Herramientas Para Respaldo y Restauración en Linux

Antonio Carrillo Ledesma

Herramientas Para Respaldo y Restauración En Linux

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

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

2025, Versión 1.0α[1]

Índice

1 Introducción

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

La informática, es la rama de la ciencia que se encarga de estudiar la administración de métodos, técnicas y procesos con el fin de almacenar, procesar y transmitir información y datos en formato digital. De esta manera, la informática se refiere al procesamiento automático de información mediante dispositivos electrónicos y sistemas computacionales. Los sistemas informáticos deben contar con la capacidad de cumplir tres tareas básicas: entrada (captación de la información), procesamiento y salida (transmisión de los resultados).

Las tecnologías de la información y comunicación (TIC) son el resultado de poner en interacción la informática y las telecomunicaciones. Todo, con el fin de mejorar el procesamiento, almacenamiento y transmisión de la información.

Consiguiendo de esta manera mejorar el nivel de nuestras comunicaciones. Creando nuevas formas de comunicación más rápida y de mayor calidad. Mejoras que reducen costes y tiempo, de aplicación tanto al mundo de la educación, los negocios como a la vida misma. Proporcionándonos una mayor comodidad y mejorando nuestra calidad de vida a la vez que se aboga por el medio ambiente.

No descubrimos nada si decimos que el impacto de internet en nuestra vida cotidiana aumentó y se aceleró con la pandemia. Sino basta con revisar las actividades que se convirtieron en rutina durante el confinamiento, como las clases a distancia, el teletrabajo, o actividades de entretenimiento, por mencionar algunas. Sin embargo, más allá de la infinidad de beneficios y oportunidades que representa internet, también tiene sus riesgos, y así como puede proporcionar satisfacciones que motivan el constante desarrollo de la tecnología, también puede ser la causa de varios dolores de cabeza. Por

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

supuesto, todo depende de la forma en que se utiliza, ya que las consecuencias de su uso dependerá de las prácticas y el comportamiento de los usuarios.

En este sentido y considerando que internet representa un pilar clave en las sociedades actuales, es importante hablar de normas de convivencia y del rol y la responsabilidad que tienen los usuarios para lograr que sea, cada vez más, un ambiente agradable, respetuoso y seguro; algo que sin duda beneficiará a todos.

Sobre Amenazas y Vulnerabilidades informáticas Antes de entrar de lleno en las Amenazas y Vulnerabilidades informáticas, dejaremos en claro brevemente qué son las mismas, y en qué se diferencian ambas.

- Una vulnerabilidad (en términos de informática) es una debilidad o fallo en un sistema de información que pone en riesgo la seguridad de la información pudiendo permitir que un atacante pueda comprometer la integridad, disponibilidad o confidencialidad de la misma, por lo que es necesario encontrarlas y eliminarlas lo antes posible. Estos «agujeros» pueden tener distintos orígenes por ejemplo: fallos de diseño, errores de configuración o carencias de procedimientos.

- Por su parte, una amenaza es toda acción que aprovecha una vulnerabilidad para atentar contra la seguridad de un sistema de información. Es decir, que podría tener un potencial efecto negativo sobre algún elemento de nuestros sistemas. Las amenazas pueden proceder de ataques (fraude, robo, virus), sucesos físicos (incendios, inundaciones) o negligencia y decisiones institucionales (mal manejo de contraseñas, no usar cifrado). Desde el punto de vista de una organización pueden ser tanto internas como externas.

Pequeñas acciones, grandes cambios el concepto de netiqueta es utilizado para referirse a un conjunto de reglas o pautas de comportamiento a la hora de utilizar e interactuar a través de algún servicio de internet, como pueden ser foros, Blogs, juegos en línea, redes sociales o el correo electrónico. En otras palabras, se trata de trasladar las normas de educación del ambiente físico al ámbito digital.

Cuando hablamos de buenas prácticas de convivencia como ciudadanos digitales no nos referimos solamente a los aspectos comunicacionales, sino también relacionados con la seguridad. Prácticas que pueden ayudar a evitar

poner en riesgo a otros usuarios. Por ejemplo, cuando un usuario rompe una cadena con información falsa o un contenido que resulta ofensivo para otros usuarios.

Dicho esto, además de evitar compartir contenido inapropiado, es importante que los usuarios tengan presente que también pueden reportar a un usuario o contenido para que sea eliminado y de esta forma evitar que llegue a más usuarios. En un sentido amplio, los usuarios también podemos involucrarnos para generar mayor conciencia entre las personas que conocemos y estimamos.

Los resultados pueden ser diversos, pero lo que es un hecho es que al igual que en el plano físico, las acciones individuales, por más pequeñas que sean, cuentan y contribuyen a que internet sea un espacio en el que se promuevan más las actitudes positivas, valores, acciones responsables y en general, la cordialidad y respeto que nos gustaría recibir de parte de los otros.

Educación, buenas prácticas y tecnología de seguridad si bien respetar las normas de comportamiento en internet es fundamental para lograr el propósito de generar un espacio más cordial, la seguridad en internet requiere de otros elementos, como la aplicación de buenas prácticas de seguridad en el uso de la tecnología, y por supuesto, de la educación de los usuarios, para de esta manera estar cada vez más y mejor informados.

En este sentido, advertir a tus contactos sobre algún engaño que está circulando, revisar la configuración de la privacidad y seguridad de tus cuentas, evitar compartir información de personal a desconocidos o información de terceros sin su autorización, así como compartir archivos sin antes verificar que sean seguros, son apenas algunas de las recomendaciones para que internet sea un espacio más seguro. Sobre todo en estos tiempos en los que pasamos más tiempo conectados y en el que la circulación de Fake News y engaños parecen ser moneda corriente.

En conjunto con lo anterior, no se debe dejar de lado el uso de las tecnologías de seguridad que en la actualidad resultan básicas debido a la cantidad, complejidad y diversidad de las amenazas informáticas que se propagan por internet. Estas acciones en conjunto nos permitirán utilizar la tecnología de una forma cada vez más consciente, responsable y, por supuesto, segura.

¿Qué componen las tecnologías de la información y comunicación? los dispositivos de cómputo, las redes y los servicios, por tanto,

también pueden ser clasificadas según hagan un uso u otro de estos elementos. En relación a los dispositivos mucho es lo que se ha avanzado. La computadora personal ha evolucionado desde su aparición y sigue haciéndolo a un ritmo vertiginoso. Al igual que los aparatos periféricos que lo complementan, ofreciendo otras posibilidades.

La tecnología no se ha estancado en las computadoras personales. Nos va sorprendiendo introduciendo nuevos tipos de terminales en nuestras vidas o mejorando sus características. ¿Qué fue de aquel teléfono móvil cuya única función era llamar?. Ahora son dispositivos mucho más sofisticados que han revolucionado la comunicación. Las vídeo llamadas, las aplicaciones de mensajes de texto gratuitas, las redes sociales, etc. son algunos ejemplos.

En cuanto a las redes que permiten que los dispositivos estén interconectados, la piedra angular sería el internet. Su impacto en la sociedad no se puede explicar en unas líneas, pero es lo que hace girar este mundo. Las TICs han hecho un arduo trabajo en el campo de las redes. Mejorando la telefonía fija, la telefonía móvil, el propio internet pasando de la conexión telefónica a la banda ancha, después a la fibra óptica y llevando la conexión a los móviles. Permitiendo así que estemos informados al momento.

El otro elemento que conforman las tecnologías de la información y la comunicación, son los servicios. Cada vez es más grande el abanico de servicios que se nos ofrece: correo electrónico, búsqueda de información, banca online, comercio electrónico, e-administración, e-gobierno, servicios privados, servicios de ocio, etc.

Ante el constante aumento -explosivo- en el uso de dispositivos como computadoras personales, Laptops, tabletas y teléfonos celulares, expertos en ciberseguridad advierten un entorno propicio para que prosperen los ciberc riminales y que, tanto individuos como empresas, se encuentran mayormente 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. En la gran mayoría de los casos, el eslabón más débil de la ciberseguridad es el factor humano, que representa la vulnerabilidad más impredecible de cualquier sistema informático. Son las personas las que "caen" en una campaña de Phishing o Whaling, que usan el nombre de la mascota como contraseña. Estas personas son las primeras en abrir las puertas a los ciberdelincuentes.

Al estar cada vez más interconectados 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 (Common Vulnerabilities and Exposures, CVE) registradas por día y estas siguen en aumento año con año. 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.

1.1 Generar Respaldos y Validar su Restauración

A veces, no importa cuán cuidadoso sea uno, existe la posibilidad de perder los datos o que te puedan Hackear[2]. Si ese es el caso, a menudo la única forma en la que puedes recuperar tu información (personal y de trabajo) es restaurar desde un respaldo. Es nuestro menester asegurarnos de realizar respaldos periódicos -preferentemente cifrados (véase sección 1.4) y comprimidos- de cualquier información importante y verificar que podamos restaurarla a partir de ellos. Es posible usar servicios en red como Google Drive y dispositivos USB para guardar en ellos los respaldos.

Debemos garantizar la disponibilidad, integridad y confidencialidad de nuestra información, tanto la que se encuentra en soporte digital, como la que se gestiona en papel. Una buena estrategia de respaldo es la siguiente:

- Mantener tres copias de cualquier fichero importante (una principal y dos respaldos).

[2]El Ransomware se ha convertido en la principal amenaza para la ciberseguridad mundial. Desde que el troyano Wanna Cry 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.

Teniendo en cuenta que un Ransomware típico puede infectar dispositivo móviles, ordenadores personales, servidores o redes, bloqueando su funcionamiento y/o acceso a una 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.

- Mantener los ficheros en dos tipos distintos de almacenamiento para protegerlos ante distintos riesgos.
- Almacenar una copia de seguridad fuera de nuestra casa u oficina.

Las copias de seguridad son nuestra salvaguarda básica para proteger la información. Dependiendo del tamaño y nuestras necesidades, los soportes digitales disponibles, la frecuencia y los procedimientos para realizar las copias de seguridad pueden ser distintos.

El soporte digital escogido dependerá del sistema de copia seleccionado, de la fiabilidad que sea necesaria y de la inversión que deseemos realizar. En la implantación de un sistema de copias debemos tener en cuenta al menos las siguientes consideraciones:

- Analizar la información de la que se va a realizar la copia, así como los sistemas y repositorios donde se encuentra. Debemos tener en cuenta las configuraciones de dispositivos en red, los equipos que dispongamos. Este paso debe permitirnos descartar información que no es necesaria respaldar o ficheros históricos de los que ya existen copias.
- Debemos definir el número de versión que vamos a almacenar de cada elemento guardado, y su periodo de conservación (es lo que se conoce como política de copias de seguridad).

La principal diferencia entre la copia completa y los otros dos tipos de copia (incremental y diferencial) es la información que se almacena en cada iteración del proceso de copia de seguridad[3]:

- Copia total, en este caso se realiza una copia completa y exacta de la información original, independientemente de las copias realizadas anteriormente.
- Copia incremental, en este caso, únicamente se copian los archivos que se hayan añadido o modificado desde la última copia realizada, sea total o incremental.

[3]Una copia de seguridad diferencial no es tan rápida como una incremental, pero es más veloz que una completa; requiere más espacio que una incremental, pero menos que una completa.

- Copias diferenciales, en este caso cada vez que se realiza una copia de seguridad, se copian todos los archivos que hayan sido modificados desde la última copia completa.

En cualquier caso, es necesario hacer pruebas de restauración periódicas, para garantizar que no se producirán problemas en caso de necesitar recuperar nuestra información. Esto es especialmente importante si no se solicitan restauraciones con frecuencia. Los sistemas de copia o los soportes digitales pueden fallar y es fundamental detectarlo antes de que sean necesarios. Además es importante documentar el proceso de respaldo y restauración de copias. Esto permitirá agilizar el proceso de recuperación ante una contingencia.

En caso de que utilicemos almacenamiento en la nube para las copias de seguridad, debemos considerar la posibilidad de que no podamos acceder a la información de manera temporal, por un fallo del servicio o de nuestra conexión a internet.

1.2 Uso de Contraseñas Robustas

Las contraseñas protegen la información que contienen los dispositivos y cuentas de los usuarios. No obstante, ante la cantidad de claves y combinaciones que cotidianamente se deben utilizar, la mayoría de las personas opta por contraseñas fáciles de recordar por la comodidad que esto implica, o bien, por la falta de conocimiento de lo fácil que puede ser para un ciberdelincuente obtenerlas. Para asegurar la efectividad de las contraseñas[4] y evitar el robo de éstas, es recomendable poner en práctica las siguientes acciones:

- Al generar las contraseñas de los dispositivos y cuentas se deben utilizar claves largas -mínimo 15 caracteres[5]- y únicas para cada caso, evitando

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

[5] Solo para tener una idea del tiempo necesario para encontrar la clave de 13 caracteres aleatorios de símbolos, números, letras mayúsculas y minúsculas usando fuerza bruta en un solo core en el que se ejecuten más de mil millones de intentos por segundo se tardaría 9.6 millones años. Si se aumentan el número de cores los tiempos se acortaran de forma drástica, por ejemplo si se usa una supercomputadora que intente 100 millones de contraseñas por segundo bajaría de 9.6 millones a 96 años.
https://i.imgur.com/e3mGIFY.png

utilizar la misma contraseña para diferentes dispositivos o cuentas[6].

- Se deben evitar las combinaciones sencillas como fechas de nacimiento, secuencias consecutivas, repeticiones de un mismo dígito o palabras simples como "password" o "contraseña".

- La mayor longitud de la contraseña, así como la incorporación de mayúsculas, minúsculas, números y símbolos (#$,.-_%&*!?), contribuyen a que ésta sea más segura y difícil de vulnerar[7].

- Se debe evitar escribir contraseñas en papeles o tener archivos con esa información que sean fácilmente accesibles para otros.

- Habilitar el doble factor de autenticación o verificación en dos pasos. Esta medida es una capa adicional de seguridad disponible para cada vez más servicios en la que, además de la contraseña, durante el inicio de sesión se solicita información sobre otro medio al que sólo el usuario autorizado tiene acceso (por ejemplo, verificación para entrar al correo electrónico mediante la recepción de un código vía SMS, llamada o mensaje de WhatsApp).

[6] Según una encuesta de Google y Harris Poll, el 52% de las personas reutiliza las mismas contraseñas para varias cuentas, y el 13% utiliza la misma para acceder a todo. Asegúrese de no formar parte de esas estadísticas. En su lugar, implante aplicaciones de gestión de contraseñas que obliguen a todo el mundo a utilizar un código único y aleatorio para cada cuenta y dispositivo.

[7] GnuPG y OpenSSl son herramientas 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.

Para instalar el paquete GnuPG, usamos:

```
# apt install gnupg
```

para generar clave aleatoria de por ejemplo 32 caracteres, usamos:

```
$ gpg –gen-random –armor 1 32
```

Para instalar el paquete OpenSSL, usamos:

```
# apt install gnupg
```

para generar clave aleatoria de por ejemplo 32 caracteres, usamos:

```
$ openssl rand -base64 32
```

- Es importante no facilitar a nadie, aunque así lo solicite, por ningún medio, contraseñas y/o códigos para el inicio de sesión.
- Es recomendable cambiar con frecuencia las contraseñas a efecto de evitar accesos no autorizados.
- Es recomendable usar un gestor de contraseñas[8] (como KeePassXC, KeePass, Bitwarden, Dashlane, NordPass, LastPass, Enpass, Keeper Password Manager, Password Safe, Password Gorilla, UPM, Buttercup, Myki, Pass, 1Password, etc) para generar y almacenar las claves de acceso estrictamente necesarias y así evitar escribirlas en un papel. Este es un programa de seguridad que almacena de forma segura todas las contraseñas en una caja fuerte virtual cifrada, los hay locales a tu dispositivo o en línea.
- Aprovechar la autenticación multifactor (MFA). Entre las más comunes se encuentra la autenticación en dos factores (2FA o TFA), la cual requiere de dos formas distintas e independientes de identificación para acceder a una cuenta.
- Usar las redes wifi gratuitas con cautela y evitar abrir o compartir datos confidenciales. Desactive el bluetooth o la transferencia de archivos, ya que estas redes suelen ser objeto fácil de los ciberataques.
- Emplear un antivirus de alta calidad para proteger sus dispositivos. Este sistema debe ser capaz de detectar vulnerabilidades y amenazas en el equipo y de bloquear ataques antes de que sucedan.

[8] Un administrador de contraseñas tiene dos funciones principales: (1) almacenar contraseñas, y (2) generar contraseñas seguras y únicas. Esta aplicación es esencialmente como un libro digital que almacena todas sus contraseñas usando una "clave maestra". Al ingresar esta clave, se le otorga acceso al resto de las contraseñas. Por lo tanto, dicha clave o contraseña debe permanecer altamente protegida. Pero su segundo uso es mucho más práctico. El administrador de contraseñas genera automáticamente contraseñas que contienen una combinación compleja de mayúsculas y minúsculas, números, símbolos y caracteres especiales, que puede complicar el descifrado o la detección de la contraseña por parte de piratas informáticos. Al utilizar un administrador de contraseñas, se evita el error común de usar una sola contraseña en las diversas plataformas en línea, evitando así ataques de relleno de credenciales (credential stuffing).

Robo de contraseñas es uno de los problemas de seguridad informática más extendidos y alarmantes. Cuando ocurre, puede dejar daños financieros, tecnológicos o de otro tipo en personas y organizaciones de cualquier nivel. Proteger la información valiosa de los ataques de intrusos o Hackers es una acción que cada día cobra mayor relevancia. Por ese motivo, lo invitamos a seguir leyendo para conocer cómo resguardar sus contraseñas y cuáles son los métodos cada vez más sofisticados que usan los ciberdelincuentes para robarlas.

¿Qué es el robo de contraseñas? es un delito informático en el que se extraen datos que vulneran la privacidad y la seguridad de los usuarios. Para ello, los Hackers roban o descifran las credenciales de acceso a un determinado sistema operativo o plataforma de información.

Luego, los usuarios maliciosos transfieren y almacenan de manera ilegal la información confidencial o financiera de una persona u organización, obteniendo réditos económicos de ello.

¿Cómo los Hackers roban contraseñas? estos son los métodos más utilizados por los Hackers para robar contraseñas y vulnerar los esquemas de Ciberseguridad:

1.- Ataques de fuerza bruta (brute-force attack) a través de este método, los piratas informáticos pueden descifrar las contraseñas débiles. Para hacerlo, utilizan herramientas que les permiten probar credenciales durante un gran número de intentos, hasta coincidir con los datos correctos. Para realizar este ataque, el ciberdelincuente toma en cuenta los requisitos de seguridad (como emplear mayúsculas, minúsculas y números), así como datos personales que los usuarios suelen introducir en sus contraseñas (fecha de nacimiento, por ejemplo).

2.- Phishing (suplantación de identidad) los cibercriminales emplean el Phishing y la ingeniería social para aprovecharse de las vulnerabilidades y brechas. Esto les permite aplicar mecanismos de manipulación psicológica para que las víctimas realicen acciones que aumentan el riesgo.

En estos ciberataques, los hackers emplean acciones como:

- Enviar a personas y organizaciones un correo electrónico con un asunto atractivo, en el cual insertan un archivo adjunto o enlace malicioso

que, de hacer clic, descarga un Software malicioso (Malware) en el dispositivo de la víctima;

- Hacerse pasar por entidades conocidas y de confianza, como un banco, una organización pública o una empresa, a través de llamadas u otras formas de comunicación.

- Compartir páginas de destino (Landing Pages) que solicitan a las personas ingresar sus datos para iniciar sesión. Una vez el sitio Web registra los datos, este pasará a manos de los atacantes.

3.- Keylogger este método consiste en insertar un Malware en un ordenador o dispositivo móvil, permitiendo a los Hackers grabar información personal sin que el usuario pueda darse cuenta. Luego de ser instalado, este Software maligno registra las pulsaciones de teclas que hacen los usuarios. De esta manera, obtienen los nombres de inicio de sesión y contraseñas.

4. Claves genéricas este no es un método tan complejo para los Hackers. Consiste en probar contraseñas genéricas que suelen traer de fábrica los dispositivos. Por defecto, estas contraseñas no son tan difíciles, por lo que la recomendación para evitar este tipo de ataques es personalizar las credenciales una vez empieza a ser utilizado un determinado dispositivo.

5. Vulnerabilidades en las plataformas las propias plataformas muchas veces presentan vulnerabilidades en su seguridad. Los piratas informáticos buscan aprovecharse de ellas para robar nombres de usuarios y contraseñas. Un ejemplo de esto es cuando alguien realiza un registro en una red social y esta plataforma presenta algún problema. Sus datos quedan expuestos en la red, facilitando que un Hacker pueda apropiarse de ellos.

6.- Spyware (Software espía) esta es otra variedad de Software malicioso que puede ser utilizado para robar las contraseñas. Un spyware o software espía es capaz de grabar la pantalla de un dispositivo, recopilando así toda la información que el usuario va mostrando mientras interactúa con el sistema. Además, este software puede captar la información de discos duros y bases de datos para transmitirla a un ente externo.

7.- Diccionario de cuentas frecuentes los piratas informáticos poseen listas con las contraseñas más empleadas por los usuarios a lo largo de los años. Contraseñas como "password", "qwerty", "abc123", "123456" y otras similares son algunas de las más frecuentes. Estas listas usadas por los Hackers son conocidas como "diccionarios de ataque". Básicamente, prueban las contraseñas una por una hasta dar con la correcta. Para prevenir este tipo de ataques, se debe establecer contraseñas comunes y, por el contrario, utilizar otras más complejas.

8.- Ataque de diccionario este tipo de ciberataques es una variante más avanzada. Se basa en recopilar información de la víctima, como:

- Nombres de mascotas o de familiares;
- Fecha de nacimiento;
- Lugares en los que vivió anteriormente.

Para hacerlo, el atacante se vale de un Software que le permite probar todas las palabras de un diccionario basado en la información de la víctima y, en consecuencia, posibles contraseñas. De existir alguna coincidencia, el Hacker tendrá acceso a la cuenta. Estos ataques buscan aprovecharse de una mala práctica, muy frecuente, que es establecer contraseñas de una sola palabra para que sean fáciles de recordar, y que además se vinculan a un dato personal.

9.- Mirar por encima del hombro este es uno de los métodos menos sofisticados y, probablemente, el más antiguo, pero no deja de ser peligroso. Consiste en mirar por encima del hombro del usuario al pasar por delante de su smartphone o computadora, y a través de una mirada a la pantalla y el teclado, descifrar la contraseña. Este tipo de ataques puede ser más frecuente en sitios concurridos, como una biblioteca, cafetería o bar. Si suele ir a este tipo de locales es recomendable que tenga especial cuidado.

10.- Ataque de hombre en el medio (man-in-the-middle) en este ataque, el Hacker intercepta la comunicación entre dos o más personas, suplantando la identidad de uno u otro para tener acceso a la información y modificarla según desee. Luego de que el ciberdelincuente intercepta las

comunicaciones, puede manipular las respuestas recibidas en uno de los extremos y hacerse pasar por el interlocutor legítimo. Con el uso de técnicas de ingeniería social puede enviar archivos adjuntos para instalar un software malicioso en el dispositivo de quien lo recibe.

1.3 Mantener Actualizado el Sistema Operativo y Aplicaciones

Mantener actualizados los sistemas operativos y las aplicaciones de los dispositivos, incluidas las computadoras personales (PC), los teléfonos inteligentes y las tabletas[9]. Estas actualizaciones incluyen cambios importantes que mejoran el rendimiento y la seguridad de los equipos; muchos de estos programas, incluso, se actualizan de manera automática. Además de:

- Cifrar el disco (véase sección 1.4) donde reside el sistema operativo y nuestros datos, sin este cifrado cualquier persona con acceso físico a nuestro equipo de cómputo, tableta, teléfono o dispositivo electrónico puede copiar sus contenidos incluso si no tienen la clave de acceso a ellos.

- Instalar sólo aquellos programas y complementos que realmente uses, porque además de que pueden mermar la velocidad de tu equipo, es posible que estés cediendo más permisos de los que te imaginas.

- Activar funcionalidades de protección, como el cortafuegos (Firewall), incorporadas en los sistemas operativos más comunes. Un cortafuegos es la primera línea de defensa ante un ataque a tu red desde internet y permite proteger el equipo de programas maliciosos o atacantes que intenten conectarse al equipo de forma remota. Además, permite establecer reglas para indicar qué conexiones de red se deben aceptar y cuáles no. Al mismo tiempo, admite el normal intercambio de datos entre la computadora y servicios verificados de internet.

[9] En el caso del sistema operativo Android hay una gran fragmentación de versiones en el mercado coexistiendo, esto debido a que los fabricantes y proveedores tienen un modelo de mercado que prioriza la venta de equipo nuevo y no la seguridad del usuario. Forzando a que las actualizaciones del sistema operativo dependan en gran medida del fabricante del dispositivo o del proveedor del servicio de telefonía, ya que ellos personalizan el sistema operativo, dificultando la actualización de los dispositivos. Lo que ocasiona que la gran mayoría de los dispositivos no reciban las actualizaciones de seguridad y nuevas versiones del sistema operativo publicadas por Android.

- En caso de usar el sistema operativo Windows o Android se debe instalar un antivirus[10], estos programas ayudan a proteger los dispositivos contra la mayoría de los virus, gusanos, troyanos y otros tipos de Malware que pueden infectar a los dispositivos, por ello se recomienda:

  ◦ Instalar y mantener actualizados los antivirus, prefiriendo aquéllos que incorporan funcionalidades de protección contra Malware y cortafuegos (Firewall), también conocidos como "suites de seguridad".

  ◦ Evitar tener dos antivirus en un mismo dispositivo. Tener dos antivirus activos no significa mayor protección; de hecho, puede ocasionar diferentes problemas en el sistema. Un antivirus que esté trabajando se convertirá en un "Software malicioso" a los ojos del otro, el cual intentará bloquearlo y eliminarlo, y se corre el riesgo de afectar el desempeño del sistema por el consumo extra de recursos.

  ◦ Todas las instalaciones y actualizaciones de programas y aplicaciones deben hacerse desde el sitio Web oficial del fabricante o desde las tiendas oficiales de apps -verificando la identidad del autor de la aplicación-, evitando descargar e instalar aquéllas de dudosa procedencia.

  ◦ Deshabilitar la auto ejecución de memorias USB, para evitar que, por ese medio, se instalen programas maliciosos.

  ◦ En los sistemas operativos que lo soporten, habilitar la limpieza remota del dispositivo en caso de pérdida o robo.

[10] Debido a que el Internet es una red abierta, cualquier computadora o dispositivo puede conectarse a este desde cualquier lugar. El uso de software antivirus sirve como un escáner inicial de cualquier actividad sospechosa o maliciosa a la que los usuarios están expuestos a través de las redes sociales. El software antivirus puede ayudar a supervisar la entrega de noticias y puede ofrecer un nivel adicional de protección en el evento que el usuario haga clic erróneamente en enlaces sospechosos que pueden contener Spam y diferentes tipos de virus, como gusanos . Pero tener instalado un software antivirus no es una protección general ya que no puede atrapar todo el malware; el dispositivo aún puede estar infectado. Sin embargo, le agrega una capa de protección que puede ser beneficiosa para el usuario. Por eso es tan importante usar el sentido común también y desconfiar de cualquier mensaje que parezca extraño o sospechoso.

