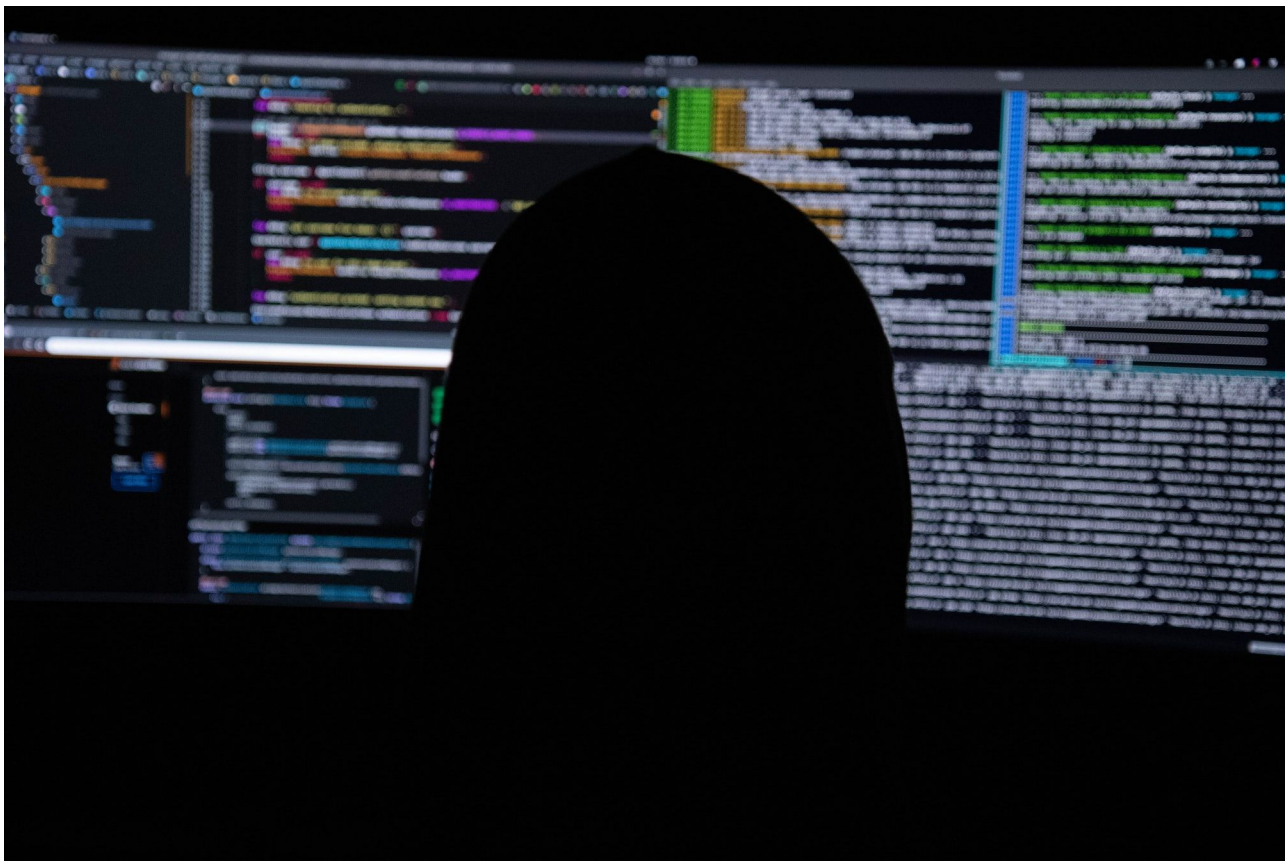
Lo Esencial de la Línea de Comandos de GNU/Linux



Antonio Carrillo Ledesma

Lo Esencial de la Línea de Comandos de GNU/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

Actualmente tenemos 3 grandes sistemas operativos en el mercado[1]:

- Windows
- Unix
- GNU[2]/Linux

De los cuales, sus dignos representantes son: Windows, macOS, iOS, Android, Chrome OS y GNU/Linux con todas sus diferentes distribuciones[3]. Y sin temor a equivocarnos aseguramos que Android es la distribución de GNU/Linux más popular e iOS es el más popular de los UNIX.

¿Qué Sistema Operativo Usar? ¿Apple o Microsoft? ¿Windows o Linux? ¿Android o iOS? Son preguntas frecuentes que todos nos hemos hecho alguna vez y es que elegir un sistema operativo, una computadora o un dispositivo móvil no es tan simple. Al menos no lo era años atrás. En la actualidad las diferencias entre sistemas operativos de escritorio son cada vez

[1] Cuotas de mercado de diferentes sistemas operativos:

https://gs.statcounter.com/os-market-share/desktop/worldwide
https://netmarketshare.com

[2] GNU -es un acrónimo recursivo de «GNU no es UNIX»- es un sistema operativo de Software libre, es decir, respeta la libertad de los usuarios. El sistema operativo GNU consiste en paquetes de GNU además de Software libre publicado por terceras partes con distintas licencias que conforman una distribución.

[3] Una distribución de Linux es un sistema operativo compuesto por el Kernel de Linux, herramientas GNU, Software adicional y un administrador de paquetes. También puede incluir un servidor de pantalla y un entorno de escritorio que se utilizarán como sistema operativo de escritorio normal. El término es distribución de Linux (o distribución en forma abreviada) porque una entidad como Debian o Ubuntu 'distribuye' el Kernel de Linux junto con todo el Software y las utilidades consideradas por cada entidad como necesarias (como administrador de red, administrador de paquetes, entornos de escritorio, etc.) para que pueda ser utilizado como sistema operativo. Sus distribuciones también asumen la responsabilidad de proporcionar actualizaciones para mantener el Kernel y otras utilidades.

Entonces, Linux es el Kernel, mientras que la distribución de Linux es el sistema operativo. Esta es la razón por la que también se les conoce como sistemas operativos basados en Linux (hay otros Kernels como son FreeBSD, NetBSD y Hurd).

menos, hasta el punto que prácticamente cualquier servicio Online es compatible con Windows, Mac, GNU/Linux y las principales firmas de Software crean aplicaciones para las tres plataformas principales, salvo excepciones.

Poco tendremos que decir del sistema operativo de Apple, Mac o iOS, ya que son los sistemas operativos más bonitos y que mejores resultados han dado a todos los usuarios que los han probado. Mac es un sistema pensado para los profesionales de los sectores que necesitan de un equipo de cómputo que sea capaz de proveer programas específicos para: desarrolladores, programadores, diseñadores, periodistas, fotógrafos, músicos, DJ's y muchos más empleos que se benefician de este sistema operativo.

Después tenemos a Windows, un sistema operativo versátil pensado principalmente para un uso doméstico, aunque eso no quita que muchas empresas utilicen Windows en sus equipos de cómputo ya que es un sistema operativo que puede dar muy buenos resultados en este aspecto.

Sin embargo, llegamos a Linux, el gran desconocido por muchos. Un sistema operativo mucho más versátil que Windows y que puede ser igual o más profesional que Mac. Sin embargo, la ventaja que tienen estos dos sistemas operativos, es que vienen ya preparados y configurados para el tipo de mercado al que van dirigidos, pero GNU/Linux no. Esto es una ventaja y una desventaja al mismo tiempo, ya que si tenemos práctica, podemos hacer que el sistema operativo se adapte a nuestras necesidades sin problemas, además es Software libre (véase apéndice 5.2).

1.1 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 programadores 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 programador 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 programador que manipula archivos de texto, esa mesa de trabajo es el intérprete de comandos. Desde el Prompt del indicador 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 programadores 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 or-

denador, 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[4]) está separado del sistema operativo[5], 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

[4]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).

[5]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.

línea de comando. Puede ejecutar comandos para cambiar permisos, 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[6] 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
```

[6] 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
```

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.

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.

1.2 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 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 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 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 su computadora, suele ser más caro que el propio equipo en el que se ejecutan.

1.2.1 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, en sector gubernamental, no se cuenta con los recursos necesarios para adquirir y mantener actualizado el Software requerido 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- con la capacidad de analizar, diseñar y programar las herramientas que requieran para sus procesos de investigación, enseñanza o desarrollo.

Existen hoy en día, una gran cantidad de paquetes y sistemas operativos comerciales de Software propietario 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.

1.2.2 Software Libre

El Software libre 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 y el programa ejecutable otorgando de esa forma la libertad 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, usuarios y empresas desarrolladoras de Software y Hardware que tratan de poner la mayor cantidad de programas a disposición de todos los interesados, de forma que permiten al usuario promedio sacar el mayor provecho del equipo de cómputo que use, sin importar el sistema operativo subyacente (pero es mejor que todo el Software de un equipo incluyendo el sistema operativo sea Software libre).

¿Qué es el Software Libre? La definición exacta y sus diversas variantes se verán en la siguiente sección, pero podemos entender su esencia a través de los documentos de la fundación para el Software libre. 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 [21], [20] y [18]). Algunos han conseguido un uso y alta calidad, por ejemplo el compilador GCC (véase [23]), el Kernel de Linux (véase [24]) 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[7]; y ¿qué es investigar y enseñar?, sino crear conocimiento y

[7] ¿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!

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 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 [18] y [19]) -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 [18] y [19]).

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.

1.3 GNU/Linux

Nos permite trabajar de manera más segura -y gratuita- que los demás sistemas operativos, este se ve y es muy parecido a cualquier otro sistema UNIX -como Mac OS X-, 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.

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 el 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 varios proyectos o creados, en colaboración con equipos como el proyecto GNU de la Free Software Foundation -Fundación por el Software libre es una organización creada en 1985 con el propósito de difundir este movimiento-.

Ante la duda de que distribución de GNU/Linux usar, lo mejor es analizar ciertos parámetros o características de las distribuciones. Los puntos más relevantes en los que te deberías fijar para elegir la mejor son:

- Robustez y estabilidad: si buscas un sistema operativo para usarlo en producción, seguro que no quieres perder tiempo con fallos o problemas. Por eso, es importante elegir las distribuciones más robustas y estables que funcionen como relojes suizos. Algunos buenos ejemplos son Debian, Ubuntu, Arch, openSUSE y Fedora.

- Seguridad: no podía faltar la seguridad, es un tema prioritario. Muchas distribuciones de Linux respetan tu privacidad mucho más que otros sistemas operativos, ya que no reportan datos del usuario, o al menos dan la elección de no hacerlo. Aunque GNU/Linux es un sistema seguro de base, no hay que confiarse, los ciberdelincuentes cada vez están más atentos a este sistema y cada vez hay más Malware que lo afecta. Por eso, si vas a elegir una distribución para una empresa o servidor, este debería ser un criterio prioritario. Algunas como Debian, SUSE, RHEL, CentOS, etc., podrían ser buenos casos para servidores. Y también tienes proyectos más específicos centrados en la seguridad como Whonix, QubeOS, TAILS, etc.

- Compatibilidad y soporte: el Kernel Linux tiene soporte para multitud de arquitecturas diferentes, como x86, ARM, RISC-V, etc. Sin embargo, no todas las distribuciones ofrecen ese soporte de forma oficial. Por eso, si vas a usar la distribución en una arquitectura diferente, es importante que te informes de si cuentan o no con dicho soporte. Por otro lado está el tema de los controladores y la compatibilidad de Software. En ese caso, Debian, Ubuntu y las distribuciones basadas en ésta son las «reinas», debido a que hay infinidad de paquetes y controladores para ella (es una de las más populares).

- Paquetería: a pesar de que los paquetes estándar deberían ser los RPM, como se especifica en LBS, lo cierto es que distribuciones tan populares como Debian, Ubuntu han conseguido que DEB sea el predominante. Con la llegada de los paquetes universales se han resuelto algunos problemas, pero si deseas contar con la mayor cantidad de Software, ya sean apps o videojuegos, la mejor opción es DEB con Debian y Ubuntu.

- Usabilidad: esto no depende de la distribución en sí, sino del entorno de escritorio, y de otras partes como el gestor de paquetes, si tiene o no utilidades que faciliten la administración como las incluidas en Debian, Linux Mint, o YaST 2 en openSUSE/SUSE, etc. Aunque, por lo general, las distribuciones actuales suelen ser bastante fáciles y amigables, con algunas excepciones...

- Ligero vs pesado: muchas distribuciones modernas suelen ser más pesadas, es decir, que demandan más recursos de Hardware o solo soportan ya 64-bit. En cambio, existen algunos entornos de escritorio ligeros como KDE Plasma (que ha «adelgazado» mucho últimamente y ha dejado de ser el escritorio pesado que era), LXDE, LXQt, Xfce, etc., así como distribuciones más livianas pensadas para ordenadores más antiguos o con pocos recursos. Todas estas opciones se encuentran en Debian.

- Otros aspectos: otro factor a tener en cuenta es tus preferencias o gustos por ciertos sistemas. Por ejemplo:

  ◦ SELinux (Fedora, CentOS, RHEL,...) vs AppArmor (Ubuntu, SUSE, openSUSE, Debian...)

  ◦ systemd (la mayoría) vs SysV init (Devuan, Void, Gentoo, Knoppix,...)

  ◦ FHS (la mayoría) vs otros como el de GoboLinux.

  ◦ etc.

1.4 ¿Qué tan Seguro es GNU/Linux?

No es ningún secreto que el sistema operativo que elijas es un determinante clave de su seguridad (no sólo en internet, también la privacidad de tus datos en el equipo que usas). Después de todo, el sistema operativo es el Software más crítico que se ejecuta en nuestra computadora o dispositivo inteligente: administra su memoria y procesos, así como todo su Software y Hardware. El consenso general entre los expertos es que Linux es un sistema operativo altamente seguro, posiblemente el sistema operativo más seguro por diseño. Examinaremos aquí algunos factores clave que contribuyen a la sólida seguridad de Linux y veremos el nivel de protección contra vulnerabilidades y ataques que Linux ofrece a los administradores y usuarios.

Seguro por Diseño cuando se trata de seguridad, los usuarios de Linux tienen una clara ventaja sobre sus contrapartes que usan Windows o MacOs. A diferencia de los sistemas operativos propietarios, Linux, en muchos sentidos, tiene seguridad integrada en su diseño central. El sistema operativo de código abierto cada vez más popular es de alta flexibilidad, configurable y diverso. También implementa un modelo estricto de privilegios de usuario y ofrece una selección de defensas de seguridad de Kernel integradas para protegerse contra vulnerabilidades y ataques. La transparencia del código fuente de Linux significa que las vulnerabilidades en él, que son inevitables hasta cierto punto en cualquier sistema operativo, casi siempre son de corta duración. Echemos un vistazo más de cerca a cada uno de estos factores y cómo contribuye a la seguridad anunciada de Linux.

La Ventaja de la Seguridad de Código Abierto el código fuente de Linux se somete a una revisión exhaustiva y constante por parte de los miembros de la vibrante comunidad global de código abierto y, como resultado de este escrutinio, las vulnerabilidades de seguridad de Linux generalmente se identifican y eliminan muy rápidamente. Por el contrario, los proveedores propietarios como Microsoft y Apple emplean un método conocido como "seguridad por oscuridad", donde el código fuente se oculta a los extraños en un intento de ocultar las vulnerabilidades de los actores de amenazas. Sin embargo, este enfoque generalmente es ineficaz para prevenir las vulnerabilidades modernas y, en realidad, socava la seguridad del código fuente "oculto" al evitar que personas ajenas identifiquen y reporten fallas antes de que sean descubiertas por actores malintencionados. Seamos realistas: cuando se trata de descubrir errores de seguridad, un pequeño equipo de desarrolladores propietarios no es rival para la comunidad mundial de usuarios-desarrolladores de Linux que están profundamente involucrados en su trabajo tanto para su propio beneficio como para el beneficio de la comunidad.

Un Modelo Superior de Privilegios de Usuario a diferencia de Windows, donde "todo el mundo es administrador", Linux restringe en gran medida el acceso a la raíz a través de un modelo estricto de privilegios de usuario. En Linux, el superusuario posee todos los privilegios, y a los usuarios comunes solo se les otorgan suficientes permisos para realizar tareas comunes. Debido a que los usuarios de Linux tienen pocos derechos de acceso automático y requieren permisos adicionales para abrir archivos adjuntos,

acceder a archivos o ajustar las opciones del Kernel, es más difícil propagar Malware y Rootkits en un sistema Linux. Por lo tanto, estas restricciones inherentes sirven como una defensa clave contra los ataques y el compromiso del sistema.

Defensas de Seguridad de Kernel Incorporadas el Kernel de Linux cuenta con una variedad de defensas de seguridad integradas que incluyen Firewalls que utilizan filtros de paquetes en el Kernel, el mecanismo de verificación de Firmware UEFI Secure Boot, la opción de configuración Linux Kernel Lockdown y los sistemas de mejora de seguridad SELinux o AppArmor Mandatory Access Control (MAC). . Al habilitar estas funciones y configurarlas para brindar el más alto nivel de seguridad en una práctica conocida como autoprotección del Kernel de Linux, los administradores pueden agregar una capa adicional de seguridad a sus sistemas.

Seguridad a Través de la Diversidad existe un alto nivel de diversidad posible dentro de los entornos de Linux como resultado de las muchas distribuciones de Linux disponibles y las diferentes arquitecturas de sistema y componentes que presentan. Esta diversidad no solo ayuda a satisfacer los requisitos individuales de los usuarios, sino que también ayuda a protegerse contra los ataques al dificultar que los actores maliciosos elaboren de manera eficiente Exploits que puedan usarse contra una amplia gama de sistemas Linux/Linux. Por el contrario, la "monocultura" homogénea de Windows convierte a Windows en un objetivo de ataque relativamente fácil y eficiente (algo parecido también les pasa a las Mac).

Además de la diversidad de diseño que se ve en Linux, ciertas distribuciones seguras de Linux se diferencian en formas que abordan específicamente las preocupaciones de seguridad y privacidad avanzadas compartidas entre los Pentesters, los ingenieros inversos y los investigadores de seguridad.

Altamente Flexible y Configurable hay muchas más opciones de configuración y control disponibles para los administradores de Linux que para los usuarios de Windows y MacOs, muchas de las cuales se pueden usar para mejorar la seguridad. Por ejemplo, los administradores de sistemas de Linux tienen la capacidad de usar SELinux o AppArmor para bloquear su sistema con políticas de seguridad que ofrecen controles de acceso granulares, proporcionando una capa adicional crítica de seguridad en todo el sistema.

Los administradores también pueden usar la opción de configuración Linux Kernel Lockdown para fortalecer la división entre los procesos de la zona de usuario y el código del Kernel, y pueden fortalecer el archivo sysctl.conf, el principal punto de configuración de parámetros del Kernel para un sistema Linux, para darle a su sistema una base más segura.

Linux: Un Objetivo Cada Vez Más Popular Entre los Ciberdelincuentes Linux alimenta la mayoría de los dispositivos y supercomputadoras de alto valor del mundo y la base de usuarios del sistema operativo está creciendo constantemente, y los ciberdelincuentes han tomado nota de estas tendencias. Los autores y operadores de Malware apuntan cada vez más a los sistemas Linux en sus campañas maliciosas. Por ejemplo, en los últimos años han estado plagados de cepas emergentes de Malware para Linux: Cloud Snooper, EvilGnome, HiddenWasp, QNAPCrypt, GonnaCry, FBOT y Tycoon se encuentran entre las más notorias. Dicho esto, Linux sigue siendo un objetivo relativamente pequeño, con el 96 % del nuevo Malware dirigido a Windows en 2022. Además, el reciente aumento de los ataques de Malware de Linux no es un reflejo de la seguridad de Linux. La mayoría de los ataques a los sistemas Linux se pueden atribuir a configuraciones incorrectas y una administración deficiente, lo que destaca una falla generalizada entre los administradores de sistemas Linux para priorizar la seguridad.

Afortunadamente, a medida que el Malware de Linux continúa siendo cada vez más frecuente y problemático, Linux cuenta con protección integrada contra ataques de Malware a través de su estricto modelo de privilegios de usuario y diversidad de diseño, y hay una selección de excelentes herramientas, Kits de herramientas y utilidades de análisis de Malware e ingeniería inversa que incluyen REMnux, Chkrootkit, Rkhunter, Lynis y Linux Malware Detect (LMD) disponibles para ayudar a los administradores a detectar y analizar Malware en sus sistemas.

Para Tomar en Cuenta la seguridad del sistema operativo que implementa es un determinante clave de su seguridad en internet, pero de ninguna manera es una protección segura contra Malware, Rootkits y otros ataques. La seguridad efectiva depende de la defensa en profundidad, y otros factores, incluida la implementación de las mejores prácticas de seguridad y el comportamiento inteligente internet, juegan un papel central en su postura de seguridad digital. Dicho esto, elegir un sistema operativo seguro es de suma

importancia, ya que el sistema operativo es la pieza de Software más crítica que se ejecuta en nuestros dispositivos computacionales, y Linux es una excelente opción ya que tiene el potencial de ser altamente seguro, posiblemente más que su contraparte propietaria, debido a su código de fuente abierta, modelo estricto de privilegios de usuario, diversidad y base de usuarios relativamente pequeña.

Sin embargo, Linux no es una "bala de plata" cuando se trata de seguridad digital: el sistema operativo debe configurarse de manera adecuada y segura, y los administradores de sistemas deben practicar una administración responsable y segura para evitar ataques. Además, es fundamental tener en cuenta que la seguridad tiene que ver con las compensaciones, tanto entre seguridad y facilidad de uso como entre seguridad y facilidad de uso. Los administradores deben configurar sus sistemas para que sean tan seguros como sea práctico dentro de su entorno. En lo que respecta a la conveniencia, Linux tiene una pequeña curva de aprendizaje, pero ofrece importantes ventajas de seguridad sobre Windows o MacOs.

1.5 Sobre los Ejemplos de este Trabajo

La documentación y los diferentes ejemplos que se presentan en este trabajo, se encuentran disponibles en la siguiente liga:

Herramientas
http://132.248.181.216/Herramientas/

para que puedan ser copiados desde el navegador y ser usados en la terminal de línea de comandos. 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.

1.6 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/

2 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.

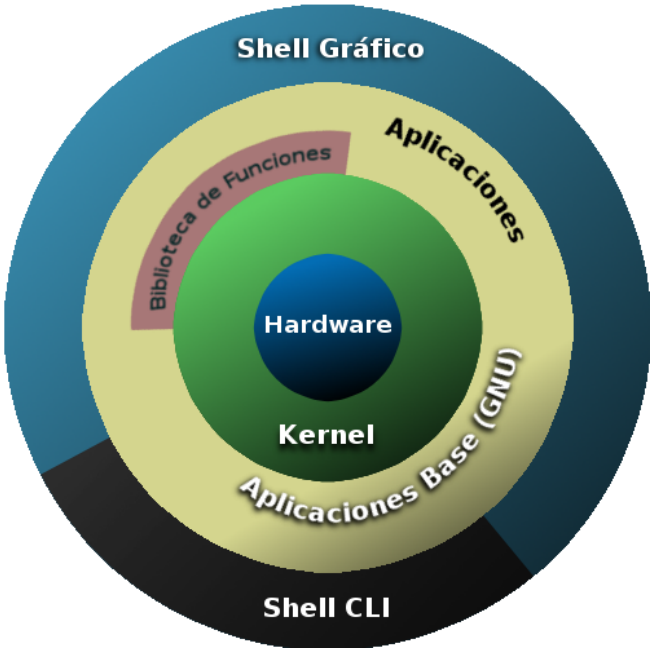


Figura 1: GNU/Linux

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, sería 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 más 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[8].

[8]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 Buffer 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 como: 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.

2.1 Interfaz de Usuario

Tanto el programador como el usuario de Linux manejan principalmente el conjunto de programas de sistemas que se han 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[9] (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

proyectos como Sway que se favorecen de las bondades de Wayland.

[9] 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.

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.

2.1.1 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 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[10], 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 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

[10] 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.

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.

2.1.2 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".

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.

Cuando uno se inicia en el mundo de GNU/Linux no nota la diferencia

entre:

- Terminal
- Shell
- Prompt
- Línea de comandos

Esto es lo que debes saber al respecto:

Terminal es una aplicación gráfica que ejecuta un Shell por defecto. Puede haber muchas aplicaciones de emulador de terminal como Terminator, Konsole, etc.

Shell es difícil visualizarlo por separado de la terminal. La terminal ejecuta un Shell, generalmente Bash. Al igual que las terminales, también hay varios shells: zsh, bash, fish, etc.

Prompt esto es lo que ve antes del espacio donde escribe los comandos. No existe un estándar establecido para el aviso. En algunas terminales antiguas, simplemente tendría un cursor parpadeante en el lugar donde puede escribir los comandos. En las terminales modernas, el mensaje le brinda información como nombre de usuario, nombre de Host y directorio de trabajo actual.

Línea de comandos no es algo específico de Linux. Cada sistema operativo tiene una interfaz de línea de comandos. Muchos lenguajes de programación tienen interfaces de línea de comandos. Es un término para la interfaz donde puedes ejecutar y ejecutar comandos.

Por lo general, escribimos los comandos en una terminal después del Prompt y el Shell los interpreta.

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.

Podemos instalar algunos emuladores de terminal usando:

```
# apt install sakura miterm xterm rxvt kitty lilyterm \
guake qterminal eterm roxterm terminator tilix tilda \
terminology yakuake konsole terminix deepin-terminal \
kitty sakura gnome-terminal stterm mate-terminal \
xfce4-terminal lxterminal cool-retro-term
```

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[11]. Es una Shell de Unix compatible con POSIX[12] y el intérprete de comandos por defecto en la mayoría de las dis-

[11] 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.

[12] 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 como -c y -h, aparecen en la versión de du provista por el proyecto GNU. Esto significa

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)[13].

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*

Podemos instalar algunos intérpretes de órdenes usando:

```
# apt install csh tcsh ksh fish zsh tsch dash dsh busybox
```

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

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.

[13] Para conocer cuál Shell estamos usando, abrimos una terminal y tecleamos:

```
$ echo $0
```

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 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*

2.1.3 SubShells

los subShells son un concepto fundamental en las secuencias de comandos Bash que pueden resultar potentes y confusos, especialmente para los principiantes. Le permiten crear entornos de ejecución aislados dentro de sus Scripts, proporcionando una forma de gestionar procesos y ejecutar comandos de forma eficaz sin afectar el entorno del Shell principal.

¿Qué es Exactamente un subShell? un subShell, también conocido como Shell secundario, es una instancia separada del Shell que se genera a partir del proceso de Shell actual. Hereda el entorno y las variables de su Shell principal, pero funciona de forma independiente, lo que permite la ejecución aislada de comandos y Scripts. Cuando se crea un subShell, se ejecuta en un proceso separado, distinto del Shell principal. Esto significa que cualquier cambio realizado en el entorno dentro del subShell, como modificar variables o definir funciones, está aislado y no persiste en el Shell principal después de que termina el subShell.

Creando un SubShell hay varias formas de crear un subShell en Bash, cada una con sus propios matices y casos de uso:

Paréntesis o Llaves los comandos entre paréntesis se ejecutan en un subShell. Esta es una de las formas más comunes y sencillas de crear un subShell en Bash.

```
# Crear un subshell
$ (pwd; ls; whoami)
```

El uso de llaves {...} alrededor de un conjunto de comandos también puede crear una subShell:

```
$ { sleep 3; echo "Hola desde el subshell"; }
# Subshell creado
$ echo "Regresando al shell padre"
```

Sustitución de Comandos

```
# Asigna la salida de un subshell a una variable
$ output=$(pwd; ls; whoami)
```

La sustitución de comandos crea un subShell y captura su salida, que puede asignarse a una variable o usarse en otro comando.

Invocación Explícita del subShell el comando integrado Bash se puede utilizar para iniciar un subShell y ejecutar comandos dentro de él explícitamente. La opción *-c* le permite especificar los comandos que se ejecutarán.

```
# Ejecutar un subshell
$ bash -c "ls; whoami"
```

Relación entre el Shell principal y el subShell el Shell principal y sus subShells tienen una relación jerárquica. Como mencioné, el subShell hereda las variables de entorno, funciones y otras configuraciones del Shell principal, pero cualquier modificación realizada en el entorno dentro del subShell está aislada y no afecta al Shell principal.

Para demostrar este aislamiento, considere el siguiente ejemplo:

```
# shell padre
$ echo "Shell padre: valor de psell $pshell"
# Crear un subshell y modificar pshell
$ (pshell=10; echo "Subshell: Valor de pshell $pshell")
# Checar el valor de pshell en el shell padre
$ echo "Shell padre: valor de pshell $pshell"
```

En este ejemplo, la variable pshell primero se desactiva en el Shell principal, lo que significa que no tiene valor. Dentro del subShell creado por paréntesis(), asignamos el valor 10 a la variable pshell e imprimimos su valor. Sin embargo, después de que finaliza el subShell, el valor de pshell del Shell principal permanece sin establecer, lo que demuestra el aislamiento del entorno del subShell.

Es importante tener en cuenta que los subniveles aíslan el entorno y afectan el alcance de las variables y funciones definidas dentro de ellos. No se puede acceder a las variables y funciones definidas en un subShell fuera de ese subShell, incluso si el subShell es parte de un Script o función más grande.

Cómo Comprobar si se ha Generado un subShell se puede comprobar si el Shell actual es un subShell inspeccionando el valor de la variable de entorno $BASH_SUBSHELL. Esta variable se establece en un valor distinto de cero en los subniveles, lo que indica el nivel de anidamiento.

```
$ echo $BASH_SUBSHELL
# Imprime 0 en el shell padre, no zero en los subshells
```

Otra excelente manera de verificar si ha generado una subShell usando el comando ps –forest. Consideremos ahora el siguiente ejemplo:

```
$ bash
$ bash
$ ps –forest
```

La ejecución del comando ps –forest muestra el anidamiento del subShell. Hablaremos de subShell anidadas en la siguiente sección.

SubShells Anidados los subniveles se pueden anidar, lo que significa que un subnivel puede generar otro subnivel dentro de sí mismo. Cada subShell anidado hereda el entorno de su subShell principal y la variable $BASH_SUBSHELL se incrementa en consecuencia.

```
$ echo "Parent shell: $BASH_SUBSHELL" (
echo "Subshell 1: $BASH_SUBSHELL" (
echo "Subshell 2: $BASH_SUBSHELL" ))
```

Aquí hay otro ejemplo:

```
$ bash
$ bash
$ bash
$ ps –forest
```

Aquí ingresamos el comando bash cuatro veces para crear cuatro subShell. Puede salir de cada subShell usando el comando bash *exit*.

¿Los Scripts de Shell se Ejecutan en subShell? ¡La respuesta es sí! De forma predeterminada, los Scripts de Shell se ejecutan en subShell. Esto significa que cualquier cambio realizado en las variables de entorno u otras configuraciones del Shell dentro del Script no se propaga al Shell principal. Sin embargo, este comportamiento se puede modificar utilizando el comando fuente o el punto (.), que ejecuta el Script en el contexto actual del Shell, permitiendo que cualquier modificación realizada en el entorno se refleje en el Shell principal:

```
# Ejecutar el script en contexto actual del shell
$ source .bashrc
```

Alternativamente, puede usar el comando exec dentro del Script para reemplazar el proceso de Shell actual con el proceso del Script, evitando que el Script se ejecute en un subShell.

```
$ exec bash
$ exec bash
$ exec bash
$ ps –forest
```

Hacer uso de SubShells los subShells se pueden utilizar de varias formas creativas para lograr los comportamientos deseados en los Scripts y comandos de Bash. Veamos algunas formas en que podemos hacer uso de subShell.

