para cifrar el contenido del actual directorio usamos:

$ tar -czf - * | openssl enc -e -aes256 -out archivoSeguro.tar.gz

y para hacer el proceso de descifrado usamos:

$ openssl enc -d -aes256 -in archivoSeguro.tar.gz | tar xz -C prueba

el cual será extraído en el subdirectorio *prueba*.

[55] El cifrado consiste en ocultar y mantener en secreto la información cifrandola, el cifrado garantiza que la información sin formato cuando se almacena o envía por la red sea ilegible. El descifrado permite volver hacer legible la información. Existen distintas formas de cifrado: cifrado simétrico y asimétrico.

Comprimir y Descomprimir Hasta el momento todo lo memos hecho empaquetando archivos, pero si lo que requerimos es reducir el tamaño total ocupado, tenemos que comprimir esos paquetes. Respecto a la compresión hay distintas opciones:

- gzip es la compresión por defecto, y la puedes declarar utilizando la opción -z.
- bzip2 en este caso, tienes que utilizar la opción -j.

Ahora necesitamos combinar estas dos opciones con todo lo que hemos visto hasta el momento. Así para empaquetar y comprimir utilizamos:

```
$ tar -cvzf empaquetado.tar.gz *.txt
```

Para el caso de descomprimir:

```
$ tar -xvzf empaquetado.tar.gz
```

Para ver el contenido del archivo:

```
$ tar -tvf empaquetado.tar.gz
```

Para revisar la integridad de un archivo usamos:

```
$ gunzip -t empaquetado.tar.gz
```

también podemos integrar lo visto anteriormente en un programa Bash como el mostrado en (véase **??**).

Ahora bien, existe una diferencia en cuanto a las posibilidades que ofrece un archivo empaquetado y otro comprimido. Esta diferencia radica en las operaciones que podemos efectuar. Así, con un archivo comprimido, no podremos actualizarlo, al menos en primera instancia. Es decir, no podemos utilizar las opciones –delete, –append o –update.

Una solución es desempaquetar, operar y comprimir, como por ejemplo:

```
$ gunzip empaquetado.tar.gz; tar -f empaquetado.tar –delete
archivo5.txt; gzip empaquetado.tar
```

Estas son algunas de las opciones que permite hacer *tar*. Y digo bien, simplemente algunas de las opciones, y sin lugar a dudas las más habituales. Y es que *tar* es toda una navaja suiza que permite empaquetar archivos y directorios de decenas de formas posibles, no solo lo que hemos visto hasta el momento. También, se puede especificar directorios, excluir archivos por nombre o incluso por patrón, y mucho más.

Cifrar y Descifrar Archivos es posible cifrar archivos usando criptografía de clave pública o sólo una clave de cifrado simétrico. La clave que se usa para el cifrado simétrico deriva de la contraseña dada en el momento de cifrar el documento.

GnuPG es un paquete completo que permite generar un par de claves públicas, intercambiar y comprobar la autenticidad de claves, cifrar y descifrar archivos, etc. Para instalar el paquete GnuPG, usamos:

```
# apt install gnupg
```

Cifrar un Archivo para cifrar un archivo se usa el programa *gpg* con la opción *-c* la cual nos permitirá indicar la clave de cifrado y su confirmación, por ejemplo:

```
$ gpg -c archivo.tar
```

el archivo resultante será: archivo.tar.pgp. Es necesario escoger claves de cifrado robustas, una opción para generarlas es GnuPG, para generar una clave aleatoria de 32 caracteres, usamos:

```
$ gpg –gen-random –armor 1 32
```

Descifrar un Archivo para descifrar un archivo cifrado con GnuPG, usamos el comando *gpg* al cual se le indica el nombre del archivo cifrado, por ejemplo:

```
$ gpg archivo.tar.pgp
```

el archivo resultante será: archivo.tar.

Cifrar y Descifrar Usando tar podemos generar un archivo compactado y cifrarlo en una mismo comando mediante:

```
$ tar -cvzf - directorio | gpg -c > archivo.tar.gz.gpg
```

también podemos indicar la clave de cifrado -no es seguro pues otro usuario podría verla- mediante:

```
$ tar -cvzf - directorio | gpg -c –passphrase miclave > archivo.tar.gz.gpg
```

y para descifrar usamos:

```
$ gpg -d archivo.tar.gz.gpg | tar -xvzf -
```

Cifrar y Descifrar Usando ccrypt este comando permite cifrar usando *AES256*, no viene instalado por omisión en el sistema, pero lo podemos instalar usando:

```
# apt install ccrypt
```

Para cifrar un archivo *archivo.tar* usamos:

```
$ ccrypt archivo.tar
```

esto nos generará un archivo *archivo.tar.cpt*, para descifrarlo usamos:

```
$ ccrypt -d archivo.tar.cpt
```

que nos retornará a nuestro archivo original.

Cifrar y Descifrar Usando age este comando permite cifrar usando una clave pública o frase, no viene instalado por omisión en el sistema, pero lo podemos instalar usando:

```
# apt install age
```

Para generar una clave pública en el archivo llave.txt usamos:

```
$ age-keygen -o llave.txt
```

Para cifrar un archivo *archivo.tar* con clave pública usamos:

```
$ tar cvz archivos | age -r llave.txt > archivo.tar.gz.age
```

para descifrarlo usamos:

```
$ age –decrypt -i llave.txt > archivo.tar.gz
```

que nos retornará a nuestro archivo original.
Para cifrar un archivo *archivo.tar* con frase usamos:

```
$ age –passphrase –output archivo.tar.gz.age archivo.tar.gz
```

para descifrarlo usamos:

```
$ age –decrypt –output archivo.tar.gz archivo.tar.gz.age
```

que nos retornará a nuestro archivo original.

Respaldos Incrementales Un respaldo incremental es aquel que almacena solamente las diferencias con respecto al respaldo anterior, tar posee dos opciones que nos van a servir para trabajar con respaldos incrementales:

- «-listed-incremental=file», o «-g file», que permite especificar un archivo de respaldo incremental.
- «-incremental» o «-G», que permite analizar un respaldo incremental y saber qué archivos del total están presentes en dicho respaldo.

El archivo de respaldo incremental almacenará metadatos (.snar) que ayudará a la herramienta tar a determinar qué archivos han cambiado desde el último incremental, cuáles se han agregado, o cuales se han eliminado, de modo que tar podrá generar el siguiente respaldo incremental solamente de los cambios del anterior. Por ejemplo:
Creamos el respaldo completo (la base para los respaldos incrementales)[56]:

```
$ tar -cvzf Respaldos/bkp0.tgz -g Respaldos/incremental.snar
datos/
```

Supongamos ahora que realizamos el respaldo incremental del siguiente día, usando;

```
$ tar -cvzf Respaldos/bkp1.tgz -g Respaldos/incremental.snar
datos/
```

de esta manera realizaremos todos los respaldos incrementales necesarios.
Veamos, antes de proceder a restaurar todo, cómo podemos analizar qué contiene cada respaldo incremental. Esto podemos llevarlo a cabo mediante la opción «-incremental» del comando tar, a la que debemos también agregar dos modificadores «-verbose» para ver la información completa. En mi caso he usado «-G» en vez de «-incremental», y «-vv» en vez de «-verbose -verbose»:

```
$ tar -tvvGf Respaldos/bkp1.tgz
```

[56] Aquí usamos para los ejemplos de los respaldos comprimidos con gzip, pero el comportamiento es exactamente el mismo al utilizar bzip2, xz, lmza o cualquier otro formato soportado.

como puede verse en la salida, delante de cada nombre de archivo tenemos un «Y» o un «N». «Y» significa que el archivo está presente en dicho respaldo, y «N» que no lo está. El resultado es consistente con los cambios que hemos realizado sobre los archivos.

Supongamos ahora que perdemos todos los datos y que necesitamos recuperarlos desde el respaldo completo y los incrementales. Para extraer todos los archivos en el mismo estado exacto en el que estaban a la hora de realizar el último respaldo incremental, debemos usar la opción «-extract» o «-x» junto con la opción «-listed-incremental». Como la información de respaldo incremental está almacenada dentro del Tarball, es decir, el Tarball contiene solo los cambios respecto del último incremental, puede pasarse cualquier argumento al «-listed-incremental» o «-g». Usualmente se utiliza /dev/null, o se usa directamente «-incremental» o «-G».

Suponiendo que el directorio datos no existe, vamos a proceder a restaurar todos los respaldo, desde el primero, completo, uno a uno hasta el último incremental disponible, para regenerar el directorio original y su contenido:

```
$ tar xvGf Respaldos/bkp0.tgz
$ tar xvGf Respaldos/bkp1.tgz
$ tar xvGf Respaldos/bkpn.tgz
```

de esta forma se puede restaurar paso por paso todos los respaldo que antes hicimos, y ahora tenemos dentro del directorio *datos/* toda la información de nuevo. Y sí, los archivos tienen el contenido original modificado tal y como quedaron en el último respaldo incremental.

Zips de la Muerte Cuando descomprimimos un archivo debemos tener cuidado, ya que el resultado suele ocupar bastante más y puede dejarnos sin espacio en el ordenador. El ratio máximo de compresión de la mayoría de zips está marcado en 1032 a uno, aunque en muchas ocasiones tampoco se alcanza. Sin embargo, desde 1996 se tiene constancia de la existencia de las llamadas "bombas zip" o lo que se conoce popularmente como "el zip de la muerte". Un archivo que sobrepasa este límite de descompresión e inunda nuestro ordenador con miles de millones de Bytes.

Una bomba zip es un archivo comprimido como cualquier otro. Su potencial peligro está en que es muy fácil de esconder y compartir. Pero al descomprimir es cuando colapsa el ordenador al liberar una gran cantidad de datos.

El zip de la muerte más conocido es 42.zip (https://unforgettable.dk), un archivo con un récord de compresión de 106.000 millones a uno. Su autor es desconocido, pero el ratio es impresionante. Estamos hablando de un archivo que pesa únicamente 42 KiloBytes (de ahí el nombre) y que al descomprimirse puede alcanzar los 4.3 GigaBytes. Pero la clave está en que el archivo contiene cinco capas, cada una con 16 archivos. Lo que en total una vez liberados todos ocupan un total de 4.5 PetaBytes.

David Fifield (https://www.bamSoftware.com/hacks/zipbomb/) nos ofrece tres archivos de ejemplo (uno de ellos no recursivo que establece el ratio en 28 millones a uno). El primero donde pasamos de 42kB a 5.5GB, un segundo de 10MB a 281TB y un tercero de 46MB a 4.5 PetaBytes, este último únicamente compatible con el formato Zip64.

Los antivirus sí las detectan afortunadamente para la seguridad de los usuarios, esta técnica de la bomba zip es bastante conocida desde hace muchos años. Pese a que el archivo comprimido pasa desapercibido en el sistema, una vez vamos a descomprimirlo es cuando los antivirus lo detectan como un posible peligro y nos advierten antes de iniciar el proceso.

También los hay en formato TAR sin tratarse de ningún gusano o virus, otro ataque similar es la 'bomba fork' o 'wabbit'. En este caso, es un archivo que crea copias de sí mismo. Una técnica de la que los usuarios de Linux tampoco se libran, ya que también funciona con archivos TAR.

Trabajando con Archivos Compactados Existe la opción de gestionar ficheros comprimidos, gracias a los comandos *zgrep, zegrep, zless, zmore, zdiff, zcmp, zcat* entre otros. Como te puedes dar cuenta la función de cada uno de estos comandos será la misma que su homónimos sin «z» (*grep, egrep, less, more, diff, cmp, cat*) pero para ficheros comprimidos con *gzip* y extensión *.gz*. A saber:

- zcat: Mostar por pantalla el contenido de un fichero comprimido.

  $ zcat archivo.gz | more

  en el caso de necesitar conocer las propiedades del archivo compactado, usamos:

```
$ zcat -l archivo.gz
```

y en el caso de que el comando *zcat* nos muestre avisos, los podemos callar usando:

```
$ zcat -q archivo.gz
```

- zless y zmore: Comandos para examinar ficheros de texto plano o comprimidos de forma paginada por la pantalla.

```
$ zless archivo.gz
```

- zgrep y zegrep: Comando para realizar búsquedas de expresiones regulares en ficheros comprimidos.

```
$ zgrep -i "cadena" archivo.gz
```

- zdiff y zcmp: Comando para comparar ficheros comprimidos.

```
$ zdiff archivo1.gz archivo2.gz
```

De forma análoga para archivos comprimidos usando *bz2*, están los comandos *bzgrep, bzegrep, bzless, bzmore, bzdiff, bzcmp, bzcat* y para archivos comprimidos usando *xz*, están los comandos *xzgrep, xzegrep, xzless, xzmore, xzdiff, xzcmp, xzcat*.

Desempaquetar o Descomprimir El uso básico para desempaquetar o descomprimir según la extensión del archivo es el siguiente:

- *.tar Desempaquetar usando: tar xf archivo.tar
- *.tar.bz2 Descomprimir usando: tar xjf archivo.tar.bz
- *.tbz Descomprimir usando: tar xjf archivo.tbz
- *.tbz2 Descomprimir usando: tar xjf archivo.tbz2
- *.tgz Descomprimir usando: tar xzf archivo.tgz
- *.tar.gz Descomprimir usando: tar xzf archivo.tar.gz

- *.tar.lz Descomprimir usando: tar -xf archivo.lz
- *.tar.xz Descomprimir usando: tar -xf archivo.tar.xz
- *.xz Descomprimir usando: tar Jxf archivo.xz
- *.gz Descomprimir usando: gunzip archivo.gz
- *.bz2 Descomprimir usando: bunzip2 archivo.bz2
- *.zip Descomprimir usando: unzip archivo.zip
- *.Z Descomprimir usando: uncompress archivo.Z
- *.7z Descomprimir usando: 7z e archivo.7z
- *.zst Descomprimir usando: zstd -d archivo.zst
- *.lha Descomprimir usando: lha x archivo.lha
- *.arj Descomprimir usando: arj x archivo.arj
- *.zoo Descomprimir usando: zoo x archivo.zoo
- *.lz Descomprimir usando: lzip -d archivo.lz
- *.cpio Desempaquetar usando: cpio -idv < archivo.cpio
- *.rar Descompactar usando: rar x archivo.rar
- * Descompactar[57] usando: unp archivoCompactado
- * Descompactar usando: dtrx archivoCompactado
- * Descompactar usando: unar archivoCompactado

[57] El programa unp permite descomprimir un archivo de casi cualquier formato, para instalar el paquete usamos:

```
# apt install unp
```

para descompactar, usamos:

```
$ unp archivoCompactado
```

El proceso de instalación y uso es similar para los paquetes: dtrx y unar.

Formato *tar.gz* para respaldar y compactar un grupo de archivos y/o directorios en formato *tar.gz*, usamos:

```
$ tar -cvf nombre.tar.gz directorio
$ gzip –best nombre.tar
```

o usamos:

```
$ tar -zcvf nombre.tar.gz directorio
```

o usamos:

```
$ tar -jcvf nombre.tar.bz2 directorio
```

para restaurar, usamos:

```
$ gunzip nombre.tar.gz
$ tar -xvf nombre.tar
```

o usamos:

```
$ tar -zxvf nombre.tar.gz
```

o usamos:

```
$ tar -jxvf nombre.tar.bz2
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.tar.gz; do tar -zcvf "$z"; done
```

Formato *gz* para compactar archivos y/o directorios al formato *gzip*, usamos:

```
$ gzip ficheros
```

y para descompactarlo usamos:

```
$ gzip -dk fichero.gz
```

si solo nos interesa descompactar y no preservar el archivo .gz:

```
$ gzip -d fichero.gz
```

o

```
$gunzip fichero.gz
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.gz; do gunzip "$z"; done
```

También podemos hacer la compactación en paralelo usando múltiples cores, mediante el paquete *pigz* con formato *gzip* (-0 sin compresión, -1 compresión más rápida, -6 compresión por omisión y -9 mejor compresión), para compactar manteniendo el archivo original, usamos:

```
$ pigz -k archivo.iso
```

generara un archivo.izo.gz. Podemos listar el contenido de un .gz usando:

```
$ pigz -l archivo.iso.gz
```

Para comprimir un directorio, es necesario usar tar, mediante:

```
$ tar cf - Directorio/ | pigz > archivo.tar.gz
```

o

$ tar –use-compress-program=pigz -cf archivo.tar.gz Directorio/

Para descomprimir usamos:

$ pigz -d archivo.iso.gz

o

$ unpigz archivo.iso.gz

Por omisión *pigz* usa todos los cores de la máquina, si necesitamos limitar el número de cores usamos la bandera -p y el número de cores a usar, por ejemplo:

$ pigz -9 -k -p3 archivo.iso

Podemos usar otros formatos de compresión, usando la opción -k -z para zlib (.zz), usando:

$ pzip -k -k archivo.iso

o usando la opción -k -K para zip (.zip):

$ pzip -k -K archivo.iso

Formato *bz2* para compactar archivos y/o directorios al formato *bz2, bz, tbz2 tbz o bzip2*, usamos:

$ bzip2 ficheros

y para descompactarlo usamos:

$ bzip2 -d fichero.bz2

podemos descomprimir usando el comando **bzcat** mediante:

$ bzcat -dc fichero.bz2

o bien descomprimir usando el comando **bunzip2** mediante:

$ bunzip2 fichero.bz2

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

$ for z in *.bz2; do bunzip2 "$z"; done

Formato *xz* para respaldar y compactar un grupo de archivos y/o directorios en formato *tar.xz*, usamos:

```
$ tar -cvJf nombre.tar.xz directorio
$ xz nombre.tar
$ xz -k archivo.tar
$ tar –xz -cf - /trayectoria | ssh usuario@maquina "cat > archivo.txz"
```

podemos controlar el nivel de compresión usando valores de 0 (sin compresión) al 9 (mejor compresión), mediante:

```
$ xz -9 archivo.tar
$ xz –k9 archivo.tar
```

podemos descomprimir usando:

```
$ tar -xf nombre.tar.xz
$ tar -xvf nombre.tar.xz
$ tar -Jxvf nombre.tar.xz
$ tar -xf nombre.tar.xz
$ tar –xz -xf archivo.txz
$ xz -v -d nombre.tar.xz
$ xz –decompress archivo.tar.xz
$ unxz archivo.xz
```

y podemos ver el contenido de un archivo *.tar.xz* si usamos:

```
$ tar ft archivo.tar.xz
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.tar.xz; do tar-xf "$z"; done
```

Formato *zip* permite respaldar en un solo archivo un grupo de archivos y/o directorios compactándolos, para ello usar:

```
$ zip archivo.zip fichero(s)
$ zip -r archivo.zip directorio/
```

también permite establecer los niveles de compresión usando una escala de 0 a 9 (por omisión es 6), por ejemplo:

```
$ zip -8 archivo.zip fichero(s)
$ zip -8 -r archivo.zip directorio/
```

y podemos borrar los archivos que se comprimirán al terminar el proceso, usando:

```
$ zip -m archivo.zip fichero(s)
$ zip -rm archivo.zip directorio/
```

Podemos agregar un archivo a un *.zip* existente, usando:

```
$ zip -u archivo.zip nuevo.txt
```

o eliminar un archivo a un .zip existente, usando:

```
$ zip -d archivo.zip porBorrar.txt
```

Podemos crear un *.zip* protegido con clave, usando:

```
$ zip -e archivo.zip fichero(s)
```

si necesitamos poner una clave de acceso (digamos 2GAXTW), usamos

```
$ zip -P 2GAXTW -r archivo.zip ficheros
```

o proteger con clave un *.zip* ya existente, usando:

```
$ zipcloak archivo.zip
```

Podemos ver la información detallada del archivo comprimido, usando:

```
$ zipdetails archivo.zip
```

Y descomprimir usando:

$ unzip archivo.zip

para descompactar un solo directorio usamos:

$ unzip -d directorio archivo.zip

también podemos indicar el directorio en en cual descomprimir, usando:

$ unzip archivo.zip -d /home/usuario/temp/

para descomprimir un *.zip* protegido con clave (digamos 2GAXTW), usamos:

$ unzip -P 2GAXTW archivo.zip

para extraer sin mostrar salida de los archivos extraídos, usamos:

$ unzip -q archivo.zip

para indicar que no se extraigan algunos archivos, usamos:

$ unzip archivo.zip -x *.eps

para sobreescribir los archivos al descomprimir sin pedir confirmación, usamos:

$ unzip -o archivo.zip

para no sobreescribir los archivos al descomprimir, usamos:

$ unzip -n arvhivo.zip

para actualizar archivos y en su caso crearlos si es necesario, usamos:

$ unzip -u archivo.zip

para actualizar archivos pero no crear nuevos, usamos:

$ unzip -f archivo.zip

para listar el contenido de un *.zip*, usamos:

$ unzip -l archivo.zip

para obtener información detallada de un *.zip*, usamos:

$ unzip -v archivo.zip

para revisar la integridad de un *.zip*, usamos:

$ unzip -t archivo.zip

Si al descompactar existe algún error, es posible recuperar parte de los archivos mediante:

$ zip -F archive.zip -O archive-fixed.zip

o usar *-FF*, después usar:

$ jar xvf archive-fixed.zip

otra alternativa es usar:

$ bsdtar xf archivo.zip

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

$ for z in *.zip; do unzip "$z"; done

Formato *7z* para respaldar y compactar un grupo de archivos y/o directorios en formato *7-zip*, usamos:

$ 7z a archivo.7z ficheros

también podemos pedir que use una clave (password$$) para cifrar el archivo:

$ 7z a archivo.7z ficheros -ppassword$$

para restaurar usamos:

$ 7z e archivo.7z

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

$ for z in *.7z; do 7z e "$z"; done

Formato *zst* para respaldar y compactar un grupo de archivos y/o directorios en formato *zst*, usamos:

```
$ zstd ficheros -o archivo.zst
```

para restaurar usamos:

```
$ zstd -d archivo.zst
```

o usamos:

```
$ unzstd archivo.zst
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.zst; do unzstd "$z"; done
```

Formato *lha* para respaldar y compactar un grupo de archivos y/o directorios en formato *lha*, usamos:

```
$ lha a archivo.lha ficheros
```

para restaurar usamos:

```
$ lha x archivo.lha
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.lha; do lha x "$z"; done
```

Formato *arj* para respaldar y compactar un grupo de archivos y/o directorios en formato *arj*, usamos:

```
$ arj a archivo.arj ficheros
```

para restaurar usamos:

```
$ arj x archivo.arj
$ unarj archivo.arj
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.arj; do arj x "$z"; done
```

Formato *zoo* para respaldar y compactar un grupo de archivos y/o directorios en formato *zoo*, usamos:

```
$ zoo a archivo.zoo ficheros
```

para restaurar usamos:

```
$ zoo x archivo.zoo
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.zoo; do zoo x "$z"; done
```

Formato lz y tlz para compactar un *archivo* y remplazarlo por *archivo.lz*, usamos:

```
$ lzip -v archivo
```

para compactar todo un dispositivo (por ejemplo */dev/sdc*), usamos:

```
$ lzip -c /dev/sdc > archivo.lz
```

para restaurar usamos:

```
$ tar -xf archivo.lz
$ lzip -d archivo.lz
```

o extraer un archivo multivolumen de un archivo tar

```
$ lzip -cd archivo.tar.lz | tar -xf -
```

Podemos usar *tarlz* que es una combinación de *tar* y *lzip* que trabaja en paralelo para hacer la compresión, por ejemplo de los archivos *a*, *b*, y *c*:

```
$ tarlz -cf archivo.tar.lz a b c
```

y podemos adicionarle los archivos *d* y *e*, usando:

```
$ tarlz -rf archivo.tar.lz d e
```

Para extraer los archivos podemos usar:

```
$ tarlz -xf archivo.tar.lz
```

También podemos usar la versión plzip que trabaja en paralelo. Para compactar un *archivo* y remplazarlo por *archivo.lz*, usamos:

```
$ plzip -v archivo
```

para compactar todo un dispositivo (por ejemplo */dev/sdc*), usamos:

```
$ plzip -c /dev/sdc > archivo.lz
```

y para restaurar usamos:

```
$ plzip -d archivo.lz
```

Formato cpio para empaquetar archivos (por ejemplo*.txt) en un archivo .cpio, usamos:

```
$ ls *.txt | cpio -ov > archivo.cpio
```

para desempaquetar, usamos:

```
$ cpio -idv < archivo.cpio
```

Formato *rar* para respaldar y compactar un grupo de archivos y/o directorios al formato *rar*, usamos:

```
$ rar a archivo.rar ficheros
```

para restaurar usamos:

```
$ rar x archivo.rar
$ unrar archivo.rar
```

En algunos casos, archivos *rar* de Windows no es posible descomprimirlos correctamente en Linux, para descomprimirlos podemos descargar utilerías GNU para Win32 de:

http://unxutils.sourceforge.net/

entre ellas, descargar *unrar* (es un sólo archivo zip) de la dirección:

http://sourceforge.net/projects/unxutils/

ahora usando Wine es posible descomprimir los archivos desde Linux mediante:

```
$ wine unrar.exe e archivo.rar
```

Para descomprimir archivos *rar* o *zip* rotos, usar:

```
$ unrar e -kb -y nombreArchivo.rar
```

o usamos:

```
$ bsdtar xf nombreArchivo.zip
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.rar; do unrar e "$z"; done
```

UNP para descompactar casi de cualquier formato existe un programa llamado *unp* (basado en un Scrip de Perl) que permite descomprimir de casi cualquier formato, para instalarlo hacemos:

```
# apt install unp
```

Es un extractor universal, su uso es de lo más sencillo, a saber:

```
$ unp archivoCompactado
```

Parte de los formatos soportados y por que programas se muestra en la siguiente lista (generada usando: *unp -s*):

7z (p7zip or p7zip-full), ace (unace), ar,deb (binutils), arj (arj), bz2 (bzip2), cab (cabextract), chm (libchm-bin or archmage), cpio,afio (cpio or afio), dat (tnef), dms (xdms), exe (orange or unzip or unrar or unarj or lha), gz (gzip), cpio,afio (cpio or afio), dat (tnef), dms (xdms), exe (orange or unzip or unrar or unarj or lha), gz (gzip), hqx (macutils), lha,lzh (lha), lz(lzip), lzma (xz-utils or lzma), lzo (lzop), lzx (unlzx), mbox (formail and mpack), pmd (ppmd), rar (rar or unrar or unrar-free), rpm (rpm2cpio and cpio), sea,sea.bin (macutils), shar (sharutils), tar (tar), tar.bz2,tbz2 (tar with bzip2), tar.lzip (tar with lzip), tar.lzop,tzo (tar with lzop), tar.xz,txz (tar with xz-utils), tar.z (tar with compress), tgz,tar.gz (tar with gzip), uu (sharutils), xz (xz-utils), zip,cbz,cbr,jar,war,ear,xpi,adf (unzip), zoo (zoo).

DTRX para descompactar de múltiples formatos esta herramienta llamada *dtrx* (Do The Right Extraction), es una aplicación de código abierto que simplifica el proceso de extracción de archivos comprimidos. Para su instalación hacemos:

```
# apt install dtrx
```

Y ya estamos listos para comenzar a usarlo en la terminal. Dtrx soporta los formatos comunes de compresión como: tar, bz2, zip, cpio, deb, rpm, gem, 7z, cab, lzh, rar, gz, bz2, lzma, xz, además de binarios deb, gem (ruby), archivos de Installshield e incluso algunos tipos de exe. También descomprime archivos comprimidos con gzip, bzip2, lzma, xz, y otros compresores.

Para descomprimir usamos:

```
$ dtrx archivo.zip
```

Podemos hacer un preview del contenido de un archivo lanzandolo con el parámetro -l:

```
$ dtrx -l archivo.zip
```

Si tenemos un archivo comprimido que a su vez incluye múltiples ficheros, al ejecutar *dtrx* nos va a dar 5 opciones diferentes, para hacer la cosa más o menos automática:

-a siempre extraer los archivos incluidos durante esta sesión

-o extraer los archivos incluidos esta vez.

-n elegir no extraer los archivos incluidos por esta vez.

-v nunca extraer los archivos incluidos durante esta sesión.

-l listar los archivos incluidos.

Si queremos que trabaje de forma recursiva, lo podemos hacer añadiendo la opción -r:

```
$ dtrx -r archivo.tar.gz
```

Además con el parámetro -m podemos extraer los metadatos de archivos *deb* y *gem*:

```
$ dtrx -m archivo
```

UNAR para descompactar de múltiples formatos esta herramienta llamada *unar*, es una aplicación de código abierto que simplifica el proceso de extracción de archivos comprimidos. Para su instalación hacemos:

```
# apt install unar
```

Y ya estamos listos para comenzar a usarlo en la terminal, *unar* soporta los formatos comunes de compresión como: zip, rar, 7z, tar, gzip, bzip2, lzma, xz, cab, msi, nsis, exe, iso, bin, arj, arc, pakm, acem, lzh, adf, entre otros.

Para descomprimir usamos:

```
$ unar archivo.zip
```

Otras opciones cuando se tiene una lista de archivos de distintas trayectorias o es resultado de una búsqueda, para compactar es preferible usar *afio*. Podemos instalarlo usando:

```
# apt install afio
```

Para compactar, digamos todos los archivos **.?pp* (programas de C++) usar:

```
$ find . -name *.?pp | afio -o -Z fuentes
```

para descompactarlos, usar:

```
$ afio -i -Z fuentes
```

Si se desea compactar usando GZIP, usar:

```
$ cat lista | afio -o -Z -G 9 fuentes
```

Si se desea ver el listado de archivos que contiene fuentes, usar:

```
$ afio -t fuentes
```

Si se desea compactar y mandar a otra máquina usar:

```
$ find . -name *.?pp | afio -o -Z user@servidor%ssh:fuentes
```

Como el uso de *afio* no necesita extensión en el archivo, para descompactarlo de cualquier formato es recomendable usar *unp*, este escoge el mejor método para el archivo en cuestión:

```
$ unp archivoCompactado
```

Respaldo y/o Restauración Remoto es posible usar *ssh*[58] en conjunción con *tar o dd*[59] para hacer respaldos y/o restauraciones de forma remota, por ejemplo:

```
$ ssh user@maq tar czf - /dir1/ > /dest/arch.tar.gz
$ ssh user@maq 'cd /dir1/ && tar -cf - file | gzip -9' >arch.tar.gz
$ tar zcvf - /dir | ssh user@maquina "cat > /dest/arch.tar.gz"
$ tar zcf - /dir/ | gpg -e | ssh user@maq 'cat - > arch.tar.gz.gpg'
$ ssh user@maq 'tar zcf - /some/dir' | tar zxf -
$ ssh user@maq 'tar czf - /home/user' | tar xvzf - -C /home/user
$ cat arch.tar.gz | ssh user@maq "tar zxvf -"
$ cat arch.tar.gz | ssh user@maq "cd /dest/; tar zxvf -"
# dd if=/dev/sdvf | ssh user@maq 'dd of=arch.img'
$ ssh root@maq 'dd if=arch.img' | dd of=/dev/sdvf
$ tar cvzf - /dir | ssh root@maq "dd of=/dest/arch.tar.gz"
$ tar cvjf - * | ssh user@maq "(cd /dest/; tar xjf -)"
$ tar cvzf - dir/ | ssh user@maq "cat > /dest/arch.tgz"
$ tar cvzf - /dir | ssh user@maq "dd of=/dest/arch.tar.gz"
$ ssh user@maq "cat /dest/arch.tar.gz" | tar xvzf -
$ tar cvjf - * | ssh root@maq "(cd /dest/; tar xjf - )"
```

Es posible usar *nc*[60] en conjunción con *tar* y *pv* para hacer respaldos y/o restauraciones de forma remota, supongamos que estamos en el IP 192.168.13.230. podemos mandar un archivo grande (digamos un *.iso)* y usar el comando *pv* para ver el progreso de la transferencia usando:

$ tar -zcf - Archivo.iso | pv | nc -l -p 5555 -q 5

y para recibirlo en el otro equipo usamos:

```
$ nc 192.168.13.230 5555 | pv | tar -zxf -
```

Además, ponemos también usar *nc* en conjunción con *tar* y *gpg* para hacer respaldos y/o restauraciones de forma remota, supongamos que estamos en el IP 192.168.13.230, podemos generar un archivo compactado, cifrarlo y enviarlo a otro equipo en una mismo comando mediante:

[58] ssh o Secure Shell, es un protocolo de administración remota que le permite al usuario controlar y modificar servidores remotos a través de un mecanismo de autentificación.

[59] dd o Data Duplicator permite copiar y convertir datos -en Linux cualquier dispositivo y partición se trata y gestiona como un archivo- de archivos a bajo nivel.

[60] El comando *netcat* (*nc*) es una utilidad que permite escribir y leer datos a través de conexiones de red usando los protocolos TCP y UDP

```
$ tar -cvzf - directorio | gpg -c | nc -l 6666
```

y para recibirlo en el otro equipo usamos:

```
$ nc 192.168.13.230 6666 | gpg -d | tar -xvzf -
```

Siempre es mejor opción usar *scp*, pero *nc* cumplirá con su cometido.

Montar Archivos Compactados con Archivemount ¿Alguna vez haz necesitado mirar el contenido de un fichero .tar, .tar.gz, entre otros, pero sin tener que descomprimirlo? Tal vez te gustaría extraer solo unos pocos ficheros, puede que lo que te interese es trabajar con una carpeta a la que le modificamos archivos, sin tener que archivar esta carpeta cada cierto tiempo.

En este caso, tenemos un sistema de ficheros bastante interesante que se llama *archivemount*, y que nos permite ver un fichero .tar, .tar.gz, entre otros. Como si de una carpeta local más se tratara. Al desmontar este sistema de fichero, se crea el mismo fichero de nuevo con los cambios hechos, de forma automática, y con previa copia del anterior.

Archivemount es un sistema de archivos basado en FUSE para variantes de Unix, incluido Linux. Su propósito es montar archivos (es decir, tar, tar.gz, etc.) en un punto de montaje donde se puede leer o escribir como con cualquier otro sistema de archivos. Esto hace que el acceso al contenido del archivo, que puede estar comprimido, sea transparente para otros programas, sin descomprimirlos.

Para instalarlo usamos:

```
# apt install archivemount
```

Para usarlo, necesitamos un directorio cualquiera donde montar nuestro *archivo.tar.gz*, por ejemplo *mnt*, entonces hacemos:

```
$ archivemount archivo.tar.gz mnt
```

ahora podemos ingresar al directorio *mnt* y trabajar con los archivos que tenia nuestro *archivo.tar.gz*, podemos visualizar el contenido, agregar o borrar archivos y al concluir para guardar los cambios necesitamos desmontarlo usando:

```
$ fusermount -u mnt
```

Esto generará un nuevo *archivo.tar.gz* con los cambios y el archivo anterior se guardará en *archivo.tar.gz.orig*.

Nota: si el archivo es de varios GigaBytes, entonces el montado y acceso a los archivos puede ser un proceso lento, para ello se desarrollo *ratarmount* (aplicación en Python), que permite montar un archivo *.tar, tar.gz*, entre otros, en modo de solo lectura, pero es notoriamente rápido para archivos compactados de varios GigaBytes de tamaño.

Montar Archivos Compactados con Ratarmount permite montar para su lectura un archivos .tar, tar.gz, entre otros. Esta es una aplicación de Python que se puede instalar para todos los usuario usando *pip3*, para ello usamos:

```
# pip3 install ratarmount
```

o instalar a nivel de usuario usando *pip3*, para ello usamos:

```
$ pip3 install ratarmount
```

usando instalación a nivel de usuario, para montar (la primera vez creará el índice que permitirá el acceso rápido a los archivos), usamos:

```
$ ~/.local/bin/ratarmount archivo.tar.gz mnt
```

una vez generado el índice (este proceso es algo lento), podemos ingresar al directorio *mnt* y trabajar con los archivos que tiene nuestro *archivo.tar.gz*, podemos visualizar el contenido y copiar archivos con una velocidad de acceso alta. Al concluir su uso, necesitamos desmontarlo usando:

```
$ fusermount -u mnt
```

como parte del proceso de acceso a nuestros archivo, se genera un archivo tmpXXXX, el cual podemos borrar usando:

```
$ rm tmp*
```

para más información del proyecto, podemos consultar:

https://github.com/mxmlnkn/ratarmount

7.4 Copiar Archivos Entre Equipos

scp permite transferir archivos y/o directorios de una máquina a otra de forma cifrada usando *SSH* (Secure Shell) que es un protocolo de administración remota que le permite al usuario controlar y modificar servidores remotos a través de un mecanismo de autentificación[61]. El comando *scp* (Secure Copy Protocol) tiene una sintaxis similar al del comando *cp*, con la salvedad que es necesario indicar el usuario, la máquina y el subdirectorio de trabajo del archivo y/o directorio para el destino, fuente o ambos.

Por ejemplo, si se desea transmitir un archivo a una máquina *192.168.13.230* con usuario *antonio*, en el directorio *~/Datos/* estando en sesión en otra máquina, se usa la siguiente sintaxis:

```
$ scp archivo.dat antonio@192.168.13.230:~/Datos/
```

Si se desea transmitir un subdirectorio a la máquina *192.168.13.230*, en el directorio *home* del usuario (denotado con .), se usa la siguiente sintaxis:

```
$ scp -r Directorio antonio@192.168.13.230:.
```

también podemos decirle que excluya algunos archivos (*.mp3) de la copia, usando:

```
$ scp -r !(*.mp3) Directorio antonio@192.168.13.230:.
```

Si se desea copiar un archivo de una máquina remota a nuestra máquina, usamos:

```
$ scp antonio@192.168.13.230:~/archivo ~/destino/
```

o de forma alternativa usamos (. indica el directorio donde el usuario se encuentra):

```
$ scp antonio@192.168.13.230:~/archivo .
```

Si se desea copiar de una máquina remota a otra máquina remota, usamos:

[61] En Android podemos hacer uso de SSH/SFTP mediante aplicaciones como: Termius, JuiceSSH, Mobile SSH, Advanced Client app, ConnectBot.

```
$ scp user1@HOST1:~/archivo user2@HOST2:~/
```

Si se desea transferir múltiples archivos podemos usar:

```
$ scp file1.txt file2.txt user@HOST:/home/user/
```

o de forma alternativa usamos (. indica el directorio donde el usuario se encuentra):

```
$ scp user@Host:/home/user/\{file1.txt,file2.txt\} .
```

En el caso que se quiera limitar el ancho de banda en la transmisión de archivos por *scp*, usar:

```
$ scp -l 400 user@server:/home/user/* .
```

En el caso de que se desee usar otro puerto distinto al de omisión (22) usar:

```
$ scp -P 4455 file.txt user@HOST:/home/user/file.txt
```

En el caso de querer incrementar la velocidad de transferencia en el uso de *scp*, la opción más viable es el cambiar el cifrado usada por omisión por otras como *3des-cbc, aes128-cbc, aes192-cbc, aes256-cbc, aes-128-ctr, aes192-ctr, eas256-ctr, arcfour256, arcfour, blowfish-cbc y cast128-cbc* mediante:

```
$ scp -c blowfish user@server:/home/user/file .
```

o de forma alternativa usamos:

```
$ scp -c arcfour256 user@HOST:/home/user/file .
```

Si adicionalmente se quiere compactar para reducir el tiempo de transferencia, usamos:

```
$ scp -C SourceFile user@HOST:/home/user/TargetFile
```

Si se desea que no se muestre información de la transferencia de los archivos al usar *scp* usar:

$ scp -q SourceFile user@HOST:/home/user/TargetFile

o si desea ver más información en la transferencia usar:

$ scp -v SourceFile user@HOST:/home/user/TargetFile

Si se instala *sshpass*, entonces hacemos:

$ sshpass -p "your_password" scp -r backup_user@target_ip:/home/ /backup/$name

rsync es una aplicación libre y multiplataforma que ofrece transmisión eficiente de datos incrementales, que opera también con datos comprimidos y cifrados. Mediante una técnica de Delta Encoding, que permite sincronizar archivos y directorios entre dos máquinas de una red o entre dos ubicaciones en una misma máquina, minimizando el volumen de datos transferidos. Es ideal para trabaja en varias máquinas en las que se desea tener sincronizada una o más carpetas, para instalar usamos:

apt install rsync

La sintaxis básica es:

$ rsync [Opciones[62]] Origen [Origen]... Destino

por ejemplo, para hacer la sincronización de la carpeta *~/Datos* a *~/Respaldo* usamos:

$ rsync ~/Datos/ ~/Respaldo/

[62] Algunas opciones son: -r recursivo, -b backups, -R relativo, -u actualiza, -p muestra el progreso, -c comprime, -p preserva permisos, -v muestra lo que hace, -q trabaja en modo silencioso, -l preserva ligas simbólicas, -H preserva enlaces duros, -t preserva tiempos de modificación, etc.