- Nunca inserte ningún dispositivo de almacenamiento externo (USB[11], disco externo, unidad Flash, etc.) que no se hayan revisado exhaustivamente antes de ser usado.
- Al desechar el dispositivo, es necesario reiniciarlo para borrar toda la información que pudiese contener.

1.4 Cifrar Dispositivos, Discos y Unidades de Respaldo

El cifrado de dispositivos, discos, unidades de respaldo -como los dispositivos USB-, datos o cifrado de archivos es un procedimiento mediante el cual los dispositivos, discos, archivos, o cualquier tipo de documento, se vuelven completamente ilegibles gracias a un algoritmo que desordena sus componentes. Así, cualquier persona que no disponga de las claves correctas no podrá acceder a la información que contiene. A menudo nos referimos a los datos cifrados como texto cifrado. El propósito del cifrado es proteger la confidencialidad de los datos digitales.

En el mundo criptográfico, cifrar es un procedimiento que utiliza un algoritmo matemático. Estos algoritmos modifican los datos, de manera que sólo sabiendo el mismo algoritmo se puede descifrar para saber qué es lo que se dice.

Como puede imaginarse, la protección de datos se ha convertido en una preocupación importante para las personas y compañías con el significativo crecimiento del volumen de datos y la sofisticación de la amenaza. A estos hechos, hay que añadir el riesgo que supone cada uno de nosotros lleva consigo grandes cantidades de datos confidenciales en sus ordenadores portátiles y dispositivos móviles.

¿Qué es cifrar? a pesar de los complicados cálculos matemáticos involucrados, la acción de cifrar datos no es difícil de entender. Simplemente,

[11] Se debe tener cuidado con los llamados USB Killer, estos son dispositivos similar a una Unidad Flash USB que envía sobretensiones de alto voltaje al dispositivo al que está conectado, lo que puede dañar los componentes del Hardware. El dispositivo extrae corriente eléctrica del conector eléctrico USB del equipo al que esté conectado, pasándola a sus condensadores, hasta que alcanza un alto voltaje y entonces libera el alto voltaje en los pines de datos. Las versiones 2, 3 y 4 del dispositivo pueden generar un voltaje de 110 a 220 voltios, súcientes para dañar irremediablemente el dispositivo al que se inserte.

consiste en bloquear los datos mediante un código secreto que oculta su verdadero significado. De esta forma, si alguien accede a ellos, encontrará que la información carece de sentido. Y es que, para que los datos cifrados tengan sentido, se necesita la clave del código.

Existen dos tipos de cifrado: simétrico y asimétrico. En términos simples, el cifrado simétrico es la técnica más antigua y conocida. Utiliza una clave secreta para cifrar y descifrar los datos. Tanto el remitente como el receptor conocen la clave.

El cifrado asimétrico es un método más nuevo y también conocido como criptografía de clave pública. Utiliza dos claves en lugar de una: una pública y una privada. Las claves públicas permiten que cualquiera envíe información a otra persona, pero solo cada individuo conoce su clave privada.

Para qué sirve el cifrado de mensajes y otros datos sirve para hacer las comunicaciones más seguras, y lo mismo se puede decir a la hora de aplicarlo a internet. La primera funcionalidad para conseguirlo es la de la confidencialidad de los mensajes, ya que al no ir al descubierto, cuando tú le envías algo a otra persona, los algoritmos criptográficos de la aplicación ayudan a que no se pueda leer fácilmente si alguien lo intercepta en el camino.

Siendo la confidencialidad la primera de las ventajas que ofrece el cifrado de mensajes, la segunda podríamos decir que es la integridad. El encapsular un mensaje dentro de un sobre de cifrado, dicho así para hacerse una mejor imagen mental, ayuda que todo lo que haya cifrado se mantenga correcto y completo.

También hay algoritmos criptográficos que proporcionan mecanismos para verificar la identidad de la persona que envía un mensaje. Además, hay métodos de cifrado que también ayudan a vincular un documento o transacción a una persona o sistema de gestión concretas. Pero a nivel general, para lo que más se suele utilizar es para proteger las comunicaciones. Si tú envías un SMS se envía en texto plano, sin cifrar, y si un operador o alguna agencia interfiere el mensaje, puede leer todo lo que hay escrito. Sin embargo, aplicaciones como WhatsApp, Gmail o Telegram aplican cifrados para que esto no sea tan fácil.

¿Qué se debe cifrar? en términos generales, hay dos tipos de datos que se deben cifrar:

- Información de identificación personal. En este grupo se incluye cual-

quier tipo de información que otra persona pueda usar para identificar a un individuo de manera única. Esto incluye la licencia de conducir, número de seguridad social, etc. Los ladrones pueden usar esta información para robar una identidad, lo que les permite cometer delitos mayores, como solicitar tarjetas de crédito y préstamos a nombre de otra persona. Combatir esta clase de ataques requiere esfuerzos en muchos frentes. La información de identificación personal reside en los teléfonos, tabletas y computadoras portátiles , por lo que esos dispositivos y su almacenamiento se deben cifrar.

- Información confidencial de Negocios y Propiedad Intelectual. Los datos a los que los empleados acceden cada día acerca de los clientes, los planes para un nuevo producto o los datos sobre la próxima campaña de marketing entrarían en este grupo. De toda esta información podrían sacar provecho los competidores y, por tanto, puede convertirse en objetivo de piratas informáticos.

Ante las dificultades para decidir si se deben o no cifrar algunos datos, basta con preguntarse si se destruirían antes de tirarlos a la basura, caso de estar en formato papel o si, en el caso de que se filtraran por accidente causarían daño a los empleados o clientes. En ambos casos, si la respuesta es afirmativa, hay que cifrar.

Cifrar Discos Los discos duros externos y los dispositivos USB son la manera perfecta de poder llevarnos en nuestros viajes todos nuestros archivos importantes. Pero cuando estos archivos son personales y privados es posible que no quieras que nadie pueda acceder a ellos en caso de que se te pierda el USB o te lo quiten de alguna manera.

Por eso, todos los sistemas operativos suelen darte la opción de aplicarles un cifrado y protegerlos con una contraseña de tu elección, un proceso diferente al de protegerlos contra escritura. Vamos a ver como hacerlo paso a paso con Windows con una de sus herramientas nativas, y también te diremos brevemente cómo hacer lo mismo en MacOS, Android y GNU/Linux pero de una forma aún más sencilla.

Cifrar en Windows la manera de cifrar unidades externas en Windows tiene un nombre: BitLocker. Se trata de una herramienta desarrollada por Microsoft que suele venir preinstalada en casi todas las versiones de su sistema

operativo, pero que de no ser así puedes conseguir en su página de descarga, donde se pueden descargar sus versiones de 32 y 64 bits.

Cifrar en MacOS en el caso de que tengas un Mac el proceso es aún más fácil, ya que lo único que tienes que hacer es tener la unidad formateada, conectarla al equipo y dar Click derecho sobre su icono cuando aparezca en el escritorio. En el menú desplegable sólo tendrás que elegir la opción: "Cifrar" para introducir con qué contraseñas lo quieres bloquear.

Cifrar en GNU/Linux en las distribuciones GNU/Linux por su parte también basta con dar Click derecho sobre la unidad, sólo que en este caso la opción a elegir es: "Formatear volumen". La clave aquí la tienes en el tipo de de formateo, ya que en él tendrás la opción Cifrado -LUKS o Cryptsetup-. De esta manera, formatearás el disco duro aplicándole directamente un cifrado con una contraseña que elijas.

Cifrar en Android siguiendo estos pasos podemos cifrar el disco:

1.- Ve a: "Ajustes".

2.- Selecciona: "Seguridad".

3.- Pulsa en: "Cifrar teléfono" y configura una contraseña (véase sección 1.2).

4.- Espera tranquilamente a que el proceso acabe, puede tardar más de media hora fácilmente.

Es importante destacar que la única forma de quitar el cifrado es reseteando de fábrica el dispositivo, lo que significa el borrado total del contenido que en él haya almacenado. Si tienes una tarjeta microSD insertada, los datos dentro también serán cifrados y no podrás acceder a ellos en otro dispositivo.

Cifrar Archivos

En Windows la herramienta que trae Windows preinstalada (ediciones Education, Pro y Enterprise) es un buen comienzo para cifrar nuestros datos.

Para ello, deberemos seguir los siguientes pasos:

1. Haz Clic con el botón derecho en un archivo o carpeta (o mantenlo presionado) y selecciona: "Propiedades".
2. Selecciona el botón: "Avanzados" y haz Clic en la casilla de verificación: "Cifrar contenido" para proteger datos.
3. Pulsa el botón: "Aceptar" para cerrar la ventana: "Atributos avanzados" y a continuación, selecciona el botón: "Aplicar" y después: "Aceptar".

Una vez cifrada la información, solo podremos acceder si disponemos de la clave de cifrado correcta.

En GNU/Linux y MacOS es posible cifrar archivos usando GnuPG, esta es una herramienta en línea de comandos de seguridad en comunicaciones electrónicas en donde se utiliza criptografía simétrica y de clave pública para que los usuarios puedan comunicarse de un modo seguro[12]. 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
```

Es posible cifrar archivos usando sólo una clave (véase sección 1.2) -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. 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
```

[12] Otras opciones son: ccrypt y mcrypt.

Para cifrar una carpeta, podemos usar:

```
$ tar -cvf archivo.tar directorio
$ gpg -c archivo.tar
```

o podemos usar el comando zip para compactar y cifrar, usando:

```
$ zip –encrypt archivo.zip archivo1 archivo2
$ zip -e archivo.zip archivo1 archivo2
$ zip -r –encrypt archivo.zip directorio
```

y podemos descifrar usando:

```
$ unzip archivo.zip
```

1.5 Cómo Tomar, Enviar y Almacenar Contenido Íntimo

Cuando gran parte de nuestra comunicación ocurre en línea, sextear, video grabar, grabar audio, fotografiar o elaborar videos reales o simulados de contenido íntimo es tan saludable y natural como tener relaciones sexuales según dicen los expertos. Poder intercambiar instantáneamente dicho material con alguien sin importar la distancia puede ser muy divertido, pero la facilidad puede hacer que ignores las posibles complicaciones.

Al igual que tener relaciones sexuales, enviar contenido íntimo puede tener consecuencias no deseadas de por vida con las que quizás no estés dispuesto a lidiar[13]. Pero podemos minimizar fácilmente los riesgos y protegernos si estamos seguros.

[13] En México existe la "Ley Olimpia", está no se refiere a una ley como tal, sino a un conjunto de reformas legislativas encaminadas a reconocer la violencia digital y sancionar los delitos que violen la intimidad sexual de las personas a través de medios digitales, también conocida como ciberviolencia.

Las siguientes son conductas que atentan contra la intimidad sexual:

- Video grabar, grabar audio, fotografiar o elaborar vídeos reales o simulados de contenido sexual íntimo, de una persona sin su consentimiento o mediante engaño.
- Exponer, distribuir, difundir, exhibir, reproducir, transmitir, comercializar, ofertar, intercambiar y compartir imágenes, audios o videos de contenido sexual íntimo de una persona, a sabiendas de que no existe consentimiento, mediante materiales impresos, correo electrónico, mensajes telefónicos, redes sociales o cualquier medio tecnológico.

Envío de Contenido Sexual Íntimo Repasemos lo básico. No existe contenido íntimo completamente seguro. Simplemente vamos a seguir adelante y decirlo: una vez que generemos el contenido íntimo, ya sea que presionemos el botón Enviar o no, perderemos el control total de dicho material.

Siempre existe la posibilidad de que se tenga acceso al medio con el cual generamos y/o almacenamos el contenido sexual íntimo y un tercero pueda exponer, distribuir, difundir, exhibir, reproducir, transmitir, comercializar, ofertar, intercambiar y compartir imágenes, audios o videos de contenido se-xual almacenado en nuestro dispositivo y a sabiendas de que no existe consentimiento, mediante materiales impresos, correo electrónico, mensajes telefónicos, redes sociales o cualquier medio tecnológico sea difundido.

La tecnología no puede arreglar a los humanos que no son de confianza, pero puede ayudarlo a expresar sus límites y hacer que sea un poco más difícil violarlos. Claro, algunas aplicaciones te notificarán cuando alguien tome una captura de pantalla de tu imagen, pero en realidad no evitarán que lo hagan. Un destinatario también puede simplemente usar otro dispositivo para tomar una foto de la pantalla sin alertarlo.

¿Usted o su pareja son menores de edad? No tome, envíe, reciba, comparta o almacene contenido sexual íntimo. No hay matices sobre esto: el contenido sexual íntimo de menores son pornografía infantil, y su producción, almacenamiento y distribución está en contra de la ley. Incluso si un intercambio de imágenes fue consensuado con entusiasmo, y usted fue quien se tomó contenido sexual íntimo, muchos países aún podrían considerarlo un delito grave solo porque es menor de edad.

Cumplir con la Etiqueta Básica de Sext Esto puede ser evidente, pero existen reglas básicas de decencia cuando se trata de enviar contenido sexual íntimo.

- Si recibe uno, no lo comparta con nadie más, es para usted y sólo para usted. Cuando comparte contenido sexual íntimo, no solo está violando la confianza de la persona que la envió, sino que también hace que sea

Por su parte, se entiende como violencia digital aquellas acciones en las que se expongan, difundan o reproduzcan imágenes, audios o videos de contenido sexual íntimo de una persona sin su consentimiento, a través de medios tecnológicos y que por su naturaleza atentan contra la integridad, la dignidad y la vida privada de las mujeres causando daño psicológico, económico o sexual tanto en el ámbito privado como en el público, además de daño moral, tanto a ellas como a sus familias.

más probable que termine en las manos equivocadas, o peor aún, en Internet. En muchos países, compartir contenido sexual íntimo sin su consentimiento también puede ser un delito grave. Simplemente no lo hagas.

- No publiques contenido sexual íntimo de otra persona en línea.
- Como con todas las cosas sexys, el consentimiento es clave. Respétalo. No envíe contenido sexual íntimo no solicitado, especialmente a personas que no conoce. Si estás en una relación con alguien, no importa cuán casual sea, ten una conversación sobre cómo se siente acerca del contenido sexual íntimo no solicitado. Algunos pueden recibir contenido sexual íntimo en medio de su jornada laboral, mientras que otros no. Habla con tu pareja sobre sus gustos y límites, y hónralos.

En Caso de Duda, Abstenerse Envía contenido sexual íntimo sólo a aquellos que conoces y en los que confías. Esto excluye a las personas con las que has coincidido en aplicaciones de citas pero que nunca conociste, los contactos en línea que ni siquiera estás seguro de que sean reales o las personas que te dan la más mínima pista de que no son confiables.

El mayor riesgo de enviar contenido sexual íntimo es que la persona al otro lado del dispositivo es menos confiable de lo que crees. O que son dignos de confianza hoy pero lo serán menos en el futuro, después de una ruptura, por ejemplo. Saber en quién confiar está lejos de ser una ciencia perfecta, pero solo hacerte la pregunta antes de enviarles contenido sexual íntimo podría ahorrarte algunos problemas.

El perder o dar acceso al dispositivo donde almacenamos contenido sexual íntimo sin la adecuada protección, puede poner nuestro contenido sexual íntimo a disposición de terceros que no dudarán en compartir dicho material. Sí, esto suena aterrador, pero no significa dejar de generar y/o enviar contenido sexual íntimo por completo. En cambio, debemos concentrarnos en administrar las cosas que podemos controlar.

Cómo Tomar Contenido Sexual Íntimo de Forma Segura La mejor manera de mantener seguro el contenido sexual íntimo es mantenerla en el anonimato. De esa manera, incluso si su material termina en línea, será difícil identificarlo.

- Recorta o Cubre tu Cara

- Ir sin rostro es la base del anonimato. Si esto no funciona con tu visión artística, sé creativo y opta por otras formas de hacerte inidentificable. Utilice ángulos nítidos para mantener su rostro fuera del marco o envuelto en sombras, o dispare con un flash brillante en un espejo, por ejemplo.
- Si usa una máscara, asegúrese de que cubra lo suficiente de su cara.
- Oculte cualquier tatuaje, marca de nacimiento y marcas de belleza
- Use el encuadre y los ángulos de la cámara para mantener los identificadores únicos ocultos a la vista. Si una gran parte de su cuerpo está cubierta de tinta, considere usar accesorios para cubrirlo. Una vez más, la creatividad es clave: una prenda de vestir, una bufanda o una cortina pueden ser útiles.

Si necesita consejos, consulte el contenido de fotografía de Boudoir en YouTube y TikTok. Esto no solo te ayudará a perfeccionar las poses y los ángulos que mostrarán tu hermoso cuerpo, sino que también aprenderás cómo hacer que tu entorno funcione a tu favor.

Considere su Entorno Si alguien echara un vistazo dentro de tu habitación, probablemente descubriría muchas cosas sobre ti. No dejes que tu estilo único te desanime. Encuentre cualquier cosa que pueda revelar información personal sobre usted y asegúrese de que no esté en el marco. Esto incluye fotos, diplomas y notas adhesivas.

Tenga en cuenta que algo no tiene que tener sus datos personales para ser revelador. Las personas que han estado en tu casa podrían identificarla fácilmente por un cartel de una banda o una pintura en tu pared.

Al elegir un escenario para tu contenido sexual íntimo, mantenlo lo más sencillo posible. Las paredes en blanco y los azulejos del baño anodinos son fondos perfectos para tomas sexys.

Finalmente, manténgase alejado de ventanas grandes y abiertas. Los puntos de referencia conocidos podrían asomarse y ser suficientes para rastrear el contenido sexual íntimo hasta su hogar y hasta usted. Además, es posible que desee mantener a los vecinos fuera de su sesión de fotos, a menos que esté interesado en eso.

Haz Siempre Algo de Postproducción. Cuando se trata de contenido sexual íntimo, las herramientas de recorte y corrección incluidas en la mayoría de los programas de edición de vídeos y fotos son tus mejores amigos. La herramienta de recorte (un icono que parece dos ángulos rectos superpuestos) te permitirá cambiar el marco de la imagen, recortando todo lo que no quieras que aparezca. Esta podría ser tu cara o ese cesto desbordante en la esquina de tu habitación.

La herramienta de curación (un ícono que parece un vendaje) lo ayudará a desenfocar la información en el fondo, junto con pequeños tatuajes, marcas de nacimiento, marcas de belleza, imperfecciones y cualquier otra cosa que desee eliminar con el aerógrafo.

La descarga de aplicaciones como Snapseed (gratis para Android e iOS) o Photoshop Express (gratis para Android e iOS) en su teléfono o computadora lo ayudará a modificar todas las cosas que podría haber olvidado mientras tomaba la fotografía. También le proporcionarán una amplia biblioteca de filtros para que se vea aún más como un bocadillo.

Desactivar los Servicios de Ubicación Cada foto que toma tiene metadatos[14] adjuntos, incluida la cámara que usó, el sistema operativo que ejecuta su dispositivo y, lo adivinó, su ubicación en el momento en que presionó el botón del obturador. Incluso si su rostro no se muestra, alguien podría usar esos metadatos (véase sección 1.6) para confirmar su identidad a través de su ubicación o dirección.

Si está utilizando un dispositivo móvil, apague los servicios de ubicación

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

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

Para visualizar los datos Exif, usamos:

```
$ exiftool imagen.gif
```

Para borrar todos los datos Exif, usamos:

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

antes de tomar la foto. En Android, deslícese hacia abajo desde la parte superior de la pantalla para abrir el menú de configuración rápida y toque el icono de ubicación para desactivar la señal de GPS. Además, abra la aplicación de su cámara y toque el icono del engranaje para acceder a su configuración. Una vez que esté allí, toque el interruptor junto a Guardar ubicación para evitar que la aplicación agregue su paradero a sus metadatos.

En iOS, vaya a Configuración, luego a Privacidad y seleccione Servicios de ubicación. Allí, busque la aplicación de la cámara y, en Permitir acceso a la ubicación, elija Nunca.

Si olvidó hacer esto, puede eliminar los metadatos de ubicación de su imagen más tarde usando macOS. Abra la foto usando Vista previa y presione el comando + I, o vaya a Herramientas y haga clic en Mostrar inspector, que le mostrará toda la información adjunta a su archivo. En la pestaña Más información (segunda a la derecha), elija la pestaña GPS (tercera a la derecha). Luego, en la parte inferior del cuadro de diálogo, haga clic en Eliminar información de ubicación. La pestaña GPS debería desaparecer.

Puedes hacer lo mismo en Windows. Haga clic derecho en uno o más archivos, seleccione Propiedades y vaya a Detalles. En la parte inferior del cuadro de diálogo, haga clic en Eliminar propiedades e información personal y luego marque la casilla junto a Eliminar las siguientes propiedades de este archivo. En la parte inferior, haga clic en Seleccionar todo y luego presione Aceptar. Esto borrará todos los metadatos de los archivos seleccionados.

Desactive la Sincronización Automática con sus Servicios en la Nube Enviar fotos directamente desde el carrete de su cámara a su espacio personal en la nube es útil, pero es una responsabilidad cuando se trata de contenido sexual íntimo.

Antes de tomar su contenido sexual íntimo, asegúrese de desactivar la sincronización entre su dispositivo y todos los servicios en la nube conectados. En Android, abre la aplicación Google Photos, toca tu avatar (arriba a la derecha) y luego Copia de seguridad. Una vez allí, apague el interruptor junto a Copia de seguridad y sincronización. En iOS, desactive las fotos de iCloud yendo a Configuración, tocando su nombre, eligiendo iCloud, luego Fotos y desactivando el interruptor junto a Fotos de iCloud.

Asegúrate de eliminar tus vídeos, fotos del carrete de la cámara y de la papelera, o muévelas a una carpeta segura antes de volver a activar la sincronización.

Cómo Enviar Contenido Sexual Íntimo de Forma Segura Ahora es el momento de entregar tu contenido sexual íntimo y sacudir el mundo de tu pareja. Elige una plataforma segura: Cualquier cosa que no tenga encriptación de extremo a extremo (E2E), que protege su contenido de la intercepción en su camino hacia el destinatario y evita que la empresa propietaria de la plataforma acceda a él, está fuera de discusión. Esto significa que no hay Facebook Messenger o Instagram. Snapchat usa encriptación E2E en fotos y videos, pero no en mensajes, y aunque te permite saber cuándo alguien tomó una captura de pantalla de tu foto, no evita que lo haga.

Tu apuesta más segura es Signal. Está encriptado E2E, puede hacer que los mensajes desaparezcan un mínimo de cinco segundos después de verlos y los chats seguros evitan que los usuarios tomen capturas de pantalla. Vale la pena señalar que esto no impide que alguien tome una foto de la pantalla, pero en lo que respecta a las aplicaciones de mensajería tradicionales, Signal puede ser su mejor opción.

Si está dispuesto a gastar dinero en su privacidad, Disckreet es una aplicación de mensajería diseñada para compartir textos e imágenes subidas de tono. Disponible para iOS y Android, esta plataforma está encriptada E2E, protegida por un código de acceso y brinda a los usuarios control unilateral sobre su contenido. Esto significa que usted decide cuándo su pareja puede ver una foto que envió y puede eliminar la imagen de forma remota desde su teléfono. La versión gratuita de Diskreet limita el tamaño y la cantidad de archivos que puede compartir en un día, pero puede suscribirse por $ 1 al mes para compartir sin restricciones.

Obtenga Toda la Ayuda que Pueda Si está atascado usando una aplicación poco segura, asegúrese de activar todas las funciones que pueden dificultar la descarga o la captura de pantalla de sus fotos en el otro extremo. Una vez que haya terminado de sextear, no olvide pedirle explícitamente a su pareja que elimine sus fotos.

Incluso si no lo hacen, esto dejará en claro que cuando compartió esas imágenes, estaban destinadas solo para los ojos de su pareja. Si los comparten o los publican en línea, legalmente constituye una violación de su privacidad.

Ejercita Algunos Buenos Elementos Esenciales de Ciberseguridad Asegúrate de estar seguro en línea en general. Comience por proteger todas sus cuentas y dispositivos con contraseñas, patrones, PIN, códigos de

acceso o datos biométricos únicos y seguros (véase sección 1.2). Si hacer un seguimiento de toda esa información es demasiado difícil para usted, descargue un administrador de contraseñas y no olvide habilitar la autenticación de dos factores en todas sus cuentas.

Ya deberías estar haciendo todo esto, pero es especialmente importante si estás intercambiando contenido sexual íntimo. Desea que sea lo más difícil posible para cualquier persona acceder a su contenido sexy al ingresar a sus cuentas o dispositivos (véase sección 1.4).

Cómo Almacenar de Forma Segura Contenido Sexual Íntimo Lo que haga después de enviar contenido sexual íntimo dependerá de si desea eliminarlo o agregarlo a su archivo personal. Si puede, elimine el contenido sexual íntimo en la plataforma que utilizó para enviarla para que ni usted ni el destinatario tengan acceso a ella. Si no desea dejar absolutamente ningún rastro, también debe eliminar el archivo en su dispositivo.

Pero tal vez hiciste contenido sexual íntimo muy bueno y no quieres que se pierda en el olvido. Aquí es cuando necesita asegurar su archivo. Los servicios de almacenamiento en la nube son susceptibles a piratería y fugas de datos, por lo que es posible que desee almacenar su contenido sexual íntimo localmente (véase sección 1.4).

La forma más fácil es mover su contenido sexual íntimo a una carpeta protegida con contraseña en su dispositivo. En Android, vaya a la aplicación Archivos y luego a Imágenes e imágenes. Seleccione su contenido sexual íntimo presionándolo prolongadamente, toque los tres puntos en la esquina superior derecha de la pantalla y elija Mover a la carpeta segura. Para acceder a esa carpeta, deberá proporcionar un patrón de seguridad, un código de acceso o una función biométrica, que puede ser la misma que usa para desbloquear su teléfono o algo completamente diferente. También puede ocultar la carpeta, de modo que si alguien entrara en su dispositivo, no podría verlo ni buscarlo.

Windows 10 y 11 tiene una característica similar que le permite usar el Explorador de archivos para proteger cualquier archivo o carpeta con una contraseña. Simplemente haga clic con el botón derecho en el elemento, vaya a Propiedades y luego a Avanzado. Haga clic en Cifrar contenido para proteger los datos en la parte inferior del cuadro de diálogo y haga clic en Aceptar y luego en Aplicar. En el siguiente cuadro de diálogo, elija si desea cifrar solo el archivo o toda la carpeta donde se encuentra, luego haga clic

en Aceptar.

El sistema operativo de la computadora de Apple también le permite crear carpetas protegidas con contraseña. Guarde su contenido sexual íntimo dentro de una carpeta, abra Disc Utility y vaya a Archivo, Nueva imagen e Imagen de la carpeta... Luego, use la ventana emergente del Finder para encontrar la carpeta de su interés y haga clic en Elegir. En Cifrado, elija su protocolo, luego ingrese y verifique su contraseña. Finalmente, en Formato de imagen, elija leer/escribir y presione Guardar. También puede proteger archivos individuales en Vista previa, siempre que se guarden como archivos PDF, y use la aplicación Notas para crear archivos protegidos con fotos incrustadas.