Poner las Listas de Procesos en Segundo Plano al incluir una lista de comandos entre paréntesis, puede crear un subShell que ejecute los comandos en segundo plano, lo que le permitirá continuar trabajando en el Shell principal mientras los procesos del subShell se ejecutan simultáneamente.

```
# Subshell corriendo en segundo plano
$ (sleep 10; echo "Esto corre en segundo plano") &
```

Coprocesamiento el coprocesamiento es similar a ejecutar un comando en segundo plano, la única diferencia es que también crean un subShell. Los coprocesos son útiles para tareas que requieren ejecución paralela o comunicación entre procesos.

```
# Creación de un coproceso que dormira 60 segundos
$ coproc sleep 60
```

Procesamiento Paralelo con SubShell los subShells también se pueden utilizar para el procesamiento paralelo, lo que permite ejecutar múltiples tareas simultáneamente. Al encerrar cada tarea entre paréntesis y separarlas con el operador &, puede crear subShell que se ejecuten en paralelo.

```
# Ejemplo de procesamiento en paralelo
$ (task1.sh) & (task2.sh) & (task3.sh) &
# Esperar a que todas las tareas se completen
$ wait
```

En el ejemplo anterior, task1.sh, task2.sh y task3.sh se ejecutan en subShells separadas, ejecutándose en paralelo. El comando de espera garantiza que el Shell principal espere a que se completen todos los procesos del subShell antes de continuar.

Esta técnica puede ser especialmente útil para tareas computacionalmente intensivas o que requieren mucho tiempo, donde la paralelización puede mejorar significativamente el tiempo de ejecución general.

Por qué a Veces Deberías Evitar el uso de subShell en Bash si bien los subShells ofrecen características y funcionalidades valiosas, hay situaciones en las que es posible que desee evitar la creación de subShell innecesarias. La creación de una subShell implica generar un nuevo proceso, lo que puede generar cierta sobrecarga, especialmente cuando se realiza con frecuencia. Además, los cambios realizados en las variables de entorno u otras configuraciones del Shell dentro de un subShell no se propagan al Shell principal, lo que a veces puede provocar comportamientos inesperados o inconsistencias. El uso excesivo de subShells también puede hacer que los Scripts sean más complejos y difíciles de leer, especialmente para aquellos que no están familiarizados con el concepto. Además, cada subShell consume recursos adicionales del sistema, como memoria y descriptores de archivos, lo que puede resultar problemático en entornos con recursos limitados o cuando se trata de una gran cantidad de subShells.

2.2 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[14]:

[14]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

- si es Hardware muy antiguo, muy moderno o muy raro.
- 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

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.

- 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 módulos 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 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[15] 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[16]. 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.

[15] 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.

[16] 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.

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 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.

2.3 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 un tipo particular de archivo. 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[17].
- Ext4: Es más rápido y permite archivos mucho más grandes con una velocidad significativa[18].
- 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.

[17] 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.

[18] 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.

- 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:

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

Qué es una partición una partición es una subdivisión realizada por Software del dispositivo de almacenamiento. El sistema operativo considerará en la práctica a cada partición como si fuera un medio de almacenamiento independiente. Esto resulta útil por ejemplo si tenemos datos a los que queremos acceder desde diferentes sistemas operativos, si hay información que queremos ocultar o datos que no queremos que se pierdan al instalar desde cero una nueva versión de Linux.

Breve introducción a las particiones en Linux antes de poder instalar en el disco duro de nuestro ordenador un sistema operativo y, para poder gestionar la información tanto en este como en dispositivos de almacenamiento externos como pendrives o tarjetas de memoria, es necesario realizar una serie de pasos previos.

Aunque muchas veces los discos rígidos vienen listos para usar e incluso con Software utilitario, lo más seguro es que solo estén preparados para Windows. Es entonces cuando tendremos que borrar su contenido (siempre se recomienda hacer una copia de lo que contenga antes de hacerlo).

Una restricción que debemos tener en cuenta es que por razones obvias no podemos modificar la partición del sistema operativo que estamos utilizando. Es el motivo de por qué para realizar cualquier cambio necesitaremos otro

sistema operativo. Es posible utilizar uno instalado en nuestro ordenador o dispositivo extraíble, una distribución Linux en modo Live -en este modo el sistema operativo se ejecuta en la memoria de la computadora por lo que no requiere instalación- diseñada para tareas de mantenimiento o reparación o el medio de instalación de cualquier distribución Linux.

Cualquier medio de almacenamiento con permiso de escritura necesita al menos de una partición de un tamaño igual o menor al de su capacidad. Si queremos que esa partición almacene datos, debe ser formateada con un sistema de archivos.

Una ventaja de las distribuciones Linux sobre Windows es que pueden leer los datos de las particiones que usa este último sin necesidad de Software adicional. Con respecto a la cantidad de particiones que se pueden crear, en la actualidad no existen restricciones, salvo las dadas por la lógica. No tiene sentido crear múltiples particiones que no son lo suficientemente grandes para resultar útiles.

Dijimos que cada partición necesita de un sistema de archivos. Un sistema de archivos es una forma de organizar los datos almacenados. Aunque en un disco pueden convivir múltiples sistemas de archivos, cada partición podrá tener solo uno.

Además de los datos almacenados, los sistemas de archivos incluyen la información necesaria para acceder a estos como su nombre, su tamaño y los permisos de acceso. Se incluye un índice que además de los contenidos señala su localización para permitir acceder a ellos más rápidamente.

Siempre es conveniente antes de cambiar el sistema de archivos de una partición hacer copias de los datos importantes, formatearla y volver a copiar la información. Hay quién dice que se puede cambiar el sistema de archivos sin formatear, pero es un riesgo que no se justifica.

Un sistema de archivos establece la forma de guardar los datos, recopilar información de los mismos y posibilitar el acceso a ellos. Las tablas de particiones indican el tipo y tamaño de particiones presentes en un dispositivo de almacenamiento y su localización. Además, señalan la presencia de sistemas operativos con gestores de arranque.

Tipos de particiones habíamos dicho que las particiones son divisiones creadas mediante Software. Su uso tiene las siguientes ventajas:

- Permite dedicar un dispositivo a diferentes usos.

- Facilita la organización asignando diferentes particiones a distintos tipos de datos
- Permite adecuar el uso del dispositivo de acuerdo a las necesidades del momento.
- Es posible asignar distintos permisos de acceso o cifrar cada una de ellas.

Si tratas de instalar Linux en un ordenador de varios años atrás te encontrarás que no puedes crear más de 4 particiones. Esa restricción no existe en los equipos más modernos ya que utiliza un sistema de tablas de particiones diferente -de todas formas, si recién te estás iniciando, lo mejor es dejar que el instalador se ocupe de todo-.

Tenemos dos tipos de tablas de particiones disponibles. MBR (Master Boot Record) para los equipos más antiguos y GPT (GUID Partition Table) para los más modernos. GPT es superior a MBR no solo porque trabaja con discos de mayor tamaño sino que permite la recuperación en caso de daño físico en el dispositivo.

Para superar sus limitaciones, MBR dispone de dos clases de particiones:

- Partición Primaria: Solo puede haber 4 estando una sola activa. Son útiles para almacenar el sistema operativo con su correspondiente gestor de arranque, la computadora accederá a la partición indicada como activa para completar el procedimiento de arranque.
- Partición Extendida: Para superar el límite de 4 particiones primarias existe la posibilidad de crear una partición extendida que actúa como contenedor para nuestro tercer tipo de partición, la partición lógica.
- Partición Lógica: Las particiones lógicas tienen parcialmente las funciones de las particiones primarias. La principal restricción es que no pueden contener el gestor de arranque. Es por eso que es necesario hacerlo en una partición lógica.

Tanto Windows como los instaladores de las diferentes distribuciones Linux incluyen su propia herramienta para el trabajo con particiones. Los escritorios GNOME y KDE disponen además de sus propias herramientas que pueden venir preinstaladas o estar en los repositorios. En el caso de GNOME se llama Gparted (Que también puede descargarse como una distribución Linux independiente) y en el de KDE Editor de particiones.

Sistema de Archivos 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[19] que a continuación describiremos brevemente:

/ 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.

[19] 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.

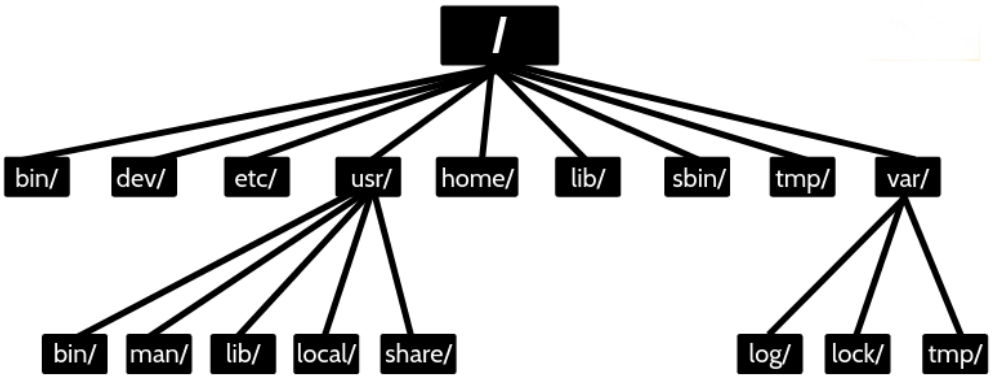


Figura 2: Jerarquía del sistema de archivos de Linux.

/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[20] 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.

/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.

[20] 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.

/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[21]. 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/
```

estas rutas suelen ser bastante largas, pero tienen como ventaja que funcionan siempre, independientemente del lugar desde el que se ejecute la orden[22].

[21] 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.

[22] 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.

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.
- 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:

Veamos por partes: El primer carácter al extremo izquierdo, representa el tipo de archivo, los posibles valores para esta posición son los siguientes:

- \- Archivo

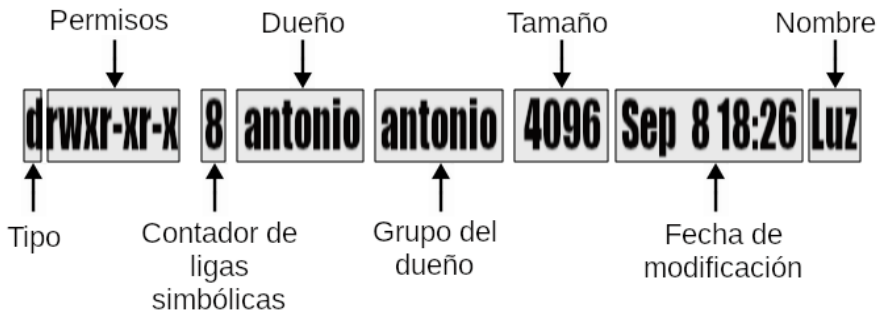


Figura 3: Estructura de permisos en la salida de: ls -l

- 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[23] este Bit a un fichero sería:

```
# chmod u+s /bin/su
```

Y para quitarlo:

```
# chmod u-s /bin/su
```

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
```

[23] Debemos utilizar este Bit con extremo cuidado ya que puede provocar una escalada de privilegios en nuestro sistema.

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.

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 [^][24].

- ***** 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:

[24] Véase también el uso de las secuencias (véase 3.7).

```
$ 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[25]:

```
$ sudo apt update
```

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:

[25] *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.

```
$ 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
```

Argumentos Extendidos xargs (argumentos extendidos) lee datos de la entrada estándar (esto es, Standard Input) y los convierte en comandos. El comando o comandos en cuestión se pasan al segundo comando como parámetro o argumento y a continuación se ejecuta/n una o varias veces. Si se prescinde de un comando especial como parámetro, xargs utiliza automáticamente el comando echo.

¿Cuál es la sintaxis de xargs en Linux? si quieres usar el comando xargs en la terminal, utiliza la sintaxis:

```
$ Primer_Comando | xargs [Opciones] [Segundo_Comando]
```

de esta forma el segundo comando se ejecuta con los argumentos del primero.

¿Cuáles son las opciones del comando xargs? hay numerosas opciones para xargs. A continuación, las más importantes:

-0 o -null: con esta opción, cada carácter se toma literalmente y el carácter NULL separa los argumentos.

-a o -arg-file: con esta opción, los argumentos se leen de un archivo en vez de la entrada estándar.

-d o -delimiter: con esta opción, también cada carácter se toma literalmente. La separación la marca el carácter delimitador y no los espacios en blanco.

-p o -interactive: con esta opción, antes de cada ejecutar xargs, se pregunta si se debe proceder.

Ejemplo de xargs la mejor forma de mostrar la funcionalidad del comando xargs es con algunos ejemplos.

```
$ find . -name "*.txt" | xargs rm
```

en este ejemplo, xargs se utliza con el comando de find y el comando de rm. Como resultado, todos los archivos con la extensión .txt se eliminan.

```
$ find . -name "*.txt" | xargs grep "Factura"
```

en este ejemplo, xargs ayuda a encontrar todos los archivos que contienen la palabra "Factura". Para ello, además de find se usa también el comando grep.

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 a 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[26]. 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[27]. 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 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:

[26] 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*.

[27] 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.

```
$ 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[28], 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:

[28] 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
```

2.4 Trabajando en Línea de Comandos

GNU/Linux es un potente sistema operativo visual y de línea de comandos[29]. En esta última se tiene una potente herramienta, en ella se encuentran desde los comandos básicos hasta los más avanzados[30]. Los comandos que se pueden usar son de dos tipos: los pertenecientes al GNU Core Utilities y los externos (que vienen instalados en la distribución de Linux o debe instalar el usuario), algunos de ellos son:

Para manipulación de archivos y directorios[31]

> *ls, pwd, cd, mkdir, rmdir, cp, mv, rename, rm, ln, unlink, cat, touch, cmp, diff, wc, tail, head, more, less, paste, cut, tr, fold, nano*

Comandos generales

> *man, help, info, whatis, which, whereis, clear, w, time, whoami, date, cal, uptime, uname, df, du, free, bc, history, echo*

Administración y Permisos

> *chmod, chown, chgrp, su, useradd, usermod, deluser, passwd, lsattr, chattr, id*

[29] Android tiene la base de Linux, por ello en cualquier dispositivo que soporte dicho sistema operativo es posible instalar una aplicación para acceder a la terminal de línea de comandos —por ejemplo ConnectBot—, y en ella podemos correr los comandos que mostramos en esta sección.

[30] En la Web se puede obtener acceso a diversos proyectos que ponen a disposición del usuario la documentación de una gran variedad de comandos de Linux, algunos de estos proyectos son:
http://man7.org/linux/man-pages/
https://linux.die.net/man/
https://www.kernel.org/doc/man-pages/

[31] Existe la opción de gestionar ficheros comprimidos, gracias a los comandos *zgrep, zegrep, zless, zmore, zdiff, zcmp, zcat*. 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*. 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*.

Búsqueda

find, grep, locate

Respaldo

tar, gzip, bzip2, zip, unp

Varios

file, stat, type, ps, kill, killall, pgrep, pwdx, awk, sort, sed, md5sum, sleep, watch

Para monitorear el desempeño

lscpu, free, top, whowatch, dstat, vmstat, iotop, iostst, lsof, lsusb, tcpdump, nmcli, ip

A continuación detallamos el uso de varios de estos comandos que se ejecutan en la línea de comandos de GNU/Linux o Terminal[32]. Hay que recalcar que cada comando tiene una gran variedad de opciones, pero la descripción completa de cada comando y opciones de este, se escapa de nuestros fines, por ello si se necesita conocer la referencia completa de dichos comandos hay varias maneras de obtenerla, entre otras haciendo uso de *man, help, info o whatis* aplicado al comando de nuestro interés, por ejemplo:

```
$ man ls
```

[32]Existen varios atajos de teclado que facilitan el navegar en la terminal de comandos, entre los que destacan:
CTRL L Limpia el contenido de la terminal
CTRL C Concluye el programa que está en ejecución
CTRL D Concluye la sesión en la terminal cerrando esta
SHIFT Page Up/Down Navega en la terminal una página arriba o abajo
CTRL A Posiciona el cursor al inicio de la línea
CTRL E Posiciona el cursor al final de la línea
CTRL U Borra lo que está a la izquierda del cursor
CTRL K Borra lo que está a la derecha del cursor
CTRL W Borra la palabra a la derecha del cursor
CTRL Y Pega lo que se quitó con CTRL U, K, W
TAB Autocompleta el nombre de archivo o comando
CTRL R Permite buscar dentro del historial de comandos
!! Permite repetir el último comando
CTRL Z Detiene la ejecución del comando actual (permite continuar la ejecución con *fg* en primer plano o *bg* en segundo plano)

Manipulación de Archivos y Directorios

ls (de listar), permite listar el contenido de un directorio o fichero. La sintaxis es:

```
$ ls /home/directorio
```

el comando ls tiene varias opciones que permiten organizar la salida, lo que resulta particularmente útil cuando es muy grande. Por ejemplo, puedes usar *-a* para mostrar los archivos ocultos, *-i* para mostrar el inodo[33], *-l* para mostrar los usuarios, permisos, tamaño en Bytes y la fecha de los archivos; *-S* ordena por tamaño, *-R* recursivo, *-r* en orden inverso, *-t* ordenados por fecha de modificación, *-h* muestra el tamaño en unidades fáciles de leer -como KB, MB o GB-, *-F* les adiciona una / al final del nombre de los directorios, *-d* sólo muestra directorios, -X ordenar por extensión, *-n* muestra *UID* y *UIG* de los archivos y directorios. Así como para todos los comandos Linux, estas opciones pueden combinarse, terminando en algo como:

```
$ ls -lha /home/directorio
```

por ejemplo para mostrar primero los archivos recientemente modificados, usamos:

```
$ ls -lt
```

o los más recientes al final, usamos:

```
$ ls -ltr
```

podemos pedir que sólo liste los directorios en la trayectoria actual:

```
$ ls -d */
```

que nos muestre aquellos archivos que coincidan con un patrón:

```
$ ls archivo*
$ ls *.txt
$ ls archivo?txt
$ ls archivo-{01,03,05}.txt
$ ls archivo-0[135].txt
```

[33] El inodo es un registro en el disco que contiene la información del archivo como su propietario, tamaño, fecha de creación, ubicación, entre otros.

pwd (de print working directory o imprimir directorio de trabajo), es un comando que imprime nuestra ruta o ubicación al momento de ejecutarlo, así evitamos perdernos si estamos trabajando con múltiples directorios y carpetas. Su sintaxis sería:

```
$ pwd
```

cd (de change directory o cambiar directorio), es como su nombre lo indica el comando que necesitarás para acceder a una ruta distinta de la que te encuentras. Por ejemplo, si estas en el directorio /home y deseas acceder a /home/ejercicios, escribimos:

```
$ cd /home/ejercicios
```

teclear el comando *cd* solo regresa al directorio home del usuario (lo mismo hace al teclear cd ~), teclear el comando *cd* - retorna al último directorio antes de hacer cambio de directorio, si estas en /home/ejercicios y deseas subir un nivel (es decir ir al directorio /home), ejecutas:

```
$ cd ..
```

mkdir (de make directory o crear directorio), crea un directorio nuevo tomando en cuenta la ubicación actual. Por ejemplo, si estas en /home y deseas crear el directorio ejercicios, sería:

```
$ mkdir /home/ejercicios
```

mkdir tiene una opción bastante útil que permite crear un árbol de directorios completo que no existe. Para eso usamos la opción *-p*:

```
$ mkdir -p /home/ejercicios/prueba/uno/dos/tres
```

o podemos pedir que cree múltiples directorios simultáneamente:

```
$ mkdir {uno, dos, tres}
```

rmdir (de remove directory o borrar directorio), borra un directorio vacío:

```
$ rmdir /home/ejercicios
```

o podemos pedir que borre múltiples directorios vacíos simultáneamente:

```
$ rmdir {uno, dos, tres}
```

cp (de copy o copiar), copia un archivo o directorio origen a un archivo o directorio destino. Por ejemplo, para copiar el archivo prueba.txt ubicado en /home a un directorio de respaldo, podemos usar:

```
$ cp /home/prueba.txt /home/respaldo/prueba.txt
```

en la sintaxis siempre se especifica primero el origen y luego el destino. Si indicamos un nombre de destino diferente, *cp* copiará el archivo o directorio con el nuevo nombre.

```
$ cp /home/prueba.txt /home/respaldo/prueba01.txt
```

El comando también cuenta con la opción *-r* que copia no sólo el directorio especificado sino todos sus directorios internos de forma recursiva. Suponiendo que deseamos hacer una copia del directorio */home/ejercicios* que a su vez tiene las carpetas *ejercicio1* y *ejercicio2* en su interior, en lugar de ejecutar un comando para cada carpeta, ejecutamos:

```
$ cp -r /home/ejercicios /home/respaldos/
```

también podemos usar *-u* para copiar aquellos archivos que no existen o son nuevos en el directorio destino:

```
$ cp -u /home/ejercicios/*.dat /home/respaldos/
```

Algunas otras opciones son: *-a* copia el archivo con la misma configuración de permisos y metadatos que el original, *-b* Crea una copia en la memoria intermedia (también llamada buffer) si el archivo original y el destino tienen el mismo nombre, pero diferente contenido, *-d* copia los enlaces simbólicos, *-f* obliga a sobrescribir al copiar, -i pide permiso antes de sobrescribir archivos con el mismo nombre, *-l* crea un enlace duro en lugar de

una copia, *-n* los archivos existentes nunca se sobrescribirán, *-p* los atributos del archivo original se heredan al copiar, *-P* los enlaces simbólicos se guardan como tales al copiar, *-R* los directorios, incluidos los subdirectorios, se copian de forma recursiva, *-s* crea un enlace simbólico para el archivo original, *-S* sobrescribe un sufijo de backup al copiar con –backup, *-u* copia el archivo solamente si el archivo de destino es más antiguo que el original.

mv (de move o mover), mueve un archivo a una ruta específica, y a diferencia de *cp*, lo elimina del origen finalizada la operación. Por ejemplo:

```
$ mv /home/prueba.txt /home/respaldos/prueba2.txt
```

al igual que *cp*, en la sintaxis se especifica primero el origen y luego el destino. Si indicamos un nombre de destino diferente, *mv* moverá el archivo o directorio con el nuevo nombre.

Si queremos solo cambiar la extensión de un archivo podemos usar:

```
$ mv archivo.{old,new}
```

rename este comando[34] permite renombrar un grupo de archivos[35], por ejemplo deseamos renombrar todos los archivos con extensión .htm y reemplazarlos con la extensión .html, entonces usamos:

```
$ rename .htm .html *.htm
```

si tenemos los directorios dir1, dir2, dir3 y los queremos renombrar como dir001, etc., usamos:

```
$ rename -v dir dir00 dir?
```

[34] En caso de no estar instalado, lo instalamos usando:

```
# apt install rename
```

[35] El comando mmv permite copiar, mover, hacer ligas simbólicas de múltiples archivos, se instala usando:

```
# apt install mmv
```

rm (de remove o remover), es el comando necesario para borrar un archivo o directorio. Para borrar el archivo *prueba.txt* ubicado en */home*, ejecutamos:

```
$ rm /home/prueba.txt
```

En algunos casos necesitamos borrar el contenido de un directorio menos un archivo, podemos usar:

```
$ rm -v !("archivo.txt")
```

o borrar todo el contenido de un directorio menos algunos archivos, podemos usar:

```
$ rm -v !("archivo1.txt"|"archivo2.txt")
```

o borrar todo el contenido de un directorio menos, por ejemplo los *.zip, usar:

```
$ rm -v !(*.zip)
```

o borrar todo el contenido de un directorio menos, por ejemplo los *.zip y *.txt, usar:

```
$ rm -v !(*.zip|*.txt)
```

Este comando también presenta varias opciones. La opción *-r* borra todos los archivos y directorios de forma recursiva. Por otra parte, *-f* borra todo sin pedir confirmación. Estas opciones pueden combinarse causando un borrado recursivo y sin confirmación del directorio que se especifique. Para realizar esto en el directorio respaldos ubicado en el */home*, usamos:

```
$ rm -rf /home/respaldos
```

Este comando es muy peligroso, por lo tanto es importante que nos documentemos bien acerca de los efectos de estas opciones en nuestro sistema para así evitar consecuencias nefastas.

ln permite crear enlaces a los archivos, tanto duros (Hard Links) como simbólicos *-s* (Soft Links). En pocas palabras, un enlace simbólico es como un acceso directo en Windows o un alias en Mac OS X mientras que un enlace duro es un nombre diferente para la misma información en disco. 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, ubicación, entre otros-.

Para crear un enlace duro usamos:

```
$ ln archivo_origen nombre_enlace
```

para conocer el *inodo* de un archivo, usamos:

```
$ ls -li archivo
```

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
```

En caso de que se necesite actualizar un enlace duro con otro archivo podemos usar:

```
$ ln -f nuevo enlace
```

Para crear un enlace simbólico, hacemos:

```
$ ln -s nombre nombre_enlace
```

si bien para conocer a donde apunta un enlace simbólico podemos usar *ls -l*, también podemos usar:

```
$ readlink nombre_enlace
```

en caso de necesitar actualizar un enlace simbólico existente con otro archivo podemos usar:

```
$ ln -fs nuevo ligaexistente
```

En ciertas aplicaciones (como en Dockers), los enlaces simbólicos no son vistos o permitidos, una opción para corregir esto es usar el comando mount para montar:

```
# mount –bind nombre nombre_enlace
```

y el comando umount para desmontar:

```
# umount nombre_enlace
```

Si queremos conocer el número de inodos (usados y libres) del disco podemos usar:

```
$ df -i
```

Si por alguna razón el archivo al que apunta la liga simbólica se borra, esto genera una liga rota, para encontrar las ligas rotas en nuestro árbol de archivos, usamos:

```
$ find . -xtype l
```

y si queremos conocer todas las ligas y a dónde apuntan, usamos:

```
$ find . -type l -print | xargs ls -ld | awk '{print $9 $10 $11}'
```

unlink sirve para remover un archivo, en caso de que el archivo sea un enlace creado por *ln*, sólo borra el enlace no el archivo, por ejemplo:

```
$ unlik nombre_enlace
```

cat (de concatenar) nos permite visualizar el contenido de uno o más archivos de texto sin la necesidad de un editor. Para utilizarlo solo debemos indicarlo junto al archivo(s) que deseamos visualizar:

```
$ cat prueba.txt
```

podemos pedir que enumere cada línea del archivo mediante:

```
cat -n prueba.txt
```

además podemos usar -e para que nos muestre un $ al final de cada línea y -T para que muestre los tabuladores con el carácter ^I.

También está el comando *tac* que muestra primero la última línea, hasta la primera línea del archivo:

```
$ tac prueba.txt
```

touch crea un archivo vacío, si el archivo existe sólo actualiza la hora de modificación. Por ejemplo, para crear el archivo prueba1.txt en */home/antonio/*, sería:

```
$ touch /home/antonio/prueba1.txt
```

cmp compara el contenido de dos archivos y devuelve 0 si los archivos son idénticos ó 1 si los archivos tienen diferencias. En caso de error devuelve -1.

```
$ cmp -s archivo1 archivo2
```

también puede mostrar algo de información sobre las diferencias (otra comando más completo es *comm*) pero para un reporte más detallado tenemos el siguiente comando:

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

```
$ diff archivo1.txt archivo2.txt
```

si necesitamos que no se distingan mayúsculas de minúsculas podemos usar:

```
$ diff archivo1.txt archivo2.txt -i
```

si necesitamos que no se distingan tabuladores de espacios podemos usar:

```
$ diff archivo1.txt archivo2.txt -E
```

si los archivos tienen líneas muy grandes, es posible usar el comando *fold* para cortar las líneas (por omisión a 80 caracteres), por ejemplo:

```
$ diff <(fold -s archivo1.txt) <(fold -s archivo2.txt)
```

También puede usarse con directorios. En este caso comparará los nombres de los archivos correspondientes en cada directorio por orden alfabético e imprimirá en pantalla los archivos que estén en un directorio pero no estén en el otro (otra opción es *sdiff* que permite mostrar las diferencias entre dos archivos y combinar sus contenidos de forma interactiva).