Si queremos saber que hará el comando pero sin hacer la operación indicada, podemos usar la opción –dry-run, por ejemplo:

$ rsync –dry-run ~/Datos/ ~/Respaldo/

hay varias opciones que podemos usar en *rsync*, ejemplo de ellas es:

$ rsync –verbose –recursive –links –hard-links –times –delete –stats ~/Datos/ ~/Respaldo/

en este caso se sincronizaría el contenido de ~/Datos con *~/Respaldo*, pero lo hará mostrando lo que transfiere, de forma recursiva, conservará enlaces simbólicos y sus tiempos, borrará los archivos que estén en el destino pero no en el origen y al terminar mostrará las estadísticas de la transferencia.

Para hacer la transmisión cifrada entre equipos, podemos usar *rsync* conjuntamente con *ssh*, supongamos que se esta en la máquina y quiere tener sincronizada la carpeta ~/Datos con mas de un equipo, mediante *ssh* usando un puerto *343*, en la máquina *192.168.13.230* y usuario *antonio*, usar:

$ rsync –partial –recursive –links –hard-links –times –verbose –delete –stats ~/Datos/ -e 'ssh -p 343' antonio@192.168.13.230: ~/Respaldo/

por supuesto esto puede hacerse en cualquier dirección, i.e. de la máquina remota a nuestra máquina o viceversa, ejemplo:

$ rsync –partial –recursive –links –hard-links –times –verbose –delete –stats -e 'ssh -p 343' antonio@192.168.13.230: ~/Respaldo/ ~/Datos/

Los siguientes ejemplos permiten copiar los archivos (por ejemplo *.mp3) pero preservan la estructura de directorios que existía en su lugar de origen:

$ rsync -a -m –include '*/' –include '*.mp3' –exclude '*' ~/miDirectorio/ ~/OtrDirectorio

$ rsync -a –prune-empty-dirs –include '*/' –include '*.mp3' –exclude '*' ~/miDirectorio/ ~/OtrDirectorio

inicia la búsqueda en *~/midirectorio/* y los copia en *~/OtroDirectorio*.

Algunas veces necesitamos cambiar la prioridad de ejecución de un proceso *rsync* para evitar saturar el sistema, para ello podemos usar *ionice* que cambiara la prioridad de los comandos en ejecución, en este caso para todos los procesos de rsync se ejecutarán solo cuando la carga en el sistema sea ligera, mediante:

```
$ pgrep rsync | xargs ionice -c3 -p
```

Entre las opciones para excluir archivos o directorios en el uso de rsync, tenemos:

```
$ rsync -av –exclude 'archivo.txt' /fuente/ /destino/
$ rsync -av –exclude 'miDirectorio' /fuente/ /destino/
$ rsync -av –exclude '*.mp3' /fuente/ /destino/
$ rsync -av –exclude '*.mp3' –exclude '*.txt' /fuente/ /destino/
$ rsync -av –exclude={'*.mp3', '*.txt'} /fuente/ /destino/
```

si tenemos un *archivo.txt* con el contenido a excluir, podemos usar:

```
$ rsync -av –exclude-form={'archivo.txt'} /fuente/ /destino/
```

podemos excluir por el tamaño de los archivos, ejemplos:

```
$ rsync -av –max-size=500m /fuente/ /destino/
$ rsync -av –min-size=1m /fuente/ /destino/
```

pssh permite transferir o copiar archivos a múltiples servidores en Linux con un mismo comando:

- pscp - es una utilería para copiar archivos en paralelo a múltiples equipos
- prsync - es una utilería para transferir de forma eficiente archivos entre múltiples equipos en paralelo

- pnuke - permite concluir procesos en múltiples equipos en paralelo
- pslurp - permite copiar archivos de múltiples equipos a un equipo central en paralelo

Si creamos un archivo *Hosts.txt*, con los *IPs* como el siguiente:

```
192.168.0.3:22
192.168.0.9:22
```

podemos usar para copiar un archivo a múltiples servidores:

```
$ pscp -h Hosts.txt -l USR -Av wine-1.7.55.tar.bz2 /tmp/
```

o de forma alternativa usamos:

```
$ pscp.pssh -h Hosts.txt -l USR -Av wine-1.7.55.tar.bz2 /tmp/
```

donde:

-h indica que se lean los *IPs* del archivo indicado
-l se indica el usuario a usar en todos los equipos.
-A solicita el password para ser enviado a *ssh*
-v visualiza las operaciones y mensajes que genera el comando

Podemos copiar directorios a múltiples servidores, usando:

```
$ pscp -h myscpHosts.txt -l USR -Av -r Android\ Games/
/tmp/
```

o de forma alternativa:

```
$ pscp.pssh -h myscpHosts.txt -l USR -Av -r Android\ Games/
/tmp/
```

nc el comando *netcat* (*nc*) es una utilidad que permite escribir y leer datos a través de conexiones de red usando los protocolos TCP y UDP. Supongamos que estamos en el IP 192.168.13.230. Podemos generar un archivo compactado, cifrarlo y enviarlo a otro equipo en una mismo comando mediante:

```
$ tar -cvzf - directorio | gpg -c | nc -l 6666
```

y para recibirlo en el otro equipo usamos:

```
$ nc 192.168.13.230 6666 | gpg -d | tar -xvzf -
```

Siempre es mejor opción usar *scp*, pero *nc* cumplirá con su cometido.

7.5 Respaldo y Restauración

Una copia de seguridad (respaldo, copia de respaldo en inglés backup y data backup) es una copia de los datos originales que se realiza con el fin de disponer de un medio para recuperarlos en caso de su pérdida. Las copias de seguridad son útiles ante distintos eventos y usos: recuperar los sistemas informáticos y los datos de una catástrofe informática, natural o ataque; restaurar una pequeña cantidad de archivos que pueden haberse eliminado accidentalmente, corrompido, infectado por un virus informático u otras causas; guardar información histórica de forma más económica que los discos duros y además permitiendo el traslado a ubicaciones distintas de la de los datos originales; etc.

El proceso de copia de seguridad se complementa con otro conocido como restauración de los datos, que es la acción de leer y grabar en la ubicación original u otra alternativa los datos requeridos. La pérdida de datos es muy común en algún momento.

Ya que los sistemas de respaldo contienen por lo menos una copia de todos los datos que vale la pena salvar, se deben de tenerse en cuenta los requerimientos de almacenamiento. La organización del espacio de almacenamiento y la administración del proceso de efectuar la copia de seguridad son tareas a las que debemos dedicar tiempo y tener la certeza que están bien hechas. Para brindar una estructura de almacenamiento adecuada y segura existen muchos tipos diferentes de dispositivos físicos y en la nube para almacenar datos que son útiles para hacer copias de seguridad, cada uno con

sus ventajas y desventajas a tener en cuenta para elegirlos, como duplicidad, seguridad en los datos y facilidad de traslado.

Antes de que los datos sean enviados a su lugar de almacenamiento se lo debe seleccionar, extraer y manipular. Se han desarrollado muchas técnicas diferentes para optimizar el procedimiento de efectuar los respaldos. Estos procedimientos incluyen entre otras optimizaciones para trabajar con archivos abiertos y fuentes de datos en uso y también incluyen procesos de compresión, cifrado, y procesos de deduplicación, entendiéndose por esto último a una forma específica de compresión donde los datos superfluos son eliminados.

Tipos de Respaldos Existen distintos tipos de respaldo, entre los que destacan:

- Respaldo Completo: Se copian todos los archivos que pueda haber en una trayectoria.
- Respaldo Diferencial: Es una copia intermedia entre la completa y la incremental. Es decir, copiara tanto los archivos que se han creado nuevos como los que se han modificado.
- Respaldo Incremental: Solo copiara los archivos que hayan sido modificados tras haber hecho una copia de seguridad previa de tipo completo o diferencial. Para ello compara las fechas de modificación de los archivos de la fuente y los de la copia previa y si hay diferencias el Software tomara la decisión de copiar solo aquellos que se hayan modificado. Lo bueno de esta copia es que no es tan pesada como la completa y permite actualizar solo lo que te interesa.
- Respaldo Espejo: Este modo de respaldo es similar al respaldo completo, con la salvedad de que los archivos no se pueden comprimir para tratar de garantizar su restauración, por lo que además de ser menos segura también ocupa más espacio de almacenamiento.

Software de Copias de Seguridad Existe una gran gama de programas en el mercado para realizar copias de seguridad, es importante definir previamente los requerimientos específicos de respaldo y restauración para determinar el Software adecuado a nuestras necesidades. Existe una gran

cantidad de programas adaptados a cada necesidad, por ejemplo para respaldar toda la máquina podemos usar:

- Clonezilla
- Mondo Rescue
- Partimage
- Ping
- Redo Backup

Para hacer respaldos desde el ambiente gráfico:

- Bacula
- Amanda
- Backupninja
- Areca Backup
- BackupPC
- Keep
- UrBackup
- Back in Time
- Timeshift
- Simple Backup Solution
- KBackup
- Kup Backup System
- Grsync

Para hacer respaldos desde la línea de comandos:

- tar (véase 7.3)

- scp (véase 7.4)
- rsync (véase 7.4)
- rsnapshot
- bup
- restic
- rdiff-backup
- duplicity
- rclone

Para el adecuado respaldo de ficheros con datos de carácter personal es común que las copias de seguridad de dichos datos se almacenen cifrados y en una ubicación diferente al lugar de origen.

La copia de seguridad es el mejor método de protección de datos de importancia, pero siempre existe la posibilidad de que la copia de datos no haya funcionado correctamente y en caso de necesidad de restauración de los datos no podamos realizarlo ya que la información de la copia de seguridad puede encontrarse corrupta por diversos motivos:

- el medio en el que se realizaba la copia se encuentra dañado.
- los automatismos de copia no se han ejecutado correctamente.
- y otros muchos motivos que pueden causar que nuestras copias de seguridad sean incorrectas, y por lo tanto inútiles.

Para evitar este problema es muy importante que nos cercioremos de que hacemos las copias correctamente y comprobemos que somos capaces de restaurar la copia de seguridad a su ubicación original, comprobando así que la copia sea correcta y que somos capaces de restaurarla y conocemos el método de restauración, ya que en caso de necesidad crítica los nervios afloran y nos pueden echar por tierra nuestra labor de copia al realizar algún paso erróneo a la hora de restaurar los datos.

Siempre que podamos debemos cifrar los respaldos, esto los mantiene más seguros. Si alguien llegara a tener acceso a datos de respaldo sin el debido cifrado, este se podría restaurar y con ello podría utilizar toda la información del mismo.

rdiff-backup es capaz de realizar una copia casi exacta del directorio origen (sin cifrar), preservando toda la información de los recursos: permisos, usuarios y grupos, fecha de modificación, enlaces simbólicos, ficheros fifos, etc., manteniéndola independientemente de la plataforma y sistema de ficheros donde se almacene, gracias a que toda esa información se guarda en ficheros independientes de metadatos. Además seremos capaces de recuperar una instantánea exacta de un determinado día, restaurando el estado que en ese momento tenía nuestro directorio (fichero eliminados, modificados, etc.).

Otra característica importante es la eficiencia del espacio usado. Los respaldos incrementales que utilizan herramientas como *backup-manager*, realizan una copia completa de los recursos modificados, en cambio, *rdiff-backup* sólo almacena las fracciones de datos que realmente han cambiado.

El respaldo de recursos remotos es otro de sus puntos fuertes y que considero importante destacar. La transferencia de recursos remotos esta optimizada; sólo se transfiere por la red la información que realmente ha cambiado, haciendo un uso eficiente del ancho de banda. El único requisito es que en la máquina remota también se encuentre instalado *rdiff-backup*. Para su instalación hacemos:

```
# apt install rdiff-backup
```

Realizar copias de seguridad con *rdiff-backup* es realmente sencillo, es como si estuviéramos usando el comando *cp* de Linux. Únicamente tendremos que indicar el directorio origen (del que se quiere realizar el respaldo) y el directorio destino (donde se almacenará el respaldo).

```
$ rdiff-backup dir_ori dir_dest
```

Podemos encontrarnos varios escenarios posibles dependiendo de donde se encuentren situados los directorios origen y destino.

- directorio origen local y destino local
- directorio origen remoto y destino local
- directorio origen local y destino remoto

Independientemente de cada caso su uso es idéntico, sólo cambia la forma de acceder a los directorios remotos.

Por ejemplo, el comando que deberíamos lanzar para realizar un respaldo de nuestro directorio local /home/autentia al directorio destino /mnt/backup situado en la misma máquina sería:

```
$ rdiff-backup /home/autentia /mnt/backup
```

Si alguno de los dos directorios estuviese en una máquina remota deberíamos hacer preceder al directorio del usuario y el nombre de la máquina. Por ejemplo, vamos a imaginar que queremos realizar el respaldo del directorio /etc de la maquina mac1 (utilizando el usuario backup para acceder a la máquina) al directorio /mnt/backup/mac1 de nuestra máquina local. El comando que deberíamos ejecutar sería:

```
$ rdiff-backup backup@mac1::/etc /mnt/backup/mac1
```

Obviamente, para que este comando funcione en modo Script, deberemos tener configurada correctamente nuestra máquina para acceder al otro equipo sin necesidad de contraseña.

Para restaura el directorio home de la copia realizada el 10 de Enero de 2021 en /tmp/home.

```
$ rdiff-backup -r 2021-01-10 /mnt/backup/home /tmp/home
```

Otro ejemplo, restauramos la copia actual situada en la máquina remota mac1 en el directorio /temp.

```
$ rdiff-backup -r now backup@mac1::/mnt/backup /temp
```

podremos obtener el listado con los cambios incrementales de un fichero:

```
$ rdiff-backup -l /mnt/backup/file
```

Con -list-changed-since podremos saber cuántos ficheros han cambiado desde una marca de tiempo en concreto. Por ejemplo con el siguiente comando estamos viendo que ficheros han cambiado en los últimos 10 días.

```
$ rdiff-backup -list-changed-since 10D /mnt/backup/etc
```

-list-at-time lista todos los ficheros que estuvieron presentes en un determinado momento. Esto incluye tanto fichero eliminados como aquellos que no han sido modificados.

```
$ rdiff-backup -list-at-time 5D /mnt/backup/home
```

Y por último podremos comparar los cambios producidos entre el resplado y un directorio en concreto. rdiff-backup proporciona dos opciones -compare y -compare-at-time; este último para poder comparar con una marca de tiempo determinada. Por ejemplo:

```
$ rdiff-backup -compare /home /mnt/backup/home
$ rdiff-backup -compare-at-time 2W /home /mnt/backup/home
```

duplicity es una herramienta avanzada de copias de seguridad cifradas, disponible para la línea de comandos. Está escrita en *Python*, usando herramientas como *librsync* y *GnuPG*.

Los archivos obtenidos se encuentran en formato *tar* y, si lo creemos oportuno, pueden estar o no cifrados. Además, permite realizar copias incrementales que nos permite ahorrar espacio. Para su instalación hacemos:

```
# apt install duplicity
```

Para hacer una copia de seguridad de los datos que queramos, solo hay que utilizar el nombre de la herramienta, seguido de la ruta de los datos que queremos copiar y, a continuación, del destino donde queremos almacenarla. Es decir, algo como esto:

```
$ duplicity ~/documentos file:///backup
```

la recuperación usando esa carpeta como destino. Para lograrlo, escribimos algo como esto:

```
$ duplicity file:///backup recupera
```

Como cabía esperar, la herramienta nos pide la frase de paso para cifrar antes de iniciar la recuperación.

A continuación, vamos a repetir la copia anterior, pero evitando que se copie el directorio borradores y todo su contenido. Lo logramos añadiendo la opción -exclude, seguida de la ruta a excluir

```
$ duplicity ~/documentos file:///backup –exclude ~/documentos/borradores
```

Podemos usar duplicity en un Scrip, por ejemplo:

```
export PASSPHRASE='ClaveCifrado'
export FTP_PASSWORD='ClaveServidor'
## Respalda localmente
duplicity /home/antonio/trabajo file:/home/antonio/Respaldos
## Verifica
duplicity verify file:/home/antonio/Respaldos /home/antonio/trabajo
## Respalda en el servidor
duplicity /home/antonio/trabajo scp://antonio@\
192.168.13.230//home/antonio/Respaldos
unset PASSPHRASE
unset FTP_PASSWORD
```

rclone es una aplicación libre para sistemas de tipo Linux, Unix y Microsoft Windows, se trata de una herramienta en línea de comando para sincronizar archivos y directorios desde la computadora con los proveedores más importantes de alojamiento de contenidos en la nube[63]. También permite efectuar copias dentro de nuestro propio sistema de archivos. Está escrito en lenguaje de programación *Go* y se basa en la utilidad *rsync*. Para instalarlo hacemos:

```
# apt install rclone
```

Para crear, editar o eliminar un dispositivo remoto, es necesario utilizar la siguiente orden

```
$ rclone config
```

Se entra a una sesión interactiva, en la que configuraremos a *rclone* con los datos de nuestra cuenta en la nube, especificando el nombre del recurso mediante el cual podremos identificar nuestro servicio de la nube (por ejemplo antonio_google) en nuestra máquina.

Por ejemplo, podemos conocer el monto de espacio disponible en el servidor, usando:

```
$ rclone about antonio_google:
```

[63] Algunos de los servicios en la nube más importantes soportados por Rclone son: Amazon Drive, Amazon S3, Box, Dropbox, Google Cloud Storage, Google Drive, Hubic, Mega, Microsoft OneDrive, Nextcloud, OpenDrive, Oracle Cloud Storage, ownClowd, pCloud, QingCloud Object Storage, Webdav, Yandex Disk.

listamos el contenido de carpetas y contenedores del servidor en la nube:

```
$ rclone lsd antonio_google:
```

podemos crear un directorio para hacer la sincronización:

```
$ rclone mkdir antonio_google:Respaldos
```

y realizamos la sincronización del contenido de nuestro equipo con el de la nube, usamos:

```
$ rclone sync /home/antonio/trabajo/ antonio_google:Respaldos
-P
```

si necesitamos sincronizar el contenido de la nube con nuestro equipo, usamos:

```
$ rclone sync antonio_google:Respaldos /home/antonio/trabajo/
-P
```

Podemos sincronizar el contenido de una unidad o directorio, por ejemplo en el directorio *nube*, usando:

```
$ rclone mount antonio_google:Respaldos ./nube
```

este permanecerá montado hasta que usemos Ctrl-C para terminar el comando rclone, en caso de ser necesario desmontar el punto de montaje manualmente, usamos:

```
$ fusermount -u ./nube
```

Algunas otras opciones de trabajo son:

- rclone config - crear, editar o eliminar un dispositivo remoto.
- rclone copy - Copiar archivos del origen al destino.
- rclone sync - Sincronizar archivos, modificando el destino únicamente.
- rclone move - Copiar archivos del origen al destino.

- rclone delete - Borrar archivos.
- rclone purge - Borrar todo el contenido de la carpeta.
- rclone mkdir - Crear una carpeta si no existe.
- rclone rmdir - Borrar una carpeta.
- rclone rmdirs - Borrar carpetas vacías.
- rclone check - Comprueba si los archivos en el origen y destino coinciden.
- rclone ls - Lista todos los archivos.
- rclone lsd - Lista todas las carpetas y contenedores.
- rclone size - Devuelve el tamaño y el número de objetos.
- rclone version - Muestra el número de versión.
- rclone authorize - Autorización remota.
- rclone cat - Concatena archivos y los envía a la salida estándar.
- rclone copyto - Copiar archivos del origen al destino, exceptuando los ya copiados.
- rclone listremotes - Lista todos los dispositivos remotos.
- rclone moveto - Mueve archivos o carpetas desde el origen al destino.
- rclone about - Devuelve información sobre el dispositivo remoto.

7.6 Control de Versiones

El control de versiones es una herramienta que permite registrar y administrar los cambios en el código fuente o en documentos. También se le conoce como control de código fuente, control de revisiones o gestión de código fuente.

El control de versiones es útil para:

- Trabajar de forma colaborativa entre desarrolladores

- Conservar la integridad del código
- Reducir el tiempo de desarrollo
- Aumentar el número de implementaciones exitosas
- Proteger el código fuente
- Resolver problemas como cambios incompatibles entre desarrolladores
- Resolver problemas como nuevos errores introducidos por los cambios

El control de versiones funciona tomando una instantánea de los archivos cada vez que un desarrollador los edita. Estas instantáneas se guardan de forma permanente para que se puedan recuperar más adelante si es necesario.

El control de versiones también se puede utilizar para gestionar los cambios en documentos. En este caso, el control de versiones organiza, etiqueta y conserva cada borrador hasta que se completa una versión final.

¿Qué es control de versiones? se define como control de versiones a la gestión de los diversos cambios (Commit es un conjunto de cambios guardados en el repositorio de Git y tiene un identificador SHA1 único) que se realizan sobre los elementos de algún producto o una configuración del mismo, es decir, es lo que se hace al momento de estar desarrollando un Software o una página Web. Exactamente es eso que haces cuando subes y actualizas tu código en la nube, o le añades alguna parte o simplemente editas cosas que no funcionan como deberían o al menos no como tú esperarías.

¿A que le llamamos sistema de control de versiones? son todas las herramientas que nos permiten hacer todas esas modificaciones antes mencionadas en nuestro código y hacen que sea más fácil la administración de las distintas versiones de cada producto desarrollado; es decir *Git*.

Antes de seguir avanzando, conviene definir algunos términos comunes que encontraremos más adelante:

- Un repositorio es una carpeta que contiene los archivos y subdirectorios de un proyecto. Puede ser público o privado, dependiendo de quién deba tener acceso.

- Una rama es una ruta de desarrollo separada en el mismo repositorio. En un mismo proyecto suelen utilizarse ramas separadas para trabajar en nuevas características sin interferir con la versión de producción. Una vez que el código es revisado y probado, un administrador del repositorio puede fusionar los cambios en la rama principal.

- Un commit es una instantánea de un repositorio en un momento dado. Permite a los programadores incluir comentarios y pedir la opinión de otras personas. Usando el Hash que lo identifica puedes volver a un estado anterior del proyecto si es necesario. Antes de que los archivos y directorios puedan ser comiteados, necesitamos decirle a Git que los rastree. Normalmente nos referimos a este paso simplemente como agregar los archivos al área de Staging.

- Un Pull Request (o simplemente PR) es un método para informar a otros desarrolladores y discutir los cambios recientes antes de incorporarlos a la ruta de desarrollo principal.

- Un Fork es un proyecto independiente que se basa en un repositorio determinado. A diferencia de las ramas, no es local a este último. Sin embargo, también puede fusionarse con él a través de un Pull Request adecuado.

- Se puede utilizar un archivo .gitignore para indicar qué contenido local no queremos incluir en el repositorio. Esto nos ayuda a evitar enviar archivos temporales o exponer información sensible en una solución basada en la Web.

7.6.1 Git

Git es un programa de control de versiones que sirve para la gestión de los diversos cambios que se realizan sobre los elementos de algún proyecto de Software y sus respectivos programas fuente o configuración del mismo guardando instantáneas (Snapshots) del código en un estado determinado, que viene dado por el autor y la fecha. Fue diseñado por Linus Torvalds y es usado para controlar los cambios de diversos proyectos como los fuentes del Kernel de Linux que tiene decenas de millones de líneas de código (en la versión 4.12 cuenta con 24,170,860 líneas de código repartidos en 59,806 archivos) y es trabajado por miles de programadores alrededor del mundo.

Git fue creado pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente, es decir *Git* nos proporciona las herramientas para desarrollar un trabajo en equipo de manera inteligente y rápida y por trabajo nos referimos a algún Software o página que implique código el cual necesitemos hacerlo con un grupo de personas.

Algunas de las características más importantes de *Git* son:

- Rapidez en la gestión de ramas (Branche es como una línea de tiempo a partir de los Commit, siempre hay siempre como mínimo una rama principal predefinida llamada Master), debido a que *Git* nos dice que un cambio será fusionado mucho más frecuentemente de lo que se escribe originalmente.
- Gestión distribuida: Los cambios se importan como ramas adicionales y pueden ser fusionados de la misma manera como se hace en la rama local.
- Gestión eficiente de proyectos grandes.
- Realmacenamiento periódico en paquetes.

Con esto en mente, vamos a hacer una introducción a Git desde cero.

Instalación de Git para instalar *Git* completo en el servidor o en la máquina de trabajo, usamos:

```
# apt install git-all
```

Para instalar lo básico de *Git*, si no está instalado:

```
# apt install git
```

otras opciones para trabajar con *Git* son:

```
# apt install git git-all gitk gitg git-cola git-gui qgit tig lighttpd
vim-fugitive
```

También existen otros proyectos parecidos a Git, algunos de ellos son:

```
# apt install mercurial
# apt install subversion rapidsvn
# apt install cvs
```

Configuración de Git si se quiere especificar la identidad del que controla el repositorio local en el equipo, debemos usar (por omisión toma la información de la cuenta del usuario y máquina):

```
$ git config –global user.name "Antonio Carrillo"
$ git config –global user.email antoniocarrillo@ciencias.unam.mx
```

si se desea configurar el editor de texto a usar por *Git*, usamos (por omisión es *vim*):

```
$ git config –global Core.editor scite
```

si se desea configurar la herramienta de control de diferencias, usamos (por omisión *vimdiff*):

```
$ git config –global merge.tool meld
```

para comprobar podemos usar:

```
git config –global -list
```

el cual creará un archivo de texto llamado .gitconfig, y podemos ver su contenido, usando:

```
cat ~/.gitconfig
```

Clonar un repositorio Git podemos iniciar clonando algún repositorio remoto de la red, para ello debemos conocer su dirección y usar:

```
$ git clone <dirección>
```

esto permitirá bajar todo el contenido del proyecto a clonar.

Crear un repositorio Git local si lo que requiero es un control personal sin necesidad de compartir los archivos con ningún otro usuario, puedo usar *Git* de forma local en cualquier directorio mediante:

```
$ git init
```

Si se desea agregar la identidad del que controla el repositorio en este directorio, se debe usar:

```
$ git config user.name "Antonio Carrillo"
$ git config user.email antoniocarrillo@ciencias.unam.mx
```

Ahora para agregar los archivos (todos los de este directorio), usar:

```
$ git add .
```

así podemos hacer la confirmación de los cambios, mediante:

```
$ git commit -m "Primer lanzamiento"
```

ahora cada que lo requiera al hacer modificaciones, puedo checar los cambios:

```
$ git status
```

o en forma gráfica con *gitk*, mediante:

```
$ gitk
```

para actualizar los cambios, usar:

```
$ git commit -a -m 'Actualizacion'
```

en caso necesario, podemos mover uno o más archivos de una trayectoria a otra, usando:

```
$ git mv archivo(s) /trayectoria
```

o podemos borrar uno o más archivos, usando:

```
$ git rm archivo(s)
```

además podemos editar el archivo *.gitignore*, para indicar los archivos a ignorar usando líneas independientes para cada tipo.

Podemos ver el histórico de las operaciones que hemos hecho, usando:

```
$ git log
```

y podemos comprobar el estado del repositorio, usando:

```
$ git status
```

La otra alternativa es preparar un directorio para el repositorio ya sea en el servidor o de forma local, mediante:

```
$ mkdir example.git
$ cd example.git
```

Para inicializar el repositorio:

```
$ git –bare init
```

Es buena opción limitar el acceso a la cuenta vía *ssh*, por ello es mejor cambiar en */etc/passwd*, la línea del usuario predeterminada:

```
tlahuiz:x:1005:1005:Tlahuizcalpan„„:/home/tlahuiz:/bin/bash
```

a esta otra:

```
tlahuiz:x:1005:1005:Tlahuizcalpan„„:/home/tlahuiz:/usr/bin/git-
Shell
```

En la máquina de trabajo o en el servidor en cualquier carpeta se genera la estructura del repositorio en un directorio temporal de trabajo para el repositorio:

```
$ mkdir tmp
$ cd tmp
$ git init
```

Para generar la estructura de trabajo para el repositorio y los archivos necesarios:

```
$ mkdir branches release trunk
$ mkdir ...
```

Para adicionar todos y cada uno de los archivos y carpetas:

```
$ git add .
```

Para subir los cambios:

```
$ git commit -m "Texto"
```

Después debemos mandarlo al servidor:

```
$ git remote add origin ssh://usr@máquina/~/trayectoria
```

o mandarlo a un directorio local:

```
$ git remote add origin ~/trayectoria
$ git push origin +master:refs/heads/master
```

Para usar el repositorio en cualquier otra máquina hay que bajar el repositorio por primera vez del servidor:

```
$ git clone ssh://usr@máquina/~/trayectoria
```

o de una carpeta local:

```
$ git clone ~/trayectoria
```

Ahora, podemos configurar algunos datos usados en el control de cambios:

```
$ git config –global usr.name "Antonio Carrillo"
$ git config –global usr.email antoniocarrillo@ciencias.unam.mx
```

cuando se requiera actualizar del repositorio los cambios:

```
$ git pull
```

para subir los cambios al repositorio:

```
$ git commit -a -m "mensaje"
$ git push
```

Comando usados para el trabajo cotidiano en *Git* para ver el estado de los archivos locales, usamos:

```
$ git status
```

para generar una nueva rama y trabajar en ella:

```
$ git branch MiIdea
$ git checkout MiIdea
```

o en un solo paso:

```
$ git checkout -b MiIdea
```

Para unificar las ramas generadas en el punto anterior:

```
$ git checkout master
$ git merge MiIdea
```

Para borrar una rama:

```
$ git branch -d MiIdea
```

Para listar ramas:

```
$ git branch
```

Para listar ramas fusionadas:

```
$ git branch –merged
```

Para listar ramas sin fusionar:

```
$ branch –no-merged
```

Para ver los cambios en el repositorio:

```
$ git log
```

o verlos en forma acortada:

$ git log –pretty=oneline

Para recuperar un archivo de una actualización anterior:

$ git show a30ab2ca64d81876c939e16e9dac57c8db6fb103:ruta/al/archivo > ruta/al/archivo.bak

Para volver a una versión anterior:

$ git reset –hard 56f8fb550282f8dfaa75cd204d22413fa6081a11:

para regresar a la versión presente (cuidado con subir cambios en ramas anteriores):

$ git pull

Si en algún momento borramos algo o realizamos cambios en nuestra máquina y necesitamos regresar los archivos como estaban en nuestra última actualización, podemos usar:

$ git reset –hard HEAD

este trabaja con la información de nuestra copia local y no necesita conexión de red para la restitución. Eventualmente es necesario optimizar la copia local de los archivos en *Git*, para ello podemos usar:

$ git gc

Visualizador gráfico para *Git*:

apt install gitk

Git es un proyecto pujante, amplio y bien documentado, ejemplos y documentación puede ser consultada en:

- https://git-scm.com/book/es/v1
- http://git-scm.com/documentation
- https://coderwall.com/p/kucyaw/protect-secret-data-in-git-repo

Git en Google Drive:

- http://www.iexplain.org/using-git-with-google-drive-a-tutorial/
- https://techstreams.github.io/2016/09/07/google-drive-as-simple-git-Host/

Un Resumen de los Comando de Git más Usados

`CONFIGURACION`

Configurar Nombre que salen en los commits

git config –global user.name "antonio"

Configurar Email

git config –global user.email antonio@gmail.com

Marco de colores para los comando

git config –global color.ui true

`INICIANDO REPOSITORIO`

Iniciamos GIT en la carpeta donde está el proyecto

git init

Clonamos el repositorio de github o bitbucket

git clone <url>

Añadimos todos los archivos para el commit

git add .

Hacemos el primer commit

git commit -m "Texto que identifique por que se hizo el commit"

Subimos al repositorio

git push origin master

`GIT CLONE`

Clonamos el repositorio de github, gitlab o bitbucket

git clone <url>

Clonamos el repositorio de github, gitlab o bitbucket

git clone <url> git-demo

`GIT ADD`

Añadimos todos los archivos para el commit

git add .

Añadimos el archivo para el commit

git add <archivo>

Añadimos todos los archivos para el commit omitiendo los nuevos

git add –all

Añadimos todos los archivos con la extensión especificada

git add *.txt

Añadimos todos los archivos dentro de un directorio y de una extensión especifica

git add docs/*.txt

Añadimos todos los archivos dentro de un directorios

git add docs/

`GIT COMMIT`

Cargar en el HEAD los cambios realizados

git commit -m "Texto que identifique por que se hizo el commit"

Agregar y Cargar en el HEAD los cambios realizados

git commit -a -m "Texto que identifique por que se hizo el commit"

De haber conflictos los muestra

git commit -a

Agregar al último commit, este no se muestra como un nuevo commit en los logs. Se puede especificar un nuevo mensaje

git commit –amend -m "Texto que identifique por que se hizo el commit"

`GIT PUSH`

Subimos al repositorio

git push <origien> <branch>

Subimos un tag

git push –tags

`GIT LOG`

Muestra los logs de los commits

git log

Muestras los cambios en los commits

git log –oneline –stat

Muestra gráficos de los commits

git log –oneline –graph

`GIT DIFF`

Muestra los cambios realizados a un archivo

```
git diff
git diff –staged
```

`GIT HEAD`

Saca un archivo del commit

```
git reset HEAD <archivo>
```

Devuelve el último commit que se hizo y pone los cambios en staging

```
git reset –soft HEAD^
```

Devuelve el último commit y todos los cambios

```
git reset –hard HEAD^
```

Devuelve los 2 últimos commit y todos los cambios

```
git reset –hard HEAD^^
```

Rollback merge/commit

```
git log
git reset –hard <commit_sha>
```

`GIT REMOTE`

Agregar repositorio remoto

```
git remote add origin <url>
```

Cambiar de remote

```
git remote set-url origin <url>
```

Remover repositorio

```
git remote rm <name/origin>
```

Muestra lista repositorios

git remote -v

Muestra los branches remotos

git remote show origin

Limpiar todos los branches eliminados

git remote prune origin

`GIT BRANCH`

Crea un branch

git branch <nameBranch>

Lista los branches

git branch

Comando -d elimina el branch y lo une al master

git branch -d <nameBranch>

Elimina sin preguntar

git branch -D <nameBranch>

`GIT TAG`

Muestra una lista de todos los tags

git tag

Crea un nuevo tags

git tag -a <verison> - m "esta es la versión x"

`GIT REBASE`

Los rebase se usan cuando trabajamos con branches esto hace que los branches se pongan al día con el master sin afectar al mismo

Une el branch actual con el mastar, esto no se puede ver como un merge

git rebase

Cuando se produce un conflicto no das las siguientes opciones:

cuando resolvemos los conflictos –*continue* continua la secuencia del rebase donde se pauso

git rebase –continue

Omite el conflicto y sigue su camino

git rebase –skip

Devuelve todo al principio del rebase

git reabse –abort

Para hacer un rebase a un branch en específico

git rebase <nameBranch>

`OTROS COMANDOS`

Lista un estado actual del repositorio con lista de archivos modificados o agregados

git status

Quita del HEAD un archivo y le pone el estado de no trabajado

git checkout – <file>

Crea un branch en base a uno Online

git checkout -b newlocalbranchname origin/branch-name

Busca los cambios nuevos y actualiza el repositorio

git pull origin <nameBranch>

Cambiar de branch

git checkout <nameBranch/tagname>

Une el branch actual con el especificado

git merge <nameBranch>

Verifica cambios en el repositorio Online con el local

git fetch

Borrar un archivo del repositorio

git rm <archivo>

`Fork`

Descargar remote de un fork

git remote add upstream <url>

Merge con master de un fork

git fetch upstream
git merge upstream/master

Git-crypt El paquete *git-crypt* es una solución que usa *GPG* por debajo de *Git* que permite cifrado y descifrado transparente de archivos en un repositorio *git*.

Los archivos que se requieran proteger serán cifrados al hacer commit y descifrados al hacer *checkout* y permite compartir libremente un repo-sitorio que contenga contenido tanto público como privado. De esta forma, permite trabajar de manera transparente con el contenido descifrado, de forma que desarrolladores que no tengan la clave secreta podrán clonar y hacer commit en un repositorio con archivos cifrados.

Esto te permite almacenar tu material secreto (como pueden ser claves) en el mismo repositorio que tu código sin tener que bloquearlo. Solo un usuario autorizado puede dar permisos a otros usuarios.

Para instalar el paquete git-crypt usamos:

```
# apt install git-crypt
```

Ya instalado debemos preparar el repositorio git, para crear la llave, entonces usar:

```
$ git-crypt keygen ~/crypt-key
```

Ahora podemos crear el repositorio:

```
$ cd repo
$ git-crypt init
```

Especifica que carpetas/archivos deben ser cifrados, como git-filters:

```
$ cat .gitattributes
```

keys filter=git-crypt diff=git-crypt

crear la lista de los archivos a cifrar

```
$ vi .gitattributes
```

Indicamos que se cifren, por ejemplo, los archivos *.html, .org*, directorio:secretdir/**secreto y archivo, con cualquier extensión o palabra que le preceda.

```
*.html filter=git-crypt diff=git-crypt‘
*.org filter=git-crypt diff=git-crypt
directorio_secreto/** filter=git-crypt diff=git-crypt
*archivo* filter=git-crypt diff=git-crypt
```

ahora cada vez que hagamos un commit, los archivos *.html y .org*, subirán cifrados.

Ya podemos usar la llave para cifrar los archivos indicados por .gitattributes mediante:

```
$ git-crypt unlock ~/crypt-key
```

y agregar los archivos que deseamos cifrar, usando *git add*, revisando el estado de los archivos cifrados mediante:

```
$ git-crypt status -f
```

y podemos hacer los commits necesarios.

Al clonar el repositorio, los archivos cifrados se mostrarán como tal, hasta hacer en el repositorio:

```
$ git-crypt unlock ~/crypt-key
```

mostrando los archivos descifrados a partir de ese momento

Si se desea respaldar el repositorio en un solo archivo se puede usar:

```
$ git bundle create /tmp/Respaldo –all
```

y para restaurar usar algo como:

```
$ git clone /tmp/Respaldo newFolder
```

También podemos añadir usuarios autorizados (identificados por su clave *GPG*), mediante:

```
$ git-crypt add-gpg-user USER_ID
```

Flujos de trabajo comunes

- En la máquina del desarrollador: Crea el vault, añádete como usuario fiable. Pide las claves públicas a los miembros de tu equipo y añádelas al vault.
- En el entorno de Integración Continua (CI): Añade una clave *GPG* común para los ejecutores jenkins/CI. Autorízarla en el repositorio.

Seguridad

- Git-crypt usa *GPG* internamente, así que el nivel de seguridad debería ser el dado por *GPG*, a excepción de posibles errores en el propio programa *git-crypt*.

- Git-crypt es más seguro que otros sistemas git de cifrado transparente, *git-crypt* cifra archivos usando *AES-256* en modo *CTR* con un synthetic IV derivado del *SHA-1 HMAC* del archivo. Este modo de operar proporciona seguridad semántica ante *CPAs* (chosen-plain attacks) determinísticos. Esto significa que pese a que el cifrado es determinística (lo cual es requerido para que git pueda distinguir cuando un archivo ha cambiado y cuando no), no filtra información más allá de mostrar si dos archivos son idénticos o no.

Limitaciones y Trucos

- Cualquier usuario no autorizado puede ver que estamos usando *git-crypt* basándose en la evidencia dejada en el archivo .gitattributes.

- Git-crypt no cifra nombres de archivo, mensajes de *commit, symlink targets, gitlinks*, u otros metadatos.

- Git-crypt se apoya en git filters, los cuales no fueron diseñados con el cifrado en mente. Así pues, *git-crypt* no es la mejor herramienta para cifrar la mayoría o totalidad de los archivos de un repositorio. Donde *git-crypt* destaca es en aquellos casos en que la mayoría del repositorio es público pero unos pocos archivos deben ser cifrados (por ejemplo, claves privadas o archivos con credenciales API). Para cifrar un repositorio entero, es mejor considerar usar un sistema como **git-remote-gcrypt**.

- Git-crypt no esconde cuando un archivo cambia o no, cuanto ocupa o el hecho de que dos archivos sean idénticos.

- Los archivos cifrados con *git-crypt* no se pueden comprimir. Incluso el más pequeño de los cambios en un archivo cifrado requiere que git archive el archivo modificado en su totalidad y no solo un delta.

- A pesar de que *git-crypt* protege el contenido de los archivos individuales con *SHA-1 HMAC*, *git-crypt* no puede ser usado de forma segura a menos que el repositorio entero esté protegido contra la alteración de

datos (un atacante que pueda mutar tu repositorio podrá alterar tu archivo *.gitattributes* para deshabilitar el cifrado). Si fuera necesario, usa características de *git* cómo signed tags en vez de contar únicamente con *git-crypt* para la integridad.