No hay una solución integrada para iOS, pero puede descargar una aplicación gratuita de bloqueo de archivos que hará el mismo trabajo que la "carpeta segura" de Android en su iPhone.

Otra alternativa es mover su contenido sexual íntimo a un disco duro externo, que puede cifrar y almacenar en un lugar seguro (véase sección 1.4).

El Contenido Sexual Íntimo Seguros son un Esfuerzo de Equipo Nada de lo que haga para garantizar que el contenido sexual íntimo seguro sea realmente seguro si el destinatario no se molesta en configurar un código de acceso para bloquear su dispositivo.

Si su pareja no tiene conocimientos tecnológicos, tómese el tiempo para enseñarle lo que debe hacer para protegerlo a usted y a ellos mismos aplicando algunos elementos esenciales de ciberseguridad (véase sección 1.2 y 1.4).

1.6 Protegiendo Nuestros Metadatos

¿Qué son los metadatos? los metadatos son "datos sobre datos" o "información sobre información" que se incluyen en archivos informáticos, normalmente de forma automática. Los metadatos se utilizan para describir, identificar, categorizar y ordenar archivos. Sin embargo, los metadatos también se pueden utilizar para desanonimizar a los usuarios y exponer información privada.

Ejemplos de metadatos incluyen:

- En archivos de imagen y vídeo:

  El lugar donde se tomó la foto o vídeo.

La fecha y hora en que se tomó la foto o vídeo.

El modelo y número de serie de la cámara utilizada.

- En archivos de documentos de texto:

  El autor del documento.

  Cambios en el documento.

Algunos tipos de archivos que guardan metadatos que pueden poner en peligro el anonimato de los usuarios:

- Audio Interchange File Format (.aiff)
- Audio Video Interleave (.avi)
- Electronic Publication (.epub)
- Free Lossless Audio Codec (.flac)
- Graphics Interchange Format (.gif)
- High Efficiency Image Format (.heic, .heif)
- Hypertext Markup Language (.html, .xhtml)
- Portable Network Graphics (PNG)
- JPEG (.jpeg, .jpg, ...)
- MPEG Audio (.mp3, .mp2, .mp1, .mpa)
- MPEG-4 (.mp4)
- Office Openxml (.docx, .pptx, .xlsx, ...)
- Ogg Vorbis (.ogg)
- Open Document (.odt, .odx, .ods, ...)
- Portable Document Fileformat (.pdf)
- Portable Pixmap Format (.ppm)

- Scalable Vector Graphics (.svg)
- Tape ARchive (.tar, .tar.bz2, .tar.gz, .tar.zx)
- Torrent (.torrent)
- Waveform Audio (.wav)
- Windows Media Video (.wmv)
- ZIP (.zip)

Es imposible encontrar y eliminar de manera confiable todos los metadatos en formatos de archivos complejos. Por ejemplo, los documentos de Microsoft Office pueden contener imágenes incrustadas, audio y otros archivos que contienen sus propios metadatos que no se pueden eliminar. Por ello se debe eliminar los metadatos de cualquier archivo antes de incrustarlo en otro documento.

Además, siempre que sea posible, debes guardar los archivos en formatos más simples. Por ejemplo, en lugar de guardar un documento de texto como un archivo .docx, puede guardarlo como un archivo .txt simple.

Metadata Anonymisation toolkit v2 permite eliminar metadatos de archivos antes de publicarlos o compartirlos, funciona en muchos formatos de archivos, incluidos:

- Archivos de imagen, como .jpeg, .png y .gif
- Archivos de LibreOffice, como .odt y .ods
- Documentos de Microsoft Office, como .docx, .xlsx y .pptx
- Archivos de audio, como .mp3, .flac y .ogg
- Archivos de vídeo, como .mp4 y .avi
- Archivar archivos, como .zip y .tar

En GNU/Linux podemos instalar el paquete mat2, mediante:

```
# apt install mat2
```

Para conocer la lista de archivos soportados usamos:

```
$ mat2 -l
```

Para visualizar los metadatos del archivo usamos:

```
$ mat2 -s nombreArchivo
```

Para remover los metadatos del archivo usamos:

```
$ mat2 -V nombreArchivo
```

en la hipótesis de eliminación de metadatos, debes tener en cuenta que mat2 no elimina el archivo en el que interviene porque deja el archivo original como está, sino que crea otro archivo.

Exchangeable Image File al tomar fotografías o vídeos es recomendable por seguridad desactivar el guardado de datos Exif (Exchangeable Image File) también conocidos como metadatos, ya que estos contienen información sobre la cámara, sobre la fotografía y sobre su origen como ubicación por GPS, la hora de creación, la última modificación, etc.

En GNU/Linux podemos instalar el paquete ExifTool que permite conocer y borrar los datos Exif en fotografías. Para instalarlo usamos:

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

Para visualizar los datos Exif, usamos:

```
$ exiftool imagen.gif
```

Para borrar todos los datos Exif, usamos:

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

1.7 Protección de Teléfonos, Tabletas y Computadoras

Estos dispositivos están omnipresentes en nuestra vida, además pueden almacenar mucha más información de la que nos imaginamos y muchos de nosotros los usamos para acceder a casi toda nuestra información digital desde ellos. Por tanto, necesitan protegerse activamente para evitar que el contenido que almacenan y al que tienen acceso termine en manos de terceras personas. En ellos se guardan todo tipo de datos personales, mensajes, imágenes y vídeos en los que aparecen los usuarios, amigos y otras personas. A veces incluso se puede haber recibido o grabado contenido íntimo o de connotación sexual[15].

Hay que hacer notar que varias empresas tecnológicas (en conjunción con las leyes de diversos países) en su lucha contra el terrorismo y por la protección de los jóvenes han anunciado nuevas funciones a su Software para la revisión de mensajes y escaneo de imágenes almacenados y enviados en busca de material de abuso y explotación sexual infantil frente a los extraños que utilizan herramientas de comunicación para reclutarlos y explotarlos. Si bien los objetivos son loables y no cabe duda que no existen respuestas fáciles cuando se trata de poner en peligro a los niños, pero rompe la promesa de cifrado de dispositivos y de mensajes de extremo a extremo, inevitablemente se abre la puerta a otros daños colaterales.

No olvidemos que cualquier método de almacenaje de información digital puede sufrir filtraciones o ataques informáticos, pero los dispositivos móviles son aún más vulnerables. Pueden extraviarse, también pueden ser robados en diversas situaciones o, simplemente, ser interceptados en un momento de despiste. Por ello estos son algunos de los hábitos que debemos practicar a diario para mejorar la protección del dispositivo y nuestros datos:

- Todos los dispositivos móviles y tarjetas de respaldo extraíbles deben ser cifrados y se debe usar una contraseña robusta para el acceso al mismo (véase sección 1.2), es decir, utilizar una combinación compleja -de al menos 15 caracteres- de letras mayúsculas, minúsculas, números y símbolos es siempre la mejor opción. Además, es imprescindible no

[15] En el caso de los adolescentes no siempre son conscientes del riesgo que supone producirlo o guardarlo, Pero, en cualquier caso, requiere una protección especial, y como adultos podemos mostrarles la importancia de proteger estos contenidos.

Este tipo de contenido es extremadamente sensible y, de hecho, cuando se trata de imágenes o vídeos en los que aparecen menores de edad ni siquiera es legal almacenarlo en ningún tipo de formato.

confiarla a sus amistades y cambiarla siempre que tengan dudas de su fiabilidad.

- Es fundamental que se comprenda que toda la información que guardan en el dispositivo es susceptible de caer en manos de terceras personas. Por eso, lo ideal es que no guarden imágenes, vídeos o datos de carácter sensible. Es preferible que revisen periódicamente sus archivos, eliminen de forma eficaz aquello de lo que puedan prescindir y, en el caso de querer guardar algunos contenidos, los almacenen en otros dispositivos más seguros, como un disco duro externo cifrado (véase sección 1.4) que mantengan en casa. Cuanta menos información lleven en su dispositivo, menor riesgo.

- Mantener siempre actualizados nuestros dispositivos y en caso necesario usar alguna aplicación para seguridad y protección de dispositivos móviles.

- Cualquier dispositivo ofrece un sistema de bloqueo para que nadie que no sea el usuario pueda acceder a él. En el caso de móviles y tabletas, tienen diferentes alternativas, como usar su huella dactilar, un sistema de reconocimiento facial, un código PIN, una contraseña o un patrón. Cualquier opción es buena, pero hay que tener en cuenta que un patrón o un código PIN siempre son más fáciles de observar y memorizar por otras personas. Obviamente, esta función pierde su eficacia si no se acostumbran a bloquear el dispositivo siempre que no lo estén utilizando.

- Algunos servicios de internet, como cuentas de correo o redes sociales, ofrecen un sistema combinado de autenticación. De esta forma, además de solicitar la contraseña para acceder, utilizan un segundo paso para comprobar la identidad del usuario. Puede ser un código que llegue a su teléfono móvil, por ejemplo, y es muy útil contra el robo de contraseñas. Para ponernos en situación, si alguien averigua la clave de acceso a su cuenta de Instagram no podrá acceder porque también necesitará disponer del dispositivo para ver el código de verificación.

- Siempre es una buena costumbre cerrar la sesión de cada servicio al concluir su uso. Por ejemplo, si se entra en redes sociales o en su cuenta de correo en el ordenador público, es fundamental que cierren la

sesión y limpiar el Cache antes de irse, comprobando que su contraseña no ha quedado guardada.

- Hoy en día todos los servicios digitales incorporan opciones de seguridad y privacidad que se han diseñado para proteger nuestra información. Una buena práctica es revisar la configuración de cada aplicación o servicio que se utilice, personalizando todas aquellas funciones que les sean de utilidad. Así, se pueden establecer cuentas de redes sociales como privadas o revisar los permisos que otorgan a cada aplicación.

- Existen muchas más funciones que pueden ser útiles para salvaguardar todas esos datos, imágenes y vídeos que se guardan en los dispositivos. Todo es cuestión de investigar y buscar soluciones siempre que sientan que su información es vulnerable. Por ejemplo, pueden bloquear la galería de su móvil o el explorador de archivos, de forma que sea necesario utilizar su huella para acceder. También pueden crear listas de amistades en las redes sociales para que las publicaciones sean más personales y no lleguen a todos los seguidores.

- Activar la función de búsqueda y bloqueo de dispositivo frente a pérdidas o robos. Cada vez hay más opciones de seguridad disponibles, solo hay que motivar su uso en la prevención de riesgos.

- Activar el Wi-Fi y Bluetooth sólo cuando se usen, de esta forma se minimiza la posibilidad que un usuario malintencionado intente conectarse a nuestro dispositivo.

- Es posible poner un código de acceso en su tarjeta SIM, esto puede protegerla de la suplantación de SIM. La configuración de este código se puede hacer en un iPhone yendo a Configuración > Celular > PIN de SIM. Ingrese su PIN existente para habilitar el bloqueo. Los usuarios de Android pueden ir a Configuración > Seguridad > Bloqueo de tarjeta SIM. Aquí puede habilitar la opción para bloquear su SIM.

- Se debe evitar el uso de puertos de carga USB no confiables, como los de energía públicos, ya que ellos se pueden Hackear, y el ciberdelincuente puede instalar, grabar datos, tomar video o sacar datos de nuestros dispositivos móviles. 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.

- Al desechar el dispositivo, es necesario reiniciarlo para borrar toda la información que pudiese contener, además se debe retirar la tarjeta SIM y destruirse.

Prácticas de Ciberseguridad para Viajeros Los avances tecnológicos han revolucionado nuestra forma de viajar, ofreciéndonos comodidad y conectividad al alcance de la mano. Sin embargo, estas mejoras tecnológicas también han hecho que los viajeros sean cada vez más vulnerables a las ciberamenazas.

Es crucial dar prioridad a las prácticas de seguridad, tanto dentro como fuera del hogar u oficina. He aquí algunas estrategias clave para garantizar la seguridad de los dispositivos y los datos, independientemente de su ubicación:

- Todos los dispositivos móviles y tarjetas de respaldo extraíbles deben ser cifrados (véase sección 1.4)y se debe usar una contraseña robusta para el acceso al mismo (véase sección 1.2), es decir, utilizar una combinación compleja -de al menos 15 caracteres- de letras mayúsculas, minúsculas, números y símbolos es siempre la mejor opción. Además, es imprescindible no confiarla a sus amistades y cambiarla siempre que tengan dudas de su fiabilidad.

- Actualizar regularmente los dispositivos con los últimos parches de seguridad y Firmware es crucial para protegerse de las vulnerabilidades conocidas (véase sección 1.3). Los proveedores publican constantemente estas actualizaciones para corregir errores y lagunas de seguridad y mantener a los usuarios a salvo de las ciberamenazas. Es importante mantenerse alerta y asegurarse de que todos los dispositivos estén siempre actualizados. Active las actualizaciones automáticas siempre que sea posible para asegurarse de que los parches de seguridad críticos se instalan rápidamente.

- Cuando se viaje se pueden tener la tentación de publicar sus experiencias en línea con amigos y familiares. Sin embargo, hay que tener cuidado al revelar públicamente información sensible. Recuerde a todos sus compañeros de viaje que eviten compartir itinerarios, detalles del hotel o información de contacto personal en las redes sociales u otras

plataformas. Se desaconseja publicar fotos de tarjetas de embarque o boletos de viaje -esa información puede atraer la atención no deseada de piratas informáticos, estafadores o incluso ladrones-. Es necesario limitar la audiencia a personas de confianza y reitera la importancia de abstenerse de compartir demasiados detalles de los viajes (y cuánto tiempo estarán fuera de casa).

- Los puntos de acceso en hoteles, centros de conferencias y zonas públicas plantean importantes riesgos de seguridad. Estas redes no suelen ser seguras, lo que permite a los piratas informáticos interceptar datos y, potencialmente, inyectar Malware en los dispositivos. Hay que evitar conectarse a Wi-Fi públicas y que, en su lugar, utilicen redes móviles seguras siempre que sea posible.

- La pérdida de datos puede ser devastadora, especialmente cuando se viaja. Haga hincapié en la importancia de realizar copias de seguridad (véase sección 1.1) periódicas de los archivos importantes, como fotos, vídeos, documentos y contactos. Se pueden emplear varios métodos, como servicios de almacenamiento en la nube, discos duros externos o USB.

- En zonas públicas como aeropuertos o cafeterías se deben evitar siempre dejar sus dispositivos desatendidos. Si es absolutamente necesario alejarse, los dispositivos deben bloquearse de forma segura con una contraseña y guardarse en un lugar seguro. Existen muchas bolsas antirrobo que pueden atarse de forma segura a los muebles si es necesario alejarse momentáneamente. Asegúrate de que las apps de rastreo están instaladas o activadas en los dispositivos que manejas para facilitar su recuperación en caso de robo o pérdida accidental.

Viajes Internacionales en una importante escalada de precauciones en los viajes internacionales, varios países aconsejan a sus ciudadanos para que utilicen teléfonos básicos y/o temporales y computadoras portátiles con datos mínimos al viajar a otros países, citando los crecientes temores de vigilancia digital y detenciones fronterizas arbitrarias. Las advertencias coordinadas reflejan la creciente preocupación por los informes de que agentes fronterizos de algunos países están inspeccionando dispositivos personales, accediendo a datos privados y deteniendo a viajeros basándose en contenido digital, lo que ha provocado una ola de nuevos protocolos de viaje entre países.

Informes recientes han revelado inspecciones generalizadas de los dispositivos personales de los viajeros por las Oficina de Aduanas. Estos incluyen acceso no autorizado a correos electrónicos, cuentas de redes sociales, fotos y comunicaciones privadas, incluso en ausencia de una orden judicial penal. Este escrutinio ha provocado un aumento en el número de viajeros que son detenido, interrogado o se le negó la entrada al país basado en contenido digital encontrado durante la inspección.

En respuesta, varios gobiernos han actualizado sus directrices de viaje, advirtiendo a los ciudadanos que Llevar teléfonos inteligentes, computadoras portátiles o tabletas personales en viajes internacionales puede poner en riesgo su privacidad.

En el centro de los nuevos avisos se encuentra la recomendación explícita de utilizar teléfonos básicos y/o temporales y computadoras portátiles o tabletas reiniciadas a modo de fabrica. Estos teléfonos de bajo costo y con funciones limitadas permiten a los viajeros comunicarse y acceder a servicios esenciales sin exponer datos personales o profesionales confidenciales a los funcionarios fronterizos. Por otro lado, si se llevan computadoras portátiles o tabletas, estas se deben reiniciar a su versión de fabrica y llevar con datos mínimos usando cuentas temporales para su registro en dichos dispositivos (por ejemplo, usando una cuenta de Gmail/Apple temporal).

También se aconseja a los viajeros que:

- Realice una copia de seguridad de todos los datos en la nube y limpie los dispositivos antes de viajar.
- Evite almacenar información confidencial, política o profesional en cualquier Hardware.
- Borrar de forma permanente y segura datos personales de todos los dispositivos antes de viajar.
- Cerrar sesión en todas las cuentas personales y utilizar navegadores de incógnito si es necesario acceder.
- Evite iniciar sesión en cuentas personales de correo electrónico o redes sociales mientras se está en el extranjero.
- Usar almacenamiento en la nube encriptado en lugar de archivos locales.
- Minimizar el uso del dispositivo en la aduana para reducir la exposición.

Lo que una vez fue visto como paranoia se está convirtiendo rápidamente protocolo y los viajeros de todo el mundo se ven obligados a sopesar la conveniencia frente al control en un mundo cada vez más vigilado.

1.8 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 (véase [4]) básico es un entorno estándar para aplicaciones y programación de los usuarios -generalmente en modo de consola no gráfico-, pero no obliga a adoptar mecanismos estándar para controlar las funcionalidades disponibles como un todo (como Windows o Mac). A medida que Linux ha madurado, se ha hecho necesaria otra capa de funcionalidad encima del sistema Linux. Una distribución de GNU/Linux incluye todos los componentes estándar del sistema Linux -modo consola y gráfico-, más un conjunto de herramientas administrativas que simplifican la instalación inicial y desinstalación de paquetes del sistema.

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

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

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

Cabe mencionar que Unix/Linux son los únicos sistemas operativos que quedan cuya interfaz gráfica (un montón de código llamado X Windows System[16]) está separado del sistema operativo[17], 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

[16] 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).

[17] 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.

que han de ejecutarse en modo GUI o no se ejecutarán, en estos casos no es posible pensar en las GUI como algo distinto del sistema operativo; ahora forman parte inalienable de los sistemas operativos a los que pertenecen -y son, con mucho, la parte mayor, más cara y difícil de crear-.

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

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

- Más control sobre tu máquina: tienes mucho poder y control con la línea de comando. Puede ejecutar comandos para cambiar permisos, 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[18] podemos hacer uso de todos los cores de nuestro procesador, por ejemplo, para cambiar el formato de los .jpg a .png podríamos hacer lo siguiente:

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

o

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

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

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.

[18] 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
```

- 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.

1.9 Software Propietario y Libre

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

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

Por otro lado, el Software comercial suele quedar obsoleto muy rápido, ya que constantemente se le agregan nuevas funcionalidades al mismo y estas en general son vendidas como versiones independientes de la adquirida originalmente. Esto obliga al usuario -si quiere hacer uso de ellas- a comprar las nuevas versiones del Software para satisfacer sus crecientes necesidades informáticas. Por lo anterior y dada la creciente complejidad de los paquetes de cómputo y el alto costo de desarrollo de aplicaciones innovadoras, en muchos casos, el costo total del Software que comúnmente los usuarios instalan -y que no necesariamente usan las capacidades avanzadas del programa, por las cuales el Software tiene un alto costo comercial- en su computadora, suele ser más caro que el propio equipo en el que se ejecutan.

1.9.1 Software Propietario

En entornos comerciales, es posible por parte de la empresa, adquirir y mantener actualizado el Software necesario para sus actividades comerciales, pues el costo del mismo se traslada al consumidor final del bien o servicio que la

empresa proporcione. En entornos educativos, de instituciones sin fines lucrativos e incluso, el sector gubernamental, no se cuenta con los recursos necesarios para adquirir y mantener actualizado el Software necesario para todas y cada una de las aplicaciones usadas en las computadoras, ya que en general, las licencias de uso del Software propietario son asignadas en forma individual a cada computadora y no es fácilmente transferible a otra computadora.

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

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

Existe hoy en día, una gran cantidad de paquetes y sistemas operativos comerciales de Software propietario (véase 4.1) que mediante un pago oneroso, permiten a los usuarios de los mismos ser productivos en todas y cada una de las ramas comerciales que involucra nuestra vida globalizada, pero el licenciamiento del uso de los programas comerciales es en extremo restrictivo en su uso y más en su distribución.

1.9.2 Software Libre

El Software libre (véase sección 4.2) son programas de cómputo -sistema operativo, paquetes de uso común y especializados-, desarrollados por usuarios y para usuarios que, entre otras cosas, comparten el código fuente, el programa ejecutable y dan libertades para estudiar, adaptar y redistribuir a quien así lo requiera el programa y todos sus derivados.

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

¿Qué es el Software Libre? La definición exacta y sus diversas variantes se plasman en 4, pero podemos entender el espíritu a través de los documentos de la fundación para el Software libre (véase [13], [2], [9], [10], [8] y [12]). El Software libre concierne a la libertad de los usuarios para ejecutar, copiar, distribuir, cambiar y mejorar el Software:

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

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

La característica más importante que aparece típicamente en un proyecto de este tipo, es que un conjunto de personas separadas a gran distancia, sean capaces, a través de la Web, de los e-mail y de foros, de aunar sus esfuerzos para crear, mejorar, distribuir un producto, de forma que todos ellos se benefician unos de otros. Evidentemente, la mayor parte del peso recae en los desarrolladores, pero también es necesaria una difusión para que los usuarios documenten, encuentren errores, hagan foros de discusión, etc.

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

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

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

En los últimos años, muchos proyectos han pasado de ser simples programas en línea de comandos a complejas aplicaciones multiplataforma -se ejecutan en distintos sistemas operativos como son Windows, Linux, Unix, Mac OS, Android- con ambientes gráficos multimedia que en muchos casos han superado a sus contrapartes comerciales -por ejemplo los navegadores Web-. Para muestra de este maravilloso avance, tomemos el proyecto del sistema operativo Android, que actualmente ejecuta en millones de equipos -como celulares, tabletas, electrodomésticos, etc.- y en los cuales se pueden descargar miles de aplicaciones y está soportado por una gran cantidad de usuarios y empresas comerciales como Google e IBM. Este ha logrado desplazar a muchos de sus competidores por sus múltiples bondades y bajo costo de desarrollo, al reusar miles de aplicaciones ya existentes que usan Software libre y permitir desarrollar otro tanto de aplicaciones bajo una plataforma que se ejecutará en los más diversos procesadores.

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

1.10 Agradecimientos

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

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

Herramientas
http://132.248.181.216/Herramientas/

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

2 Herramientas Administrativas y de Seguridad

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

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

¿Qué otros sabores de GNU/Linux hay?

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

Existen distintas distribuciones de GNU/Linux[19] para instalar, puedes conocer y descargar las diferentes distribuciones desde:

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

y ver cual es la que más te conviene:

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

o probar alguna versión Live[20]:

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

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

https://livecdlist.com/

también las puedes correr como máquina virtual para VirtualBox:

https://www.osboxes.org/

o máquina virtual para QEMU/KVM:

https://docs.openstack.org/image-guide/obtain-images.html

https://github.com/palmercluff/qemu-images

https://bierbaumer.net/qemu/

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

Entre las distintas distribuciones de GNU/Linux para instalar, una de las más ampliamente usadas es Debian GNU/Linux[21] y sus derivados como Ubuntu. La comunidad de Debian GNU/Linux te apoya para que lo obtengas, instales y de una vez por todas puedas usar GNU/Linux en tu computadora.

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

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

de aquí puedes descargar las imágenes de instalación:

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

además de manuales de instalación y administración:

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

con distintas versiones para todos los gustos:

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

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

también lo puedes correr como máquina virtual (VirtualBox):

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

o en versiones Live:

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

¿Por que usar Debian GNU/Linux?

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

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

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

Entornos de escritorio soportados:

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

Derivados de Debian:

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

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

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

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

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

Sistemas Operativos

2.1 Cuentas de Usuario

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

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

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

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

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

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

2.1.1 El Usuario Administrador del Sistema

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

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

```
$ su antonio
```

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

```
$ su - administracion
```

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

```
$ su
```

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

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

Para usar *sudo* en la línea de comandos, simplemente escriba *sudo* antes del comando que desea ejecutar, sudo le pedirá su contraseña[22]:

```
$ sudo apt update
```

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

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

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

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

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

- id - Nos proporciona información del usuario, uso:

  $ id antonio

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

  $ groups antonio

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

  $ finger antonio

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

  $ getent passwd antonio

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

  $ grep -i antonio /etc/passwd

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

  $ lslogins -u

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

  $ users

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

  $ who -u

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

  $ w

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

  $ last

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

  $ last -ap now

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

  $ lastlog

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

  # lastb

2.1.2 Creación, Modificación y Eliminación de Cuentas

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

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

Se puede hacer a través de tres comandos:

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

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

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

Archivos de inicialización generales:

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

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

Contraseñas las contraseñas o Passwords protegen la información que contienen los dispositivos y cuentas de los usuarios. No obstante, ante la cantidad de claves y combinaciones que cotidianamente se deben utilizar, la mayoría de las personas opta por contraseñas fáciles de recordar por la comodidad que esto implica, o bien, por la falta de conocimiento de lo fácil que puede ser para un ciberdelincuente obtenerlas. Para asegurar la efectividad de las contraseñas[23] y evitar el robo de éstas, es recomendable poner en práctica las siguientes acciones:

- Al generar las contraseñas de los dispositivos y cuentas se deben utilizar claves largas (mínimo 13 caracteres[24]) y únicas para cada caso, evitando utilizar la misma contraseña para diferentes dispositivos o cuentas.

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

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

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

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

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

```
$ su
```

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

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

```
# adduser antonio
```

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

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

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

```
$ id antonio
```

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

```
# passwd antonio
```

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

```
$ passwd -S antonio
```

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

```
# passwd -Sa
```

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

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

```
# passwd -r usuario
```

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

```
# passwd -l antonio
```

la opción -u con passwd desbloquea una cuenta:

```
# passwd -u antonio
```

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

```
# userdel antonio
```

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

```
# userdel -r antonio
```

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

```
# userdel -f antonio
```

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

Crear Grupos de Trabajo[25] podemos crear grupos completos de usuario, para ello hacemos:

```
# groupadd nombre_grupo
```

y podemos cambiar el nombre de un grupo mediante:

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

si necesitamos crear un grupo del sistema, usamos:

```
# groupadd -r nombre_grupo
```

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

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

```
$ groups usuario
```

podemos agregar un usuario a un grupo, mediante:

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

y podemos borrar un usuario de un grupo, usando:

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

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