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

```
$ wc archivo.txt
```

podemos solo pedir que cuente el número de líneas usando *-l*, para el número de caracteres usamos *-c* o para el número de palabras usamos *-w*.

tail muestra en pantalla las últimas 10 líneas de un archivo:

```
$ tail archivo.txt
```

pero podemos indicarle un número diferente de líneas a visualizar usando el parámetro *-n*:

```
$ tail -n 30 archivo.txt
```

si el archivo solo tiene 43 líneas, podemos pedir que solo visualice la 42 y 43, mediante:

```
$ tail -n +42 archivo.txt
```

por último, si el archivo es generado por el sistema de mensajes de Linux(un archivo Log), podemos ver como se van generando sus entradas en tiempo real, usando:

```
$ tail -f /var/log/archivo.log
```

head es el comando opuesto a *tail*, muestra las primeras líneas de un archivo.

```
$ head archivo.txt
```

al igual que *tail*, muestra por defecto las 10 primeras líneas pero podemos indicarle un número diferente usando el parámetro *-n*:

```
$ head -n 50 archivo.txt
```

o que muestre todo excepto las últimas N líneas, usando:

```
$ head -n -15 archivo.txt
```

también es posible visualizar las primeras líneas de múltiples archivos usando:

```
$ head -n 2 archivo1.txt archivo2.txt
```

a la salida de este último comando le podemos quitar el nombre del archivo, mediante:

```
$ head –q n 2 archivo1.txt archivo2.txt
```

Podemos combinarlo con *tail* para mostrar sólo una determinada línea de un archivo (digamos la 13), usando:

```
$ head -n 13 archivo.txt | tail +13
```

y para mostrar un rango de líneas (digamos de la 10 a 15), usamos:

```
$ head -n 15 archivo.txt | tail -n +10
```

por último, podemos indicarle que solo nos muestre los primeros caracteres, mediante -C<número>, ejemplo:

```
$ head -C5 archivo.txt
```

more es un filtro que permite paginar el contenido de un archivo para que se vea a razón de una pantalla a la vez. Para utilizarlo simplemente ejecutamos:

```
$ more archivo.txt
```

para navegar a través del contenido del archivo usamos las flechas direccionales *Arriba* y *Abajo*, *Espacio* o la tecla *Enter* y para buscar una cadena usamos */*. Para salir de more usamos la tecla *q*. Además, podemos indicar que salte hasta donde se encuentra una determinada cadena usando *+/cadena*, o iniciar a partir de la línea *+n* del texto, por ejemplo:

```
$ more +/cadena archivo.txt
$ more +30 archivo.txt
```

less Aunque su nombre es lo opuesto de more es realmente una versión mejorada de éste último. Es otro filtro que permite paginar el contenido de un archivo pero que además de permitir la navegación hacia adelante y hacia atrás, esta optimizado para trabajar con archivos muy grandes. Ejecutarlo es tan simple como escribir:

```
$ less archivo.txt
```

permite navegar a través del contenido del archivo usando las flechas direccionales *Arriba* y *Abajo*, *Espacio* o la tecla *Enter*. Para salir de *less* también usamos la tecla *q*.

paste unifica (en salida estándar) dos o más archivos de texto, fusionando líneas de manera que las entradas en la primera columna pertenecen al primer archivo, las de la segunda columna son para el segundo archivo (separadas por tabuladores), y así sucesivamente, ejemplo:

```
$ paste uno.txt dos.txt tres.txt
```

se puede definir delimitadores entre las entradas de cada fila resultante, usando *-d <separador>*, ejemplo:

```
$ paste -d : uno.txt dos.txt tres.txt
```

además, se puede cambiar la operación de fusión, para que se realice por filas, de manera que el primer renglón son las entradas del primer archivo (separadas por tabuladores), el segundo renglón son las entradas del segundo archivo, y así sucesivamente, para ello se emplea la opción *-s*, ejemplo:

```
$ paste -s uno.txt dos.txt tres.txt
```

cut se encarga de cortar columnas o campos seleccionados de uno o más ficheros (o entrada estándar), por ejemplo para conocer los usuarios del sistema y su directorio de trabajo, usamos:

```
$ cut -d ":" -f 1,6 /etc/passwd
```

para indicar el delimitador usamos -d y para indicarle las columnas a visualizar usamos -f.

tr se encarga de sustituir un delimitador por otro de uno o más ficheros (o entrada estándar), por ejemplo:

Se sustituye el delimitador ":" por un tabulador:

```
$ cut -d ":" -f 1,6 /etc/passwd | tr -s ":" "\t"
```

o elimina las nuevas líneas (\n) de un archivo, usando:

```
$ tr -d \n < entrada.txt > salida.txt
```

entre otras opciones podemos pedir que cambie de mayúsculas a minúsculas o viceversa, usando:

```
$ echo "Esto es una prueba" | tr '[:lower:]' '[:upper:]'
$ echo "Esto es una prueba" | tr '[:upper:]' '[:lower:]'
```

también podemos transformar un conjunto de caracteres (aeiout) en otro ( ), ejemplo:

```
$ echo "Esto es una prueba" | tr '[aeiout]' ' '
```

o hacer el opuesto, es decir, transformar cualquier carácter que no este en conjunto (aeiout) en otro ( ), ejemplo:

```
$ echo "Esto es una prueba" | tr -c '[aeiout]' ' '
```

podemos borrar los caracteres indicados (minúsculas), ejemplo:

```
$ echo "Esto Es Una Prueba" | tr -d '[:lower:]'
```

o cambiar las vocales de mayúsculas a minúsculas, ejemplo:

```
$ echo "Esto Es Una Prueba" | tr 'aeiou' 'AEIOU'
```

es posible el reemplazo de rango de caracteres (a-e) o números(1-4) por otro (x), ejemplos:

```
$ echo "Esto es una prueba" | tr 'a-e' 'x'
$ echo "5uch l337 5p34k" | tr '1-4' 'x'
```

y podemos indicar que reemplace una o múltiples ocurrencias (espacios) por una sola, ejemplo:

```
$ echo "Esto       es          una       prueba" | tr -s ' '
```

fold se usa para que todas las líneas de un archivo se dividan a un ancho especificado, ejemplo:

```
$ cat /etc/services | fold -w 20
```

se puede usar -s para solicitar que si puede corte sobre un espacio, ejemplo:

```
$ cat /etc/services | fold -sw 20
```

nano Es un pequeño editor de texto que esta disponible en casi todas las distribuciones actuales de GNU/Linux[36], funciona con un menú en la parte de inferior que se activa con la tecla *Ctrl* (por ejemplo para grabar se usa *Ctrl-o* y para salir *Ctrl-x*).

```
$ nano archivo.txt
```

Comandos Generales

man muestra la documentación completa de todos los comandos. Por ejemplo, para ver la documentación del comando *clear*:

```
$ man clear
```

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
```

[36] Existe una versión moderna de este editor que no viene instalada por omisión, para instalarla usamos:

```
# apt install micro
```

y su uso es similar:

```
$ micro archivo.txt
```

whatis proporciona una ayuda breve de lo que hacen los comandos, sin mostrar sus opciones, ejemplo:

```
$ whatis ls
```

clear es un sencillo comando que limpiará nuestra terminal por completo dejándola como recién abierta. Para ello escribimos:

```
$ clear
```

w nos proporciona la lista de los usuarios activos en la computadora -recordemos que Linux es un sistema multiusuario-, su uso es:

```
$ w
```

time proporciona el tiempo de ejecución, que es dividido en real, usuario y del sistema, muestra de su uso es la siguiente:

```
$ time ls
```

whoami (del inglés Who Am I o Quien Soy Yo en español) muestra el identificador del usuario actual, para ejecutarlo solo basta con invocarlo:

```
$ whoami
```

date nos muestra la fecha y hora que tiene actualmente la computadora, ejemplo:

```
$ date
```

es posible formatear la salida del comando mediante:

```
$ date + "Weekday: %A Month: %B"[37]
$ date + "Week: %V Year: %y"
$ date -d last-week[38]
```

[37] Para conocer las diferentes banderas de date, usar:

```
$ date - -help
```

[38] podemos usar: tomorrow, last-year, next-year, next-month, entre otras.

cal muestra el calendario[39] del mes actual, con -y nos muestra el calendario del año completo, con -jy nos muestra el calendario con el número de día del año y con -A *n* y -B *m* nos muestra el calendario de *m* meses antes de la fecha actual y *n* meses después, ejemplo:

```
$ cal -A 3 -B 2
```

uptime muestra el tiempo que el equipo de cómputo ha pasado encendido sin ser reiniciado, así como el Load Average (carga promedio del sistema) que es el número de trabajos que se han realizado en los últimos 1, 5 y 15 minutos. Para ver su salida, solo escribimos en la terminal:

```
$ uptime
```

uname es un programa de sistemas operativos de tipo Unix que imprime detalles de la máquina y del sistema operativo que se esta ejecutando. Su salida es diferente dependiendo de las opciones, por ejemplo, uname solo muestra el nombre del sistema operativo pero cuando le pasamos la opción *-r* muestra la versión del Kernel y con *-a* de *all*, su salida es mucho mas completa. Se ejecuta de la siguiente forma:

```
$ uname -a
```

df nos muestra información de los discos y particiones en ellos, además de cuanto esta usado y libre en Bytes en cada una de las particiones, para ver la salida usando información en unidades Gb, Mb y Kb usamos la opción -h, para conocer el numero de inodos disponibles usamos la opción -i, podemos también conocer la información de todos los sistemas de ficheros usando -a y para conocer la información de los sistemas de archivos usamos -T, ejemplo:

```
$ df -hT
```

[39] Otra opción es *ncal*, que se puede instalar usando:

```
# apt install ncal
```

y se usa de forma similar a *cal*.

du nos muestra en Bytes cuanto ocupan los directorios de nuestra trayectoria actual de archivos, usamos la opción *-h* para que muestra el tamaño en unidades fáciles de leer -como KB, MB o GB-, *-a* para conocer el tamaño de archivos y directorios y *-s* para conocer el total de la trayectoria, ejemplo:

```
$ du -sh
```

free nos muestra la cantidad de memoria y Swap usada y libre del sistema, ejemplo:

```
$ free
```

podemos usar los modificadores para ver en Bytes *-b*, ver en Kilobytes *-k*, ver en Megabytes *-m*, ver en Gigabytes *-g*, desplegar una línea con los totales *-t*, ver en intervalos regulares *-s tiempo*, que nos muestre las estadísticas de bajo y alto uso *-l*.

bc es un lenguaje que soporta números de precisión arbitraria con ejecución interactiva, ejemplo:

```
$ bc -l
```

al escribir, por ejemplo:

```
scale = 100
1/3
quit
```

mostrará el resultado con 100 dígitos de precisión.

history muestra el historial de comandos ejecutados en la terminal, ejemplo:

```
$ history
```

para borrar dicho historial se usa la opción *-c*, ejemplo:

```
$ history -c
```

echo sirve para mostar texto en la pantalla, su uso es el siguiente

```
$ echo "Esto es un texto"
```

permite con la opción -e la interpretación de caracteres de escape como retroceso \b, nueva línea \n, tabulador \t, tabulador vertical \v, retorno de carro \r y diagonal invertida \\, ejemplo:

```
$ echo "Esto \ves un \ttexto"
```

Permisos

chmod (del inglés change mode) es un comando que permite cambiar los permisos de acceso de un directorio o archivo. Su sintaxis es:

```
$ chmod [opciones] <modo> <archivo>
```

donde *opciones* nos permite entre otras cosas, cambiar los permisos recursivamente para un directorio con *-R*, el *modo* son los permisos de lectura, escritura y ejecución representados en notación octal[40] y *archivo* es el nombre del directorio o archivo que queremos modificar[41].

Por ejemplo, para asignar permisos de lectura, escritura y ejecución para el dueño, el grupo y remover los permisos para el resto de los usuarios al archivo *prueba.txt* sería:

```
$ chmod 770 prueba.txt
```

[40]

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

[41] 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.

chown (del inglés change owner) nos permite cambiar el propietario de un archivo o directorio. Su sintaxis es:

$ chown [opciones] <nuevo-propietario> <archivo>

donde *opciones* son las opciones del comando, como *-R* para cambiar recursivamente el propietario de un directorio y todo su contenido, *nuevo-propietario* será el nuevo propietario y archivo es el nombre del directorio o archivo que queremos modificar.

Por ejemplo, para cambiarle el propietario del directorio */home/ejercicios* y todo su contenido y asignarlo al usuario *pedro*, hacemos:

$ chown -R pedro /home/ejercicios

También es posible hacer el cambio de propietario y grupo en un solo comando, usando:

$ chown -R pedro:pedro /home/ejercicios

chgrp (del inglés change group) nos permite cambiar el grupo de un archivo o directorio. Su sintaxis es:

$ chgrp [opciones] <nuevo-grupo> <archivo>

donde *opciones* son las opciones del comando, como *-R* para cambiar recursivamente el grupo de un directorio y todo su contenido, *nuevo-grupo* será el nuevo grupo y *archivo* es el nombre del directorio o archivo que queremos modificar.

Por ejemplo, para cambiarle el grupo del directorio */home/ejercicios* y todo su contenido y asignarlo al usuario *pedro*, hacemos:

$ chgrp -R pedro /home/ejercicios

su permite cambiar las credenciales del usuario, es decir ser otro usuario, por ejemplo:

$ su antonio

el usuario del que comúnmente se desea adquirir sus credenciales es el de *root*, para ello usamos:

$ su

useradd (de agregar usuario) se utiliza para crear nuevos usuarios en tu sistema, su sintaxis es:

$ useradd [opciones] <nombre-usuario>

donde *opciones* nos permite asignar un grupo al usuario con *-g*, asignar el directorio */home* con *-d*, crearlo con *-m* si no existía previamente y *-s* para asignarle un intérprete de comandos o Shell, entre otras.

Así, para crear el usuario andrea cuyo grupo principal será *editores*, ejecutamos:

$ useradd -g editores -d /home/andrea -m -s /bin/bash andrea

usermod (de modificar usuario) modifica algunos parámetros de un usuario existente, como el nombre, su directorio /home y los grupos a los que pertenece, entre otros. Su sintaxis es:

$ usermod [opciones] <nombre-usuario>

donde *opciones* cambia el directorio home con *-d*, mueve todo el contenido del directorio anterior con *-m* y cambia el nombre de usuario con *-l*, entre otras. Para cambiar el nombre al usuario *andrea* por *violeta*, sería:

$ usermod -l violeta andrea

deluser (del inglés delete user) es un sencillo comando para borrar usuarios. Tiene la opción *-r* que adicionalmente borra su directorio */home*. Para borrar el usuario *violeta* con su */home*, ejecutamos:

$ deluser -r violeta

passwd (del inglés password) es una utilidad que se usa para cambiar o generar la contraseña de un usuario existente. Al invocarlo, pedirá la contraseña actual (si existe) y luego que la contraseña nueva sea introducida dos veces para verificar que fue escrita correctamente. Por ejemplo para asignar una contraseña al usuario *antonio*, sería:

$ passwd antonio

para conocer la información del Password de mi cuenta puedo usar:

$ passwd -S antonio

lsattr permite listar los atributos[42] asignados a los ficheros y directorios, para ver los atributos del directorio actual usamos:

```
$ lsattr -a
```

o de forma recursiva

```
$ lsattr -Ra
```

chattr permite cambiar los atributos asignados a los ficheros y directorios, las opciones se agregan con + y se quitan con - y podemos hacerlo de forma recursiva *R*, por ejemplo podemos poner el permiso de inmutable, es decir, que no puede eliminar ni cambiar de nombre, mediante:

```
# chattr +i archivo
```

o quitar el permiso:

```
# chattr -i archivo
```

Podemos deshabilitar la modificación de la fecha de acceso al fichero (atime) mediante:

```
# chattr +A archivo
```

[42] Algunos valores posibles son: *(A)* indica que el valor de la fecha de acceso sobre un archivo no será cambiado en cada lectura, *(a)* el archivo solo puede ser abierto en edición para escritura por redireccionamiento (>>) y no puede ser eliminado, *(c)* el archivo tiene activa la compresión de datos, *(D)* indica que los datos escritos en un directorio se sincronizan en el disco de forma automática, *(d)* elimina el fichero o directorio de las copias de seguridad realizadas con la utilidad *dump*, *(e)* los archivos con este atributo están usando extensiones para mapear los bloques en el disco, (E) el sistema de archivos cifra el archivo, directorio o enlace simbólico que tenga este atributo, *(I)* indica que la carpeta está indexada, *(i)* indica que el fichero o directorio es de solo lectura y no puede ser modificado, borrado, renombrado o ligado simbólicamente ni siquiera por el usuario *root*, *(j)* indica que esta activo el "journaling" de los archivos, (S) indica que el archivo es síncrono, las escrituras en el archivo son inmediatamente efectuadas, *(s)* indica que cuando el archivo sea eliminado, el espacio que ocupaba será rellenado por bloques de ceros, *(T)* indica que el directorio con este atributo se escribirá en las partes más rápidas del disco, *(t)* indica que el fichero no presenta fragmentación en el sistema de ficheros, *(u)* indica que los archivos con este atributo pueden ser recuperados después de ser borrados, *(V)* los archivos con este atributo no se podrá escribir en el y el sistema de archivos verificará automáticamente los datos leídos.

esto permite que cuando se acceda al fichero no se modifique el registro de atime. De este modo no quedarán registrada la fecha de último acceso.

Es posible comprimir automáticamente el fichero en disco por el Kernel y cuando se lea el archivo se verá descomprimido, ejemplo:

```
# chattr +c archivo
```

Además se puede permitir la recuperación de un archivo aunque sea eliminado, usando:

```
# chattr +u archivo
```

y el caso opuesto, al eliminar un archivo, sobrescribir con ceros todos sus bloques, mediante:

```
# chattr +e archivo
```

Podemos establecer que el archivo solo se pueda abrir en modo de adición para escritura, mediante:

```
# chattr +a archivo
```

id es un programa que muestra el identificador del usuario (UID) que ejecuta el programa, el identificador de su grupo (GID), así como todos los grupos a los cuales pertenece el usuario, ejemplo:

```
$ id usuario
```

Búsqueda

find permite buscar[43] dentro del sistema de archivos un directorio o archivos que concuerden con el patrón dado, por ejemplo:

[43] Otra opción al comando *find* es el comando *fdfind*, el cual se puede instalar usando:

```
# apt install fd-find
```

y podemos usarlo mediante:

```
$ fdfind debian
```

```
$ find /home -name *.pdf
```

busca desde la trayectoria */home*, todos los archivos que concluyan con *.pdf* y nos muestra las trayectorias a los archivos que concuerdan con lo solicitado. También podemos pasar la salida de *find* a otro comando como por ejemplo:

```
$ find . -name Portada.pdf | xargs ls -l
```

que nos mostrará la salida larga del comando *ls* de todos los archivos que encuentre *find*.

Además, podemos hacer uso de expresiones regulares como:

```
$ find . -regex "[a-f0-9\-]\{36\}\.jpg"
```

grep permite buscar[44] en archivos un determinado patrón[45], mostrando la línea que lo contiene, por ejemplo:

busca en todos los archivos **.txt* la cadena *chmod:*

```
$ grep chmod *.txt
```

si necesitamos hacer la búsqueda sobre todos los archivos de la carpeta actual y todas sus subcarpetas, podemos usar:

```
$ grep -r chmod *.txt
```

si requerimos seguir sobre todos los enlaces simbólicos, usamos:

[44] El comando grep comparte funcionalidad con egrep y fgrep. ¿Cuál es la diferencia entre grep, egrep y fgrep?, la e en egrep significa extendido (grep -E) y la f en fgrep significa fijo (grep -F). egrep permite el uso de expresiones regulares extendidas y fgrep no permite la expresión regular en absoluto.

[45] Grep es una herramienta muy poderosa, pero existe otra: ripgrep, que es eficiente en búsquedas recursivas, combinando con opciones avanzadas como el filtrado, uso de expresiones regulares, colores, soporte Unicode. Se instala usando:

```
# apt install ripgrep
```

y lo usamos, por ejemplo buscando foobar en los archivos *.py:

```
$ rg -tpy foobar
```

```
$ grep -R chmod *.txt
```

para que nos indice el nombre de archivo en los cuales se encontró la cadena *main*:

```
$ grep -l main *.java
```

busca en todos los archivos **.log* la cadena *error*, ignorando si esta en mayúsculas o minúsculas:

```
$ grep -i error *.log
```

busca en todos los archivos **.log* aquellas que no tengan la cadena *error*:

```
$ grep -v error *.log
```

busca en todos los archivos **.log* aquellos que tengan a la cadena *error* o *fatal:*

```
$ grep 'error\|fatal' *.log
```

busca en todos los archivos **.log* aquellos que tengan a la cadena *error* o *fatal:*

```
$ grep -E 'error\|fatal' *.log
```

busca en todos los archivos **.log* aquellos que tengan a la cadena *error* o *fatal* e indica cuantas coincidencias encontró*:*

```
$ grep -c 'error\|fatal' *.log
```

busca en todos los archivos **.log* aquellos que tengan a la cadena *error* o *fatal*, marcando en color las cadenas encontradas:

```
$ grep –color -E "error|fatal" *.log
```

regresa las líneas que no tengan la cadena indicada:

```
$ grep -v // Ejemplo.java
```

regresa las líneas que no contienen la cadena indicada y no son vacías:

```
$ grep -Ev "^//|^$" Ejmplo.java
```

Podemos usar ^ y $ para forzar a que se encuentre solo al inicio o al final de la línea respectivamente, por ejemplo:

```
$ grep ^antonio /etc/passwd
$ grep sh$ /etc/passwd
```

o buscar una línea que solo contenga una palabra, por ejemplo:

```
$ grep '^texto$' archivo
```

o líneas en blanco usando:

```
$ grep '^$' archivo
```

También podemos indicar que busque usando rango de caracteres mediante[46]:

```
$ grep '[Aa]ntonio' archivo
$ grep '[Aa][Nn]tonio[0-9]' archivo
$ grep '[A-Z][a-z]' archivo
$ grep [[:upper:]] archivo
```

locate permite buscar archivos o directorios cuyo nombre coincida con el patrón dado[47], por ejemplo:

[46] Podemos usar
[[:alnum:]] - Caracteres alfanuméricos
[[:alpha:]] - Caracteres alfabéticos
[[:blank:]] - Espacio o tabulador
[[:digit:]] - Dígitos
[[:lower:]] - Letras minúsculas
[[:space:]] - Espacio, tabulador
[[:upper:]] - Letras mayúsculas

[47] En caso de no estar instalado en el sistema, lo instalamos usando:

```
# apt install mlocate
```

y para actualizar la base de datos del comando, usamos:

```
# updatedb
```

en caso de haber borrado archivos, es necesario actualizar la base de datos para excluirlos.

```
$ locate *dir2*
```

si necesitamos que la búsqueda sea insensible a mayúsculas y minúsculas usamos:

```
$ locate -i desktop.iso
```

podemos limitar la búsqueda a un determinado número de ocurrencias, por ejemplo:

```
$ locate "*.html" -n 20
```

o solo visualizar el número de ocurrencias encontradas, ejemplo:

```
$ locate -c "*.html"
```

Para conocer las estadísticas del comando *locate*, usamos:

Respaldo

tar permite respaldar en un solo archivo un grupo de archivos y/o directorios sin compactarlos, para ello usar:

```
$ tar -cvf nombre.tar directorio
```

para restaurar usar:

```
$ tar -xvf nombre.tar
```

gzip permite respaldar en un solo archivo un grupo de archivos y/o directorios compactándolos usando *gzip*, para ello usamos:

```
$ tar -cvf nombre.tar directorio
$ gzip –best nombre.tar
```

o en un solo paso, usamos:

```
$ tar -zcvf nombre.tar.gz directorio
```

para restaurar se usar:

```
$ tar -zxvf nombre.tar.gz
```

o alternativamente podemos usar:

```
$ gunzip nombre.tar.gz
$ tar -xvf nombre.tar
```

con *gzip* podemos sólo comprimir o descomprimir respectivamente usando:

```
$ gzip fichero
$ gzip -d fichero.gz
```

bzip2 permite respaldar en un solo archivo un grupo de archivos y/o directorios compactándolos usando *bzip2*, para ello escribimos:

```
$ bzip fichero
$ bzip -d fichero.bz2
```

zip permite respaldar en un solo archivo un grupo de archivos y/o directorios compactándolos, para ello usamos:

```
$ zip archivo.zip fichero(s)
$ zip -r archivo.zip directorio/
```

también permite establecer el nivel 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 -r -m 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, mediante:

```
$ zip -e archivo.zip fichero(s)
```

o proteger con clave un *.zip* ya existente, usando:

```
$ zipcloak archivo.zip
```

Podemos ver la información detallada del archivo comprimido, mediante:

```
$ zipdetails archivo.zip
```

y descomprimir mediante:

```
$ unzip archivo.zip
```

también podemos indicar el directorio en en cual descomprimir, usando:

```
$ unzip archivo.zip -d /home/usuario/temp/
```

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
```

unp permite descomprimir de casi cualquier formato de respaldo, su uso es de lo más sencillo, a saber:

```
$ unp archivo.compactado
```

Varios

file determina el tipo de un archivo y te imprime en pantalla 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.

```
$ file un_archivo_de_texto.txt
```

stat nos da información de un archivo, datos como: tamaño, blocks usados para almacenarlo, número de ligas, datos del dueño y grupo; fechas de acceso, modificación y cambio.

```
$ stat archivo
```

type permite identificar el comando pasado como parámetro indicando la trayectoria si es comando externo o si es comando interno al Shell, ejemplo:

```
$ type ls
```

ps nos muestra los procesos activos del sistema junto con información de la ejecución de los mismos, para ver todos los procesos en el sistema usar -ef, para conocer los procesos de un usuario usamos -U <usuario>, para conocer los procesos de un determinado grupo usamos -G <grupo>, para conocer todos los procesos de un determinado programa usamos -C <programa>, para ver todos los procesos en forma de árbol y saber que proceso depende de que otros, usamos -ejH, ejemplo:

```
$ ps -ejH
```

Los procesos pueden ser: uninterruptible sleep (D), idle (I), running (R), sleeping (S), stopped by job control signal (T), stopped by debugger during trace (t), zombie (Z), los podemos ver usando:

```
$ ps -au
```

Podemos pedirle al comando que nos muestre algunos datos informativos de los comandos en ejecución y mandar a un archivo los procesos que más consumen memoria, ejemplo:

```
ps -eo cmd,pid,ppid,%mem,%cpu –sort=-%mem | head | tee
topprocs.txt
```

si queremos adicionar datos al archivo previamente creado usamos:

```
$ ps -eo cmd,pid,ppid,%mem,%cpu –sort=-%mem | head | tee
-a topprocs.txt
```

kill es un comando utilizado para enviar mensajes sencillos a los procesos ejecutándose en segundo plano en el sistema. Por defecto el mensaje que se envía es la señal de terminación. Su sintaxis más sencilla es:

```
$ kill [-s] <pid>
```

donde *-s* es la señal a enviar, de no ser especificada ninguna se manda la señal por defecto (*SIGTRM*) y *pid* es el identificador del proceso. Otra de sus opciones es *-9* (*SIGKILL*) que fuerza la terminación de un proceso, para conocer los posibles mensajes de *kill* usar:
$ kill -l.
En Linux cada comando, proceso o ventana gráfica tiene un número de proceso *(PID)*, este se puede obtener mediante el comando *ps* o *top*, y el comando *kill* puede concluir con la ejecución del *PID* indicado y todos sus subprocesos -el usuario sólo puede matar sus propios procesos, *root* puede finanizar (matar) los procesos de cualquier usuario-, ejemplo:
Por ejemplo, para terminar un proceso cuyo *PID* es *3477*, ejecutamos:

```
$ kill 3477
```

Otra señal importante es 1 (SIGHUP) que permite parar y reinicializar el proceso indicado, por ejemplo para detener el proceso 2434, usamos:

```
$ kill -1 2434
```

y lo reiniciamos usando:

```
$ kill -1 2434
```

killall permite finalizar (matar) todas nuestras instancias de ejecución de un comando, por ejemplo:

```
$ killall firefox-esr
```

o podemos finalizar la sesión de un usuario, usando:

```
# killall -u antonio
```

pgrep permite conocer los identificadores de proceso de una determinada aplicación corriendo en el sistema, por ejemplo:

```
$ pgrep firefox
```

pwdx permite conocer el directorio de trabajo de una aplicación a través de su identificador de proceso pasado como parámetro a *pwdx*, ejemplo:

```
$ pwdx 4534
```

o

```
$ pwdx $(pgrep firefox)
```

awk permite procesar, analizar archivos de texto que estén organizados por filas y columnas, ejemplo:

$ awk -F':' '{ print $1 }' /etc/passwd

nos mostrarán todos los usuarios que tiene el sistema, los cuales están dados de alta en el archivo del sistema */etc/passwd.*

Si necesitamos visualizar una determinada línea de un archivo (digamos la 5), podemos usar:

```
$ awk 'NR==5' archivo.txt
```

en el caso de necesitar visualizar un rango de líneas de un archivo (digamos de la 20 a 25), usamos:

```
$ awk 'NR>=20 && NR<=25' archivo.txt
```

sort imprime en pantalla las líneas de un archivo ordenadas alfabéticamente. Para ejecutarlo basta con:

```
$ sort archivo.txt
```

podemos solicitar que lo haga en orden inverso usando -r, que haga el ordenamiento numérico usando -n, podemos omitir los duplicados usando -u, ordenar ignorando mayúscula y minúsculas con -f, ordenamiento tomando en cuenta valores alfanuméricos -h, ordenamiento aleatorio con -R, que ordene por meses -M, que haga el ordenamiento de los renglones tomando como índice cierto renglón, por ejemplo:
$ ls -al | sort -k 4 -n

sed es considerado un editor de texto orientado a "flujo" -en contraposición a los clásicos editores «interactivos»- el cual acepta como entrada un archivo o entrada estándar; cada línea es procesada y el resultado es enviado a la salida estándar. Por ejemplo, borrar las líneas tres a cinco de archivo archivo.txt:

```
$ sed '3,5d' archivo.txt
```

otro ejemplo, borrar todas las líneas en blanco (no las que sólo tengan tabuladores y/o espacios) del archivo fichero.txt

```
$ sed '/^$/d' archivo.txt
```

para quitar las líneas en blanco y las que sólo tengan tabuladores y/o espacios, usamos:

```
$ sed sed '/^[ \t]*$/d' archivo.txt
```

si deseamos visualizar una determinada línea de un archivo (digamos la 13), usamos:

```
$ sed -n '13p' archivo.txt
```

si queremos visualizar un rango de líneas de un archivo (digamos de la 20 a 25), usamos:

```
$ sed -n '20,25p' archivo.txt
```

si necesitamos cambiar las vocales de un archivo de minúsculas a mayúsculas, podemos usar:

```
$ sed 'y/aeiou/AEIOU/' archivo.txt
$ sed 's/[aeiou]/\U&/g' archivo.txt
```

md5sum genera la suma de verificación md5 (Compute and Check MD5 Message Digest) o Hash usada para verificar la integridad de los archivos, esta puede haber cambiado como resultado de una transferencia de archivos defectuosa, un error en disco o una interferencia maliciosa, ejemplo[48]:

```
$ md5sum debian.testing-amd64-netinst.iso
```

sleep se utiliza para temporalizar un intervalo de tiempo determinado, la unidad por defecto es el segundo (s), pero se puede usar minutos (m), horas (h) o días (d), por ejemplo:

```
$ sleep 3m && ls -al
```

watch permite correr un comando de forma repetitiva y a intervalos regulares (2 segundos por omisión) mostrando su salida, por ejemplo:

```
$ watch free
```

de ser necesario, podemos quitarle el título, usando:

```
$ watch -t free
```

podemos indicar el intervalo de ejecución usando:

```
$ watch -n 5 free
```

Podemos solicitarle que nos muestre los cambios sobre la última salida, usando:

```
$ watch -n 5 -d free
```

y podemos indicarle que concluya la ejecución si la salida actual es distinta a la anterior, usando:

```
$ watch -n 5 -g free
```

[48] Otras opciones son:

```
$ shasum debian.testing-amd64-netinst.iso
$ shasum -a256 debian.testing-amd64-netinst.iso
```

Monitorear el Desempeño Existen múltiples herramientas para ser usadas en línea de comandos y ambiente gráfico que permiten monitorear el desempeño y uso de una computadora con GNU/Linux, estas se pueden usar para administrar el sistema y las comunicaciones por red, estos comandos[49] están disponibles en todas las distribuciones de GNU/Linux y son normalmente usados para determinar problemas de desempeño en nuestro sistema de cómputo.

lscpu para conocer el tipo de CPU y sus características usamos:

```
$ lscpu
```

podemos usar también *cat /proc/cpuinfo*, si deseamos un análisis más detallado están los comandos:

```
lscpi, cpuid, dmidecode, inxi, hardinfo, lshw, hwinfo, nproc
```

free despliega la memoria total, usada, compartida, en Cache y libre del sistema:

```
$ free
```

podemos usar también cat /proc/meminfo, si deseamos un análisis más detallado están los comandos:

```
top, vmstat, dstat
```

top muestra el desempeño de nuestro equipo actualizando cada segundo el uso del CPU, memoria, Swap, Cache, Buffer y los procesos que están corriendo en el sistema actualmente y en cada proceso que corre se muestra el identificador, el porcentaje de CPU, prioridad y memoria usada, etc. Para usarlo usamos:

```
$ top
```

[49] Algunos comandos son utilizados por cualquier usuario y otros solo por el administrador.

para salir del programa se debe presionar la tecla q. Los procesos pueden ser: uninterruptible sleep (D), idle (I), running (R), sleeping (S), stopped by job control signal (T), stopped by debugger during trace (t), zombie (Z).

Otras variantes de este comando son:

> bashtop, htop, glances, conky, nmon, atop, vtop, gtop, ps, mpstat, collectl, sar, pstree, pmap, pgrep, pkill, kill, killall, xkill, Linux Process Viewer, etc.