- El *diff* del *commit* varía cuando el vault está abierto vs cuando está cerrado. Cuando está abierto, los contenidos del archivo están en formato plano, es decir, descifrados. En consecuencia puedes ver el *diff*. Cuando el vault está cerrado, no se puede apreciar un *diff* efectivo ya que el texto cifrado cambia, pero el ojo humano no puede distinguir los contenidos.

Además de Git usado de forma local, existen diversos servicios en la nube[64] que permiten dar soporte a proyectos mediante *Git,* en los cuales es necesario crear una cuenta y subir los datos usando *Git,* algunos de estos servicios son:

7.6.2 GitLab vs GitHub

Aunque GitHub y GitLab tienen similitudes, incluso en el propio nombre que comienza por Git debido a que ambas se basa en la famosa herramienta de control de versiones escrita por Linus Torvalds, pero ni una ni otra son exactamente iguales. Por ello, el ganador de la batalla GitHub vs GitLab no está tan claro, tienen algunas diferencias que hacen que tengan sus ventajas y desventajas para los usuarios y desarrolladores que las suelen usar.

¿Qué es GitHub? es una plataforma de desarrollo colaborativo, también llamado forja. Es decir, una plataforma enfocada hacia la cooperación entre desarrolladores para la difusión y soporte de su Software (aunque poco a poco se ha ido usando para otros proyectos más allá del Software).

Como su propio nombre indica, se apoya sobre el sistema de control de versiones *Git*. Así, se puede operar sobre el código fuente de los programas y llevar un desarrollo ordenado. Además, esta plataforma está escrita en Ruby on Rails.

[64] Algunos de estos proyectos gratuitos son: Gitlab, Github, Bitbucket, Beanstalk, Launchpad, SourceForge, Phabricator, GitBucket, Gogs, Gitea, Apache Allura, entre otros.

Tiene una enorme cantidad de proyectos de código abierto almacenada en su plataforma y accesibles de forma pública. Tal es su valor que Microsoft optó por comprar esta plataforma en 2018, aportando una cifra de nada menos que de 7,500 millones de dólares.

Pese a las dudas sobre esa compra, la plataforma continuó operando como de costumbre, y continúa siendo una de las más usadas. En ella se alojan proyectos tan importantes como el propio Kernel Linux.

¿Qué es GitLab? es otra alternativa a GitHub, otro sitio de forja con un servicio Web y un sistema de control de versiones también basado en Git. Por supuesto, se ideó para el alojamiento de proyectos de código abierto y para facilitar la vida de los desarrolladores, pero existen algunas diferencias con el anterior.

Esta Web, además de la gestión de repositorios y control de versiones, también ofrece alojamiento para Wikis, y sistema de seguimiento de errores. Una completa suite para crear y gestionar los proyectos de todo tipo, ya que, al igual que GitHub, actualmente se alojan proyectos que van más allá del código fuente.

Fue escrito por unos desarrolladores ucranianos, Dmitry Zaporozhets y Valery Sizov, usando lenguaje de programación Ruby y algunas partes en Go. Después se mejoró su arquitectura con Go, Vue.js, y Ruby on Rails, como en el caso de GitHub.

A pesar de ser muy conocida y ser la gran alternativa a GitHub, no cuenta con tantos proyectos. Eso no quiere decir que la cantidad de código alojado sea muy grande, con organizaciones que confían en ella de la talla del CERN, la NASA, IBM, Sony, etc.

Personalmente, te diría que no existe un claro ganador en la batalla GitHub vs GitLab. No es tan sencillo elegir una plataforma que sea infinitamente superior a la otra, de hecho, cada una tiene sus puntos fuertes y sus puntos débiles. Y todo dependerá de lo que realmente busques para que te tengas que decantar por una u otra.

Diferencias GitHub vs GitLab A pesar de todas las similitudes, una de las claves a la hora de decantarse en la comparativa GitHub vs GitLab pueden ser las diferencias entre ambas:

Niveles de autenticación: GitLab puede establecer y modificar permisos a los diferentes colaboradores según su función. En el caso de GitHub, puede

decidir quién tiene derechos de lectura y escritura en un repositorio, pero es más limitado en ese sentido.

- Alojamiento: aunque ambas plataformas permiten alojar el contenido de los proyectos en las propias plataformas, en el caso de GitLab también te puede permitir autoalojar tus repositorios, lo que puede ser una ventaja en algunos casos. GitHub ha agregado esa característica también, pero solo con ciertos planes de pago. GitHub ofrece máximo 3 colaboradores gratis y hasta 500 MB por repositorio, en cambio en GitLab no hay límite al número de colaboradores y hasta 10 GB por repositorio.

- Importación y exportación: GitLab contiene información muy detallada de cómo poder importar proyectos para moverlos de una plataforma a otra, como GitHub, Bitbucket, o traerlos a GitLab. Además, a la hora de exportar, GitLab ofrece un trabajo muy sólido. En el caso de GitHub no se ofrece documentación detallada, aunque se puede usar GitHub Importer como herramienta, aunque puede ser algo más restrictivo cuando se trata de exportar.

- Comunidad: ambos tienen una buena comunidad tras de sí, aunque GitHub parece haber ganado la batalla en popularidad. Actualmente aglutina millones de desarrolladores. Por eso, será más sencillo encontrar ayuda al respecto.

- Versiones Enterprise: ambos las ofrecen si pagas la cuota, por lo que se podría pensar que la comparativa GitHub vs GitLab no tiene sentido en este punto, pero lo cierto es que GitLab ofrece unas prestaciones muy interesantes, y se ha hecho popular entre los equipos de desarrollo muy grandes.

Ventajas y Desventajas de GitLab una vez conocidas las diferencias y semejanzas entre GitHub vs GitLab, las ventajas y desventajas de estas plataformas te pueden ayudar a decidirte.

Ventajas

- Plan gratuito y sin limitaciones, aunque tiene planes de pago.

- Es de licencia de código abierto.

- Permite el autohospedaje en cualquier plan.
- Está muy bien integrado con Git.

Desventajas

- Su interfaz puede ser algo más lenta con respecto a la competencia.
- Existen algunos problemas habituales con los repositorios.

Ventajas y Desventajas de GitHub Por otro lado, GitHub también tiene sus pros y contras, entre los que destacan los siguientes:

Ventajas

- Servicio gratuito, aunque también tiene servicios de pago.
- Búsqueda muy rápida en las estructura de los repos.
- Amplia comunidad y fácil encontrar ayuda.
- Ofrece prácticas herramientas de cooperación y buena integración con Git.
- Fácil integración con otros servicios de terceros.
- Trabaja también con TFS, HG y SVN.

Desventajas

- No es absolutamente abierto.
- Tiene limitaciones de espacio, ya que no puedes exceder de 100MB en un solo archivo, mientras que los repositorios están limitados a 1GB en la versión gratis.

Como ves, no existe un claro ganador. La elección no es fácil y, como comenté, deberías vigilar muy bien las ventajas, desventajas y diferencias de cada uno para poder identificar cuál se adapta mejor a tus necesidades.

Personalmente te diría que si quieres disponer de un entorno totalmente abierto, mejor usa GitLab. En cambio, si prefieres más facilidades y usar el servicio Web con más presencia, entonces ve a por GitHub. Incluso incluiría un tercero en discordia y te diría que si buscas trabajar con servicios Atlassian deberías mirar del lado de Bitbucket.

Uso GitLab (https://about.gitlab.com/)

Para configurar:

```
git config –global user.name "Antonio Carrillo Ledesma"
git config –global user.email "antoniocarrillo@ciencias.unam.mx"
```

Para crear nuevo repositorio:

```
git clone https://gitlab.com/antoniocarrillo69/MDF.git
cd MDF
touch README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

Para usar una carpeta existente:

```
cd existing_folder
git init
git remote add origin https://gitlab.com/antoniocarrillo69/MDF.git
git add .
git commit -m "Initial commit"
git push -u origin master
```

Para usar un repositorio existente:

```
cd existing_repo
git remote rename origin old-origin
git remote add origin https://gitlab.com/antoniocarrillo69/MDF.git
git push -u origin –all
git push -u origin –tags
```

Uso Github (https://github.com/)

Para configurar:

```
git config –global user.name "Antonio Carrillo Ledesma"
git config –global user.email "antoniocarrillo@ciencias.unam.mx"
```

Para configurar un nuevo repositorio:

```
$ touch README.md
$ git init
$ git add .
$ git commit -m "mi primer commit"
$ git remote add origin https://github.com/antoniocarrillo69/ejemploPruebas.git
$ git pull origin master
$ git push origin master
```

7.7 Incrementando la Seguridad a Nivel Usuario

La mejor opción, es elegir una distribución de GNU/Linux que nos permita mantener el sistema actualizado, instalar sólo los paquetes que necesitamos y que estos provengan de una fuente confiable, además de cifrar las particiones del sistema operativo y de datos del usuario.

Borrado de Archivos o Particiones de Forma Segura La herramienta *shred* ("ha-cer trizas" en español) -provista por el paquete *coreutils* instalado por omisión-, permite sobrescribir un archivo o dispositivo para ocultar y eliminar todo su contenido, sin dejar rastro alguno. A fin de que sea imposible de recuperar para herramientas de Hardware que hacen análisis magnético de sectores[65], *shred* sobrescribe repetidamente el contenido con valores aleatorios, haciendo un uso intensivo de disco[66]. A su vez permite eliminar el archivo al finalizar.

Por ejemplo, se procede a sobreescribir y borrar el archivo con *shred*:

```
$ shred -n 5 -u -v -z miArchivo.dat
```

[65] Es importante aclarar que esta herramienta se basa en la suposición importante, de que el sistema de archivos sobrescribe los datos en el lugar. Generalmente es así, pero muchos sistemas de archivos modernos no lo respetan. Por ejemplo los sistemas de archivos con Journal o estructurados en Log (por ejemplo *ext4*); sistemas de archivos que escriben datos redundantes como los esquemas *RAID*; sistemas de archivos que soportan Snapshots (*LVM* o *ZFS*); sistemas de archivos que hacen un uso intensivo de Cache en locaciones temporales (*NFS*); sistemas de archivos comprimidos; etc.

En tales casos *shred* no es efectivo, o no se garantiza que lo sea.

Para el caso de los sistemas de archivos *ext3* y *ext4* esta advertencia aplica sólo si el Journal está habilitado tanto para los datos como metadatos (modo data=journal). En los modos data=ordered (por defecto) y data=writeback, *shred* funciona correctamente.

[66] No es recomendado su uso en discos SSD por el desgaste acumulativo que provoca al mismo.

las opciones utilizadas son las siguientes:

-n 5: 5 iteraciones de sobrescritura.

-u: borrar el archivo al finalizar.

-v: muestre el progreso.

-f: fuerza el cambio de permisos para permitir la escritura de ser necesario.

-z: rellenar con ceros al final para ocultar el propio Shredding.

Se observa que en cada pasada se alterna entre rellenado con ceros, con datos aleatorios y con unos (ffffff). Esto se hace para maximizar la efectividad del borrado o Shredding, de forma que todos los Bits resulten modificados al menos una vez. Finalmente se renombra y se borra. Esto ocurre para hacer el Shredding del propio nombre del archivo en la entrada de directorio.

Lo podemos usar en múltiples archivos a la vez, mediante:

```
$ shred -u -v *.txt
```

o en una partición del disco (/dev/sda2/), mediante:

```
# shred -vfz /dev/sda2
```

También es posible usar el paquete *wipe*, para instalarlo usamos

```
# apt install wipe
```

y lo usamos mediante:

```
$ wipe archivo
$ wipe -r -q directorio/*
# wipe /dev/sda2
```

o el paquete *secure-delete*, para instalarlo usamos:

```
# apt install secure-delete
```

este paquete consta de cuatro herramientas, a saber:

- srm - utilizado para borrar archivos o directorios que se encuentren en disco

  ```
  $ srm archivo
  $ srm -r directorio/*
  ```

- sdmem - utilizado para limpiar restos de información en la memoria (RAM) de la máquina

  ```
  # sdmem -f -v
  ```

- sfill - utilizado para limpiar los restos de información del espacio vacío del disco

  ```
  # sfill -v /home/usuario
  ```

- sswap - utilizado para limpiar los restos de información de la partición Swap, para conocerla usamos:

  ```
  $ swapon
  ```

  supongamos que la partición Swap es: /dev/sda2, entonces:

  ```
  # swapoff /dev/sda2
  # sswap /dev/sd2
  # swapon /dev/sda2
  ```

Cifrado de Discos y Particiones Al momento de instalar el sistema operativo una buena práctica de seguridad es solicitar que use particiones cifradas para nuestros datos y el sistema operativo. Esto permite mantener a nuestros datos lejos de miradas indiscretas. Además cada disco externo o unidad Flash se recomienda cifrar, esto es posible al formatear el dispositivo -casi todos los escritorios de GNU/Linux tienen integrada una opción para ello-, así en caso de perdida o robo del dispositivo nuestros datos permanecen ilegibles para cualquiera.

En caso que nuestro escritorio no tenga instalada una opción gráfica para formatear y cifrar dispositivos, podemos usar GNOME Disks (también conocida como *gnome-disk-utility*, o *GNOME Disks*) es una aplicación gráfica de *udisks* incluido en el paquete *gnome-disk-utility*. Se puede utilizar GNOME Disk para la gestión de particiones (cifrar), motorización de tipo *SMART*, evaluación comparativa y Software *RAID*.

Para instalar usamos:

```
# apt install gnome-disk-utility
```

y para usar:

```
# gnome-disks
```

esto nos permite entre otras cosas al formatear una unidad y cifrarla.

Cifrar Discos y Particiones con cryptsetup es uno de los Softwares de cifrado de disco más usado. Este se encarga de cifrar/descifrar los datos escritos en un dispositivo de almacenamiento. El cifrado a nivel de disco garantiza que los archivos siempre se almacenen en el disco de forma cifrada. Los archivos solo están disponibles para poder ser leídos mientras el sistema está en ejecución y desbloqueado por uno de los usuarios con acceso a la clave de descifrado. Una persona no autorizada que intente acceder al contenido del disco, solo encontrará datos confusos, en lugar de los archivos reales.

Todos los métodos de cifrado de disco funcionan de tal manera que, aunque el disco realmente contiene datos cifrados, el sistema operativo lo ve como los datos legibles normales, siempre que el contenedor cifrado (es decir, la parte del disco que contiene los datos cifrados) ha sido desbloqueado y montado en alguna partición del sistema.

Instalar Paquete usamos:

```
# apt install cryptsetup
```

Es bueno conocer el rendimiento del programa *cryptsetup* para los diferentes tipos de cifrado soportado, para ello usamos:

```
# cryptsetup benchmark
```

algunos de ellos son:

> PBKDF2-sha1, PBKDF2-sha256, PBKDF2-sha512, PBKDF2-ripemd160, PBKDF2-whirlpool, argon2i, argon2id, aes-cbc 128, serpent-cbc 128, twofish-cbc 128, aes-cbc 256, serpent-cbc 256, twofish-cbc 256, aes-xts 256, serpent-xts 256, twofish-xts 256, aes-xts 512, serpent-xts 512, twofish-xts 512

esto nos permitirá elegir el algoritmo óptimo para nuestro equipo, que nos de el mejor rendimiento en el cifrado y mayor seguridad posible.

El primer paso es escoger la unidad de disco que vamos a cifrar, podemos usar el comando *lsblk* para obtener una lista de dispositivos de bloque montados en el equipo de cómputo, mediante:

```
$ lsblk
```

supongamos que trabajaremos con: */dev/sdb1*.
Crear partición cifrada

```
# cryptsetup -v -y -c aes-xts-plain -s 512 luksFormat /dev/sdb1
# cryptsetup luksOpen /dev/sdb1 Respaldos
# ls -l /dev/mapper/Respaldos
# cryptsetup -v status Respaldos
```

Formatear partición

```
# mkfs.ext4 /dev/mapper/Respaldos
# mkdir -p /opt/Respaldos_crypt
# mount /dev/mapper/Respaldos /opt/Respaldos_crypt
# df -Th | grep crypt
# cd /opt/Respaldos_crypt
```

Montar y activar sistema de archivos

```
# cryptsetup luksOpen /dev/sdc1 Respaldos
# mkdir -p /opt/Respaldos_crypt
# mount /dev/mapper/Respaldos /opt/Respaldos_crypt
# df -Th | grep crypt
# cd /opt/Respaldos_crypt
```

Desmontar y desactivar el sistema de archivos

```
# umount /opt/Respaldos_crypt
# cryptsetup luksClose Respaldos
```

El uso del dispositivo cifrado en la gran mayoría de las distribuciones de GNU/Linux que tengan instalado el paquete *cryptsetup* reconocerán el dispositivo al momento de ser introducido en el equipo de cómputo y solicitarán la clave de acceso. A partir de ese momento será posible usar el dispositivo como cualquier otro -sin cifrado- y todas las tareas de montaje/desmontaje serán transparentes para el usuario.

Podemos usar el comando fsck en sistemas basados en LUKS, mediante:

```
# umount /opt/Respaldos_crypt
# fsck -vy /dev/mapper/Respaldos
# mount /dev/mapper/Respaldos /opt/Respaldos_crypt
```

Para cambiar la contraseña para la partición cifrada, usamos:

```
# cryptsetup luksDump /dev/sdb1
# cryptsetup luksAddKey /dev/sdb1
```

se pueden configurar un máximo de 8 contraseñas para cada dispositivo
Remover o eliminar la contraseña:

```
# cryptsetup luksRemoveKey /dev/sdb1
```

tengamos en cuenta que debemos ingresar la contraseña guardada.

Cifrado Basado en Archivos con gocryptfs Podemos tratar de mantener nuestros datos fuera de miradas indiscretas usando gocryptfs en las máquina a las que tengamos acceso (incluso del usuario administrador *root)*[67]. Además, podemos respaldarlos y transportarlos manteniendo estos siempre cifrados y que sea casi transparente para nosotros el uso de dicho programa.

El programa *gocryptfs* utiliza cifrado basado en archivos que se implementa como un sistema de archivos *FUSE* que se puede montar. Cada archivo en *gocryptfs* se almacena como un archivo cifrado correspondiente en el disco. Los archivos cifrados se pueden almacenar en cualquier carpeta del disco, unidad *USB* o incluso dentro de una carpeta en la nube como *Google Drive* o *Dropbox*. Una ventaja del cifrado basado en archivos en comparación con el cifrado de disco es que los archivos cifrados de pueden sincronizar de

[67] Existen múltiples proyectos -algunos multiplataforma- que permiten hacer lo mismo que *gocryptfs* como son: *encfs, ecrptfs, cryptomator, securefs y CryFS*, entre otros.

manera eficiente utilizando herramientas como rsync. Además, el tamaño del sistema de archivos cifrado es dinámico y solo está limitado por el espacio disponible en disco.

Para instalarlo usamos:

```
# apt install gocryptfs
```

después, es necesario crear los directorios para guardar los datos cifrados (por ejemplo en el directorio ~/.cifrados) y los descifrados (por ejemplo en el directorio ~/descifrados), mediante:

```
$ mkdir -p ~/.cifrados ~/descifrados
```

Para inicializar la carpeta usamos:

```
$ gocryptfs -init ~/.cifrados
```

nos pedirá la clave de acceso y su confirmación. Se pueden crear tantas carpetas independientes con *gocyptfs* como se requieran y el cualquier parte del sistema de archivos al que tengamos acceso.

Ahora ya podemos montar el directorio mediante:

```
$ gocryptfs ~/.cifrados ~/descifrados
```

el sistema nos pedirá nuestra clave de acceso y de ser correcta nos permitirá hacer el montaje. De esta forma, la carpeta virtual *~/cifrados* mostrará nuestros datos descifrados, solo visibles para el usuario que monta la carpeta (no para root) y la carpeta *~/.cifrados*, contendrá nuestros datos de forma cifrada[68], esta carpeta puede ser copiada, respaldada y restaurada aún estando montada, manteniendo el anonimato de nuestros archivos.

Una vez que ya no sea requerida la carpeta, la desmontamos usando:

```
$ fusermount -u ~/descifrados
```

Es posible usar *gocryptfs* en modo inverso, primero debemos inicializar la carpeta usando:

```
$ gocryptfs -reverse -init ~/.descifrados
```

y usarla mediante:

```
$ gocryptfs -reverse ~/.descifrados ~/.cifrados
```

[68] El nombre de los archivos tienen un límite, por ello hay que tenerlo en cuenta si se usan nombre largos en el sistema de archivos (el límite aproximado es 255 caracteres), pero no importa el número de archivos o carpetas almacenadas internamente.

Cifrar Archivos con GnuPG es una herramienta en línea de comandos de seguridad en comunicaciones electrónicas en donde se utiliza criptografía de clave pública para que los usuarios puedan comunicarse de un modo seguro. En un sistema de claves públicas cada usuario posee un par de claves, compuesto por una clave privada y una clave pública. Cada usuario debe mantener su clave privada secreta; no debe ser revelada nunca. La clave pública se puede entregar a cualquier persona con la que el usuario desee comunicarse. GnuPG implementa un esquema algo más sofisticado en el que un usuario tiene un par de claves primario, y ninguno o más de un par de claves adicionales subordinadas. Los pares de claves primarios y subordinados se encuentran agrupados para facilitar la gestión de claves, y el grupo puede ser considerado como un sólo par de claves.

Dentro de las funciones de GnuPG se incluyen generar un par de claves, intercambiar y comprobar la autenticidad de claves, cifrar y descifrar documentos, etc. Para instalar el paquete GnuPG, usamos:

```
# apt install gnupg
```

La opción de la línea de comandos *–gen-key* se usa para generar un nuevo par de claves primario, mediante:

```
$ gpg –gen-key
```

también podemos pedir –full-generate-key, mediante:

```
$ gpg –full-generate-key
```

GnuPG es capaz de crear varios tipos diferentes de pares de claves, pero debe existir una clave primaria capaz de generar firmas. Al momento de ejecutar el programa, este pide[69] nombre, correo del usuario y contraseña[70], para después generar la clave.

[69] Según la versión de GnuPG puede solicitar otros datos como: el tipo de clave, longitud, cuando expira está, entre otros posibles datos de configuración.

[70] Cuanto más larga sea la contraseña, más segura será contra ataques de «fuerza bruta». No hay límite para la longitud de una contraseña, y esta debe ser escogida con sumo cuidado. Desde un punto de vista de seguridad, la contraseña que desbloquea la clave privada es uno de los puntos más débiles en GnuPG (y en otros sistemas de cifrado de clave pública), ya que es la única protección que tiene el usuario si alguien se apodera de su clave privada. Para una contraseña lo ideal es que no se usen palabras de un diccionario, y que se mezclen mayúsculas y minúsculas, dígitos, y otros caracteres. Una buena contraseña es crucial para el uso seguro de GnuPG.

Para poder comunicarse con otros, el usuario debe intercambiar las claves públicas. Para obtener una lista de las claves en el fichero («anillo») de claves públicas, se puede usar la opción de la línea de comandos *–list-keys*, mediante:

```
$ gpg –list-keys
```

también podemos usar:

```
$ gpg –list-secret-keys
$ gpg –list-public-keys
```

Para poder enviar una clave pública a un interlocutor, antes hay que exportarla. Para ello se usará la opción de la línea de comandos *–export*. Es necesario un argumento adicional para poder identificar la clave pública que se va a exportar, por ejemplo:

```
$ gpg –output antonio.gpg –export ant@na.mx
```

La clave se exporta en formato binario, y esto puede no ser conveniente cuando se envía la clave por correo electrónico o se publica en una página Web. Por tanto, GnuPG ofrece una opción de la línea de comandos *–armor* que fuerza que la salida de la orden sea generada en formato *armadura-ASCII*, parecido a los documentos codificados con *UUEncode*. Por regla general, cualquier salida de una orden de GnuPG, v.g. claves, documentos cifrados y firmas, pueden ir en formato *armadura-ASCII* añadiendo a la orden la opción *–armor*, por ejemplo:

```
$ gpg –armor –output antonio.gpg –export ant@na.mx
```

Cada clave pública y privada tiene un papel específico en el cifrado y descifrado de documentos. Se puede pensar en una clave pública como en una caja fuerte de seguridad. Cuando un remitente cifra un documento usando una clave pública, ese documento se pone en la caja fuerte, la caja se cierra, y el bloqueo de la combinación de esta se gira varias veces. La parte correspondiente a la clave privada, esto es, el destinatario, es la combinación que puede volver a abrir la caja y retirar el documento. Dicho de otro modo, sólo la persona que posee la clave privada puede recuperar un documento cifrado usando la clave pública asociada al cifrado.

Con este modelo mental se ha mostrado el procedimiento de cifrar y descifrar documentos de un modo muy simple. Si el usuario quisiera cifrar un mensaje para Javier, lo haría usando la clave pública de Javier, y lo descifraría con su propia clave privada. Si Javier quisiera enviar un mensaje al usuario, lo haría con la clave pública del usuario, y este lo descifraría con su propia clave privada.

Cifrar un Archivo para cifrar un archivo se usa la opción *–encrypt*. El usuario debe tener las claves públicas de los pretendidos destinatarios. El programa espera recibir como entrada el nombre del archivo que se desea cifrar o, si este se omite, una entrada estándar. El resultado cifrado se coloca en la salida estándar o donde se haya especificado mediante la opción *–output*. El archivo se comprime como medida adicional de seguridad, aparte de cifrarlo, por ejemplo:

$ gpg –output doc.gpg –encrypt –recipient ant@na.mx doc

La opción *–recipient* se usa una vez para cada destinatario, y lleva un argumento extra que específica la clave pública con la que sería cifrado el archivo. El archivo cifrado sólo puede ser descifrado por alguien con una clave privada que complemente una de las claves públicas de los destinatarios. El usuario, en este caso el remitente, no podría descifrar un archivo cifrado por sí mismo a menos que haya incluido su propia clave pública en la lista de destinatarios.

Descifrar un Archivo para descifrar un archivo se usa la opción *–decrypt*. Para ello es necesario poseer la clave privada para la que el archivo ha sido cifrado. De igual modo que en el proceso de cifrado, el archivo a descifrar es la entrada, y el resultado descifrado la salida, por ejemplo:

$ gpg –output doc –decrypt doc.gpg

Cifrar y Descifrar Usando Cifrado Simétrico también es posible cifrar archivos sin usar criptografía de clave pública. En su lugar, se puede usar sólo una clave de cifrado simétrico para cifrar el archivo. La clave que se usa para el cifrado simétrico deriva de la contraseña dada en el momento de cifrar el documento, y por razones de seguridad, no debe ser la misma contraseña que se esta usando para proteger la clave privada. El cifrado simétrico es útil

para asegurar archivos cuando no sea necesario dar la contraseña a otros. Un archivo puede ser cifrado con una clave simétrica usando la opción –*symmetric*, por ejemplo:

```
$ gpg –symmetric doc
```

o

```
$ gpg -c doc
```

y podemos descifrar usando:

```
$ gpg doc.gpg
```

Otras Opciones para Cifrar y Descifrar

ccrypt es otra herramienta para cifrar y descifrar archivos basada en el cifrado de bloque Rijndael, Se cree que *ccrypt* (basada en el cifrado de bloque *Rijndael*) proporciona una seguridad fuerte. Tengamos en cuenta que *ccrypt* elimina el archivo original (cifra el archivo en su lugar), no cambia significativamente el tamaño del archivo y no altera la fecha/hora del archivo para reflejar la hora en que se realizó el cifrado.

```
$ ccrypt -e archivo
```

para descifrar usamos:

```
$ ccrypt -d archivo.cpt
```

mcrypt el comando permite cifrar y descifrar archivos, cuando cifra deja el archivo original intacto y cambia los permisos del archivo para que el archivo cifrado solo proporcione permisos de acceso de lectura y escritura al propietario del archivo. Ofrece muchas opciones con respecto a los algoritmos de cifrado y también ofrece opciones para comprimir los archivos antes del cifrado (opciones -z comprime GZip y -p comprime Bz2). Puede funcionar con varios archivos, pero los cifra por separado, ejemplo:

$ mcrypt archivo

para descifrar usamos:

$ mcrypt -d archivo.nc

para enumerar los algoritmos de cifrado disponibles, usamos:

$ *mycrypt –list*,

El comando *mcrypt* parece usar *Rijndael-128* como su algoritmo de cifrado predeterminado.

age este comando permite cifrar usando una clave pública o frase, no viene instalado por omisión en el sistema, pero lo podemos instalar usando:

apt install age

Para generar una clave pública en el archivo llave.txt usamos:

$ age-keygen -o llave.txt

Para cifrar un archivo *archivo.tar* con clave pública usamos:

$ tar cvz archivos | age -r llave.txt > archivo.tar.gz.age

para descifrarlo usamos:

$ age –decrypt -i llave.txt > archivo.tar.gz

que nos retornará a nuestro archivo original.
Para cifrar un archivo *archivo.tar* con frase usamos:

$ age –passphrase –output archivo.tar.gz.age archivo.tar.gz

para descifrarlo usamos:

$ age –decrypt –output archivo.tar.gz archivo.tar.gz.age

que nos retornará a nuestro archivo original.

Generar Contraseñas Para generar contraseñas disponemos de múltiples opciones, algunas de ellas son:

- Podemos usar GnuPG, para instalarlo usamos:

  # apt install gnupg

  para generar una contraseña de 32 caracteres, usamos:

  $ gpg –gen-random –armor 1 32

- Podemos usar OpenSSL, para instalarlo usamos:

  # apt install openssl

  para generar una contraseña de 32 caracteres mediante:

  $ openssl rand -base64 32

- Podemos usar APG, para instalarlo usamos:

  # apt install apg

  para generar una contraseña de mínimo 31 y máximo 32 caracteres y genere 4 claves:

  $ apg -n 4 -m 31 -x 32 -a 1

- Podemos usar PWGEN, para instalarlo usamos:

  # apt install pwgen

  para generar una contraseña de 32 caracteres:

  $ pwgen -ys 32 1

- Podemos usar Makepasswd, para instalarlo usamos:

  # apt install makepasswd

  para generar una contraseña de 32 caracteres:

  $ makepasswd –char 32

Cifrado Avanzado En los procesadores modernos es común encontar el motor Intel/AMD Advanced Encryption Standard (AES) o New Instructions (AES-NI) que permite el cifrado y descifrado por Hardware de alta velocidad para el cifrado de disco completo, OpenSSL, ssh, VPN, Linux / Unix / OSX y más. ¿Cómo verifico la compatibilidad con Intel/AMD AES-NI en mi sistema basado en Linux, incluido OpenSSL?

El conjunto de instrucciones del estándar de cifrado avanzado y las nuevas instrucciones del estándar de cifrado avanzado permiten que Intel/AMD específicas y otras CPU realicen un cifrado y descifrado de hardware extremadamente rápido.

Tengamos en cuenta que la compatibilidad con AES-NI se habilita automáticamente si el procesador detectado lo soporta. Para obtener una lista de procesadores que admiten el motor AES-NI, consulte el sitio y la documentación del procesador. El AES-NI es una extensión de la arquitectura del conjunto de instrucciones x86 para microprocesadores de Intel y AMD. Aumenta la velocidad de las aplicaciones que realizan el cifrado y el descifrado mediante AES.

Uno puede descubrir que el procesador tiene el conjunto de instrucciones AES / AES-NI usando el comando:

```
$ lscpu
```

o

```
$ grep -o aes / proc / cpuinfo
```

o

```
$ grep -m1 -o aes / proc / cpuinfo
```

o

```
# cpuid | grep -i aes | sort | uniq
```

En cualquiera de los casos si aparece la bandera *aes* el conjunto de instrucciones AES / AES-NI está presente y activo.

También podemos usar el siguiente comando para verificar la compatibilidad con AES-NI en nuestro procesador:

```
$ sort -u /proc/crypto | grep module
```

Ahora que hemos verificado la compatibilidad, es hora de probarla, para ello usamos:

```
$ openssl engine
```

ejemplo de una salida que soporta AES:

```
(padlock) VIA PadLock (no-RNG, no-ACE)
(dynamic) Dynamic engine loading support
```

ejemplo de una salida que soporta AES-NI:

```
(aesni) Intel AES-NI engine
(dynamic) Dynamic engine loading support
```

y podemos probar su desempeño entre un equipo que soporte el cifrado:

```
$ dd if=/dev/zero count=1000 bs=1M | ssh -l usuario -c
aes128-cbc maquina "cat >/dev/null"

1000+0 records in

1000+0 records out

1048576000 bytes (1.0 GB) copied, 10.6691 s, 98.3 MB/s
```

y uno que no la soporte:

```
$ dd if=/dev/zero count=1000 bs=1M | ssh -l usuario -c
aes128-cbc maquina "cat >/dev/null"

1000+0 records in

1000+0 records out

1048576000 bytes (1.0 GB) copied, 31.6675 s, 33.1 MB/s
```

¿Cómo puedo comparar el rendimiento de OpenSSL? podemos correr los comandos:

```
$ openssl speed
```

o

```
$ openssl speed aes-128-cbc
```

Para la última versión de OpenSSL, podemos probar los dos comandos, el segundo comando debería tener números más altos que el primero gracias a EntropyZer0:

```
$ openssl speed aes-256-cbc
$ openssl speed -evp aes-256-cbc
```

8 Consideraciones y Comentarios Finales

Los paquetes comerciales -de Software privativo- en general proveen un ambiente integrado de trabajo ideal para las empresas de todo tipo, pero además puede ser usado en la preparación de estudiantes para aplicar sus conocimientos al egresar en las diversas áreas de las carreras universitarias, esto les permite laborar en empresas pequeñas, medianas y grandes con un mínimo de capacitación técnica adicional.

En un mercado tan competitivo como el actual, las organizaciones actuales focalizan sus recursos en las estrategias más adecuadas para conducir a la compañía hacia el éxito. Los paquetes comerciales y los incipientes paquetes de Software libre pueden ayudar a conseguir este objetivo, completando la inversión ya realizada en sistemas operacionales.

Pero el hecho de que las organizaciones actuales, manejan una gran cantidad de información, la cual puede o no estar dispersa en sus múltiples sistemas operacionales, requiere usar paquetes que tengan integrado el manejo de las grandes bases de datos distribuidas o centralizadas, esta integración ofrece beneficios adicionales.

Por otro lado, notemos que, una vez que un producto de Software libre ha empezado a circular, rápidamente está disponible a un costo muy bajo. Al mismo tiempo, su utilidad no decrece. El Software libre, en general, podría ser considerado un bien de uso inagotable, tomando en cuenta que su costo marginal es pequeño y que no es un bien sujeto a rivalidad (la posesión del bien por un agente económico no impide que otro lo posea).

Puesto que el Software libre permite el libre uso, modificación y redistribución, a menudo encuentra un hogar entre usuarios para los cuales el coste del Software no libre es a veces prohibitivo, o como alternativa a la piratería (véase 9.5). También es sencillo modificarlo localmente, lo que permite que sean posibles los esfuerzos de traducción a idiomas que no son necesariamente rentables comercialmente.

La mayoría del Software libre se produce por equipos internacionales que cooperan a través de la libre asociación. Los equipos están típicamente compuestos por individuos con una amplia variedad de motivaciones, y pueden provenir tanto del sector privado, del sector voluntario o del sector público.

En México el Software libre nació en las Universidades y los Centros de Investigación. Es por eso que, desde hace cuatro décadas, los estudiantes y los profesores usan Software libre para fines didácticos y de investigación. Las universidades suelen optar por el uso de Software libre en vez de utilizar

Software privativo porque satisface de una mejor manera sus necesidades de cómputo, dada su naturaleza de apertura del código y la libertad de compartir los resultados obtenidos. De forma colateral, no se tienen gastos adicionales derivados del pago de licenciamientos.

Computólogos, físicos, químicos, matemáticos, otros profesionistas y científicos utilizan Software libre como herramienta de investigación y creación. Un claro ejemplo de ello es la llamada Delta Metropolitana, que es una red de supercomputadoras que están en varios puntos de la Ciudad de México, en el CINESTAV, el IPN, la UAM y la UNAM. Esa red de supercómputo utiliza Software libre para consolidar sus recursos, hacer investigación y generar conocimiento.

Por otro lado, dadas las características del Software de código cerrado, un usuario común ignora absolutamente el contenido del mismo y por tanto si existe dentro de las líneas del código alguna amenaza contra su equipo o su información, el usuario no sólo tiene prohibido el intentar eliminar o cambiar esa parte del código sino que puede ser perseguido por la ley por el hecho de intentar conocer si existe tal amenaza en dicho Software.

Además, en una sociedad de la información, el Software se ha convertido en una herramienta importante de productividad y una licencia de Software privativo constituye un acuerdo o contrato entre dos sujetos jurídicos que voluntariamente acuerdan las condiciones de uso de un programa, pero el costo económico de dicha licencia es cada vez más alto y en el caso de instituciones educativas, gubernamentales y sociedades civiles es en la mayoría de los casos -por decir lo menos- prohibitivo.

Hace un tiempo, en varios periódicos de circulación nacional (véase [32]) fue publicado el siguiente anuncio:

> El Instituto Mexicano de la Propiedad Industrial (IMPI) anunció que en las próximas semanas dará inicio una serie de clausuras a negocios que utilicen Software licenciado de manera ilegal; esto como parte del acuerdo que tiene la dependencia con The Software Alliance (BSA) desde el 2002, el cual busca fomentar el uso de programas informáticos legales y disminuir el índice de piratería en el país.
>
> De acuerdo a la BSA, el porcentaje de Software ilegal utilizado en el territorio aún se ubica en un 56 por ciento, cifra considerablemente menor a lo visto en el 2005, cuando el número ascendía

> a más del 65 por ciento. Sin embargo, México continúa siendo uno de los mayores compradores de piratería a nivel mundial, y lo que se busca con este tipo de acciones en el 2013 es disminuir, al menos, un punto porcentual.
>
> "Si como consecuencia de una visita de inspección completa se encuentra la existencia de Software ilegal, se procede a la sanción. En el 2012 incrementaron hasta un 200% las sanciones por el uso ilegal de Software", dijo Kiyoshi Tsuru, director general en México de la BSA.
>
> Aquí es donde algunos se preguntarán, ¿y qué autoridad tiene The Software Alliance para ejecutar estas determinaciones? Pese a que cuenta con el apoyo de empresas como Microsoft, Apple, Autodesk, Adobe, Aveva, AVG, CISCO, Dell, Hewlett Packard, IBM, SAP y Symantec, lo cierto es que la BSA no puede clausurar legalmente ningún negocio. La verdadera autoridad llega en su acuerdo con el IMPI, el cual sí tiene las facultades para aplicar sanciones.

Además, la UNAM, desde junio 9 del 2009 firmó un acuerdo (véase [33]):

> Con el objetivo de fomentar la cultura de la legalidad en lo que se refiere al uso del Software entre los estudiantes, la Universidad Nacional Autónoma de México y la Business Software Alliance (BSA) firmaron un convenio general de colaboración.
>
> Mediante este acuerdo, se promoverá el uso ético de las tecnologías de la información y comunicación, y se compartirán conocimientos en materia de propiedad intelectual y Software, a fin de apoyar el desarrollo y explotación de bienes digitales en la UNAM, así como definir programas para contribuir al avance de un mundo digital seguro, informaron ambas organizaciones en un comunicado.
>
> El secretario general de la máxima casa de estudios, Sergio M. Alcocer Martínez de Castro, reconoció que la UNAM necesita hacer un esfuerzo en el ámbito institucional y en cada una de las entidades que la conforman, para brindar educación a sus alumnos, profesores y trabajadores en este campo.

> "Se pretende", destacó, "que el convenio sea operativo y que se desarrolle en cada una de las entidades con la impartición de cursos y capacitación y en una rendición de cuentas para que el Software que se utilice sea legal".
>
> El funcionario reconoció a los miembros de Business Software Alliance en Latinoamérica, y apuntó que la Universidad Nacional hará lo necesario para facilitar el uso responsable, ético y seguro del Software.
>
> Informó también que ambas partes se reunirán seis meses después del inicio de este convenio de colaboración para analizar los avances.
>
> En tanto, el director General de BSA-México, Kiyoshi Tsuru Alberú, resaltó que con la firma de este convenio podrán impulsar un plan conjunto relacionado con alta tecnología, ética y legalidad "Estamos seguros que en el mediano plazo se tendrán resultados importantes y se podrá hacer la diferencia", señaló.
>
> Por su parte, el abogado general, Luis Raúl González Pérez, comentó que la UNAM busca difundir estos valores entre su comunidad, y llegar especialmente a los estudiantes que inician el bachillerato, porque desde edad temprana es importante fomentar la cultura de la legalidad.

Ante este escenario, una alternativa viable podría ser optar por el Software libre, aunque, pese a su incipiente desarrollo es seguro que en un futuro podría alcanzar a suplir todas las necesidades básicas de los usuarios, dejando la adquisición de paquetes especializados sólo para los cursos avanzados que justifique el uso de Software privativo.

8.1 El Cómputo en Instituciones Educativas

Hace algunos años la disposición de un equipo de cómputo por cada estudiante era algo difícil de satisfacer para las instituciones educativas. Ahora, las cosas son distintas, cada vez más estudiantes disponen y tienen acceso a dispositivos de cómputo -computadoras de escritorio, portátiles, tabletas, y teléfonos inteligentes- que en principio pareciera que permitirían satisfacer la creciente demanda de recursos computacionales de los estudiantes.

Pero una computadora requiere de un sistema operativo además de los diversos paquetes de Software -que estén disponibles para esa versión del sistema operativo- que permitan resolver los problemas para los cuales usa el equipo de cómputo. Aquí es donde empiezan los problemas para los usuarios de equipos de cómputo, puesto que hay una gran cantidad de equipos de cómputo con diversas tecnologías y recursos que soportan alguna versión de sistema operativo acorde a los recursos computacionales del equipo adquirido que no necesariamente soportan a todos y cada uno de los programas de cómputo que el usuario requiere.

Ante la creciente necesidad de programas de cómputo podríamos pensar en que cada usuario que requiera hacer uso de ellos tenga acceso a un equipo de cómputo adecuado, conjuntamente con el sistema operativo que lo soporte. Pero esto dista mucho de la realidad, puesto que la gran mayoría de los usuarios no pueden hacer esos gastos y menos una institución educativa y sus respectivos estudiantes.

¿Entonces qué opciones tenemos para satisfacer la creciente demanda de recursos computacionales?

- Por un lado, si ya disponemos de un equipo de cómputo con su respectivo sistema operativo, entonces hacer uso de sólo aquellos programas de cómputo que nuestro equipo soporte, teniendo cuidado de no instalar programas de cómputo antagonistas.

- Otra opción es, si ya disponemos de un equipo de cómputo, entonces tener dos o más versiones de sistema operativo que permitan instalar una mayor diversidad de programas de cómputo y tener el cuidado de no instalar programas de cómputo incompatibles. Así, dependiendo de nuestras necesidades podemos hacer uso de uno u otro sistema operativo y sus respectivos programas.

- La opción más viable, es una que conjugue las dos anteriores. Pero además, podríamos emular Hardware del que no disponemos mediante el uso de máquinas virtuales, escritorios remotos y virtuales que nos permitirían en un sólo equipo de cómputo usar simultáneamente diversos sistemas operativos para distintas arquitecturas y sus respectivos programas que ahora es posible instalar en las máquinas virtuales programas de cómputo incompatibles de forma aislada unos de otros.

Usando esta última opción es posible satisfacer en un sólo equipo de cómputo una gran variedad de necesidades computacionales. Esto permite que a nivel de usuario (estudiante, ayudante y profesor) o institución educativa, el equipo de cómputo usando Software de virtualización pueda proporcionar un marco que permita satisfacer las diversas y crecientes necesidades computacionales. Pero hay que notar que aún esta opción no está exenta de problemas legales y técnicos, pero en principio es una opción viable para la gran mayoría de los usuarios y la institución educativa.

Tomando esto en cuenta, es viable tener una cantidad adecuada de paquetes de cómputo, que permitieran satisfacer las necesidades especializadas de la gran mayoría de los cursos y estos estar instalados en aquellos espacios en los cuales se asignarían los cursos, además de las áreas comunes de cómputo en la que los estudiantes requiriesen hacer uso de dichos paquetes. Además, de proporcionar un mecanismo para que los profesores y ayudantes que requieran enseñar algo con alguna versión privativa que no se disponga, sea implementada -en medida de lo posible- en los paquetes disponibles.

Pero hay que hacer notar, que no todas aquellas funciones que hace una versión particular de un paquete, es posible hacerlas con otras versiones o paquetes alternativos. Esto es muy común con ciertas actividades especializadas -al hacer cálculo simbólico, cálculo numérico, manejo de datos y trabajar en entornos de desarrollo-. Ello implicaría, por un lado restringir el Software instalado en los equipos de cómputo o por el otro instalar todas y cada una de las solicitudes de Software, aún cuando se requiera más de una versión de un paquete particular.

El restringir el Software instalado, impediría al profesor -que así lo requiera por la libertad de cátedra- enseñar aquello que considera que es necesario -en particular el manejo de uno o más paquetes especializados de cómputo- para proporcionar las herramientas básicas a sus alumnos y que estos deben de dominar para aprobar su curso.

En el caso de dar flexibilidad, para que cada profesor solicite la instalación del paquete o los paquetes que requiera para sus cursos, implica que el Software solicitado puede o no contar con licencia adecuada de uso. Así, se estaría permitiendo que se tenga instalado Software del que se viola la licencia de uso.

En cuanto a tener la lista definitiva de Software que usarán todos y cada uno de los profesores o ayudantes de los cursos asignados a un espacio es difícil tener antes del inicio del curso -por la constante evolución del Software y las cambiantes necesidades de la enseñanza-, además de depender de la forma de

asignación de estos en los laboratorios y talleres de cómputo. En cuanto a la solicitud para hacer la instalación correspondiente, se requiere tener certeza de en qué espacio serán asignados todos y cada uno de los cursos.

Por ello se han buscado opciones[71] -no siempre las más adecuadas o lícitas (véase 9.5)- para que sin importar en qué espacio sea asignado el curso -siempre y cuando el equipo de cómputo lo soporte- se tenga desde los primeros días de uso del espacio el paquete solicitado y en casos excepcionales el tiempo de espera sea menor a unos horas o días sin importar la plataforma -Windows o Linux- o el tipo de Software solicitado -libre o privativo-.

Por ejemplo, se puede optar por la virtualización[72], usando como sistema operativo base Debian GNU/Linux estable, instalando como paquete de virtualización a KVM/QEMU. Aquí, se montarían las múltiples máquinas virtuales que serían ejecutadas según las necesidades del usuario -para cualquier versión de Windows, Linux u otro sistema operativo de cualquier arquitectura de Hardware soportada por QEMU-. Para controlar la actualización de las máquinas virtuales sin que se requiera intervención del usuario, se usaría RSYNC tunelizado mediante SSH que sincronizaría las máquinas virtuales y la configuración del equipo base de forma remota.

Para tener la flexibilidad anteriormente comentada, es necesario poder contar con distintas versiones de sistemas operativos, de cada una de las versiones -en caso de Windows, tener independientemente los Service Pack-. De tal forma que sea posible instalar cada versión de Software solicitada en la plataforma adecuada, teniendo en cuenta que muchas versiones del Software son mutuamente excluyentes para ser instaladas en una misma versión del sistema operativo simultáneamente.

Por todo lo anterior, el uso de máquinas virtuales -que permiten tener múltiples versiones de sistemas operativos independientemente, así como de una versión particular tener por separado cada una de ellas con los respectivos Service Pack- es una opción viable para proporcionar el servicio de

[71] En el caso que el equipo sólo tenga un sistema operativo sin virtualización, es necesario esperar a que las asignaciones de los cursos y sus respectivas peticiones de uso de paquetes de cómputo estén completas, para entonces proceder a realizar instalación del Software que no sean antagónicos. Nótese que, por lo general, los cursos requieren el uso de los equipos de cómputo y el Software solicitado de forma inmediata, por lo cual esperar tiempo (días) para tener acceso al mismo no es una opción viable.

[72] Una vez creada la máquina virtual, esta es un archivo que puede ser copiado o descargado de la red, por ello el usuario -estudiante, ayudante o profesor- puede llevarse la máquina virtual para hacer uso de ella en el equipo al que tenga acceso, teniendo como único requisito tener instalado el programa de virtualización.

instalación centralizada de los diversos paquetes de cómputo solicitados por los profesores de las diversas carreras universitarias. Esta opción minimiza los tiempos de espera para la instalación de un paquete en particular y agiliza las prestaciones a todos y cada uno de los grupos que se atienden semestralmente en los cientos de equipos en los laboratorios y talleres de cómputo.

8.2 Integración del Cómputo en Ciencias e Ingenierías

El uso de programas de cómputo está integrado a las carreras de Ciencias e Ingenierías desde hace mucho tiempo, pero la gran mayoría se realiza con productos propietarios, lo cual no representa ningún problema técnico, pero sí un problema para la institución y estudiantes, ya que las versiones actualmente usadas, no son del todo compatibles entre sí, ello implica que se requiere o tener la última versión del producto o diferentes versiones del mismo para trabajos cotidianos en una misma computadora.

El uso de programas de cómputo de Software libre está cada día más integrado al uso cotidiano que hacen profesores, ayudantes y estudiantes en la carreras de Ciencias e Ingenierías, pero todavía para el Sistema Operativo Windows, así como para paquetes de uso común, no ha sido posible encontrar un adecuado reemplazo, los más comunes son MATLAB, Mathematica, Maple, SPSS, SAS y Microsoft Office.

Para las Universidades, el contar con las licencias necesarias para que cada máquina a la que los alumnos tienen acceso cuente con una, es en extremo prohibitivo por el costo. Esto mismo sucede en el caso de los estudiantes, pues el costo de una sola licencia para uso académicos es onerosa más si consideramos la diversidad de programas requeridos para una sola meteria y esto pasa con cada uno de los cursos de la carrera.

Es por ello que el uso de herramientas de Software libre se visualiza como un reemplazo natural a los paquetes propietarios, pero la realidad dista de ser tan simple. Ya que, actualmente no es posible obtener las características mínimas en Software libre para que puedan ser un reemplazo real de los paquetes de propietarios. Este hecho ha ocasionado que existe un uso cada vez más generalizado entre profesores y alumnos a usar Software sin la licencia respectiva (véase 9.5).

por ejemplo, en la UNAM, a través de la Dirección General de Cómputo y de Tecnologías de la Información y Comunicación se dispone de un restringido número de paquetes y versiones que son puestos a disposición de la comunidad universitaria para usar en los equipos personales sin aparente costo para el

usuario final -pero el costo de dichos paquetes son deducidos por la empresa como una donación, lo cual sí implica un costo real que se deduce en el ejercicio fiscal de la empresa donante y éste repercute en los ingresos que el gobierno no recaudará por motivo de impuestos-.

8.3 Ventajas, Desventajas y Carencias del Software Libre

Notemos que la ventaja de tener múltiples herramientas para realizar operaciones elementales y avanzadas de paquetes de cálculo numérico, simbólico, estadístico y ofimático es en sí misma una gran ventaja. Para los centros universitarios y usuarios ocasionales, las herramientas de Software libre son una herramienta invaluable, en el caso de empresas que requieren usar opciones avanzadas o generadas por terceros, los paquetes propietarios destacan como herramientas de trabajo óptimas. Pero para todos los casos, hay que destacar:

- Funcionalidades básicas: Todos los paquetes implementan las funcionalidades básicas, ya que todos los paquetes llevan años desarrollándose.
- Funcionalidades avanzadas: Por mucho, los paquetes propietarios tienen implementadas cientos de funciones avanzadas que pueden ser muy útiles para usuarios avanzados, pero rara vez son usados por los usuarios noveles o cotidianos.
- Fiabilidad: En los paquetes en desarrollo son comunes las caídas del programa, pero en los de Software propietario se destaca por ser más fiable que los demás.
- Información: El Software propietario son paquetes con una abundante bibliografía y la propia ayuda del programa.
- Facilidad de Manejo: Ninguno de los programas presenta grandes dificultades a la hora de su uso. Pero en menor o mayor medida, todos los paquetes del Software libre presentan entornos de desarrollo funcional, pero perfectible.
- Costo: El costo de las diversas versiones de Software propietario suele ser prohibitivo para instituciones educativas y usuarios ocasionales, en

el caso del Software libre, los paquetes se pueden descargar de la red sin más costo que el acceso a Internet y los medios de instalación cuando son requeridos.

El Software libre es aún joven, en los miles de proyectos actuales se está trabajando a diario en mejorar la parte computacional de los algoritmos involucrados en el paquete, haciendo y puliendo interfaces gráficas, generando ayuda en línea así como la documentación necesaria para que usuarios noveles y avanzados usen la mayor cantidad de opciones programadas en los paquetes.

Para muestra de este maravilloso avance, tomemos el proyecto del Kernel de Linux y su uso en los sistemas operativos Android, Ubuntu, Debian GNU/Linux, que actualmente se ejecuta en millones de equipos y contiene miles de aplicaciones y están soportados por una gran cantidad de usuarios y empresas comerciales. Estos han logrado desplazar a muchos de sus competidores por sus múltiples bondades y bajo costo de desarrollo, al reusar miles de aplicaciones ya existentes que usan Software libre y permitir desarrollar otro tanto de aplicaciones bajo una plataforma que se ejecutan en los más diversos procesadores.

Así también, en los últimos años, muchos proyectos han pasados de ser simples programas en línea de comandos a complejas aplicaciones multiplataforma -ejecutan en distintos sistemas operativos como son Windows, Linux y Mac- con ambientes gráficos multimedia que en muchos casos han superado a sus contrapartes comerciales, por ejemplo los navegadores Web tipo FireFox y la suite ofimática tipo LibreOffice, entre muchos otros.

8.4 Comentarios Finales

A diferencia de otros paquetes, SPSS, SAS, Microsoft Office, etc. Ofrecen soluciones en forma de una suite completa para la gestión de información para encontrar el llamado poder del conocimiento, pero el costo de las versiones completas y aún las educativas es prohibitivo para la gran mayoría de las instituciones educativas, en particular para la UNAM. Por ello, el resto de los paquetes propietarios y libres ofrecen una ventaja competitiva, al permitir al profesor y sus estudiantes contar con versiones completas y funcionales en las que pueden ser aplicados los conocimientos adquiridos en los diversos cursos de la carrera.

Por otro lado, para reforzar la apropiación del Software libre por parte de la comunidad de la UNAM, es necesario proporcionar a la comunidad

demostraciones y cursos cortos de las herramientas de Software libre, iniciando con mostrar el uso de sistemas operativos libres basados en Linux. Ello es posible haciendo uso de los sistemas llamados Live[73], ya que cada alumno puede probar y usar el sistema operativo en conjunto con cientos de herramientas libres, sin la necesidad de instalar Software en la máquina que utilice para practicar. Cuando el alumno se sienta cómodo con el sistema, es posible ayudarlo a instalar mediante tutoriales en línea y/o presenciales el sistema en su equipo de cómputo.

Lo mismo es posible hacer, al preparar demostraciones del Software que puede reemplazar paquetes muy difundidos en la comunidad como son: MATLAB, Mathematica, Maple, SPSS, SAS y Microsoft Office. Estos cursos no necesariamente se centrarían en las similitudes o diferencias entre paquetes libres y propietarios, más bien, para cautivar a usuarios noveles y futuros ayudantes a dar cursos completos de las herramientas libres mostrando su

[73]Un Live CD/DVD o USB, más genéricamente Live Distro, es un sistema operativo almacenado en un medio extraíble, tradicionalmente un CD/DVD o USB (de ahí sus nombres), que puede ejecutarse directamente en una computadora.

En la historia más reciente de Linux, las llamadas distribuciones Live Distro se han vuelto muy populares porque le permiten probar una distribución de Linux sin siquiera instalarla en el equipo. Esto es excelente porque no tiene todas las molestias de volver a particionar el disco o instalarlo sobre su sistema operativo (Windows/Mac OS). Simplemente puede colocar el CD/DVD o USB para una distribución en vivo e iniciar la computadora desde ahí. Por lo general, obtiene la mayor parte de la funcionalidad principal de la distribución, por lo que realmente puede evaluar si la distribución es para usted antes de elegir instalarla de verdad.

Normalmente, una versión Live viene acompañado de un par de aplicaciones. Algunos Live CD/DVD o USB incluyen una herramienta que permite instalarlos en el disco duro. Otra característica es que por lo general no se efectúan cambios en la computadora utilizada.

Para usar una versión Live es necesario obtener uno (muchos de ellos distribuyen libremente una imagen ISO que puede bajarse de Internet y grabarse en disco/USB) y configurar la computadora para que arranque desde la unidad lectora, reiniciando luego la computadora con el disco en la lectora o USB, con lo que el que el sistema Live se iniciará manualmente.

Uno de los mayores inconvenientes de este sistema es el mal uso de una gran cantidad de memoria RAM, una parte para su uso habitual y otra para funcionar como el disco virtual del sistema. En el arranque, se le pueden dar distintos parámetros para adaptar el sistema a la computadora, como la resolución de pantalla o para activar o desactivar la búsqueda automática de determinado Hardware.

Otro inconveniente es el rendimiento de la Live Distro, pues la velocidad de transferencia de las unidades lectoras CD/DVD o USB es muy inferior a la de los discos duros. Una vez instalada en la computadora se apreciará la velocidad real de la distribución.

aplicabilidad en diferentes ramas de las matemáticas aplicadas.

Para realizar dichos cursos, se cuenta con todos los recursos necesarios. Por un lado, se dispone de laboratorios y talleres con Software libre instalado en los equipos de cómputo, además, se pueden usar los sistemas "Live" que pueden ser proporcionados en DVDs o en unidades flash USB. Estas últimas, proporcionan mejor rendimiento, pueden ser actualizadas y reutilizados tantas veces como sea necesario para conocer uno o más sistemas operativos. Estos sistemas "Live" pueden ser generados por el propio usuario, usando las decenas de paquetes disponibles en Windows o Linux que generan sistemas "Live" a partir de las imágenes ISO bajadas de la red -por ejemplo, de sistemas operativos como Ubuntu, Debian, etc.-.

De esta forma, se puede coadyuvar a que alumnos, ayudantes y profesores conozcan el mundo del Software libre, para que con el tiempo se adopte su uso, sin dejar de lado, el proporcionar cuando sea necesario, cursos de Software privativo pero siempre teniendo en cuenta que se puede -en medida de lo posible- trabajar con paquetes alternativos, como los que proporciona el Software libre.

Además, el Software libre ofrece una ventaja competitiva, al permitirle al profesor y sus estudiantes contar con versiones completas y funcionales en las que pueden ser aplicados los conocimientos adquiridos en los diversos cursos de las carreras de Ciencias e Ingenierías, dejando el manejo especializado de paquetes a cursos avanzados o para cuando el educando realice sus prácticas profesionales. De esta forma se pueden preparar a los estudiantes para aplicar sus conocimientos al egresar en diversas áreas de la carreras de Ciencias e Ingenierías y con pocos conocimientos técnicos adicionales puedan laborar en pequeñas, medianas y grandes empresas.

9 Apéndice A: Software Libre y Propietario

Con el constante aumento de la comercialización de equipos de cómputo y/o comunicación (teléfonos inteligentes, tabletas, computadoras portátiles y de escritorio, etc.) y su relativo bajo costo, estos equipos se han convertido en objetos omnipresentes en nuestra vida diaria, ya que estos permiten realizar un creciente número de actividades cotidianas de miles de millones de usuarios.

Dichos equipos de cómputo y/o comunicación por sí solos tienen poca utilidad, pero su uso en conjunción con el Software adecuado forman un dúo que nos ha permitido tener los avances de los que actualmente disfrutamos. El Software -sistema operativo y los programas de aplicaciones- son los que realmente generan las soluciones al interactuar uno o más paquetes informáticos con los datos del usuario. También, es común que al comprar un equipo de cómputo y/o comunicación, en el costo total, se integre el del sistema operativo, aplicaciones ofimáticas y de antivirus, sean estos usados por el usuario o no y en la mayoría de los casos no es posible solicitar que no sean incluidos en el costo del equipo.

Por otro lado, el Software comercial suele quedar obsoleto muy rápido, ya que constantemente se le agregan nuevas funcionalidades al mismo y estas en general son vendidas como versiones independientes de la adquirida originalmente. Esto obliga al usuario -si quiere hacer uso de ellas- a comprar las nuevas versiones del Software para satisfacer sus crecientes necesidades informáticas y la obsolescencia programada.

Por lo anterior y dada la creciente complejidad de los paquetes de cómputo y el alto costo de desarrollo de aplicaciones innovadoras, en muchos casos, el costo total del Software que comúnmente los usuarios instalan -y que no necesariamente usan las capacidades avanzadas del programa, por las cuales el Software tiene un alto costo comercial- en sus equipos, suele ser más caro que el propio equipo en el que se ejecutan.

Hoy en día los usuarios disponemos de dos grandes opciones para adquirir el Software necesario para que nuestros equipos funcionen, a saber:

- Por un lado, podemos emplear programas comerciales (Software propietario), de los cuales no somos dueños del Software, sólo concesionarios al adquirir una licencia de uso del Software y nos proporcionan un instalable del programa adquirido. La licencia respectiva es en la gran mayoría de los casos muy restrictiva, ya que restringe su uso a un solo

equipo y/o usuario simultáneamente.

- Por otro lado, existe el Software libre[74], desarrollado por usuarios y para usuarios que, entre otras cosas, comparten los códigos fuente, el programa ejecutable y dan libertades para estudiar, adaptar y redistribuir a quien así lo requiera el programa y todos sus derivados.

Sobre la Obsolescencia Programada Es un conjunto de estrategias deliberadas destinadas a asegurarse que la versión actual de un determinado producto quedará desfasada o inservible en un plazo de tiempo predeterminado. De esta manera, los fabricantes se aseguran que los consumidores se verán obligados a reemplazarlo aunque funcione adecuadamente.

La obsolescencia puede lograrse mediante la introducción de un modelo con características superiores o diseñando intencionadamente un producto para que deje de funcionar correctamente en un plazo determinado. En cualquiera de los dos casos, se espera que los consumidores opten por el nuevo producto de la misma marca. Muchas veces la obsolescencia no es sobre el propio producto sino aplicando restricciones al producto de un competidor con la ayuda de una tercera empresa.

Tipos de Obsolescencia Programada Podemos dividir la obsolescencia programada en 4 tipos:

1- Establecimiento artificial del plazo de duración: Los productos se fabrican con piezas cuya duración tienen una vida útil limitada cuando, si se usaran otras de calidad superior ese plazo se extendería.

2- Actualizaciones de Software: Los desarrolladores de Software sacan nuevas versiones de sus aplicaciones que en un momento determinado dejan de ser compatibles con dispositivos antiguos. En muchos casos se ha podido comprobar que esa incompatibilidad es absolutamente artificial ya que al «engañar» al Software este funcionaba sin problemas.

[74] A veces también se han usado términos como FOSS y FLOSS. Ambas cosas son similares, ya que FOSS (Free and Open Source Software) traducido como "Software de código abierto" y FLOSS (Free/Libre and Open Source Software) "Software libre y de código abierto". Según quienes adoptan estos términos, lo hacen por tener una imparcialidad entre la carga filosófica del Software libre y el aspecto técnico y/o las ventajas que brinda este modelo de desarrollo. Richard Stallman nos invita a no usarlas y no se trata de un ad hóminem. Stallman y el proyecto GNU nos aconsejan que hablemos siempre de Software libre y aquí no cabe imparcialidad.

3- Obsolescencia percibida: Esta es una táctica psicológica, se trata de convencer al consumidor mediante publicidad y el uso de influenciadores de que el producto que se tiene actualmente está viejo y que se necesita uno nuevo. Como por ejemplo: ¿cuantos megapíxeles necesitas en tu teléfono para sacar una buena foto de tu mascota?

4- Trabas a la reparación: En el caso de los teléfonos por ejemplo, lo de impedir sacar la batería (con la excusa de hacer los teléfonos más delgados) es una forma de obligar a los consumidores a recurrir a los servicios oficiales y a disuadirlos de reemplazarlas por sustitutos más económicos. Otras tácticas son la utilización de piezas no estándar o que necesitan herramientas específicas para la reparación. Muchas veces se suele restringir el acceso a estas piezas o hacer una reducida producción de las mismas para aumentar artificialmente el costo.

Ejemplos de Obsolescencia Programada

- iPhone cada vez más lentos: La Justicia francesa comprobó que actualizaciones de Software hacían cada vez más lento el rendimiento de los modelos más viejos. La empresa le echó la culpa a las baterías, pero pagó una compensación de decenas de millones de dólares. Además rebajó los precios de sus baterías de repuesto para que los teléfonos fueran más rápidos con el nuevo Software y se comprometió a hacer más en el futuro para garantizar que los teléfonos no volvieran a ser más lentos. Con la salida de un nuevo modelo de teléfono cada año, seguro que hay algo de obsolescencia planificada en alguna parte.

- Impresoras: Esto es algo que todos conocemos. Muchas veces nos encontramos con impresoras a precio rebajado, pero al momento de tener que comprar un cartucho de tinta nos encontramos con que este tiene un precio igual o superior a comprar una nueva. Además, se ponen restricciones a la recarga o al uso de cartuchos alternativos. Hubo denuncias de que algunos modelos dejaban de funcionar a partir de cierta cantidad de páginas impresas o cierto tiempo desde la primera impresión.

- Certificados de seguridad: Por ejemplo, el pasado 30 de septiembre de 2021 caduco otro certificado de autentificación (CA de DST Root CA X3 de Let's Encrypt) que ayudaba a validar la conexión en internet a

los dispositivos que no fueron actualizados a otro certificado más actual -en la mayoría de los casos por no ser del interés económico de sus creadores-. Esto ocasionó que millones de dispositivos (teléfonos inteligentes, Smart TV, tabletas, computadoras portátiles y de escritorio, etc.) con algunos años de ser creados y perfectamente funcionales dejarán de conectarse a internet de un día para otro, forzando a sus dueños a desechar el dispositivo por carecer del servicio de internet en las aplicaciones instaladas.

- Cambio de la versión del sistema operativo: En el caso del sistema operativo Windows 10 a 11, la solicitud de requisitos mínimos de Hardware es para muchos equipos excesivo, ya que se estima que dejará fuera en su actualización a casi todos los equipos con más de 4 años de antigüedad por no contar por ejemplo con el Chip TPM 2.0 o GPU compatible con DirectX 12, siendo perfectamente funcionales con la versión actual del sistema operativo. Si bien Windows 10 seguirá con soporte hasta 2025, los usuarios que deseen tener las nuevas características del sistema operativo tendrán que cambiar de equipo.

9.1 Software Propietario

No existe consenso sobre el término a utilizar para referirse al opuesto del Software libre. La expresión «Software propietario (Proprietary Software)» (véase [11]), en la lengua anglosajona, "Proprietary" significa «poseído o controlado privadamente (Privately Owned and Controlled)», que destaca la manutención de la reserva de derechos sobre el uso, modificación o redistribución del Software. Inicialmente utilizado, pero con el inconveniente de que la acepción proviene de una traducción literal del inglés, no correspondiendo su uso como adjetivo en el español, de manera que puede ser considerado como un barbarismo.

El término "propietario" en español resultaría inadecuado, pues significa que «tiene derecho de propiedad sobre una cosa», por lo que no podría calificarse de "propietario" al Software, porque éste no tiene propiedad sobre nada (es decir, no es dueño de nada) y además, no podría serlo (porque es una cosa y no una persona). Así mismo, la expresión "Software propietario" podría ser interpretada como: "Software sujeto a propiedad" (derechos o titularidad) y su opuesto, el Software libre, también está sujeto al derecho de autor. Otra interpretación es que contrariamente al uso popular del término, se puede

afirmar que "todo Software es propietario", por lo que la forma correcta de referirse al Software con restricciones de uso, estudio, copia o mejora es la de Software privativo, según esta interpretación el término "propietario" podría aplicarse tanto para Software libre como Software privativo, ya que la diferencia entre uno y otro está en que el dueño del Software privativo lo licencia como propiedad privada y el de Software libre como propiedad social.

Con la intención de corregir el defecto de la expresión "Software propietario" aparece el llamado "Software con propietario", sin embargo se argumenta contra el término "con propietario" y justamente su similitud con Proprietary en inglés, que sólo haría referencia a un aspecto del Software que no es libre, manteniendo una de las principales críticas a éste (de "Software sujeto a derechos" o "propiedad"). Adicionalmente, si "propietario" se refiere al titular de los derechos de autor -y está claro que no se puede referir al usuario, en tanto éste es simplemente un cesionario-, no resuelve la contradicción: todo el Software libre tiene también titulares de derechos de autor.

La expresión Software no libre (en inglés Non-Free Software) es usado por la FSF para agrupar todo el Software que no es libre, es decir, incluye al llamado en inglés "Semi-Free Software" (Software semilibre) y al "Propietary Software". Asimismo, es frecuentemente utilizado para referirse al Software que no cumple con las Directrices de Software libre de Debian GNU/Linux, las cuales siguen la misma idea básica de libertad en el Software, propugnada por la FSF y sobre las cuales está basada la definición de código abierto de la Open Source Initiative.

Adicionalmente el Software de código cerrado nace como antónimo de Software de código abierto y por lo tanto se centra más en el aspecto de ausencia de acceso al código que en los derechos sobre el mismo, éste se refiere sólo a la ausencia de una sola libertad por lo que su uso debe enfocarse sólo a este tipo de Software y aunque siempre signifique que es un Software que no es libre, no tiene que ser Software de código cerrado.

La expresión Software privado es usada por la relación entre los conceptos de tener y ser privado. Este término sería inadecuado debido a que, en una de sus acepciones, la palabra "privado" se entiende como antónimo de "público", es decir, que «no es de propiedad pública o estatal, sino que pertenece a particulares», provocando que esta categoría se interpretará como no referente al Estado, lo que produciría la exclusión del Software no libre generado por el aparato estatal. Además, el "Software público" se asocia generalmente con Software de dominio público.

9.2 Software Libre

La definición de Software libre (véase [13], [2], [9], [10], [8] y [12]) estipula los criterios que se tienen que cumplir para que un programa sea considerado libre. De vez en cuando se modifica esta definición para clarificarla o para resolver problemas sobre cuestiones delicadas. «Software libre» significa que el Software respeta la libertad de los usuarios y la comunidad. En términos generales, los usuarios tienen la libertad de copiar, distribuir, estudiar, modificar y mejorar el Software. Con estas libertades, los usuarios -tanto individualmente como en forma colectiva- controlan el programa y lo que hace.

Cuando los usuarios no controlan el programa, el programa controla a los usuarios. Los programadores controlan el programa y a través del programa, controlan a los usuarios. Un programa que no es libre, llamado «privativo o propietario», es considerado por muchos como un instrumento de poder injusto.

El Software libre es la denominación del Software que respeta la libertad de todos los usuarios que adquirieron el producto y por tanto, una vez obtenido el mismo puede ser usado, copiado, estudiado, modificado y redistribuido libremente de varias formas. Según la Free Software Foundation (véase [13]), el Software libre se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir y estudiar el mismo, e incluso modificar el Software y distribuirlo modificado.

Un programa es Software libre si los usuarios tienen las cuatro libertades esenciales:

0. La libertad de usar el programa, con cualquier propósito.

1. La libertad de estudiar cómo funciona el programa y modificarlo, adaptándolo a tus necesidades.

2. La libertad de distribuir copias del programa, con lo cual puedes ayudar a tu prójimo.

3. La libertad de mejorar el programa y hacer públicas esas mejoras a los demás, de modo que toda la comunidad se beneficie.

Un programa es Software libre si los usuarios tienen todas esas libertades. Por tanto, el usuario debe ser libre de redistribuir copias, tanto con o sin modificaciones, ya sea gratuitamente o cobrando una tarifa por la distribución,

a cualquiera en cualquier parte. El ser libre de hacer estas cosas significa, entre otras cosas, que no tiene que pedir ni pagar el permiso.

También debe tener la libertad de hacer modificaciones y usarlas en privado para su propio trabajo o pasatiempo, sin siquiera mencionar que existen. Si publica sus cambios, no debe estar obligado a notificarlo a nadie en particular, ni de ninguna manera.

La libertad de ejecutar el programa significa que cualquier tipo de persona u organización es libre de usarlo en cualquier tipo de sistema de computación, para cualquier tipo de trabajo y finalidad, sin que exista obligación alguna de comunicarlo al programador ni a ninguna otra entidad específica. En esta libertad, lo que importa es el propósito de los usuarios, no el de los programadores. El usuario es libre de ejecutar el programa para alcanzar sus propósitos y si lo distribuye a otra persona, también esa persona será libre de ejecutarlo para lo que necesite; nadie tiene derecho a imponer sus propios objetivos.

La libertad de redistribuir copias debe incluir las formas binarias o ejecutables del programa, así como el código fuente, tanto para las versiones modificadas como para las que no lo estén. Distribuir programas en forma de ejecutables es necesario para que los sistemas operativos libres se puedan instalar fácilmente. Resulta aceptable si no existe un modo de producir un formato binario o ejecutable para un programa específico, dado que algunos lenguajes no incorporan esa característica, pero debe tener la libertad de redistribuir dichos formatos si encontrara o programara una forma de hacerlo.

Para que se de la libertad que se menciona en los puntos 1 y 3 de realizar cambios y publicar las versiones modificadas tenga sentido, el usuario debe tener acceso al código fuente del programa. Por consiguiente, el acceso al código fuente es una condición necesaria para el Software libre. El «código fuente» compilado no es código fuente real y no cuenta como código fuente.

La libertad 1 incluye la libertad de usar su versión modificada en lugar de la original. Si el programa se entrega con un producto diseñado para ejecutar versiones modificadas de terceros, pero rechaza ejecutar las suyas, una práctica conocida como «tivoización» o «arranque seguro» [«Lockdown»] la libertad 1 se convierte más en una ficción teórica que en una libertad práctica, esto no es suficiente, en otras palabras, estos binarios no son Software libre, incluso si se compilaron desde un código fuente que es libre.

Una manera importante de modificar el programa es agregándole subrutinas y módulos libres ya disponibles. Si la licencia del programa especifica que no se pueden añadir módulos que ya existen y que están bajo una licencia

apropiada, por ejemplo si requiere que usted sea el titular de los derechos de autor del código que desea añadir, entonces se trata de una licencia demasiado restrictiva como para considerarla libre.

La libertad 3 incluye la libertad de publicar sus versiones modificadas como Software libre. Una licencia libre también puede permitir otras formas de publicarlas; en otras palabras, no tiene que ser una licencia de Copyleft. No obstante, una licencia que requiera que las versiones modificadas no sean libres, no se puede considerar libre.

«Software libre» no significa que «no es comercial». Un programa libre debe estar disponible para el uso comercial, la programación comercial y la distribución comercial. La programación comercial de Software libre ya no es inusual; el Software libre comercial es muy importante, ejemplo de ello es la empresa RedHat (ahora propiedad de IBM). Puede haber pagado dinero para obtener copias de Software libre, o puede haber obtenido copias sin costo. Pero sin tener en cuenta cómo obtuvo sus copias, siempre tiene la libertad de copiar y modificar el Software, incluso de vender copias.

El término Software no libre se emplea para referirse al Software distribuido bajo una licencia de Software más restrictiva que no garantiza estas cuatro libertades. Las leyes de la propiedad intelectual reservan la mayoría de los derechos de modificación, duplicación y redistribución para el dueño del Copyright; el Software dispuesto bajo una licencia de Software libre rescinde específicamente la mayoría de estos derechos reservados.

Los manuales de Software deben ser libres por las mismas razones que el Software debe ser libre y porque de hecho los manuales son parte del Software. También tiene sentido aplicar los mismos argumentos a otros tipos de obras de uso práctico, es decir, obras que incorporen conocimiento útil, tal como publicaciones educativas y de referencia. Wikipedia es el ejemplo más conocido.

La lista de proyectos de este tipo es realmente impresionante, algunos han conseguido un uso y alta calidad, por ejemplo el compilador GCC, el Kernel de Linux y el sistema operativo Debian GNU/Linux y Android. Mientras que otros proyectos han caído en el olvido, pero de la gran mayoría se tiene copia del código fuente que permitiría a quienes estén interesados en dicho proyecto poder reusarlo y en su caso ampliarlo.

La característica más importante que aparece típicamente en un proyecto de este tipo, es que un conjunto de personas separadas a gran distancia, sean capaces, a través de la Web, de los E-mail y de foros de aunar sus esfuerzos para crear, mejorar y distribuir un producto, de forma que todos

ellos se benefician unos de otros. Evidentemente, la mayor parte del peso recae en los desarrolladores, pero también es necesaria una difusión para que los usuarios documenten, encuentren errores, hagan foros de discusión, etc.

Si bien, el Software libre no es más seguro (en el sentido de invulnerable) que el propietario, la diferencia estriba en que el código fuente en el Software libre está disponible para todos y cualquiera puede aportar una solución, y por lo general al poco tiempo de detectarse una vulnerabilidad (a veces en cuestión de horas) se puede disponer de una solución para la misma. Además, al tener acceso al código fuente se puede detectar fácilmente si alguien introdujo código malicioso a una determinada aplicación.

¿Por qué se Interesan los Autores, Alumnos y Profesores Universitarios en el Software Libre? La ventaja principal es porque bajo el Software libre subyace la idea de compartir conocimiento y favorecer la existencia de nuevas ideas[75]; y ¿qué es investigar y enseñar?, sino crear conocimiento y procurar que los alumnos aprendan e incluso vayan más allá de lo aprendido. Se comparte la idea, que el espíritu del Software libre es similar al que debería reinar en las instituciones educativas:

- Porque así no se condiciona a los estudiantes a usar siempre lo mismo.
- No se fomenta la piratería en los estudiantes y se evita pagar licencias que no son necesarias al existir alternativas gratuitas.
- Es mucho más seguro ya que el Software libre es público y se puede ver qué hace exactamente sin recelos.
- Se ofrece libertad de elección a los estudiantes y profesores al no limitarlos a usar una solución determinada, ampliando sus opciones y permitiendo un mayor aprendizaje.

Concretando estas ideas, profesores e investigadores necesitan herramientas para la investigación y docencia y estas deben tener una calidad mínima y ser fácilmente distribuibles entre los alumnos. En muchos casos las compañías desarrolladoras y distribuidoras de programas de cómputo no han

[75] ¿Por qué el Software creado con dinero de los impuestos no se publica como Software Libre?

¡El código pagado por los ciudadanos debería estar disponible para los ciudadanos y el mismo gobierno!

sabido ofrecer sus productos con la flexibilidad adecuada para las labores docentes o, en otros casos, los productos desarrollados no tienen la calidad esperada.

El Software libre es aún joven, pese a las decenas de miles de proyectos actuales -en los que se trabaja constantemente en mejorar la parte computacional de los algoritmos involucrados en el proyecto, haciendo y puliendo interfaces gráficas, generando ayuda en línea así como la documentación necesaria para que usuarios noveles y avanzados usen la mayor cantidad de opciones programadas- existen muchas otras necesidades profesionales y de investigación que requieren el desarrollo innovador de programas de cómputo para automatizarlas y hacerlas eficientes. Esto queda plasmado en las decenas de proyectos que a diario son registrados en las páginas especializadas en busca de difusión y apoyo para su proyecto.

En los últimos años, muchos proyectos han pasado de ser simples programas en línea de comandos a complejas aplicaciones multiplataforma -se ejecutan en distintos sistemas operativos como son Windows, Linux, Unix, Mac OS, Android- con ambientes gráficos multimedia que en muchos casos han superado a sus contrapartes comerciales -por ejemplo los navegadores Web-. Para muestra de este maravilloso avance, tomemos el proyecto del sistema operativo Android, que actualmente se ejecuta en millones de equipos -como celulares, tabletas, electrodomésticos, etc.- y en los cuales se pueden descargar miles de aplicaciones y está soportado por una gran cantidad de usuarios y empresas comerciales como Google, IBM y últimamente Microsoft -que años atrás era acérrima enemiga del Software libre-.

El Software libre ha logrado desplazar a muchos de sus competidores por sus múltiples bondades y bajo costo de desarrollo -es el caso de Windows Phone que fue reemplazado por Android de Google-, al reusar miles de aplicaciones ya existentes que usan Software libre y permitir desarrollar otro tanto de aplicaciones bajo una plataforma que se ejecuta en los más diversos procesadores. Además, el uso de Software libre y su posibilidad de ampliarlo y/o especializarlo según sea necesario, ha permitido crear de forma cada vez más rápida y confiable; para poner a disposición de un gran público programas de uso común, así como especializado que satisfagan las nuevas necesidades de los usuarios.

Software Libre en Ciencia y Educación Algunos puntos y reflexiones sobre porqué se considera que es interesante el Software libre en Ciencia y

Educación son:

- Accesible a todo el mundo aunque no sea rentable su desarrollo: El Software libre entre sus libertades permite que se pueda ejecutar por terceros, copiarlo, distribuirlo y estudiarlo/modificarlo. Eso hace que si en ciencia se usa Software libre el acceso a esos programas no suponga una barrera (se puede distribuir y ejecutar).

- Muchos de los desarrollos en el campo de la accesibilidad se realizan en universidades y son distribuidos como Software libre: Aunque no sea rentable muchas veces el desarrollo de herramientas de accesibilidad (no sea algo monetizable) en la universidad se consigue escapar a esa la lógica capitalista de solamente invertir en lo que pueda ofrecer beneficio económico.

- Transparencia: En ciencia es importante ver las costuras para comprobar si es verdad lo que se afirma, tener acceso al código fuente del Software empleado permite poder estudiarlo por si realiza algún cálculo mal.

- Propicia el espíritu crítico: Si no tienes acceso a las revistas o el acceso es privativo para los bolsillos de mucha gente no puedes comprobar la información. Se nos pide que seamos críticos con la información que se nos da del mundo científico pero no podemos entrenar el espíritu crítico sin acceso al conocimiento libre.

- Caramelos con droga en la puerta del colegio: Muchas empresas buscan introducir en los colegios, institutos y universidad su Software. Ofrecer un programa que permita trabajar y genere una dependencia quedando los datos muchas veces en las nubes (ordenadores de otras personas). Un ejemplo es Microsoft con Office 365. Dando cuentas gratuitas durante un tiempo para que se use su Software. Otro ejemplo podría ser Unity3D en vez de por ejemplo Godot.