```
$ groups usuario
```

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

```
$ getent group
```

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

```
# groudel nombre_grupo
```

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

```
$ getfacl archivo
```

y cambiarlos usando:

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

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

```
# ls -la

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

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

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

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

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

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

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

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

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

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

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

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

Cada línea contiene algo similar a:

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

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

/etc/passwd

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

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

Podemos ver las características de la cuenta usando:

```
# chage -l antonio
```

y

```
$ id antonio
```

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

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

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

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

/etc/shadow

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

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

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

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

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

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

Podemos ver las características de la cuenta usando:

```
# chage -l antonio
```

y

```
$ id antonio
```

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

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

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

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

2.2 Información del Equipo y Sistema Operativo

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

Algunos comando usados para conocer y manipular el Hardware:

- lscpu - Información de procesador, uso:

  $ lscpu

- lshw - Lista de Hardware en Linux, uso:

  $ lshw

- hwinfo - Información del Hardware en Linux, uso:

  # hwinfo

- dmidecode - Lista de Hardware de Linux, uso:

  # dmidecode

- hardinfo - Información del Hardware en Linux, uso:

  # hardinfo

- cpuid - Información del CPU, uso:

  $ cpuinfo

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

  # cpufreq-info

- lspci - Lista todos los dispositivos PCI, uso:

  $ lscpi

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

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

- lspcmcia - Información de PCMCIA, uso:

  # lspcmcia

- lsscsi - Listar dispositivos SCSI, uso:

  $ lsscsi

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

  $ lsusb

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

  $ lsof

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

  $ swapon

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

  $ inxi -F

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

$ i7z

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

  $ stress –cpu 2

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

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

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

  $ nproc

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

  $ free

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

  $ lsipc

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

  $ lslocks

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

$ lsmem

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

  $ lsmode

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

  $ tuptime

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

  # modprobe vmhgfs
  # modprobe -r vmhgfs

- sensors - Lee información del CPU como la temperatura, voltaje y velocidad de los ventiladores[26], uso:

[26] Para instalarlo usamos:

```
$ sensors
```

Algunos comando usados para conocer y manipular la red:

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

```
$ ip address
```

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

```
# macchange -r wlp3s0
```

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

```
$ nmcli
```

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

```
$ ping 192.168.1.1
```

- whois - Proporciona información de un dominio, uso:

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

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

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

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

```
$ iperf -s
```

```
# apt install lm_sensors
```

lo configuramos mediante:

```
# sensors-detect
```

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

  $ iperf3 -s -f K

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

  $ ifstat

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

  $ nc -v -n 127.0.0.1 1-100

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

  $ ss

Algunos comando usados para conocer discos y sus particiones:

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

  $ df -h

- lsblk - lista de dispositivos de bloque, uso:

  $ lsblk

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

  # fdisk /dev/sda

- fsck[27] - permite detectar y corregir errores de los discos, uso:

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

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

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

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

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

  # parted /dev/sda1

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

  # gparted

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

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

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

  $ mount

- hdparm - Información de disco duro, uso:

  # hdparam -I /dev/sda2

Algunos comando para conocer datos del sistema operativo:

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

  $ uname -a

- lsb_release, Imprime información del sistema operativo, uso:

  $ lsb_release -a

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

$ hostnamectl

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

  $ cat /etc/os-release

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

  $ cat /etc/issue

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

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

Comandos que nos proporcionan informaciòn sobre dependencias y bibliotecas son:

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

  $ ldd /bin/ls

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

  $ ltrace ls

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

  $ hexdump -c /bin/ls

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

  $ strings /bin/ls

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

  $ readelf -h /bin/ls

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

  $ objdump -d /bin/ls

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

  $ strace -f /bin/ls

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

  $ nm archivoobjeto

- gdb - El depurador de GNU, uso:

  $ gdb -q ./ejecutable

Comandos que nos permiten apagar o reinicializar el sistema:

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

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

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

  ```
  #halt
  ```

- poweroff - apaga el equipo, usó:

  ```
  # poweroff
  ```

- reboot - reinicializa el equipo, usó:

  ```
  # reboot
  ```

2.3 Instalar, Actualizar y Borrar Paquetes

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

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

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

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

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

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

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

La Retrocompatibilidad es un enorme dolor de cabeza, tomar Software hecho para Linux de hace 10 o 5 años y ejecutarlo en una distribución moderna[28]. Cualquier cosa de mínima complejidad o que use una GUI, simplemente no funciona. Mientras la retrocompatibilidad en Windows es simplemente increíble. En Linux somos dependientes de los repo-sitorios en línea, y cuando una aplicación depende de ciertas librerías que empiezan a de-

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

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

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

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

2.3.1 apt-get

Diariamente se actualizan decenas de paquetes en los servidores de Debian GNU/Linux, por ello es necesario hacer el mantenimiento a los paquetes del sistema cada vez que usemos el equipo de cómputo y tengamos acceso a internet, para ello debemos ser el usuario administrador *root*, mediante el comando[29]:

```
$ su
```

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

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

ya siendo *root*, podemos solicitar la descarga de las actualizaciones disponibles, usando[30]:

```
# apt-get update
```

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

```
# apt-get check
```

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

```
# apt-get upgrade
```

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

```
# apt-get clean
```

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

```
# dpkg –configure -a
```

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

```
# apt-get autoremove
```

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

```
# apt-get autoclean
```

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

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

Actualizar Listas:

```
# apt-get update
```

Revisar actualización de listas:

```
# apt-get check
```

Instalar paquete:

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

si sólo queremos simular la instalación usamos:

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

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

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

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

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

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

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

Reinstalar paquete:

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

Actualizar Distro:

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

Actualizar paquete:

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

Actualizar paquetes usando dselect:

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

Eliminar paquetes:

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

Purgar paquetes:

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

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

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

Conocer paquete:

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

Listar paquetes:

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

Listar dependencias de un paquete:

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

Listar paquetes instalados:

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

Validar dependencias incumplidas de un paquete:

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

Configurar dependencias de un paquete:

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

Descargar paquetes:

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

Corregir problemas post-instalación de paquetes:

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

Forzar ejecución de orden de comando:

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

Eliminar descargas de paquetes:

```
# apt-get clean
```

Eliminar paquetes obsoletos y sin usos:

```
# apt-get autoclean
```

Trabajar con las claves GPG de un repositorio

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

Otros importantes:

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

2.3.2 apt

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

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

para usarlo debemos ser el usuario administrador *root*, mediante el comando[31]:

```
$ su
```

ya siendo *root*, podemos solicitar la descarga de las actualizaciones disponibles, usando[32]:

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

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

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

u otra forma de hacer lo mismo:

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

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

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

[31] 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*

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

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

para completar las instalaciones pendientes usar

```
# apt -f upgrade
```

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

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

El comando *apt* soporta las siguientes opciones:

Editar la información de /etc/apt/sourcelist

```
# apt edit-sources
```

Actualizar:

```
# apt update
```

Conocer los paquetes susceptibles a ser actualizados:

```
# apt list –upgradable
```

Buscar paquetes:

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

Instalar paquete:

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

Reinstalar paquete:

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

Actualizar la distribución:

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

Actualizar paquete:

```
# apt upgrade paquete
```

Eliminar paquete:

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

Purgar paquete:

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

Conocer paquete:

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

Listar paquetes:

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

Listar paquetes instalados / actualizables:

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

Corregir problemas post-instalación de paquetes:

```
# apt install -f
```

Forzar ejecución de orden de comando:

```
# apt comando -y
```

Eliminar descargas de paquetes:

```
# apt clean
```

Eliminar paquetes obsoletos y sin usos:

```
# apt autoclean
```

Otros importantes:

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

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

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

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

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

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

p- El paquete se purgó o nunca se instaló.

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

i- Este paquete está instalado en su sistema.

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

2.4 Instalación de los Paquetes más Usados

Existen distintas distribuciones de Linux[33] para instalar, una de las más ampliamente usadas es Debian GNU/Linux y sus distribuciones derivadas como Ubuntu.

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

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

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

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

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

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

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

Estable

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

En pruebas

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

Inestable

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Instalación de Debian GNU/Linux Debian[34] GNU/Linux se puede descargar de múltiples ligas (sin y con Firmware no libre integrado en el *.ISO*), una de ellas es:

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

Réplicas de Debian GNU/Linux en el Mundo En el mundo hay decenas de réplicas de Debian GNU/Linux, para acceder a alguna de ellas (en este apartado supondremos que se trabaja con la versión estable de Debian GNU/Linux), hay que modificar el archivo */etc/apt/sources.list*[35], mediante:

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

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

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

o

apt edit-sources

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

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

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

apt update
apt upgrade

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

apt clean

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

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

[36] 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*

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

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

o

```
# apt edit-sources
```

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

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



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

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

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

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

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

```
# apt clean
```

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

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

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

para luego correrlo usando:

```
# netselect-apt
```

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

Rama de Desarrollo de Backports Algunas veces es necesario instalar en la versión estable algún paquete de la versión testing, pero que no rompa la estabilidad del sistema, para ello se desarrolló Backports[38]. Para usarlo, lo primero es agregar en el archivo */etc/apt/sources.list* las líneas:

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

también existen para los paquetes contrib y non-free:

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

después de concluir la edición, es necesario ejecutar:

```
# apt update
```

Para instalar algún paquete, usar:

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

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

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

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

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

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

también existen para los paquetes contrib y non-free:

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

después de concluir la edición, es necesario ejecutar:

```
# apt update
```

Para instalar algún paquete, usar:

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

2.4.1 Paquetes para Diferentes Necesidades

Metapaquetes Son una solución para grupos de personas con necesidades específicas, no solamente proporciona colecciones manejables (metapaquetes) de paquetes de programas específicos, sino que también facilita la instalación y configuración para el propósito convenido. Cubren los intereses de distintos grupos de personas, su objetivo común es simplificar la instalación y administración de ordenadores para una audiencia determinada[39].

Por ejemplo los hay para:

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

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

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

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

```
$ apt search textlive
```

para saber qué hace cada paquete usar:

```
$ apt show texlive
```

para instalar, usar:

```
# apt install texlive
```

A continuación daremos un listado paquetes[40] agrupados según su uso, muchos de ellos hacen cosas parecidas o se complementan, para saber qué hace cada paquete usar:

```
$ apt show paquete
```

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

```
$ apt search textlive
```

para instalar, usar:

```
# apt install paquete(s)
```

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

Drivers para nuestro Hardware

drive

Firmware libre y no libre

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

firmware-misc-nonfree

Actualización del Microcódigo

amd64-microcode intel-microcode

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

fwupd

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

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

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

Trusted Plataform Module (TPM) de Trusted Computing Group (TCG)

tmp2-tools

tpm-tools trousers

Aplicaciones de monitoreo y rendimiento

sysstat dfc htop atop vtop gtop iotop iftop ifstat lsof \
glances nmon collectl itop vnstat vnstati dnsutils fio \
nmap iptraf iptraf-ng nethogs nload bmon tcpdump \
nethogs slurm tcptrack bwm-ng cbm speedometer \
netwatch trafshow netload traceroute bashtop stress \
nslookup sysbench whowatch macchanger sysstat \
netstat-nat jnettop fping pktstat dstat tuptime \
darkstat iperf3 netperf monitorix psmisc smem \
smemcap smemstat

Aplicaciones para conocer el Hardware de un equipo

procinfo lshw hwinfo inxi cpuid hardinfo cpufrequtils i7z \
dmidecode cpufreq-info lspcmcia lsscsi gparted

Aplicaciones para administración

rcconf powertop cpulimit timelimit

Aplicaciones para probar el ancho de banda entre dos servidores

iperf iperf3 bwctl-client bwctl-server netcat

Comandos usuales, nueva implementación

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

Manejador de tareas

tasque

Seguridad

wipe secure-delete scrub

Recuperador de datos

gddrescue

Sincronización de archivo y directorios

rsync grsync rclone luckybackup syncthing

Trabajar en múltiples servidores

clusterssh pssh pscp tmux-cssh

Descarga de archivos URL

wget uget curl lftp woof httpie youtube-dl
persepolis

Acelerador de descargas de archivos URL

aria2 axel

Logo de la distribución

linuxlogo screenfetch neofetch

Terminales

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

Intérpretes de órdenes

csh tcsh ksh fish zsh tsch dash dsh busybox

Manejo de archivo sobre CLI

mc ranger vifm nnn lfm wcm ytree

Navegadores de red

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

Actualizar el navegador por omisión

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

Navegadores de red CLI

elinks elvis-tiny links links2 lynx lynx-cur w3m browsh

Actualizar el navegador por omisión

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

Búsquedas en Google CLI

googler

Búsquedas en Duck Duck Go CLI

ddgr

Manejo de particiones NTFS

ntfs-3g ntfsprogs scrounge-ntfs

Manejo de SAMBA

samba smbclient cifs-utils

Manejo de particiones

parted partimage gparted testdisk genisoimage f3

Servidor de SSH

openssh-server

dropbear

Clientes de SSH

openssh-client filezilla gftp mosh

Respaldo y recuperación de datos

timeshift bacula dump

Control de cambios en un proyecto

git git-all gitk gitg git-cola git-gui qgit tig vim-fugitive git-extras

mercurial

subversion rapidsvn

cvs tkcvs

Remplazo de grep para proyectos grandes

ripgrep

silversearcher-ag

ack-grep

```
# dpkg-divert –local –divert /usr/bin/ack –rename –add \
/usr/bin/ack-grep
```

Servidores

apache2 libapache2-mod-evasive apachetop httpd

php libapache2-mod-php php-mysql php-gd phpmyadmin

mysql-common mysql-client mysql-server mytop mysql-admin \

ferret mysql-workbench mysql-workbench-data mycli \

automysqlbackup

postgresql postgresql-client postgresql-doc postgresql-contrib \
autopostgresqlbackup
sqlite3 sqlite3-doc sqlitebrowser
redis-server
mongodb mongodb-clients mongodb-server
mariadb-client mariadb-server mariadb-common
nginx lighttpd tomcat9 nodejs caddy openlitespeed cherokee
firebird-server firebird-utils
httpie
samba smbclient
vsftpd
openssh-server openssh-client
nfs-kernel-server
rsync
cups

Instalar distribuciones Linux/BSD en USB drives

gnome-disk-utility unetbootin unetbootin-translations gksu \
testdisk

Poder correr aplicaciones de Windows mediante WINE

wine

```
# dpkg –add-architecture i386 && aptitude update && \
aptitude install wine32
```

Paquetes para instalar impresoras locales y remotas

system-config-printer system-config-printer-udev cups \
cups-pk-helper
hplip

Búsqueda de archivos duplicados

fslint fdupes jdupes findimagedupes duff rdfind hardlink

Utilerias de compactación y descompactación

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

Crear y montar sistemas de archivos cifrados

gocryptfs cryfs encfs ecryptfs-utils

Cifrado de archivos y discos

tomb cryptmount gnupg cryptsetup gnome-disk-utility
gnupg openssl ccrypt mcrypt bcrypt zip 7-zip age

Generar contraseñas

gnupg openssl apg pwgen makepasswd

Gestor de contraseñas

kwalletmanager seahorse keepass2 keepassx keepassxc \
password-gorilla passwordsafe
kwalletcli lastpass-cli pass gopass yapet kpcli

Recobrar archivos borrados

testdisk

Respaldo de información

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

2.4.2 Manejadores de Paquetes Multiplataforma

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

Los programas empaquetados con formatos independientes incluyen todas las dependencias que necesitan para su funcionamiento. Es decir que la dependencia 1 se instalará cada vez que se instale un programa que la necesite.

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

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

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

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

aunque poco a poco va creciendo el número de Software que se empaqueta en estos tipos de paquetes.

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

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

¿Qué es Snap? Mientras que Flatpak tuvo sus orígenes en la comunidad de desarrollo de Fedora/Red Hat, Snap lo tuvo en Canonical, la empresa que desarrolló este tipo de gestión de paquetería tan peculiar. Un tipo de paquete universal que ya aceptan gran cantidad de distribuciones y apps empaquetadas en él. En este caso, los paquetes se ejecutan dentro de AppArmor, aunque se pueden ejecutar fuera de la sandbox.

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

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

Las dependencias de los paquetes Flatpak y Snap que nadie habla En Linux hay muchas maneras de instalar un mismo Software, pero uno de

los reclamos que podemos encontrar es que al instalar se incluye Software principal y dependencias en un mismo paquete, lo que les hace funcionar desde un principio, son más limpios y esas cosas, pero esto es una verdad a medias.

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

Dependencias compartidas, pero dependencias al fin y al cabo Por ejemplo, cuando queremos instalar un programa que convierte archivos multimedia a otros formatos, si no lo tenemos ya es probable que nos descargue *FFmpeg* e *ImageMagick*, cada uno con unas cuantas dependencias más. Estas sí son dependencias al uso, pero las que se instalan junto a un paquete Flatpak o snap son lo necesario para que se pueda ejecutar ese programa en nuestra plataforma. Si la aplicación está escrita en GTK o tiene componentes de GNOME, nos instalará la plataforma de GNOME y sus traducciones. Cuando instalemos otro programa GTK/GNOME, esto ya lo tendremos, por lo que no será necesario y el peso de la aplicación ya será el que vemos en las tiendas de Software. En el caso de los paquetes Snap tenemos un poco lo mismo.

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

```
$ flatpak uninstall –delete-data
```

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

```
$ flatpak uninstall –unused
```

Si lo que queremos es eliminarlo todo, deberemos escribir:

$ flatpak uninstall –all

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

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

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

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

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

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

Para instalar Snap. usamos:

```
# apt install snapd
```

Para buscar un paquete usamos:

```
$ snap find libreoffice
```

Para instalar un paquete usamos:

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

Para listar los paquetes instalados usamos

```
$ snap list
```

Para actualizar un paquete previamente instalado usamos:

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

Para desinstalar un paquete usamos:

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

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

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

Para instalar Flatpak, usamos:

```
# apt install flatpak
```

para activar su integración con Gnome, usamos:

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

Para buscar algún paquete, usamos:

```
$ flatpak search aplicación
```

Para instalar paquetes usamos:

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

por ejemplo:

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

Algunos desarrolladores tienen su propio repositorio, podemos usarlo mediante:

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

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

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

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

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

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

```
$ flatpak update -v
```

Podemos correr un Flatpak mediante:

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

si hemos instalado spotify, lo podemos correr usando:

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

Para listar los Flatpak instalados usamos:

$ flatpak list

Para desinstalar un Flatpak, usamos:

$ flatpak uninstall <ApplicationID>

por ejemplo:

$ flatpak uninstall com.spotify.Client

Para actualizar todas las aplicaciones Flatpak instaladas usamos:

$ flatpak update

Podemos remover las aplicaciones Flatpak no usadas mediante:

$ flatpak uninstall –unused

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

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

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

apt install cargo

para auto actualizar el paquete usamos:

$ cargo update

para instalar algún paquete usamos:

$ cargo install paquete

para desinstalar un paquete usamos:

$ cargo uninstall paquete

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

- Linux Standard Base (LSB)
- LSB-compliant .rpm packages
- .deb
- Stampede (.slp)
- Solaris (.pkg)
- Slackware (.tgz, .txz, .tbz, .tlz)

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

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

Para instalar el paquete usamos:

```
# apt install alien
```

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

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

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

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

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

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

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

- -d, –to-deb - Crea paquetes Debian (este es el predeterminado)
- -r, –to-rpm: crea paquetes rpm
- -l, –to-lsb: crea un paquete LSB
- -t, –to-tgz - Crea paquetes tgz
- –to-slp - Crea paquetes slp
- -p, –to-pkg: crea paquetes pkg de Solaris

2.4.3 Windows en Linux

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

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

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

Entre toda la variedad de librerías, bibliotecas y recursos, podemos encontrarnos con prácticamente todas las bibliotecas de interrupciones para programas, lo que permite hacer llamadas INT en tiempo real. De esta manera, los programas no saben que se están ejecutando en un sistema operativo que no es Windows, simplemente se ejecutan. Y lo hacen igual que en él. Si algún programa, o juego, tiene dependencias especiales (por ejemplo, una DLL concreta) podemos añadirla fácilmente a Wine. Todas las librerías se encuentran dentro del directorio «~/.wine/drive_c/windows/system32», que equivale al directorio System32 de Windows.

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

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

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

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

funcionan.

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

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

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

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

```
# apt install playonlinux
```

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

```
# apt install wine
```

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

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

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

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

2.4.4 macOS en Linux

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

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

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

```
$ darling shell
```

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

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

Podemos verificar el núcleo con el siguiente comando:

```
$ uname
```

y averiguar la versión de macOS con:

```
$ sw_vers
```

salimos de la terminal con

```
$ exit
```

y apagamos el contenedor con:

```
$ darling shutdown
```

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

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

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

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

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

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

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

Podemos desinstalar los programas con:

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

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

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

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

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

2.4.5 Debian GNU/Linux User Repository

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

2.4.6 Debian Janitor Paquetes Desde Repositorio Git

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

3 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[41]. 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.

[41] 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.

3.1 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 *" ! "*[42], 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:

[42] 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[43] 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
```

[43]

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
```

3.2 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[44]:

$ 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[45]

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'

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

[45] 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[46] 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*.

[46] 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 3.9).

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)[47]:

```
$ 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
```

[47] 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[48] usando: unp archivoCompactado
- * Descompactar usando: dtrx archivoCompactado
- * Descompactar usando: unar archivoCompactado

[48] El programa unp permite descomprimir un archivo de casi cualquier formato, para instalar el paquete usamos:

```
# apt install unp
```

para descompactar, usamos:

```
$ unp archivoCompactado
```

El proceso de instalación y uso es similar para los paquetes: dtrx y unar.

Formato *tar.gz* para respaldar y compactar un grupo de archivos y/o directorios en formato *tar.gz*, usamos:

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

o usamos:

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

o usamos:

```
$ tar -jcvf nombre.tar.bz2 directorio
```

para restaurar, usamos:

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

o usamos:

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

o usamos:

```
$ tar -jxvf nombre.tar.bz2
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.tar.gz; do tar -zcvf "$z"; done
```

Formato *gz* para compactar archivos y/o directorios al formato *gzip*, usamos:

```
$ gzip ficheros
```

y para descompactarlo usamos:

```
$ gzip -dk fichero.gz
```

si solo nos interesa descompactar y no preservar el archivo .gz:

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

o

```
$gunzip fichero.gz
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.gz; do gunzip "$z"; done
```

También podemos hacer la compactación en paralelo usando múltiples cores, mediante el paquete *pigz* con formato *gzip* (-0 sin compresión, -1 compresión más rápida, -6 compresión por omisión y -9 mejor compresión), para compactar manteniendo el archivo original, usamos:

```
$ pigz -k archivo.iso
```

generara un archivo.izo.gz. Podemos listar el contenido de un .gz usando:

```
$ pigz -l archivo.iso.gz
```

Para comprimir un directorio, es necesario usar tar, mediante:

```
$ tar cf - Directorio/ | pigz > archivo.tar.gz
```

o

$ tar –use-compress-program=pigz -cf archivo.tar.gz Directorio/

Para descomprimir usamos:

$ pigz -d archivo.iso.gz

o

$ unpigz archivo.iso.gz

Por omisión *pigz* usa todos los cores de la máquina, si necesitamos limitar el número de cores usamos la bandera -p y el número de cores a usar, por ejemplo:

$ pigz -9 -k -p3 archivo.iso

Podemos usar otros formatos de compresión, usando la opción -k -z para zlib (.zz), usando:

$ pzip -k -k archivo.iso

o usando la opción -k -K para zip (.zip):

$ pzip -k -K archivo.iso

Formato *bz2* para compactar archivos y/o directorios al formato *bz2, bz, tbz2 tbz o bzip2*, usamos:

$ bzip2 ficheros

y para descompactarlo usamos:

$ bzip2 -d fichero.bz2

podemos descomprimir usando el comando **bzcat** mediante:

$ bzcat -dc fichero.bz2

o bien descomprimir usando el comando **bunzip2** mediante:

$ bunzip2 fichero.bz2

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

$ for z in *.bz2; do bunzip2 "$z"; done

Formato *xz* para respaldar y compactar un grupo de archivos y/o directorios en formato *tar.xz*, usamos:

```
$ tar -cvJf nombre.tar.xz directorio
$ xz nombre.tar
$ xz -k archivo.tar
$ tar –xz -cf - /trayectoria | ssh usuario@maquina "cat > archivo.txz"
```

podemos controlar el nivel de compresión usando valores de 0 (sin compresión) al 9 (mejor compresión), mediante:

```
$ xz -9 archivo.tar
$ xz –k9 archivo.tar
```

podemos descomprimir usando:

```
$ tar -xf nombre.tar.xz
$ tar -xvf nombre.tar.xz
$ tar -Jxvf nombre.tar.xz
$ tar -xf nombre.tar.xz
$ tar –xz -xf archivo.txz
$ xz -v -d nombre.tar.xz
$ xz –decompress archivo.tar.xz
$ unxz archivo.xz
```

y podemos ver el contenido de un archivo *.tar.xz* si usamos:

```
$ tar ft archivo.tar.xz
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.tar.xz; do tar-xf "$z"; done
```

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

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

también permite establecer los niveles de compresión usando una escala de 0 a 9 (por omisión es 6), por ejemplo:

```
$ zip -8 archivo.zip fichero(s)
$ zip -8 -r archivo.zip directorio/
```

y podemos borrar los archivos que se comprimirán al terminar el proceso, usando:

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

Podemos agregar un archivo a un *.zip* existente, usando:

```
$ zip -u archivo.zip nuevo.txt
```

o eliminar un archivo a un .zip existente, usando:

```
$ zip -d archivo.zip porBorrar.txt
```

Podemos crear un *.zip* protegido con clave, usando:

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

si necesitamos poner una clave de acceso (digamos 2GAXTW), usamos

```
$ zip -P 2GAXTW -r archivo.zip ficheros
```

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

```
$ zipcloak archivo.zip
```

Podemos ver la información detallada del archivo comprimido, usando:

```
$ zipdetails archivo.zip
```

Y descomprimir usando:

$ unzip archivo.zip

para descompactar un solo directorio usamos:

$ unzip -d directorio archivo.zip

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

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

para descomprimir un *.zip* protegido con clave (digamos 2GAXTW), usamos:

$ unzip -P 2GAXTW archivo.zip

para extraer sin mostrar salida de los archivos extraídos, usamos:

$ unzip -q archivo.zip

para indicar que no se extraigan algunos archivos, usamos:

$ unzip archivo.zip -x *.eps

para sobreescribir los archivos al descomprimir sin pedir confirmación, usamos:

$ unzip -o archivo.zip

para no sobreescribir los archivos al descomprimir, usamos:

$ unzip -n arvhivo.zip

para actualizar archivos y en su caso crearlos si es necesario, usamos:

$ unzip -u archivo.zip

para actualizar archivos pero no crear nuevos, usamos:

$ unzip -f archivo.zip

para listar el contenido de un *.zip*, usamos:

```
$ unzip -l archivo.zip
```

para obtener información detallada de un *.zip*, usamos:

```
$ unzip -v archivo.zip
```

para revisar la integridad de un *.zip*, usamos:

```
$ unzip -t archivo.zip
```

Si al descompactar existe algún error, es posible recuperar parte de los archivos mediante:

```
$ zip -F archive.zip -O archive-fixed.zip
```

o usar *-FF*, después usar:

```
$ jar xvf archive-fixed.zip
```

otra alternativa es usar:

```
$ bsdtar xf archivo.zip
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.zip; do unzip "$z"; done
```

Formato *7z* para respaldar y compactar un grupo de archivos y/o directorios en formato *7-zip*, usamos:

```
$ 7z a archivo.7z ficheros
```

también podemos pedir que use una clave (password$$) para cifrar el archivo:

```
$ 7z a archivo.7z ficheros -ppassword$$
```

para restaurar usamos:

```
$ 7z e archivo.7z
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.7z; do 7z e "$z"; done
```

Formato *zst* para respaldar y compactar un grupo de archivos y/o directorios en formato *zst*, usamos:

```
$ zstd ficheros -o archivo.zst
```

para restaurar usamos:

```
$ zstd -d archivo.zst
```

o usamos:

```
$ unzstd archivo.zst
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.zst; do unzstd "$z"; done
```

Formato *lha* para respaldar y compactar un grupo de archivos y/o directorios en formato *lha*, usamos:

```
$ lha a archivo.lha ficheros
```

para restaurar usamos:

```
$ lha x archivo.lha
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.lha; do lha x "$z"; done
```

Formato *arj* para respaldar y compactar un grupo de archivos y/o directorios en formato *arj*, usamos:

```
$ arj a archivo.arj ficheros
```

para restaurar usamos:

```
$ arj x archivo.arj
$ unarj archivo.arj
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.arj; do arj x "$z"; done
```

Formato *zoo* para respaldar y compactar un grupo de archivos y/o directorios en formato *zoo*, usamos:

```
$ zoo a archivo.zoo ficheros
```

para restaurar usamos:

```
$ zoo x archivo.zoo
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.zoo; do zoo x "$z"; done
```

Formato lz y tlz para compactar un *archivo* y remplazarlo por *archivo.lz*, usamos:

```
$ lzip -v archivo
```

para compactar todo un dispositivo (por ejemplo */dev/sdc*), usamos:

```
$ lzip -c /dev/sdc > archivo.lz
```

para restaurar usamos:

```
$ tar -xf archivo.lz
$ lzip -d archivo.lz
```

o extraer un archivo multivolumen de un archivo tar

```
$ lzip -cd archivo.tar.lz | tar -xf -
```

Podemos usar *tarlz* que es una combinación de *tar* y *lzip* que trabaja en paralelo para hacer la compresión, por ejemplo de los archivos *a*, *b*, y *c*:

```
$ tarlz -cf archivo.tar.lz a b c
```

y podemos adicionarle los archivos *d* y *e*, usando:

```
$ tarlz -rf archivo.tar.lz d e
```

Para extraer los archivos podemos usar:

```
$ tarlz -xf archivo.tar.lz
```

También podemos usar la versión plzip que trabaja en paralelo. Para compactar un *archivo* y remplazarlo por *archivo.lz*, usamos:

```
$ plzip -v archivo
```

para compactar todo un dispositivo (por ejemplo */dev/sdc*), usamos:

```
$ plzip -c /dev/sdc > archivo.lz
```

y para restaurar usamos:

```
$ plzip -d archivo.lz
```

Formato cpio para empaquetar archivos (por ejemplo*.txt) en un archivo .cpio, usamos:

```
$ ls *.txt | cpio -ov > archivo.cpio
```

para desempaquetar, usamos:

```
$ cpio -idv < archivo.cpio
```

Formato *rar* para respaldar y compactar un grupo de archivos y/o directorios al formato *rar*, usamos:

```
$ rar a archivo.rar ficheros
```

para restaurar usamos:

```
$ rar x archivo.rar
$ unrar archivo.rar
```

En algunos casos, archivos *rar* de Windows no es posible descomprimirlos correctamente en Linux, para descomprimirlos podemos descargar utilerías GNU para Win32 de:

http://unxutils.sourceforge.net/

entre ellas, descargar *unrar* (es un sólo archivo zip) de la dirección:

http://sourceforge.net/projects/unxutils/

ahora usando Wine es posible descomprimir los archivos desde Linux mediante:

```
$ wine unrar.exe e archivo.rar
```

Para descomprimir archivos *rar* o *zip* rotos, usar:

```
$ unrar e -kb -y nombreArchivo.rar
```

o usamos:

```
$ bsdtar xf nombreArchivo.zip
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.rar; do unrar e "$z"; done
```

UNP para descompactar casi de cualquier formato existe un programa llamado *unp* (basado en un Scrip de Perl) que permite descomprimir de casi cualquier formato, para instalarlo hacemos:

```
# apt install unp
```

Es un extractor universal, su uso es de lo más sencillo, a saber:

```
$ unp archivoCompactado
```

Parte de los formatos soportados y por que programas se muestra en la siguiente lista (generada usando: *unp -s*):

7z (p7zip or p7zip-full), ace (unace), ar,deb (binutils), arj (arj), bz2 (bzip2), cab (cabextract), chm (libchm-bin or archmage), cpio,afio (cpio or afio), dat (tnef), dms (xdms), exe (orange or unzip or unrar or unarj or lha), gz (gzip), cpio,afio (cpio or afio), dat (tnef), dms (xdms), exe (orange or unzip or unrar or unarj or lha), gz (gzip), hqx (macutils), lha,lzh (lha), lz(lzip), lzma (xz-utils or lzma), lzo (lzop), lzx (unlzx), mbox (formail and mpack), pmd (ppmd), rar (rar or unrar or unrar-free), rpm (rpm2cpio and cpio), sea,sea.bin (macutils), shar (sharutils), tar (tar), tar.bz2,tbz2 (tar with bzip2), tar.lzip (tar with lzip), tar.lzop,tzo (tar with lzop), tar.xz,txz (tar with xz-utils), tar.z (tar with compress), tgz,tar.gz (tar with gzip), uu (sharutils), xz (xz-utils), zip,cbz,cbr,jar,war,ear,xpi,adf (unzip), zoo (zoo).

DTRX para descompactar de múltiples formatos esta herramienta llamada *dtrx* (Do The Right Extraction), es una aplicación de código abierto que simplifica el proceso de extracción de archivos comprimidos. Para su instalación hacemos:

```
# apt install dtrx
```

Y ya estamos listos para comenzar a usarlo en la terminal. Dtrx soporta los formatos comunes de compresión como: tar, bz2, zip, cpio, deb, rpm, gem, 7z, cab, lzh, rar, gz, bz2, lzma, xz, además de binarios deb, gem (ruby), archivos de Installshield e incluso algunos tipos de exe. También descomprime archivos comprimidos con gzip, bzip2, lzma, xz, y otros compresores.

Para descomprimir usamos:

```
$ dtrx archivo.zip
```

Podemos hacer un preview del contenido de un archivo lanzandolo con el parámetro -l:

```
$ dtrx -l archivo.zip
```

Si tenemos un archivo comprimido que a su vez incluye múltiples ficheros, al ejecutar *dtrx* nos va a dar 5 opciones diferentes, para hacer la cosa más o menos automática:

-a siempre extraer los archivos incluidos durante esta sesión

-o extraer los archivos incluidos esta vez.

-n elegir no extraer los archivos incluidos por esta vez.

-v nunca extraer los archivos incluidos durante esta sesión.

-l listar los archivos incluidos.

Si queremos que trabaje de forma recursiva, lo podemos hacer añadiendo la opción -r:

```
$ dtrx -r archivo.tar.gz
```

Además con el parámetro -m podemos extraer los metadatos de archivos *deb* y *gem*:

```
$ dtrx -m archivo
```

UNAR para descompactar de múltiples formatos esta herramienta llamada *unar*, es una aplicación de código abierto que simplifica el proceso de extracción de archivos comprimidos. Para su instalación hacemos:

```
# apt install unar
```

Y ya estamos listos para comenzar a usarlo en la terminal, *unar* soporta los formatos comunes de compresión como: zip, rar, 7z, tar, gzip, bzip2, lzma, xz, cab, msi, nsis, exe, iso, bin, arj, arc, pakm, acem, lzh, adf, entre otros.

Para descomprimir usamos:

```
$ unar archivo.zip
```

Otras opciones cuando se tiene una lista de archivos de distintas trayectorias o es resultado de una búsqueda, para compactar es preferible usar *afio*. Podemos instalarlo usando:

```
# apt install afio
```

Para compactar, digamos todos los archivos **.?pp* (programas de C++) usar:

```
$ find . -name *.?pp | afio -o -Z fuentes
```

para descompactarlos, usar:

```
$ afio -i -Z fuentes
```

Si se desea compactar usando GZIP, usar:

```
$ cat lista | afio -o -Z -G 9 fuentes
```

Si se desea ver el listado de archivos que contiene fuentes, usar:

```
$ afio -t fuentes
```

Si se desea compactar y mandar a otra máquina usar:

```
$ find . -name *.?pp | afio -o -Z user@servidor%ssh:fuentes
```

Como el uso de *afio* no necesita extensión en el archivo, para descompactarlo de cualquier formato es recomendable usar *unp*, este escoge el mejor método para el archivo en cuestión:

```
$ unp archivoCompactado
```

Respaldo y/o Restauración Remoto es posible usar *ssh*[49] en conjunción con *tar o dd*[50] 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*[51] 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:

[49] 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.

[50] 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.

[51] 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

3.3 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[52]. 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:

[52] 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[53]] Origen [Origen]... Destino
```

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

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

[53] Algunas opciones son: -r recursivo, -b backups, -R relativo, -u actualiza, -p muestra el progreso, -c comprime, -p preserva permisos, -v muestra lo que hace, -q trabaja en modo silencioso, -l preserva ligas simbólicas, -H preserva enlaces duros, -t preserva tiempos de modificación, etc.

Si queremos saber que hará el comando pero sin hacer la operación indicada, podemos usar la opción –dry-run, por ejemplo:

$ rsync –dry-run ~/Datos/ ~/Respaldo/

hay varias opciones que podemos usar en *rsync*, ejemplo de ellas es:

$ rsync –verbose –recursive –links –hard-links –times –delete –stats ~/Datos/ ~/Respaldo/

en este caso se sincronizaría el contenido de ~/Datos con *~/Respaldo*, pero lo hará mostrando lo que transfiere, de forma recursiva, conservará enlaces simbólicos y sus tiempos, borrará los archivos que estén en el destino pero no en el origen y al terminar mostrará las estadísticas de la transferencia.

Para hacer la transmisión cifrada entre equipos, podemos usar *rsync* conjuntamente con *ssh*, supongamos que se esta en la máquina y quiere tener sincronizada la carpeta ~/Datos con mas de un equipo, mediante *ssh* usando un puerto *343*, en la máquina *192.168.13.230* y usuario *antonio*, usar:

$ rsync –partial –recursive –links –hard-links –times –verbose –delete –stats ~/Datos/ -e 'ssh -p 343' antonio@192.168.13.230: ~/Respaldo/

por supuesto esto puede hacerse en cualquier dirección, i.e. de la máquina remota a nuestra máquina o viceversa, ejemplo:

$ rsync –partial –recursive –links –hard-links –times –verbose –delete –stats -e 'ssh -p 343' antonio@192.168.13.230: ~/Respaldo/ ~/Datos/

Los siguientes ejemplos permiten copiar los archivos (por ejemplo *.mp3) pero preservan la estructura de directorios que existía en su lugar de origen:

$ rsync -a -m –include '*/' –include '*.mp3' –exclude '*' ~/miDirectorio/ ~/OtrDirectorio

$ rsync -a –prune-empty-dirs –include '*/' –include '*.mp3' –exclude '*' ~/miDirectorio/ ~/OtrDirectorio

inicia la búsqueda en *~/midirectorio/* y los copia en *~/OtroDirectorio*.

Algunas veces necesitamos cambiar la prioridad de ejecución de un proceso *rsync* para evitar saturar el sistema, para ello podemos usar *ionice* que cambiara la prioridad de los comandos en ejecución, en este caso para todos los procesos de rsync se ejecutarán solo cuando la carga en el sistema sea ligera, mediante:

```
$ pgrep rsync | xargs ionice -c3 -p
```

Entre las opciones para excluir archivos o directorios en el uso de rsync, tenemos:

```
$ rsync -av –exclude 'archivo.txt' /fuente/ /destino/
$ rsync -av –exclude 'miDirectorio' /fuente/ /destino/
$ rsync -av –exclude '*.mp3' /fuente/ /destino/
$ rsync -av –exclude '*.mp3' –exclude '*.txt' /fuente/ /destino/
$ rsync -av –exclude={'*.mp3', '*.txt'} /fuente/ /destino/
```

si tenemos un *archivo.txt* con el contenido a excluir, podemos usar:

```
$ rsync -av –exclude-form={'archivo.txt'} /fuente/ /destino/
```

podemos excluir por el tamaño de los archivos, ejemplos:

```
$ rsync -av –max-size=500m /fuente/ /destino/
$ rsync -av –min-size=1m /fuente/ /destino/
```

pssh permite transferir o copiar archivos a múltiples servidores en Linux con un mismo comando:

- pscp - es una utilería para copiar archivos en paralelo a múltiples equipos
- prsync - es una utilería para transferir de forma eficiente archivos entre múltiples equipos en paralelo

- pnuke - permite concluir procesos en múltiples equipos en paralelo
- pslurp - permite copiar archivos de múltiples equipos a un equipo central en paralelo

Si creamos un archivo *Hosts.txt*, con los *IPs* como el siguiente:

```
192.168.0.3:22
192.168.0.9:22
```

podemos usar para copiar un archivo a múltiples servidores:

```
$ pscp -h Hosts.txt -l USR -Av wine-1.7.55.tar.bz2 /tmp/
```

o de forma alternativa usamos:

```
$ pscp.pssh -h Hosts.txt -l USR -Av wine-1.7.55.tar.bz2 /tmp/
```

donde:

-h indica que se lean los *IPs* del archivo indicado
-l se indica el usuario a usar en todos los equipos.
-A solicita el password para ser enviado a *ssh*
-v visualiza las operaciones y mensajes que genera el comando

Podemos copiar directorios a múltiples servidores, usando:

```
$ pscp -h myscpHosts.txt -l USR -Av -r Android\ Games/
/tmp/
```

o de forma alternativa:

```
$ pscp.pssh -h myscpHosts.txt -l USR -Av -r Android\ Games/
/tmp/
```

nc el comando *netcat* (*nc*) es una utilidad que permite escribir y leer datos a través de conexiones de red usando los protocolos TCP y UDP. Supongamos que estamos en el IP 192.168.13.230. Podemos generar un archivo compactado, cifrarlo y enviarlo a otro equipo en una mismo comando mediante:

```
$ tar -cvzf - directorio | gpg -c | nc -l 6666
```

y para recibirlo en el otro equipo usamos:

```
$ nc 192.168.13.230 6666 | gpg -d | tar -xvzf -
```

Siempre es mejor opción usar *scp*, pero *nc* cumplirá con su cometido.

3.4 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 3.2)

- scp (véase 3.3)
- rsync (véase 3.3)
- 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[54]. 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:
```

[54] 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.

3.5 Control de Versiones

El control de versiones es una herramienta que permite registrar y administrar los cambios en el código fuente o en documentos. También se le conoce como control de código fuente, control de revisiones o gestión de código fuente.

El control de versiones es útil para:

- Trabajar de forma colaborativa entre desarrolladores

- Conservar la integridad del código
- Reducir el tiempo de desarrollo
- Aumentar el número de implementaciones exitosas
- Proteger el código fuente
- Resolver problemas como cambios incompatibles entre desarrolladores
- Resolver problemas como nuevos errores introducidos por los cambios

El control de versiones funciona tomando una instantánea de los archivos cada vez que un desarrollador los edita. Estas instantáneas se guardan de forma permanente para que se puedan recuperar más adelante si es necesario.

El control de versiones también se puede utilizar para gestionar los cambios en documentos. En este caso, el control de versiones organiza, etiqueta y conserva cada borrador hasta que se completa una versión final.

¿Qué es control de versiones? se define como control de versiones a la gestión de los diversos cambios (Commit es un conjunto de cambios guardados en el repositorio de Git y tiene un identificador SHA1 único) que se realizan sobre los elementos de algún producto o una configuración del mismo, es decir, es lo que se hace al momento de estar desarrollando un Software o una página Web. Exactamente es eso que haces cuando subes y actualizas tu código en la nube, o le añades alguna parte o simplemente editas cosas que no funcionan como deberían o al menos no como tú esperarías.

¿A que le llamamos sistema de control de versiones? son todas las herramientas que nos permiten hacer todas esas modificaciones antes mencionadas en nuestro código y hacen que sea más fácil la administración de las distintas versiones de cada producto desarrollado; es decir *Git*.

Antes de seguir avanzando, conviene definir algunos términos comunes que encontraremos más adelante:

- Un repositorio es una carpeta que contiene los archivos y subdirectorios de un proyecto. Puede ser público o privado, dependiendo de quién deba tener acceso.

- Una rama es una ruta de desarrollo separada en el mismo repositorio. En un mismo proyecto suelen utilizarse ramas separadas para trabajar en nuevas características sin interferir con la versión de producción. Una vez que el código es revisado y probado, un administrador del repositorio puede fusionar los cambios en la rama principal.

- Un commit es una instantánea de un repositorio en un momento dado. Permite a los programadores incluir comentarios y pedir la opinión de otras personas. Usando el Hash que lo identifica puedes volver a un estado anterior del proyecto si es necesario. Antes de que los archivos y directorios puedan ser comiteados, necesitamos decirle a Git que los rastree. Normalmente nos referimos a este paso simplemente como agregar los archivos al área de Staging.

- Un Pull Request (o simplemente PR) es un método para informar a otros desarrolladores y discutir los cambios recientes antes de incorporarlos a la ruta de desarrollo principal.

- Un Fork es un proyecto independiente que se basa en un repositorio determinado. A diferencia de las ramas, no es local a este último. Sin embargo, también puede fusionarse con él a través de un Pull Request adecuado.

- Se puede utilizar un archivo .gitignore para indicar qué contenido local no queremos incluir en el repositorio. Esto nos ayuda a evitar enviar archivos temporales o exponer información sensible en una solución basada en la Web.

3.5.1 Git

Git es un programa de control de versiones que sirve para la gestión de los diversos cambios que se realizan sobre los elementos de algún proyecto de Software y sus respectivos programas fuente o configuración del mismo guardando instantáneas (Snapshots) del código en un estado determinado, que viene dado por el autor y la fecha. Fue diseñado por Linus Torvalds y es usado para controlar los cambios de diversos proyectos como los fuentes del Kernel de Linux que tiene decenas de millones de líneas de código (en la versión 4.12 cuenta con 24,170,860 líneas de código repartidos en 59,806 archivos) y es trabajado por miles de programadores alrededor del mundo.

Git fue creado pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente, es decir *Git* nos proporciona las herramientas para desarrollar un trabajo en equipo de manera inteligente y rápida y por trabajo nos referimos a algún Software o página que implique código el cual necesitemos hacerlo con un grupo de personas.

Algunas de las características más importantes de *Git* son:

- Rapidez en la gestión de ramas (Branche es como una línea de tiempo a partir de los Commit, siempre hay siempre como mínimo una rama principal predefinida llamada Master), debido a que *Git* nos dice que un cambio será fusionado mucho más frecuentemente de lo que se escribe originalmente.
- Gestión distribuida: Los cambios se importan como ramas adicionales y pueden ser fusionados de la misma manera como se hace en la rama local.
- Gestión eficiente de proyectos grandes.
- Realmacenamiento periódico en paquetes.

Con esto en mente, vamos a hacer una introducción a Git desde cero.

Instalación de Git para instalar *Git* completo en el servidor o en la máquina de trabajo, usamos:

```
# apt install git-all
```

Para instalar lo básico de *Git*, si no está instalado:

```
# apt install git
```

otras opciones para trabajar con *Git* son:

```
# apt install git git-all gitk gitg git-cola git-gui qgit tig lighttpd
vim-fugitive
```

También existen otros proyectos parecidos a Git, algunos de ellos son:

```
# apt install mercurial
# apt install subversion rapidsvn
# apt install cvs
```

Configuración de Git si se quiere especificar la identidad del que controla el repositorio local en el equipo, debemos usar (por omisión toma la información de la cuenta del usuario y máquina):

```
$ git config –global user.name "Antonio Carrillo"
$ git config –global user.email antoniocarrillo@ciencias.unam.mx
```

si se desea configurar el editor de texto a usar por *Git*, usamos (por omisión es *vim*):

```
$ git config –global Core.editor scite
```

si se desea configurar la herramienta de control de diferencias, usamos (por omisión *vimdiff*):

```
$ git config –global merge.tool meld
```

para comprobar podemos usar:

```
git config –global -list
```

el cual creará un archivo de texto llamado .gitconfig, y podemos ver su contenido, usando:

```
cat ~/.gitconfig
```

Clonar un repositorio Git podemos iniciar clonando algún repositorio remoto de la red, para ello debemos conocer su dirección y usar:

```
$ git clone <dirección>
```

esto permitirá bajar todo el contenido del proyecto a clonar.

Crear un repositorio Git local si lo que requiero es un control personal sin necesidad de compartir los archivos con ningún otro usuario, puedo usar *Git* de forma local en cualquier directorio mediante:

```
$ git init
```

Si se desea agregar la identidad del que controla el repositorio en este directorio, se debe usar:

```
$ git config user.name "Antonio Carrillo"
$ git config user.email antoniocarrillo@ciencias.unam.mx
```

Ahora para agregar los archivos (todos los de este directorio), usar:

```
$ git add .
```

así podemos hacer la confirmación de los cambios, mediante:

```
$ git commit -m "Primer lanzamiento"
```

ahora cada que lo requiera al hacer modificaciones, puedo checar los cambios:

```
$ git status
```

o en forma gráfica con *gitk*, mediante:

```
$ gitk
```

para actualizar los cambios, usar:

```
$ git commit -a -m 'Actualizacion'
```

en caso necesario, podemos mover uno o más archivos de una trayectoria a otra, usando:

```
$ git mv archivo(s) /trayectoria
```

o podemos borrar uno o más archivos, usando:

```
$ git rm archivo(s)
```

además podemos editar el archivo *.gitignore*, para indicar los archivos a ignorar usando líneas independientes para cada tipo.

Podemos ver el histórico de las operaciones que hemos hecho, usando:

```
$ git log
```

y podemos comprobar el estado del repositorio, usando:

```
$ git status
```

La otra alternativa es preparar un directorio para el repositorio ya sea en el servidor o de forma local, mediante:

```
$ mkdir example.git
$ cd example.git
```

Para inicializar el repositorio:

```
$ git –bare init
```

Es buena opción limitar el acceso a la cuenta vía *ssh*, por ello es mejor cambiar en */etc/passwd*, la línea del usuario predeterminada:

```
tlahuiz:x:1005:1005:Tlahuizcalpan„„:/home/tlahuiz:/bin/bash
```

a esta otra:

```
tlahuiz:x:1005:1005:Tlahuizcalpan„„:/home/tlahuiz:/usr/bin/git-
Shell
```

En la máquina de trabajo o en el servidor en cualquier carpeta se genera la estructura del repositorio en un directorio temporal de trabajo para el repositorio:

```
$ mkdir tmp
$ cd tmp
$ git init
```

Para generar la estructura de trabajo para el repositorio y los archivos necesarios:

```
$ mkdir branches release trunk
$ mkdir ...
```

Para adicionar todos y cada uno de los archivos y carpetas:

```
$ git add .
```

Para subir los cambios:

```
$ git commit -m "Texto"
```

Después debemos mandarlo al servidor:

```
$ git remote add origin ssh://usr@máquina/~/trayectoria
```

o mandarlo a un directorio local:

```
$ git remote add origin ~/trayectoria
$ git push origin +master:refs/heads/master
```

Para usar el repositorio en cualquier otra máquina hay que bajar el repositorio por primera vez del servidor:

```
$ git clone ssh://usr@máquina/~/trayectoria
```

o de una carpeta local:

```
$ git clone ~/trayectoria
```

Ahora, podemos configurar algunos datos usados en el control de cambios:

```
$ git config –global usr.name "Antonio Carrillo"
$ git config –global usr.email antoniocarrillo@ciencias.unam.mx
```

cuando se requiera actualizar del repositorio los cambios:

```
$ git pull
```

para subir los cambios al repositorio:

```
$ git commit -a -m "mensaje"
$ git push
```

Comando usados para el trabajo cotidiano en *Git* para ver el estado de los archivos locales, usamos:

```
$ git status
```

para generar una nueva rama y trabajar en ella:

```
$ git branch MiIdea
$ git checkout MiIdea
```

o en un solo paso:

```
$ git checkout -b MiIdea
```

Para unificar las ramas generadas en el punto anterior:

```
$ git checkout master
$ git merge MiIdea
```

Para borrar una rama:

```
$ git branch -d MiIdea
```

Para listar ramas:

```
$ git branch
```

Para listar ramas fusionadas:

```
$ git branch –merged
```

Para listar ramas sin fusionar:

```
$ branch –no-merged
```

Para ver los cambios en el repositorio:

```
$ git log
```

o verlos en forma acortada:

$ git log –pretty=oneline

Para recuperar un archivo de una actualización anterior:

$ git show a30ab2ca64d81876c939e16e9dac57c8db6fb103:ruta/al/archivo > ruta/al/archivo.bak

Para volver a una versión anterior:

$ git reset –hard 56f8fb550282f8dfaa75cd204d22413fa6081a11:

para regresar a la versión presente (cuidado con subir cambios en ramas anteriores):

$ git pull

Si en algún momento borramos algo o realizamos cambios en nuestra máquina y necesitamos regresar los archivos como estaban en nuestra última actualización, podemos usar:

$ git reset –hard HEAD

este trabaja con la información de nuestra copia local y no necesita conexión de red para la restitución. Eventualmente es necesario optimizar la copia local de los archivos en *Git*, para ello podemos usar:

$ git gc

Visualizador gráfico para *Git*:

apt install gitk

Git es un proyecto pujante, amplio y bien documentado, ejemplos y documentación puede ser consultada en:

- https://git-scm.com/book/es/v1
- http://git-scm.com/documentation
- https://coderwall.com/p/kucyaw/protect-secret-data-in-git-repo

Git en Google Drive:

- http://www.iexplain.org/using-git-with-google-drive-a-tutorial/
- https://techstreams.github.io/2016/09/07/google-drive-as-simple-git-Host/

Un Resumen de los Comando de Git más Usados

`CONFIGURACION`

Configurar Nombre que salen en los commits

git config –global user.name "antonio"

Configurar Email

git config –global user.email antonio@gmail.com

Marco de colores para los comando

git config –global color.ui true

`INICIANDO REPOSITORIO`

Iniciamos GIT en la carpeta donde está el proyecto

git init

Clonamos el repositorio de github o bitbucket

git clone <url>

Añadimos todos los archivos para el commit

git add .