En circunstancias particulares, podemos necesitar que un proceso tenga una distinta prioridad de ejecución (menor o mayor del valor por omisión), para ello podemos usar los comando *nice* y *renice* para cambiar dicho valor. Además podemos pedir que un proceso determinado se restrinja a un procesador particular mediante el uso del comando *taskset* que por lo general mejora el rendimiento.

whowatch es una utilidad simple que muestra de forma interactiva y en tiempo real los usuarios y procesos activos en el sistema:

```
$ whowatch
```

Otras opciones son:

> bashtop, htop, glances, conky, nmon, atop, gtop, ps, mpstat, collectl, sar, pstree, pmap, pgrep, pkill, kill, killall, xkill, Linux Process Viewer, etc.

dstat muestra las estadísticas de recursos de todo el sistema de forma versátil en tiempo real:

```
$ dstat -c –top-cpu -dn –top-mem –mem
```

combina la capacidad de comandos como *iostat, vmstat, netstat* e *ifstat.* Otras opciones son:

> nload, collectl, iptraf, nethogs, iftop, mtr, bmon, slurn, tcptrack, monitorix, nmon, glances

vmstat muestra las estadísticas de la memoria virtual, hilos del Kernel, uso de discos, procesos del sistema, entradas y salidas de bloque, interruptores y actividad del CPU, entre otras opciones, este comando esta contenido en el paquete systat:

$ vmstat 1

otras opciones son:

dstat, sar, vnstat, vnstati, mpstat, iostat, iotop, ioping, atop, top, collectl, nmon, glances

netstat permite monitorizar los paquetes de red que entran y salen, genera estadísticas de su uso, es un paquete que permite encontrar problemas de desempeño en las comunicaciones de red:

$ netstat

podemos solicitar que sólo nos muestre lo referente a un solo puerto, usando:

netstat -ltnp | grep -w ':80'

otras opciones son:

dstat, collectl, iptraf, nethogs, iftop, ifstat, mtr, monitorix, nmon, bwm-ng, cbm, speedometer, pkstat, netwatch, trafshow, netload, glances

iotop permite conocer qué procesos están generando actividades de lectura y grabación en los discos del sistema, así es posible conocer qué procesos están sobrecargando el sistema:

$ iotop

otras opciones son:

iostat, ioping, vmstat, atop, htop, dstat, glances, netdata, netstat, nmon, collectl, glances

iostat este permite conocer estadísticas de uso del sistema de entrada/salida incluyendo dispositivos, discos locales, discos remotos tales como *NFS* y *SAMBA*:

```
$ iostat
```

Otras opciones son:

```
iotop, ioping, iostat, atop, dstat, nfsstat, ifstat, atop, nmon,
collectl, glances
```

lsof permite conocer la lista de archivos abiertos además de Sockets Network, Pipes, dispositivos y procesos:

```
$ lsof
```

Lista todos los procesos que tiene abierto el archivo:

```
$ lsof /trayectoria/archivo
```

Lista todos los archivos abiertos por el usuario:

```
$ lsof -u usuario
```

también se pueden indicar múltiples usuarios:

```
$ lsof -u usuario1, usuario2
```

o bién por todos los usuarios menos uno, por ejemplo root:

```
$ lsof -u ^root
```

Lista todos los archivos abiertos en un directorio:

```
$ lsof +D /trayectoria/directorio/
```

Lista todos los archivos abiertos por un identificador de proceso:

```
$ lsof -p <pid>
```

también podemos especificar múltiples identificadores de proceso:

```
$ lsof -p pid1, pid2, pid3
```

Lista todos los archivos abiertos por un comando:

```
$ lsof -c <comando>
```

Busca archivos abiertos por un usuario, comando o proceso:

```
$ lsof -a -u usuario -c comando
```

Lista las conexiones y puertos de red abiertos:

```
$ lsof -i
```

si usamos IPV4 o IPV6 podemos ver esos puertos abiertos:

```
$ lsof -i 4
$ lsof -i 6
```

podemos pedirle que nos muestre puertos tcp o udp:

```
$ lsof -i tcp
```

podemos conocer los procesos que usan el puerto TCP:80, mediante:

```
$ lsof -i TCP:80
```

o para los puertos TCP:1-1024, mediante:

```
$ lsof -i TCP:1-1024
```

Podemos pedirle que nos muestre la actividad de un usuario y que archivos están involucrados, usando:

```
$ lsof -i -u usuario
```

Uno de sus principales usos es conocer qué proceso tiene acceso a un disco o partición que no se puede desmontar y manda un error de que un archivo esta siendo usado, pare ello usamos:

```
$ lsof /dev/sda2
```

También podemos matar toda la actividad de un usuario particular:

```
# kill -9 'lsof -t -u antonio'
```

lsusb lista los dispositivos USB del sistema además información del fabricante del mismo, ejemplo:

```
$ lsusb
```

tcpdump es uno de los comando más usados para analizar paquetes de red y es usado para capturar o filtrar paquetes *TCP/IP* que se reciben o se transfieren en una interfaz de red especifica:
tcpdump -i eth0
también permite grabar los paquetes capturados para un análisis posterior. Otras opciones son:

arpwatch, suricata, wireshark, vnstat, vnstati, negios, collectl, glances, ss, iptraf, nethogs, iftop, mtr

ip muestra información y permite manipular los dispositivos de red (interfaces y tuneles), uso:

```
$ ip address
# ip link set ens3 up
# ip link set ens3 down
```

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

```
$ nmcli
```

2.5 GNU Core Utilities

Coreutils (o GNU Core Utilities) es un paquete de Software desarrollado por el proyecto GNU que contiene varias de las herramientas básicas como cat, ls y rm necesarias para sistemas operativos del tipo Unix. Es una combinación de tres paquetes: utilidades de ficheros (fileutils), utilidades de intérpretes de comandos (shellutils) y utilidades de proceso de textos (textutils).

Las utilidades GNU core soportan opciones de cadena larga como parámetros a los comandos, así como cierta permisividad en la convención al especificar opciones antes de los argumentos regulares (siempre que la variable de entorno POSIXLY_CORRECT esté definida, hecho que habilita una diferente funcionalidad en BSD). Adicionalmente, como la filosofía GNU emplea

información desde páginas de manual (y usa herramientas como info), la información proporcionada es mayor. La siguiente lista enumera algunas utilidades importantes:

Salida de archivos completos

- cat: Concatenar y escribir archivos
- tac: Concatenar y escribir archivos al revés
- nl: Rectas numéricas y archivos de escritura.
- od: escribir archivos en formato octal u otros formatos
- base32: Transformar datos en datos imprimibles
- base64: Transformar datos en datos imprimibles
- basenc: Transformar datos en datos imprimibles

Formatear el contenido del archivo

- fmt: reformatear el texto del párrafo
- pr: Paginar o dividir archivos en columnas para imprimir
- fold: Ajustar las líneas de entrada para que quepan en el ancho especificado

Salida de partes de archivos.

- head: Generar la primera parte de los archivos.
- tail: Generar la última parte de los archivos.
- split: Divide un archivo en pedazos.
- csplit: Dividir un archivo en partes determinadas por el contexto

Archivos resumidos

- wc: Imprimir recuentos de nuevas líneas, palabras y bytes
- sum: Imprimir suma de comprobación y recuentos de bloques

- cksum: Imprimir y verificar sumas de verificación de archivos
- md5sum: Imprima o consulte resúmenes de MD5
- b2sum: Imprima o consulte resúmenes de BLAKE2
- sha1sum: Imprima o consulte resúmenes SHA-1
- sha2: imprima o consulte resúmenes SHA-2

Operar con archivos ordenados

- sort: Ordenar archivos de texto
- shuf: Texto aleatorio
- uniq: Unificar archivos
- comm: Compara dos archivos ordenados línea por línea
- ptx: producir índices permutados
- tsort: clasificación topológica

Operando en el campo

- cut: Imprimir partes seleccionadas de líneas
- paste: Fusionar líneas de archivos
- join: Unir líneas en un campo común

Operando con personajes

- tr: Traducir, comprimir y/o eliminar caracteres
- expand: Convertir tabulaciones a espacios
- unexpand: Convertir espacios en pestañas

Listado de directorio

- ls: Listar el contenido del directorio

- dir: enumerar brevemente el contenido del directorio
- vdir: enumerar detalladamente el contenido del directorio
- dircolors: Configuración de color parals

Operaciones básicas

- cp: Copiar archivos y directorios
- dd: Convertir y copiar un archivo
- install: Copiar archivos y establecer atributos
- mv: Mover (cambiar nombre) archivos
- rm: Eliminar archivos o directorios
- shred: Elimina archivos de forma más segura

Tipos de archivos especiales

- link: Realizar un enlace físico a través de la llamada al sistema de enlace
- ln: Hacer enlaces entre archivos
- mkdir: Crear directorios
- mkfifo: Crear FIFO (canalizaciones con nombre)
- mknod: Crear archivos especiales de bloques o caracteres
- readlink: Imprimir valor de un enlace simbólico o nombre de archivo canónico
- rmdir: Eliminar directorios vacíos
- unlink: Eliminar archivos mediante la llamada al sistema de desvinculación

Cambiar los atributos del archivo

- chown: Cambiar propietario y grupo de archivos
- chgrp: Cambiar la propiedad del grupo
- chmod: Cambiar permisos de acceso
- touch: Cambiar marcas de tiempo de archivos

Uso del espacio de archivos

- df: Informar del uso de espacio del sistema de archivos
- du: Estimar el uso del espacio de archivos
- stat: Informe del estado del archivo o del sistema de archivos
- sync: Sincronizar escrituras en caché con almacenamiento persistente
- truncate: Reducir o ampliar el tamaño de un archivo

Imprimir texto

- echo: Imprimir una línea de texto
- printf: Formatear e imprimir datos
- yes: Imprime una cadena hasta que se interrumpa

Condiciones

- false: No hacer nada, sin éxito.
- true: No hacer nada, con éxito
- test: Verifique tipos de archivos y compare valores
- expr: Evaluar expresiones

Redirección

- tee: Redirigir la salida a múltiples archivos o procesos

Manipulación de nombres de archivos

- basename: Eliminar directorio y sufijo de un nombre de archivo
- dirname: Eliminar el componente del último nombre del archivo
- pathchk: Verificar la validez y portabilidad del nombre del archivo
- mktemp: Crear archivo o directorio temporal
- realpath: Imprime el nombre del archivo resuelto.

Contexto de trabajo

- pwd: Imprimir directorio de trabajo
- stty: Imprimir o cambiar características del terminal
- printenv: Imprimir todas o algunas variables de entorno
- tty: Imprimir el nombre del archivo del terminal en la entrada estándar

Información del usuario

- id: Imprimir identidad del usuario
- logname: Imprimir el nombre de inicio de sesión actual
- whoami: Imprimir nombre de usuario efectivo
- groups: Imprimir nombres de grupos en los que se encuentra un usuario
- users: Imprimir nombres de inicio de sesión de los usuarios actualmente conectados
- who: Imprimir quién está conectado actualmente
- pinky: Imprimir información sobre los usuarios

Contexto del sistema

- date: Imprimir o configurar la fecha y hora del sistema
- arch: Imprimir el nombre del hardware de la máquina
- nproc: Imprime el número de procesadores disponibles.

- uname: Imprimir información del sistema
- hostname: Imprimir o configurar el nombre del sistema
- hostid: Imprimir identificador de host numérico
- uptime: Tiempo de actividad y carga del sistema de impresión

Contexto SELinux

- chcon: Cambiar el contexto SELinux del archivo
- runcon: Ejecutar un comando en el contexto SELinux especificado

Invocación de comando modificada

- chroot: Ejecute un comando con un directorio raíz diferente
- env: Ejecutar un comando en un entorno modificado

Expansión de variables ambientales

- nice: Ejecutar un comando con amabilidad modificada
- nohup: Ejecute un comando inmune a los cuelgues
- stdbuf: Ejecute un comando con almacenamiento en búfer de flujo de E/S modificado
- timeout: Ejecutar un comando con un límite de tiempo

Control de procesos

- kill: Enviar una señal a los procesos.

Retraso

- sleep: Retraso por un tiempo específico

Operaciones numéricas

- factor: Imprimir factores primos
- numfmt: Reformatear números
- seq: Imprimir secuencias numéricas

2.6 Desde la Nube

Existen diferentes servicios Web[50] que permiten instalar, configurar y usar cientos de sistemas operativos Linux y Unix -máquinas virtuales usando servicios Web en Debian GNU/Linux y QEMU- desde el navegador, esto en aras de que los usuarios que cuenten con algún sistema de acceso a red y un navegador puedan usar, configurar e instalar algún sistema operativo y su respectiva paquetería sin hacer instalación alguna en su equipo de cómputo, tableta o teléfono celular[51].

Una muestra de estos proyectos son: Distrotest (https://distrotest.net), JSLinux (https://bellard.org/jslinux) y OnWorks (https://www.onworks.net).

Algunas versiones listas para usar son:

4mLinux, AbsoluteLinux, Academix, AlpineLinux, Antergos, antiX Linux, Aptosid, ArchBang, ArchLabs, ArchLinux, Archman, ArchStrike, ArcoLinux, ArtixLinux, AryaLinux, AV Linux, BackBoxLinux, BigLinux, Bio-Linux, BlackArch, BlackLab, BlackPantherOS, BlackSlash, blag, BlankOn, Bluestar, Bodhi, BunsenLabs, ByzantineOS, Caine, Calculate Linux Desktop, CentOS, Chakra, ChaletOS, ClearOS, Clonezilla, ConnochaetOS, Cucumber, Damn Small Linux, Damn Small Linux Not, Debian, DebianEdu, deepin, DEFT, Devil-Linux, Devuan, DragonFly BSD, Dragora, DuZeru, Dyne:bolic, Edubuntu, elementaryOS, Elive Linux, Emmabuntüs, Emmi OS, Endless OS, EnsoOS, Exe GNU/Linux, ExTiX, Fatdog64, Fedora Atomic, Fedora Server, Fedora Workstation, FerenOS, FreeBSD, FreeDOS, Frugalware, G4L, GeckoLinux, Gentoo, GNewSense, GoboLinux, Gparted, GreenieLinux, GRML, GuixSD, Haiku, Heads, Kali Linux, Kanotix, KaOS, Knoppix, Kodachi, KolibriOS, Korora, Kubuntu, Kwort, Linux Lite, Linux Mint, LiveRaizo, LMDE, Lubuntu, LXLE OS, Macpup, Mageia, MakuluLinux, Manjaro, Matriux, MauiLinux, MenuetOS, MinerOS, MiniNo, Modicia, Musix, MX Linux, Nas4Free, Neptune, NetBSD, Netrunner, NixOs, NST, NuTyX, OpenIndiana, OpenMandriva, openSUSE, OracleLinux, OSGeo live, OviOS, Parabola CLI, Parabola LXDE, Pardus, Parrot Home, Parrot Security, Parrot Studio, Parsix, PCLinuxOS, PeachOSI, Pentoo, Peppermint, PeppermintOS, Pinguy, PinguyOS, plopLinux, PointLinux, Pop!_OS, PORTEUS, Puppy Linux, PureOS, Q4OS, QubesOS, Quirky, Raspberry Pi Desktop, ReactOS, Redcore, Rescatux, RevengeOS, RoboLinux, Rockstor, ROSA FRESH, Runtu, Sabayon, SalentOS, Salix, ScientificLinux, Siduction, Slackware, Slax, SliTaz, Solus, SolydK, SolydX, SparkyLinux, Springdale, StressLinux, SubgraphOS, SwagArch, Tails, Tanglu, Tiny Core, Trisquel, TrueOS, TurnKey Linux, Ubuntu, Ubuntu Budgie, Ubuntu Studio, UbuntuKylin, Uruk, VectorLinux, VineLinux, VoidLinux, Voyager, VyOS, WattOs, Xubuntu, Zentyal, Zenwalk, Zevenet, Zorin OS

[50] 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 el navegador.

[51] Estos servicios son conocidos como computación en la nube (Cloud Computing).

Terminales de Linux en la Web

- https://www.tutorialspoint.com/execute_bash_online.php
- http://www.webminal.org/
- https://bellard.org/jslinux/
- https://codeanywhere.com/
- https://copy.sh/v86/
- https://www.masswerk.at/jsuix/
- https://linuxcontainers.org/lxd/try-it/
- http://cb.vu/

Editores BAHS en la Web

- https://www.shellcheck.net/
- https://www.learnshell.org/
- https://www.tutorialspoint.com/execute_bash_online.php
- https://paiza.io/en/projects/new?language=bash
- https://www.jdoodle.com/test-bash-shell-script-online
- http://rextester.com/l/bash_online_compiler

Usar Linux en Dispositivos Android En los dispositivos Android es posible usar un simulador de la línea de comandos del Shell usado en Linux, de forma que podremos introducir todos los comandos habituales para trabajar desde ahí en la comunidad de nuestra terminal Android. Uno de los paquetes más completo es:

https://termux.com

El paquete cuenta con una página *Wiki* en:

https://wiki.termux.com

Usando este paquete, las aplicaciones instaladas disponen de varias mejoras respecto al clásico Android Terminal Emulator, como el hecho de tener acceso a una gran biblioteca de paquetes de Linux para instalar desde la terminal -usando el comando *apt*-, así como algunos atajos de teclado transformados en combinaciones con los botones físicos de volumen y apagado de la terminal. Igualmente, es compatible con todo tipo de teclados físicos externos. Siendo posible trabajar con lenguajes como *NodeJ, Rubi, Python, C* y paquetes como *Nano, Vi, SSH, Git, Subversion, zsh Shell*, etc.

Usar Linux en Formato Live 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[52], una de las listas más completas de versiones Live esta en:

https://livecdlist.com

En el caso de tener un archivo ISO de algún sistema operativo (por ejemplo ubuntu-11.10-desktop-i386.iso) y se quiere ejecutar su contenido desde una máquina virtual con QEMU/KVM sólo es necesario usar:

```
$ kvm -m 512 -cdrom ubuntu-11.10-desktop-i386.iso
```

en este ejemplo usamos en KVM la arquitectura por omisión y memoria de 512 MB (-m 512).

Knoppix es una versión Live ampliamente conocida y completa, esta se puede descargar de:

https://www.knopper.net/knoppix/

y usar mediante:

```
$ kvm -m 1024 -cdrom KNOPPIX_V8.2-2018-05-10-EN.iso
```

aquí se usa la arquitectura por omisión y memoria de 1024 MB.

[52] Una imagen ISO es un archivo informático donde se almacena una copia exacta de un sistema de archivos y de esta se puede generar una imagen para CDROM, DVD o USB.

Usar Máquinas Virtuales de Linux Existen diversos proyectos que permiten descargar decenas de máquinas virtuales listas para ser usadas, para los proyectos VirtualBox y VMWare (y por ende para KVM/QEMU), estas se pueden descargar de múltiples ligas, algunas de ellas son:

https://www.osboxes.org
https://virtualboxes.org/images/

Si desargamos y descomprimimos el archivo lubuntu1210.7z, esto dejará la imagen de VirtualBox de LUBUNTU cuyo nombre es lubuntu1210.vdi. Entonces esta imagen la usaremos directamente en KVM/QEMU, mediante:

```
$ kvm -m 2000 -hda lubuntu1210.vdi
```

Nota: esta imagen usa como usuario y clave de acceso: lubuntu/lubuntu

Distribuciones de Sistemas Operativos 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 que distribución usar o empezar a usar sin tanta incertidumbre.

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

3 Herramientas en Línea de Comandos

En esta sección mostraremos el uso de varios comandos útiles que se pueden usar desde la línea de comandos

3.1 Prompt de la Línea de Comandos

Si acostumbras trabajar en la línea de comandos, muy posiblemente uses el Shell Bash (Bourne Again Shell, derivado del Bourne Shell de Unix), con un simple echo de la variable $SHELL puedes determinarlo: echo $SHELL. Si es el caso, entonces tu Prompt[53] debe de verse parecido a este:

antonio@miMaquina:~$

Que indicaría al usuario (antonio) y el equipo en el que está (miMaquina), ~ indica HOME (en este caso /home/antonio) o directorio de inicio, esta parte cambia cada vez que se ingresa a otro directorio:

antonio@miMaquina:~$ cd /etc

antonio@miMaquina:etc$

Algo útil, pero porque mejor no personalizarlo a nuestro gusto, así que empecemos por partes.

Secuencias de escape para el Prompt El Prompt se establece a través de la variable de entorno PS1:

antonio@miMaquina;~$ echo $PS1

\u@\h:\W\$

[53] 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*

Se puede apreciar que se utilizan secuencias de escapes para ir construyendo el Prompt, cada secuencia se indica con '\' seguido de un comando como 'u' (user) o 'h' (Host), los demás caracteres como [,],@, espacio, etc. son opcionales y puedes elegirlos tu al acomodo que desees, las secuencias de escape son las siguientes:

- \a un carácter ASCII de ring
- \d la fecha actual en formato "dia_sem mes día", "dom nov 18"
- \e un carácter ASCII de escape
- \h el nombre del equipo hasta el primer ., ejemplo miMaquina de miMaquina.fciencias.unam.mx
- \H el nombre del equipo
- \n nueva línea
- \r retorno de carro, Enter
- \s el nombre del Shell
- \t el tiempo actual en formato de 24 horas HH:MM:SS
- \T el tiempo actual en formato de 12 horas HH:MM:SS
- \@ el tiempo actual en formato de 12 horas con am/pm
- \u el usuario actual
- \v la versión de Bash
- \V el número de release de batch, versión + parche
- \w el directorio de trabajo actual, Path
- \W el nombre del directorio actual
- \! el número en el historial del comando
- \# el número de comando de este comando
- \$ si el usuario es root (UID=0) se indica un '#' o usuario normal '$'

- \\ diagonal
- \[inicio de una secuencia de caracteres no imprimibles
- \] fin de la secuencia de caracteres no imprimibles

Conociendo lo anterior podemos ahora tener un nuevo Prompt:

antonio@miMaquina:~$ PS1='(\t)[\u-\W]\$> '

obteniendo:

(11:26:02)[antonio-etc]$>

Solo se indica el cambio a PS1, con PS1=' ', entre las comillas simples va la nueva secuencia que se desea, asi que personaliza el tuyo.

Añade color a tu Prompt El Shell está lleno comandos muy poco usados, uno de estos es tput, que permite cambiar las características o capacidades disponibles para la terminal, disponibles a través de la base de datos llamada terminfo. Entre las características (hay bastantes) que podemos modificar de una terminal están el color de fondo y de frente (Background y Foreground) del texto a través de las siguientes opciones:

- setaf [0-7] cambia el color de frente
- setab [0-7] cambia el color de fondo
- bold modo negritas
- dim modo de poco brillo
- sgr0 apaga las características o atributos que se hayan indicado previamente

En cuanto a los códigos de color son los siguientes

- 0 Negro
- 1 Rojo
- 2 Verde

- 3 Café
- 4 Azul
- 5 Morado
- 6 Cyan
- 7 Gris

Puedes probar en una terminal escribiendo lo siguiente: tput setaf 1 y el texto se cambiará a rojo y puedes añadir por ejemplo un fondo verde tput setb 2 y te dará un fondo verde para el texto. Así que digamos, en base al Prompt anterior, que se desea la hora en rojo y negritas, esto lo haría:

```
$ PS1='\[$(tput setaf 1)(\t)$(tput sgr0)\][\u-\W]\$> '
(12:06:43)[antonio-~]$>
```

mmmm, un poco complicado, veamos por partes:
\[inicio de secuencia de caracteres no imprimibles
$(tput setaf 1) cambia a color rojo el texto, $(comando) expande el resultado de un comando que se ejecuta
(\t) lo que se ve visible en pantalla (20:06:43)
$(tput sgr0) apagamos los atributos, si no lo hacemos todo quedará en rojo
\] termina la secuencia de caracteres no imprimibles
No es tan complicado una vez que entendemos lo que sucede. Y es posible agregar más características en un sola invocación de $(), $(tput bold; tput setaf 1).

Ahora bien, una vez cambiado tu Prompt, éste no permanecerá así, si cierras la sesión o la terminal y vuelves a ingresar, notarás que sigues con tu mismo y aburrido Prompt de siempre, el cambio a la variable PS1 hay que agregarlo a tu archivo de inicialización de tu sesión, generalmente '.bashrc' o '.bash_profile', para recargar el ambiente hacer:

```
$ source ~/.bashrc
```

Incluso podrías poner los colores en variables, para facilitar el uso de la definición de PS1:

```
# se añade lo siguiente a .bashrc
# colores del texto
rojo=$(tput setaf 1)
verde=$(tput setaf 2)
# colores de fondo
azulF=$(tput setab 4)
grisF=$(tput setab 7)
# sin color
sc=$(tput sgr0)
PS1='\[$rojo(\t)$sc\][\u-\W]\$> '
```

Aquí mostramos otra forma de códigos de color para Bash:

- 0;30 Negro
- 0;34 Azul
- 0;32 Verde
- 0;36 Cyan
- 0;31 Rojo
- 0;35 Púrpura
- 0;33 Café
- 0;37 Gris Claro
- 1;30 Gris Oscuro
- 1;34 Azul Claro
- 1;32 Verde Claro
- 1;36 Cyan Claro
- 1;31 Rojo Claro
- 1;35 Fucsia
- 1;33 Amarillo

- 1;37 Blanco

con ellos podemos personalizar los caracteres especiales de escape usando las siguientes secuencias de caracteres no imprimibles:

\[comienza un secuencia de caracteres no imprimibles

\] termina un secuencia de caracteres no imprimibles

Por ejemplo:

```
PS1='\[\e[0;31m\]\u\[\e[m\] \[\e[1;34m\]\w\[\e[m\] \[\e[0;31m\]\$\[\e[m\]\[\e[0;32m\] '
```

Este indicador tiene las características de que el nombre 'root' está en rojo, el directorio de trabajo en azul claro, un indicador # en rojo y la escritura de texto, verde

Estas son solo algunas secuencias de escape comunes para cambiar el formato del mensaje Bash. En mi caso prefiero tener un Prompt para mi y otro para el usuario root, esto se logra mediante:

```
if [ "$UID" = 0 ]; then
    PS1='\[\033[01;31m\]\u\[\033[01;32m\]@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\n# '
else
    PS1='\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\n$ '
fi
```

obteniendo para el usuario antonio, algo como:

```
antonio@miMaquina:~
$
```

y para el usuario root, algo como:

```
root@miMaquina:/home/antonio/
#
```

Hay algunas secuencias más disponibles, podemos verlas en la página del manual de Bash.

3.2 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[54]:

  PROMPT_COMMAND="history -a"

[54] 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 «;».

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
```

3.3 Grabar y Reproducir Contenido de la Terminal

El comando *history* es una gran utilidad de línea de comando que ayuda a los usuarios a almacenar los comandos utilizados anteriormente, aunque no almacena la salida de estos. En caso de requerirlo, podemos grabar lo que hagamos en la terminal de línea de comandos usando el comando *script* (este comando pertenece al paquete *util-linux*) que nos proporcionará una funcionalidad poderosa que nos ayuda a registrar todo lo que se visualiza en la terminal grabándolo en un archivo de registro *.log* en texto plano y otro archivo de tiempo que temporaliza los comandos que tecleamos y sus respectivas salidas.

Podemos reproducir lo que se grabó usando el comando *scriptreplay* que usa la información auxiliar de tiempo que temporaliza los comandos que tecleamos y sus respectivas salidas.

La sintaxis básica para iniciar a grabar es:

```
$ script –timing=archivo_tiempo.txt archivo_grabacion.log
```

esto nos permite grabar todo lo que hagamos en la terminal hasta que finalicemos la sesión usando:

```
$ exit
```

Si deseamos seguir grabando sobre un archivo ya existente usamos:

```
$ script -a –timing=archivo_tiempo.txt archivo_grabacion.log
```

Para reproducir lo grabado usamos:

```
$ scriptrelay –timing=archivo_tiempo.txt archivo_grabacion.log
```

También está disponible el comando *asciinema*, el cual permite al igual que *script* grabar la sesión, pero además se puede subir lo grabado al sitio Web *asciinema.org*. Para instalarlo usamos:

```
# apt install asciinema
```

Para iniciar la grabación usamos:

```
$ asciinema rec [nombre]
```

para reproducirlo, usamos:

```
$ asciinema play [nombre]
```

3.4 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.

3.5 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
```

3.6 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.

3.7 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 [^][55].

- ***** 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
  ```

[55] Véase también el uso de las secuencias (véase 3.7).

- **?** 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}
```

3.8 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.

3.9 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.

3.10 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[56].

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-

[56] 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 &
```

3.11 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)
```

4 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[57]. 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.

[57] 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.

4.1 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 *" ! "*[58], 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:

[58] Otras opciones son -not (!), -or (-o), y -and (-a) aunque este último está implícito cuando se enumeran dos requisitos.

```
$ 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
```

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[59] 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
```

[59]

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
```

4.2 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[60]:

$ 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[61]

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'

[60] Esto también se puede poner por separado, como:
$ tar -c -v -f empaquetado.tar *.txt

[61] 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[62] 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*.

[62] 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 4.8).

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)[63]:

```
$ 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
```

[63] 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[64] usando: unp archivoCompactado
- * Descompactar usando: dtrx archivoCompactado
- * Descompactar usando: unar archivoCompactado

[64] 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 Directo-
rio/
```

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*[65] en conjunción con *tar o dd*[66] 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*[67] 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:

[65] 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.

[66] 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.

[67] 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

4.3 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[68]. 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:

[68] 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[69]] Origen [Origen]... Destino
```

por ejemplo, para hacer la sincronización de la carpeta *~/Datos* a *~/Respaldo* usamos:

```
$ rsync ~/Datos/ ~/Respaldo/
```

[69] 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 '*' ~/miDi-
rectorio/ ~/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.

4.4 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 4.2)

- scp (véase 4.3)
- rsync (véase 4.3)
- 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[70]. 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:
```

[70] 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.

4.5 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[71], *shred* sobrescribe repetidamente el contenido con valores aleatorios, haciendo un uso intensivo de disco[72]. 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
```

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:

[71] 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.

[72] No es recomendado su uso en discos SSD por el desgaste acumulativo que provoca al mismo.

```
$ 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)*[73]. 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 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
```

[73] Existen múltiples proyectos -algunos multiplataforma- que permiten hacer lo mismo que *gocryptfs* como son: *encfs, ecrptfs, cryptomator, securefs y CryFS*, entre otros.

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[74], 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
```

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
```

[74] 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.

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[75] nombre, correo del usuario y contraseña[76], para después generar la clave.