- Especificaciones de protocolos abiertas VS cerradas: Gracias a que los protocolos TCP/IP, HTTP, POP, SNMP, DHCP, etc. son abiertos es posible construir herramientas por cualquier con conocimientos de programación. Con protocolos cerrados solamente quienes tuvieran acceso a las especificaciones podrían desarrollar y conocer cómo funcionan.

- Uso de estándares: Existiendo un estándar para documentos ofimáticos (procesador de textos, hoja de cálculo, presentaciones, etc.) algunas empresas como Microsoft se empeñan en ir con su propio formato y estándar en vez de sumarse a que sea más sencillo ir a una y que el usuario pueda optar por que herramientas usar para editar o trabajar con documentos ofimáticos.

- Software libre para la Ciencia ciudadana: Un ejemplo en el que es importante la colaboración ciudadana es el cambio climático y la defensa medioambiental del territorio. Desde imvec.tech usan herramientas de Software libre para medición y monitorización de contaminación.

Software Libre: Beneficios Más Allá de la Informática El uso de las tecnologías de código abierto supone cultivar el conocimiento y la puesta en valor de la libertad individual, lo personal y lo privado, todo ello sin menospreciar lo público y la construcción de una sociedad. El movimiento del Software libre, con Linux a la cabeza, ha capitaneado durante décadas planteamientos para un cambio en el modo de producción: el abaratamiento de costes empresariales para grandes y pequeños, el trabajo en línea, el desarrollo de Software y Hardware a pequeña escala, el replanteamiento del negocio informático, el sistema de normas éticas que rigen los grupos, la documentación abierta, etc.

Todo ello derivado de una simple idea: la libertad. Ha sido la clave que ha llevado a todo este movimiento hacia una autonomía y motivación que pocas veces se ve en otros sectores. El Open Source está lleno de alternativas con la libertad como pilar y consecuencia filosófica, de hecho muchos Forks (derivaciones) de proyectos aparecen cuando la disputa sobre la misma toma relevancia. Esta manera de hacer las cosas debería trasladarse directamente a la sociedad promoviendo esos valores para evitar caer en un mundo despótico que toma fuerza a pasos agigantados.

No estaría mal promover la comprensión de las licencias libres. Leer y entender una licencia GPL, MIT o BSD es infinitamente más sencillo y rápido que hacerlo con otras, siendo unas normas fáciles de cumplir porque encajan con un modelo ético y práctico comprensible por muchos.

Podríamos decir que GPL, BSD y MIT son las Constituciones que vertebran todo el movimiento del Software Libre, cosechando derechos de uso y logros como el ahorro o la legítima copia privada. Lo mismo podría ocurrir con las licencias Creative Commons para cierto contenido, un arma poderosa

en el sector divulgativo que está poco extendida por desconocimiento y el Status Quo de la propiedad intelectual.

La cooperación libre y voluntaria supuso un éxito hasta ahora pero está siendo amenazada por la dinámica actual de la Web, donde la centralización de las principales plataformas supone estar bajo el yugo de normas que rastrean con lupa y cambian sus términos con demasiada frecuencia. Ya ni digamos cuando eso se junta con el ansia de monetización: si quieres dinero más te vale no cabrear a los anunciantes o no tener un Strike por uso de contenido que podría ser reclamado.

Los claroscuros de este sistema hace que los creadores cada vez hagan menos de forma libre y altruista, y ello podría verse potenciado por las búsquedas con inteligencia artificial, lo que apunta a un empobrecimiento del contenido cultural fresco y renovador. Las polémicas con GitHub Copilot o ChatGPT sobre el entrenamiento de las IAs hace sobrevolar una vez más la cuestión del Copyright y el uso legítimo de los resultados brindados por éstas, lo que también condiciona la creación.

La libertad de expresión, de uso, de creación, de modificación ... esas cuestiones llevan irremediablemente a una libertad de pensamiento y acción, a una adaptación creativa que puede ser la motivación para romper moldes en todos los aspectos. El código abierto y sus licencias son, por tanto, un beneficio personal y social mucho más grande que el simple hecho de usar Linux o Software libre. El contenido despreocupado, que no prioriza el dinero y el posicionamiento/visualizaciones, se vuelve esencial para el inconformismo.

9.3 Seguridad del Software

Si bien, el Software Libre no es más seguro (en el sentido de invulnerable) que el propietario, la diferencia puede estribar en que el código fuente en el Software libre está disponible para todos y cualquiera puede aportar una solución y por lo general al poco tiempo de detectarse una vulnerabilidad (a veces en cuestión de horas) se puede disponer de una solución para la misma. Además, al tener acceso al código fuente se puede detectar si alguien introdujo código malicioso a una determinada aplicación.

Pero de todos es sabido, que los usuarios de Software de código abierto, como por ejemplo los que de manera habitual trabajan con equipos comandados por sistemas Linux, por regla general se sienten orgullosos de la seguridad que estos programas aportan con respecto a los sistemas cerrados propios de otras firmas, dígase Microsoft Windows o Mac de Apple.

¿Es Seguro el Software Libre? En primer lugar definiremos el concepto de "seguridad" como salvaguarda de las propiedades básicas de la información. Entre las características que debe cumplir para ser seguro, encontramos la integridad, es decir, que sólo los usuarios autorizados pueden crear y modificar los componentes del sistema, la confidencialidad, sólo estos usuarios pueden acceder a esos componentes, la disponibilidad, que todos los componentes estén a disposición de los usuarios siempre que lo deseen y el "no repudio", o lo que es lo mismo, la aceptación de un protocolo de comunicación entre el servidor y un cliente, por ejemplo, mediante certificados digitales.

Entre las diferencias de seguridad entre un Software Libre y el Software Propietario, podemos destacar:

- Seguridad en el Software Propietario: En el caso de tener "agujeros de seguridad", puede que no nos demos cuenta y que no podamos repararlos. Existe una dependencia del fabricante, retrasándose así cualquier reparación y la falsa creencia de que es más seguro por ser oscuro (la seguridad por oscuridad determina los fallos de seguridad no parcheados en cada producto).

- Seguridad en el Software Libre: Por su carácter público y su crecimiento progresivo, se van añadiendo funciones y se nos permite detectar más fácilmente los agujeros de seguridad para poder corregirlos. Los problemas tardan mucho menos en ser resueltos por el apoyo que tiene de los Hackers y una gran comunidad de desarrolladores y al ser un Software de código libre, cualquier empresa puede aportar soporte técnico.

Sin embargo esta es una pregunta sobre la que los expertos al día de hoy, tras muchos años de discusiones, siguen sin ponerse de acuerdo. ¿Es más seguro el Software de código abierto que los programas cerrados, o viceversa? Lo cierto es que, en términos generales, ambos bandos tienen sus razones con las que defender sus argumentos. Por un lado, los usuarios de las aplicaciones y sistemas de código abierto, defienden que, al estar el código fuente disponible a los ojos de todo el mundo, es mucho más fácil localizar posibles agujeros de seguridad y vulnerabilidades que pongan en peligro los datos de los usuarios.

Por otro lado, aquellos que consideran que los sistemas cerrados son más seguros en este sentido, afirman que al tener acceso tan solo los expertos al código fuente de sus aplicaciones, es más complicado que se produzcan

filtraciones o inserciones de Software malicioso en este tipo de sistemas. Hay que tener en cuenta que, por ejemplo, Google premia a las personas que descubren fallos de seguridad en su Software como Chrome, aunque no es el único gigante de la tecnología en utilizar estas tácticas.

De hecho muchas empresas están gastando miles de millones de dólares y/o euros en hacer que sus propuestas sean lo más seguras posible, argumentando que la seguridad de sus proyectos es una de sus prioridades, todo con el fin de intentar frenar que los atacantes vulneren sus sistemas. Por otro lado, otros aseguran que cuando el código fuente es público, más ojos están disponibles para detectar posibles vulnerabilidades o errores en dicho código, por lo que siempre será más rápido y sencillo poner soluciones con el fin de ganar en seguridad.

Sea como sea, en cualquiera de los dos casos, lo que ha quedado más que demostrado es que la seguridad no está garantizada en ningún momento, ya sean propuestas de código abierto, o no. Pero también es cierto que lo que se procura es que los riesgos de ser atacados se reduzcan en medida de lo posible. Los sistemas Linux son considerados desde hace mucho tiempo como un sistema operativo seguro, en buena parte debido a las ventajas que ofrece su diseño. Dado que su código está abierto, son muchas las personas que incorporan mejoras de las que el resto de usuarios de Linux se benefician, a diferencia de las propuestas de Windows o MacOS, donde estas correcciones generalmente se limitan a las que detectan Microsoft y Apple.

No obstante, en nuestra defensa del Software libre, diremos que su código abierto permite que los errores sean encontrados y solucionados con mayor rapidez, por lo que determinamos que es el Software más recomendable.

En general, puede afirmarse que el Software libre es más seguro, ya que debido a su carácter abierto y distribuido, un gran número de programadores y personas expertas pueden estar atentas al código fuente -especialmente en los grandes proyectos-, lo cual permite hacer auditorías con objeto de detectar errores y puertas traseras (Backdoor, en inglés) que pongan en riesgo nuestros datos.

Así, los grandes programas y proyectos de Software libre, con una extensa comunidad de desarrollo y usuarios que lo respalden, presentan niveles muy altos de seguridad, un alto grado de protección y una rápida respuesta a posibles vulnerabilidades.

9.4 Tipos de Licencias

Tanto la Open Source Initiative como la Free Software Foundation mantienen en sus páginas Web (véase [13], [2], y [12]) listados oficiales de las licencias de Software libre que aprueban.

Una licencia es aquella autorización formal con carácter contractual que un autor de un Software da a un interesado para ejercer "actos de explotación legales". Pueden existir tantas licencias como acuerdos concretos se den entre el autor y el licenciatario. Desde el punto de vista del Software libre, existen distintas variantes del concepto o grupos de licencias:

Licencias GPL Una de las más utilizadas es la Licencia Pública General de GNU (GNU GPL). El autor conserva los derechos de autoría (Copyright), y permite la redistribución y modificación bajo términos diseñados para asegurarse de que todas las versiones modificadas del Software permanecen bajo los términos más restrictivos de la propia GNU GPL. Esto hace que sea imposible crear un producto con partes no licenciadas GPL, el conjunto tiene que ser GPL.

En la práctica, esto hace que las licencias de Software libre se dividan en dos grandes grupos, aquellas que pueden ser mezcladas con código licenciado bajo GNU GPL (y que inevitablemente desaparecerán en el proceso, al ser el código resultante licenciado bajo GNU GPL) y las que no lo permiten al incluir mayores u otros requisitos que no contemplan ni admiten la GNU GPL y que por lo tanto no pueden ser enlazadas ni mezcladas con código gobernado por la licencia GNU GPL.

GPL Versión 1 la versión 1 de GNU GPL, fue presentada el 25 de febrero de 1989, impidió lo que eran las dos principales formas con las que los distribuidores de Software restringían las libertades definidas por el Software libre. El primer problema fue que los distribuidores publicaban únicamente los archivos binarios, funcionales y ejecutables, pero no entendibles o modificables por humanos. Para prevenir esto, la GPLv1 estableció que cualquier proveedor de Software libre además de distribuir el archivo binario debía liberar a su vez código fuente entendible y que pudiera ser modificado por el ser humano bajo la misma licencia (secciones 3a y 3b de la licencia).

El segundo problema era que los distribuidores podían añadir restricciones adicionales, añadiendo restricciones a la licencia o mediante la combinación del Software con otro que tuviera otras restricciones en su distribución. Si

esto se hacía, entonces la unión de los dos conjuntos de restricciones sería aplicada al trabajo combinado, entonces podrían añadirse restricciones inaceptables. Para prevenir esto, GPLv1 obligaba a que las versiones modificadas en su conjunto, tuvieran que ser distribuidas bajo los términos GPLv1 (secciones 2b y 4 de la licencia). Por lo tanto, el Software distribuido bajo GPLv1 puede ser combinado con Software bajo términos más permisivos y no con Software con licencias más restrictivas, lo que entraría en conflicto con el requisito de que todo Software tiene que ser distribuido bajo los términos de la GPLv1.

GPL Versión 2 según Richard Stallman, el mayor cambio en GPLv2 fue la cláusula "Liberty or Death" («libertad o muerte»). Esta sección dice que si alguien impone restricciones que le prohíben distribuir código GPL de tal forma que influya en las libertades de los usuarios (por ejemplo, si una ley impone que esa persona únicamente pueda distribuir el Software en binario), esa persona no puede distribuir Software GPL. La esperanza es que esto hará que sea menos tentador para las empresas el recurrir a las amenazas de patentes para exigir una remuneración de los desarrolladores de Software libre.

En 1991 se hizo evidente que una licencia menos restrictiva sería estratégicamente útil para la biblioteca C y para las bibliotecas de Software que esencialmente hacían el trabajo que llevaban a cabo otras bibliotecas comerciales ya existentes. Cuando la versión 2 de GPL fue liberada en junio de 1991, una segunda licencia Library General Public License fue introducida al mismo tiempo y numerada con la versión 2 para denotar que ambas son complementarias. Los números de versiones divergieron en 1999 cuando la versión 2.1 de LGPL fue liberada, esta fue renombrada como GNU Lesser General Public License para reflejar su lugar en esta filosofía.

GPL Versión 3 A finales de 2005, la Free Software Foundation (FSF) anunció estar trabajando en la versión 3 de la GPL (GPLv3). El 16 de enero de 2006, el primer borrador de GPLv3 fue publicado y se inició la consulta pública. La consulta pública se planeó originalmente para durar de nueve a quince meses, pero finalmente se extendió a dieciocho meses, durante los cuales se publicaron cuatro borradores. La GPLv3 oficial fue liberada por la FSF el 29 de junio de 2007.

Según Stallman los cambios más importantes se produjeron en el campo

de las patentes de Software, la compatibilidad de licencias de Software libre, la definición de código fuente y restricciones a las modificaciones de Hardware. Otros cambios están relacionados con la internacionalización, cómo son manejadas las violaciones de licencias y cómo los permisos adicionales pueden ser concedidos por el titular de los derechos de autor. También añade disposiciones para quitar al DRM su valor legal, por lo que es posible romper el DRM (Digital Rights Management) en el Software de GPL sin romper leyes como la DMCA (Digital Millennium Copyright Act).

GPLv2 vs GPL v3 GPLv3 contiene la intención básica de GPLv2 y es una licencia de código abierto con un Copyleft estricto. Sin embargo, el idioma del texto de la licencia fue fuertemente modificado y es mucho más completo en respuesta a los cambios técnicos, legales y al intercambio internacional de licencias.

La nueva versión de la licencia contiene una serie de cláusulas que abordan preguntas que no fueron o fueron cubiertas de manera insuficiente en la versión 2 de la GPL. Las nuevas regulaciones más importantes son las siguientes:

- GPLv3 contiene normas de compatibilidad que hacen que sea más fácil combinar el código GPL con el código que se publicó bajo diferentes licencias. Esto se refiere en particular al código bajo la licencia de Apache v. 2.0.

- Se insertaron normas sobre gestión de derechos digitales para evitar que el Software GPL se modifique a voluntad, ya que los usuarios recurrieron a las disposiciones legales para protegerse mediante medidas técnicas de protección (como la DMCA o la directiva sobre derechos de autor).

- La licencia GPLv3 contiene una licencia de patente explícita, según la cual las personas que licencian un programa bajo licencia GPL otorgan derechos de autor y patentes, en la medida en que esto sea necesario para utilizar el código que ellos otorgan. Por lo tanto, no se concede una licencia de patente completa. Además, la nueva cláusula de patente intenta proteger al usuario de las consecuencias de los acuerdos entre los titulares de patentes y los licenciatarios de la licencia pública general que solo benefician a algunos de los licenciatarios (correspondientes al

acuerdo Microsoft / Novell). Los licenciatarios deben garantizar que todos los usuarios disfrutan de tales ventajas (licencia de patente o liberación de reclamos) o que nadie puede beneficiarse de ellos.

- A diferencia de la GPLv2, la GPLv3 establece claramente que no es necesario divulgar el código fuente en un uso ASP (Application Service Provider) de los programas GPL, siempre que no se envíe una copia del Software al cliente. Si el efecto Copyleft debe extenderse al uso de ASP, debe aplicarse la Licencia pública general de Affero, versión 3 (AGPL) que solo difiere de la GPLv3 en esta consideración.

Licencias Estilo BSD Llamadas así porque se utilizan en gran cantidad de Software distribuido junto a los sistemas operativos BSD. El autor, bajo tales licencias, mantiene la protección de Copyright únicamente para la renuncia de garantía y para requerir la adecuada atribución de la autoría en trabajos derivados, pero permite la libre redistribución y modificación, incluso si dichos trabajos tienen propietario. Son muy permisivas, tanto que son fácilmente absorbidas al ser mezcladas con la licencia GNU GPL con quienes son compatibles. Puede argumentarse que esta licencia asegura "verdadero" Software libre, en el sentido que el usuario tiene libertad ilimitada con respecto al Software y que puede decidir incluso redistribuirlo como no libre.

Licencia Copyleft Hay que hacer constar que el titular de los derechos de autor (Copyright) de un Software bajo licencia Copyleft puede también realizar una versión modificada bajo su Copyright original y venderla bajo cualquier licencia que desee, además de distribuir la versión original como Software libre. Esta técnica ha sido usada como un modelo de negocio por una serie de empresas que realizan Software libre (por ejemplo MySQL); esta práctica no restringe ninguno de los derechos otorgados a los usuarios de la versión Copyleft.

Licencia estilo MIT Las licencias MIT son de las más permisivas, casi se consideran Software de dominio público. Lo único que requieren es incluir la licencia MIT para indicar que el Software incluye código con licencia MIT.

Licencia Apache License La licencia Apache trata de preservar los derechos de autor, incluir la licencia en el Software distribuido y una lista de

los cambios realizados. En modificaciones extensivas del Software original permite licenciar el Software bajo otra licencia sin incluir esas modificaciones en el código fuente.

Licencia Mozilla Public License MPL Esta licencia requiere que los archivos al ser distribuidos conserven la misma licencia original pero pueden ser usados junto con archivos con otra licencia, al contrario de la licencia GPL que requiere que todo el código usado junto con código GPL sea licenciado como código GPL. También en caso de hacer modificaciones extensivas permite distribuirlas bajo diferentes términos y sin incluir el código fuente en las modificaciones.

Licencia Código de Dominio Público Es un código que no está sujeto a derechos de autor que puede utilizarse sin restricciones.

Licencia Creative Commons Las licencias de Creative Commons son más utilizadas para cualquier creación digital que para el Software, entendiendo como creación digital desde fotos, artículos en blogs, música, vídeos, este trabajo, etc. Hay varios tipos de licencias de Creative Commons diferenciando entre permitir modificaciones a la obra original, solicitando crédito de la creación o permitiendo un uso comercial de la obra.

Licencias de Código Abierto Las licencias de código abierto son un intermedio entre las licencias privativas y las licencias de Software libre. Las licencias de código abierto permiten el acceso al código fuente pero no todas se consideran licencias de Software libre al no otorgar otros derechos que se requieren para considerar un Software como Software libre como el derecho al uso o con cualquier propósito, modificación y distribución.

Dado el éxito del Software libre como modelo de desarrollo de Software algunas empresas cuyo Software era privativo pueden decidir hacerlo de código abierto con la intención de suplir algunas carencias de Software privativo pero sin perder ciertos derechos que son la fuente de sus ingresos como la venta de licencias.

Las expresiones «Software libre» y «código abierto» se refieren casi al mismo conjunto de programas. No obstante, dicen cosas muy diferentes acerca de dichos programas, basándose en valores diferentes. El movimiento del Software libre defiende la libertad de los usuarios de ordenadores, en un

movimiento en pro de la libertad y la justicia. Por contra, la idea del código abierto valora principalmente las ventajas prácticas y no defiende principios. Esta es la razón por la que gran parte de la comunidad de Software libre está en desacuerdo con el movimiento del código abierto y nosotros no empleamos esta expresión en este texto.

Licencia Microsoft Public License La Microsoft Public License es una licencia de código abierto que permite la distribución del Software bajo la misma licencia y la modificación para un uso privado. Tiene restricciones en cuanto a las marcas registradas.

En caso de distribuir el Software de forma compilada o en forma de objeto binario no se exige proporcionar los derechos de acceso al código fuente del Software compilado o en forma de objeto binario. En este caso esta licencia no otorga más derechos de los que se reciben, pero si permite otorgar menos derechos al distribuir el Software (compilado o en forma de objeto binario).

Modelo de Desarrollo de Software Bazar y Catedral El tipo de licencia no determina qué Software es mejor o peor, si el privativo o el Software libre, la diferencia entre las licencias está en sus características éticas y legales. Aunque el modelo de desarrollo con una licencia de código abierto a la larga suele tener un mejor desarrollo y éxito que el Software privativo, más aún con un medio como internet que permite colaborar a cualquier persona independiente de donde esté ubicada en el mundo.

Comparación con el Software de Código Abierto Aunque en la práctica el Software de código abierto y el Software libre comparten muchas de sus licencias, la Free Software Foundation opina que el movimiento del Software de código abierto es filosóficamente diferente del movimiento del Software libre. Los defensores del término «código abierto (Open Source)», afirman que éste evita la ambigüedad del término en ese idioma que es «Free» en «Free Software».

Mucha gente reconoce el beneficio cualitativo del proceso de desarrollo de Software cuando los desarrolladores pueden usar, modificar y redistribuir el código fuente de un programa. El movimiento del Software libre hace especial énfasis en los aspectos morales o éticos del Software, viendo la excelencia técnica como un producto secundario de su estándar ético. El movimiento de código abierto ve la excelencia técnica como el objetivo prioritario, siendo

el compartir el código fuente un medio para dicho fin. Por dicho motivo, la FSF se distancia tanto del movimiento de código abierto como del término «código abierto (Open Source)».

Puesto que la «iniciativa de Software libre Open Source Initiative (OSI)» sólo aprueba las licencias que se ajustan a la «definición de código abierto (Open Source Definition)», la mayoría de la gente lo interpreta como un esquema de distribución, e intercambia libremente "código abierto" con "Software libre". Aún cuando existen importantes diferencias filosóficas entre ambos términos, especialmente en términos de las motivaciones para el desarrollo y el uso de tal Software, raramente suelen tener impacto en el proceso de colaboración.

Aunque el término "código abierto" elimina la ambigüedad de libertad frente a precio (en el caso del inglés), introduce una nueva: entre los programas que se ajustan a la definición de código abierto, que dan a los usuarios la libertad de mejorarlos y los programas que simplemente tienen el código fuente disponible, posiblemente con fuertes restricciones sobre el uso de dicho código fuente. Mucha gente cree que cualquier Software que tenga el código fuente disponible es de código abierto, puesto que lo pueden manipular, sin embargo, mucho de este Software no da a sus usuarios la libertad de distribuir sus modificaciones, restringe el uso comercial, o en general restringe los derechos de los usuarios.

9.4.1 Licencias Creative Commons

Las Licencias[76] Creative Commons (CC) de forma general no tienen una definición oficial, sin embargo, entre las muchas definiciones aceptadas están la de la UNESCO, la cual expresa la siguiente descripción:

Las Licencias Creative Commons (CC) son modelos de contratos que sirven para otorgar públicamente el derecho de utilizar una publicación protegida por los derechos de autor. Entre menos restricciones implique una licencia, mayores serán las posibilidades de utilizar y distribuir un contenido. Las Licencias CC permiten a cualquier usuario descargar, copiar, distribuir, traducir, reutilizar, adaptar y desarrollar su contenido sin costo alguno.

Sin embargo, en la Web oficial de la Organización Creative Commons se nos dice sobre las mismas lo siguiente:

[76] Las licencias de Creative Commons son más utilizadas para cualquier creación digital que para el Software, entendiendo como creación digital desde fotos, artículos en blogs, música, vídeos, este trabajo, etc.

Las Licencias Creative Commons (CC) brindan a todos, desde creadores individuales hasta grandes instituciones, una forma estandarizada de otorgar permiso al público para usar su trabajo creativo bajo la ley de derechos de autor. Desde la perspectiva del reutilizador, la presencia de una licencia Creative Commons sobre una obra protegida por derechos de autor responde a la pregunta: ¿Qué puedo hacer con esta obra?.

Las «Licencias Creative Commons» que hoy en día pertenecen a la organización mundial Creative Commons[77], y buscan regularizar y mantener, de forma equilibrada y satisfactoria, todo lo relacionado con el derecho de utilizar una publicación protegida por los derechos de autor a nivel mundial, han logrado un buen trabajo, sin duda alguna. Y seguramente en el tiempo, se irán adaptando a las nuevas realidades sociales y tecnológicas para poder seguir manteniendo de forma armónica las posibilidades de utilizar y distribuir cualquier contenido libre y abierto sobre la Internet, y más allá.

¿Cuáles son y cómo funcionan o para qué se usan? Las 7 distintas Licencias Creative Commons son las siguientes:

CC BY Esta licencia permite a los reutilizadores distribuir, remezclar, adaptar y desarrollar el material en cualquier medio o formato, siempre que se otorgue la atribución al creador. La licencia permite el uso comercial.

CC BY-SA Esta licencia permite a los reutilizadores distribuir, remezclar, adaptar y desarrollar el material en cualquier medio o formato, siempre que se otorgue la atribución al creador. La licencia permite el uso comercial. Si remezcla, adapta o construye sobre el material, debe licenciar el material modificado bajo términos idénticos.

CC BY-NC Esta licencia permite a los reutilizadores distribuir, remezclar, adaptar y desarrollar el material en cualquier medio o formato, únicamente con fines no comerciales y siempre que se otorgue la atribución al creador.

[77] Una organización mundial sin ánimo de lucro que permite compartir y reutilizar la creatividad y el conocimiento mediante el suministro de herramientas legales gratuitas. Y cuyas herramientas legales (licencias) ayudan a quienes quieren fomentar la reutilización de sus obras ofreciéndolas para su uso bajo términos generosos y estandarizados; a quienes quieren hacer usos creativos de las obras; y a quienes quieren beneficiarse de esta simbiosis.

CC BY-NC-SA Esta licencia permite a los reutilizadores distribuir, remezclar, adaptar y desarrollar el material en cualquier medio o formato, únicamente con fines no comerciales y siempre que se otorgue la atribución al creador. Si remezcla, adapta o construye sobre el material, debe licenciar el material modificado bajo términos idénticos.

CC BY-ND Esta licencia permite a los reutilizadores copiar y distribuir el material en cualquier medio o formato únicamente en forma no adaptada, y siempre y cuando se otorgue la atribución al creador. La licencia permite el uso comercial.

CC BY-NC-ND Esta licencia permite a los reutilizadores copiar y distribuir el material en cualquier medio o formato únicamente en forma no adaptada, únicamente con fines no comerciales y siempre que se otorgue la atribución al creador.

CC0 (CC Cero) Esta licencia es una herramienta de dedicación pública que permite a los creadores renunciar a sus derechos de autor y poner sus obras en el dominio público mundial. CC0 permite a los reutilizadores distribuir, remezclar, adaptar y desarrollar el material en cualquier medio o formato, sin condiciones.

9.4.2 Nuevas Licencias para Responder a Nuevas Necesidades

El mundo de la tecnología avanza mucho más rápido que las leyes y estas tienen que esforzarse para alcanzarlo. En el caso del Software libre y de código abierto, tanto la Free Software Foundation como la Open Source Initiative (los organismos encargados de regular las diferentes licencias) enfrentan periódicamente el problema de cómo mantener sus principios y al mismo tiempo evitar que alguien se aproveche indebidamente.

En el último tiempo, la Open Source Initiative le dio el sello de aprobación a otras nuevas licencias para propósitos específicos.

Nuevas Licencias de Código Abierto

- Cryptographic Autonomy License version 1.0 (CAL-1.0)

Fue creada en el 2019 por el equipo del proyecto de código abierto Holochain, esta licencia fue desarrollada para ser utilizada con aplicaciones criptográficas distribuidas. El inconveniente con las licencias tradicionales es que no obligaba a compartir los datos. Esto podría perjudicar el funcionamiento de toda la red. Por eso la CAL también incluye la obligación de proporcionar a terceros los permisos y materiales necesarios para utilizar y modificar el Software de forma independiente sin que ese tercero tenga una pérdida de datos o capacidad.

- Open Hardware Licence (OHL)

De la mano de la Organización Europea para la Investigación Nuclear (CERN) llegó esta licencia con tres variantes enfocadas en la posibilidad de compartir libremente tanto Hardware como Software.

Hay que hacer una aclaración. La OSI fue creada en principio pensando en el Software por lo que no tiene mecanismos para la aprobación de licencias de Hardware. Pero, como la propuesta del CERN se refiere a ambos rubros, esto posibilitó la aprobación:

- CERN-OHL-S es una licencia fuertemente recíproca: El que utilice un diseño bajo esta licencia deberá poner a disposición las fuentes de sus modificaciones y agregados bajo la misma licencia.

- CERN-OHL-W es una licencia débilmente recíproca: Sólo obliga a distribuir las fuentes de la parte del diseño que fue puesta originalmente bajo ella. No así los agregados y modificaciones.

- CERN-OHL-P es una licencia permisiva: Permite a la gente tomar un proyecto, relicenciarlo y utilizarlo sin ninguna obligación de distribuir las fuentes.

Hay que decir que la gente del CERN parece haber encontrado la solución a un problema que viene afectando a algunos proyectos de código abierto. Una gran empresa utiliza ese proyecto para comercializar servicios y no solo hace ningún aporte al proyecto original (ya sea con código o dando apoyo financiero) si no que también compite en el mismo mercado.

Post Open Zero-Cost en el año 2024 se desarrolla la licencia «Post Open Zero-Cost» con la cual busca abordar los desafíos surgidos en la interacción entre desarrolladores de código abierto y empresas comerciales, especialmente en lo que respecta a la compensación justa por el uso comercial del código.

La característica distintiva de la licencia «Post-Open» en comparación con las licencias abiertas existentes, como la GPL, es la introducción de un componente contractual que puede ser rescindido en caso de violación de los términos. Esta licencia ofrece dos tipos de acuerdos contractuales: gratuitos y de pago. El acuerdo de pago permite negociar derechos adicionales para la distribución comercial de productos o modificaciones sin requerir su divulgación pública.

Las situaciones que podrían llevar a la terminación del acuerdo contractual incluyen: violación de los términos de la licencia; reclamaciones por infracción de patentes; imposición de condiciones adicionales (como sanciones en contratos con clientes por divulgación de información sensible); cambios sujetos a leyes de control de exportaciones; ocultamiento de información sobre vulnerabilidades; y uso del código para entrenamiento de modelos de aprendizaje automático bajo términos no permitidos. Las relaciones contractuales no se rescinden de inmediato, sino que se notifica la infracción y se otorgan 60 días para corregirla antes de la rescisión efectiva del acuerdo.

Uno de los problemas que la nueva licencia busca abordar está relacionado con las limitaciones de la GPL, la cual se centra en otorgar derechos sin la capacidad de revocarlos, lo que permite a las empresas encontrar formas de eludir sus requisitos, especialmente en lo que respecta al acceso al código fuente. Estas lagunas son utilizadas para restringir la disponibilidad del código subyacente en productos comerciales mediante la imposición de términos contractuales adicionales con los usuarios finales.

Un claro ejemplo es el de RHEL, la cual los clientes firman un acuerdo con Red Hat que limita la redistribución del código fuente al imponer condiciones sobre la coincidencia de las copias instaladas y compradas de RHEL. Esto coloca a los usuarios en la disyuntiva entre su libertad para disponer del software y mantener su estatus de cliente de Red Hat. Aunque la GPL permite la distribución de parches que solucionan vulnerabilidades en el código de RHEL, esto podría interpretarse como una violación del acuerdo con Red Hat y podría resultar en la terminación de los servicios por parte de la empresa.

9.5 Implicaciones Económico-Políticas del Software Libre

Una vez que un producto de Software libre ha empezado a circular, rápidamente está disponible a un costo muy bajo. Al mismo tiempo, su utilidad no decrece. El Software, en general, podría ser considerado un bien de uso inagotable, tomando en cuenta que su costo marginal es pequeñísimo y que no es un bien sujeto a rivalidad -la posesión del bien por un agente económico no impide que otro lo posea-.

9.5.1 Software Libre y la Piratería

Puesto que el Software libre permite el libre uso, modificación y redistribución, a menudo encuentra un hogar entre usuarios para los cuales el coste del Software no libre es a veces prohibitivo, o como alternativa a la piratería. También es sencillo modificarlo localmente, lo que permite que sean posibles los esfuerzos de traducción a idiomas que no son necesariamente rentables comercialmente, además:

- Porque así no se condiciona a los usuarios a usar siempre lo mismo.
- Porque así no se fomenta la piratería en los usuarios al no pagar licencias.
- Porque así no se obliga a usar una solución concreta y se ofrece libertad de elección a los usuarios.
- Porque es mucho más seguro ya que el Software libre es público y se puede ver qué hace exactamente sin recelos.

La mayoría del Software libre se produce por equipos internacionales que cooperan a través de la libre asociación. Los equipos están típicamente compuestos por individuos con una amplia variedad de motivaciones y pueden provenir tanto del sector privado, del sector voluntario o del sector público. En los últimos años se ha visto un incremento notable de grandes corporativos (como IBM, Microsoft, Intel, Google, Samsung, Red Hat, etc.) que han dedicado una creciente cantidad de recursos humanos y computacionales para desarrollar Software libre, ya que esto apoya a sus propios negocios.

9.5.2 ¿Cuánto Cuesta el Software Libre?

En esta sección intentaremos dar una idea de cuál es el costo del desarrollo del Software Libre, por supuesto que no se tratará más que de una conjetura aproximada basada en las cifras proporcionadas por desarrolladores de Software comercial (al año 2020).

Gratis no Significa Gratuito Supongamos que todos los recursos humanos participantes en el desarrollo de un proyecto de Software libre lo hagan de forma voluntaria. De todas formas tenemos lo que los contables llaman «Costo de oportunidad» esto es, los ingresos que podrían haber generado esas personas si hubieran dedicado el tiempo y los conocimientos invertidos en el proyecto a uno en el que les pagaran. Así, el calcular el costo promedio por hora que cobra un programador, por la cantidad de horas invertidas al proyecto, nos da un razonable costo mínimo. Lo mismo puede hacerse con los voluntarios dedicados a la difusión en las redes. El costo de una campaña de marketing digital puede estimarse fácilmente.

Muchos proyectos de código abierto como una distribución Linux, son construidos a partir de la integración de otros proyectos, por los que sus costos de desarrollo también deberían sumarse.

Por otra parte, necesitamos recursos físicos. Aún cuando los voluntarios trabajen desde su casa, siguen teniendo que comprar y mantener sus equipos, además de pagar la electricidad que los hace funcionar.

Bases para el Cálculo Hay muchos factores que determinan el costo de desarrollar una pieza de Software. En un extremo tenemos una aplicación simple que requiere muy poca interacción del usuario o procesamiento del lado del servidor. Tal es el caso de un cliente de escritorio para redes sociales. Por el otro sistemas operativos que deben operar en múltiples plataformas realizando múltiples tareas (por ejemplo Debían que aspira a ser el sistema operativo universal). Sin embargo, el costo de una aplicación simple puede elevarse debido a que tiene múltiples pantallas diferentes. Por ejemplo un juego desarrollado con HTML5 y Javascript.

Los dos aspectos claves son la cantidad de horas de trabajo necesarias y las tecnologías involucradas. Para una aplicación de escritorio como un procesador de textos con las prestaciones habituales, optimizado para un determinado escritorio Linux, se estima que se tendría que contar con al menos el equivalente a 42,000 euros en trabajo voluntario. Un gestor de contenidos

para comercio electrónico con seguimiento de pedidos e integración con las principales plataformas de pago implicaría desembolsar unos 210,000 euros o su equivalente en trabajo voluntario.

Tomando en cuenta que este cálculo incluye lo que costó el desarrollo de las bibliotecas y otros proyectos libres y de código abierto incluidos, pero no los gastos que efectivamente deben desembolsarse en efectivo como la compra de equipos, Software de seguridad y desarrollo y el pago de electricidad e internet.

Por otro lado, el proceso de medición de costes del Software es un factor realmente importante en el análisis de un proyecto. Hay distintos métodos de estimación de costes de desarrollo de Software (también conocido como métrica del Software). La gran mayoría de estos métodos se basan en la medición del número de Líneas de Código que contiene el desarrollo (se excluyen comentarios y líneas en blanco de los fuentes).

Desarrollo de Fedora 9 La Linux Foundation ha calculado que costaría desarrollar el código de la distribución Fedora 9 que fue puesta a disposición del público el 13 de mayo de 2008, en el informe citado "Estimating the Total Development Cost of a Linux Distribution" se calcula que Fedora 9 tiene un valor de 10.8 mil millones de dólares y que el coste únicamente del Kernel (2.6.25 con 8,396,250 líneas de código) tendría un valor de 1.4 mil millones de dólares.

Esta distribución tiene unas 205 millones de líneas de código, el proyecto debería ser desarrollado por 1000 - 5000 desarrolladores (el trabajo invertido por una única persona desarrollándolo se alargaría durante unos 60.000 años) y esa estimación no va muy desencaminada ya que en los 2 últimos años del desarrollo de esa versión contribuyeron unos 3,200 desarrolladores aunque el número de trabajadores en la historia de la distribución es mucho mayor.

¿Qué pasa con GNU/Linux? en el año 2015 (las estadísticas más actuales que conseguimos) la Linux Foundation analizó el costo de desarrollo del núcleo. Combinando el aporte de los recursos humanos (voluntarios y de pago) y los desembolsos necesarios, la cuenta sumó 476,767,860,000.13 euros.

Todos sabemos que el hecho de tener desarrolladores asalariados no garantiza necesariamente Software de calidad. Pero, tener desarrolladores que pueden dedicar toda su atención a un proyecto en lugar de hacerlo en sus horas libres si lo hace. Lamentablemente, por el momento el único modo de

lograr eso es obtener el apoyo de corporaciones (Intel, Google, IBM, AMD, Sun Microsystems, Dell, Lenovo, Asus, HP, SGI, Oracle, RedHat, etc.) que solo lo hacen con los proyectos que son de su interés como el Kernel de Linux, hay que notar que para el Kernel de Linux un porcentaje importante (más del 10 %) lo hacen programadores independientes.

Costes Recordemos que la segunda de las cuatro libertades de un programa para ser Software libre es:

- Libre redistribución

y esta puede ser a través de un pago o sin costo. Es por ello que existen distintas empresas, organizaciones y usuarios que pueden apoyar a los usuarios finales en el desarrollo y soporte de algún programa de Software libre o una distribución personalizada de Linux por un costo determinado.

Mucha gente, en especial ejecutivos de empresas, se acercan a Linux bajo la promesa de que es una solución de bajo costo -muchos piensan que incluso es gratis-. Pero la realidad es que detrás de Linux y los programas de Software libre (y aquí la traducción correcta de la palabra inglesa, Free es libre, no gratis) pueden llevar una serie de costos ocultos que deben ser considerados al momento de decidir si se implementa una solución propietaria o una libre.

Los costos ocultos aparecen cuando se intenta instalar y capacitar en el uso de algún Software y se necesita la ayuda de un informático, al que se tiene que pagar, o alguna empresa quiere personalizar la interfaz de un programa y necesita la ayuda de un programador, que también tienen un coste, por lo que finalmente el comentario suele ser "el Software libre no es barato".

El primer punto a considerar al evaluar ambas alternativas es el costo de la licencia. Los productos de Software libre no suelen tener un costo de licencia asociado, que sí existe en los programas propietarios. De hecho es allí en donde los fabricantes de Software recargan sus costos de investigación y desarrollo, de producción e incluso sus ganancias. En este primer punto el ganador claro es el Software libre y es lo que los adeptos de este esquema publicitan: "su compañía puede ahorrar miles de dólares al año usando Software libre".

El segundo punto a considerar es el costo de instalación, configuración y capacitación. Dependiendo de su complejidad, algunos productos comerciales no contemplan costos extra por este concepto y otros -como Windows, por ejemplo- son tan populares que se puede encontrar numerosas opciones

de instalación -a través de empresas o profesionales- donde escoger en el mercado. A veces en el Software libre la configuración puede implicar recompilar el producto con algunas opciones particulares, algo que sólo pueden realizar técnicos con un nivel adecuado de conocimientos y que puede que no sean tan fáciles de encontrar. Aquí por lo general la ventaja va hacia el Software comercial.

Una vez instalado el Software, toca realizar actualizaciones de mantenimiento. Si bien es cierto lo que algunos de los fanáticos del Software libre dicen, que nadie lo obliga a mejorar el Software con que cuenta, la realidad -especialmente en lo que a seguridad se refiere- obliga a las empresas a mantener su Software actualizado. Aún así, los costos de actualizar Software libre suelen ser significativamente más bajos que los de productos comerciales y suelen ser menos exigentes con el Hardware necesario para ejecutarlos. La mayoría de las veces la ventaja es para el Software libre, pero hay que evaluar ya que varía dependiendo de cada caso.

Por último hay algunas casas que desarrollan productos de Software libre -dan el código y permiten que cualquiera lo modifique o reutilice- pero fijan contratos con cargos mensuales o anuales para mantenimiento del mismo, algo que se parece mucho a un cobro por licencia, por lo que hay que estar seguro de conocer todas las condiciones cuando una empresa nos ofrece una solución propia y la califica como Software libre.

Además de estas consideraciones hay una que debe sumarse eventualmente a esta evaluación: el costo de migrar a otra solución. En Software libre suelen usarse estándares abiertos para almacenar los datos, lo que facilita las migraciones. En cambio muchas soluciones propietarias suelen tener formatos propietarios que pueden dejar "amarrados" los datos de la empresa a una aplicación específica.

Sólo después de evaluar estos aspectos del Software, que pueden tener implicaciones importantes en el presupuesto, es que un CIO (Chief Information Officer) puede decir si una solución de Software libre le conviene más a una empresa o no, algo que va más allá de que la aplicación sea gratis o no.

9.5.3 La Nube y el Código Abierto

Desde hace años se han creado nuevos desafíos para el código abierto que plantea la nube, un término que para el usuario promedio puede significar cosas diferentes, pero que para la empresa se resume en servicios. Y es que los beneficios económicos que genera el mero Software de código abierto no

son comparables a los que se obtienen cuando se ofrece ese mismo Software a través de servicios, más allá -pero incluyendo- del soporte.

Este hecho diferencial lleva tiempo provocando fricciones entre desarrolladores y proveedores y hay quien adelantó incluso el fin del modelo de desarrollo del código abierto tal y como lo conocemos. ¿Quién tiene la razón?, ¿es para tanto la situación?.

En sus exposiciones, representantes de compañías y proyectos de código abierto muy populares en el ámbito empresarial, explican el supuesto perjuicio que les ocasiona el uso que los grandes proveedores de servicios en la nube hacen del Software que ellos desarrollan y cómo algunos han considerado y aplicado un enfoque más cerrado para sus productos con el fin de evitar lo que denominan como expolio. Hay declaraciones que merecen ser rescatadas para dotar de contexto a la discusión:

- El papel que juega el código abierto en la creación de oportunidades comerciales ha cambiado, durante muchos años les permitimos que las empresas de servicios tomaran lo que se ha desarrollado y ganasen dinero con ello.

- Empresas como Amazon Web Services, Azure de Microsoft, etc. Han ganado cientos de millones de dólares ofreciendo a sus clientes servicios basados en Software libre sin contribuir tanto a la comunidad de código abierto que construye y mantiene ese proyecto. Es imposible saber exactamente de cuánto dinero estamos hablando, pero es cierto que los proveedores de la nube se benefician del trabajo de los desarrolladores de código abierto que no emplean.

- Hay un mito ampliamente instalado en el mundo de código abierto que dice que los proyectos son impulsados por una comunidad de contribuyentes, pero en realidad, los desarrolladores pagados contribuyen con la mayor parte del código en la mayoría de los proyectos de código abierto modernos.

En resumen, todas estas voces se quejan de dos cosas: los grandes beneficios que obtienen los proveedores de servicios en la nube con su Software sin retribuirles en consecuencia, y la falta de colaboración manteniendo los productos con los que lucran. Sin embargo, no nos engañemos, el quid de la cuestión está principalmente en el dinero: la opinión generalizada de la

comunidad es que el Software de código abierto nunca fue pensado para que las empresas de servicios en la nube lo tomasen y lo vendieran.

Por otro lado, si es posible bifurcar un proyecto libre que se cierra, ¿no hubiese sido mejor colaborar con él antes y haber evitado el cierre? Si no se invierte y se mantiene con salud aquello que da beneficios, puede terminar por desaparecer. A medio camino entre el depredador y el parásito: así es como ven muchas desarrolladoras de código abierto a los proveedores de servicios en la nube.

Vale la pena retomar ahora la frase «el Software de código abierto nunca fue pensado para que las empresas de servicios en la nube lo tomasen y lo vendieran». ¿Para qué fue pensado el código abierto entonces? No hay ninguna licencia de código abierto o Software libre reconocida por la Free Software Foundation o la Open Source Initiative que prohíba hacer negocio con el Software. Lo que prohíben es la discriminación en la capacidad y alcance de su uso en función de la parte, se trate de un individuo o de la mayor multinacional imaginable.

¿Cuál es la solución a un embrollo de tamaña envergadura? Lo único claro es que no es una cuestión de blancos y negros y las consideraciones son demasiadas como para seguir ahondando: empresas que cotizan en bolsa quieren más dinero de otras compañías -que también cotizan en bolsa y por mucho más-, que además del Software ponen la infraestructura sobre la que distribuyen sus ofertas y que tienen la capacidad de clonar tu producto en un abrir y cerrar de ojos, no solo porque tienen el capital, sino porque tienen la experiencia necesaria tras contribuir técnica, pero también en muchos casos, económicamente, durante largo tiempo.

Pese a ello, esta situación está alterando el paradigma actual, en el que el modelo de desarrollo del código abierto se ha impuesto como impulsor de la innovación en el sector empresarial y ya hay quien habla de que nos acercamos al fin, o al principio del fin de la Era del Open Source, cuya preponderancia estaría sentenciada por la revolución de la nube, a la postre el mayor estímulo que haya tenido el código abierto hasta la fecha.

El futuro, pues, pasaría por el Shared Source Software, bajo el cual diferentes compañías con intereses alineados colaborarán en el desarrollo de proyectos concretos, pero limitando su explotación comercial a sí mismas. Todavía no estamos ahí, no obstante, y no parece tampoco que el relevo se vaya a dar en breve. De suceder, será muy llamativo: la muerte del código abierto por un éxito mal entendido.

9.5.4 El Código Abierto como Base de la Competitividad

En septiembre del 2021, se publicó un amplio y detallado informe llevado a cabo por el Fraunhofer ISI y por OpenForum Europe para la Comisión Europea, «The impact of open source software and hardware on technological independence, competitiveness and innovation in the EU economy», cuantifica la importancia económica del código abierto aplicado tanto al Software como al Hardware, su efecto en la contribución al producto interior bruto generado, la reducción en aspectos como el coste total de propiedad, dependencia del proveedor y autonomía digital; lanza una serie de recomendaciones específicas de políticas públicas destinadas a lograr un sector público digitalmente autónomo, una investigación y desarrollo abierto que fomente el crecimiento europeo y una industria más digitalizada y competitiva.

En las estimaciones del informe se apunta que las empresas europeas invirtieron alrededor de mil millones de euros en Software de código abierto en 2018, lo que resultó en un impacto en la economía europea de entre 65,000 y 95,000 millones de euros. El análisis estima una relación costo-beneficio superior a 1:4 y predice que un aumento del 10% de las contribuciones de a repositorios de código abierto sería susceptible de generar anualmente entre un 0.4% y un 0.6% adicional en el PIB, así como más de seiscientas nuevas empresas tecnológicas en la Unión Europea.

El análisis de las contribuciones a repositorios de Software de código abierto en la Unión Europea revela que el ecosistema tiene una naturaleza diferente frente al norteamericano, con un volumen de contribuciones que provienen sobre todo de empleados de compañías pequeñas o muy pequeñas, frente a un escenario en los Estados Unidos en el que predominan grandes compañías tecnológicas que se benefician en sus modelos de negocio de la gran cantidad y de la rápida mejora del Software disponible. En Europa, los contribuyentes individuales ascendieron a más de 260,000, lo que representa el 8% de los casi 3.1 millones de empleados de la UE en el sector del desarrollo de Software en 2018. En total, los más de 30 millones de desarrollos consolidados en repositorios en los estados miembros de la Unión Europea representan una inversión de personal equivalente a casi mil millones de euros, que han pasado a estar disponibles en el dominio público y que, por lo tanto, no tienen que ser desarrollados por otros actores.

Según el análisis, cuanto más pequeña es la empresa, mayor es la inversión relativa en Software de código abierto (las empresas con 50 empleados o menos asumieron casi la mitad de los desarrollos en la muestra de las com-

pañías más activas). Aunque más del 50% de los contribuyentes pertenecen a la industria tecnológica (el 8% del total de sus empleados participaron en estos desarrollos), también hubo participación significativa de empresas de consultoría, científicas, técnicas y, en menor medida, de distribuidores, minoristas y empresas del ámbito financiero.