Hacemos el primer commit

git commit -m "Texto que identifique por que se hizo el commit"

Subimos al repositorio

git push origin master

`GIT CLONE`

Clonamos el repositorio de github, gitlab o bitbucket

git clone <url>

Clonamos el repositorio de github, gitlab o bitbucket

git clone <url> git-demo

`GIT ADD`

Añadimos todos los archivos para el commit

git add .

Añadimos el archivo para el commit

git add <archivo>

Añadimos todos los archivos para el commit omitiendo los nuevos

git add –all

Añadimos todos los archivos con la extensión especificada

git add *.txt

Añadimos todos los archivos dentro de un directorio y de una extensión especifica

git add docs/*.txt

Añadimos todos los archivos dentro de un directorios

git add docs/

`GIT COMMIT`

Cargar en el HEAD los cambios realizados

git commit -m "Texto que identifique por que se hizo el commit"

Agregar y Cargar en el HEAD los cambios realizados

git commit -a -m "Texto que identifique por que se hizo el commit"

De haber conflictos los muestra

git commit -a

Agregar al último commit, este no se muestra como un nuevo commit en los logs. Se puede especificar un nuevo mensaje

git commit –amend -m "Texto que identifique por que se hizo el commit"

`GIT PUSH`

Subimos al repositorio

git push <origien> <branch>

Subimos un tag

git push –tags

`GIT LOG`

Muestra los logs de los commits

git log

Muestras los cambios en los commits

git log –oneline –stat

Muestra gráficos de los commits

git log –oneline –graph

`GIT DIFF`

Muestra los cambios realizados a un archivo

git diff
git diff –staged

`GIT HEAD`

Saca un archivo del commit

git reset HEAD <archivo>

Devuelve el último commit que se hizo y pone los cambios en staging

git reset –soft HEAD^

Devuelve el último commit y todos los cambios

git reset –hard HEAD^

Devuelve los 2 últimos commit y todos los cambios

git reset –hard HEAD^^

Rollback merge/commit

git log
git reset –hard <commit_sha>

`GIT REMOTE`

Agregar repositorio remoto

git remote add origin <url>

Cambiar de remote

git remote set-url origin <url>

Remover repositorio

git remote rm <name/origin>

Muestra lista repositorios

git remote -v

Muestra los branches remotos

git remote show origin

Limpiar todos los branches eliminados

git remote prune origin

`GIT BRANCH`

Crea un branch

git branch <nameBranch>

Lista los branches

git branch

Comando -d elimina el branch y lo une al master

git branch -d <nameBranch>

Elimina sin preguntar

git branch -D <nameBranch>

`GIT TAG`

Muestra una lista de todos los tags

git tag

Crea un nuevo tags

git tag -a <verison> - m "esta es la versión x"

`GIT REBASE`

Los rebase se usan cuando trabajamos con branches esto hace que los branches se pongan al día con el master sin afectar al mismo

Une el branch actual con el mastar, esto no se puede ver como un merge

git rebase

Cuando se produce un conflicto no das las siguientes opciones:

cuando resolvemos los conflictos –*continue* continua la secuencia del rebase donde se pauso

git rebase –continue

Omite el conflicto y sigue su camino

git rebase –skip

Devuelve todo al principio del rebase

git reabse –abort

Para hacer un rebase a un branch en específico

git rebase <nameBranch>

`OTROS COMANDOS`

Lista un estado actual del repositorio con lista de archivos modificados o agregados

git status

Quita del HEAD un archivo y le pone el estado de no trabajado

git checkout – <file>

Crea un branch en base a uno Online

git checkout -b newlocalbranchname origin/branch-name

Busca los cambios nuevos y actualiza el repositorio

git pull origin <nameBranch>

Cambiar de branch

git checkout <nameBranch/tagname>

Une el branch actual con el especificado

git merge <nameBranch>

Verifica cambios en el repositorio Online con el local

git fetch

Borrar un archivo del repositorio

git rm <archivo>

`Fork`

Descargar remote de un fork

git remote add upstream <url>

Merge con master de un fork

git fetch upstream
git merge upstream/master

Git-crypt El paquete *git-crypt* es una solución que usa *GPG* por debajo de *Git* que permite cifrado y descifrado transparente de archivos en un repositorio *git*.

Los archivos que se requieran proteger serán cifrados al hacer commit y descifrados al hacer *checkout* y permite compartir libremente un repo-sitorio que contenga contenido tanto público como privado. De esta forma, permite trabajar de manera transparente con el contenido descifrado, de forma que desarrolladores que no tengan la clave secreta podrán clonar y hacer commit en un repositorio con archivos cifrados.

Esto te permite almacenar tu material secreto (como pueden ser claves) en el mismo repositorio que tu código sin tener que bloquearlo. Solo un usuario autorizado puede dar permisos a otros usuarios.

Para instalar el paquete git-crypt usamos:

```
# apt install git-crypt
```

Ya instalado debemos preparar el repositorio git, para crear la llave, entonces usar:

```
$ git-crypt keygen ~/crypt-key
```

Ahora podemos crear el repositorio:

```
$ cd repo
$ git-crypt init
```

Especifica que carpetas/archivos deben ser cifrados, como git-filters:

```
$ cat .gitattributes
```

keys filter=git-crypt diff=git-crypt

crear la lista de los archivos a cifrar

```
$ vi .gitattributes
```

Indicamos que se cifren, por ejemplo, los archivos *.html, .org*, directorio:secretdir/**secreto y archivo, con cualquier extensión o palabra que le preceda.

*.html filter=git-crypt diff=git-crypt‘

*.org filter=git-crypt diff=git-crypt

directorio_secreto/** filter=git-crypt diff=git-crypt

archivo filter=git-crypt diff=git-crypt

ahora cada vez que hagamos un commit, los archivos *.html y .org*, subirán cifrados.

Ya podemos usar la llave para cifrar los archivos indicados por .gitattributes mediante:

```
$ git-crypt unlock ~/crypt-key
```

y agregar los archivos que deseamos cifrar, usando *git add*, revisando el estado de los archivos cifrados mediante:

```
$ git-crypt status -f
```

y podemos hacer los commits necesarios.

Al clonar el repositorio, los archivos cifrados se mostrarán como tal, hasta hacer en el repositorio:

```
$ git-crypt unlock ~/crypt-key
```

mostrando los archivos descifrados a partir de ese momento

Si se desea respaldar el repositorio en un solo archivo se puede usar:

```
$ git bundle create /tmp/Respaldo –all
```

y para restaurar usar algo como:

```
$ git clone /tmp/Respaldo newFolder
```

También podemos añadir usuarios autorizados (identificados por su clave *GPG*), mediante:

```
$ git-crypt add-gpg-user USER_ID
```

Flujos de trabajo comunes

- En la máquina del desarrollador: Crea el vault, añádete como usuario fiable. Pide las claves públicas a los miembros de tu equipo y añádelas al vault.
- En el entorno de Integración Continua (CI): Añade una clave *GPG* común para los ejecutores jenkins/CI. Autorízarla en el repositorio.

Seguridad

- Git-crypt usa *GPG* internamente, así que el nivel de seguridad debería ser el dado por *GPG*, a excepción de posibles errores en el propio programa *git-crypt*.

- Git-crypt es más seguro que otros sistemas git de cifrado transparente, *git-crypt* cifra archivos usando *AES-256* en modo *CTR* con un synthetic IV derivado del *SHA-1 HMAC* del archivo. Este modo de operar proporciona seguridad semántica ante *CPAs* (chosen-plain attacks) determinísticos. Esto significa que pese a que el cifrado es determinística (lo cual es requerido para que git pueda distinguir cuando un archivo ha cambiado y cuando no), no filtra información más allá de mostrar si dos archivos son idénticos o no.

Limitaciones y Trucos

- Cualquier usuario no autorizado puede ver que estamos usando *git-crypt* basándose en la evidencia dejada en el archivo .gitattributes.

- Git-crypt no cifra nombres de archivo, mensajes de *commit, symlink targets, gitlinks*, u otros metadatos.

- Git-crypt se apoya en git filters, los cuales no fueron diseñados con el cifrado en mente. Así pues, *git-crypt* no es la mejor herramienta para cifrar la mayoría o totalidad de los archivos de un repositorio. Donde *git-crypt* destaca es en aquellos casos en que la mayoría del repositorio es público pero unos pocos archivos deben ser cifrados (por ejemplo, claves privadas o archivos con credenciales API). Para cifrar un repositorio entero, es mejor considerar usar un sistema como **git-remote-gcrypt**.

- Git-crypt no esconde cuando un archivo cambia o no, cuanto ocupa o el hecho de que dos archivos sean idénticos.

- Los archivos cifrados con *git-crypt* no se pueden comprimir. Incluso el más pequeño de los cambios en un archivo cifrado requiere que git archive el archivo modificado en su totalidad y no solo un delta.

- A pesar de que *git-crypt* protege el contenido de los archivos individuales con *SHA-1 HMAC*, *git-crypt* no puede ser usado de forma segura a menos que el repositorio entero esté protegido contra la alteración de

datos (un atacante que pueda mutar tu repositorio podrá alterar tu archivo *.gitattributes* para deshabilitar el cifrado). Si fuera necesario, usa características de *git* cómo signed tags en vez de contar únicamente con *git-crypt* para la integridad.

- El *diff* del *commit* varía cuando el vault está abierto vs cuando está cerrado. Cuando está abierto, los contenidos del archivo están en formato plano, es decir, descifrados. En consecuencia puedes ver el *diff*. Cuando el vault está cerrado, no se puede apreciar un *diff* efectivo ya que el texto cifrado cambia, pero el ojo humano no puede distinguir los contenidos.

Además de Git usado de forma local, existen diversos servicios en la nube[55] que permiten dar soporte a proyectos mediante *Git,* en los cuales es necesario crear una cuenta y subir los datos usando *Git,* algunos de estos servicios son:

3.5.2 GitLab vs GitHub

Aunque GitHub y GitLab tienen similitudes, incluso en el propio nombre que comienza por Git debido a que ambas se basa en la famosa herramienta de control de versiones escrita por Linus Torvalds, pero ni una ni otra son exactamente iguales. Por ello, el ganador de la batalla GitHub vs GitLab no está tan claro, tienen algunas diferencias que hacen que tengan sus ventajas y desventajas para los usuarios y desarrolladores que las suelen usar.

¿Qué es GitHub? es una plataforma de desarrollo colaborativo, también llamado forja. Es decir, una plataforma enfocada hacia la cooperación entre desarrolladores para la difusión y soporte de su Software (aunque poco a poco se ha ido usando para otros proyectos más allá del Software).

Como su propio nombre indica, se apoya sobre el sistema de control de versiones *Git*. Así, se puede operar sobre el código fuente de los programas y llevar un desarrollo ordenado. Además, esta plataforma está escrita en Ruby on Rails.

[55] Algunos de estos proyectos gratuitos son: Gitlab, Github, Bitbucket, Beanstalk, Launchpad, SourceForge, Phabricator, GitBucket, Gogs, Gitea, Apache Allura, entre otros.

Tiene una enorme cantidad de proyectos de código abierto almacenada en su plataforma y accesibles de forma pública. Tal es su valor que Microsoft optó por comprar esta plataforma en 2018, aportando una cifra de nada menos que de 7,500 millones de dólares.

Pese a las dudas sobre esa compra, la plataforma continuó operando como de costumbre, y continúa siendo una de las más usadas. En ella se alojan proyectos tan importantes como el propio Kernel Linux.

¿Qué es GitLab? es otra alternativa a GitHub, otro sitio de forja con un servicio Web y un sistema de control de versiones también basado en Git. Por supuesto, se ideó para el alojamiento de proyectos de código abierto y para facilitar la vida de los desarrolladores, pero existen algunas diferencias con el anterior.

Esta Web, además de la gestión de repositorios y control de versiones, también ofrece alojamiento para Wikis, y sistema de seguimiento de errores. Una completa suite para crear y gestionar los proyectos de todo tipo, ya que, al igual que GitHub, actualmente se alojan proyectos que van más allá del código fuente.

Fue escrito por unos desarrolladores ucranianos, Dmitry Zaporozhets y Valery Sizov, usando lenguaje de programación Ruby y algunas partes en Go. Después se mejoró su arquitectura con Go, Vue.js, y Ruby on Rails, como en el caso de GitHub.

A pesar de ser muy conocida y ser la gran alternativa a GitHub, no cuenta con tantos proyectos. Eso no quiere decir que la cantidad de código alojado sea muy grande, con organizaciones que confían en ella de la talla del CERN, la NASA, IBM, Sony, etc.

Personalmente, te diría que no existe un claro ganador en la batalla GitHub vs GitLab. No es tan sencillo elegir una plataforma que sea infinitamente superior a la otra, de hecho, cada una tiene sus puntos fuertes y sus puntos débiles. Y todo dependerá de lo que realmente busques para que te tengas que decantar por una u otra.

Diferencias GitHub vs GitLab A pesar de todas las similitudes, una de las claves a la hora de decantarse en la comparativa GitHub vs GitLab pueden ser las diferencias entre ambas:

Niveles de autenticación: GitLab puede establecer y modificar permisos a los diferentes colaboradores según su función. En el caso de GitHub, puede

decidir quién tiene derechos de lectura y escritura en un repositorio, pero es más limitado en ese sentido.

- Alojamiento: aunque ambas plataformas permiten alojar el contenido de los proyectos en las propias plataformas, en el caso de GitLab también te puede permitir autoalojar tus repositorios, lo que puede ser una ventaja en algunos casos. GitHub ha agregado esa característica también, pero solo con ciertos planes de pago. GitHub ofrece máximo 3 colaboradores gratis y hasta 500 MB por repositorio, en cambio en GitLab no hay límite al número de colaboradores y hasta 10 GB por repositorio.

- Importación y exportación: GitLab contiene información muy detallada de cómo poder importar proyectos para moverlos de una plataforma a otra, como GitHub, Bitbucket, o traerlos a GitLab. Además, a la hora de exportar, GitLab ofrece un trabajo muy sólido. En el caso de GitHub no se ofrece documentación detallada, aunque se puede usar GitHub Importer como herramienta, aunque puede ser algo más restrictivo cuando se trata de exportar.

- Comunidad: ambos tienen una buena comunidad tras de sí, aunque GitHub parece haber ganado la batalla en popularidad. Actualmente aglutina millones de desarrolladores. Por eso, será más sencillo encontrar ayuda al respecto.

- Versiones Enterprise: ambos las ofrecen si pagas la cuota, por lo que se podría pensar que la comparativa GitHub vs GitLab no tiene sentido en este punto, pero lo cierto es que GitLab ofrece unas prestaciones muy interesantes, y se ha hecho popular entre los equipos de desarrollo muy grandes.

Ventajas y Desventajas de GitLab una vez conocidas las diferencias y semejanzas entre GitHub vs GitLab, las ventajas y desventajas de estas plataformas te pueden ayudar a decidirte.

Ventajas

- Plan gratuito y sin limitaciones, aunque tiene planes de pago.

- Es de licencia de código abierto.

- Permite el autohospedaje en cualquier plan.
- Está muy bien integrado con Git.

Desventajas

- Su interfaz puede ser algo más lenta con respecto a la competencia.
- Existen algunos problemas habituales con los repositorios.

Ventajas y Desventajas de GitHub Por otro lado, GitHub también tiene sus pros y contras, entre los que destacan los siguientes:

Ventajas

- Servicio gratuito, aunque también tiene servicios de pago.
- Búsqueda muy rápida en las estructura de los repos.
- Amplia comunidad y fácil encontrar ayuda.
- Ofrece prácticas herramientas de cooperación y buena integración con Git.
- Fácil integración con otros servicios de terceros.
- Trabaja también con TFS, HG y SVN.

Desventajas

- No es absolutamente abierto.
- Tiene limitaciones de espacio, ya que no puedes exceder de 100MB en un solo archivo, mientras que los repositorios están limitados a 1GB en la versión gratis.

Como ves, no existe un claro ganador. La elección no es fácil y, como comenté, deberías vigilar muy bien las ventajas, desventajas y diferencias de cada uno para poder identificar cuál se adapta mejor a tus necesidades.

Personalmente te diría que si quieres disponer de un entorno totalmente abierto, mejor usa GitLab. En cambio, si prefieres más facilidades y usar el servicio Web con más presencia, entonces ve a por GitHub. Incluso incluiría un tercero en discordia y te diría que si buscas trabajar con servicios Atlassian deberías mirar del lado de Bitbucket.

Uso GitLab (https://about.gitlab.com/)

Para configurar:

```
git config –global user.name "Antonio Carrillo Ledesma"
git config –global user.email "antoniocarrillo@ciencias.unam.mx"
```

Para crear nuevo repositorio:

```
git clone https://gitlab.com/antoniocarrillo69/MDF.git
cd MDF
touch README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

Para usar una carpeta existente:

```
cd existing_folder
git init
git remote add origin https://gitlab.com/antoniocarrillo69/MDF.git
git add .
git commit -m "Initial commit"
git push -u origin master
```

Para usar un repositorio existente:

```
cd existing_repo
git remote rename origin old-origin
git remote add origin https://gitlab.com/antoniocarrillo69/MDF.git
git push -u origin –all
git push -u origin –tags
```

Uso Github (https://github.com/)

Para configurar:

```
git config –global user.name "Antonio Carrillo Ledesma"
git config –global user.email "antoniocarrillo@ciencias.unam.mx"
```

Para configurar un nuevo repositorio:

```
$ touch README.md
$ git init
$ git add .
$ git commit -m "mi primer commit"
$ git remote add origin https://github.com/antoniocarrillo69/ejemploPruebas.git
$ git pull origin master
$ git push origin master
```

3.6 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[56], *shred* sobrescribe repetidamente el contenido con valores aleatorios, haciendo un uso intensivo de disco[57]. 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
```

[56] 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.

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

las opciones utilizadas son las siguientes:

-n 5: 5 iteraciones de sobrescritura.

-u: borrar el archivo al finalizar.

-v: muestre el progreso.

-f: fuerza el cambio de permisos para permitir la escritura de ser necesario.

-z: rellenar con ceros al final para ocultar el propio Shredding.

Se observa que en cada pasada se alterna entre rellenado con ceros, con datos aleatorios y con unos (ffffff). Esto se hace para maximizar la efectividad del borrado o Shredding, de forma que todos los Bits resulten modificados al menos una vez. Finalmente se renombra y se borra. Esto ocurre para hacer el Shredding del propio nombre del archivo en la entrada de directorio.

Lo podemos usar en múltiples archivos a la vez, mediante:

```
$ shred -u -v *.txt
```

o en una partición del disco (/dev/sda2/), mediante:

```
# shred -vfz /dev/sda2
```

También es posible usar el paquete *wipe*, para instalarlo usamos

```
# apt install wipe
```

y lo usamos mediante:

```
$ wipe archivo
$ wipe -r -q directorio/*
# wipe /dev/sda2
```

o el paquete *secure-delete*, para instalarlo usamos:

```
# apt install secure-delete
```

este paquete consta de cuatro herramientas, a saber:

- srm - utilizado para borrar archivos o directorios que se encuentren en disco

  ```
  $ srm archivo
  $ srm -r directorio/*
  ```

- sdmem - utilizado para limpiar restos de información en la memoria (RAM) de la máquina

  ```
  # sdmem -f -v
  ```

- sfill - utilizado para limpiar los restos de información del espacio vacío del disco

  ```
  # sfill -v /home/usuario
  ```

- sswap - utilizado para limpiar los restos de información de la partición Swap, para conocerla usamos:

  ```
  $ swapon
  ```

  supongamos que la partición Swap es: /dev/sda2, entonces:

  ```
  # swapoff /dev/sda2
  # sswap /dev/sd2
  # swapon /dev/sda2
  ```

Cifrado de Discos y Particiones Al momento de instalar el sistema operativo una buena práctica de seguridad es solicitar que use particiones cifradas para nuestros datos y el sistema operativo. Esto permite mantener a nuestros datos lejos de miradas indiscretas. Además cada disco externo o unidad Flash se recomienda cifrar, esto es posible al formatear el dispositivo -casi todos los escritorios de GNU/Linux tienen integrada una opción para ello-, así en caso de perdida o robo del dispositivo nuestros datos permanecen ilegibles para cualquiera.

En caso que nuestro escritorio no tenga instalada una opción gráfica para formatear y cifrar dispositivos, podemos usar GNOME Disks (también conocida como *gnome-disk-utility*, o *GNOME Disks*) es una aplicación gráfica de *udisks* incluido en el paquete *gnome-disk-utility*. Se puede utilizar GNOME Disk para la gestión de particiones (cifrar), motorización de tipo *SMART*, evaluación comparativa y Software *RAID*.

Para instalar usamos:

```
# apt install gnome-disk-utility
```

y para usar:

```
# gnome-disks
```

esto nos permite entre otras cosas al formatear una unidad y cifrarla.

Cifrar Discos y Particiones con cryptsetup es uno de los Softwares de cifrado de disco más usado. Este se encarga de cifrar/descifrar los datos escritos en un dispositivo de almacenamiento. El cifrado a nivel de disco garantiza que los archivos siempre se almacenen en el disco de forma cifrada. Los archivos solo están disponibles para poder ser leídos mientras el sistema está en ejecución y desbloqueado por uno de los usuarios con acceso a la clave de descifrado. Una persona no autorizada que intente acceder al contenido del disco, solo encontrará datos confusos, en lugar de los archivos reales.

Todos los métodos de cifrado de disco funcionan de tal manera que, aunque el disco realmente contiene datos cifrados, el sistema operativo lo ve como los datos legibles normales, siempre que el contenedor cifrado (es decir, la parte del disco que contiene los datos cifrados) ha sido desbloqueado y montado en alguna partición del sistema.

Instalar Paquete usamos:

```
# apt install cryptsetup
```

Es bueno conocer el rendimiento del programa *cryptsetup* para los diferentes tipos de cifrado soportado, para ello usamos:

```
# cryptsetup benchmark
```

algunos de ellos son:

> PBKDF2-sha1, PBKDF2-sha256, PBKDF2-sha512, PBKDF2-ripemd160, PBKDF2-whirlpool, argon2i, argon2id, aes-cbc 128, serpent-cbc 128, twofish-cbc 128, aes-cbc 256, serpent-cbc 256, twofish-cbc 256, aes-xts 256, serpent-xts 256, twofish-xts 256, aes-xts 512, serpent-xts 512, twofish-xts 512

esto nos permitirá elegir el algoritmo óptimo para nuestro equipo, que nos de el mejor rendimiento en el cifrado y mayor seguridad posible.

El primer paso es escoger la unidad de disco que vamos a cifrar, podemos usar el comando *lsblk* para obtener una lista de dispositivos de bloque montados en el equipo de cómputo, mediante:

```
$ lsblk
```

supongamos que trabajaremos con: */dev/sdb1*.
Crear partición cifrada

```
# cryptsetup -v -y -c aes-xts-plain -s 512 luksFormat /dev/sdb1
# cryptsetup luksOpen /dev/sdb1 Respaldos
# ls -l /dev/mapper/Respaldos
# cryptsetup -v status Respaldos
```

Formatear partición

```
# mkfs.ext4 /dev/mapper/Respaldos
# mkdir -p /opt/Respaldos_crypt
# mount /dev/mapper/Respaldos /opt/Respaldos_crypt
# df -Th | grep crypt
# cd /opt/Respaldos_crypt
```

Montar y activar sistema de archivos

```
# cryptsetup luksOpen /dev/sdc1 Respaldos
# mkdir -p /opt/Respaldos_crypt
# mount /dev/mapper/Respaldos /opt/Respaldos_crypt
# df -Th | grep crypt
# cd /opt/Respaldos_crypt
```

Desmontar y desactivar el sistema de archivos

```
# umount /opt/Respaldos_crypt
# cryptsetup luksClose Respaldos
```

El uso del dispositivo cifrado en la gran mayoría de las distribuciones de GNU/Linux que tengan instalado el paquete *cryptsetup* reconocerán el dispositivo al momento de ser introducido en el equipo de cómputo y solicitarán la clave de acceso. A partir de ese momento será posible usar el dispositivo como cualquier otro -sin cifrado- y todas las tareas de montaje/desmontaje serán transparentes para el usuario.

Podemos usar el comando fsck en sistemas basados en LUKS, mediante:

```
# umount /opt/Respaldos_crypt
# fsck -vy /dev/mapper/Respaldos
# mount /dev/mapper/Respaldos /opt/Respaldos_crypt
```

Para cambiar la contraseña para la partición cifrada, usamos:

```
# cryptsetup luksDump /dev/sdb1
# cryptsetup luksAddKey /dev/sdb1
```

se pueden configurar un máximo de 8 contraseñas para cada dispositivo
Remover o eliminar la contraseña:

```
# cryptsetup luksRemoveKey /dev/sdb1
```

tengamos en cuenta que debemos ingresar la contraseña guardada.

Cifrado Basado en Archivos con gocryptfs Podemos tratar de mantener nuestros datos fuera de miradas indiscretas usando gocryptfs en las máquina a las que tengamos acceso (incluso del usuario administrador *root)*[58]. 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

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

manera eficiente utilizando herramientas como rsync. Además, el tamaño del sistema de archivos cifrado es dinámico y solo está limitado por el espacio disponible en disco.

Para instalarlo usamos:

```
# apt install gocryptfs
```

después, es necesario crear los directorios para guardar los datos cifrados (por ejemplo en el directorio ~/.cifrados) y los descifrados (por ejemplo en el directorio ~/descifrados), mediante:

```
$ mkdir -p ~/.cifrados ~/descifrados
```

Para inicializar la carpeta usamos:

```
$ gocryptfs -init ~/.cifrados
```

nos pedirá la clave de acceso y su confirmación. Se pueden crear tantas carpetas independientes con *gocyptfs* como se requieran y el cualquier parte del sistema de archivos al que tengamos acceso.