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
```

[75] Según la versión de GnuPG puede solicitar otros datos como: el tipo de clave, longitud, cuando expira está, entre otros posibles datos de configuración.

[76] 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.

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

4.6 AntiVurus

Aunque es poco probable que un virus afecte a un sistema Linux, es posible que sea un vector de transmisión a través de los servicios de E-Mail o del servidor de archivos, por ejemplo. Además, estos mismos servicios pueden integrar un antivirus, lo que contribuye a aumentar la seguridad de la red local.

Tenemos varias opciones pero por múltiples razones preferimos *ClamAV* y de ser requerida su interfaz gráfica ClamTK, para su instalación mínima podemos usar:

```
# apt install clamav clamav-freshclam
```

para que *ClamAV* también pueda verificar archivos comprimidos, debemos instalar algunos paquetes para descomprimir archivos:

```
# apt install arc arj bzip2 cabextract lzop nomarch p7zip \
pax tnef unrar-free unzip
```

Si tenemos acceso a los repositorios "non-free", podemos instalar otros paquetes adicionales:

```
# apt install lha unrar
```

La actualización de la base de datos de firmas de virus es descargada de internet, después de la instalación y antes de usar el antivirus, debemos realizar la actualización de la base de datos de firmas de virus, usando:

```
# freshclam
```

y podemos revisar la ruta que deseemos:

```
# clamscan /home/antonio/Descargas/
```

La distribución Debian ofrece un paquete de archivos de prueba "infectados" con la firma de un virus falso. *ClamAV* debe ser capaz de identificar correctamente estos archivos en la prueba. Para instalarlo usamos:

```
# apt install clamav-testfiles
```

y podemos efectuar la prueba, mediante:

```
# clamscan /usr/share/clamav-testfiles/
```

Si así lo queremos, podemos eliminar el paquete de pruebas:

```
# apt remove clamav-testfiles
```

También, podemos hacer la instalación completa y usarlo como demonio:

```
# apt install clamav clamav-docs clamav-daemon clamav-
freshclam
# apt install arc arj bzip2 cabextract lzop nomarch p7zip \
pax tnef unrar-free unzip
# apt install lha unrar
```

La actualización de la base de datos de firmas de virus es descargada de internet por el demonio *clamav-freshclam* 24 veces al día. Esta frecuencia puede modificarse en el archivo: */etc/clamav/freshclam.conf*

y debemos reiniciar el servicio, para que tenga en cuenta las alteraciones de configuración:

```
# service clamav-freshclam restart
```

También se puede probar el demonio *clamdscan*:

```
# clamdscan /usr/share/clamav-testfiles/
```

ahora el antivirus está listo para ser usado manualmente y ser integrado en otros sistemas y servicios.

Para la detección de virus, puede utilizarse los comandos *clamscan* y *clamdscan*. Pero, la segunda forma *clamdscan* es mucho más veloz, porque al ser un demonio, ya está presente en la memoria. Al contrario, cuando se ejecuta el comando *clamscan*, el sistema primero lee el disco.

4.7 Programando en Bash

Un lenguaje de programación interpretado es aquel que no necesita ser compilado para ejecutarse, sino que se puede ejecutar directamente desde el código fuente usando un intérprete, que no es más que un programa que puede traducir el código a unas instrucciones comprensibles por la máquina. Esto aporta algunas ventajas:

- Multiplataforma: al no ser binario, se pueden ejecutar en diversas plataformas sin modificaciones, lo que es una clara ventaja si queremos que el código funcione en cualquier sistema.
- Portabilidad: si el intérprete está listo para una plataforma, entonces el Script o lenguaje interpretado funcionará en dicha plataforma.

Sin embargo, estos lenguajes interpretados también tienen sus desventajas:

- Una de ellas es el rendimiento, ya que necesitan del intérprete siempre ejecutándose en segundo plano para que funcione.
- La propia dependencia del intérprete.

Como ejemplo de lenguajes interpretados se pueden citar algunos como Bash, Java, C#, JavaScript, Visual Basic .NET y VBScript, Perl, Python, Lips, Ruby, PHP, ASP, etc.

Un Script Bash no es más que un código creado con un lenguaje de programación interpretado para realizar una tarea. Generalmente es un programa sencillo, con un suceso de comandos u órdenes que se van ejecutándose de forma secuencial.

#!/bin/bash, que se conoce en la jerga de Unix como Shebang. Aunque este es el más habitual, no siempre se necesita usar para que el Script funcione. En otros proyectos también tienen sus propios Shebangs, como puede ser #!/usr/bin/env python3, #!/bin/sh, etc.

El objetivo del Shebang es simplemente indicar la ruta completa del intérprete de órdenes, para que se pueda ser localizado sea donde sea donde se ejecute el Script. Además, como puedes comprobar, no solo se determina la ruta en él, también el intérprete, en estos casos a Bash, Python 3, y otros intérpretes con los que trabajar.

La mayoría de los *Shell Scripts*[77] (guiones de intérprete de órdenes) Bourne pueden ejecutarse por Bash sin ningún cambio, con la excepción de aquellos Scripts del intérprete de órdenes o consola Bourne que hacen referencia a variables especiales de Bourne o que utilizan una orden interna de Bourne. La sintaxis de órdenes de Bash incluye ideas tomadas desde los intérpretes Korn Shell (ksh) y C Shell (csh), como la edición de la línea de órdenes, el historial de órdenes, la pila de directorios, las variables *$RANDOM* y *$PPID*, y la sintaxis de substitución de órdenes *POSIX*: $(...). Cuando se utiliza como un intérprete de órdenes interactivo, Bash proporciona autocompletado de nombres de programas, nombres de archivos, nombres de variables, etc., cuando el usuario pulsa la tecla *TAB*.

Redirecciones de entrada/salida la sintaxis de Bash permite diferentes formas de redirección de entrada/salida de las que la Shell Bourne tradicional carece. Bash puede redirigir la salida estándar y los flujos de error estándar a la vez utilizando la sintaxis:

```
orden >& archivo
```

que es más simple que teclear la orden Bourne equivalente: "orden > archivo 2>&1". Desde la versión 2.05b, Bash puede redirigir la entrada estándar desde una cadena utilizando la siguiente sintaxis (denominada "Here Strings"):

```
orden <<< "cadena a leer como entrada estándar"
```

si la cadena contiene espacios en blanco, deben utilizarse comillas.

[77] Para generar un archivo de BASH o Script, usemos cualquier editor de texto, por ejemplo *nano miScript*. En el, la primera línea se acostumbra poner el Shebang que se utiliza para indicar al sistema qué intérprete de comandos se debe llamar:
#!/bin/bash
ya creado el Script, es necesario hacerlo ejecutable, para ello usamos:

```
$ chmod u+x miScript
```

o

```
$ chmod 755 miScript
```

y lo ejecutamos en la línea de comandos mediante:

```
$ ./miScript
```

Argumentos Pondremos pasar múltiples argumentos al Shell Script desde la línea de comandos, ejemplo de ellos son:

- $0 El nombre del Script Bash.
- $1, $2 ... $n Los argumentos del Script Bash.
- !$ El último argumento.
- $$ La identificación del proceso del Shell actual.
- $# El número total de argumentos pasados al Script.
- $@ El valor de todos los argumentos pasados al Script.
- $? El estado de salida del último comando ejecutado.
- $! La identificación del proceso del último comando ejecutado.

Ejemplo, si llamamos a este archivo como *mi-Script*, con este contenido:

```
#!/bin/bash
echo "Nombre del Script: $0"
echo "Identificador del Script: $$"
echo "Numero total de argumentos: $#"
echo "Valor de todos los argumentos: $@"
```

lo hacemos ejecutable, mediante:

```
$ chmod u+x mi-Script
```

entonces podemos ejecutarlo usando:

```
$ ./mi-Script texto.txt:
```

También podemos usarlo para simplificar el trabajo en la consola, por ejemplo:

```
$ mkdir -p dir1/dir2/dir3
$ cd !$
```

Uso de condicionales La estructura más fundamental en cualquier estructura de toma de decisiones es una condición *if*. La sintaxis general de una instrucción *if* básica es la siguiente:

```
if [condicion]; then
    código
else
    código
fi
```

y podemos usar las siguientes condiciones:

Condición	Equivalencia
$a -lt $b	$a < $b
$a -gt $b	$a > $b
$a -le $b	$a <= $b
$a -ge $b	$a >= $b
$a -eq $b	$a es igual a $b
$a = $b	$a es igual a $b
$a -ne $b	$a no es igual a $b
|| Operador lógico OR	
&& Operador lógico AND	
-e $FILE	$FILE existe
-d $FILE	$FILE existe y es un directorio
-f $FILE	$FILE existe y es un archivo regular
-L $FILE	$FILE existe y es una liga suave
$STRING1 = $STRING2	$STRING1 es igual a $STRING2
$STRING1 != $STRING2	$STRING1 no es igual a $STRING2
-z $STRING1	$STRING1 es vacía
$a =~ .*subcad*.	$a contiene a las subcadena buscada

por ejemplo:

```
#!/bin/bash
if [ $(whoami) = 'root']; then
    echo "Eres root"
else
    echo "No eres root"
fi
```

o en una sola línea:

```
if [ $(whoami) = 'root' ]; then echo "Eres root"; else echo "No
eres root"; fi
```

Podemos utilizar una declaración *elif* (*else-if*) siempre que se desee probar más de una expresión al mismo tiempo, por ejemplo:

```
#!/bin/bash
if [ $1 -lt 13 ]; then
        echo "Eres un niño"
elif [ $1 -lt 18 ]; then
        echo "Eres un adolecente"
elif [ $1 -lt 65 ]; then
        echo "Eres un adulto"
else
        echo "Eres un adulto mayor"
fi
```

Podemos utilizar una instrucción *if* dentro de otra instrucción *if*. Por ejemplo:

```
#!/bin/bash
if [ $1 -gt 5 ]; then
        if [ $1 -lt 15 ]; then
                echo "Esta fresco"
        elif [ $1 -lt 25 ]; then
                echo "Esta rico el clima"
        else
                echo "Hace mucho calor"
        fi
else
        echo "Esta muy frio ..."
fi
```

otro ejemplo:

```
#!/bin/bash
if [ $# -ne 1 ]; then
      echo "Error: Invalido el numero de argumentos"
      exit 1
fi
file=$1
if [ -f $file ]; then
      echo "$file es un archivo regular"
elif [ -L $file ]; then
      echo "$file es una liga suave"
elif [ -d $file ]; then
      echo "$file es un directorio"
else
```

También podemos usar declaraciones de casos para remplazar varias declaraciones *if*, ya que a veces pueden llegar a ser confusas y difíciles de leer. La sintaxis general de una construcción de casos es la siguiente:

```
#!/bin/bash
CHAR=$1
case $CHAR in
[a-z])
      echo "Letras minusculas" ;;
[A-Z])
      echo "Letras mayusculas" ;;
[0-9])
      echo "Numeros" ;;
*)
      echo "Caracteres especiale"
esac
echo "$file el archivo no existe"
fi
```

Ciclos con for podemos usar ciclos al estilo del lenguaje de programación *C* usando la sintaxis:

```
for ((inicialización; condición; incremento)); do
      comandos
done
```

ejemplo:

```
for ((i=0; i < 10; i++)); do
    echo $i
done
```

O usando una lista o rango, ejemplo

```
for i in {1..10}; do
    echo $i
done
```

o

```
for i in /var/*; do
    echo $i
done
```

otro ejemplo:

```
primos=(2 3 5 7 11 13 17 19 23 29)
for i in "${primos[@]}"; do
    echo $i
done
```

Podemos usar *break* en un ciclo *for* mediante:

```
for ((i=0; i < 10; i++)); do
    echo $i
    if [ $i -eq 3 ]; then
        break
    fi
```

o podemos usar *continue* en un ciclo *for* mediante:

```
for ((i=0; i <= 10; i++)); do
    if [ $(($i % 2)) -ne 1 ]; then
        continue
    fi
    echo $i
done
```

o un ciclo infinito mediante:

```
for ((;;)); do
    comandos
done
```

Ciclos con while la sintaxis general del comando *while* es:

```
while [ condicion ]; do
    comandos
done
```

ejemplo

```
num=1
while [ $num -le 10]; do
    echo $(($num * 3))
    num=$(($num+1))
done
```

o un ciclo infinito mediante:

```
while [ true ]; do
    comandos
done
```

Ciclos con until la sintaxis general del comando *until* es:

```
until [ condicion ]; do
    comandos
done
```

ejemplo:

```
num=1
until [ $num -gt 10]; do
    echo $(($num * 3))
    num=$(($num+1))
done
```

Funciones como en casi todo lenguaje de programación, puede utilizar funciones para agrupar trozos de código de una manera más lógica, o practicar el divino arte de la recursión.

Declarar una función es sólo cuestión de escribir function mi_func { mi_código }.

Llamar a la función es como llamar a otro programa, sólo hay que escribir su nombre.

```
#!/bin/bash
function salir {
    exit
}
function hola {
    echo ¡Hola!
}
hola
salir
echo algo
```

Ejemplo de funciones con parámetros

```
#!/bin/bash
function salir {
   exit
}
function e {
    echo $1
}
e Hola
e Mundo
salir
echo algo
```

Este Script es casi idéntico al anterior. La diferencia principal es la función 'e'. Esta función imprime el primer argumento que recibe. Los argumentos, dentro de las funciones, son tratados de la misma manera que los argumentos proporcionados al Script.

La sintaxis de Bash tiene muchas extensiones que no proporciona el intérprete Bourne. Varias de las extensiones mencionadas se enumeran a continuación:

- Los guiones o Scripts de Bash reciben los argumentos que se le pasa al Shell como *$1, $2, ..., $n*. Se puede obtener el número total de argumentos con el símbolo: *$#*, otras opciones son:*$0* el nombre del Script, *$$* la identificación de ejecución del Script, *$@* el valor de todos los argumentos pasados al Script, *$?* el estado de salida del último comando ejecutado, *$!* la identificación del proceso del último comando ejecutado.

- Usando $# es posible comprobar el número de argumentos entregados al guión antes de realizar alguna acción con ellos:

```
#!/bin/bash
if [ $# -lt 2 ]; then
   echo "Necesitas pasar dos argumentos."
   exit 1
fi
```

- Otra forma de acceder a los argumentos es a través del array: *$@*, por medio del cual se puede iterar sobre todos los argumentos dados:

```
#!/bin/bash
for arg in "$@"
do
   echo "$arg"
done
```

- Una gran limitación del intérprete Bourne es que no puede realizar cálculos con enteros sin lanzar un proceso externo. En cambio, un proceso Bash puede realizar cálculos con enteros[78] utilizando la orden ((...)) y la sintaxis de variables $[...] de la siguiente manera:

  asigna el valor entero 55 a la variable VAR.

[78] Reconoce suma (+), resta (-), multiplicación (*), división entera (/), residuo de la división (%), exponenciación (**).

VAR=55

suma uno a la variable VAR. Observe la ausencia del carácter '$':

((VAR = VAR + 1))

otra forma de sumar uno a VAR. Preincremento estilo C:

((++VAR))

otra forma de sumar uno a VAR. Postincremento estilo C:

((VAR++))

multiplica la variable VAR por 22 y sustituye la orden por el resultado:

echo $[VAR * 22]

otra forma de realizar lo mismo:

echo $((VAR * 22))

- Podemos hacer cálculos de punto flotante instalando el comando *bc*, mediante:

  # apt install bc

  y lo podemos usar de esta forma:

  ```
  F=$(echo "3/5" | bc -l)
  echo "Resultado $F"
  ```

- La orden: ((...)) también se puede utilizar en sentencias condicionales, ya que su código de retorno es 0 ó 1 dependiendo de si la condición es cierta o falsa:

  ```
  #!/bin/bash
  if ((VAR == Y * 3 + X * 2))
  then
      echo Si
  fi
  ((Z > 23)) && echo Si
  ```

- La orden ((...)) soporta los siguientes operadores relacionales: '==', '!=', '>', '<', '>=', y '<='.
- Manipulación de cadenas, para definir una cadena usamos:

```
var="Valor"
echo $var
```

podemos concatenar dos cadenas, mediante:

```
var1="Prueba"
var2=" de Cadenas"
var3=$var1$var2
```

extraer una porción de la cadena:

```
var="Esto es una prueba"
echo ${var:1:6}
```

extraer desde una posición hasta el final de la cadena

```
var="Esto es una prueba"
echo ${var:5}
```

remplazar una subcadena de una cadena[79]:

```
var="Linux es un gran sistema operativo"
echo ${var/Linux/Debian}
```

borrar una subcadena de una cadena:

```
var="9938-333-334-334"
echo ${var/-}
```

borrar todas las apariciones de la subcadena:

[79] Para ver por separado cada uno de los directorios que componen el PATH podemos usar:
$ echo "${PATH//:/$'\n'}"

```
var="9938-333-334-334"
echo ${var//-}
```

también podemos utilizar [] para especificar rangos de caracteres:

```
var="esta es una prueba"
echo ${var//[a-z]/*}
```

convertir a mayúsculas una cadena:

```
var="esta es una prueba"
echo ${var^^}
```

si sólo se necesita convertir a mayúscula la primera letra usamos:

```
echo ${var^}
```

convertir a minúsculas una cadena:

```
var="Esta es una Prueba"
echo ${var,,}
```

si sólo se necesita convertir a minúscula la primera letra usamos:

```
echo ${var,}
```

- Manejo de ciclos con el contenido arrojado por algún comando, ejemplo:

```
#!/bin/bash
for i in $( ls *.cpp); do
    echo archivo: $i
done
```

- Manejo de secuencias usando: *seq* Primero Incremento Ultimo, ejemplos:

```
$ seq 7
$ seq 3 5
$ seq 0.1 0.1 1
$ seq 10 -4 2
$ seq -w 100
$ seq -f 'Mayo %g, 2020' 5
$ seq -s"  " 5
```

- Manejo de ciclos con una secuencia numérica, ejemplo:

```
#!/bin/bash
for i in ‘seq 1 10‘;
do
   echo $i
done
```

- Manejo de ciclos con *while*, ejemplo:

```
#!/bin/bash
COUNTER=0
while [ $COUNTER -lt 10 ]; do
   echo The counter is $COUNTER
   let COUNTER=COUNTER+1
done
```

- Manejo de ciclos con *until*, ejemplo:

```
#!/bin/bash
COUNTER=20
until [ $COUNTER -lt 10 ]; do
   echo COUNTER $COUNTER
   let COUNTER-=1
done
```

- Códigos de salida, cuando un programa Bash concluye emite un código que puede ser visualizado y atrapado, estos códigos son:

  0, ejecución exitosa
  1, error general no definido
  2, mal uso de caracteres
  126, comando no encontrado
  128, argumento no valido
  128+nm error fatal
  130, terminado por Ctrl-c
  255, código de salida fuera de rango

  Ejemplo de esto, probemos para código cero:

```
$ date ; echo $?
```

para código 127:

```
$ fallo; echo $?
```

- Solicitud de datos por teclado mediante el comando *read*, algunas de sus opciones son:

```
$ read variable ; echo $variable
$ read -p "Nombre del archivo a buscar" archivo ; echo $archivo
$ read -s -p "Escriba su clave de acceso" secreta ; echo $secreta
$ read -n 5 -p "Nombre de usuario" usuario ; echo $usuario
$ read -a -p "Introduzca los valores a buscar" arreglo; echo ${arreglo[@]}
```

- Si queremos mostrar mensajes de texto en color, podemos usar los siguientes códigos de escpe ANSI:

0;30	Negro
0;34	Azul
0;32	Verde
0;36	Cyan
0;31	Rojo
0;35	Púrpura
0;33	Café
0;37	Gris Claro
1;30	Gris Oscuro
1;34	Azul Claro
1;32	Verde Claro
1;36	Cyan Claro
1;31	Rojo Claro
1;35	Fucsia
1;33	Amarillo
1;37	Blanco

Al escribir uno de los códigos anteriores, activaremos el color correspondiente. Para desactivarlo, tan sólo tenemos que usar el código *0m*, por ejemplo:

```
VERDE='\033[0;32m'
NC='\033[0m'
echo -e "${VERDE}\nActualizando paquetes ...\n${NC}"
```

- Tenemos dos tipos de variables del Shell: Variables de medio ambiente (global), las del Shell y definidas por el usuario.

Para conocer las variables del medio ambiente podemos usar:

```
$ printenv
```

o

```
$ env
```

o conocer el valor de solo una, usamos:

```
$ printenv PATH
```
[80]

Para conocer la lista de todas las variables incluidas las definidas por el usuario, usamos:

```
$ set
```

Para asignar un valor a una variable usamos:

```
$ var=valor
```

o

```
$ export var=valor
```

para ver el valor de una variable podemos usar al comando *echo*, por ejemplo:

```
$ echo "$PS1"
```

[80] Para ver por separado cada uno de los directorios que componen el PATH podemos usar:
$ echo "${PATH//:/$'\n'}"

y si deseamos limpiar una variable usamos:

```
$ unset var
```

Podemos también definir variables de solo lectura, usando:

```
$ readonly var=valor
```

Observación 3 *Un proceso Bash no puede realizar cálculos en punto flotante sin lanzar un proceso externo (como bc). Las únicas Shell Unix capaces de esto son Korn Shell (versión de 1993) y zsh (a partir de la versión 4.0).*

Manejo de Errores El manejo de errores es importante en los Scripts Bash, sin él es posible que ni siquiera sepamos que algo ha fallado y encontrar la fuente del error en los Scripts problemáticos se vuelve mucho más difícil. A continuación, mostramos algunas formas de manejar los errores en los Scripts de Bash.

Si bien existe la opción *set -x* que permite habilitar el modo depuración, una mejor opción es usar *set -e* (set -o errexit) para salir de un Script cuando falla un comando, esto también establece el estado de salida del Script al del comando fallido. Si no usamos *set -e*, la secuencia de comandos continuará ejecutándose incluso cuando un comando haya fallado y el estado de salida de la secuencia de comandos será el del último comando en la secuencia de comandos. Utilice *set +e* para desactivar la funcionalidad *set -e* dentro de un Script. Ejemplo:

```
#!/bin/bash
set -e
echo "Script iniciado."
date
sleep 2
date
sleep 2
# El comando dateeeee no existe y debe fallar.
# set -e causa que el Script falle y salga con el código de
estado de dateeeee.
dateeeee
echo "Script finalizado exitosamente."
```

Además, podemos usar set -u (set -o nounset) para salir de un Script cuando se intente usar una variable que no ha sido inicializada y podemos solicitar que ciertas variables sean declaradas estáticas usando:

```
#!/bin/bash
set -u
readonly password_file="/etc/passwd"
```

Manejo de errores personalizado si deseamos hacer algo más que simplemente salir en caso de error, podemos verificar un comando en busca de fallas y luego manejarlo. No usemos $? para comprobar si hay fallos, no es necesario. Ejemplo:

```
#!/bin/bash
set -e
echo "Script iniciado."
date
sleep 2
date
sleep 2
# El comando dateeeee no existe y debe fallar.
if ! dateeeee 2>/dev/null
then
    echo "Debe haber un problema, lo volvemos a intentar"
    date
fi
echo "Script finalizado exitosamente."
```

Manejar errores en Pipeline si usamos Pipelines en nuestros Scripts, usemos *set -e* en combinación con *set -o* pipefail para salir de un Script si falla algún comando en un Pipeline; de lo contrario, el estado de salida de un Pipeline será el último comando en el Pipeline y cualquier comando que falló anteriormente en la canalización no se detectará. Ejemplo:

```
#!/bin/bash
set -eo pipefail
echo "Script started."
# El comando dateeeee no existe y debe fallar.
```

```
    # set -eo pipefail causa que el Script salga con el código de
  salida de dateeeee.
    dateeeee | echo "todo bien"
    echo "Script finalizado exitosamente."
```

Compilar un Script algunas veces es necesario compilar un Script (el resultado dejará al Script ejecutable pero no podrás ver su código fuente), para ello instalamos la herramienta *shc*, mediante:

```
# apt install shc
```

El comando realiza un proceso en dos pasos. Primero genera un código en C a partir del Shell Script, que luego compila para crear un programa binario. Si tenemos un Script HolaMundo.sh, entonces el código en C será HolaMundo.sh.c y el ejecutable HolaMundo.sh.x, para ello usamos:

```
$ shc -v -f HolaMundo.sh
```

y lo ejecutamos usando:

```
$ ./HolaMundo.sh.x
```

Podemos indicarle al ejecutable que tenga fecha de expiración *-e* (y *-m* un mensaje sobre la expiración) o crear un binario redistribuible *-r* para que se ejecute en diversas distribuciones de Linux, entre otros parámetros.

¿Qué es un Código de Estado de Salida? Un código de salida o estado de salida nos informa sobre el estado del último comando ejecutado. Si el comando se completó correctamente o finalizó con un error. Esto se obtiene después de que finaliza el comando.

La ideología básica es que los programas devuelven el código de salida 0 para indicar que se ejecutó exitosamente y sin problemas. El código 1 o cualquier otro distinto de 0 se considera fallido. Hay muchos más códigos de salida además del 0 y el 1, que veremos en esta sección.

Echemos un vistazo rápido a los códigos de salida destacados en el Shell de Linux:

Código de Salida y Significado del código

0 Comando ejecutado sin errores

1 Código para errores genéricos.

2 Uso incorrecto de comandos (o argumentos)

126 Permiso denegado (o) incapaz de ejecutar

127 Comando no encontrado o error de RUTA

$128 + n$ El comando finalizó externamente al pasar señales o encontró un error fatal

130 Terminación mediante Ctrl+C o SIGINT (código de terminación 2 o interrupción del teclado)

143 Terminación por SIGTERM (terminación por defecto)

255/$*$ El código de salida superó el rango 0-255, por lo que se cerró

Recuperando el Código de Salida el código de salida del comando ejecutado previamente se almacena en la variable especial $?. Puede recuperar el estado de salida ejecutando:

```
$ echo $?
```

esto se utilizará en todas nuestras demostraciones para recuperar el código de salida.

Código de Salida 0 el código de salida 0 significa que el comando se ejecuta sin errores. Idealmente, este es el mejor caso para completar comandos. Por ejemplo, ejecutemos un comando básico como este:

```
$ neofetch
$ echo $?
```

este código de salida 0 significa que el comando en particular se ejecutó exitosamente, ni más ni menos. Puedes intentar matar un proceso; también devolverá el código 0.

```
$ pkill lxappearance
```

matar una aplicación (mismo Shell) da como resultado el código 0. Ver el contenido de un archivo también devolverá un código de salida 0, lo que implica sólo que el comando 'cat' se ejecutó correctamente.

Código de Salida 1 el código de salida 1 también es común. Generalmente significa que el comando terminó con un error genérico. Por ejemplo, usar el administrador de paquetes sin permisos da como resultado el código 1, si intento esto:

$ su root

me dará un código de salida 1, lo que significa que el último comando resultó en un error. Si bien esto es un entendimiento general, también podemos interpretarlo como "operación no permitida". Operaciones como dividir por cero también dan como resultado el código 1.

Código de Salida 2 este código de salida se proporciona cuando el comando ejecutado tiene un error de sintaxis. El mal uso de los argumentos de los comandos también produce este error. Generalmente sugiere que el comando no se pudo ejecutar debido a un uso incorrecto. Por ejemplo, agregué dos guiones a una opción que se supone que tiene un guión:

$ grep –z archivo.txt

regresa un código 2. cuando se deniega el permiso, como acceder a la carpeta /root, aparece el código de error 2.

Código de Salida 126 126 es un código de salida peculiar ya que se utiliza para indicar que un comando o Script no se ejecutó debido a un error de permiso. Este error se puede encontrar cuando intenta ejecutar un Script de Shell sin otorgar permisos de ejecución.

Código de Salida 127 este es otro común. El código de salida 127 se refiere a "comando no encontrado". Suele ocurrir cuando hay un error tipográfico en el comando ejecutado o el ejecutable requerido no está en la variable $PATH.

Código de Salida Serie 128+n cuando se finaliza una aplicación o comando o su ejecución falla debido a un error fatal, se produce el código adyacente a 128 (128+n), donde n es el número de señal. Esto incluye todos los tipos de códigos de terminación, como SIGTERM, SIGKILL, etc., que se aplican al valor 'n' aquí.

Código 130 o SIGINT la señal SIGINT o Señal para interrupción del teclado se induce interrumpiendo el proceso mediante la señal de terminación 2 o mediante Ctrl+C. Como la señal de terminación es 2, obtenemos un código 130 (128+2).

Código 137 o SIGKILL la señal de terminación SIGKILL que finaliza el proceso instantáneamente tiene una señal de terminación 9. Este es el último método que se debe utilizar al finalizar una aplicación.

Código 143 o SIGTERM la señal SIGTERM o Señal para terminar es el comportamiento predeterminado cuando se finaliza un proceso sin especificar argumentos. El código de terminación para SIGTERM es 15, por lo tanto, esta señal obtiene un código de salida de 143

¿Qué pasa si el Código Supera 255? las versiones recientes de BASH conservan el valor del código de salida original incluso más allá de 255, pero generalmente, si el código excede 255, se cierra. Es decir, el código 256 se convierte en '0', 257 se convierte en '1', 383 se convierte en '127', y así sucesivamente. Para garantizar una mejor compatibilidad, mantenga los códigos de salida entre 0 y 255.

Observación 4 *Si está utilizando estos códigos en un Script de Shell, asegúrese de comprender el significado de cada código para facilitar la resolución de problemas.*

Algunos Consejos para Programar en Bash Un Bash Script es un archivo en el que se codifican múltiples comandos de Shell para realizar una tarea en particular, algunas sanas costumbres para escribir un Script, principalmente para mejorar la eficiencia y hacerlo más legible son:

Comentar el código definitivamente algo básico, pero que muchos olvidan y que es siempre de mucha utilidad, sea para uno mismo, o para otros que quieran revisar o modificar el Script, es sin duda determinante para lograr un código limpio y que logre explicar varias partes de códigos complejos. También es eficaz cuando se trabaja en grupo en un gran proyecto, ya que ayuda a comprender qué hace realmente la función o el método.

Uso de Funciones una función es un conjunto de comandos agrupados para realizar una tarea específica que ayuda a modular el flujo de trabajo, eliminando la repetición del código. Hace que el código sea limpio y más legible, así como fácil de mantener:

```
#!/bin/bash
function check_root() {
    echo "la función ha sido llamada";
}
```

Uso de comillas dobles ayudará a eliminar el englobamiento innecesario, así como la división de palabras, incluidos los espacios en blanco, cuando los valores de las variables contienen un carácter separador o un espacio en blanco.

Terminar ejecución por error a veces, al ejecutar un Script, puede haber algún error de ejecución. Incluso si un comando no se ejecuta, la secuencia de comandos puede continuar corriendo y afectar a los demás comandos del Script. Para evitar más errores lógicos, debemos incluir *set -o errexit* o *set -e* para finalizar el comando en caso de error:

```
#!/bin/bash
# Terminar el Script si hay error
set -o errexit
```

Declarando Variables según su tipo de datos y usos. Cuando la variable no es declarada, es posible que Bash no pueda ejecutar el comando relacionado. Las variables se pueden declarar global o localmente en el Script. Por ejemplo:

```
    #!/bin/bash
    # Declaración de variable, como notas es de formato vari-
able=valor
    declare -r -i x=30
    function my_variable(){
        local -r name = ${HOME}
    }
```

El uso de llaves vincula las variables con llaves mientras usa la concatenación de variables con cadenas para evitar usos innecesarios de variables. Esto también ayuda a identificar fácilmente la variable mientras se usa en una cadena. Por ejemplo:

```
#!/bin/bash
# Termina el Script si hay error
set -o errexit
# Variable personalizada
data= "${USER}_data is being used"
```

Sustitución del comando al asignar la salida del comando a la variable, Bash utiliza la función de sustitución de comandos. Necesitamos usar $() en lugar de comillas inversas para asignar la salida a las variables como se recomienda:

```
#!/bin/bash
# Termina el Script si hay error
set -o errexit
#Muestra la Fecha
date_now = ${date}
```

Nomenclatura de variables en nuestro sistema, todas las variables de entorno se nombran con letras mayúsculas. Entonces, cuando declaramos una variable local, debemos declararla usando letras minúsculas para evitar conflictos entre el entorno y el nombre de la variable local:

```
#!/bin/bash
# Termina el Script si hay error
set -o errexit
user_var = "$HOME es tu login correcto."
```

Declarar Variables Estáticas si tienes datos estáticos que permanecen sin cambios durante todo el Script, puedes asignar el valor a una variable estática cuyo valor no se puede modificar. Puedes declarar la variable estática usando el comando de solo lectura:

```
#!/bin/bash
# Probando configuración nginx
readonly test_conf_path = "/etc/nginx/conf.d/test.conf"
```

Depuración la depuración es la parte esencial para identificar un problema. Podemos verificar el error de sintaxis del Script, por lo que para verificarlo necesitamos ejecutar el Script Bash con *-n* usando el comando Bash:

```
$ bash -n script_name
```

además, podemos habilitar y ejecutar el Script en modo de depuración usando el siguiente comando.