¿Puede una filosofía de desarrollo como el código abierto, disponible para todo el mundo, llegar a convertirse en una fuente de ventajas diferenciales para el resto de los países, que se ha visto tradicionalmente muy superado en su relevancia en el entorno tecnológico por los gigantes tecnológicos de Estados Unidos o de China? El informe afirma que su uso puede llegar a incidir en gran medida en el desarrollo de una independencia tecnológica superior, de una mayor competitividad y de más innovación. Veremos si llegamos a ver en el resto del mundo políticas que incentiven el uso del código abierto como una variable estratégica clave para ello. La idea, capitalizar la tecnología de una forma más orientada al procomún y al desarrollo colaborativo, suena sin duda atractiva e interesante.

9.5.5 Software Libre en Empresas y Corporaciones

En esta sección exploraremos algunas de las claves por las cuales el Software libre está hoy en el punto de mira de todo tipo de empresas y grandes corporaciones (algunas de las cuales ayer eran sus acérrimos enemigos). Pero hay que empezar destacando que corporativos como Google, Amazon Web Services, Azure de Microsoft, Microsoft, IBM, entre otras, en los últimos años han ganado miles de millones de dólares ofreciendo a sus clientes servicios y/o productos basados en Software libre y con una queja recurrente por su pobre o nula contribución a la comunidad de código abierto que construyó y mantiene esos proyectos.

Si bien es imposible saber exactamente de cuánto dinero estamos hablando, pero es cierto que empresas y corporaciones se benefician diariamente del trabajo de los desarrolladores de código abierto que no emplean.

Grandes Equipos de Programadores GNU/Linux ha demostrado que los equipos de desarrolladores grandes, distribuidos, aunque desorganizados pueden crear Software viable.

Antes de la llegada de GNU/Linux, la mayoría del Software era desarrollado por pequeños equipos de programadores que trabajaban en estrecha coordinación entre sí. Ese era el enfoque recomendado por informáticos de hace

unos decenios, que advertían que añadir más programadores a un proyecto tendía a disminuir su eficiencia. Y estaban muy equivocados.

Desde el principio, el Kernel de Linux se desarrolló con un enfoque diferente, en el que programadores de todo el mundo, que en la mayoría de los casos no se conocían, escribieron e integraron el código de forma rápida y poco organizada. Gracias a la publicación temprana y frecuente, consiguieron que funcionara y hoy día es el Kernel más usado en informática en supercomputadoras y dispositivos móviles.

Pero actualmente hay un mito ampliamente instalado en el mundo de código abierto, que dice que los proyectos son impulsados por una comunidad de contribuyentes gratuitos, pero en realidad, los desarrolladores pagados contribuyen con la mayor parte del código en la mayoría de los proyectos de código abierto modernos de los cuales las corporaciones pueden sacar provecho. Claro ejemplo es el propio Kernel de Linux, en el cual una gran cantidad de desarrolladores actuales pertenecen o son subvencionados por empresas, fundaciones o corporaciones (actualmente cientos de ellas), como en el caso de Linus Torvalds que trabaja bajo los auspicios de la Fundación de Linux.

Reutilización de Software Parte de la razón por la que Linux se hizo muy popular entre los ingenieros de Software con relativa rapidez fue que Linux -y el Software libre en general- facilita la reutilización del código escrito por otras personas.

Hoy en día, la reutilización de Software de terceros es habitual, incluso entre los equipos de desarrollo cuyos productos no son de código libre. Es difícil imaginar la construcción de una aplicación hoy en día sin hacer uso de las bibliotecas de Software de origen, las API de terceros u otros recursos externos a su propio proyecto.

Es cierto que proyectos como GNU, que precedió al Kernel de Linux en siete años, promovían la reutilización de código antes de que apareciera el núcleo. Pero, podría decirse que Linux fue el proyecto que trajo las prácticas de codificación libre a la corriente que tanto parece interesar a ciertas grandes empresas, ayudando a crear el modelo de ingeniería de Software de componentes de Software modulares y reutilizables.

Gestión Actual del Código Fuente Linus Torvalds, que creó el núcleo de Linux cuando era estudiante en Helsinki, es el más famoso por ese trabajo.

Pero un hecho a menudo olvidado es que Torvalds es también el padre de Git, el masivamente popular gestor de código fuente libre.

Torvalds creó Git para ayudar a gestionar el código fuente de Linux. Si Linux no existiera, tampoco existiría Git. Tampoco existiría GitHub, ni GitLab, ni GitOps. Y, lo que es más importante, sin la idea de Software libre y colaborativo de Richard Stallman, tampoco existiría la cultura de intercambio y colaboración abierta que sostienen estas tecnologías.

Estrategias de Despliegue de Software "App Store" Apple puede atribuirse el mérito de haber lanzado la primera App Store, un lugar donde los desarrolladores pueden compartir aplicaciones y los usuarios pueden instalarlas fácilmente, utilizando un catálogo Online centralizado.

Pero al igual que con muchas cosas que ha hecho Apple, el concepto de App Store (que ahora es una estrategia de despliegue de Software apilado como servicio, especialmente pero no sólo en el ecosistema móvil) se parece mucho a lo que los desarrolladores de GNU/Linux estaban haciendo a través de los repositorios de Software mucho antes de que las tiendas de aplicaciones se convirtieran en algo común en el mundo del Software propietario, como también lo es la Tienda de Windows.

Los repositorios de Software en GNU/Linux hacen más o menos lo mismo que las tiendas de aplicaciones: Permiten a los usuarios seleccionar las aplicaciones que quieren de una lista centralizada y en línea, y luego instalarlas con unos pocos clics o bien órdenes de terminal (como el famoso *apt* de Debian).

Es cierto que empresas como Apple parecen tener el mérito de crear tiendas de aplicaciones muy fáciles de usar de hacer clic e ir, pero no es un invento de ellos. Y la historia del concepto de tienda de aplicaciones en general implica a más actores que sólo la comunidad de Apple. Aún así, creo que se podría argumentar con fuerza que, sin GNU/Linux y los repositorios de Software de GNU/Linux, las tiendas de aplicaciones tal y como las conocemos hoy no existirían.

Formatos Abiertos para Intercambio de Información Hay una gran variedad de tecnologías disponibles para producir y almacenar datos. Como son: hojas de cálculo, bases de datos, Software estadístico más específico y más. Esto genera una enorme diversidad de formatos, a veces esto es por decir lo menos caótico.

La ventaja de los archivos de formatos abiertos, es que permiten a los de-

sarrolladores producir varios paquetes de Software y servicios utilizando esos formatos. Esto entonces reduce al mínimo los obstáculos para la reutilización de la información que contienen.

El advenimiento del Software libre ha generado algunos de los formatos abiertos más usados para el intercambio de información, pero los entes generadores de información no siempre se adecuan a los niveles de apertura deseados, algunos de estos formatos abiertos son: XML, JSON, YAML, RDF, REBOL, PDF, CSV, ODF, OOXML, TXT, HTML, HDF.

Pero, incluso si la información se proporciona en formato electrónico, formato legible por máquina y en detalle, puede existir problemas relacionados con el formato del archivo en sí (principalmente el generado por los diversos sistemas operativos). Los formatos en los cuales la información es publicada -en otras palabras, la base digital en la cual la información es almacenada- puede ser "abierta" o "cerrada".

Un formato abierto es aquel donde las especificaciones del Software están disponibles para cualquier persona, de forma gratuita, así cualquiera puede usar dichas especificaciones en su propio Software sin ninguna limitación en su reutilización que fuere impuesta por derechos de propiedad intelectual.

Si el formato del archivo es "cerrado", esto puede ser debido a que el formato es propietario y sus especificaciones no están disponibles públicamente, o porque el formato es propietario y aunque las especificaciones se han hecho públicas, su reutilización es limitada. Si la información es liberada en un formato de archivo cerrado, esto puede causar grandes obstáculos para reutilizar la información codificada en él, forzando a aquellos que deseen usar la información a comprar Software innecesario.

El uso de formatos de archivo con propiedad, para el que la especificación no está disponible públicamente, puede crear dependencia de Software de terceros o de los titulares de licencias de los formatos de archivos. En el peor de los casos, esto puede significar que la información sólo se puede leer con cierto Software específico, que puede ser caro, o que puede quedar obsoleto.

La preferencia del término Gobierno de Datos Abiertos, es que la información se publicará en formatos de archivo abiertos, los cuales son de lectura mecánica y esto es una aportación más del Software libre.

Ciencia Abierta La ciencia abierta (Open Science) es el movimiento creciente para hacer que la ciencia sea abierta. La ciencia en sí misma se utilizó como un ejemplo principal de la eficacia del movimiento de código abierto, ci-

tando prácticas como la difusión abierta de información, métodos y revisión por pares de la literatura científica. Podría decirse que la ciencia abierta comenzó en el siglo XVII con el advenimiento de la revista científica y la práctica de repetir los experimentos presentados en los artículos académicos. Estas revistas se imprimirían y distribuirían en todo el mundo, a menudo supervisadas por sociedades científicas como la Royal Society.

¿Qué impulsó la necesidad de un movimiento de ciencia abierta? La Royal Society tenía el famoso lema "Nullius in verba", traducido de forma aproximada como "no tome la palabra de nadie". Esto encarnaba un principio general en la ciencia de que todas las teorías están abiertas a ser cuestionadas y los resultados declarados deben ser repetibles. De hecho, es una práctica generalizada que fue realizada por la sociedad en esos primeros años. En los últimos años esta práctica no ha sido tan común, con más y más ciencia confiando en elementos cerrados, lo que en última instancia conduce a errores que son más difíciles de detectar sin un intercambio completo de información: datos, métodos y publicaciones.

El movimiento de ciencia abierta afirma en términos generales que la ciencia debe realizarse de manera abierta y reproducible donde todos los componentes de la investigación estén abiertos. Muchas revistas permanecen estancadas en un formato en el que se imprimían físicamente, a pesar de que en la actualidad se distribuyen en gran medida en línea. A menudo, todavía utilizan archivos PDF como una forma de "papel electrónico" con publicaciones fijas, procesos cerrados de revisión por pares y poco o ningún acceso a los datos. Este fue sin duda el modo más eficiente de difundir el conocimiento científico antes de los albores de internet, pero ahora un número cada vez mayor lo considera lejos de ser el óptimo.

La ciencia abierta encarna una serie de aspectos, en el núcleo esto incluye acceso abierto, datos abiertos, código abierto y estándares abiertos que ofrecen una diseminación sin restricciones del discurso científico. Estas cosas permiten una ciencia reproducible al brindar acceso completo a los componentes principales de la investigación científica. Hay una serie de componentes adicionales que también se están explorando, como la revisión por pares abierta, donde los revisores de publicaciones científicas publican revisiones abiertamente con su nombre adjunto y la ciencia de libreta abierta donde las libretas (tradicionalmente cerradas) se publican abiertamente en línea a medida que se realiza la investigación.

¿Por qué la ciencia abierta es tan importante en la era digital? También existe una creciente comprensión de que, dado que la investigación científica

depende cada vez más del código informático para simulaciones, cálculos, análisis, visualización y procesamiento de datos en general, es importante tener acceso a este código tal como tradicionalmente ha sido importante mostrar (y derivar) cualquier nueva técnica matemática introducida para el análisis. Hay revistas como PLOS ONE y F1000 que exploran el significado de las publicaciones, ya sea que se deben congelar en el tiempo o se pueden actualizar. Los repositorios de datos también están ganando importancia a medida que las agencias de financiación requieren la publicación y preservación de los datos generados por la investigación financiada.

En esencia, la ciencia abierta se trata de volver a esos valores fundamentales inculcados por algunos de los primeros científicos de que no debemos confiar en la palabra de nadie, que es esencial que todos los elementos pertinentes a un descubrimiento pretendido se publiquen para que los resultados puedan repetirse y validarse. El movimiento de la ciencia abierta varía en el grado en que lo requiere, pero están surgiendo patrones. Se están estableciendo recomendaciones sobre licencias, como CC0 para datos, CC-BY para publicaciones, licencias compatibles con OSI para código fuente y formatos abiertos para datos. En última instancia, se trata de empoderar a todos para que participen en la ciencia, con internet como vehículo principal para la amplia difusión de este conocimiento.

Este movimiento está cambiando la forma en que se hace la ciencia, está recibiendo el respaldo de muchas agencias de financiamiento, ya que requieren planes de gestión de datos, planes de distribución de código fuente y una mayor validación de los resultados a través del acceso abierto a estos resultados para todos. Esto también mejora la transferencia de conocimientos de la academia a la industria, ya que se brinda acceso completo en el momento de la publicación o después de un período de embargo. El movimiento de la ciencia abierta se limita en gran medida a la investigación que está financiada por las agencias de financiación nacionales de todo el mundo y exige que todos los que financian la investigación tengan acceso total e igualitario a ella.

Open Hardware El concepto de Software libre también se permeó al Hardware. El término Open Hardware u Open Source Hardware, se refiere al Hardware cuyo diseño se hace públicamente disponible para que cualquiera pueda estudiarlo, modificarlo y distribuirlo, además de poder producir y vender Hardware basado en ese diseño. Tanto el Hardware como el Software

que lo habilita, siguen la filosofía del Software libre. Hoy en día, el término "hágalo usted mismo" (DIY por sus siglas en inglés) se está popularizando en el Hardware gracias a proyectos como Arduino que es una fuente abierta de prototipos electrónicos, una plataforma basada en Hardware flexible y fácil de utilizar que nació en Italia en el año 2005.

El movimiento de Hardware abierto o libre, busca crear una gran librería accesible para todo el mundo, lo que ayudaría a las compañías a reducir en millones de dólares en trabajos de diseño redundantes. Ya que es más fácil tener una lluvia de ideas propuesta por miles o millones de personas, que por solo una compañía propietaria del Hardware, tratando así de que la gente interesada entienda cómo funciona un dispositivo electrónico, pueda fabricarlo, programarlo y poner en práctica esas ideas en alianza con las empresas fa-bricantes, además se reduciría considerablemente la obsolescencia programada y en consecuencia evitaríamos tanta basura electrónica que contamina el medio ambiente. Al hablar de Open Hardware hay que especificar de qué tipo de Hardware se está hablando, ya que está clasificado en dos tipos:

- Hardware estático. Se refiere al conjunto de elementos materiales de los sistemas electrónicos (tarjetas de circuito impreso, resistencias, capacitores, LEDs, sensores, etcétera).

- Hardware reconfigurable. Es aquél que es descrito mediante un HDL (Hardware Description Language). Se desarrolla de manera similar a como se hace Software. Los diseños son archivos de texto que contienen el código fuente.

Para tener Hardware reconfigurable debemos usar algún lenguaje de programación con licencia GPL (General Public License). La licencia GPL, al ser un documento que cede ciertos derechos al usuario, asume la forma de un contrato, por lo que usualmente se le denomina contrato de licencia o acuerdo de licencia. La Organización Europea para la investigación Nuclear (CERN) publicó el 8 de julio de 2011 la versión 1.1 de la Licencia de Hardware Abierto.

Existen programas para diseñar circuitos electrónicos y aprender de la electrónica como EDA (Electronic Design Automation) y GEDA (GPL Electronic Design Automation), son aplicaciones de Software libre que permiten poner en práctica las ideas basadas en electrónica.

Es posible realizar el ciclo completo de diseño de Hardware reconfigurable desde una máquina con GNU/Linux, realizándose la compilación, simulación,

síntesis y descarga en una FPGA (Field Programmable Gate Arrays). Para la compilación y simulación se puede usar GHDL (https://ghdl.free.fr) junto con GTKWave (https://gtkwave.sourceforge.net) y para la síntesis el entorno ISE de Xilinx. Este último es Software comercial pero existe una versión gratuita con algunas restricciones.

Sabemos que tanto en el caso del Software como el Hardware, libre no es lo mismo que gratis. Específicamente, en el caso del Hardware, como estamos hablando de componentes físicos que son fabricados, la adquisición de componentes electrónicos puede ser costosa. Aun así, es un campo que no solo es apasionante sino que también tiene mucho futuro y representa grandes oportunidades.

Entusiasmo de la Comunidad Por último, pero no menos importante, probablemente el mayor impacto duradero de GNU/Linux en el modelo de ingeniería de Software se reduce a lo que podría llamarse entusiasmo de la comunidad. Me refiero a la forma en que GNU/Linux en particular, y el Software libre en general, ha animado a los desarrolladores de todo tipo a considerar las contribuciones a la comunidad como uno de sus objetivos finales y esto ahora es notorio no solo en Software, sino en Hardware abierto, obras literarias, escritos técnicos (como este trabajo), imágenes, vídeo, música y un largo etc.

En un mundo de código libre en el que las contribuciones a los proyectos de código pueden ser aceleradores de carrera y el código de licencia libre se reutiliza ampliamente, los desarrolladores entienden que hay un valor real en la construcción de Software que puede beneficiar a tantos usuarios como sea posible.

Tal vez los desarrolladores valorarían a la comunidad en su conjunto si GNU/Linux y el código libre nunca hubieran aparecido. Pero me cuesta imaginar un mundo en el que corporaciones como Microsoft y Google trabajarán juntos en la construcción de Software para GNU/Linux si GNU/Linux no hubiera popularizado el concepto de proyectos de Software impulsados por la comunidad que nadie posee realmente, pero que todos pueden utilizar.

Si bien, es innegable que todo lo anterior puso en la mira de las empresas de todos los tamaños y de las grandes corporaciones el Software libre, la principal razón es el poder utilizar una gran cantidad Software funcional, depurado y ampliamente usado para ofrecer servicios y/o productos basados en Software libre y así beneficiarse económicamente de ello.

Por otro lado, se ha visto a través de múltiples estudios, el impacto y la cuantificación de la importancia económica del código abierto aplicado tanto al Software como al Hardware, su efecto en la contribución al producto interior bruto generado, la reducción en aspectos como el coste total de propiedad, dependencia del proveedor y autonomía digital. Además de generar políticas públicas destinadas a lograr un sector público digitalmente autónomo, una investigación y desarrollo abierto que fomente el crecimiento de los países y una industria más digitalizada y competitiva.

Retomando la frase «el Software de código abierto nunca fue pensado para que las empresas de servicios lo tomasen y lo vendieran». ¿Para qué fue pensado el código abierto, entonces? No hay ninguna licencia de código abierto o Software libre reconocida por la Free Software Foundation o la Open Source Initiative que prohíba hacer negocio con el Software. Lo que prohíben es la discriminación en la capacidad y alcance de su uso en función de la parte, se trate de un individuo o de la mayor multinacional imaginable.

Pese a ello, esta situación está alterando el paradigma actual, en el que el modelo de desarrollo del código abierto se ha impuesto como impulsor de la innovación en el sector empresarial, a la postre el mayor estímulo que haya tenido el código abierto hasta la fecha.

¿Puede una filosofía de desarrollo como el código abierto, disponible para todo el mundo, llegar a convertirse en una fuente de ventajas diferenciales para el resto de los países, que se ha visto tradicionalmente muy superado en su relevancia en el entorno tecnológico por los gigantes tecnológicos de Estados Unidos o de China? Se afirma que su uso puede llegar a incidir en gran medida en el desarrollo de una independencia tecnológica superior, de una mayor competitividad y de más innovación. La idea, capitalizar la tecnología de una forma más orientada al procomún y al desarrollo colaborativo, suena sin duda atractiva e interesante pero no libre de inconvenientes para algunos sectores de desarrolladores de Software libre.

9.6 Código Abierto y las Organizaciones Internacionales

Aunque la Organización de las Naciones Unidas (ONU) ha hablado previamente bien del desarrollo del código abierto, varios eventos recientes muestran que la ONU está tomando medidas definitivas para presentar al mundo entero el camino del código abierto. En julio del 2021, el Consejo Económico y Social de la ONU (ECOSOC) adoptó un proyecto de resolución presentado por el representante de Pakistán titulado: Tecnologías de fuente abierta para

el desarrollo sostenible.

9.6.1 Las Naciones Unidas y el Código Abierto

El ECOSOC destacó la disponibilidad de tecnologías de código abierto que pueden contribuir a los Objetivos de Desarrollo Sostenible (ODS). El consejo invitó al Secretario General a "desarrollar propuestas específicas sobre formas de aprovechar mejor las tecnologías de código abierto para el desarrollo sostenible basadas en las aportaciones de los Estados Miembros interesados y otras partes interesadas".

El desarrollo de tecnología de código abierto puede ser una herramienta rápida y eficaz para la innovación. Aplicarlo a tecnologías apropiadas para ayudar a alcanzar los ODS es extremadamente prometedor. Las "tecnologías apropiadas" abarcan opciones y aplicaciones tecnológicas que son a pequeña escala, económicamente asequibles, descentralizadas, energéticamente eficientes, ambientalmente racionales y fácilmente utilizadas por las comunidades locales para satisfacer sus necesidades.

Existe un caso particularmente fuerte para las tecnologías apropiadas de código abierto OSAT (Outsourced Semiconductor Assembly and Test). OSAT podría ayudar a todos a salir de la pobreza y alcanzar un estado sostenible aprovechando el mismo tipo de desarrollo que hace que el Software de código abierto sea un éxito rotundo.

La Declaración Ministerial del Foro político de alto nivel sobre desarrollo sostenible también destacó la importancia de "tecnologías no patentadas que pueden contribuir a los Objetivos de Desarrollo Sostenible, a través de diversas fuentes de acceso abierto". Pidió "el desarrollo y la puesta en funcionamiento de una plataforma en línea en el marco del Mecanismo de facilitación de la tecnología para establecer un mapeo integral y servir como puerta de entrada a la información sobre iniciativas, mecanismos y programas de ciencia, tecnología e innovación existentes, dentro y fuera de las Naciones Unidas".

Es un pequeño paso, pero muy emocionante, porque las Naciones Unidas no se demoran una vez que ven formas de ayudar a sus Estados Miembros y a las personas que los integran. Ahora, el Departamento de Asuntos Económicos y Sociales de las Naciones Unidas (DESA) está trabajando para que esto suceda. DESA está utilizando una Nota sobre una base de datos centralizada propuesta de las Naciones Unidas de tecnologías apropiadas de código abierto publicada por la Conferencia de las Naciones Unidas sobre Comercio

y Desarrollo (UNCTAD) para hacerlo.

La Nota de la UNCTAD aboga por una base de datos centralizada de OSAT para acelerar el descubrimiento y la innovación en todos los sectores asociados con los ODS al tiempo que se minimizan los obstáculos legales o financieros. Esto es importante para la difusión del acervo mundial de conocimientos, especialmente en los países en desarrollo.

Actualmente, no existe un repositorio completo o una base de datos central de OSAT y Appropedia.org, quizás sea el mejor ejemplo. Sin embargo, la Nota de la UNCTAD dice: "Muchas organizaciones, organizaciones sin fines de lucro y empresas con fines de lucro están desarrollando OSAT y manteniendo bases de datos existentes a pequeña escala. Si bien hay muchos OSAT disponibles, se encuentran dispersos en varias bases de datos para tecnologías particulares. Mientras tanto, sigue existiendo una clara necesidad de aumentar la tasa de uso de OSAT.

Por lo tanto, existe una necesidad urgente de una base de datos de código abierto centralizada global (COSD) confiable. Al tener un alcance global, un repositorio de COSD proporcionaría una ventanilla única a la que todos pueden acceder para resolver los desafíos locales".

Concluye: "La ONU está bien posicionada para liderar el establecimiento de un COSD dado su papel bien establecido en la promoción de la tecnología de código abierto a través de varios foros y publicaciones intergubernamentales. En particular, 2030 Connect es una plataforma tecnológica en línea de la ONU que se desarrolló como parte del trabajo del Equipo de Trabajo Interinstitucional de la ONU. El COSD podría mejorarlo".

Con el liderazgo de la ONU, quizás no estemos demasiado lejos de cuándo, sí tiene un problema local (sin importar en qué parte del mundo se encuentre), pueda descargar una solución de código abierto examinada y probada. Quizás, esta es la potencia de fuego que necesitamos para alcanzar los ambiciosos Objetivos de Desarrollo Sostenible.

9.6.2 La Comisión Europea se Compromete a Liberar Todo el Software que Pueda Beneficiar a la Sociedad

A finales del 2021, la Unión Europea (UE) y su órgano legislativo, la Comisión Europea siguen avanzando en su estrategia digital con el Software de código abierto como uno de los pilares fundamentales. En esta ocasión ha sido esta última la que anuncia novedades para con la distribución del Software desarrollado para cubrir necesidades internas de la organización.

De acuerdo a la información publicada, la Comisión Europea ha aprobado una nueva regulación que favorece el libre acceso al Software que producen siempre y cuando existan beneficios potenciales para «los ciudadanos, las empresas u otros servicios públicos», lo que de la teoría a la práctica bien puede abarcar todo lo que se desarrolle bajo su tejado.

Esta nueva disposición se apoya a su vez en un reciente estudio realizado también por la Comisión sobre el impacto del Software de código abierto en áreas como la independencia tecnológica, la competitividad y la innovación en la economía de la Unión Europea. El objetivo, hallar evidencias sólidas con las que conformar las políticas europeas de código abierto para los próximos años.

En términos económicos, de hecho, los cálculos son de lo más optimistas y apuntan un impacto económico contundente, de miles de millones de euros de ahorro al año -a modo de ejemplo, se estimó entre 65 y 95 mil millones de euros solo en 2018- y con un incremento mínimo en la apuesta, se podría dar un crecimiento del PIB de la UE de en torno a los 100,000 millones de euros.

Con semejante escenario, no es de extrañar que la misma Comisión Europea esté interesada en promover las soluciones de código abierto dentro y fuera de las instituciones y no solo se basan en el beneficio económico directo: son muchas otras las ventajas del modelo también recogidas en el informe, tal y como se ha mencionado: independencia, competitividad, innovación... y en el caso de las administraciones públicas, colaboración, reutilización y transparencia.

En palabras de Johannes Hahn, comisario de Presupuesto y Administración: «El código abierto ofrece grandes ventajas en un ámbito en el que la UE puede desempeñar un papel de liderazgo. Las nuevas normas aumentarán la transparencia y ayudarán a la Comisión, así como a los ciudadanos, las empresas y los servicios públicos de toda Europa, a beneficiarse del desarrollo de Software de código abierto. Poner en común los esfuerzos para mejorar el Software y la creación conjunta de nuevas funciones reduce los costes para la sociedad, ya que también nos beneficiamos de las mejoras realizadas por otros desarrolladores. Esto también puede mejorar la seguridad, ya que especialistas externos e independientes comprueban los fallos y las deficiencias de seguridad de los programas informáticos».

La comisaría de Innovación, Investigación, Cultura, Educación y Juventud, Mariya Gabriel, ha declarado: «La Comisión pretende, con su ejemplo, estar al frente de la transición digital en Europa. Con las nuevas normas, la

Comisión aportará un valor significativo a las empresas, también las emergentes, a los innovadores, a los ciudadanos y las administraciones públicas, poniendo a su disposición el código abierto de sus soluciones informáticas. Esta decisión también ayudará a estimular la innovación, gracias al código de la Comisión disponible públicamente».

Como muestra del Software desarrollado bajo el amparo de la Comisión Europea que va a ser liberado se incluyen proyectos como eSignature, «un conjunto de normas, herramientas y servicios gratuitos que ayudan a las administraciones públicas y a las empresas a acelerar la creación y verificación de firmas electrónicas jurídicamente válidas en todos los Estados miembros de la UE»; o LEOS (Legislation Editing Open Software), «el Software utilizado en toda la Comisión para elaborar textos jurídicos. LEOS, escrito originalmente para la Comisión, se está desarrollando en estrecha colaboración con Alemania, España y Grecia».

Esta nueva iniciativa de la Comisión Europa contempla asimismo la creación de un repositorio centralizado para facilitar el descubrimiento, el acceso y la reutilización del Software incluido, el cual se sumará a todos los proyectos realizados por las diferentes administraciones públicas comunitarias en base al mismo modelo de desarrollo. Y viene de lejos este impulso, aun cuando comienza a unificarse ahora.

Sin ir más lejos, hace años que la propia Comisión Europea puso en marcha el programa Interoperable Delivery of European eGovernment Services to Public Administrations, Businesses and Citizens que dio origen al observatorio JoinUp (https://joinup.ec.europa.eu/), en cuyas páginas se recogen casi 3,000 soluciones de Software abierto, 133 colecciones de recursos y cuantiosa información relacionada.

Más tarde, de 2014 a 2017 se inició la «primera fase» en la estrategia de código abierto de la Unión Europea, especialmente dentro de la propia Comisión, estableciendo determinados requisitos en materia de Software de código abierto; actualmente se está desarrollando la nueva «estrategia de código abierto 2020-2023», con la que la Comisión Europea pretende ampliar y afianzar los objetivos de la estrategia digital y la contribución al programa Europa Digital.

10 Apéndice B: El Sistema Operativo GNU/Linux

GNU/Linux se ve y se siente muy parecido a cualquier otro sistema UNIX, y de hecho la compatibilidad con UNIX ha sido una importante meta del diseño del proyecto Linux. No obstante, Linux es mucho más joven que la mayor parte de los sistemas UNIX. Su desarrollo se inició en 1991, cuando un estudiante finlandés, Linus Torvalds, escribió y bautizó un pequeño pero autosuficiente núcleo para el procesador 80386, el primer procesador de 32 bits verdadero en la gama de CPU compatibles con el PC de Intel.

En los albores de su desarrollo, el código fuente de Linux se ofrecía gratuitamente en internet. En consecuencia, su historia ha sido una colaboración de muchos usuarios de todo el mundo que se han comunicado casi exclusivamente a través de internet. Desde un núcleo inicial que implementaba parcialmente un subconjunto pequeño de los servicios de UNIX, Linux ha crecido para incluir cada vez más funcionalidades UNIX.

En sus inicios, el desarrollo de Linux giraba en gran medida alrededor del núcleo del sistema operativo central: el ejecutivo privilegiado que administra todos los recursos del sistema e interactúa directamente con el Hardware. Desde luego, se requiere mucho más que este núcleo para producir un sistema operativo completo.

Resulta útil hacer la distinción entre el núcleo (Kernel) de Linux y un sistema Linux: el núcleo en Linux es una identidad de Software totalmente original desarrollada desde cero por la comunidad Linux (suele encontrarse en el directorio */boot* en el sistema de archivos); el sistema Linux, tal como lo conocemos hoy, incluye una multitud de componentes, algunos escritos desde cero, otros tomados en préstamo de otros proyectos o creados en colaboración con otros equipos como el proyecto GNU de la Free Software Foundation.

El sistema Linux básico es un entorno estándar para aplicaciones y programación de los usuarios, pero no obliga a adoptar mecanismos estándar para controlar las funcionalidades disponibles como un todo.

GNU/Linux en sus inicios trabajaba en modo consola o terminal -línea de comandos o *Shell*- usando la interfaz de línea de comando (CLI por Command Line Interface) o interfaz de usuario de terminal (TUI por Terminal User Interface). A medida que Linux ha madurado, se ha hecho necesaria otra capa de funcionalidad encima del sistema Linux: El servidor de Pantalla

Un servidor de pantalla es un programa cuya tarea principal es coordinar la entrada y salida de sus clientes hacia y desde el resto del sistema operativo, el Hardware y entre sí. El servidor de pantalla se comunica con sus clientes a

través del protocolo del servidor de pantalla. Un servidor de visualización es un componente clave en cualquier interfaz gráfica de usuario, específicamente el sistema de ventanas. Es el componente básico de la interfaz gráfica de usuario (GUI) que se encuentra entre la interfaz gráfica y el Kernel.

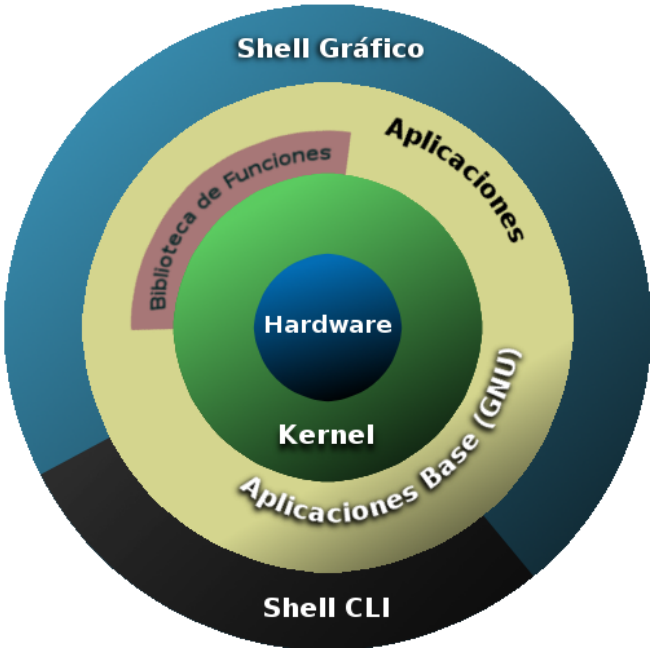


Figura 2: GNU/Linux

Entonces, gracias a un servidor de pantalla, podemos usar la computadora con GUI. Sin él, solo estaría restringido a una interfaz de línea de comandos, seria placentero, pero la GUI también tiene magia. No confundir el servidor de visualización con el entorno de escritorio (GNOME, KDE, LXQt, LXDE, Xfce, Unity, MATE, Cinnamon, Pantheon, Deepin, Budgie, PIXEL, Enlightenment, Trinity, Moksha, Ukui, etc.), los entornos de escritorio usan uno, pero en una capa por debajo.

El servidor de pantalla se comunica con sus clientes a través del protocolo del servidor de pantalla. Hay tres protocolos principales de servidor de pantalla disponibles: X11, Mir (desarrollado por Canonical) y Wayland.

X Window System, a menudo denominado simplemente X, es el mas antiguo de todos (1984), terminó siendo el sistema de ventanas predeterminado para la mayoría de los sistemas operativos similares a UNIX, incluido

GNU/Linux. El servidor X.Org es la implementación gratuita y de código abierto del servidor de visualización X Window System administrado por la Fundación X.Org. Es una aplicación que interactúa con aplicaciones cliente a través del protocolo X11 para dibujar cosas en una pantalla y enviar eventos de entrada como movimientos del mouse, Clics y pulsaciones de teclas. Normalmente, uno iniciaría un servidor X que esperará a que las aplicaciones de los clientes se conecten a él. Xorg se basa en un modelo cliente/servidor y, por lo tanto, permite que los clientes se ejecuten de forma local o remota en una máquina diferente. La mayoría de las funciones que proporciona el protocolo X Server ya no se utilizaban. Casi todo el trabajo que hizo X11 se volvió a delegar a las aplicaciones individuales y al administrador de ventanas. Y, sin embargo, todas esas características antiguas siguen ahí, pesando sobre todas estas aplicaciones, perjudicando el rendimiento y la seguridad. Es por ello y otras cosas que nace Wayland[78].

[78] Wayland fue iniciado por Kristian Hogsberg, un desarrollador de X.Org, como un proyecto personal en 2008. Es un protocolo de comunicación que especifica la comunicación entre un servidor de pantalla y sus clientes. Wayland se desarrolla como un proyecto gratuito y de código abierto impulsado por la comunidad con el objetivo de reemplazar el sistema de ventanas X11 o Xorg, por un sistema de ventanas moderno, seguro y más simple.

En Wayland, el compositor es el servidor de visualización. Compositor, es un administrador de ventanas que proporciona a las aplicaciones un búffer fuera de la pantalla para cada ventana. El administrador de ventanas compone los búffers de la ventana en una imagen que representa la pantalla y escribe el resultado en la memoria de visualización. El protocolo Wayland permite al compositor enviar los eventos de entrada directamente a los clientes y permite que el cliente envíe el evento de daños directamente al compositor.

La principal ventaja de Wayland sobre X es que comienza desde cero. Una de las principales razones de la complejidad de X es que, a lo largo de los años, su función ha cambiado. Como resultado, hoy en día, X11 actúa en gran medida como un protocolo de comunicaciones "realmente terrible" entre el cliente y el administrador de ventanas. Wayland también es superior cuando se trata de seguridad. Con X11, es posible hacer Keylogging al permitir que cualquier programa exista en segundo plano y lea lo que está sucediendo con otras ventanas abiertas en el área de X11. Con Wayland, esto simplemente no sucederá, ya que cada programa funciona de forma independiente.

Sin embargo, el sistema X Window todavía tiene muchas ventajas sobre Wayland. Aunque Wayland elimina la mayoría de los defectos de diseño del Xorg, tiene sus propios problemas. A pesar de que el proyecto Wayland ha estado activo durante más de diez años, las cosas no son 100% estables, de ahí que aun sigamos usando Xorg. Aún hoy, la mayoría de los videojuegos y las aplicaciones de gráficos intensivos para Linux todavía se escriben para X11. Además, muchos controladores de gráficos de código cerrado, como los de las GPU NVIDIA, aún no ofrecen soporte completo para Wayland. Desde mi punto de vista aún tendremos Xorg para una década más, aunque ya comiencen a salir algunos

Una distribución de GNU/Linux incluye todos los componentes estándar del sistema Linux, más un conjunto de herramientas administrativas que simplifican la instalación inicial y desinstalación de paquetes del sistema.

GNU/Linux puede funcionar tanto en entorno gráfico (GUI por Graphical User Interface) como en modo consola o terminal -línea de comandos o *Shell*- usando la interfaz de línea de comando (CLI por Command Line Interface) o interfaz de usuario de terminal (TUI por Terminal User Interface). La consola es común en distribuciones para servidores, mientras que la interfaz gráfica está orientada al usuario final del hogar como empresarial. Así mismo, también existen los entornos de escritorio (GNOME, KDE, LXQt, LXDE, Xfce, Unity, MATE, Cinnamon, Pantheon, Deepin, Budgie, PIXEL, Enlightenment, Trinity, Moksha, Ukui, etc.), que son un conjunto de programas conformado por ventanas, íconos y muchas aplicaciones que facilitan el uso de la computadora.

Hardware en GNU/Linux El soporte del Hardware en Linux es un asunto complicado, es lo que más problemas da. Linux soporta la mayor parte del Hardware, pero a veces pueden existir problemas[79]:

- si es Hardware muy antiguo, muy moderno o muy raro.

proyectos como Sway que se favorecen de las bondades de Wayland.

[79] Mito: Linux/Unix se puede usar para revivir un equipo de cómputo viejo. La realidad es que si bien, hay múltiples distribuciones de Linux/Unix que corren en una gran cantidad de procesadores antiguos y actuales, los Drivers necesarios para reconocer periféricos como tarjetas gráficas, de red alámbrica e inalámbrica, entre muchos otros, no tienen soporte en Linux/Unix, lo cual hará imposible su uso en Linux/Unix. Esto es cierto en cualquier computadora no importa de cuál generación es el equipo de cómputo. La verdad de todo esto, es que los fabricantes están enfocados en producir Hardware y Drivers que corran en los sistemas operativos con mayor cuota de mercado y por el momento Linux/Unix en equipos personales no son de ellos.

La compatibilidad del Hardware depende en gran medida de la versión de Kernel de GNU/Linux instalado, es de esperarse que en versiones anteriores del Kernel cierto Hardware no se pueda detectar, pero lo contrario también pasa, hay Drivers que solo corren correctamente en versiones anteriores del Kernel y no en las últimas versiones, lo que ocasiona que muchos usuarios se desesperen al tratar de usar sus equipos con GNU/Linux. Y en caso de logar que funcione el Hardware, se fuerza a los usuarios a usar una determinada versión del Kernel (y todas las aplicaciones de la distribución) no actualizable, por la imposibilidad de hacer funcionar el Hardware del equipo en una más moderna con la consiguiente obsolescencia del Software instalado en el equipo.

- si es un dispositivo exclusivo para Windows, como los Winmodems (linmodems.org).

Si el Hardware es "de verdad" (no Winmodems), de marca conocida y actual, casi con toda seguridad estará soportado por Linux.

Los instaladores de Linux reconocen prácticamente todo el Hardware durante la instalación, por lo que la mejor manera de evitar problemas con el Hardware es instalarlo desde el principio. Si añadimos Software para nuestro Hardware posteriormente a la instalación nos costará hacerlo funcionar. ¡Incluso puede ser más rápido instalar el sistema desde cero!.

¿Cómo puedo saber si mi Hardware está soportado por Linux (antes de comprarlo y cometer un error de forma irreparable)? Fácil: consultando en internet.

¿Qué sabe Linux de mi Hardware? El Kernel se encarga de la gestión del Hardware usando herramientas como *Udev* (sistema de nombrado del Hardware), *Hotplug* (mecanismo de avisos), *Dbus* (comunicaciones entre procesos) o *Hal* (capa de abstracción de Hardware), y mapea todo el Hardware en archivos de dispositivos ubicados en los directorios */dev* y */sys*.

Algunos comando usados para conocer el Hardware son:

- lscpu - Información de procesador
- lshw - Lista de Hardware en Linux
- hwinfo - Información del Hardware en Linux
- lspci - Lista PCI
- lsscsi - Listar dispositivos SCSI
- lsusb - Lista de los buses usb y detalles del dispositivo
- inxi - Script mega Bash para usuarios no técnicos
- lsblk - Lista de dispositivos de bloque
- df - espacio en disco de los sistemas de archivos

- fdisk - Informa y permite modificar las particiones de disco
- mount - Permite montar y desmontar y ver sistema de archivo montados
- free - Información de la memoria RAM y Swap
- lsmem - Lista los rangos de memoria disponible.
- hdparm - Información de disco duro
- lsmod - Información de los modulos de los drivers cargados al Kernel

Archivos del directorio */proc*, contienen información accesible usando el comando *cat:*

- Información de CPU
- Información del Kernel de Linux
- Dispositivos Sata / SCSI
- Particiones

Componentes de un Sistema GNU/Linux El sistema GNU/Linux se compone de tres cuerpos principales de código, al igual que la mayor parte de las implementaciones de UNIX.

- El núcleo se encarga de mantener todas las abstracciones importantes del sistema operativo, incluidas cosas tales como memoria virtual y procesos.
- Las bibliotecas del sistema definen un conjunto estándar de funciones a través de las cuales las aplicaciones pueden interactuar con el núcleo y que implementan gran parte de la funcionalidad del sistema operativo que no necesita todos los privilegios del código del núcleo.
- Las utilerías del sistema que son programas que realizan tareas de administración especializadas individuales. Algunos programas utilitarios

pueden invocarse una sola vez para asignar valores iniciales y configurar algún aspecto del sistema; otros llamados demonios[80] podrían ejecutarse de forma permanente, realizando tareas tales como responder a conexiones de red entrantes, aceptar solicitudes de ingreso al sistema desde terminales, o actualizar archivos de bitácora.

Principios de Diseño Unix y posteriormente Linux se diseñaron como sistemas de tiempo compartido. La interfaz estándar con el usuario (el Shell) es sencilla y puede ser sustituida por otra si se desea[81]. El sistema de archivos es un árbol invertido con múltiples niveles, que permite a los usuarios crear sus propios subdirectorios. Cada archivo de datos de usuario es tan solo una secuencia de Bytes.

El sistema UNIX/Linux fue diseñado por programadores para programadores; por ello, siempre ha sido interactivo, y las funciones para desarrollar programas siempre han tenido prioridad. Tales recursos incluyen a los programas *make, gcc, git*, etc.

Los archivos de disco y los dispositivos de Entrada/Salida (E/S) se tratan de la manera más similar posible. Así, la dependencia de dispositivos y las peculiaridades se mantienen en el núcleo hasta donde es posible; aún en el núcleo, la mayor parte de ellas están confinadas a los Drivers de los dispositivos.

Un archivo en Unix/Linux es una secuencia de Bytes. Los diferentes programas esperan distintos niveles de estructura, pero el núcleo no impone ninguna estructura a los archivos. Por ejemplo, la convención para los archivos de texto es líneas de caracteres *ASCII* separadas por un solo carácter de nueva línea (que es el carácter de salto de línea en *ASCII*), pero el núcleo nada sabe de esta convención.

Los archivos se organizan en directorios en estructura de árbol. Los directorios también son archivos que contienen información sobre cómo encontrar otros archivos. Un nombre de camino, trayectoria o ruta de un archivo es

[80] En sistemas UNIX/LINUX se conoce como demonio o Daemon (Disk And Execution Monitor) a un proceso que se ejecuta en segundo plano del sistema operativo, se ejecuta en todo momento y no posee interacción directa con el usuario, también se le conoce genéricamente como servicio o proceso, del cual no percibimos su ejecución. Un demonio realiza una operación específica en tiempos predefinidos o en respuesta a ciertos eventos del sistema.

[81] Algunos de los distintos tipos de Shell son: Shell Bourne, Shell Zsh, Shell C, Shell Korn, Shell Bourne-Again (mejor conocido como Bash, Bourne again shell), etc.

una cadena de texto que identifica un archivo especificando una ruta a través de la estructura de directorios hasta el archivo. Sintácticamente, una trayectoria consiste en nombres de archivos individuales separados por el carácter diagonal. Por ejemplo */usr/local/fuente*, la primera diagonal indica la raíz del árbol de directorios, llamado directorio *raíz* o *root*. El siguiente elemento, *usr*, es un subdirectorio de la raíz, *local* es un subdirectorio de *usr* y *fuente* es un archivo o directorio que está en el directorio *local*. No es posible determinar a partir de la sintaxis del nombre de una trayectoria si la *fuente* es un archivo ordinario o un directorio.

Un archivo puede conocerse por más de un nombre en uno o más directorios. Tales nombres múltiples se denominan enlaces, también se manejan enlaces simbólicos, que son archivos que contienen el nombre de una ruta de otro archivo o directorio. Las dos clases de enlaces también se conocen como enlaces duros y enlaces blandos. Los enlaces blandos (simbólicos), a diferencia de los duros pueden apuntar a directorios y pueden cruzar fronteras de sistemas de archivos (apuntar a otros sistemas de archivos) y el sistema operativo trata igualmente todos los enlaces.

El nombre del archivo *"."* en un directorio es un enlace duro al directorio mismo. El nombre de archivo *".."* es un enlace al directorio padre. Por tanto, si el directorio actual es */usr/jlp/programa*, entonces *../bin/wdf* se refiere a */usr/jpl/bin/wdf*.

Los dispositivos de Hardware tienen nombres en el sistema de archivos. El núcleo sabe que estos archivos especiales de dispositivos o archivos especiales son interfaces con dispositivos, pero de todos modos el usuario accede a ellos prácticamente con las mismas llamadas al sistema que otros archivos.

10.1 Sistema de Archivos y Estructura de Directorios

El Sistema de Archivos de Unix/Linux o cualquier sistema de archivos, generalmente es una capa bajo el sistema operativo la cual maneja el posicionamiento de tus datos en el almacenamiento, sin este el sistema no puede saber dónde empieza y termina un archivo.

Éste consiste en un conjunto de archivos, cada uno de los cuales tiene un nombre, hay tres clases de archivos:

- Archivos regulares, que contienen información.
- Directorios o carpetas, que contienen un conjunto de archivos. Los

conjuntos de archivos se identifican por el nombre del directorio que los agrupa.

- Archivos especiales FIFO (First In First Out), algunas veces llamados Pipes, que proveen un canal para comunicación entre procesos independientes y que tienen un nombre asociado.

Como la mayoría de los sistemas operativos modernos, Unix/Linux organiza su sistema de archivos como una jerarquía de directorios. La jerarquía de directorios en un árbol invertido, un directorio especial, root '/', es la raíz de la jerarquía. Un directorio puede contener subdirectorios, archivos regulares y archivos especiales. Unix/Linux trata a los subdirectorios como tipo particulares de archivos. Además Unix/Linux ve a un archivo, no importando su tipo, como una sucesión de Bytes, almacenados en dispositivos como Discos, CD, DVDs o unidades de almacenamiento USB.

Un nombre de archivo puede tener hasta 255 caracteres en la mayoría de los sistemas. y contener cualquier carácter excepto '/' o el carácter NULL; sin embargo, algunos caracteres como '&' y ' ' pueden causar problemas por sus significados especiales para su interpretación en la línea de comandos. Los nombres de archivos son sensibles a las mayúsculas y minúsculas.

En los sistemas Unix/Linux, cada usuario tiene un directorio personal, en el cual almacena sus archivos. Dicho directorio se conoce como el hogar (Home) del usuario. Una vez que entres en sesión, cada programa que se ejecutase encuentra situado en un directorio de trabajo.

Tipos de Sistema de Archivos de GNU/Linux Cuando intentas instalar Linux, notaras que ofrece distintos sistemas de archivos como los siguientes:

Ext, Ext2, Ext3, Ext4, JFS, XFS, Btrfs, FAT32, exFAT, NTFS y Swap

Así que, ¿qué son estos sistemas de archivos que ofrece Linux?

- Ext: Antiguo y descontinuado debido a sus limitaciones.
- Ext2: Primer sistema de archivos de Linux que permite 2 Terabytes de datos.

- Ext3: Evolución del Ext2, con actualizaciones y retrocompatibilidad[82].
- Ext4: Es más rápido y permite archivos mucho más grandes con una velocidad significativa[83].
- F2FS: (Flash-Friendly File System) es un sistema de archivos que toma en cuenta las características de los dispositivos de almacenamiento basados en memorias Flash NAND como las unidades de estado sólido (SSD) y tarjetas eMMC y SD.
- JFS: Sistemas de archivos hechos por IBM. Funcionan bien con archivos grandes y pequeños, pero existen reportes de fallas y los archivos se corrompen después de un largo tiempo de uso.
- XFS: Sistema de archivos creado por SGI que funciona lento con archivos pequeños.
- Btrfs: Hecho por Oracle, no es tan estable como Ext en algunas distribuciones, pero es un buen reemplazo, si es necesario.
- FAT32: Establecido en 1996, es robusto, versátil pero anticuado, sólo permite guardar archivos de hasta 4 GB.
- exFAT: Es una actualización de FAT32 para acabar con la limitación de 4 GB por archivo.
- NTFS: Es una alternativa a FAT32 sin sus limitaciones, usada en discos duros y unidades externas.
- Swap: Es un espacio de intercambio que es utilizado para almacenar datos temporales, reduciendo así el uso de la RAM, normalmente es del doble del tamaño de la RAM del equipo.

Otras opciones de sistemas de archivos son: ZFS, ReiserFS, UFS, FFS, HFS, APFS. Algunas de sus características de los sistemas de archivo más usados son:

[82] El único problema que tiene es que los servidores no utilizan este tipo de sistema de archivos debido a que no soporta recuperación de archivos o Snapshots del disco.

[83] Es una muy buena opción para discos de estado sólido SSD, además puedes darte cuenta que cuando intentas instalar cualquier distribución de Linux este es el sistema de archivo por defecto que sugiere Linux.

Sistema	Tamaño máx. volumen	Tamaño máx. archivo
$FAT32$	8 TB	4 GB
$NTFS$	1.845^7 TB	256 TB
$EXT4$	1.153^6 TB	17.5921 TB

En el sistema de archivos de Linux (Ext2/3/4, UFS, ReiserFS, FFS, XFS, Btrfs, etc.) se tiene asociado un elemento en la tabla que guarda a los archivos y directorios dentro del sistema de archivos, que contiene un número entero único. Este número identifica la ubicación del archivo dentro del área de datos llamado *inodo (i-node)*.

Cada *inodo* contiene información de un fichero o directorio. En concreto, en un *inodo* se guarda la siguiente información:

- El identificador del dispositivo que alberga al sistema de archivos.
- El número de *inodo* que identifica al archivo dentro del sistema de archivos.
- La longitud del archivo en Bytes.
- El identificador de usuario del creador o un propietario del archivo con derechos diferenciados.
- El identificador de grupo de un grupo de usuarios con derechos diferenciados.
- El modo de acceso: capacidad de leer, escribir, y ejecutar el archivo por parte del propietario, del grupo y de otros usuarios.
- Las marcas de tiempo con las fechas de última modificación (*mtime*), acceso (*atime*) y de alteración del propio *inodo* (*ctime*).
- El número de enlaces (*Hard Links*), esto es, el número de nombres (entradas de directorio) asociados con este *inodo*.
- Tabla de direccionamiento donde se detallan los bloques del disco duro en que está almacenado el fichero.

podemos conocer la información del *inodo* de un archivo/directorio/enlace usando:

```
$ stat nombre
```

o bien mediante:

```
$ ls -i
```

si conocemos el *inodo* de un archivo y queremos conocer los nombres de archivo de los enlaces, usamos *(por ejemplo 3432343)*:

```
$ find / -inum 3432343
```

y si queremos conocer el número de inodos de disco podemos usar:

```
$ df -i
```

Además están las Dentries que tienen la función de definir la estructura de un directorio, éstas conjuntamente con los inodos, serán los encargados de representar un fichero en la memoria. Notemos que en cada directorio del sistema existen dos entradas especiales:

- La entrada con un punto (.) hace referencia al propio directorio.
- La entrada con dos puntos (..) hace referencia al inodo del directorio que contiene el directorio (el padre).

El área de datos ocupa el resto del disco y es equivalente a la zona de datos en el *FAT*. En esta zona, como su nombre indica, están almacenados los ficheros y directorios de nuestro sistema.

Estructura de Directorios en GNU/Linux Además de los sistemas de archivos que difiere de la de Windows, la estructura de directorios en Linux es distinta, y es necesario conocerla para encontrar ficheros de configuración, instalar ciertos paquetes en el lugar adecuado, localizar las fuentes del Kernel, o la imagen de este, nuestros ficheros personales, entre otros.

De hecho, la Fundación Linux mantiene un estándar de jerarquía del sistema de archivos, este define la estructura de directorios y el contenido de los directorios en las distribuciones Linux. Gracias a este estándar es posible encontrar la misma estructura de directorios en (casi) todas las distribuciones de Linux[84] que a continuación describiremos brevemente:

[84] Recordemos que Linux se basa en UNIX y, por tanto, toma prestada su jerarquía de sistema de archivos de UNIX. Encontramos una estructura similar en sistemas operativos similares a UNIX, como BSD y MacOS.

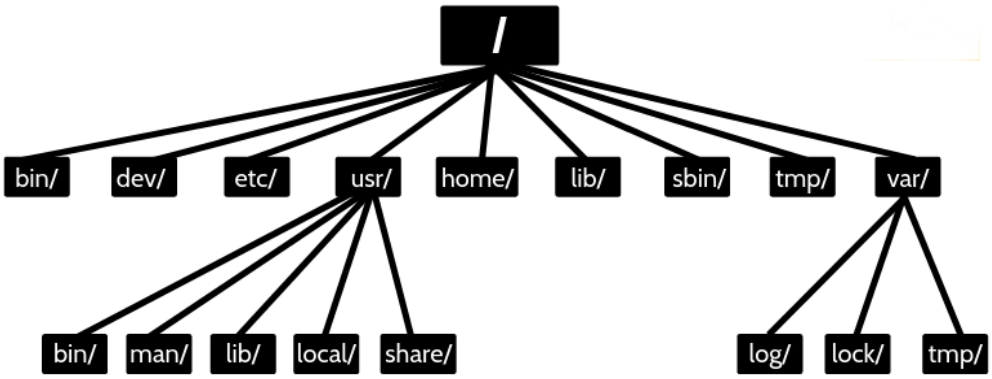


Figura 3: Jerarquía del sistema de archivos de Linux.

/ es el directorio principal, la *raíz* o *root*. Contiene el resto de directorios, es decir, todos los demás serían subdirectorios de este (incluso si están en particiones o discos diferentes). Sin duda es el más importante.

/bin es el directorio donde se almacenan los binarios, es decir, los programas que emplea el sistema para labores administrativas como los comandos *cp, echo, grep, mv, rm, ls, kill, xkill, ps, su, tar*, etc.

/sbin la S es de System, y como su nombre indica, aquí se almacenan los binarios o programas que emplea el propio sistema operativo para tareas de arranque, restauración, etc. Por ejemplo, *fsck, mount, mkfs, reboot, swapon.*

/boot es el directorio de arranque, donde está la o las imágenes del Kernel Linux que se cargarán durante el arranque, también directorios y configuración del propio gestor de arranque.

/etc muy importante para el administrador, ya que aquí residen los ficheros de configuración de los componentes del sistema y otros programas instalados.

/dev es un directorio muy especial donde se encuentran los dispositivos de bloques o caracteres, es decir, ficheros que representan la memoria, particiones, discos, dispositivos de Hardware, etc. Ya sabes que en LINUX y

UNIX todo es un archivo, y no unidades como en Windows. Por ejemplo, el disco duro o particiones serían */dev/sda, /dev/hda, /dev/scd*, /dev/fd/, etc.

/proc es otro directorio muy especial, más que un directorio es una interfaz por decirlo de un modo sencillo. Y aquí el sistema nos presenta los procesos[85] como directorios numerados con el identificador de procesos *PID* (Process ID). Dentro de cada uno de ellos estaría toda la información necesaria para la ejecución de cada proceso en marcha. Además, encontrarás ficheros de los que extraer información importante, como *cpuinfo, meminfo*, etc. Es precisamente de estos ficheros de los que extraen información algunos comandos que usamos habitualmente, como por ejemplo, cuando hacemos uso de *free* para consultar la memoria disponible, este comando realmente estaría mostrando el contenido de */proc/meminfo* de una forma ordenada.

/media o /mnt son los directorios donde se establecen generalmente los puntos de montaje. Es decir, cuando insertamos algún medio extraíble o recurso de red compartido, etc., que hayamos montado, estaría aquí si lo hemos puesto como punto de montaje. El primero es más específico para medios que se montan de una forma temporal.

/home es el directorio para los usuarios estándar. Por ejemplo, aquí se almacenan dentro de directorios separados (uno para cada usuario con su nombre), los ficheros personales. Por ejemplo, */home/antonio* sería mi directorio personal.

/lib o /lib64 es donde se alojan las bibliotecas necesarias para los programas ejecutables presentes en el sistema. En /lib64 estarían las de las aplicaciones de 64 bits y en */lib* estarían las aplicaciones de 32 bits.

[85] Existen procesos activos y dormidos, procesos huérfanos y procesos zombies.

Los procesos activos son aquellos que están en ejecución en el sistema y los procesos dormidos son aquellos que esperan algún recurso o señal para continuar con su ejecución.

Los procesos huérfanos son aquellos que se siguen ejecutando a pesar que su proceso padre concluyó su operación.

Los procesos zombies es un proceso que ha concluido pero aún están presentes en la tabla de procesos.

/opt es un directorio que almacenará los paquetes o programas instalados en el sistema que son de terceros. Por ejemplo, si instalamos algún antivirus, Chrome, Arduino IDE o ciertos paquetes grandes, suelen instalarse aquí.

/root no hay que confundirlo con /, una cosa es el directorio raíz o *root* y otra muy diferente */root*. En este caso, se puede asemejar a un */home* pero es exclusivo para el usuario *root* o usuario administrador.

/srv almacena ficheros y directorios relativos a servidores que tienes instalados en el sistema, como Web, FTP, CVS, etc.

/sys junto con */dev* y */proc*, es otro de los especiales. Y como */proc*, realmente no almacena nada, sino que es una interfaz también. En este caso, son ficheros virtuales con información del Kernel e incluso, se pueden emplear algunos de sus ficheros para configurar ciertos parámetros del Kernel.

/tmp es el directorio para ficheros temporales de todo tipo. Es empleado por los usuarios para almacenar de forma temporal ciertos ficheros o incluso para almacenar Caché o ciertos ficheros volátiles de navegadores Web, etc. No obstante, hay otro directorio para lo mismo en */var/tmp*.

/var se trata de un directorio con directorios y ficheros que suelen crecer de tamaño, como bases de datos, *logs*, etc. Es precisamente los *logs* o registros del sistema por lo que es más popular este directorio, y allí encontrarás muchísima información de todo lo que ocurre en el sistema: */var/logs/*. Dentro de dicho directorio encontrarás separados por directorios, los *logs* de multitud de Software, incluido el sistema.

/usr son las siglas de *User System Resources*, y actualmente almacena ficheros de solo lectura relativos a utilidades del usuario, como los paquetes que instalamos mediante el gestor de paquetes en nuestra distribución. Dentro hay como una jerarquía de árbol de directorios vistos hasta ahora (casi todos) como si de un segundo nivel se tratase. Vas a encontrar */usr/bin*, */usr/lib*, */usr/sbin*, */usr/src,* etc., que por lo dicho anteriormente y sus nombres, es intuitivo saber lo que almacenan. Por ejemplo en /usr/src es donde permanecerán los ficheros de código fuente.

Ten en cuenta que no todas las distribuciones de Linux siguen este esquema y puede haber pequeñas variaciones, pero si se adaptan al estándar, no tendrás problemas al navegar por la estructura de archivos.

Rutas Absolutas y Relativas cuando se empieza a manejar un intérprete de comandos, una de las cosas que más cuesta es acostumbrarte a encontrar y hacer referencia a elementos del sistema de ficheros. Mientras que en un entorno gráfico tenemos que hacer Clic en carpetas y subcarpetas hasta llegar al elemento deseado, en el intérprete de comandos tendremos que conseguir lo mismo, pero indicando el lugar mediante una cadena de texto compuesta por los nombres de las carpetas que hay que recorrer hasta el lugar donde se encuentra el elemento deseado. Según el sistema cada nombre de carpeta se separa por un carácter especial, que en Linux es la diagonal (/).

Estas rutas serán usadas por los comandos para saber dónde encontrar los elementos sobre los que se tiene que realizar la acción correspondiente[86]. Hay dos formas de utilizar rutas, una es de forma absoluta y la otra de forma relativa. Vamos a explicar la diferencia a continuación:

Rutas Absolutas el sistema de ficheros es una estructura jerárquica que en el caso de Linux tiene una raíz que se indica cuando se pone solamente el carácter diagonal / . En la raíz están los directorios principales del sistema que a su vez tendrán subdirectorios en su interior. Cuando quiero indicar dónde se encuentra un elemento usando una ruta absoluta, tendré que indicar todos los directorios por los que hay que pasar empezando desde la raíz del sistema. O lo que es lo mismo, siempre empezarán por /. Veamos algunos ejemplos:

```
/etc/apt/sources.list
/var/log/syslog
/home/alumno/.bashrc
/usr/bin/
```

[86] Por ejemplo, si quiero posicionarme en un directorio determinado, utilizaré el comando cd y para indicar el sitio adonde quiero ir usaré una ruta, por ejemplo cd /home/. El comando cp copia elementos, en este caso necesitaremos dos rutas una para el origen (elemento que quiero copiar) y otra para el destino (elemento nuevo que voy a crear o lugar donde voy a dejar las copias). Por lo tanto podría poner:
cp /etc/passwd /home/copia_passwd.

estas rutas suelen ser bastante largas, pero tienen como ventaja que funcionan siempre, independientemente del lugar desde el que se ejecute la orden[87].

Rutas Relativas las rutas relativas indican el camino para encontrar un elemento, pero basándonos en el directorio desde el que se ejecuta la orden. Son mucho más cortas que las absolutas, pero para saber si son correctas o no, tenemos que saber siempre desde dónde se han utilizado.

Un atajo fundamental para la construcción de rutas relativas es conocer que al escribir .. en la ruta hace referencia al directorio padre. Por lo tanto si ejecuto:

```
$ cd ..
```

estoy dando la orden de cambiar de directorio al padre del actual, es decir, al que está justo antes en la estructura jerárquica. El único elemento que no tiene padre es la propia raíz del sistema (/).

Las rutas relativas harán referencia a un elemento que se encuentre en el directorio desde el que ejecutamos la orden, o usará los dos puntos para ascender a directorios superiores. Siempre que sean correctos, podemos combinarlos de la forma que necesitemos separando cada directorio por una diagonal. Por ejemplo una ruta correcta podría ser: ../../fotos/personales/

Permisos de Archivos y Directorios GNU/Linux, al ser un sistema diseñado fundamentalmente para trabajo en red, la seguridad de la información que almacenamos en nuestros equipos (y no se diga en los servidores) es fundamental, ya que muchos usuarios tendrán o podrán tener acceso a parte de los recursos de Software (tanto a aplicaciones, como a información) y Hardware que están gestionados en estos equipos de cómputo. ¿Ahora podemos ver la necesidad de un sistema de permisos?

En GNU/Linux, los permisos o derechos que los usuarios pueden tener sobre determinados archivos contenidos en él se establecen en tres niveles claramente diferenciados. Estos tres niveles son los siguientes:

- Permisos del propietario.

[87] Es muy recomendable utilizar la facilidad que brinda el BASH de completar el nombre de un elemento del sistema de ficheros pulsando la tecla tabulador. Ahorrará mucho tiempo y errores.

- Permisos del grupo.
- Permisos del resto de usuarios (o también llamados "los otros").

Para tener claros estos conceptos, en los sistemas en red siempre existe la figura del administrador, superusuario o root. Este administrador es el encargado de crear y dar de baja a usuarios, así como también, de establecer los privilegios que cada uno de ellos tendrá en el sistema. Estos privilegios se establecen tanto para el directorio de trabajo (Home) de cada usuario como para los directorios y archivos a los que el administrador decida que el usuario pueda acceder.

Permisos del propietario el propietario es aquel usuario que genera o crea un archivo/carpeta dentro de su directorio de trabajo, o en algún otro directorio sobre el que tenga derechos. Cada usuario tiene la potestad de crear, por defecto, los archivos que quiera dentro de su directorio de trabajo. En principio, él y solamente él será el que tenga acceso a la información contenida en los archivos y directorios que hay en su directorio trabajo o Home -bueno, no es del todo cierto esto, ya que el usuario *root* siempre tiene acceso a todos los archivos y directorios del sistema-.

Permisos del grupo lo más normal es que cada usuario pertenezca a un grupo de trabajo. De esta forma, cuando se gestiona un grupo, se gestionan todos los usuarios que pertenecen a éste. Es decir, es más fácil integrar varios usuarios en un grupo al que se le conceden determinados privilegios en el sistema, que asignar los privilegios de forma independiente a cada usuario.

Permisos del resto de usuarios por último, también los privilegios de los archivos contenidos en cualquier directorio, pueden tenerlos otros usuarios que no pertenezcan al grupo de trabajo en el que está integrado el archivo en cuestión. Es decir, a los usuarios que no pertenecen al grupo de trabajo en el que está el archivo, pero que pertenecen a otros grupos de trabajo, se les denomina resto de usuarios del sistema.

¿cómo puedo identificar todo esto? sencillo, abre una terminal y escribe lo siguiente:

$ ls -l

entregará varias salidas como esta:

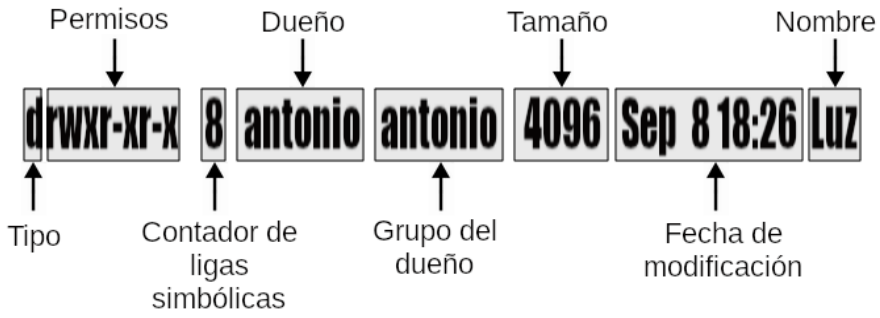


Figura 4: Estructura de permisos en la salida de: ls -l

Veamos por partes: El primer carácter al extremo izquierdo, representa el tipo de archivo, los posibles valores para esta posición son los siguientes:

- \- Archivo
- d Directorio
- l Liga simbólica
- s Socket (paso de datos entre dos procesos)
- p Pipe
- c Dispositivo de carácter (comunicación de Hardware)
- b Dispositivo de bloque (para el control de dispositivos)

Los siguientes 9 restantes, representan los permisos del archivo y deben verse en grupos de 3 y representan:

- \- Sin permiso
- r Permiso de lectura
- w Permiso de escritura

- x Permiso de ejecución

Los tres primeros representan los permisos para el propietario del archivo, los tres siguientes son los permisos para el grupo del archivo y los tres últimos son los permisos para el resto del mundo u otros.

Luego viene el contador de ligas simbólicas, el dueño del archivo, grupo al que pertenece, el tamaño en Bytes, la fecha de última modificación y finalmente el nombre del archivo o directorio.

Permisos Especiales Aún hay otro tipo de permisos que hay que considerar. Se trata del Bit de permisos SUID (Set User ID), el Bit de permisos SGID (Set Group ID) y el Bit de permisos de persistencia (Sticky Bit).

Setuid el Bit setuid es asignable a ficheros ejecutables, y permite que cuando un usuario ejecute dicho fichero, el proceso adquiera los permisos del propietario del fichero ejecutado. El ejemplo más claro de fichero ejecutable y con el Bit setuid es:

```
$ su
```

Podemos ver que el Bit está asignado como "s" en:

```
$ ls -l /bin/su
```

Para asignar[88] este Bit a un fichero seria:

```
# chmod u+s /bin/su
```

Y para quitarlo:

```
# chmod u-s /bin/su
```

[88] Debemos utilizar este Bit con extremo cuidado ya que puede provocar una escalada de privilegios en nuestro sistema.

Setgid el Bit setid permite adquirir los privilegios del grupo asignado al fichero, también es asignable a directorios. Esto será muy útil cuando varios usuarios de un mismo grupo necesiten trabajar con recursos dentro de un mismo directorio.

Para asignar este Bit hacemos lo siguiente:

```
$ chmod g+s /carpeta_compartida
```

Y para quitarlo:

```
$ chmod g-s /carpeta_compartida
```

Sticky este Bit suele asignarse en directorios a los que todos los usuarios tienen acceso, y permite evitar que un usuario pueda borrar ficheros/directorios de otro usuario dentro de ese directorio, ya que todos tienen permiso de escritura.

Podemos ver que el Bit está asignado como "t" en el directorio */tmp*.

Para asignar este Bit hacemos lo siguiente:

```
# chmod o+t /tmp
```

Y para quitarlo:

```
# chmod o-t /tmp
```

Entrada y Salida Estándar los procesos pueden abrir archivos a discreción, pero la mayor parte de los procesos esperan a que estén abiertos tres descriptores de archivos (numerados 0, 1 y 2) cuando inician. Estos descriptores se conocen como entrada estándar (0), salida estándar (1) y error estándar (2). Es común que los tres estén abiertos en la terminal del usuario. Así, el programa puede leer lo que el usuario teclea leyendo la entrada estándar, y puede enviar salidas a la pantalla del usuario escribiendo en la salida estándar. El descriptor de archivo de error estándar también está abierto para escritura, y se usa para los mensajes de error.

Standard Input la Entrada estándar, en inglés *standard input* (mejor conocido como *stdin*) es el mecanismo por el cual un usuario le indica a los programas la información que estos deben procesar. Por omisión, el teclado es la entrada estándar. La entrada estándar representa los datos que necesita una aplicación para funcionar, como por ejemplo un archivo de datos o información ingresada desde la terminal y es representado en la terminal como el tipo 0.

Standard Output la Salida estándar, en inglés *standard output* (mejor conocido como *stdout*) es el método por el cual el programa puede comunicarse con el usuario. Por omisión, la salida estándar es la pantalla donde se ejecutaron las instrucciones. La salida estándar es la vía que utilizan las aplicaciones para mostrarte información, allí podemos ver el progreso o simplemente los mensajes que la aplicación quiera darte en determinado momento y es representado en la terminal como el tipo 1.

Standard Error por último existe un flujo conocido como Error estándar, en inglés *standard error output* (mejor conocido como *stderr*) que es utilizado por las instrucciones para desplegar mensajes de error que surjan durante el transcurso de su ejecución. Al igual que *stdout*, el error estándar será la pantalla donde se procesaron las instrucciones. El error estándar es la forma en que los programas te informan sobre los problemas que pueden encontrarse al momento de la ejecución y es representado en la terminal como el tipo 2.

Redirección Mediante Pipe las tuberías (Pipe) unen la salida estándar de un comando con la entrada estándar de otro, es decir, la salida de un comando se emplea como entrada del siguiente. Para ello se emplea el símbolo pipe "|". El uso de tuberías evita la generación constante de archivos intermedios reduciendo el tiempo de procesamiento.

Redirección Hacia el Dispositivo Nulo en GNU/Linux, */dev/null* es un archivo especial al que se envía cualquier información que quiera ser descartada. Aunque al principio no lo parezca, el uso del dispositivo nulo es muy útil.

10.2 Interfaz de Usuario

Tanto el programador como el usuario de Linux manejan principalmente el conjunto de programas de sistemas que se ha escrito y están disponibles para ejecutarse. Estos programas efectúan llamadas al sistema operativo necesarias para apoyar su función, pero las llamadas al sistema en sí están contenidas dentro del programa y no tienen que ser obvias para el usuario.

GNU/Linux puede funcionar tanto en entorno gráfico[89] (Graphical User Interface, GUI) como en modo línea de comandos (Command-Line Interface, CLI) también conocida como consola o *Shell*. La consola es común en distribuciones para servidores, mientras que la interfaz gráfica está orientada al usuario final del hogar, como empresarial.

Los entornos de escritorio pertenecen a la interfaz gráfica, son un conjunto de programas conformado por ventanas, íconos, imágenes y muchas aplicaciones que facilitan el uso de la computadora. Los entornos de escritorio más populares en GNU/Linux son: GNOME, KDE, LXQt, LXDE, Xfce, Unity, MATE, Cinnamon, Pantheon, Deepin, Budgie, PIXEL, Enlightenment, Trinity, Moksha, Ukui, etc. Dependiendo de la distribución se pueden tener uno o más escritorios instalados, por ejemplo en Debian GNU/Linux están disponibles los más usados y si el usuario los instala, puede decidir al iniciar sesión cuál usar.

10.2.1 Interfaz Gráfica de Usuario

La interfaz gráfica de usuario es un tipo de visualización que permite al usuario elegir comandos, iniciar programas y ver listas de archivos y otras opciones utilizando las representaciones visuales (iconos) y las listas de elementos del menú. Las selecciones pueden activarse bien a través del teclado o con el ratón.

Para los autores de aplicaciones, las interfaces gráficas de usuario ofrecen

[89] Un servidor de pantalla en GNU/Linux es un programa que es responsable de la coordinación de entrada y salida de sus clientes, hacia y desde en resto del sistema operativo, y entre el Hardware y el sistema operativo. El servidor de visualización proporciona el marco para un entorno gráfico para que se pueda utilizar el Mouse y el teclado para interactuar con las aplicaciones. El servidor de pantalla se comunica con sus clientes a través del protocolo del servidor de pantalla como: X11, Wayland o Mir. El servidor de visualización es un componente clave en cualquier interfaz gráfica de usuario, específicamente el sistema de ventanas. No debemos confundir el servidor de visualización con el entorno de escritorio. El entorno de escritorio utiliza un servidor de pantalla debajo.

un entorno que se encarga de la comunicación con la computadora. Esto hace que el programador pueda concentrarse en la funcionalidad, ya que no está sujeto a los detalles de la visualización ni a la entrada a través del ratón o del teclado. También permite a los programadores crear programas que realicen de la misma forma las tareas más frecuentes, cómo guardar un archivo, porque la interfaz proporciona mecanismos estándar de control como ventanas y cuadros de diálogo. Otra ventaja es que las aplicaciones escritas para una interfaz gráfica de usuario son independientes de los dispositivos: a medida que la interfaz cambia para permitir el uso de nuevos dispositivos de entrada y salida, como un monitor de pantalla grande o un dispositivo óptico de almacenamiento, las aplicaciones pueden utilizarlos sin necesidad de cambios.

¿Qué es un Entorno de Escritorio? un entorno de escritorio es un conjunto de Software para ofrecer al usuario de una computadora una interacción amigable y cómoda. El Software es una solución completa de interfaz gráfica de usuario, ofrece iconos, barras de herramientas, carpetas, fondos de pantalla y Widgets de escritorio, e integración entre aplicaciones con habilidades como, arrastrar y soltar. En general cada entorno de escritorio se distingue por su aspecto y comportamiento particular, aunque algunos tienden a imitar características de escritorios ya existentes. El primer entorno moderno de escritorio que se comercializó fue desarrollado por Xerox en los años 80. Actualmente el entorno más conocido es el ofrecido por la familia Windows aunque existen otros como los de: Macintosh (Classic y Cocoa) y de código abierto como: GNOME, KDE, LXQt, LXDE, Xfce, Unity, MATE, Cinnamon, Pantheon, Deepin, Budgie, PIXEL, Enlightenment, Trinity, Moksha, Ukui, etc.

¿Qué son los Gestores de Ventanas? un gestor de ventanas o en inglés Windows Manager, es un programa que controla la ubicación y apariencia de las aplicaciones bajo el sistema X Windows. Las computadoras suelen ofrecer una interfaz gráfica de usuario que facilita la interacción con el sistema operativo. Las plataformas Windows y Macintosh ofrecen métodos de visualización y control de las ventanas e interacción con las aplicaciones, estandarizados por sus vendedores. En cambio el sistema gráfico X Windows, popular en el ámbito de sistemas Unix y similares, como GNU/Linux, permite al usuario escoger entre varios gestores según sus gustos o necesidades. Los

gestores de ventanas difieren entre sí de muchas maneras, incluyendo apariencia, consumo de memoria, opciones de personalización, escritorios múltiples o virtuales y similitud con ciertos entornos de escritorio ya existentes. Estos se dividen en 3 tipos, que son los siguientes:

- Stacking: Aquellos que imitan las apariencias y funcionalidades de Windows y Mac OS X, por ende, gestionan las ventanas como pedazos de papel en un escritorio, que pueden ser apiladas unas sobre otras.
- Tiling: Aquellos de tipo "mosaico" donde las ventanas no se superponen, y donde suelen hacerse un uso muy extenso de atajos de teclado, y se obtiene una menor dependencia del uso del ratón.
- Dynamics: Aquellos que permiten alterar dinámicamente el diseño de las ventanas entre mosaicos o flotantes.

Las acciones asociadas al gestor de ventanas suelen ser, abrir, cerrar, minimizar, maximizar, mover, escalar y mantener un listado de las ventanas abiertas. Es también muy común que el gestor de ventanas integre elementos como: el decorador de ventanas, un panel, un visor de escritorios virtuales, iconos y un tapiz.

Entornos de Escritorios más Conocidos:

KDE (https://kde.org)
proyecto que fue iniciado en octubre de 1996 por el programador alemán Matthias Ettrich, quien buscaba crear una interfaz gráfica unificada para sistemas Unix. En sus inicios imitó a CDE (Common Desktop Environment), un entorno de escritorio utilizado por varios Unix. Este es un entorno de escritorio con multitud de aplicaciones e infraestructura de desarrollo para diversos sistemas operativos como GNU/Linux, Mac OS X, Windows, etc. Los principales componentes de Software elaborados por KDE se agrupan bajo el nombre KDE Frameworks, KDE Plasma y KDE Applications. Las aplicaciones KDE están traducidas a aproximadamente 88 idiomas y están construidas con los principios de facilidad de uso y de accesibilidad moderna en mente y funcionan de forma completamente nativa en GNU/Linux, BSD, Solaris, Windows y Mac OS X.

GNOME (https://www.gnome.org)
este proyecto fue iniciado por los programadores mexicanos Miguel de Icaza y Federico Mena, forma parte oficial del proyecto GNU. Nació como una alternativa a KDE bajo el nombre de GNU Network Object Model Environment (Entorno de Modelo de Objeto de Red GNU). Actualmente, GNOME se está traduciendo a 193 idiomas. Está disponible en las principales distribuciones GNU/Linux, incluyendo Fedora, Debian, Ubuntu, Manjaro Linux, Red Hat Enterprise Linux, SUSE Linux Enterprise, CentOS, Oracle Linux, Arch Linux, Gentoo, SteamOS, entre otras. También, se encuentra disponible en Solaris, un importante sistema operativo UNIX y en Sistemas operativos Unix-like como FreeBSD.

Xfce (https://www.xfce.org)
es un entorno de escritorio libre para sistemas tipo Unix como GNU/Linux, BSD, Solaris y derivados. Su objetivo es ser rápido y ligero, sin dejar de ser visualmente atractivo y fácil de usar. Consiste en varios componentes empaquetados por separado que en conjunto proporcionan la funcionalidad completa del entorno de escritorio, pero se pueden seleccionar por separado para que el usuario pueda adaptar el ambiente de trabajo a sus necesidades. Puede ser instalado en varias plataformas como: Linux, NetBSD, FreeBSD, OpenBSD, Solaris, Cygwin y MacOS X, sobre x86, PPC, Sparc, Alpha.

LXDE (https://lxde.org)
es un entorno de escritorio libre para Unix y otras plataformas POSIX[90], como Linux o BSD. El nombre corresponde a "Lightweight X11 Desktop Environment", que en español significa Entorno de escritorio X11 ligero. Es un proyecto que apunta a entregar un nuevo entorno de escritorio ligero y rápido. No está diseñado para ser tan complejo como KDE o GNOME, pero es bastante usable y ligero, y mantiene una baja utilización de recursos y energía. A diferencia de otros ambientes de escritorio, los componentes no se integran firmemente. Al contrario, los componentes son independientes, y

[90] Significa Portable Operating System Interface. Consiste en una familia de estándares especificadas por la IEEE con el objetivo de facilitar la interoperabilidad de sistemas operativos. Además, POSIX establece las reglas para la portabilidad de programas. Por ejemplo, cuando se desarrolla Software que cumple con los estándares POSIX existe una gran probabilidad de que se podrá utilizar en sistemas operativos del tipo Unix. Si se ignoran tales reglas, es muy posible que el programa o librería funcione bien en un sistema dado pero que no lo haga en otro.

cada uno de ellos se puede utilizar independientemente con muy pocas dependencias. Usa Openbox como gestor de ventanas predeterminado y apunta a ofrecer un escritorio ligero y rápido basado en componentes independientes que pueden ser utilizados en otros entornos.

LXQt (https://lxqt.github.io)
es un entorno de escritorio libre y de código abierto para Linux, resultado de la fusión entre los proyectos LXDE y Razor-qt. LXQt conjuga la filosofía de LXDE con las librerías QT, usa el gestor de ventanas del escritorio Razor-qt, es un escritorio muy liviano y es considerado por muchos como el sucesor de LXDE.

Gestores de Ventanas más Conocidos:

Enlightenment (https://www.enlightenment.org)
también conocido simplemente como E, es un gestor de ventanas X11 ligero para UNIX y GNU/Linux. Uno de sus objetivos es llegar a ser un entorno de escritorio completo. Es muy configurable y muy atractivo visualmente. Durante un tiempo fue el gestor de ventanas de GNOME.

IceWM (https://ice-wm.org)
es un gestor de ventanas para el X Windows System gráfico de infraestructura escrito por Marko Macek. Se ha codificado desde cero en C++ y es liberado bajo GNU. IceWM es ligero y personalizable. Se puede configurar a partir de archivos de texto almacenados en un directorio Home del usuario, haciendo fácil la personalización y copia de configuraciones. Posee soporte oficial para menús de GNOME y KDE previamente disponible como un paquete separado.

Windows Maker (https://www.windowsmaker.org)
es un popular gestor de ventanas para X Windows System diseñado para emular NeXT del GUI como OpenStep compatible, ha sido descrito como "uno de los más útiles y universales gestores de ventanas disponibles. Windows Maker tiene la reputación de ser rápido, eficiente y altamente estable y es muy popular entre las soluciones de código abierto para su uso tanto en nuevas como en viejas máquinas. Como con la mayoría de gestores de ventanas, soporta un montón de temas disponibles.

Fluxbox (http://fluxbox.org)
es un gestor de ventanas X basado en Blackbox 0.61.1.Su objetivo es ser ligero y personalizable, y cuenta con un apoyo mínimo de iconos gráficos. Su interfaz de usuario sólo tiene una barra de tareas y un menú al que se puede acceder pulsando con el botón derecho sobre el escritorio. Todas las configuraciones básicas están controladas por ficheros de texto. Fluxbox puede mostrar algunos Eye Candies: colores, gradientes, bordes y una que otra apariencia básica. Las versiones recientes soportan esquinas redondeadas y elementos gráficos. Fluxbox también tiene varias características de las cuales Blackbox carece, incluyendo ventanas con pestañas y un título configurable.

Openbox (http://openbox.org/new/)
es un famoso gestor de ventanas libre para el sistema de ventanas X, licenciado bajo la GNU General Public License. Openbox fue originalmente derivado de Blackbox 0.65.0, pero ha sido totalmente reescrito en el lenguaje de programación C y desde la versión 3.0 no se basa en ningún código de Blackbox. Su sistema de menú tiene un método para utilizar los menús dinámicos.

Otros existen muchos otros gestores de ventanas que pudiéramos mencionar, tales como: 9wm, Awesome, AfterStep, Scwm, Blackbox, Bspwm, Byobu, Cinnamon, Cwm, Deepin, Dwm, Fvwm, Icewm, Jwm, Kahakai, Lumina, Qtile, Wmii, WindowLab, Ratpoison, Sawfish, Sway, wm2, Wmx, StumpWM, Twm, Waimea, Xmonad, I3, E16.

10.2.2 Línea de Comandos y Órdenes

La mayoría de los usuarios de ordenadores de hoy sólo están familiarizados con la interfaz gráfica de usuario o GUI, los vendedores y los expertos les han enseñado que la interfaz de línea de comandos o CLI es una cosa espantosa del pasado. Es una pena, porque una buena interfaz de línea de comandos es una maravillosa y expresiva forma de comunicarse con el ordenador, muy parecida a lo que el lenguaje escrito es para los seres humanos. Se ha dicho que "las interfaces gráficas de usuario hacen fáciles las tareas fáciles, mientras que las interfaces de línea de comandos hacen posibles las tareas difíciles" y eso es muy cierto aún hoy.

Dado que Linux fue desarrollado desde la familia de sistemas operativos Unix, comparte la misma rica herencia de herramientas de línea de comandos

que Unix. Unix saltó a la fama en los primeros años ochenta (aunque fue desarrollado una década antes), antes de que se extendiera la adopción de las interfaces gráficas de usuario, y por eso, se desarrolló una amplia interfaz de línea de comandos en su lugar. De hecho, una de las razones más potentes para que los primeros que utilizaron Linux lo eligieran sobre, digamos, Windows NT, era la poderosa interfaz de línea de comandos que hacía las "tareas difíciles posibles".

Emuladores de Terminal Cuando utilizamos una interfaz gráfica de usuario, necesitamos otro programa llamado emulador de terminal para interactuar con el Shell. Si buscamos en nuestros menús de escritorio, probablemente encontraremos uno. KDE usa Konsole y GNOME usa Gnome-terminal, aunque es probable que se llame simplemente "Terminal" en nuestro menú. Hay muchos otros emuladores de terminal disponibles para Linux, pero todos hacen básicamente lo mismo; nos dan acceso al Shell.

El Shell Los programas, tanto los escritos por el usuario como los de sistemas, normalmente se ejecutan con un intérprete de órdenes. El intérprete de órdenes en Linux es un proceso de usuario como cualquier otro y recibe el nombre de Shell (concha o cáscara) por que rodea al núcleo del sistema operativo.

El Intérprete de Órdenes de Consola o Shell es un programa informático, cuya función consiste en interpretar órdenes, y un lenguaje de consola. El más conocido es Bash[91]. Es una Shell de Unix compatible con POSIX[92] y el intérprete de comandos por defecto en la mayoría de las dis-

[91] Su nombre es un acrónimo de Bourne-again Shell ("Shell Bourne otra vez"), haciendo un juego de palabras (Born-Again significa "nacido de nuevo") sobre la Bourne Shell (sh), que fue uno de los primeros intérpretes importantes de Unix.

[92] Significa Portable Operating System Interface. Consiste en una familia de estándares especificadas por la IEEE con el objetivo de facilitar la interoperabilidad de sistemas operativos. Además, POSIX establece las reglas para la portabilidad de programas. Por ejemplo, cuando se desarrolla Software que cumple con los estándares POSIX existe una gran probabilidad de que se podrá utilizar en sistemas operativos del tipo Unix. Si se ignoran tales reglas, es muy posible que el programa o librería funcione bien en un sistema dado pero que no lo haga en otro.

Sin ir más lejos, examinemos en la última versión de la especificación IEEE 1003 (designación formal de los estándares POSIX). Al buscar la referencia sobre el comando du, podemos ver que solamente las opciones -a, -H, -k, -L, -s, y -x son obligatorias. Otras, tales

tribuciones GNU/Linux, además de Mac OS. También se ha llevado a otros sistemas como Windows y Android.

Algunas alternativas a Bash son: SH (Bourne Shell), TCSH/CSH (C Shell), KSH (Korn Shell), Fish (Friendly Interactive Shell), ZSH (Z Shell), TSCH (TS Shell), Dash (Debian Almquist Shell), DSH (Distributed Shell).

El Shell indica que está listo para aceptar otra orden exhibiendo una señal de espera (*Prompt*) y el usuario teclea una orden en una sola línea. En el Bourne Shell y sus derivados como Bash el *Prompt* que nos permite escribir los diferentes comandos, generalmente termina con el carácter:

- $ para usuario sin privilegios
- # para el administrador, conocido como *root*

Sintaxis Estándar de Comandos Se ha definido un estándar para comandos POSIX (Portable Operating System Interfaz) por la IEEE, pero no todos los comandos la siguen, Muchos comandos definidos por POSIX tienen una forma moderna y una sintaxis obsoleta, por compatibilidad con sistemas viejos. Por ejemplo el estándar especifica que las opciones de los comandos (no así, los operandos) pueden aparecer en cualquier orden, pero algunos comandos pueden requerir que las opciones aparezcan en un orden particular y muchas veces esto no está mencionado explícitamente el la página del manual del comando.

Un comando consiste en una sucesión de palabras separadas por espacios en blanco. La primera palabra es el nombre del comando y las palabras subsecuentes son sus argumentos u opciones. Los operandos consisten de las opciones del programa, si hay alguna, seguidas de sus operandos, si hay alguno. Las opciones controlan o modifican lo que el comando hace. Los operadores especifican nombres de trayectorias o con lo que va a trabajar el comando.

Las opciones se especifican con una sucesión de palabras. Cada palabra es un grupo de opciones o una opción argumento. Las opciones generalmente se denotan por letras. Se pueden escribir en cualquier orden y se pueden combinar varias opciones en un solo grupo (siempre y cuando no reciban

como -c y -h, aparecen en la versión de du provista por el proyecto GNU. Esto significa que si utilizamos estas últimas en un Script desarrollado en Linux e intentamos hacerlo funcionar en FreeBSD o AIX, es muy probable que no funcione como esperamos.

ningún argumento). Cada grupo de opciones está precedido por un guión (-). Por ejemplo, los comandos:

```
$ ls -al
$ ls -l -a
```

son equivalentes e indican que el comando *ls* (list archives) sea llamado con las opciones *a* (all files) y *l* (long listing).

Hay otras convenciones comunes para especificar argumentos:

- Dos guiones (- -) indica el final de las opciones. Esto es útil cuando quieres pasar argumentos que empiecen con un - a un comando, por ejemplo:

  ```
  $ rm - - -miArchivo
  ```

es una forma de indicarle que el archivo es *-miArchivo*, también podemos usar:

```
$ rm ./-miArchivo
```

- Un solo guión representa a la entrada estándar en un contexto en que el programa espera una trayectoria. Por ejemplo, el comando:

  ```
  $ diff - miArchivo
  ```

encuentra las diferencias entre la entrada estándar y el archivo *miArchivo*

Metacarácter o Shell Globbing los metacaracteres o Shell Globbing son caracteres que tienen un significado especial en la línea de comandos, para usar estos caracteres como caracteres ordinarios en la línea de comandos, debes marcarlos de manera especial para que el Shell no los interprete. Por ejemplo:

- La diagonal inversa (\) siempre sirve como carácter de escape para uno o más caracteres que le suceden, lo cual le da a esos caracteres un significado especial. Si un carácter es un metacaracter, el significado especial es el carácter mismo. Por ejemplo \\ usualmente significa una solo \ y \$ el signo de pesos. En estos casos, la diagonal inversa le quita su significado a los caracteres.

- Cuando un texto se encierra entre comillas (" "), la mayoría de los de los metacaracteres que estén en dicho texto aon tratados como caracteres normales, excepto por $, que generalmente indica sustituciones a realizar.

- Otra forma de citar muy similar a la anterior, pero más fuerte es usar comillas sencillas (' ') ya que ni siquiera el carácter $ es interpretado

También existen otros metacaracteres que son comodines que el sistema permite usar para especificar los nombres de archivos que satisfacen el filtro especificado a la hora de buscar, eliminar o filtrar nombres de archivo, estos metacaracteres son: *, ?, [ ] y [^].

- ***** Se utiliza para reemplazar cero o más caracteres. Puede ser sustituido por cualquier cadena de caracteres, ejemplo:

  $ ls a*.pdf

- **?** Sustituye un carácter cualquiera, ejemplo:

  $ ls a?chivo.pdf

- [ ] Se usa para definir rangos o conjuntos de caracteres a localizar, para definir los rangos se debe usar el guión -, si son varios caracteres se separan por coma, ejemplo:

  $ ls [Aa]rchivo[0-9].pdf

- [^] o **[!]**Este caso es contrario al anterior, este representa que se busque algo exceptuando lo que se encuentra entre los corchetes, también trabaja con rangos, ejemplo:

  $ls [^A]rchivo.pdf

Cambiar de Usuario en Linux El comando *su* (Switch User) se utiliza para cambiar de usuario cuando estamos dentro de la consola de Linux, ejemplo:

```
$ su antonio
```

si delante del usuario ponemos - nos abrirá una nuevo Shell con las preferencias del usuario al que cambiemos, por ejemplo:

```
$ su - administracion
```

Por otro lado, si usamos el comando *su* sin usuario, nos permitirá ingresar como el usuario administrador del sistema *root* (por defecto), pidiendo la clave o *Password* de dicho usuario, ejemplo:

```
$ su
```

El usuario root en GNU/Linux es el usuario que tiene acceso administrativo al sistema. Los usuarios normales no tienen este acceso por razones de seguridad. Sin embargo, en múltiples sistemas derivados de Debian GNU/Linux no incluye el usuario root (por ejemplo en todos los derivados de Ubuntu). En su lugar, se da acceso administrativo a usuarios individuales, que pueden utilizar la aplicación *sudo* para realizar tareas administrativas. La primera cuenta de usuario que creó en su sistema durante la instalación tendrá, de forma predeterminada, acceso a *sudo*.

Cuando se necesite ejecutar una aplicación que requiere privilegios de administrador, *sudo* le pedirá que escriba su contraseña de usuario normal. Esto asegura que aplicaciones incontroladas no puedan dañar su sistema, y sirve como recordatorio de que está a punto de realizar acciones administrativas que requieren que tenga cuidado.

Para usar *sudo* en la línea de comandos, simplemente escriba *sudo* antes del comando que desea ejecutar, *sudo* le pedirá su contraseña[93]:

```
$ sudo apt update
```

[93] *Sudo* recordará su contraseña durante un periodo de tiempo (predeterminado a 15 minutos). Esta característica se diseñó para permitir a los usuarios realizar múltiples tareas administrativas sin tener que escribir su contraseña cada vez.

Trabajando en Línea de Comandos Las órdenes se pueden agrupar en varias categorías; la mayor parte de ellas están orientadas hacia archivos o directorios. Por ejemplo los programas de sistema que manipulan directorios son *mkdir* para crear un directorio nuevo, *rmdir* para eliminar un directorio, *cd* para cambiar el directorio actual a otro y *pwd* para visualizar el nombre de la ruta absoluta del directorio actual (de trabajo).

El programa *ls* lista los nombres de los archivos del directorio actual. Cualquiera de las más de 20 opciones de *ls* puede hacer que se exhiban también las propiedades de los archivos, por ejemplo, la opción *-l* pide un listado largo, que muestra el nombre de cada archivo, su dueño, la protección, su tamaño, la fecha y hora en que se creó. El programa *cp* crea un archivo nuevo que es una copia de uno ya existente. El programa *mv* cambia de lugar un archivo dentro del árbol de directorios. En la mayor parte de los casos, este movimiento sólo requiere un cambio de nombre de archivo, pero si es necesario el archivo se copia en su nueva posición y la copia vieja se elimina. Los archivos se eliminan con el programa *rm*.

Para mostrar el contenido de un archivo en la terminal, un usuario puede ejecutar *cat*. El programa *cat* toma una lista de archivos y los concatena, copiando el resultado en la salida estándar, que normalmente es la terminal. Claro que en la terminal el archivo podría exhibirse con demasiada rapidez como para leerse. El programa *more* exhibe el archivo pantalla por pantalla y hace una pausa hasta que el usuario teclea un carácter para continuar con la siguiente pantalla. El programa *head* exhibe sólo las primeras líneas del archivo; *tail* muestra las últimas líneas.

Estos son algunos de los programas que usa Linux, además hay editores de texto (*ed, sed, emacs, nano, vi*, etc.), compiladores (de C, C++, Java, Python, etc.), formateadores de texto (LaTeX*, troff*, etc.), programas para ordenar (*sort*) y comparar (*cmp, diff*) archivos, buscar patrones (*grep, awk*) y muchas otras actividades.

La ejecución de una orden se efectúa con una o más llamadas al sistema operativo. Por lo regular, el Shell ejecuta una pausa cuando se le pide ejecutar una orden, queda en espera a que ésta se termine de ejecutar. Existe una sintaxis sencilla (un signo & al final de la línea de órdenes) para indicar que el Shell no debe esperar hasta que termine de ejecutarse la orden. Una orden que deja de ejecutarse de esta manera mientras el Shell sigue interpretando órdenes subsecuentes es una orden en segundo plano, o que se ejecuta en segundo plano. Los procesos para los cuales el Shell sí espera, se ejecutan en

primer plano.

El Shell del sistema GNU/Linux ofrece un recurso llamado control de trabajos (y visualizarlos con los comandos *ps* o *top*) implementado especialmente en el núcleo. El control de trabajos permite transferir procesos entre el primer y segundo plano. Los procesos pueden detenerse y reiniciarse según diversas condiciones, como que un trabajo en segundo plano requiera entradas desde la terminal del usuario. Este esquema hace posible la mayor parte del control de procesos que proporcionan las interfaces de ventanas, pero no requiere Hardware especial. Cada ventana se trata como una terminal, y permite a múltiples procesos estar en primer plano (uno por ventana) en cualquier momento. Desde luego pueden haber procesos de segundo plano en cualquiera de las ventanas.

Algunos Comandos para Conocer el Sistema en el que Trabajamos
Siempre es una buena práctica saber que componentes de Hardware se tienen en el equipo en que corremos nuestro GNU/Linux, esto nos puede ayudar a lidiar problemas de compatibilidad cuando se trata de instalar paquetes, controladores en nuestro sistema.

El primer comando a ejecutar es aquel que nos dé información del sistema en el que trabajamos, este es *uname* con la bandera *-a*, que nos dará información de la versión del Kernel, la versión y distribución del sistema operativo y en qué tipo de Hardware es que estamos corriendo nuestro GNU/Linux, para ello usamos:

```
$ uname -a
```

Para conocer las características de nuestro CPU, podemos usar varios comandos, entre ellos está *lscpu*, este nos dará la arquitectura del sistema, el número de CPUs, Cores, el modelo de la familia de CPU, Cache del CPU, Threds, entre otros. Para usarlo escribimos:

```
$ lscpu
```

si sólo queremos conocer el número de cores del equipo, usamos:

```
$ nproc
```

Podemos visualizar la memoria total, usada, libre y Swap (de intercambio) de nuestro equipo, usando:

```
$ free -g
```

Para visualizar información sobre los discos duros, unidades de estado sólido, en nuestro equipo, podemos usar *lsblk* que nos reportará dicha información, para ello escribimos:

```
$ lsblk -a
```

o bien podemos usar el comando *df* o en modo administrador a comando *fdisk*, escribiendo:

```
$ df -h
# fdisk -l
```

Si necesitamos conocer la información sobre los controladores USB y todos los dispositivos que están conectados a ellos, usamos:

```
$ lsusb
```

En caso de necesitar conocer los dispositivos PCI que pueden incluir puertos USB, tarjetas gráficas, adaptadores de red, entre otros, más los dispositivos que están conectados a ellos usamos:

```
$ lspci
```

Si lo que deseamos es un listado detallado del Hardware de nuestro equipo de cómputo, entonces podemos usar cualquiera de estos comandos (que previamente deberemos instalar):

```
# lshw
# dmidecode
# hwinfo
```

Así como podemos conocer las características de nuestro equipo de cómputo, podemos también conocer todos los comandos (-c), alias (-a), comandos internos (-b), palabras reservadas Bahs (-k) y funciones de Bash (-A function) disponibles para el usuario, para ello usamos:

```
$ compgen -c
$ compgen -a
$ compgen -b
$ compgen -k
$ compgen -A function
```

Correr Múltiples Comandos en uno Solo Supongamos que se tienen que ejecutar varios comandos uno tras otro -sin conocer si el comando anterior fue exitoso para ejecutar el nuevo-, para este propósito se usa el separador ";", de esta manera se pueden ejecutar una serie de comandos en una línea, ejemplo:

```
$ mkdir tmp ; ls ; cd tmp ; ls
```

en este ejemplo se crea en directorio, luego visualiza el contenido del directorio, para luego cambiar de directorio y finalmente visualiza el contenido del directorio recién creado.

Si necesitamos ejecutar múltiples comandos en uno solo, sólo si el comando anterior fue exitoso, se usa el separador "&&", ejemplo:

```
$ mkdir tmp && ls && cd tmp && ls
```

Notemos que si ejecutamos:

```
$ mkdir tmp & ls & cd tmp & ls
```

el resultado obtenido en nada se parece a lo que esperamos, ya que los comandos pasarán a ser ejecutados en segundo plano "&" y sin orden alguno, por lo que el resultado no será el deseado.

Ya vimos cómo usar un solo comando para ejecutar varios comandos. Pero, ¿qué debo hacer en caso que el primer comando no se ejecute correctamente?, ¿deseo ejecutar el comando siguiente?. Podemos usar || para que si el primer comando falla se ejecute el segundo, pero si no falla no ejecutará el segundo comando, por ejemplo:

```
$ cd miDirectorio || mkdir miDirectorio && cd miDirectorio
```

También podemos combinar los comandos && y || los cuales se comportarán como el operador ternario de C y C++ (condición? expresión_verdadera; expresión_falsa):

```
$ comando1 && comando2 || comando3
```

por ejemplo:

```
$ [-f archivo.txt ] && echo "Archivo existe" || echo "Archivo
  no existe"
```

podemos combinar ;, && y || para correr múltiples comandos. Como por ejemplo:

```
$ sudo apt update && sudo apt upgrade && sudo apt clean
```

Instalar Paquetes Saber cómo instalar paquetes y programas en Linux y cualquiera de sus distribuciones es uno de los temas más complejos para los nuevos usuarios. El método más común para instalar un nuevo Software en Linux es través de la Terminal. No obstante, los comandos varían según el tipo de gestor de paquetes que posee el sistema.

Uno de los apartados que debes tener en cuenta antes de instalar un paquete o programa en Linux, es conocer qué es un sistema de gestión de paquetes. En términos simples, hace referencia a un conjunto de formatos de archivos y herramientas empleadas para actualizar, instalar o desinstalar algún Software en el sistema operativo[94]. En la actualidad, se divide en dos grandes sistemas de gestión de paquetes: Debian y Red Hat.

Distribuciones como Fedora, Mandriva, SuSE, CentOS, entre otros emplean el sistema de gestión de Red Hat, que se caracteriza por utilizar la extensión de archivos (*.rpm*). Por otro lado, las distribuciones como Ubuntu, Peppermint, Linux Mint, y muchos otros hacen uso del sistema *dpkg* de Debian, el cual se representa con la extensión (*.deb*) en sus archivos. Ambos sistemas de gestión trabajan de forma diferente para mantenerse activos en el dispositivo y, a su vez, gestionar las descargas.

Los procesos de instalación entre un sistema de gestión y otro difieren en diversos aspectos, especialmente en los comandos empleados. Por ello es fundamental repasar cuáles son los comandos utilizados para instalar un Software en Linux según el sistema de gestión de paquetes. En el caso de Debian GNU/Linux , puedes hacer uso del programa *apt-get* para instalar el programa deseado desde un repositorio[95]. Puedes escribir únicamente *apt* o utilizar *apt-get* en el comando.

Por otro lado, también utilizar el programa *dpkg* para la instalación de Software con extensión (*.deb*). Por otro lado, los sistemas basados en (*.rpm*) hacen uso de los comandos esenciales para las distribuciones de Linux Red Hat. Estos son *yum* y *dnf*. En el caso de *yum*, el usuario puede instalar

[94] El usuario root en GNU/Linux es el usuario que tiene acceso administrativo al sistema, los usuarios normales no tienen este acceso por razones de seguridad. Sin embargo, en múltiples sistemas no incluye el usuario root. En su lugar, se da acceso administrativo a usuarios individuales, que pueden utilizar la aplicación *sudo* para realizar tareas administrativas. La primera cuenta de usuario que se creó en su sistema durante la instalación tendrá de forma predeterminada acceso a *sudo*.

[95] Un repositorio es un lugar en la red que siempre está actualizado desde donde nuestra distribución de GNU/Linux puede buscar y descargar fácilmente todo tipo de programas y herramientas para su instalación en nuestro equipo.

o actualizar programas directamente desde el repositorio oficial de Linux, o bien desde un repositorio de terceros. Mientras que el comando *dnf* facilita la administración de programas.

Para Debian GNU/Linux usamos:

```
# apt install paquete
# dpkg -i paquete
```

En Ubuntu y derivados se usa:

```
$ sudo apt install paquete
$ sudo dpkg -i paquete
```

Para openSUSE:

```
$ sudo rpm -Uvh paquete
$ su zypper intall paquete
```

En Fedora se usa:

```
$ sudo rpm -Uvh paquete
$ su -c 'yum install paquete'
# yum install paquete
```

Para Mandriva:

```
$ sudo rpm -Uvh paquete
$ su urpmi *.rpm
```

En Gentoo, FreeBSD se usa:

```
# emerge -vq paquete
```

Para Red Hat, Fedora o CentOS se usa:

```
$ sudo yum install paquete
$ sudo rpm -i paquete
$ sudo dfn install paquete
# yum install paquete
# dfn install paquete
```

En Archn Manjaro Linux se usa:

```
# pacman -s paquete
```

Para paquetes SNAP:

```
$ snap install paquete
```

Instalar Paquetes en Debian y Derivados Una de las funciones del usuario administrador es hacer la instalación, actualización y borrado de paquetes, el comando más usado actualmente es apt (Advanced Packaging Tool, herramienta avanzada de empaquetado que es una interfase del paquete *apt-get*), es un programa de gestión de paquetes *.deb* creado por el proyecto Debian, *apt* simplifica en gran medida la instalación, actualización y eliminación de programas en GNU/Linux. Existen también programas que proporcionan estos mismos servicios como: *apt-get*, *aptitude*, *tasksel* y *dpkg*.

Una vez siendo el usuario *root*, podemos solicitar la descarga de las actualizaciones disponibles, usando:

```
# apt update
```

para conocer los paquetes que se necesitan actualizar, usamos:

```
# apt list –upgradeable
```

para actualizar los paquetes instalados en el sistema[96], usamos:

```
# apt upgrade
```

algunas veces ciertos paquetes ya no se usarán al actualizar el sistema, para borrarlos usamos:

```
# apt autoremove
```

para buscar paquetes que podemos instalar, usamos:

```
# apt search nombre
```

para conocer las características de un paquete a instalar, usamos:

```
# apt show nombre
```

para instalar uno o más paquetes, usamos:

[96] Podemos hacer la actualización en un solo comando, usando:

```
# apt update && apt upgrade && apt clean
```

```
# apt install paquete
```

para remover uno o más paquetes, usamos:

```
# apt purge paquete
```

al final, se solicita el borrado de los archivos *.deb* de los paquetes descargados (Caché de descarga), mediante:

```
# apt clean
```

podemos conocer todos los paquetes *.deb* instalados en el sistema, usando:

```
$ apt list –installed
```

o

```
$ dpkg -l
```

Existen otro tipo de paquetes para GNU/Linux que son multiplataforma como son: *Snap, Flatpak, Pyhon 3*. Para conocer si tenemos alguno de ellos instalados usamos:
para conocer todos los paquetes *Snap* instalados en el sistema, usamos:

```
$ snap list
```

para conocer todos los paquetes *Flatpak* instalados en el sistema, usamos:

```
$ flatpak list
```

para conocer todos los paquetes *python3* instalados en el sistema, usamos:

```
$ pip3 list
```

11 Bibliografía

Este texto es una recopilación de múltiples fuentes, mi aportación —si es que puedo llamarla así— es plasmarlo en este documento, en el que trato de dar coherencia a mi visión de los temas desarrollados.

En la realización de este texto se han revisado —en la mayoría de los casos indico la referencia, pero pude omitir varias de ellas, por lo cual pido una disculpa— múltiples páginas Web, artículos técnicos, libros, entre otros materiales bibliográficos, los más representativos y de libre acceso los pongo a su disposición en la siguiente liga:

Herramientas
http://132.248.181.216/Herramientas/

Referencias

[1] Proyectos de Software Sourceforge, http://sourceforge.net/ 17, 18

[2] GNU Operating System, http://www.gnu.org/ 17, 379, 389

[3] Google Code, http://code.google.com 18

[4] http://es.wikipedia.org/wiki/Linux 5

[5] The Linux Kernel Archives, http://www.Kernel.org/ 17

[6] Debian el Sistema Operativo Universal, http://www.debian.org 17

[7] http://es.wikipedia.org/wiki/Android 17

[8] http://www.gnu.org/philosophy/free-sw.es.html 17, 379

[9] http://es.wikipedia.org/wiki/Software_libre 17, 379

[10] http://www.hispalinux.es/SoftwareLibre 17, 379

[11] http://es.wikipedia.org/wiki/Software_propietario 377

[12] Diferentes Tipos de Licencias para el Software, http://www.gnu.org/licenses/license-list.html 4, 17, 379, 389

[13] FSF, Free Software Fundation, http://www.fsf.org/ 17, 379, 389

[14] GCC, the GNU Compiler Collection, http://gcc.gnu.org/ 17

[15] Microsoft Office, http://office.microsoft.com/ 90

[16] OPEN OFFICE, Apache OpenOffice, http://www.openoffice.org 91

[17] LibreOffice the Document Foundation, http://www.libreoffice.org 90

[18] CALLIGRA The Integrated Work Applications Suit, http://www.calligra.org/ 91

[19] LaTeX, A Document Preparation System, http://www.latex-project.org/ 4

[20] Mathtype, http://www.dessci.com/en/products/mathtype/ 90

[21] Scientific WorkPlace, http://www.mackichan.com/ 90

[22] Gummi LaTeX Editor, https://github.com/alexandervdm/gummi 90

[23] Kile LaTeX Editor, http://kile.sourceforge.net/ 90

[24] Led LaTex Editor, http://www.latexeditor.org/ 90

[25] Lyx LaTeX Editor, http://www.lyx.org/ 90

[26] Texmaker LaTeX Editor, http://www.xm1math.net/texmaker/ 90

[27] TeXnicCenter LaTeX Editor, http://www.texniccenter.org/ 90

[28] TextPad LaTeX Editor, http://www.textpad.com/ 90

[29] TeXstudio LaTeX Editor, http://texstudio.sourceforge.net/ 90

[30] WinEdt LaTeX Editor, http://www.winedt.com/ 90

[31] LaTeX Beamer Class, https://bitbucket.org/rivanvx/beamer/wiki/Home 102

[32] El economista, http://eleconomista.com.mx/tecnociencia/2013/01/22/clausuraran-negocios-mexico-uso-ilegal-Software 363

[33] PCworld, http://www.pcworld.com.mx/UNAM-y-BSA-promueven-el-uso-de-software-legal/

364

Declaro terminado este trabajo sufrido, ideado y llevado a cabo entre los años 2015 al 2025, aún y a pesar de impedimentos tales como: la mala suerte, la desventura, el infortunio, la incomprensión, la gripe, el Covid-19, las horas de frío y de calor, la tristeza, la desesperanza, el cansancio, el presente, el pasado y mi futuro, el que dirán, la vergüenza, mis propias incapacidades y limitaciones, mis aversiones, mis temores, mis dudas y en fin, todo aquello que pudiera ser tomado por mi, o por cualquiera, como obstáculo en este tiempo de mentiras, verdades, de incredulidad e ignorancia o negación de la existencia real y física de la mala fe.

Atentamente

Antonio Carrillo Ledesma