Ahora ya podemos montar el directorio mediante:

```
$ gocryptfs ~/.cifrados ~/descifrados
```

el sistema nos pedirá nuestra clave de acceso y de ser correcta nos permitirá hacer el montaje. De esta forma, la carpeta virtual *~/cifrados* mostrará nuestros datos descifrados, solo visibles para el usuario que monta la carpeta (no para root) y la carpeta *~/.cifrados*, contendrá nuestros datos de forma cifrada[59], 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
```

[59] El nombre de los archivos tienen un límite, por ello hay que tenerlo en cuenta si se usan nombre largos en el sistema de archivos (el límite aproximado es 255 caracteres), pero no importa el número de archivos o carpetas almacenadas internamente.

Cifrar Archivos con GnuPG es una herramienta en línea de comandos de seguridad en comunicaciones electrónicas en donde se utiliza criptografía de clave pública para que los usuarios puedan comunicarse de un modo seguro. En un sistema de claves públicas cada usuario posee un par de claves, compuesto por una clave privada y una clave pública. Cada usuario debe mantener su clave privada secreta; no debe ser revelada nunca. La clave pública se puede entregar a cualquier persona con la que el usuario desee comunicarse. GnuPG implementa un esquema algo más sofisticado en el que un usuario tiene un par de claves primario, y ninguno o más de un par de claves adicionales subordinadas. Los pares de claves primarios y subordinados se encuentran agrupados para facilitar la gestión de claves, y el grupo puede ser considerado como un sólo par de claves.

Dentro de las funciones de GnuPG se incluyen generar un par de claves, intercambiar y comprobar la autenticidad de claves, cifrar y descifrar documentos, etc. Para instalar el paquete GnuPG, usamos:

```
# apt install gnupg
```

La opción de la línea de comandos *–gen-key* se usa para generar un nuevo par de claves primario, mediante:

```
$ gpg –gen-key
```

también podemos pedir –full-generate-key, mediante:

```
$ gpg –full-generate-key
```

GnuPG es capaz de crear varios tipos diferentes de pares de claves, pero debe existir una clave primaria capaz de generar firmas. Al momento de ejecutar el programa, este pide[60] nombre, correo del usuario y contraseña[61], para después generar la clave.

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

[61] Cuanto más larga sea la contraseña, más segura será contra ataques de «fuerza bruta». No hay límite para la longitud de una contraseña, y esta debe ser escogida con sumo cuidado. Desde un punto de vista de seguridad, la contraseña que desbloquea la clave privada es uno de los puntos más débiles en GnuPG (y en otros sistemas de cifrado de clave pública), ya que es la única protección que tiene el usuario si alguien se apodera de su clave privada. Para una contraseña lo ideal es que no se usen palabras de un diccionario, y que se mezclen mayúsculas y minúsculas, dígitos, y otros caracteres. Una buena contraseña es crucial para el uso seguro de GnuPG.

Para poder comunicarse con otros, el usuario debe intercambiar las claves públicas. Para obtener una lista de las claves en el fichero («anillo») de claves públicas, se puede usar la opción de la línea de comandos *–list-keys*, mediante:

```
$ gpg –list-keys
```

también podemos usar:

```
$ gpg –list-secret-keys
$ gpg –list-public-keys
```

Para poder enviar una clave pública a un interlocutor, antes hay que exportarla. Para ello se usará la opción de la línea de comandos *–export*. Es necesario un argumento adicional para poder identificar la clave pública que se va a exportar, por ejemplo:

```
$ gpg –output antonio.gpg –export ant@na.mx
```

La clave se exporta en formato binario, y esto puede no ser conveniente cuando se envía la clave por correo electrónico o se publica en una página Web. Por tanto, GnuPG ofrece una opción de la línea de comandos *–armor* que fuerza que la salida de la orden sea generada en formato *armadura-ASCII*, parecido a los documentos codificados con *UUEncode*. Por regla general, cualquier salida de una orden de GnuPG, v.g. claves, documentos cifrados y firmas, pueden ir en formato *armadura-ASCII* añadiendo a la orden la opción *–armor*, por ejemplo:

```
$ gpg –armor –output antonio.gpg –export ant@na.mx
```

Cada clave pública y privada tiene un papel específico en el cifrado y descifrado de documentos. Se puede pensar en una clave pública como en una caja fuerte de seguridad. Cuando un remitente cifra un documento usando una clave pública, ese documento se pone en la caja fuerte, la caja se cierra, y el bloqueo de la combinación de esta se gira varias veces. La parte correspondiente a la clave privada, esto es, el destinatario, es la combinación que puede volver a abrir la caja y retirar el documento. Dicho de otro modo, sólo la persona que posee la clave privada puede recuperar un documento cifrado usando la clave pública asociada al cifrado.

Con este modelo mental se ha mostrado el procedimiento de cifrar y descifrar documentos de un modo muy simple. Si el usuario quisiera cifrar un mensaje para Javier, lo haría usando la clave pública de Javier, y lo descifraría con su propia clave privada. Si Javier quisiera enviar un mensaje al usuario, lo haría con la clave pública del usuario, y este lo descifraría con su propia clave privada.

Cifrar un Archivo para cifrar un archivo se usa la opción *–encrypt*. El usuario debe tener las claves públicas de los pretendidos destinatarios. El programa espera recibir como entrada el nombre del archivo que se desea cifrar o, si este se omite, una entrada estándar. El resultado cifrado se coloca en la salida estándar o donde se haya especificado mediante la opción *–output*. El archivo se comprime como medida adicional de seguridad, aparte de cifrarlo, por ejemplo:

$ gpg –output doc.gpg –encrypt –recipient ant@na.mx doc

La opción *–recipient* se usa una vez para cada destinatario, y lleva un argumento extra que específica la clave pública con la que sería cifrado el archivo. El archivo cifrado sólo puede ser descifrado por alguien con una clave privada que complemente una de las claves públicas de los destinatarios. El usuario, en este caso el remitente, no podría descifrar un archivo cifrado por sí mismo a menos que haya incluido su propia clave pública en la lista de destinatarios.

Descifrar un Archivo para descifrar un archivo se usa la opción *–decrypt*. Para ello es necesario poseer la clave privada para la que el archivo ha sido cifrado. De igual modo que en el proceso de cifrado, el archivo a descifrar es la entrada, y el resultado descifrado la salida, por ejemplo:

$ gpg –output doc –decrypt doc.gpg

Cifrar y Descifrar Usando Cifrado Simétrico también es posible cifrar archivos sin usar criptografía de clave pública. En su lugar, se puede usar sólo una clave de cifrado simétrico para cifrar el archivo. La clave que se usa para el cifrado simétrico deriva de la contraseña dada en el momento de cifrar el documento, y por razones de seguridad, no debe ser la misma contraseña que se esta usando para proteger la clave privada. El cifrado simétrico es útil

para asegurar archivos cuando no sea necesario dar la contraseña a otros. Un archivo puede ser cifrado con una clave simétrica usando la opción –*symmetric*, por ejemplo:

```
$ gpg –symmetric doc
```

o

```
$ gpg -c doc
```

y podemos descifrar usando:

```
$ gpg doc.gpg
```

Otras Opciones para Cifrar y Descifrar

ccrypt es otra herramienta para cifrar y descifrar archivos basada en el cifrado de bloque Rijndael, Se cree que *ccrypt* (basada en el cifrado de bloque *Rijndael*) proporciona una seguridad fuerte. Tengamos en cuenta que *ccrypt* elimina el archivo original (cifra el archivo en su lugar), no cambia significativamente el tamaño del archivo y no altera la fecha/hora del archivo para reflejar la hora en que se realizó el cifrado.

```
$ ccrypt -e archivo
```

para descifrar usamos:

```
$ ccrypt -d archivo.cpt
```

mcrypt el comando permite cifrar y descifrar archivos, cuando cifra deja el archivo original intacto y cambia los permisos del archivo para que el archivo cifrado solo proporcione permisos de acceso de lectura y escritura al propietario del archivo. Ofrece muchas opciones con respecto a los algoritmos de cifrado y también ofrece opciones para comprimir los archivos antes del cifrado (opciones -z comprime GZip y -p comprime Bz2). Puede funcionar con varios archivos, pero los cifra por separado, ejemplo:

$ mcrypt archivo

para descifrar usamos:

$ mcrypt -d archivo.nc

para enumerar los algoritmos de cifrado disponibles, usamos:

$ *mycrypt –list*,

El comando *mcrypt* parece usar *Rijndael-128* como su algoritmo de cifrado predeterminado.

age este comando permite cifrar usando una clave pública o frase, no viene instalado por omisión en el sistema, pero lo podemos instalar usando:

apt install age

Para generar una clave pública en el archivo llave.txt usamos:

$ age-keygen -o llave.txt

Para cifrar un archivo *archivo.tar* con clave pública usamos:

$ tar cvz archivos | age -r llave.txt > archivo.tar.gz.age

para descifrarlo usamos:

$ age –decrypt -i llave.txt > archivo.tar.gz

que nos retornará a nuestro archivo original.
Para cifrar un archivo *archivo.tar* con frase usamos:

$ age –passphrase –output archivo.tar.gz.age archivo.tar.gz

para descifrarlo usamos:

$ age –decrypt –output archivo.tar.gz archivo.tar.gz.age

que nos retornará a nuestro archivo original.

Generar Contraseñas Para generar contraseñas disponemos de múltiples opciones, algunas de ellas son:

- Podemos usar GnuPG, para instalarlo usamos:

  # apt install gnupg

  para generar una contraseña de 32 caracteres, usamos:

  $ gpg –gen-random –armor 1 32

- Podemos usar OpenSSL, para instalarlo usamos:

  # apt install openssl

  para generar una contraseña de 32 caracteres mediante:

  $ openssl rand -base64 32

- Podemos usar APG, para instalarlo usamos:

  # apt install apg

  para generar una contraseña de mínimo 31 y máximo 32 caracteres y genere 4 claves:

  $ apg -n 4 -m 31 -x 32 -a 1

- Podemos usar PWGEN, para instalarlo usamos:

  # apt install pwgen

  para generar una contraseña de 32 caracteres:

  $ pwgen -ys 32 1

- Podemos usar Makepasswd, para instalarlo usamos:

  # apt install makepasswd

  para generar una contraseña de 32 caracteres:

  $ makepasswd –char 32

Cifrado Avanzado En los procesadores modernos es común encontar el motor Intel/AMD Advanced Encryption Standard (AES) o New Instructions (AES-NI) que permite el cifrado y descifrado por Hardware de alta velocidad para el cifrado de disco completo, OpenSSL, ssh, VPN, Linux / Unix / OSX y más. ¿Cómo verifico la compatibilidad con Intel/AMD AES-NI en mi sistema basado en Linux, incluido OpenSSL?

El conjunto de instrucciones del estándar de cifrado avanzado y las nuevas instrucciones del estándar de cifrado avanzado permiten que Intel/AMD específicas y otras CPU realicen un cifrado y descifrado de hardware extremadamente rápido.

Tengamos en cuenta que la compatibilidad con AES-NI se habilita automáticamente si el procesador detectado lo soporta. Para obtener una lista de procesadores que admiten el motor AES-NI, consulte el sitio y la documentación del procesador. El AES-NI es una extensión de la arquitectura del conjunto de instrucciones x86 para microprocesadores de Intel y AMD. Aumenta la velocidad de las aplicaciones que realizan el cifrado y el descifrado mediante AES.

Uno puede descubrir que el procesador tiene el conjunto de instrucciones AES / AES-NI usando el comando:

```
$ lscpu
```

o

```
$ grep -o aes / proc / cpuinfo
```

o

```
$ grep -m1 -o aes / proc / cpuinfo
```

o

```
# cpuid | grep -i aes | sort | uniq
```

En cualquiera de los casos si aparece la bandera *aes* el conjunto de instrucciones AES / AES-NI está presente y activo.

También podemos usar el siguiente comando para verificar la compatibilidad con AES-NI en nuestro procesador:

```
$ sort -u /proc/crypto | grep module
```

Ahora que hemos verificado la compatibilidad, es hora de probarla, para ello usamos:

$ openssl engine

ejemplo de una salida que soporta AES:

(padlock) VIA PadLock (no-RNG, no-ACE)
(dynamic) Dynamic engine loading support

ejemplo de una salida que soporta AES-NI:

(aesni) Intel AES-NI engine
(dynamic) Dynamic engine loading support

y podemos probar su desempeño entre un equipo que soporte el cifrado:

$ dd if=/dev/zero count=1000 bs=1M | ssh -l usuario -c aes128-cbc maquina "cat >/dev/null"

1000+0 records in

1000+0 records out

1048576000 bytes (1.0 GB) copied, 10.6691 s, 98.3 MB/s

y uno que no la soporte:

$ dd if=/dev/zero count=1000 bs=1M | ssh -l usuario -c aes128-cbc maquina "cat >/dev/null"

1000+0 records in

1000+0 records out

1048576000 bytes (1.0 GB) copied, 31.6675 s, 33.1 MB/s

¿Cómo puedo comparar el rendimiento de OpenSSL? podemos correr los comandos:

```
$ openssl speed
```

o

```
$ openssl speed aes-128-cbc
```

Para la última versión de OpenSSL, podemos probar los dos comandos, el segundo comando debería tener números más altos que el primero gracias a EntropyZer0:

```
$ openssl speed aes-256-cbc
$ openssl speed -evp aes-256-cbc
```

3.7 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.

3.8 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*[62] (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

[62] 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
```

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.

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[63] utilizando la orden ((...)) y la sintaxis de variables $[...] de la siguiente manera:

  asigna el valor entero 55 a la variable VAR.

  ```
  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
  ```

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

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[64]:

```
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:

```
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,}
```

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

- 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[65]

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"

y si deseamos limpiar una variable usamos:

$ unset var

Podemos también definir variables de solo lectura, usando:

$ readonly var=valor

Observación 1 *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).*

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

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 2 *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
```

3.9 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 3.2):

```
#!/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
```

4 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[66], 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.

[66] 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.

4.1 Software Propietario

No existe consenso sobre el término a utilizar para referirse al opuesto del Software libre. La expresión «Software propietario (Proprietary Software)» (véase [11]), en la lengua anglosajona, "Proprietary" significa «poseído o controlado privadamente (Privately Owned and Controlled)», que destaca la manutención de la reserva de derechos sobre el uso, modificación o redistribución del Software. Inicialmente utilizado, pero con el inconveniente de que la acepción proviene de una traducción literal del inglés, no correspondiendo su uso como adjetivo en el español, de manera que puede ser considerado como un barbarismo.

El término "propietario" en español resultaría inadecuado, pues significa que «tiene derecho de propiedad sobre una cosa», por lo que no podría calificarse de "propietario" al Software, porque éste no tiene propiedad sobre nada (es decir, no es dueño de nada) y además, no podría serlo (porque es una cosa y no una persona). Así mismo, la expresión "Software propietario" podría ser interpretada como: "Software sujeto a propiedad" (derechos o titularidad) y su opuesto, el Software libre, también está sujeto al derecho de autor. Otra interpretación es que contrariamente al uso popular del término, se puede

afirmar que "todo Software es propietario", por lo que la forma correcta de referirse al Software con restricciones de uso, estudio, copia o mejora es la de Software privativo, según esta interpretación el término "propietario" podría aplicarse tanto para Software libre como Software privativo, ya que la diferencia entre uno y otro está en que el dueño del Software privativo lo licencia como propiedad privada y el de Software libre como propiedad social.

Con la intención de corregir el defecto de la expresión "Software propietario" aparece el llamado "Software con propietario", sin embargo se argumenta contra el término "con propietario" y justamente su similitud con Proprietary en inglés, que sólo haría referencia a un aspecto del Software que no es libre, manteniendo una de las principales críticas a éste (de "Software sujeto a derechos" o "propiedad"). Adicionalmente, si "propietario" se refiere al titular de los derechos de autor -y está claro que no se puede referir al usuario, en tanto éste es simplemente un cesionario-, no resuelve la contradicción: todo el Software libre tiene también titulares de derechos de autor.

La expresión Software no libre (en inglés Non-Free Software) es usado por la FSF para agrupar todo el Software que no es libre, es decir, incluye al llamado en inglés "Semi-Free Software" (Software semilibre) y al "Propietary Software". Asimismo, es frecuentemente utilizado para referirse al Software que no cumple con las Directrices de Software libre de Debian GNU/Linux, las cuales siguen la misma idea básica de libertad en el Software, propugnada por la FSF y sobre las cuales está basada la definición de código abierto de la Open Source Initiative.

Adicionalmente el Software de código cerrado nace como antónimo de Software de código abierto y por lo tanto se centra más en el aspecto de ausencia de acceso al código que en los derechos sobre el mismo, éste se refiere sólo a la ausencia de una sola libertad por lo que su uso debe enfocarse sólo a este tipo de Software y aunque siempre signifique que es un Software que no es libre, no tiene que ser Software de código cerrado.

La expresión Software privado es usada por la relación entre los conceptos de tener y ser privado. Este término sería inadecuado debido a que, en una de sus acepciones, la palabra "privado" se entiende como antónimo de "público", es decir, que «no es de propiedad pública o estatal, sino que pertenece a particulares», provocando que esta categoría se interpretará como no referente al Estado, lo que produciría la exclusión del Software no libre generado por el aparato estatal. Además, el "Software público" se asocia generalmente con Software de dominio público.

4.2 Software Libre

La definición de Software libre (véase [13], [2], [9], [10], [8] y [12]) estipula los criterios que se tienen que cumplir para que un programa sea considerado libre. De vez en cuando se modifica esta definición para clarificarla o para resolver problemas sobre cuestiones delicadas. «Software libre» significa que el Software respeta la libertad de los usuarios y la comunidad. En términos generales, los usuarios tienen la libertad de copiar, distribuir, estudiar, modificar y mejorar el Software. Con estas libertades, los usuarios -tanto individualmente como en forma colectiva- controlan el programa y lo que hace.

Cuando los usuarios no controlan el programa, el programa controla a los usuarios. Los programadores controlan el programa y a través del programa, controlan a los usuarios. Un programa que no es libre, llamado «privativo o propietario», es considerado por muchos como un instrumento de poder injusto.

El Software libre es la denominación del Software que respeta la libertad de todos los usuarios que adquirieron el producto y por tanto, una vez obtenido el mismo puede ser usado, copiado, estudiado, modificado y redistribuido libremente de varias formas. Según la Free Software Foundation (véase [13]), el Software libre se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir y estudiar el mismo, e incluso modificar el Software y distribuirlo modificado.

Un programa es Software libre si los usuarios tienen las cuatro libertades esenciales:

0. La libertad de usar el programa, con cualquier propósito.

1. La libertad de estudiar cómo funciona el programa y modificarlo, adaptándolo a tus necesidades.

2. La libertad de distribuir copias del programa, con lo cual puedes ayudar a tu prójimo.

3. La libertad de mejorar el programa y hacer públicas esas mejoras a los demás, de modo que toda la comunidad se beneficie.

Un programa es Software libre si los usuarios tienen todas esas libertades. Por tanto, el usuario debe ser libre de redistribuir copias, tanto con o sin modificaciones, ya sea gratuitamente o cobrando una tarifa por la distribución,

a cualquiera en cualquier parte. El ser libre de hacer estas cosas significa, entre otras cosas, que no tiene que pedir ni pagar el permiso.

También debe tener la libertad de hacer modificaciones y usarlas en privado para su propio trabajo o pasatiempo, sin siquiera mencionar que existen. Si publica sus cambios, no debe estar obligado a notificarlo a nadie en particular, ni de ninguna manera.

La libertad de ejecutar el programa significa que cualquier tipo de persona u organización es libre de usarlo en cualquier tipo de sistema de computación, para cualquier tipo de trabajo y finalidad, sin que exista obligación alguna de comunicarlo al programador ni a ninguna otra entidad específica. En esta libertad, lo que importa es el propósito de los usuarios, no el de los programadores. El usuario es libre de ejecutar el programa para alcanzar sus propósitos y si lo distribuye a otra persona, también esa persona será libre de ejecutarlo para lo que necesite; nadie tiene derecho a imponer sus propios objetivos.

La libertad de redistribuir copias debe incluir las formas binarias o ejecutables del programa, así como el código fuente, tanto para las versiones modificadas como para las que no lo estén. Distribuir programas en forma de ejecutables es necesario para que los sistemas operativos libres se puedan instalar fácilmente. Resulta aceptable si no existe un modo de producir un formato binario o ejecutable para un programa específico, dado que algunos lenguajes no incorporan esa característica, pero debe tener la libertad de redistribuir dichos formatos si encontrara o programara una forma de hacerlo.

Para que se de la libertad que se menciona en los puntos 1 y 3 de realizar cambios y publicar las versiones modificadas tenga sentido, el usuario debe tener acceso al código fuente del programa. Por consiguiente, el acceso al código fuente es una condición necesaria para el Software libre. El «código fuente» compilado no es código fuente real y no cuenta como código fuente.

La libertad 1 incluye la libertad de usar su versión modificada en lugar de la original. Si el programa se entrega con un producto diseñado para ejecutar versiones modificadas de terceros, pero rechaza ejecutar las suyas, una práctica conocida como «tivoización» o «arranque seguro» [«Lockdown»] la libertad 1 se convierte más en una ficción teórica que en una libertad práctica, esto no es suficiente, en otras palabras, estos binarios no son Software libre, incluso si se compilaron desde un código fuente que es libre.

Una manera importante de modificar el programa es agregándole subrutinas y módulos libres ya disponibles. Si la licencia del programa especifica que no se pueden añadir módulos que ya existen y que están bajo una licencia

apropiada, por ejemplo si requiere que usted sea el titular de los derechos de autor del código que desea añadir, entonces se trata de una licencia demasiado restrictiva como para considerarla libre.

La libertad 3 incluye la libertad de publicar sus versiones modificadas como Software libre. Una licencia libre también puede permitir otras formas de publicarlas; en otras palabras, no tiene que ser una licencia de Copyleft. No obstante, una licencia que requiera que las versiones modificadas no sean libres, no se puede considerar libre.

«Software libre» no significa que «no es comercial». Un programa libre debe estar disponible para el uso comercial, la programación comercial y la distribución comercial. La programación comercial de Software libre ya no es inusual; el Software libre comercial es muy importante, ejemplo de ello es la empresa RedHat (ahora propiedad de IBM). Puede haber pagado dinero para obtener copias de Software libre, o puede haber obtenido copias sin costo. Pero sin tener en cuenta cómo obtuvo sus copias, siempre tiene la libertad de copiar y modificar el Software, incluso de vender copias.

El término Software no libre se emplea para referirse al Software distribuido bajo una licencia de Software más restrictiva que no garantiza estas cuatro libertades. Las leyes de la propiedad intelectual reservan la mayoría de los derechos de modificación, duplicación y redistribución para el dueño del Copyright; el Software dispuesto bajo una licencia de Software libre rescinde específicamente la mayoría de estos derechos reservados.

Los manuales de Software deben ser libres por las mismas razones que el Software debe ser libre y porque de hecho los manuales son parte del Software. También tiene sentido aplicar los mismos argumentos a otros tipos de obras de uso práctico, es decir, obras que incorporen conocimiento útil, tal como publicaciones educativas y de referencia. Wikipedia es el ejemplo más conocido.

La lista de proyectos de este tipo es realmente impresionante, algunos han conseguido un uso y alta calidad, por ejemplo el compilador GCC, el Kernel de Linux y el sistema operativo Debian GNU/Linux y Android. Mientras que otros proyectos han caído en el olvido, pero de la gran mayoría se tiene copia del código fuente que permitiría a quienes estén interesados en dicho proyecto poder reusarlo y en su caso ampliarlo.

La característica más importante que aparece típicamente en un proyecto de este tipo, es que un conjunto de personas separadas a gran distancia, sean capaces, a través de la Web, de los E-mail y de foros de aunar sus esfuerzos para crear, mejorar y distribuir un producto, de forma que todos

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

Si bien, el Software libre no es más seguro (en el sentido de invulnerable) que el propietario, la diferencia estriba en que el código fuente en el Software libre está disponible para todos y cualquiera puede aportar una solución, y por lo general al poco tiempo de detectarse una vulnerabilidad (a veces en cuestión de horas) se puede disponer de una solución para la misma. Además, al tener acceso al código fuente se puede detectar fácilmente si alguien introdujo código malicioso a una determinada aplicación.

¿Por qué se Interesan los Autores, Alumnos y Profesores Universitarios en el Software Libre? La ventaja principal es porque bajo el Software libre subyace la idea de compartir conocimiento y favorecer la existencia de nuevas ideas[67]; 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

[67] ¿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.

4.3 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.

4.4 Tipos de Licencias

Tanto la Open Source Initiative como la Free Software Foundation mantienen en sus páginas Web (véase [13], [2], y [12]) listados oficiales de las licencias de Software libre que aprueban.

Una licencia es aquella autorización formal con carácter contractual que un autor de un Software da a un interesado para ejercer "actos de explotación legales". Pueden existir tantas licencias como acuerdos concretos se den entre el autor y el licenciatario. Desde el punto de vista del Software libre, existen distintas variantes del concepto o grupos de licencias:

Licencias GPL Una de las más utilizadas es la Licencia Pública General de GNU (GNU GPL). El autor conserva los derechos de autoría (Copyright), y permite la redistribución y modificación bajo términos diseñados para asegurarse de que todas las versiones modificadas del Software permanecen bajo los términos más restrictivos de la propia GNU GPL. Esto hace que sea imposible crear un producto con partes no licenciadas GPL, el conjunto tiene que ser GPL.

En la práctica, esto hace que las licencias de Software libre se dividan en dos grandes grupos, aquellas que pueden ser mezcladas con código licenciado bajo GNU GPL (y que inevitablemente desaparecerán en el proceso, al ser el código resultante licenciado bajo GNU GPL) y las que no lo permiten al incluir mayores u otros requisitos que no contemplan ni admiten la GNU GPL y que por lo tanto no pueden ser enlazadas ni mezcladas con código gobernado por la licencia GNU GPL.

GPL Versión 1 la versión 1 de GNU GPL, fue presentada el 25 de febrero de 1989, impidió lo que eran las dos principales formas con las que los distribuidores de Software restringían las libertades definidas por el Software libre. El primer problema fue que los distribuidores publicaban únicamente los archivos binarios, funcionales y ejecutables, pero no entendibles o modificables por humanos. Para prevenir esto, la GPLv1 estableció que cualquier proveedor de Software libre además de distribuir el archivo binario debía liberar a su vez código fuente entendible y que pudiera ser modificado por el ser humano bajo la misma licencia (secciones 3a y 3b de la licencia).

El segundo problema era que los distribuidores podían añadir restricciones adicionales, añadiendo restricciones a la licencia o mediante la combinación del Software con otro que tuviera otras restricciones en su distribución. Si

esto se hacía, entonces la unión de los dos conjuntos de restricciones sería aplicada al trabajo combinado, entonces podrían añadirse restricciones inaceptables. Para prevenir esto, GPLv1 obligaba a que las versiones modificadas en su conjunto, tuvieran que ser distribuidas bajo los términos GPLv1 (secciones 2b y 4 de la licencia). Por lo tanto, el Software distribuido bajo GPLv1 puede ser combinado con Software bajo términos más permisivos y no con Software con licencias más restrictivas, lo que entraría en conflicto con el requisito de que todo Software tiene que ser distribuido bajo los términos de la GPLv1.

GPL Versión 2 según Richard Stallman, el mayor cambio en GPLv2 fue la cláusula "Liberty or Death" («libertad o muerte»). Esta sección dice que si alguien impone restricciones que le prohíben distribuir código GPL de tal forma que influya en las libertades de los usuarios (por ejemplo, si una ley impone que esa persona únicamente pueda distribuir el Software en binario), esa persona no puede distribuir Software GPL. La esperanza es que esto hará que sea menos tentador para las empresas el recurrir a las amenazas de patentes para exigir una remuneración de los desarrolladores de Software libre.

En 1991 se hizo evidente que una licencia menos restrictiva sería estratégicamente útil para la biblioteca C y para las bibliotecas de Software que esencialmente hacían el trabajo que llevaban a cabo otras bibliotecas comerciales ya existentes. Cuando la versión 2 de GPL fue liberada en junio de 1991, una segunda licencia Library General Public License fue introducida al mismo tiempo y numerada con la versión 2 para denotar que ambas son complementarias. Los números de versiones divergieron en 1999 cuando la versión 2.1 de LGPL fue liberada, esta fue renombrada como GNU Lesser General Public License para reflejar su lugar en esta filosofía.