```
$ bash -x script_name
```

4.8 Algunos Ejemplos en Bash

Para generar un archivo de Bash o Script, usemos cualquier editor de texto, por ejemplo:

```
$ nano miScript
```

en el, la primera línea se acostumbra poner:

```
#!/bin/bash
```

ya creado el Script, es necesario hacerlo ejecutable, para ello usamos:

```
$ chmod u+x miScript
```

o

```
$ chmod 755 miScript
```

y lo ejecutamos en la línea de comandos mediante:

```
$ ./miScript
```

si necesitamos conocer la línea que se esta ejecutando de un Script podemos usar:

```
$ bash -x ./miScript
```

Estos y otros ejemplos de Bash se pueden descargar de:
Bash

Conocer el uso de memoria de los procesos Podemos usar el comando *top* que nos informa en tiempo real el consumo de RAM o algunas de estas opciones:

```
$ ps aux –sort -rss | head
$ top -c -b -o +%MEM | head -n 20
```

Uso de comillas dobles y sencillas este pequeño ejemplo nos muestra visualmente el uso de comillas sencillas y dobles:

```
$ a=7; echo $a; echo "$a"; echo '$a'; echo "'$a'"; echo '"$a"'
```

Convertir imágenes usando xargs para convertir imagenes .png a .jpg, usaremos el comando *ls* para obtener los nombres de archivo que pasaremos a *xargs* que nos permite llamar a *conver* mediante *bash* para hacer la conversión, mediante:

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

y para convertir imagenes de .jpg a png, usamos:

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

Números pseudoaleatorios se pueden generar números pseudoaleatorios entre 0 y 32767, por ejemplo generamos números entre 0 y 100 mediante:

```
$ echo $((RANDOM % 100+1))
```

Longitud de una cadena disponemos de varias formas de conocer la longitud de una cadena, por ejemplo:

```
#!/bin/bash
cadena="Esta es una cadena"
tam1=${#cadena}
tam2=$(expr length "$cadena")
tam3=$(echo $cadena | awk '{print length}')
tam4=$(echo -n $cadena | wc -m)
echo $tam1 $tam2 $tam3 $tam4
```

Buscar en una cadena una subcadena podemos buscar en una cadena una subcadena y preguntar sobre ella, mediante:

```
#!/bin/bash
url="http://www.mmc.geofisica.unam.mx/acl/Textos/"
if [[ $url =~ .*acl*. ]]
then
    echo "Se encontro la cadena acl en ${url}."
else
    echo "No se encontro."
fi
```

Emoticonos en la consola podemos también generar emoticonos, muestra de ello es:

```
$ printf $(printf '\\U%x\\x20' {{128512..128591},{128640..128725}})
```

Formatear números es posible dar formato a números mediante el uso de *numfmt*, por ejemplo:

```
$ numfmt –grouping 123456789
```

acepta sufijos como 1K, 1Ki, 1M, 1Mi, 1G, 1Gi, por ejemplo:

```
$ numfmt –from=iec-i 2Gi
$ numfmt –to=iec 1000000
```

y lo podemos usar por ejemplo en un ls:

```
$ ls -l | numfmt –header –field 5 –to=iec
```

También podemos agregar ceros a la izquierda al visualizar una cantidad numérica, por ejemplo:

```
for((i=1; i<=10; ++i)): do
    printf "%03d\n" $i
done
```

en caso de no querer mostrar directamente el resultado, podemos asignarlo a una variable usando el parámetro *-v* del comando *printf*:

```
for((i=1; i<=10; ++i)): do
    printf -v j "%03d\n" $i
    echo $j
done
```

Realizar lista de potencias de 3 podemos usar el Bash para visualizar las primeras 19 potencias de 3, usando:

```
$ printf "%s\n" $[3**{1..19}] | nl
```

o

```
$ printf "%s\n" 3^{1..19} | bc | nl
```

o

```
$ echo -n 3^{1..19} | xargs -d' ' -I@ sh -c 'printf "%6s =
%48s\n" @ $(echo "@" | bc)'
```

o

```
$ for i in {1..19}; do printf "%0.2d %s\n" $i $[3**$i]; done
```

Manipular ruta de un archivo dada la ruta de un archivo:

```
archivo="/usr/share/icons/64x64/application-wireshark-doc.png"
```

para obtener sólo la ruta usamos:

```
echo "${archivo%/*}"
```

para obtener sólo el nombre de archivo usamos:

```
echo "${archivo##*/}"
```

para eliminar la extensión del nombre del archivo usamos:

```
nombre="${archivo##*/}"
echo ${nombre%.*}
```

para extraer la extensión del nombre del archivo usamos:

```
echo ${achivo##*.}
```

También podemos usar al comando basename, por ejemplo:

```
$ basename /etc/passwd
$ basename /etc/sysctl.conf .conf
```

de esta forma podemos renombar todos los archivos con extensión .jpeg a jpg, mediante:

```
for file in *.jpeg; do
    mv – "$file" "$(basename $file .jpeg).jpg"
done
```

en caso de necesitar la trayectoria pero no el nombre de archivo, podemos usar:

```
$ dirname /var/spool/mail/antonio.txt??
```

y podemos combinar a *basename* y *dirname*, por ejemplo:

```
Trayectoria="/home/antonio/data/fichero.txt"
tray=$(dirname "$Trayectoria")
nomb=$(basename "$Trayectoria")
echo $tray
echo $nomb
```

Manipular nombre de archivos Cambiar los espacios por subrayado:

```
$ for FILE in * ; do NEWFILE=‘echo $FILE | sed ’s/ /_/g’‘
; mv "$FILE" $NEWFILE ; done
```

Añadir un sufijo .txt a cada archivo:

```
$ for FILE in * ; do mv $FILE $FILE.txt ; done
```

Quitar el sufijo .txt a cada archivo:

```
$ for FILE in *.txt ; do NEWFILE=‘echo $FILE | sed ’s/.txt$//’‘
; mv $FILE $NEWFILE ; done
```

Convertir el sufijo .TXT a .txt

```
$ for FILE in *.TXT ; do NEWFILE=‘echo $FILE | sed
’s/.TXT$/.txt/’‘ ; mv $FILE $NEWFILE ; done
```

Añadir un prefijo maths_ a cada archivo:

```
$ for FILE in * ; do mv $FILE maths_$FILE ; done
```

Quitar el prefijo maths_ a cada archivo_

```
$ for FILE in * ; do NEWFILE=‘echo $FILE | sed ’s/^maths_//’‘
; mv $FILE $NEWFILE ; done
```

Convertir nombre de mayúsculas a minúsculas a cada archivo:

```
$ for FILE in * ; do mv $FILE ‘echo $FILE | tr ’[:upper:]’
’[:lower:]’‘ ; done
$ for FILE in * ; do mv $FILE ‘echo $FILE | tr ’[A-Z]’ ’[a-z]’‘
; done
```

Convertir nombre de minúsculas a mayúsculas a cada archivo:

```
$ for FILE in * ; do mv $FILE ‘echo $FILE | tr ’[:lower:]’
’[:upper:]’‘ ; done
$ for FILE in * ; do mv $FILE ‘echo $FILE | tr ’[a-z]’ ’[A-Z]’‘
; done
```

Quiero que los espacios cambien a un guión, salvo cuando haya dos o más espacios consecutivos, en el que necesito un solo subrayado. Mientras estamos en ello, si un espacio está al lado del punto de un prefijo, vamos a quitarlo. También quiero hacer los sufijos minúsculas.

```
$ for FILE in * ; do NEWFILE=‘echo $FILE | sed -e ’s/.TXT$/.txt/’
-e ’s/[ ]*[ ]/_/g’ -e ’s/_[.]/./g’‘ ; mv "$FILE" $NEWFILE ; done
```

Dar a todos los archivos el mismo nombre, pero con un número diferente:

```
$ NUM=0 ; for FILE in * ; do NUM=‘expr $NUM + 1‘ ; mv
$FILE Archivo\($NUM\) ; done
```

Mantener el nombre original y separar el número secuencial con un subrayado:

```
$ NUM=0 ; for FILE in * ; do NUM=‘expr $NUM + 1‘ ; mv
$FILE ${FILE}_$NUM ; done
```

Crear y remover archivos podemos usar al comando *seq* para crear y borrar un grupo de archivos:

```
$ touch $(seq -f "Archivo%g" 4)
$ rm $(seq -f "Archivo%g" 4)
```

otra forma es:

```
$ for i in web{0..10} db{0..2} otro_{a,b}{1..4}; do touch $i;
done
$ for i in web{0..10} db{0..2} otro_{a,b}{1..4}; do rm $i;
done
```

Manejo de parámetros Si generamos un Script con nombre *test.sh*, entonces podemos controlar por ejemplo dos parámetros (*–alpha* o *-a* y *–config* o *-c*) mediante:

```
#!/bin/bash
while [ True ]; do
if [ "$1" = "–alpha" -o "$1" = "-a" ]; then
      ALPHA=1
      shift 1
elif [ "$1" = "–config" -o "$1" = "-c" ]; then
      CONFIG=$2
      shift 2
else
      break
fi
done
echo $ALPHA
echo $CONFIG
ARG=( "${@}" )
for i in ${ARG[@]}; do
      echo $i
done
```

y lo podemos usar mediante:

```
$ ./test.sh –config my.conf foo bar
$ ./test.sh -a –config my.conf baz
```

Cree un árbol de carpetas usando el comando *mkdir* podemos crear un árbol de carpetas, con carpetas A - Z en el primer nivel, luego 0 a 100 en cada uno de ellos. Y todas las demás letras que comienzan en a en las carpetas pares y todas las demás que comienzan con b en impares

```
$ mkdir -p tree/{A..Z}/{0..100..2}/{a..z..2} tree/{A..Z}/{1..100..2}/{b..z..2}
```

Busca archivos ASCII y extrae direcciones IP de ellos podemos buscar en la trayectoria actual a todos los archivos ASCII que contienen direcciones IP validas y pedir que nos muestre estas, usando:

```
$ find . -type f -exec grep -Iq . {} \; -exec grep -oE "(25[0-5]|2[0-4]\
[0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25\
[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9]\
[0-9]?)" {} /dev/null \;
```

Crear directorios a partir del contenido de un archivo si tenemos un archivo con distintos nombres de directorios que necesitamos crear, por ejemplo:

```
servidores
comando/Linux
comando/Bash
editores/vin
editores/nano
```

y están en el archivo *directorios.txt*, podemos usar para crearlos:

```
$ xargs -I{} mkdir -p "{}" < directorios.txt
```

Crear archivos temporales cuando se trabaja en Bash se requiere construir archivos, muchos de ellos temporales, para ellos usamos:

```
$ tmp_file=$(mktemp)
```

si queremos ver el archivo creado podemos usar:

```
$ tmp_file=$(mktemp) ; echo $tmp_file
```

y para borrarlo podemos usar:

```
$ rm ${temp_file}
```

Otras opciones son:

```
$ tmp_file=/tmp/foobar.$$
$ tmp_file=/tmp/foobar.$RANDOM
$ tmp_file=‘date ’+%Y-%m-%d-%H-%M’‘-‘uuidgen -t | head
-c 5‘
```

Partir un archivo de texto en múltiples archivos el comando *split* nos permite partir un archivo de texto en múltiples archivos de un tamaño indicado, por ejemplo en 50 líneas:

```
$ split –lines=50 archivo.txt
```

Formateador de texto simple el comando *fmt* nos permite formatear la entrada que se le de, básicamente rellena y junta líneas de texto mientras que mantiene líneas en blanco y sangrías, por ejemplo reformateamos el texto del archivo para ajustarlo a una columna de 50 caracteres de ancho:

```
$ fmt -w 50 archivo.txt
```

Crear directorios temporales cuando se trabaja en Bash se requiere construir directorios donde contener los archivos de nuestro programa, muchos de ellos temporales, para ellos usamos:

```
$ tmp_dir=$(mktemp -d)
```

si queremos ver el directorio creado podemos usar:

```
$ tmp_dir=$(mktemp -d) ; echo $tmp_dir
```

y para borrarlo podemos usar:

```
$ rmdir ${temp_dir}
```

o

```
$ rm -R ${temp_dir}
```

Otras opciones son:

```
$ tmp_dir=$(mktemp -d -t ci-XXXXXXXXXX)
$ tmp_dir=$(mktemp -d -t ci-$(date +%Y-%m-%d-%H-%M-
%S)-XXXXXXXXXX)
$ tmp_dir=$(mktemp -d -t ci-XXXXXXXXXX –tmpdir=/home/antonio/tmp)
```

Leer un nombre de archivo dentro del Script se solicita al usuario un nombre de archivo mediante el comando *read*, para por ejemplo, contar el número de líneas del archivo:

```
#!/bin/bash
echo -n "Introduzca un nombre de archivo: "
read archivo
nlineas=$(wc -l < $archivo)
echo "Tiene $nlineas lineas el archivo $archivo"
```

Correr el Scrypt solo por el usuario root hay veces que debemos tener certeza de que solo el usuario root deba correr el contenido del Script, para ello podemos usar:

```
#!/bin/bash
if [[ "$(whoami)" != root ]]; then
    echo "Solo el usuario root puede correr este script."
    exit 1
fi
echo "Haciendo algo..."
exit 0
```

Pasar argumentos al Script podemos pasar argumentos al Script en la línea de comandos al momento de su ejecución, si llamamos a este archivo como *mi-Script*, con este contenido:

```
#!/bin/bash
echo "Nombre del Script: $0"
echo "Numero total de argumentos: $#"
echo "Valor de todos los argumentos: $@"
nlineas=$(wc -l < $1)
echo "Tiene $nlineas lineas el archivo $1"
```

entonces al ejecutar usando:

```
$ ./mi-Script texto.txt
```

nos permitirá contar el número de líneas del archivo *texto.tx*t

Agregar pausa a un Script podemos agregar a nuestro Script que haga una pausa antes de continuar su ejecución, algunas opciones son:

```
$ read -p "Presione [Enter] para continuar ..."
$ read -t 5 -p "Esperaremos 5 segundos para continuar"
$ sleep 5 && comando
```

Hacer cálculos de punto flotante podemos hacer conversión de grados Celsius a Fahrenheit, mediante en programa ConvierteC2F:

```
#!/bin/bash
# bc se le indica que use dos decimales: scale=2
F=$(echo "scale=2; $1 * (9/5) + 32" | bc -l)
echo "$1 grados Celsius es igual a $F grados Fahrenheit."
```

y para usarlo:

```
$ ./ConvierteC2F 25
```

Leer un archivo línea por línea en muchos ejemplos de Bash es necesario leer un archivo línea por línea, aquí mostramos algunas opciones:

```
$ while read line; do echo -e "$line\n"; done < archivo.txt
$ cat emails.txt |while read line; do echo "$line"; done
```

Leer un archivo en formato .CSV Hay muchas aplicaciones que generan archivos de texto con valores separados por comas (*.CSV*), los cuales son fácilmente leíble en BASH, por ejemplo un archivo *a.csv* con dos columnas, puede ser leído usando:

```
#!/bin/bash
while IFS=, read -r C1 C2
do
    echo "$C1 y C2"
done
```

si lo grabamos con nombre leeCSV.sh, entonces lo podemos ejecutar mediante:

```
$ leeCSV.sh < a.csv
```

Uso del comando column permite mostrar la salida de algún comando separado por columnas:

```
$ mount | column -t
```

Poner un texto al revés el comando *rev* permite que cualquier texto que se le pase lo ponga al revés, ejemplo:

```
$ rev <<< "esta es una cadena"
```

Encontrar los factores primos de un número el comando factor permite encontrar los factores primos de un número, ejemplo:

```
$ factor <<< 27832878
```

Conversión de base convierte el número hexadecimal 2F al decimal 47, usando:

```
$ echo $((0x2F))
```

o mediante:

```
$ echo $((16#2F))
```

Crear un archivo compactado por cada archivo indicado

```
$ for f in *.txt ; do tar cvf "$f.tar.bz2" "$f"; done
```

Crear un archivo tar por cada directorio hijo de nuestra actual posición:

```
$ find . -maxdepth 1 -mindepth 1 -type d -exec tar cvf \
{}.tar {} \;
```

Renombrar archivos si bien existe el comando *rename* para renombrar archivos **.bak* a **.txt*, usamos:

```
$ rename .bak .txt *.bak
```

también podemos hacer un Script para renombrar todos los archivos **.bak* a **.txt*, ejemplos:

```
$ for j in *.bak; do mv – "$j" "${j%.bak}.txt"; done
$ for j in *.bak; do mv -v – "$j" "${j%.bak}.txt"; done
```

Encontrar archivos y directorios grandes para encontrar, digamos los diez directorios más grandes en nuestra actual trayectoria, usamos:

```
$ du -h | sort -hr | head -n 10
```

para encontrar, digamos los diez archivos más grandes en nuestra actual trayectoria, usamos:

```
$ du -ah | sort -hr | head -n 10
```

Buscar un directorio y borrar su contenido un opción para buscar un directorio y borrar su contenido es:

```
$ find . -type d -iname nombre -delete
```

este comando encontrará todos los directorios y tratará de borrarlos, pero mandará error si el directorio no esta vació. Para solucionarlo podemos usar:

```
$ find . -type d -name "nombre" -exec rm -rf {} +
```

otras opciones son:

```
$ find . -type d -name "nombre" -exec rm -rf \;
$ find . -type d -name "nombre" -exec rm -rf "{}" \;
$ find . -type d -name "nombre*" -print0 | xargs -0 -I {}
/bin/rm -rf "{}"
```

podemos indicarle que realice la búsqueda en hasta 4 niveles de profundidad en el presente directorio:

```
$ find . -type d -name "nombre" -depth +4 -print0 -exec rm
-rf {} +
```

Algunas veces es necesario solo borrar los directorios vacíos, una forma de hacerlo es:

```
$ find . -type d -empty -print0 | xargs -0 -I {} /bin/rm -rf
"{}"
```

Conocer la velocidad de grabación del disco usando el comando *dd* y generando un archivo temporal, podemos conocer la velocidad de grabación de nuestro disco, mediante:

```
$ dd if=/dev/zero of=/tmp/salida.img bs=5G count=1 oflag=dsync;
rm -rf /tmp/salida.img
```

y para medir la latencia usamos:

```
$ dd if=/dev/zero of=/tmp/salida.img bs=512 count=1000
oflag=dsync; rm -rf /tmp/salida.img
```

Aprender algo nuevo nos da el manual de un comando diferente instalado en nuestro sistema en cada llamada:

```
$ man $(ls /bin | shuf | head -1)
```

Comandos usados y cuantas veces si deseamos conocer cuales son los comandos usados y cuantas veces los hemos usado, escribimos:

```
$ history | awk '{print $2}' | sort | uniq -c | sort -nr
```

Generando contraseñas si bien los programas GPG y OpenSSL nos permiten generar contraseñas, podemos usar en Bash para ello, en todos los ejemplos generamos contraseñas de 32 caracteres:

```
$ date +%s | sha256sum | base64 | head -c 32 ; echo
$ < /dev/urandom tr -dc _A-Z-a-z-0-9 | head -c${1:-32};echo;
$ tr -cd '[:alnum:]' < /dev/urandom | fold -w32 | head -n1
$ strings /dev/urandom | grep -o '[[:alnum:]]' | head -n 33 | tr
-d '\n'; echo
$ < /dev/urandom tr -dc _A-Z-a-z-0-9 | head -c32
$ dd if=/dev/urandom bs=1 count=32 2>/dev/null | base64
-w 0 | rev | cut -b 2- | rev
```

```
$ </dev/urandom tr -dc '12345!@#$%qwertQWERTasdfgAS-
DFGzxcvbZXCVB' | head -c32; echo ""
$ randpw(){ < /dev/urandom tr -dc _A-Z-a-z-0-9 | head -
c${1:-32};echo;} && randpw
```

Visualiza la IP del equipo imprime las direcciones de IPv4 asociada a las redes del equipo:

```
$ ip address | grep inet | grep -v inet6 | awk '{ print $2 }'
```

Lanzar procesos después de cierto tiempo algunas veces queremos que el sistema lance eventos después de cierto tiempo, supongamos queremos que nos avise que en 5 minutos -pueden ser segundos (s), minutos (m), horas (h) o días (d)- tenemos junta, usamos:

```
$ sleep 5m && echo "ir a la junta ... ahora"
```

pero podemos lanzar una aplicación, por ejemplo que toque música para avisarnos de la junta:

```
$ sleep 5m && vlc /home/usuario/alarma.mp3
```

Buscar archivos duplicados y cambiarlos por ligas simbólicas duras cuando se trabaja en proyectos es común terminar con múltiples copias de algunos archivos, pero podemos encontrar todos y remplazarlos por ligas simbólicas duras, para ello usaremos el programa hardlink, lo instalamos usando:

```
# apt install hardlink
```

podemos usar hardlink con *-tnv* para que nos diga los archivos que puede procesar:

```
$ hardlink -tnv temp
```

si nos satisface, ahora hacemos el cambio de los archivos duplicados por ligas simbólicas duras:

```
$ hardlink -tv temp
```

Para conocer el número de enlaces que tiene un determinado archivo, podemos usar:

```
$ ls -l archivo
```

o

```
$ stat archivo
```

Descargar archivos de la red una actividad común es la descarga de archivos de red (*.iso*, *.deb*, etc.) para ello hay varios programas como: *wget, uget, curl, lftp, woof, httpie*, entre otros, su uso básico es el siguiente:

```
$ curl http://132.248.182.159/acl/hcl/HerramientasComputacionalesEnLinux.pdf
```

descargará este texto. Se pueden descargar archivos y/o directorios de los protocolos *HTTP, HTTPS, FTP, POP3, SMTP* y *Telnet*. Además es posible interactuar con APIs usando los métodos: *DELETE, GET, POST* y *PUT*

Uso del comando awk Visualiza solo las líneas que son de 65 caracteres o más:

```
$ cat archivo.txt | awk "length > 65"
```

Conocer todas las ligas y a quien apuntan desde nuestra actul trayectoria, usamos:

```
$ find . -type l -print | xargs ls -ld | awk '{print $9 $10 $11}'
```

Nos mostrarán todos los usuarios que tiene el sistema, los cuales están dados de alta en el archivo del sistema /etc/passwd

```
$ awk -F':' '{ print $1 }' /etc/passwd
```

Nos indicará cuantos procesos tiene corriendo cada usuario:

```
$ ps -ef | awk '{print$1}' | sort | uniq -c | sort -nr
```

Nos muestra la primera y última línea de un archivo:

```
$ awk 'NR==1{print} END {print}' archivo.txt
```

o

```
$ awk 'NR==1;END {print}' archivo.txt
```

Programa *Buscar* para encontrar los archivos y directorios modificados en una determinada trayectoria [$1] en los últimos días [$2]:

```
#!/bin/bash
# Encuentra los archivos modificados en los ultimos $2 dias
if [ -z "$1" ]; then
    echo uso: $0 [directorio] [dias]
    exit
fi
find $1 -type f -mtime -$2 -exec ls -gGh –full-time '{}' \; | cut
-d ' ' -f 4,5,7
```

por ejemplo, para buscar desde /home los archivos modificados desde el último día, usamos:

```
$ ./Buscar /home 1
```

Programa *Diferencia* para encontrar la diferencia entre los archivos y directorios de dos trayectorias [$1] y [$2]:

```
#!/bin/bash
# Encuentra las diferencias entre dos directorios y subdirectorios
if [ -z "$1" ]; then
    echo uso: $0 [directorio1] [directorio2]
    exit
fi
diff <(cd $1 && find | sort) <(cd $2 && find | sort)
```

por ejemplo, buscar la diferencia de contenido entre /home/user/a1 y /home/user/b3, usamos:

```
$ ./Diferencia /home/user/a1 /home/user/b3
```

Ejemplo de uso de rsync ejemplo de uso de rsync en el cual el usuario podría indicar cual función ejecutar dentro del Script:

```
#!/bin/bash
sync_root(){
     spath="/srv/www/unam/https"
     echo "Running rsync..."
     rsync -ar $spath/* root@backup.unam.mx:$spath
}
case "$1" in
     *unam*) sync_root ;;
     *) echo "Error: El dominio no existe.";;
esac
```

Intentar sincronizar con srync hasta lograrlo al intentar sincronizar una carpeta usando *rsync*, puede que por cualquier razón no se logre completar la operación, por ello podemos usar el comando *until* para que siga intentando la sincronización por *rsync* a través de una conexión no confiable. Cuando *rsync* falla, se iniciará un reintento después de 60 segundos. Cuando se complete *rsync*, el ciclo terminará, para ello usamos:

```
$ until rsync -avy –delete-after /volume1/backup/ \
root@backup.unam.mx:.; do sleep 60; done
```

Programa Respalda para generar el respaldo de la trayectoria indicada [$2] con el nombre [$1] (véase 4.2):

```
#!/bin/bash
#Respalda el contenido dado $2 con el nombre $1
if [ -z "$1" ]; then
    echo uso: $0 [Archivo].tar [Archivo o Directorio]
    exit
fi
# Variables de trabajo
A=$1
B=$(date +%Y%m%d)
# Genera el archivo TAR
echo Generando el archivo TAR ...
shift 1
tar -zcvpf $A-$B.tar.gz $*
# Visualiza el nuevo contenido
/bin/ls -al –color=tty
```

por ejemplo, para respaldar el contenido de /home con el nombre respaldo, usamos:

```
$ ./Respalda respaldo /home
```

Programa para un menú este ejemplo muestra un menú del cual el usuario puede elegir entre las diferentes opciones, supongamos que lo llamamos menu y tiene el siguiente código:

```
#!/bin/bash
PS3='Elige tu comida favorita: '
comida=("Pizza" "Pho" "Tacos" "Salir")
select op in "${comida[@]}"; do
  case $op in
    "Pizza")
      echo "Se come en el mundo 1000 acres de $op cada dia"
      # opcional llamar a funcion o correr mas comandos
      break
      ;;
    "Pho")
      echo "$op es un tipo de sopa Vietnamita"
      # opcional llamar a funcion o correr mas comandos
      break
      ;;
    "Tacos")
      echo "Se comen al año 3 mil millones de $op cada año"
      # opcional llamar a funcion o correr mas comandos
      break
      ;;
    "Salir")
      echo "Usuario solicito salir"
      exit
      ;;
    *) echo "Invalida la opcion $REPLY";;
  esac
done
```

y lo ejecutamos usando:

```
$ ./menu
```

Jugar a viborita podemos usar Bash para jugar, usando las flechas podemos mover la barra dentro de toda la terminal usando el código:

```
$ reset;clear;x=$[COLUMNS/2];y=$LINES;u=0;v=-1;while :;do
read -sr -t0.02 -n3 d; case "${d:2:1}" in A) v=-1;u=0;;B)v=1;u=0;;C)
v=0;u=1;;D) v=0;u=-1;;esac;s=$x;t=$y;x=$[x+u];y=$[y+v];printf
"\033[%s;%sH\033[46m \033[0m\033[%s;%sH\033[44m \033[0m\033[0;0H"
$y $x $t $s;sleep .01;done
```

una versión más completa es:

```
$ clear;x=$(($COLUMNS/2)); y=$LINES;unset spacesused[*];declare
-a spacesused;xdir=0;ydir=-1;while true ; do read -s -r -t0.02 -
n3 direction ; case "${direction:2:1}" in A) ydir=-1;xdir=0 ;;
B)ydir=1;xdir=0 ;; C) ydir=0;xdir=1 ;; D) ydir=0;xdir=-1 ;;
esac ; space=$(( $COLUMNS * $y + $x )) ; if [[ "${space-
sused[$space]}" == "1" || $x -gt $COLUMNS || $x -lt 0 || $y -gt
$LINES || $y -lt 0 ]]; then printf '\033[%d;%dHBOOM!\nEND
OF LINE.\n' $y $x; break ; else spacesused[$space]=1 ; fi ;
ox=$x; oy=$y ; x=$(( $x + $xdir )) ; y=$(( $y + $ydir ))
; printf "\033[%s;%sH\033[46m \033[0m\033[%s;%sH\033[44m
\033[0m\033[0;0H" $y $x $oy $ox ; sleep 0.01 ; done
```

Generar disco USB a partir de un archivo ISO es posible generar un USB de cualquier imagen *ISO* descargada de la red, para ello primero debemos conocer el punto de montaje después de insertar la *USB*, mediante:

```
$ lsblk
```

o

```
# fdisk -l
```

o

```
dmseg | grep -i usb
```

si suponemos que el punto de montaje es:/dev/sdb, entonces procederemos a desmontarlo mediante:

```
$ umount /dev/sdb
```

ahora podemos usar el comando *dd* para generar el *USB* del archivo *ISO* descargado, mediante:

```
# dd if=/ruta-del-iso of=/dev/sdb bs=512M; sync
```

si queremos conocer el progreso de la generación del ISO, podemos usar:

```
# dd if=/ruta-del-iso of=/dev/sdb bs=512M status=progress;
sync
```

5 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[81], 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.

[81] 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.

5.1 Software Propietario

No existe consenso sobre el término a utilizar para referirse al opuesto del Software libre. La expresión «Software propietario (Proprietary Software)» (véase [16]), 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.

5.2 Software Libre

La definición de Software libre (véase [21], [22], [14], [15], [13] y [17]) 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 [21]), 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[82]; 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

[82] ¿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.

5.3 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.

5.4 Tipos de Licencias

Tanto la Open Source Initiative como la Free Software Foundation mantienen en sus páginas Web (véase [21], [22], y [17]) 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.

5.4.1 Licencias Creative Commons

Las Licencias[83] 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:

[83] 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[84], 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.

[84] 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.

5.4.2 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.

5.5 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-.

5.5.1 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.

5.5.2 ¿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.

5.5.3 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.

5.5.4 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.

5.5.5 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.

5.6 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.

5.6.1 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.

5.6.2 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.

6 Apéndice B:Seguridad, Privacidad y Vigilancia

Ante el constante aumento -explosivo- en el uso de dispositivos conectados a internet como computadoras personales, Laptops, tabletas y teléfonos celulares, expertos en ciberseguridad advierten un entorno propicio para que prosperen los cibercriminales y que, tanto individuos como empresas, se encuentren expuestos a múltiples amenazas de ciberseguridad.

En la actualidad, aproximadamente la mitad de la población mundial accede de algún modo a internet. Con tantos accesos concurrentes a la red de redes, la posible amenaza de seguridad a los sistemas informáticos crece y se complejiza, a pesar de las diversas y especializadas maneras de contrarrestarlas. Por su propia naturaleza, una conexión a internet hace que tu equipo de cómputo quede expuesto a ataques y pueda ser accesible por otro equipo, llegando a tener acceso a tu información.

¿Por qué?

- En muchas ocasiones, los usuarios no cuentan con la suficiente sensibilización sobre su exposición al riesgo, o bien, no están familiarizados con las herramientas y/o capacitación de sus organizaciones para prevenir y enfrentar amenazas de ciberseguridad.

- Los usuarios de internet a menudo utilizan redes Wi-Fi no seguras y usan dispositivos que no están configurados con los controles de políticas de seguridad básicos, lo cual los vuelve excepcionalmente vulnerables a ataques cibernéticos.

- Es común que los usuarios compartan dispositivos (computadoras, tabletas, teléfonos) para educación, trabajo y esparcimiento aumentando con ello su exposición a amenazas de ciberseguridad.

- Los piratas informáticos eligen como blanco la dependencia, cada vez mayor, de las personas con respecto a las herramientas digitales, además de que más tiempo en línea incrementa la potencial exposición de los usuarios a amenazas de ciberseguridad.

- Un error común es creer que los ciberatacantes utilizan únicamente herramientas muy avanzadas y técnicas para Hackear las computadoras

o cuentas de las personas. Los atacantes han aprendido que la forma más sencilla de robar la información de los usuarios, comprometer sus cuentas o infectar los sistemas es simplemente engañar al usuario para que lo hagan por ellos con una técnica denominada ingeniería social.

- Los piratas informáticos son extremadamente creativos al idear formas de aprovecharse de los usuarios y de la tecnología para acceder a contraseñas, redes y datos, a menudo sirviéndose de herramientas de ingeniería social y de temas y tendencias populares para tentar a los usuarios a tener comportamientos inseguros en línea.

Todo lo anterior, ha creado una enorme superficie de exposición a ataques cibernéticos dirigidos a los usuarios, la red, la computadora portátil, el teléfono inteligente, la tableta, etc. con la intención de cometer delitos informáticos. Por ello estos lineamientos generales de ciberseguridad son sugeridos para el uso seguro de redes y dispositivos de telecomunicaciones en apoyo al usuario, contiene recomendaciones sencillas y prácticas para fomentar una buena salud cibernética de los dispositivos utilizados para el trabajo a distancia, educación y diversión; y promover la ciberseguridad entre las personas y sus organizaciones.

La seguridad de la información en la red es más que un problema de protección de datos, y debe estar básicamente orientada en asegurar la información importante de las personas, de las organizaciones y la propiedad intelectual. Los riesgos de la información están presentes cuando confluyen fundamentalmente dos elementos: las amenazas de ataques, y las vulnerabilidades de la tecnología; conceptos íntimamente relacionados donde no es posible ninguna consecuencia sin la presencia conjunta de estos.

6.1 ¿Qué es la Privacidad y por qué es Importante?

Consideramos que la privacidad debería ser un derecho básico y una protección necesaria en la era digital para evitar la victimización y la manipulación. Una sociedad no puede tener libertad sin privacidad. Puede parecer un lujo, pero es importante para el bienestar de una sociedad libre y justa. Muchos gobiernos y empleadores hacen vigilancia[85] activa a sus ciudadanos o empleados para monitorear ideas, discusiones o disensiones no deseadas. Los

[85] ¿Qué es vigilancia? Vigilancia es el seguimiento de las comunicaciones, acciones o movimientos de una persona por un gobierno, empresa, grupo o persona.

infractores son luego procesados o reeducados para alinearse con lo que las autoridades consideran apropiado. Sin el beneficio del anonimato, el deseo de los ciudadanos de expresar sus pensamientos se reprime efectivamente.

En la era digital, la privacidad va más allá del anonimato, al proteger a las personas de la victimización y la manipulación. La sociedad ha adoptado la tecnología para educarse, comunicarse, realizar negocios y formar relaciones. Nuestros puntos de vista y opiniones están fuertemente influenciados por lo que aprendemos de las fuentes de noticias locales, nacionales e internacionales.

Contribuimos en gran medida al panorama digital a través de nuestras acciones y decisiones. Nuestras huellas digitales están en todas partes. Cuentan una historia de a dónde vamos, qué hacemos, quién nos gusta o no, y qué pensamos. Se crean con cada Clic que hacemos y con cada archivo, aplicación y dispositivo que usamos. Cuando esos datos se agregan, pueden proporcionar información tremendamente poderosa sobre una persona o comunidad, lo suficiente como para construir personas complejas y precisas.

Esta información se usa comúnmente para manipular las creencias y comportamientos de las personas. Las compras en línea son un ejemplo perfecto: el Marketing dirigido y la publicidad basada en datos es un gran negocio porque logra que la gente gaste dinero. Todo se reduce a saber lo que las personas hacen, piensan, dicen, consumen y miran[86]. Tener acceso a grandes

¿Cuándo es legal la vigilancia? En general, cuando es selectiva, se basa en indicios suficientes de conducta delictiva y está autorizada por una autoridad estrictamente independiente, como un juez.

¿Qué es la vigilancia masiva? La vigilancia masiva indiscriminada es el control de las comunicaciones por internet y telefónicas de un gran número de personas -a veces de países enteros- sin que existan indicios suficientes de conducta delictiva. Este tipo de vigilancia no es legal.

¿Qué datos recogen? Algunos gobiernos almacenan y analizan historiales de navegación, búsquedas en internet, mensajes de correo electrónico, mensajes instantáneos, conversaciones por Webcam y llamadas telefónicas. También reúnen metadatos -datos sobre datos-, como destinatarios de correo electrónico, horas de llamadas y registros de ubicación.

¿Qué hacen con mis datos? Se guardan en grandes centros de datos donde unos algoritmos informáticos pueden hacer búsquedas en ellos y analizarlos. También están a disposición de las autoridades de las agencias de seguridad a través de potentes bases de datos como XKeyscore.

[86]Has notado que si buscas algo en un buscador comercial -como Google, Yahoo, Bing, Ask, Baidu, etc.-, minutos después en tus distintas redes sociales aparecerán mensajes relacionados a tu búsqueda. Esto ocurre porque los buscadores comerciales en cada búsqueda que haces, vídeo o los anuncios que ves o en los que haces Clic comparten tu ubicación,

cantidades de datos privados les brinda a los anunciantes la capacidad de crear mensajes oportunos y significativos que atraigan a las personas a los comportamientos deseados.

Pero si los minoristas pueden hacer que las personas compren cosas que no necesitan, ¿para qué más se pueden usar los datos privados? ¿Qué tal cambiar lo que piensa la gente, a quiénes apoyan, sus opiniones políticas, qué debería convertirse en ley y en qué creer? El uso de información privada se ha aprovechado durante mucho tiempo para promover, vilipendiar o perseguir a diversas religiones, partidos políticos y líderes.

En las últimas décadas, ha cambiado la forma en que los ciudadanos del mundo reciben sus noticias. Los segmentos de noticias y entretenimiento han comenzado a mezclarse, a menudo informando hechos con adornos e historias obstinadas para influir en las opiniones públicas. Cuanta más información privada se conozca, más fácil será influir, convencer, engatusar o amenazar a las personas.

Según informes de Oxford Internet Institute, a pesar de los esfuerzos para combatir la propaganda computacional, el problema ha crecido a gran escala. El mayor crecimiento proviene de propaganda política que es difundida junto con desinformación y noticias basura alrededor de los períodos electorales. Se incrementa el número de campañas políticas a nivel mundial que usan Bots, noticias basura y desinformación para polarizar y manipular a los votantes.

Un velo de privacidad puede proteger tanto los beneficios como los abusos. La tendencia actual es establecer y ampliar los derechos de privacidad en beneficio de los ciudadanos. Esto reduce la victimización, manipulación y explotación digitales al proteger los datos confidenciales y permite actividades que promueven la libertad y la libertad de expresión.

Sin leyes, los gobiernos y las empresas han desarrollado prácticas que aprovechan el poder de recopilar información confidencial y utilizarla en su propio beneficio. Las nuevas leyes de privacidad están cambiando el panorama y muchas empresas éticas reducen sus esfuerzos de cobranza para ser más conservadoras y respetuosas. También muestran flexibilidad en la forma en que tratan, protegen y comparten dichos datos.

Algunos gobiernos y agencias también están reduciendo la recolección, limitando la retención o terminando los programas domésticos que los ciu-

los sitios que visitas, los dispositivos, navegadores y aplicaciones que usas para acceder a los servicios del buscador con fines comerciales.

Una opción para nuestra privacidad es usar buscadores que no dejan rastro de nuestras búsquedas como puede ser: DuckDuckGo.

dadanos consideran invasivos. Al mismo tiempo, los organismos encargados de hacer cumplir la ley quieren conservar la capacidad para detectar e investigar delitos, para proteger la seguridad de los ciudadanos.

También se hace un mal uso de la privacidad. Es la herramienta preferida por quienes cometen delitos y permite que los actos atroces contra otros no se detecten. Puede ocultar actos terribles y permitir una coordinación generalizada entre el fraude, abuso y terror.

Se argumenta que las puertas traseras digitales, las claves maestras y los algoritmos de cifrado que obtienen acceso a los sistemas y la información privada ayudarían en la detección legal de actividades delictivas y en las investigaciones para identificar a los terroristas. Aunque suena como una gran herramienta contra los delincuentes, es una caja de Pandora.

Las puertas traseras y las llaves maestras no limitan el acceso para una investigación específica donde existe una causa probable, sino que permiten una vigilancia generalizada[87] y la recolección de datos de toda una población, incluidos los ciudadanos respetuosos de la ley. Esto viola el derecho de las personas a la privacidad y abre la puerta a la manipulación y el enjuiciamiento político. La capacidad de leer cada texto, correo electrónico, mensaje y conversación en línea para "monitorear" a la población crea un camino claro hacia el abuso. El riesgo de control y explotación es real.

Incluso para aquellos que no tienen ninguna objeción a que su gobierno

[87] El 5 de junio de 2013, el exanalista de la CIA y de la NSA Edward Snowden decidió revelar la existencia de programas de vigilancia sobre las comunicaciones de millones de ciudadanos de todo el mundo. A través de The Guardian y The Washington Post, supimos que, en nombre de la seguridad y sin ningún control judicial, la NSA y el gobierno británico habían rastreado e-mails, llamadas telefónicas y mensajes encriptados. Empresas como Facebook, Google y Microsoft habían sido obligadas a entregar información de sus clientes por órdenes secretas de la NSA. Esta misma agencia grabó, almacenó y analizó los 'metadatos' de las llamadas y de los mensajes de texto enviados en México, Kenia y Filipinas. Incluso llegaron a espiar el móvil de la canciller alemana Angela Merkel.

Snowden hizo las filtraciones a la prensa desde Hong Kong y actualmente vive en Rusia, donde se le concedió asilo. No puede volver a su país porque está acusado de revelar información clasificada a personas no autorizadas y de robo de propiedad del gobierno federal. Para unos es un héroe y para otros un traidor, gracias a las revelaciones de Snowden la opinión pública es más consciente de su derecho a la privacidad y ha reaccionado oponiéndose al espionaje masivo.

Aunque queda un largo camino para asegurar que los Gobiernos no invadan la vida privada de las personas, los tribunales han declarado ilegales algunos aspectos de estos programas y las empresas tecnológicas han tenido que posicionarse ante un escándalo capaz de dañar seriamente su reputación.

tenga acceso, debemos considerar el hecho de que tales puertas traseras y claves maestras serían buscadas por ciberdelincuentes y otros actores del estado-nación. Ningún sistema es infalible. Con el tiempo, los delincuentes encontrarían y utilizarían estas herramientas en detrimento de la comunidad digital mundial. Algunas puertas traseras podrían valer decenas de miles de millones de dólares para el comprador adecuado, ya que podrían desbloquear un poder inimaginable para apoderarse de la riqueza, afectar a la gente, dañar naciones, socavar la independencia y reprimir el pensamiento libre.

Proteger la privacidad no se trata de ocultar información. Se trata de la capacidad de liberarse de influencias no deseadas, de la tiranía y de comunicarse con los demás de formas que desafíen el statu quo. La privacidad protege a las personas, pero también los cimientos de una sociedad libre. La privacidad no es un tema fácil y no existe una solución perfecta. Es una situación dinámica y seguirá cambiando con el sentimiento público.

Todos quieren cierto nivel de discreción, confidencialidad y espacio. Nadie quiere que se expongan sus contraseñas, finanzas familiares, detalles de relaciones personales, historial médico, ubicación, compras y discusiones privadas. Las personas tampoco disfrutan de verse inundadas de Spam, Phishing y llamadas de ventas incesantes. La privacidad no se trata necesariamente de ocultar algo, sino de limitar la información a quienes tienen derecho a saber.

Muy poca privacidad puede socavar la libertad de expresión, la libertad y la denuncia de victimizaciones. También empodera a entidades poderosas para manipular el mundo digital de las personas para coaccionarlas, manipularlas y victimizarlas. Demasiada privacidad puede permitir que los actores criminales prosperen y se escondan de las autoridades. Debe lograrse un equilibrio[88].

[88]Los gobiernos nos enfrentan a un dilema falso: seguridad o libertad. En un estado de derecho, donde las leyes equilibran ambos conceptos, las personas son inocentes hasta que se demuestra lo contrario y tienen derecho a que se respete su vida privada. Por tanto, antes de violar estos derechos, los gobiernos deben de tener indicios de que se está cometiendo un delito. No pueden buscar pruebas aleatoriamente en nuestras comunicaciones privadas antes de que se cometa ese delito.

Varios países dan "carta blanca" a la vigilancia indiscriminada en sus leyes. En Francia se permite interceptar masivamente comunicaciones, retener información durante largos períodos de tiempo y se ha eliminado la autorización judicial previa. También Reino Unido ha introducido en su legislación mayores poderes de espionaje. Polonia ha otorgado poderes de vigilancia incompatibles con respecto a la privacidad a la policía y o otras agencias. Otros países como China o Rusia también vigilan internet con total desprecio a

Algunos gobiernos y empresas tecnológicas quieren que aceptemos que no tenemos derechos cuando estamos en internet. Que cuando usamos el teléfono o entramos en nuestra cuenta de correo electrónico, todo lo que hacemos o decimos les pertenece. No permitiríamos este grado de intrusión en nuestra vida fuera de internet, así que no debemos permitirlo dentro.

Cuando los gobiernos no protegen los derechos humanos, sólo la acción de la gente común y corriente puede hacer que las cosas cambien y que los responsables de los abusos rindan cuentas. Muchas organizaciones trabajan para que los gobiernos prohíban la vigilancia masiva y el intercambio ilegal de información confidencial. Esta será una larga lucha no exenta de dificultades, pero cuyo impulso va creciendo con el tiempo.

¿Qué son los datos biométricos? en general entenderemos por datos biométricos a las características físicas, fisiológicas, morfológicas, de comportamiento y rasgos de personalidad distintivas de cada persona que permite distinguir ciertas singularidades e identificar al individuo en cuestión.

El uso de datos biométricos no es nuevo e incluso es algo en lo que constantemente se ven envueltos las redes sociales[89], aunque los fines de uso son diversos. Recientemente Texas demandó a Meta, dueña de Facebook, por almacenar millones de estos datos y comercializarla con terceros.

Las Amenazas a los Usuarios en un entorno dinámico de interconectividad, pueden venir de cualquier parte, sea interna o externa, e íntimamente relacionada con el entorno de las organizaciones. Las vulnerabilidades son una debilidad en la tecnología o en los procesos asociados con la información, y como tal, se consideran características propias de los sistemas o de la infraestructura que los soporta.

Las personas o usuarios de internet que emplean las tecnologías para vulnerar los sistemas en la red, robar información y/o infectarlos con comportamientos dudosos, son comúnmente conocidos como *Crackers*.

---

la intimidad de las personas.

[89] Por ejemplo en el caso de X antes Twitter, podrá hacerse de tu foto de perfil, así como otros datos que incluyen historial de empleo, educación, preferencias de empleo, capacidades y habilidades, así como búsqueda de trabajo, datos que utilizaría para ofrecer servicios similares a los de Linkedln, "recomendarte potenciales trabajos, compartir con potenciales empleadores cuando solicitas un trabajo, permitir que los empleadores encuentren potenciales candidatos y mostrarte publicidad más relevante".

El término Hacker[90] en el sentido más filosófico tiende a promover una conciencia colectiva de la libertad del conocimiento y la justicia social, por lo que muchas veces se los encuentra en situaciones de activismo (llamado en este caso Hacktivismo) en pos de dicha ideología. Su forma de actuar, por lo general, determina su clasificación en:

- *Hacker* de Sombrero Blanco (White Hat): éticos, expertos en seguridad informática, especializados en realizar test de intrusión y evaluaciones de seguridad.
- *Hacker* de Sombrero Negro (Black Hat): también conocidos como *Crackers*, vulneran los sistemas de información con fines maliciosos.
- *Hacker* de Sombrero Gris (Grey Hat): en ocasiones vulneran la ley, y de forma general no atacan malintencionadamente o con intereses personales, sino que sus motivaciones se relacionan a protestas o desafíos personales.
- *Hacker* de Sombrero Verde (Green Hat): son novatos que pueden evolucionar y buscan convertirse en expertos.
- *Hacker* de Sombrero Azul (Blue Hat): son personas expertas que se les paga para probar Software en busca de errores antes de que éste salga al mercado.

Para una entidad, la fuga de información provocada por el actuar de algunos de estos usuarios de la red, puede ocurrir deliberadamente como resultado de una acción intencional de algún empleado descontento, como consecuencia de un ciberataque, o inadvertidamente, por un colaborador desprevenido víctima de un Software malicioso.

Una de las principales amenazas para los dispositivos tecnológicos utilizados para el trabajo y estudio a distancia es el Malware, también conocido como Software o código malicioso. Éste se define como cualquier programa informático que se coloca de forma oculta en un dispositivo, con la intención de comprometer la confidencialidad, integridad o disponibilidad de los datos, las aplicaciones o el sistema operativo.

[90] Existen algunos otros términos en el ciberespacio, por ejemplo: Newbie, que significa principiante; Lammer, persona que presume tener conocimientos que realmente no posee; Phreaker, Hacker orientado a los sistemas telefónicos; Script Kiddie, quien utiliza programas creados por terceros sin conocer su funcionamiento.

Los tipos más comunes de amenazas de Malware incluyen Virus, gusanos, troyanos, Rootkits y Spyware. Las amenazas de Malware pueden infectar cualquier dispositivo por medio del correo electrónico, los sitios Web, las descargas y el uso compartido de archivos, el Software punto a punto y la mensajería instantánea.

Además, existen amenazas relacionadas con la ingeniería social como el Phishing, Smishing y Vishing, por medio de los cuales los atacantes intentan engañar a las personas para que revelen información confidencial o realicen ciertas acciones, como descargar y ejecutar archivos que parecen ser benignos, pero que en realidad son maliciosos.

6.2 Las Vulnerabilidades y Exposiciones Comunes

De manera global, la última década ha sido testigo del cambio de paradigma en que los atacantes buscan explotar vulnerabilidades dentro de las organizaciones y las infraestructuras de redes. Con el fin de contrarrestar estos ataques, las políticas de seguridad persiguen constantemente aprender de ellos para estar preparados lo mejor posible, y en este sentido, intentar garantizar confianza y tranquilidad a los usuarios de la red, sobre el empleo de sus datos, finanzas y propiedades intelectuales.

Los ataques de seguridad vienen en muchas formas y usan varios puntos de entrada. Cada tipo de ataque viene en varios tipos, ya que generalmente hay más de una forma en que se pueden configurar o camuflar en función de la experiencia, los recursos y la determinación del pirata informático.

Las Vulnerabilidades y Exposiciones Comunes (Common Vulnerabilities and Exposures, CVE[91]) que tienen los distintos sistemas operativos (productos), es una lista de información registrada sobre vulnerabilidades de seguridad conocidas, en la que cada referencia tiene un número de identificación llamada CVE-ID, descripción de la vulnerabilidad, que versiones del Software están afectadas, posible solución al fallo (si existe) o como confi-

[91] https://www.cvedetails.com/
https://www.cvedetails.com/top-50-products.php
https://thebestvpn.com/vulnerability-alerts/
Hay que notar que en la base de datos se diferencian distintos productos de Windows (como Windows 7.0 u 8.1) pero no se hace lo mismo con los demás productos (por ejemplo para Debian GNU/Linux están los errores desde 1999 a la fecha) con lo que es engañosa la comparación del número vulnerabilidades para un sistema operativo con tantas versiones generadas en dicho tiempo con otro con un tiempo de vida corto.

gurar para mitigar la vulnerabilidad y referencias a publicaciones o entradas de foros o Blogs donde se ha hecho pública la vulnerabilidad o se demuestra su explotación. Además suele también mostrarse un enlace directo a la información de la base de datos de vulnerabilidades (https://nvd.nist.gov, https://openssf.org y https://docs.aws.amazon.com/security), en la que pueden conseguirse más detalles de la vulnerabilidad y su valoración.

El mundo está cada vez más interconectado y, como resultado de esto, la exposición a las vulnerabilidades de seguridad también ha aumentado dramáticamente. Las complejidades de mantener las plataformas de cómputo actuales hacen que sea muy difícil para los desarrolladores cubrir cada punto de entrada potencial. En 2019 hubo un promedio de más de 45 vulnerabilidades y exposiciones comunes registradas por día y estas siguen en aumento año con año.

El CVE-ID ofrece una nomenclatura estándar para identificación de la vulnerabilidad de forma inequívoca que es usada en la mayoría de repositorios de vulnerabilidades.

Es definido y es mantenido por The MITRE Corporation (por eso a veces a la lista se la conoce por el nombre MITRE CVE List) con fondos de la National Cyber Security Division del gobierno de los Estados Unidos de América. Forma parte del llamado Security Content Automation Protocol.

Por otro lado, como parte de las amplias revelaciones sobre vigilancia masiva filtradas en 2013, 2014 y años posteriores, se descubrió que las agencias de inteligencia estadounidenses y británicas, la Agencia de Seguridad Nacional (NSA, por sus siglas en inglés) y el Cuartel General de Comunicaciones del Gobierno (GCHQ, por sus siglas en inglés), respectivamente, tienen acceso a los datos de los usuarios de dispositivos Android. Estas agencias son capaces de leer casi toda la información del teléfono como los SMS, geolocalización, correos, notas o mensajes.

Documentos filtrados en enero de 2014, revelaron que las agencias interceptan información personal a través de internet, redes sociales y aplicaciones populares, como Angry Birds, que recopilan información para temas comerciales y de publicidad. Además, según The Guardian, el GCHQ tiene una wiki con guías de las diferentes aplicaciones y redes de publicidad para saber los diferentes datos que pueden ser interceptados. Una semana después de salir esta información a la luz, el desarrollador finlandés Rovio, anunció que estaba reconsiderando sus relaciones con las distintas plataformas publicitarias y exhortó a la industria en general a hacer lo mismo.

Las informaciones revelaron que las agencias realizan un esfuerzo adi-

cional para interceptar búsquedas en Google Maps desde Android y otros teléfonos inteligentes para recopilar ubicaciones de forma masiva. La NSA y el GCHQ insistieron en que estas actividades cumplen con las leyes nacionales e internacionales, aunque The Guardian afirmó que «las últimas revelaciones podrían sumarse a la creciente preocupación pública acerca de cómo se acumula y utiliza la información, especialmente para aquellos fuera de Estados Unidos de Norte América, que gozan de menos protección en temas de privacidad que los estadounidenses».

6.3 Alfabetismo Digital

Según la organización Common Sense Media, el alfabetismo digital es la capacidad de encontrar, identificar, evaluar y usar la información encontrada en medios digitales de manera efectiva. Básicamente, es la misma definición tradicional de alfabetismo, pero adaptada a la era digital y a fuentes no tradicionales de información.

El anuario del 2016 de la UNESCO en "Alfabetización mediática e informacional para los objetivos de desarrollo sostenible" hace referencia a las "Cinco leyes de la alfabetización mediática e informacional":

1. La información, la comunicación, las bibliotecas, los medios de comunicación, la tecnología, el internet y otras fuentes de información se encuentran en la misma categoría. Ninguna es más relevante que la otra ni debe ser tratada como tal.

2. Cada ciudadano es un creador de información o conocimiento y tiene un mensaje. Todas las personas deben estar facultadas para acceder a nueva información y expresarse.

3. La información, el conocimiento y los mensajes no siempre están exentos de valores o prejuicios. Cualquier conceptualización, uso y aplicación de alfabetismo digital debe presentar este hecho de manera transparente y comprensible para todos los ciudadanos.

4. Todo ciudadano desea conocer y comprender información, conocimientos y mensajes nuevos, así como comunicarse; y sus derechos nunca deben ser comprometidos.

5. El alfabetismo digital es un proceso dinámico de experiencias vividas. Se considera completa cuando incluye conocimientos, habilidades y actitudes, cuando abarca el acceso, la evaluación, el uso, la producción y la comunicación de información, de contenido mediático y tecnológico.

De igual manera, en el anuario de la UNESCO se exponen 10 habilidades que deben desarrollarse para lograr la alfabetización digital, o como lo define el anuario, alfabetización mediática e informacional. Estas habilidades son:

- Interactuar con información referente a los medios y la tecnología.
- Ser capaz de aplicar habilidades técnicas de comunicación de información para procesar la información y producir contenido mediático.
- Utilizar, de manera ética y responsable la información y comunicar su comprensión o conocimiento adquirido a una audiencia o lectores en una forma y medio apropiados.
- Extraer y organizar información y contenidos.
- Evaluar de forma crítica la información y el contenido presentado en los medios y otras fuentes de información, incluyendo medios en línea, en términos de autoridad, credibilidad, propósito y posibles riesgos.
- Localizar y acceder a información de contenido relevante.
- Sintetizar las ideas extraídas del contenido.
- Comprender las condiciones bajo las cuales se pueden cumplir esas ideas o funciones.
- Comprender el papel y las funciones de los medios, incluyendo medios en línea, en la sociedad y su desarrollo.
- Reconocer y articular la necesidad de información y de los medios.

Internet ha sido una herramienta que ha transformado y definido la comunicación en el siglo XXI. A través de sus múltiples interfaces, internet ha tenido éxito, para que tanto individuos como organizaciones se conecten, se comuniquen e intercambien información. Las plataformas tecnológicas y las redes sociales han acelerado la velocidad a través de la cual los usuarios

pueden acceder y recuperar información, simplificando el proceso en el que las noticias se difunden, actualizan e incluso se comunican. Hoy en día, es prácticamente instantáneo darse cuenta de un evento de noticias sin que sea necesariamente comunicado por medios tradicionales como los periódicos o la radio.

La facilidad a través de la cual las personas ahora pueden comunicarse ha traído una sensación de democratización a la libertad de expresión. Transformar la libertad de expresión ha posibilitado nuevas capacidades para crear y editar contenido, generando nuevas oportunidades para el periodismo alternativo; nuevas capacidades de organización y movilización (que apoyan en gran medida otros derechos, como la libertad de asociación); y nuevas posibilidades para innovar y generar desarrollo económico (apoyando los derechos sociales y económicos).

Sin embargo, esta facilidad en el intercambio y la creación de información también presenta desafíos tanto para las organizaciones como para las personas que son usuarias de dichas redes, tanto como fuente, como usuario final. Aunque estos desafíos varían en escala, todos son igualmente significativos. Algunos de los más destacados incluyen el desafío a la calidad superficial de la información, la susceptibilidad a la información errónea y la exposición a ataques cibernéticos. Por lo tanto, la necesidad de mitigar y mantener la integridad de esta información se ha convertido en un área de trabajo creciente para muchas organizaciones públicas y privadas.

En las siguientes secciones se ofrece una variedad de técnicas y mejores prácticas para mitigar y contrarrestar los desafíos mencionados anteriormente. Sin embargo, es importante tener en cuenta, al leer estas recomendaciones, la posición que representas o con la que te asocias. Algunas de las recomendaciones pueden no ser aplicables a una figura pública, como políticos, activistas u otros actores cuyas mejores prácticas en las redes sociales están sujetas a un mayor escrutinio. En este sentido, el ejercicio de los derechos de expresión, reunión y protesta se debe respetar, y debe ser respetado, en el ámbito digital al tiempo que se garanticen prácticas más seguras de internet.

6.4 Amenazas a la Ciberseguridad

La aparición de vulnerabilidades en los sistemas operativos y los métodos de encubrimiento de los atacantes, lo convierten en una práctica en aumento. Algunos de los principales ataques en la red, hacia donde dirigen su mirada

los *Hackers* para vulnerar la seguridad, pueden definirse como:

- Shoulder Surfing: es una técnica mediante la que el ciberdelincuente consigue nuestra información mirando "por encima del hombro" desde una posición cercana, mientras utilizamos los dispositivos sin darnos cuenta.

- Dumpster Diving: se le conoce como el proceso de buscar en nuestra basura para obtener información útil sobre nuestra persona o empresa que luego puede utilizarse contra nosotros para otro tipo de ataques.

- Malware: el término se refiere de forma genérica a cualquier Software malicioso que tiene por objetivo infiltrarse en un sistema para dañarlo. Comúnmente se asocian como tipos de Malware a los virus, gusanos y troyanos.

- Virus: es un código que infecta los archivos del sistema mediante un programa maligno, pero para ello necesita que el usuario lo ejecute directamente. Una vez activo, se disemina por todo el sistema a donde el equipo o cuenta de usuario tenga acceso, desde dispositivos de Hardware hasta unidades virtuales o ubicaciones remotas en una red.

- Gusanos: es un programa que, una vez infectado el equipo, realiza copias de sí mismo y las difunde por la red. A diferencia del virus, no necesita la intervención del usuario, ya que pueden transmitirse utilizando las redes o el correo electrónico. Son difíciles de detectar, pues su objetivo es difundirse e infectar a otros equipos, y no afectan inicialmente el funcionamiento normal del sistema. Su uso principal es el de la creación de redes zombies (Botnets), utilizadas para ejecutar acciones de forma remota como ataque de denegación de servicio (DoS) a otro sistema.

- Troyanos: similares a los virus, sin embargo, mientras que este último es destructivo por sí mismo, el troyano lo que busca es abrir una puerta trasera (Backdoor) para favorecer la entrada de otros programas maliciosos. Su misión es precisamente pasar desapercibido e ingresar a los sistemas sin que sea detectado como una amenaza potencial. No se propagan a sí mismos y suelen estar integrados en archivos ejecutables aparentemente inofensivos.

- Spyware: es un programa espía, cuyo objetivo es recopilar información de un equipo y transmitirla a una entidad externa sin el consentimiento del propietario. Su trabajo suele ser silencioso, sin dar muestras de su funcionamiento, llegando incluso a instalar otros programas sin que se perciban. Las consecuencias de su infección incluyen, además, pérdida considerable del rendimiento del sistema y dificultad para conectarse a internet.

- AdWare: su función principal es la de mostrar publicidad. Aunque su intención no es la de dañar equipos, es considerado por algunos una clase de Spyware, ya que puede llegar a recopilar y transmitir datos para estudiar el comportamiento de los usuarios y orientar mejor el tipo de publicidad.

- Ransomware[92]: este es uno de los más sofisticados y modernos ataques, ya que lo que hace es secuestrar datos (encriptándolos) y pedir un

[92] El Ransomware se ha convertido en la principal amenaza para la ciberseguridad mundial. Desde que el troyano WannaCry afectó en la primavera de 2017 al menos a 200.000 equipos y servidores de 150 países, poniendo 'contra las cuerdas' a importantes empresas, el uso de este tipo de ataques informáticos no ha dejado de aumentar y son cada vez más numerosos, sofisticados, peligrosos y masivos.

Si en sus inicios los atacantes se conformaban infectando ordenadores de consumo a cambio de unos pocos dólares, todos los informes apuntan que los ciberdelincuentes están enfocando su ámbito de actuación preferente al segmento empresarial, organizaciones, administraciones e infraestructuras públicas con Malware tan peligroso como 'CryptoWall', 'Babuk', 'Black Kingdom', 'Ryuk' o 'CryptoLocker' que destacan por su alto nivel de código y su capacidad de control de ordenadores y redes mediante el cifrado de archivos.

El número de víctimas por Ransomware es ya interminable. La última conocida (abril 2021) ha sido la cadena de tiendas Phone House, pero la lista de los últimos meses es amplísima: la multinacional Acer; el estudio CD Projekt Red; la asociación deportiva NBA; el líder en cámaras Canon; el desarrollador japonés Capcom; la firma de seguros Mapfre; municipios y hospitales estadounidenses o la infraestructura del SEPE, el Servicio Público de Empleo Estatal de España que gestiona subsidios y maneja datos personales de millones de desempleados.

Solo es un ejemplo de afectados y, además, se sospecha que otras empresas han recibido ataques, aunque han preferido no divulgarlos públicamente ante la pérdida reputacional que suponen estos incidentes que son una lacra que parece imparable. Realmente, no hay sistema operativo, plataforma, dispositivo o red informática que esté a salvo, porque el Ransomware emplea cualquier tipo de vulnerabilidad, tipo de Malware o de ataque para secuestrar los equipos. Y no en todas las ocasiones es detectado a tiempo por los sistemas de seguridad y Software antimalware.

Teniendo en cuenta que un Ransomware típico puede infectar dispositivo móviles, ordenadores personales, servidores o redes, bloqueando su funcionamiento y/o acceso a una

rescate por ellos. Normalmente, se solicita una transferencia en dinero electrónico (Bitcoins), para evitar el rastreo y localización. Este tipo de ciberataque va en aumento y es uno de los más temidos en la actualidad.

- Stalkerware: se trata de un programa que permite el seguimiento y monitorización de la actividad del usuario en un dispositivo móvil como teléfono, tableta o computadora. El problema de este tipo de programas es que no han sido creados puntualmente para el espionaje y el acoso, sino para el intercambio de datos entre dispositivos de manera más sencilla. Y qué sí se instalan sin el consentimiento de la persona, permite espiar sus comunicaciones y toda su actividad gracias a las funcionalidades y sensores incorporados en el dispositivo.

- Escaneo de Puertos: técnica empleada para auditar dispositivos y redes con el fin de conocer qué puertos están abiertos o cerrados, los servicios que son ofrecidos, así como comprobar la existencia de algún cortafuegos (Firewall), la arquitectura de la red, o el sistema operativo, entre otros aspectos. Su empleo permite al atacante realizar un análisis preliminar del sistema y sus vulnerabilidades, con miras a algún otro tipo de ataque, pues cada puerto abierto en un dispositivo, es una potencial puerta de entrada al mismo.

- Phishing: no es un Software, se trata más bien de diversas técnicas de suplantación de identidad para obtener datos privados de las víctimas, como contraseñas o datos bancarios. Los medios más utilizados son el correo electrónico, mensajería o llamadas telefónicas, y se hacen pasar por alguna entidad u organización conocida, solicitando datos confidenciales, para posteriormente ser utilizado por terceros en su beneficio.

- Whaling: es un método para simular ocupar cargos de nivel superior en una organización con el objeto de conseguir información confidencial u obtener acceso a sistemas informáticos con fines delictivos.

- El Smishing: ocurre cuando se recibe un mensaje de texto corto (SMS)

parte o a todo el equipo apoderándose de los archivos con un cifrado fuerte y exigiendo una cantidad de dinero como "rescate" para liberarlos, el mejor (y casi único) de los consejos en ciberseguridad es la prevención con las copias de seguridad como máximo exponente. La opción de pagar el rescate para recuperar el acceso a los archivos es muy negativa para la industria ya que retroalimenta aún más esta amenaza.

en el teléfono celular, por medio del cual se solicita al usuario llamar a un número de teléfono o ir a un sitio Web.

- El Vishing: es la estafa que se produce mediante una llamada telefónica que busca engañar, suplantando la identidad de una persona o entidad para solicitar información privada o realizar alguna acción en contra de la víctima.

- Juice-jacking o Video-jacking: son los nombres que se han puesto a los procesos por los cuales, a través de un puerto (generalmente USB) nos conectamos a un puerto Hackeado, de esta forma el ciberdelincuente puede instalar, grabar datos, tomar video o sacar datos de nuestros dispositivos móviles. Esto se ha vuelto común en los puertos de recarga de energía públicos a través del puerto USB. Si se hará uso de este tipo de servicios, es recomendable adquirir un dispositivo del tipo PortaPow de carga rápida con adaptador USB que inhabilitan los pines de datos permitiendo sólo la carga del dispositivo.

- Keylogger (registrador de teclas): es un Software malicioso o dispositivo en Hardware -generalmente conectado al teclado- que se encarga de registrar las pulsaciones que se realizan en el teclado, para posteriormente usarlas para robar información privada.

- Criptomining es un Malware diseñado para la extracción de Criptomonedas en nuestros dispositivos. Si el usuario accesa a un sitio Web que esté infectado, el Malware se puede descargar de forma inadvertida a través de una descarga automática y nuestro dispositivo comenzará a desenterrar una moneda criptográfica seleccionada para los Hackers; la infección será notoria por un uso intensivo de nuestro dispositivo.

- Botnets (Redes de robots): Son computadoras o dispositivos conectados a la red (teléfonos inteligentes, tabletas, etc.) infectados y controlados remotamente, que se comportan como robots (Bots) o zombies, quedando incorporados a redes distribuidas, las cuales envían de forma masiva mensajes de correo Spam o código malicioso, con el objetivo de atacar otros sistemas o dejarlos fuera de servicio.

- Denegación de Servicios: tiene como objetivo inhabilitar el uso de un sistema o computadora, con el fin de bloquear el servicio para el que está destinado. Los servidores Web poseen la capacidad de resolver un

número determinado de peticiones o conexiones de usuarios de forma simultánea, en caso de superar ese número, comienzan a ralentizarse o incluso bloquearse y desconectarse de la red. Existen dos técnicas para este ataque: la denegación de servicio o DoS (Denial of Service) y la denegación de servicio distribuido o DDoS (Distributed Denial of Service); la diferencia entre ambos es el número de equipos de cómputo que realizan el ataque. En el primero, las peticiones masivas al servicio se realizan desde una misma máquina o dirección IP, consumiendo así los recursos que ofrece el servicio hasta que no tiene capacidad de respuesta y comienza a rechazar peticiones (denegar el servicio); en el segundo, las peticiones o conexiones se realizan empleando un gran número de computadoras o direcciones IP, todas al mismo tiempo y hacia el mismo servicio objeto del ataque, de forma general, las computadoras que lo realizan se encuentran infestadas, formando parte de una Botnet, y comportándose como zombis.

- Ataque MITM (Man In The Middle): conocido como "hombre en el medio", ocurre cuando una comunicación entre dos sistemas es interceptada por una entidad externa simulando una falsa identidad. En este sentido, el atacante tiene control total de la información que se intercambia, pudiendo manipularla a voluntad, sin que el emisor y el receptor lo perciban rápidamente. Es común que se realice empleando redes WI-FI públicas y abiertas, y es muy peligroso ya que se puede obtener información sensible de las víctimas, y es difícil identificarlo si no se poseen los mínimos conocimientos sobre el tema.

- Rootkit: es un tipo de Malware diseñado para infectar una PC, el cual permite instalar diferentes herramientas que dan acceso remoto al equipo de cómputo. Este Malware se oculta en la máquina, dentro del sistema operativo y sortea obstáculos como aplicaciones antimalware o algunos productos de seguridad. El Rootkit contiene diferentes herramientas maliciosas como un módulo para robar los números de tarjeta o cuentas bancarias, un Bot para ataques y otras funciones que pueden desactivar el Software de seguridad.

- SIM Swapping: es la duplicación del SIM de un número telefónico que permite a un atacante usurpe nuestra identidad, pudiendo autentificarse por medio de SMS en diversos servicios que usen la autentificación de dos pasos, incluyendo los servicios bancarios.

- SIM Jacker: es el envió de un mensaje malicioso al dispositivo destino, que si se abre el enlace adjunto, el dispositivo inteligente queda comprometido y se puede extraer toda la información del mismo incluyendo la ubicación.
- 0-Day (día cero): es una nueva vulnerabilidad para la cual no se han creado parches o revisiones, y que se emplea para llevar a cabo un ataque. El nombre se debe a que no existe ninguna revisión para mitigar el aprovechamiento de la(s) vulnerabilidad(es), estas pueden ser utilizadas para qué troyanos, Rootkits, virus, gusanos y otros Malwares se propaguen e infecten más equipos.
- IP Spoofing: el ciberdelincuente consigue falsear su dirección IP y hacerla pasar por una dirección válida en la cual confiamos, de este modo, consigue saltarse las restricciones y puede hacernos conectar a una Web maliciosa.
- Web Spoofing: consiste en la suplantación de una página Web real por otra falsa. La Web falsa es una copia del diseño original, llegando incluso a utilizar una URL muy similar para que demos nuestras credenciales de acceso.
- Email Spoofing: consiste en suplantar la dirección de correo de una persona o entidad de confianza para solicitarnos datos o que descarguemos archivos maliciosos.
- DNS Spoofing: a través de programas maliciosos específicos y aprovechándose de las vulnerabilidades en las medidas de protección, los atacantes consiguen infectar y acceder a nuestro Router suplantando la DNS (Domain Name System). Así cuando tratamos de acceder a una determinada Web desde el navegador, este nos lleva a otra Web elegida por el atacante.
- Sniffing: se trata de una técnica utilizada para escuchar todo lo que se transmite dentro de una red, de esta forma se monitorea el tráfico y se puede capturar información que viaja de forma no cifrada para analizarla y hacerse de nuestros datos.
- Ataque de inyección de SQL (Structured Query Language): es aprovechar una o más vulnerabilidades de servidores SQL que hace que se divulgue información confidencial de la base de datos que de otra manera

no haría al inyectar código SQL malicioso. Esto representa un riesgo enorme si la base de datos almacena información de identificación personal, como números de tarjetas de crédito, información personal y contraseñas.

- Baiting o Gancho: se sirve de algún medio físico como una unidad USB infectada que el atacante deja al alcance de los usuarios, para que cuando este lo introduzca en su equipo se infecte. También puede usar anuncios en Webs con la que promociona cursos o premios que nos inciten a compartir datos o descargar Software malicioso.

En general, para poder llevar a cabo alguno de estos ataques, los intrusos deben disponer de los medios técnicos, los conocimientos y las herramientas adecuadas, deben contar con una determinada motivación o finalidad, y se tiene que dar además una oportunidad que facilite el desarrollo del ataque, como podría ser un fallo en la seguridad del sistema informático elegido.

6.5 Dicen que si no Pagas por un Producto, Entonces el Producto Eres Tú.

¿Cuál es el modelo de negocio de empresas tecnológicas como Facebook? usar Google, Facebook, Messenger, Instagram y WhatsApp es completamente gratis. Y pese a ello, por ejemplo Facebook saca cada vez más dinero por usuario. Esto es posible gracias al uso que Facebook hace de nuestros datos. Teóricamente, y escándalos por fallos de seguridad al margen, Facebook no vende nuestros datos a terceros, sino que vende a terceros el acceso a nosotros gracias al uso de nuestros datos.

Así, las empresas tecnológicas (no importa si pagamos o no por un servicio o producto) van detectando nuestros gustos e intereses en base al dispositivo desde el que accedemos, las páginas que seguimos, nuestro historial de navegación y otros factores, y crea un perfil sobre cada uno de nosotros. Luego vende espacios de nuestro Feed a las empresas que busquen gente como nosotros (franja de edad determinada, localización, aficiones...). Y ya se está planteando formas de ir más allá mediante acuerdos con terceros, como uno con la banca para poder saber el dinero que tenemos en nuestra cuenta.

Esta red publicitaria que tan optimizada está merece el aplauso por su logro técnico, pero quizás no sea tan plausible si tenemos en cuenta ese rastreo

tan agresivo y la escasa preocupación de las empresas por la privacidad de sus usuarios.

Por otro lado, el uso generalizado de las redes sociales entraña algunos riesgos que, siguiendo recomendaciones básicas, se pueden evitar. Como cualquier comunidad frecuentada por miles de usuarios (o, como sucede a veces con las redes sociales, por miles de millones), se deben conocer los mecanismos de control y de seguridad para poder utilizarlos con fiabilidad para mantener nuestra privacidad y es por eso que el usuario tiene que ser especialmente cuidadoso con el uso que hace de la red social, a continuación daremos algunas recomendaciones:

- No necesitamos informar de todo y a todos, y menos con información sensible. Por ello, cuanta menos información pongamos, mejor.
- A menudo publicamos notas o mensajes en el muro de un amigo. Esto también es visible para otros usuarios de Facebook o de la red social que sea.
- Se dice que una foto vale más que mil palabras. Estamos dando información sobre nuestros hábitos, nuestros movimientos, a posibles atacantes.
- Hay que asegurarse de que nuestro contenido en las redes sociales sea visible solo para amigos y familiares.
- Google, Twitter, Facebook e Instagram usan el geoetiquetado mediante el GPS para ayudar a los usuarios a marcar la ubicación donde se hizo una foto, vídeo y ayudar a que el perfil sea más «social». Cualquier usuario puede interpretar fácilmente información como nuestro estado económico, estilo de vida, lugares frecuentes y la rutina diaria a través de los medios con etiquetas geográficas.
- En una red social lo normal es que cada usuario se identifique con su nombre y apellido real y que aporte datos personales, como si estudia o trabaja, con quién se relaciona o en qué ciudad vive.Esto hace que su exposición pública sea mucho mayor que antes, esto implica la pérdida del anonimato algo que antes era común y habitual en internet.
- En las redes sociales, una vez que se pulsa el botón de "publicar", esa información es enviada a los contactos del usuario. Eso significa que si

más adelante el usuario se arrepiente de lo dicho, publicado o mostrado y trata de borrarlo, solo conseguirá eliminarlo de su propio perfil pero no de internet.

- Quizás lo más importante de estos consejos para mantener la seguridad en redes sociales es el sentido común en lo que publicamos y comentamos en ellas.

Acuerdos de Privacidad la lectura de los acuerdos de privacidad[93] orientará al usuario sobre qué datos se comparten o no, y también se ofrece la opción de seleccionar o anular las opciones de privacidad, seguridad o administrativas escogidas para proteger la cuenta y el dispositivo.

Al registrarte en una cuenta de redes sociales, por defecto, toda la información anotada en un perfil se hace pública, lo que significa que cualquier persona puede acceder al contenido que hayas registrado en una cuenta. Sin embargo, las necesidades y preferencias de privacidad varían de persona a persona. Mientras que algunos usuarios prefieren tener una mayor exposición y así poder promocionar su contenido en redes sociales, otros prefieren incluir muy poca o ninguna información.

Para lograr una mayor protección del usuario y su información, es importante evaluar en qué medida la persona está dispuesta a incluir información personal en su perfil. Por consiguiente, ten en cuenta lo siguiente al:

- Seleccionar un nombre de usuario: el nombre de usuario es el "nombre digital" que una persona se asigna a sí misma o a su organización para ser identificada en línea. Si existe la preferencia de no ser fácilmente identificada en ninguna plataforma, pero poder continuar usando estas redes, la persona puede asignar y usar un seudónimo que puede estar relacionado o no con esa persona. Además, la persona puede cambiar su nombre de usuario en cualquier momento simplemente ingresando

[93] Si de verdad leyéramos los términos y condiciones de uso de plataformas Online seleccionadas (datos de abril 2020), tardaríamos (con una velocidad de 240 palabras por minuto) aproximadamente:

Microsoft 1:03:30 s (15,260 palabras), Spotify 35:48 s (8,600 palabras), Tik Tok 31:24 s (7,459 palabras), Apple 30:30 s (7,314 palabras), Zoom 30:12 s (7,243 palabras), Tinder 25:54 s (6,215 palabras), Uber 25:36 s (5,658 palabras), Twitter 25:30 s (5,633 palabras), LinkedIn 18:06 s (4,346 palabras), Facebook 17:12 s (4,132 palabras), Amazon 14:12 s (3,416 palabras), YouTube 13:42 s (3,308 palabras, Netflix 11:00 s (2,628 palabras), Instagram 9:42 s (2,451 palabras).

a la configuración de su (s) cuenta (s). El nombre de usuario no tiene que ser coherente en todas las redes sociales; estas pueden variar según las preferencias en cada una.

- Incluir una imagen en la cuenta: el usuario tiene la opción de personalizar una cuenta con la inclusión de una foto del perfil. Cuando un usuario prefiere no ser identificado, se sugiere elegir una imagen en la que no pueda ser identificado y cambiarla cuando sea necesario. Ten en cuenta que cuando se usa la misma imagen en todas las redes sociales, la simple búsqueda de imágenes puede llevar a otras cuentas.

- Incluir una ubicación: cuando se activan los servicios de ubicación en la plataforma de redes sociales, estos permiten a los usuarios rastrear el origen de cualquier actividad de medios en línea. Es importante tener en cuenta que una vez que se activa esta función, permanecerá activa hasta que se elija deshabilitarla en la configuración de privacidad. A pesar de que se permitía que esta característica estuviera activa en el pasado, las plataformas tienen la funcionalidad de deshabilitar la ubicación de cualquier contenido que se haya publicado en sus cuentas.

Sin embargo, aunque un usuario active o desactive la función de compartir la ubicación, potencialmente, la ubicación de un usuario podrá ser descubierta por el contenido que comparta o las imágenes que haya elegido para compartir.

6.6 Datos que Recopila Google

Desde el escándalo de Cambridge Analytica[94] en 2018, poco a poco todos nos hemos ido haciendo más conscientes de nuestra privacidad, y cómo las grandes empresas recopilan nuestros datos casi sin que nos demos cuenta. Ya no solo Facebook y las redes sociales, ya que hay otras empresas como Google que pueden recopilar muchos más datos sobre lo que hacemos cada día.

En parte, esto se debe a que es una empresa que ofrece una gran cantidad de servicios gratuitos que utilizamos casi sin pensar, y en otra, porque es la dueña de todo lo que buscamos en la red (si usamos sus productos y servicios).

[94] Se le acusa de tratar de influenciar los resultados de las elecciones presidenciales de 2016 en los Estados Unidos de Norteamérica.

Para hacerme una mejor idea de hasta dónde llega, podemos descargar todos nuestros datos y sí los revisamos es seguro que tendremos una desagradable sorpresa, que nos llevan a la conclusión de que Google sabe mucho más sobre nosotros que la propia Facebook, como por ejemplo, por dónde he viajado y qué aplicaciones utilizo y cuándo.

Algunos de estos datos los puedes encontrar fácilmente en algunas páginas de datos. Pero para otros, hemos utilizado la herramienta Google Takeout para descargar una copia de seguridad de todo lo que tienen sobre nosotros. Ya es bastante significativo que esta copia ocupe varios Gigas. Algo que debes saber es que puedes configurarlo para que muchos de los datos que Google tiene sobre ti se autodestruyan cuando haya pasado determinado tiempo.

Una cosa que hay que tener en cuenta es que Google no tiene por qué estar vendiendo o revisando minuciosamente estos datos, aunque tampoco hay manera de asegurarse de que no lo hagan con alguno. Sin embargo, en algunos casos nos parece excesivo incluso el simple hecho de que recopilen algunos de ellos. Para que sepas de lo que estamos hablando, aquí tienes una lista del tipo de datos personales que he visto que Google recopila.

Lo primero que llama la atención al mirar los datos de Google, es que sabe por dónde te has estado moviendo. Por ejemplo, tiene una función de cronología en la que puedes ver todos los movimientos que ha ido registrando sobre ti cuando tienes habilitado el historial de ubicaciones de Google Maps. Este suele estar activado por defecto en tus dispositivos, lo que quiere decir que si no cambias la configuración deliberadamente seguirá todos tus pasos a través de tu móvil, tableta o cualquier otro dispositivo con el que estés identificado con tu cuenta de Google.

Todos estos datos Google te los va presentando de forma ordenada y cronológica, mostrándote a qué hora has salido de un sitio, el medio de transporte por el que has ido y qué paradas has hecho. Sorprende la minuciosidad con la que Google lo registra todo. Además, en el caso de que te hayas ido de viaje sacando muchas fotos, Google también va a saber dónde y cuándo sacaste las fotografías con tu móvil, y todo te lo va a mostrar de forma ordenada viendo cómo hiciste el camino y dónde hiciste las fotos. Incluso distingue los trayectos que hiciste en coche y los que has hecho a pie. Esto lo hace utilizando los metadatos de tus fotos y cruzándolos con la información de ubicaciones que recopila.

Y más allá de esta herramienta, en la copia de seguridad de Google también puedes encontrar otras sorpresas. Por una parte, hay una carpeta con un historial de ubicaciones, en la que puedes ver las coordenadas por las que

te has movido en los últimos días y la manera en la que lo has hecho. Es como lo de Google Maps, pero con el código en bruto.

También guarda las actividades que hayas registrado utilizando Google Fit. Aquí puedes encontrar dos carpetas diferentes, una con todos los ejercicios que has hecho utilizando esta aplicación, con horas y coordenadas, y otra con un archivo diario en el que se puede ver todos los movimientos agregados en cada momento. Da igual que lleves años sin utilizar la aplicación guarda los datos obtenidos de aplicaciones de terceros al vincularlas con el servicio.

Google también tiene una sección Mi Actividad en la que recoge y almacena todas las búsquedas que haces en Internet, así como el contenido que consumes dentro de portales pertenecientes a Google. Aquí no solo vas a encontrar las búsquedas hechas a través del buscador, sino las búsquedas realizadas en Chrome y tu Android.

La verdad es que al mirar mi cronología no he podido evitar sentirme un poco incómodo, sobre todo porque todos los datos están perfectamente organizados y van acompañados de enlaces. De esta manera, puedes saber que has buscado determinados términos y volver a pulsar sobre ellos para ver los resultados de búsqueda, pero también puedes saber qué perfiles de Twitter o Facebook has visitado o las páginas a las que has entrado.

Además de esto, Google también recopila cuando utilizas otras plataformas como Stadia. Incluso si simplemente pulsas en el botón de Discover de la aplicación de Google para ver los temas que la propia Google considera importantes para ti, va recopilando cuáles son los temas sobre los que te ha mostrado información cada vez.

Es más, dentro de algunas páginas también te dice en qué secciones has navegado. Esto se vuelve un poco más preocupante con el tema de los foros, ya que te dice el título de cada hilo que has visitado dentro de un mismo foro, e incluso te ofrece enlaces para volver a él.

La Web también registra cuántas veces utilizas cada aplicación móvil y lo que es peor, todos estos datos siguen apareciendo aunque borres el historial de Chrome. He probado borrar las páginas a las que he entrado hoy con el navegador, y estas seguían apareciendo en la página de My Activity.

¿Y qué implicaciones tiene esto? Pues no sólo que Google sabe exactamente todo lo que haces en Internet si utilizas sus productos, sino que cualquiera que se ponga frente a tu computadora, tableta o teléfono inteligente (si tiene acceso a tu cuenta) va a poder saberlo, ya que al entrar a My Activity Google no pide que confirme mi identidad. Así que cualquiera puede con relativa facilidad tener acceso a esta información.

En resumen Google tiene acceso a:

- Tu nombre, tu dirección, tu edad, tu correo electrónico. Tu modelo de teléfono, tu proveedor de telefonía celular, tu plan y tu consumo telefónico y de internet.
- Las palabras que usas con más frecuencia dentro de tus correos electrónicos. Todos los correos que hayas escrito o recibido, incluido Spam. Los nombres de tus contactos y sus direcciones y teléfonos.
- Las fotografías que tomas con tu teléfono Android, aunque las hayas borrado y aunque no las subas nunca a ninguna red social. Los sitios a los que vas, dentro y fuera del país; la fecha en la que fuiste y la ruta que tomaste. Qué tan rápido llegaste. La tarjeta de crédito o débito que usas para pagar.
- Todos los sitios de internet que has visitado en Google, con qué frecuencia y lo que viste dentro de cada uno. En qué idioma buscas. A qué hora navegas. Con quién has hablado vía Hangouts. Qué videos te gustan. Qué música oyes, etc.

Para tratar de minimizar nuestra huella digital, si soy usuario de los servicios de Google puedo desactivar todos los servicios de rastreo a los que me da opción en su página de "Administrar tu cuenta de Google", no es notoria la degradación del servicio por dichas desactivaciones. Pero esto no impide que Google y servicios de terceros recolecten y transmitan nuestros datos, solo no se almacenarán en nuestra cuenta, ni tendremos acceso a los mismos.

También puedo prescindir de los servicios de Google usando otras alternativas no tan invasivas como describimos más adelante en este texto.

6.7 ¿Cómo me Protejo?

Algunas normas básicas de ciberseguridad para nuestra seguridad, tener privacidad y evitar la vigilancia son:

- Usar contraseñas largas, robustas (incorporar mayúsculas, minúsculas, números y símbolos) y diferentes para cada servicio que lo solicite, para facilitar el manejo de las contraseñas es recomendable el uso de gestores de contraseñas.

- Cifrar Dispositivos, Discos y Unidades de Respaldo para mantener la confidencialidad de nuestra información.
- Mantener actualizado el sistema operativo y las aplicaciones (sólo instalar las necesarias) de nuestros dispositivos además de usar mecanismos que coadyuven en la seguridad como cortafuegos, antivirus, entre otras aplicaciones de seguridad.
- Generar respaldos periódicos -compactados y cifrados- de nuestra información y garantizar que es posible restituir nuestros datos de dichos respaldos (una buena estrategia de respaldo es la siguiente: mantener tres copias de cualquier fichero importante -una principal y dos respaldos-, mantener los ficheros en dos tipos distintos de almacenamiento para protegerlos ante distintos riesgos y almacenar una copia de seguridad fuera de nuestra casa u oficina).
- Para ciertas actividades en la red o el uso de programas poco confiables es recomendable el uso de máquinas virtuales que limitan los posibles daños por programas maliciosos.
- Al navegar en internet es necesario hacerlo de forma segura para no exponernos de forma innecesaria, mediante un uso seguro y aceptable de las herramientas en la Nube.
- Conocer las medidas mínimas de protección para una navegación segura en internet.
- Conocer las técnicas usadas en ataques de inteligencia social, para no ser víctimas de la ciberdelincuencia.
- Al hacer uso de videoconferencias conocer las políticas de privacidad y las medidas de seguridad para no exponernos a ciberacoso y pérdida de datos.
- Hacer un uso correcto de los dispositivos personales cuando estos son usados para el trabajo.
- Uso de escritorios remotos y virtuales como una forma de mitigar los riesgos cuando se trabaja de forma remota.

Entre otras tantas cosas que el usuario actual de las Tecnologías de la Información y la Comunicación (TIC) debe dominar. Además, debemos conocer algunas recomendaciones útiles para reducir nuestra huella en internet, como:

- No compartir nuestro correo electrónico y número telefónico, pese a que es habitual utilizarlos para registrarnos en sitios Web, si los compartimos libremente por internet nos exponemos a ser víctimas de Spam, Phishing y todo tipo de ciberataques basados en ingeniería social.

- El uso de cuentas de correo alternativas nos permite registrarnos en diferentes sitios Web sin utilizar datos y/o cuentas personales o de trabajo.

- Usar cuentas de correo seguro, anónimo y bidireccional como el servicio de Tutanota, Mailfence o ProtonMail. O usar el modo confidencial -en el cual podemos establecer fecha de vencimiento y contraseña para cada correo- al enviar correos desde cuentas Google.

- Acceder a las opciones del navegador para eliminar cada cierto tiempo la información almacenada en formas de Cookies, Caché o el historial de navegación.

- El uso de modo privado o incógnito[95] en el navegador nos permite no almacenar información sobre las páginas Web visitadas ni se guardan las Cookies, ya que se eliminarán al salir del navegador.

- Evitar revelar información sensible en redes sociales sobre nosotros, familia y conocidos, en medida de lo posible debemos vigilar lo que publicamos, quién puede verlo y configurar las opciones de privacidad.

[95] En Chrome y otros navegadores Web, el modo incógnito -del cual Google sostiene que este modo- está diseñado para evitar el almacenamiento local de datos, y realmente no protege nuestra privacidad, aunque otros usuarios en el mismo dispositivo no verán la actividad en modo incógnito, los sitios Web y servicios, incluidos los de Google, aún pueden recopilar datos. La actividad, como descargas, favoritos y elementos de la lista de reproducción, se almacenará.

Cabe mencionar que ningún navegador ofrece el 100% de privacidad ni anonimato al usuario, lo que si, es que entre los diferentes navegadores existentes, podremos encontrar navegador web (como por ejemplo Brave) que ofrecen capas adicionales de protección de los datos del usuario, pero esto no los vuelve 100% eficientes en ello, pues incluso navegadores como Tor (para la Dark Web) tiene sus fallos.

- El uso de Webs seguras (https y certificados digitales) aseguran que la información que intercambiamos con ellas, como datos bancarios o contraseñas viajen de forma cifrada.
- Usar navegadores especializados (Tor) y lugares de búsqueda de información (DuckDuckGo) que nos permitan una navegación segura al no guardar lo que buscamos, ni los sitios donde accedemos.
- El uso del GPS en nuestros dispositivos móviles es una gran herramienta en nuestra vida digital, pero debemos tenerlo activado sólo cuando sea necesario, ya que muchas aplicaciones comparte nuestra ubicación en tiempo real sin nuestro conocimiento y alguien puede conocer donde vivimos, los lugares que frecuentamos o cuando no estamos en casa.
- Al tomar fotografías o vídeos desactivar el guardado de datos *Exif* (Exchangeable Image File) también conocidos como metadatos: datos sobre la cámara, datos sobre la fotografía y datos sobre el origen como ubicación por GPS, etc.
- Nunca tomar vídeos o fotos comprometedoras, de carácter íntimo o sexual y menos publicarlas, ya que suponen una gran amenaza a nuestra seguridad y pueden tener consecuencias muy graves, como la exposición pública, extorsión o el ciberacoso.
- No compartir fotos, vídeos o audios de menores.
- No debemos compartir vídeos, fotos o audios de terceros sin su aprobación, en especial los que contienen conversaciones privadas que difundan datos personales o información que podría considerarse como revelación de secretos y que la otra persona preferiría no difundir.
- Al emitir opiniones, quejas o comentarios subidos de tono u ofensivos en medios electrónicos nos expone a ser atacados o censurados y en ciertos casos podrían ameritar acciones legales.
- No publicar documentos personales en forma de fotos, vídeos o archivos (*.Docx*, *.PDF*, etc) ya que con su revelación nos expone a una suplantación de identidad y al uso de nuestros datos de forma fraudulenta.

- Cumplir las normas mínimas de protección de dispositivos personales en redes corporativas.
- Hacer uso correcto de escritorios remotos y virtuales en equipos personales para el trabajo remoto.

7 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] https://es.wikipedia.org/wiki/Microsoft_Windows

[2] https://es.wikipedia.org/wiki/Linux

[3] https://es.wikipedia.org/wiki/Unix

[4] https://es.wikipedia.org/wiki/Berkeley_Software_Distribution

[5] https://es.wikipedia.org/wiki/Mac_OS

[6] https://es.wikipedia.org/wiki/Android

[7] https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)

[8] Julia, The Julia Programming Language, https://julialang.org

[9] https://es.wikipedia.org/wiki/Python

[10] https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n)

[11] https://es.wikipedia.org/wiki/C%2B%2B

[12] https://es.wikipedia.org/wiki/Fortran

[13] https://www.gnu.org/philosophy/free-sw.es.html 317

[14] https://es.wikipedia.org/wiki/Software_libre 317

[15] https://www.hispaLinux.es/SoftwareLibre 317

[16] https://es.wikipedia.org/wiki/Software_propietario 315

[17] Diferentes Tipos de Licencias para el Software, https://www.gnu.org/licenses/license-list.html 317, 327

[18] Proyectos de Software Sourceforge, http://sourceforge.net/ 18, 19

[19] Google Code, http://code.google.com 19

[20] Software proyecto GNU, http://www.gnu.org/Software/Software.es.html 18

[21] FSF, Free Software Fundation, http://www.fsf.org/ 18, 317, 327

[22] GNU Operating System, http://www.gnu.org/ 317, 327

[23] GCC, the GNU Compiler Collection, http://gcc.gnu.org/ 18

[24] The Linux Kernel Archives, http://www.Kernel.org/ 18

[25] Debian GNU/Linux el Sistema Operativo Universal, http://www.debian.org

[26] Microsoft Office, http://office.microsoft.com/

[27] OPEN OFFICE, Apache OpenOffice, http://www.openoffice.org

[28] LibreOffice the Document Foundation, http://www.libreoffice.org

[29] QEMU, http://wiki.qemu.org/Main_Page

[30] KVM, http://www.Linux-kvm.org/page/Main_Page

[31] Máquinas Virtuales, https://es.wikipedia.org/wiki/M%C3%A1quina_virtual

[32] Oracle MV VirtualBox, https://www.virtualbox.org

[33] VMware, https://www.vmware.com

[34] Virtual PC, https://www.microsoft.com/es-mx/download/details.aspx?id=3702

[35] Hyper-V, https://msdn.microsoft.com/es-es/library/mt16937(v=ws.11).aspx

[36] Parallels, https://www.parallels.com

[37] Algunos usos de máquinas Virtuales, http://www.configurarequipos.com/doc747.html

[38] Mathtype, https://www.dessci.com/en/products/mathtype/

[39] Scientific WorkPlace, https://www.mackichan.com/

[40] Gummi LaTeX Editor, https://github.com/alexandervdm/gummi

[41] Kile LaTeX Editor, https://kile.sourceforge.net/

[42] Led LaTeX Editor, https://www.latexeditor.org/

[43] Lyx LaTeX Editor, https://www.lyx.org/

[44] Calligra Suite, https://www.calligra.org

[45] Texmaker LaTeX Editor, https://www.xm1math.net/texmaker/

[46] TeXnicCenter LaTeX Editor, https://www.texniccenter.org/

[47] TextPad LaTeX Editor, https://www.textpad.com/

[48] TeXstudio LaTeX Editor, https://texstudio.sourceforge.net/

[49] WinEdt LaTeX Editor, https://www.winedt.com/

[50] LaTeX Beamer Class, https://bitbucket.org/rivanvx/beamer/wiki/Home

[51] LaTeX, A Document Preparation System, http://www.latex-project.org/

[52] The R Proyect for Statistical Computing, http://www.r-project.or/

[53] RWeka, http://cran.r-project.org/Web/packages/RWeka/index.html

[54] Tinn-R Edit code and run it in R, http://www.sciviews.org/Tinn-R/

[55] RStudio Software, Education, and Services for the R community, http://www.rstudio.com/

[56] El economista, https://eleconomista.com.mx/tecnociencia/2013/01/22/clausuraran-negocios-mexico-uso-ilegal-Software

[57] PCworld, http://www.pcworld.com.mx/UNAM-y-BSA-promueven-el-uso-de-Software-legal/

[58] W. Gropp, E. Lusk, A. Skjellem, *Using MPI, Portable Parallel Programming Whit the Message Passing Interface*. Scientific and Engineering Computation Series, 2ed, 1999.

[59] I. Foster; *Designing and Building Parallel Programs*. Addison-Wesley Inc., Argonne National Laboratory, and the NSF, 2004.

[60] Jorge L. Ortega-Arjona, *Patterns for Parallel Software Design*, Wiley series in Software Design Patterns, 2010.

[61] S.J. Pennycook, S.D. Hammond, S.A. Jarvis and G.R. Mudalige, *Performance Analysis of a Hybrid MPI/CUDA Implementation of the NAS-LU Benchmark*. ACM SIGMETRICS Perform. Eval. Rev. 38 (4). ISSN 0163-5999, (2011).

[62] XMPI, *A Run/Debug GUI for MPI*.
http://www.lam-mpi.org/Software/xmpi/

[63] VAMPIR, *Performance Optimization*.
http://www.vampir.eu/

[64] https://es.wikipedia.org/wiki/SQL

[65] https://es.wikipedia.org/wiki/NoSQL

Sitios Web revisados:

- https://www.ibm.com/security
- https://www.welivesecurity.com/la-es/
- http://blogs.eset-la.com/
- http://www.criptored.upm.es/
- https://www.securityfocus.com/
- https://securiteam.com/
- https://www.cert.unam.mx/
- http://www.securitytube.net/
- https://www.issa.org/
- https://www.isaca.org/
- https://www.sans.org/
- https://www.eccouncil.org/
- https://www.isc2.org/
- https://www.comptia.org/
- http://oval.mitre.org/
- https://www.offensive-security.com/
- https://www.giac.org/certification/penetration-tester-gpen
- https://www.isecom.org/

- https://www.exploit-db.com/google-hacking-database
- https://www.ccn-cert.cni.es/
- https://www.securityfocus.com/
- https://linuxsecurity.com/
- https://www.identidadrobada.com/
- https://www.ietf.org/
- https://www.iso.org/home.html
- https://www.bis.org/bcbs/
- http://www.mit.edu/hacker/hacker.html

El siguiente índice esta desfasado por cuatro páginas, estamos trabajando para corregirlo.

Declaro terminado este trabajo sufrido, ideado y llevado a cabo entre los años 2010 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