GPL Versión 3 A finales de 2005, la Free Software Foundation (FSF) anunció estar trabajando en la versión 3 de la GPL (GPLv3). El 16 de enero de 2006, el primer borrador de GPLv3 fue publicado y se inició la consulta pública. La consulta pública se planeó originalmente para durar de nueve a quince meses, pero finalmente se extendió a dieciocho meses, durante los cuales se publicaron cuatro borradores. La GPLv3 oficial fue liberada por la FSF el 29 de junio de 2007.

Según Stallman los cambios más importantes se produjeron en el campo

de las patentes de Software, la compatibilidad de licencias de Software libre, la definición de código fuente y restricciones a las modificaciones de Hardware. Otros cambios están relacionados con la internacionalización, cómo son manejadas las violaciones de licencias y cómo los permisos adicionales pueden ser concedidos por el titular de los derechos de autor. También añade disposiciones para quitar al DRM su valor legal, por lo que es posible romper el DRM (Digital Rights Management) en el Software de GPL sin romper leyes como la DMCA (Digital Millennium Copyright Act).

GPLv2 vs GPL v3 GPLv3 contiene la intención básica de GPLv2 y es una licencia de código abierto con un Copyleft estricto. Sin embargo, el idioma del texto de la licencia fue fuertemente modificado y es mucho más completo en respuesta a los cambios técnicos, legales y al intercambio internacional de licencias.

La nueva versión de la licencia contiene una serie de cláusulas que abordan preguntas que no fueron o fueron cubiertas de manera insuficiente en la versión 2 de la GPL. Las nuevas regulaciones más importantes son las siguientes:

- GPLv3 contiene normas de compatibilidad que hacen que sea más fácil combinar el código GPL con el código que se publicó bajo diferentes licencias. Esto se refiere en particular al código bajo la licencia de Apache v. 2.0.

- Se insertaron normas sobre gestión de derechos digitales para evitar que el Software GPL se modifique a voluntad, ya que los usuarios recurrieron a las disposiciones legales para protegerse mediante medidas técnicas de protección (como la DMCA o la directiva sobre derechos de autor).

- La licencia GPLv3 contiene una licencia de patente explícita, según la cual las personas que licencian un programa bajo licencia GPL otorgan derechos de autor y patentes, en la medida en que esto sea necesario para utilizar el código que ellos otorgan. Por lo tanto, no se concede una licencia de patente completa. Además, la nueva cláusula de patente intenta proteger al usuario de las consecuencias de los acuerdos entre los titulares de patentes y los licenciatarios de la licencia pública general que solo benefician a algunos de los licenciatarios (correspondientes al

acuerdo Microsoft / Novell). Los licenciatarios deben garantizar que todos los usuarios disfrutan de tales ventajas (licencia de patente o liberación de reclamos) o que nadie puede beneficiarse de ellos.

- A diferencia de la GPLv2, la GPLv3 establece claramente que no es necesario divulgar el código fuente en un uso ASP (Application Service Provider) de los programas GPL, siempre que no se envíe una copia del Software al cliente. Si el efecto Copyleft debe extenderse al uso de ASP, debe aplicarse la Licencia pública general de Affero, versión 3 (AGPL) que solo difiere de la GPLv3 en esta consideración.

Licencias Estilo BSD Llamadas así porque se utilizan en gran cantidad de Software distribuido junto a los sistemas operativos BSD. El autor, bajo tales licencias, mantiene la protección de Copyright únicamente para la renuncia de garantía y para requerir la adecuada atribución de la autoría en trabajos derivados, pero permite la libre redistribución y modificación, incluso si dichos trabajos tienen propietario. Son muy permisivas, tanto que son fácilmente absorbidas al ser mezcladas con la licencia GNU GPL con quienes son compatibles. Puede argumentarse que esta licencia asegura "verdadero" Software libre, en el sentido que el usuario tiene libertad ilimitada con respecto al Software y que puede decidir incluso redistribuirlo como no libre.

Licencia Copyleft Hay que hacer constar que el titular de los derechos de autor (Copyright) de un Software bajo licencia Copyleft puede también realizar una versión modificada bajo su Copyright original y venderla bajo cualquier licencia que desee, además de distribuir la versión original como Software libre. Esta técnica ha sido usada como un modelo de negocio por una serie de empresas que realizan Software libre (por ejemplo MySQL); esta práctica no restringe ninguno de los derechos otorgados a los usuarios de la versión Copyleft.

Licencia estilo MIT Las licencias MIT son de las más permisivas, casi se consideran Software de dominio público. Lo único que requieren es incluir la licencia MIT para indicar que el Software incluye código con licencia MIT.

Licencia Apache License La licencia Apache trata de preservar los derechos de autor, incluir la licencia en el Software distribuido y una lista de

los cambios realizados. En modificaciones extensivas del Software original permite licenciar el Software bajo otra licencia sin incluir esas modificaciones en el código fuente.

Licencia Mozilla Public License MPL Esta licencia requiere que los archivos al ser distribuidos conserven la misma licencia original pero pueden ser usados junto con archivos con otra licencia, al contrario de la licencia GPL que requiere que todo el código usado junto con código GPL sea licenciado como código GPL. También en caso de hacer modificaciones extensivas permite distribuirlas bajo diferentes términos y sin incluir el código fuente en las modificaciones.

Licencia Código de Dominio Público Es un código que no está sujeto a derechos de autor que puede utilizarse sin restricciones.

Licencia Creative Commons Las licencias de Creative Commons son más utilizadas para cualquier creación digital que para el Software, entendiendo como creación digital desde fotos, artículos en blogs, música, vídeos, este trabajo, etc. Hay varios tipos de licencias de Creative Commons diferenciando entre permitir modificaciones a la obra original, solicitando crédito de la creación o permitiendo un uso comercial de la obra.

Licencias de Código Abierto Las licencias de código abierto son un intermedio entre las licencias privativas y las licencias de Software libre. Las licencias de código abierto permiten el acceso al código fuente pero no todas se consideran licencias de Software libre al no otorgar otros derechos que se requieren para considerar un Software como Software libre como el derecho al uso o con cualquier propósito, modificación y distribución.

Dado el éxito del Software libre como modelo de desarrollo de Software algunas empresas cuyo Software era privativo pueden decidir hacerlo de código abierto con la intención de suplir algunas carencias de Software privativo pero sin perder ciertos derechos que son la fuente de sus ingresos como la venta de licencias.

Las expresiones «Software libre» y «código abierto» se refieren casi al mismo conjunto de programas. No obstante, dicen cosas muy diferentes acerca de dichos programas, basándose en valores diferentes. El movimiento del Software libre defiende la libertad de los usuarios de ordenadores, en un

movimiento en pro de la libertad y la justicia. Por contra, la idea del código abierto valora principalmente las ventajas prácticas y no defiende principios. Esta es la razón por la que gran parte de la comunidad de Software libre está en desacuerdo con el movimiento del código abierto y nosotros no empleamos esta expresión en este texto.

Licencia Microsoft Public License La Microsoft Public License es una licencia de código abierto que permite la distribución del Software bajo la misma licencia y la modificación para un uso privado. Tiene restricciones en cuanto a las marcas registradas.

En caso de distribuir el Software de forma compilada o en forma de objeto binario no se exige proporcionar los derechos de acceso al código fuente del Software compilado o en forma de objeto binario. En este caso esta licencia no otorga más derechos de los que se reciben, pero si permite otorgar menos derechos al distribuir el Software (compilado o en forma de objeto binario).

Modelo de Desarrollo de Software Bazar y Catedral El tipo de licencia no determina qué Software es mejor o peor, si el privativo o el Software libre, la diferencia entre las licencias está en sus características éticas y legales. Aunque el modelo de desarrollo con una licencia de código abierto a la larga suele tener un mejor desarrollo y éxito que el Software privativo, más aún con un medio como internet que permite colaborar a cualquier persona independiente de donde esté ubicada en el mundo.

Comparación con el Software de Código Abierto Aunque en la práctica el Software de código abierto y el Software libre comparten muchas de sus licencias, la Free Software Foundation opina que el movimiento del Software de código abierto es filosóficamente diferente del movimiento del Software libre. Los defensores del término «código abierto (Open Source)», afirman que éste evita la ambigüedad del término en ese idioma que es «Free» en «Free Software».

Mucha gente reconoce el beneficio cualitativo del proceso de desarrollo de Software cuando los desarrolladores pueden usar, modificar y redistribuir el código fuente de un programa. El movimiento del Software libre hace especial énfasis en los aspectos morales o éticos del Software, viendo la excelencia técnica como un producto secundario de su estándar ético. El movimiento de código abierto ve la excelencia técnica como el objetivo prioritario, siendo

el compartir el código fuente un medio para dicho fin. Por dicho motivo, la FSF se distancia tanto del movimiento de código abierto como del término «código abierto (Open Source)».

Puesto que la «iniciativa de Software libre Open Source Initiative (OSI)» sólo aprueba las licencias que se ajustan a la «definición de código abierto (Open Source Definition)», la mayoría de la gente lo interpreta como un esquema de distribución, e intercambia libremente "código abierto" con "Software libre". Aún cuando existen importantes diferencias filosóficas entre ambos términos, especialmente en términos de las motivaciones para el desarrollo y el uso de tal Software, raramente suelen tener impacto en el proceso de colaboración.

Aunque el término "código abierto" elimina la ambigüedad de libertad frente a precio (en el caso del inglés), introduce una nueva: entre los programas que se ajustan a la definición de código abierto, que dan a los usuarios la libertad de mejorarlos y los programas que simplemente tienen el código fuente disponible, posiblemente con fuertes restricciones sobre el uso de dicho código fuente. Mucha gente cree que cualquier Software que tenga el código fuente disponible es de código abierto, puesto que lo pueden manipular, sin embargo, mucho de este Software no da a sus usuarios la libertad de distribuir sus modificaciones, restringe el uso comercial, o en general restringe los derechos de los usuarios.

4.4.1 Licencias Creative Commons

Las Licencias[68] 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:

[68] 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[69], 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.

[69] 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.

4.4.2 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.

4.5 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-.

4.5.1 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.

4.5.2 ¿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.

4.5.3 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.

4.5.4 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.

4.5.5 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.

4.6 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.

4.6.1 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.

4.6.2 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.

5 Apéndice B: El Sistema Operativo GNU/Linux

GNU/Linux se ve y se siente muy parecido a cualquier otro sistema UNIX, y de hecho la compatibilidad con UNIX ha sido una importante meta del diseño del proyecto Linux. No obstante, Linux es mucho más joven que la mayor parte de los sistemas UNIX. Su desarrollo se inició en 1991, cuando un estudiante finlandés, Linus Torvalds, escribió y bautizó un pequeño pero autosuficiente núcleo para el procesador 80386, el primer procesador de 32 bits verdadero en la gama de CPU compatibles con el PC de Intel.

En los albores de su desarrollo, el código fuente de Linux se ofrecía gratuitamente en internet. En consecuencia, su historia ha sido una colaboración de muchos usuarios de todo el mundo que se han comunicado casi exclusivamente a través de internet. Desde un núcleo inicial que implementaba parcialmente un subconjunto pequeño de los servicios de UNIX, Linux ha crecido para incluir cada vez más funcionalidades UNIX.

En sus inicios, el desarrollo de Linux giraba en gran medida alrededor del núcleo del sistema operativo central: el ejecutivo privilegiado que administra todos los recursos del sistema e interactúa directamente con el Hardware. Desde luego, se requiere mucho más que este núcleo para producir un sistema operativo completo.

Resulta útil hacer la distinción entre el núcleo (Kernel) de Linux y un sistema Linux: el núcleo en Linux es una identidad de Software totalmente original desarrollada desde cero por la comunidad Linux (suele encontrarse en el directorio */boot* en el sistema de archivos); el sistema Linux, tal como lo conocemos hoy, incluye una multitud de componentes, algunos escritos desde cero, otros tomados en préstamo de otros proyectos o creados en colaboración con otros equipos como el proyecto GNU de la Free Software Foundation.

El sistema Linux básico es un entorno estándar para aplicaciones y programación de los usuarios, pero no obliga a adoptar mecanismos estándar para controlar las funcionalidades disponibles como un todo.

GNU/Linux en sus inicios trabajaba en modo consola o terminal -línea de comandos o *Shell*- usando la interfaz de línea de comando (CLI por Command Line Interface) o interfaz de usuario de terminal (TUI por Terminal User Interface). A medida que Linux ha madurado, se ha hecho necesaria otra capa de funcionalidad encima del sistema Linux: El servidor de Pantalla

Un servidor de pantalla es un programa cuya tarea principal es coordinar la entrada y salida de sus clientes hacia y desde el resto del sistema operativo, el Hardware y entre sí. El servidor de pantalla se comunica con sus clientes a

través del protocolo del servidor de pantalla. Un servidor de visualización es un componente clave en cualquier interfaz gráfica de usuario, específicamente el sistema de ventanas. Es el componente básico de la interfaz gráfica de usuario (GUI) que se encuentra entre la interfaz gráfica y el Kernel.

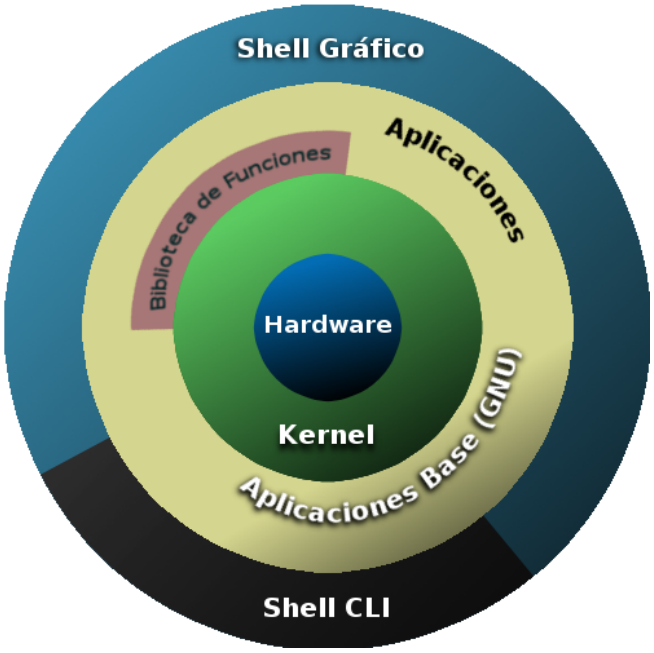

Figura 1: GNU/Linux

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, seria placentero, pero la GUI también tiene magia. No confundir el servidor de visualización con el entorno de escritorio (GNOME, KDE, LXQt, LXDE, Xfce, Unity, MATE, Cinnamon, Pantheon, Deepin, Budgie, PIXEL, Enlightenment, Trinity, Moksha, Ukui, etc.), los entornos de escritorio usan uno, pero en una capa por debajo.

El servidor de pantalla se comunica con sus clientes a través del protocolo del servidor de pantalla. Hay tres protocolos principales de servidor de pantalla disponibles: X11, Mir (desarrollado por Canonical) y Wayland.

X Window System, a menudo denominado simplemente X, es el mas antiguo de todos (1984), terminó siendo el sistema de ventanas predeterminado para la mayoría de los sistemas operativos similares a UNIX, incluido

GNU/Linux. El servidor X.Org es la implementación gratuita y de código abierto del servidor de visualización X Window System administrado por la Fundación X.Org. Es una aplicación que interactúa con aplicaciones cliente a través del protocolo X11 para dibujar cosas en una pantalla y enviar eventos de entrada como movimientos del mouse, Clics y pulsaciones de teclas. Normalmente, uno iniciaría un servidor X que esperará a que las aplicaciones de los clientes se conecten a él. Xorg se basa en un modelo cliente/servidor y, por lo tanto, permite que los clientes se ejecuten de forma local o remota en una máquina diferente. La mayoría de las funciones que proporciona el protocolo X Server ya no se utilizaban. Casi todo el trabajo que hizo X11 se volvió a delegar a las aplicaciones individuales y al administrador de ventanas. Y, sin embargo, todas esas características antiguas siguen ahí, pesando sobre todas estas aplicaciones, perjudicando el rendimiento y la seguridad. Es por ello y otras cosas que nace Wayland[70].

[70]Wayland fue iniciado por Kristian Hogsberg, un desarrollador de X.Org, como un proyecto personal en 2008. Es un protocolo de comunicación que especifica la comunicación entre un servidor de pantalla y sus clientes. Wayland se desarrolla como un proyecto gratuito y de código abierto impulsado por la comunidad con el objetivo de reemplazar el sistema de ventanas X11 o Xorg, por un sistema de ventanas moderno, seguro y más simple.

En Wayland, el compositor es el servidor de visualización. Compositor, es un administrador de ventanas que proporciona a las aplicaciones un búffer fuera de la pantalla para cada ventana. El administrador de ventanas compone los búffers de la ventana en una imagen que representa la pantalla y escribe el resultado en la memoria de visualización. El protocolo Wayland permite al compositor enviar los eventos de entrada directamente a los clientes y permite que el cliente envíe el evento de daños directamente al compositor.

La principal ventaja de Wayland sobre X es que comienza desde cero. Una de las principales razones de la complejidad de X es que, a lo largo de los años, su función ha cambiado. Como resultado, hoy en día, X11 actúa en gran medida como un protocolo de comunicaciones "realmente terrible" entre el cliente y el administrador de ventanas. Wayland también es superior cuando se trata de seguridad. Con X11, es posible hacer Keylogging al permitir que cualquier programa exista en segundo plano y lea lo que está sucediendo con otras ventanas abiertas en el área de X11. Con Wayland, esto simplemente no sucederá, ya que cada programa funciona de forma independiente.

Sin embargo, el sistema X Window todavía tiene muchas ventajas sobre Wayland. Aunque Wayland elimina la mayoría de los defectos de diseño del Xorg, tiene sus propios problemas. A pesar de que el proyecto Wayland ha estado activo durante más de diez años, las cosas no son 100% estables, de ahí que aun sigamos usando Xorg. Aún hoy, la mayoría de los videojuegos y las aplicaciones de gráficos intensivos para Linux todavía se escriben para X11. Además, muchos controladores de gráficos de código cerrado, como los de las GPU NVIDIA, aún no ofrecen soporte completo para Wayland. Desde mi punto de vista aún tendremos Xorg para una década más, aunque ya comiencen a salir algunos

Una distribución de GNU/Linux incluye todos los componentes estándar del sistema Linux, más un conjunto de herramientas administrativas que simplifican la instalación inicial y desinstalación de paquetes del sistema.

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

Hardware en GNU/Linux El soporte del Hardware en Linux es un asunto complicado, es lo que más problemas da. Linux soporta la mayor parte del Hardware, pero a veces pueden existir problemas[71]:

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

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

[71] Mito: Linux/Unix se puede usar para revivir un equipo de cómputo viejo. La realidad es que si bien, hay múltiples distribuciones de Linux/Unix que corren en una gran cantidad de procesadores antiguos y actuales, los Drivers necesarios para reconocer periféricos como tarjetas gráficas, de red alámbrica e inalámbrica, entre muchos otros, no tienen soporte en Linux/Unix, lo cual hará imposible su uso en Linux/Unix. Esto es cierto en cualquier computadora no importa de cuál generación es el equipo de cómputo. La verdad de todo esto, es que los fabricantes están enfocados en producir Hardware y Drivers que corran en los sistemas operativos con mayor cuota de mercado y por el momento Linux/Unix en equipos personales no son de ellos.

La compatibilidad del Hardware depende en gran medida de la versión de Kernel de GNU/Linux instalado, es de esperarse que en versiones anteriores del Kernel cierto Hardware no se pueda detectar, pero lo contrario también pasa, hay Drivers que solo corren correctamente en versiones anteriores del Kernel y no en las últimas versiones, lo que ocasiona que muchos usuarios se desesperen al tratar de usar sus equipos con GNU/Linux. Y en caso de logar que funcione el Hardware, se fuerza a los usuarios a usar una determinada versión del Kernel (y todas las aplicaciones de la distribución) no actualizable, por la imposibilidad de hacer funcionar el Hardware del equipo en una más moderna con la consiguiente obsolescencia del Software instalado en el equipo.

- si es un dispositivo exclusivo para Windows, como los Winmodems (linmodems.org).

Si el Hardware es "de verdad" (no Winmodems), de marca conocida y actual, casi con toda seguridad estará soportado por Linux.

Los instaladores de Linux reconocen prácticamente todo el Hardware durante la instalación, por lo que la mejor manera de evitar problemas con el Hardware es instalarlo desde el principio. Si añadimos Software para nuestro Hardware posteriormente a la instalación nos costará hacerlo funcionar. ¡Incluso puede ser más rápido instalar el sistema desde cero!.

¿Cómo puedo saber si mi Hardware está soportado por Linux (antes de comprarlo y cometer un error de forma irreparable)? Fácil: consultando en internet.

¿Qué sabe Linux de mi Hardware? El Kernel se encarga de la gestión del Hardware usando herramientas como *Udev* (sistema de nombrado del Hardware), *Hotplug* (mecanismo de avisos), *Dbus* (comunicaciones entre procesos) o *Hal* (capa de abstracción de Hardware), y mapea todo el Hardware en archivos de dispositivos ubicados en los directorios */dev* y */sys*.

Algunos comando usados para conocer el Hardware son:

- lscpu - Información de procesador
- lshw - Lista de Hardware en Linux
- hwinfo - Información del Hardware en Linux
- lspci - Lista PCI
- lsscsi - Listar dispositivos SCSI
- lsusb - Lista de los buses usb y detalles del dispositivo
- inxi - Script mega Bash para usuarios no técnicos
- lsblk - Lista de dispositivos de bloque
- df - espacio en disco de los sistemas de archivos

- fdisk - Informa y permite modificar las particiones de disco
- mount - Permite montar y desmontar y ver sistema de archivo montados
- free - Información de la memoria RAM y Swap
- lsmem - Lista los rangos de memoria disponible.
- hdparm - Información de disco duro
- lsmod - Información de los modulos de los drivers cargados al Kernel

Archivos del directorio */proc*, contienen información accesible usando el comando *cat:*

- Información de CPU
- Información del Kernel de Linux
- Dispositivos Sata / SCSI
- Particiones

Componentes de un Sistema GNU/Linux El sistema GNU/Linux se compone de tres cuerpos principales de código, al igual que la mayor parte de las implementaciones de UNIX.

- El núcleo se encarga de mantener todas las abstracciones importantes del sistema operativo, incluidas cosas tales como memoria virtual y procesos.
- Las bibliotecas del sistema definen un conjunto estándar de funciones a través de las cuales las aplicaciones pueden interactuar con el núcleo y que implementan gran parte de la funcionalidad del sistema operativo que no necesita todos los privilegios del código del núcleo.
- Las utilerías del sistema que son programas que realizan tareas de administración especializadas individuales. Algunos programas utilitarios

pueden invocarse una sola vez para asignar valores iniciales y configurar algún aspecto del sistema; otros llamados demonios[72] 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[73]. El sistema de archivos es un árbol invertido con múltiples niveles, que permite a los usuarios crear sus propios subdirectorios. Cada archivo de datos de usuario es tan solo una secuencia de Bytes.

El sistema UNIX/Linux fue diseñado por programadores para programadores; por ello, siempre ha sido interactivo, y las funciones para desarrollar programas siempre han tenido prioridad. Tales recursos incluyen a los programas *make, gcc, git*, etc.

Los archivos de disco y los dispositivos de Entrada/Salida (E/S) se tratan de la manera más similar posible. Así, la dependencia de dispositivos y las peculiaridades se mantienen en el núcleo hasta donde es posible; aún en el núcleo, la mayor parte de ellas están confinadas a los Drivers de los dispositivos.

Un archivo en Unix/Linux es una secuencia de Bytes. Los diferentes programas esperan distintos niveles de estructura, pero el núcleo no impone ninguna estructura a los archivos. Por ejemplo, la convención para los archivos de texto es líneas de caracteres *ASCII* separadas por un solo carácter de nueva línea (que es el carácter de salto de línea en *ASCII*), pero el núcleo nada sabe de esta convención.

Los archivos se organizan en directorios en estructura de árbol. Los directorios también son archivos que contienen información sobre cómo encontrar otros archivos. Un nombre de camino, trayectoria o ruta de un archivo es

[72] 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.

[73] Algunos de los distintos tipos de Shell son: Shell Bourne, Shell Zsh, Shell C, Shell Korn, Shell Bourne-Again (mejor conocido como Bash, Bourne again shell), etc.

una cadena de texto que identifica un archivo especificando una ruta a través de la estructura de directorios hasta el archivo. Sintácticamente, una trayectoria consiste en nombres de archivos individuales separados por el carácter diagonal. Por ejemplo */usr/local/fuente*, la primera diagonal indica la raíz del árbol de directorios, llamado directorio *raíz* o *root*. El siguiente elemento, *usr*, es un subdirectorio de la raíz, *local* es un subdirectorio de *usr* y *fuente* es un archivo o directorio que está en el directorio *local*. No es posible determinar a partir de la sintaxis del nombre de una trayectoria si la *fuente* es un archivo ordinario o un directorio.

Un archivo puede conocerse por más de un nombre en uno o más directorios. Tales nombres múltiples se denominan enlaces, también se manejan enlaces simbólicos, que son archivos que contienen el nombre de una ruta de otro archivo o directorio. Las dos clases de enlaces también se conocen como enlaces duros y enlaces blandos. Los enlaces blandos (simbólicos), a diferencia de los duros pueden apuntar a directorios y pueden cruzar fronteras de sistemas de archivos (apuntar a otros sistemas de archivos) y el sistema operativo trata igualmente todos los enlaces.

El nombre del archivo *"."* en un directorio es un enlace duro al directorio mismo. El nombre de archivo *".."* es un enlace al directorio padre. Por tanto, si el directorio actual es */usr/jlp/programa*, entonces *../bin/wdf* se refiere a */usr/jpl/bin/wdf*.

Los dispositivos de Hardware tienen nombres en el sistema de archivos. El núcleo sabe que estos archivos especiales de dispositivos o archivos especiales son interfaces con dispositivos, pero de todos modos el usuario accede a ellos prácticamente con las mismas llamadas al sistema que otros archivos.

5.1 Sistema de Archivos y Estructura de Directorios

El Sistema de Archivos de Unix/Linux o cualquier sistema de archivos, generalmente es una capa bajo el sistema operativo la cual maneja el posicionamiento de tus datos en el almacenamiento, sin este el sistema no puede saber dónde empieza y termina un archivo.

Éste consiste en un conjunto de archivos, cada uno de los cuales tiene un nombre, hay tres clases de archivos:

- Archivos regulares, que contienen información.
- Directorios o carpetas, que contienen un conjunto de archivos. Los

conjuntos de archivos se identifican por el nombre del directorio que los agrupa.

- Archivos especiales FIFO (First In First Out), algunas veces llamados Pipes, que proveen un canal para comunicación entre procesos independientes y que tienen un nombre asociado.

Como la mayoría de los sistemas operativos modernos, Unix/Linux organiza su sistema de archivos como una jerarquía de directorios. La jerarquía de directorios en un árbol invertido, un directorio especial, root '/', es la raíz de la jerarquía. Un directorio puede contener subdirectorios, archivos regulares y archivos especiales. Unix/Linux trata a los subdirectorios como tipo particulares de archivos. Además Unix/Linux ve a un archivo, no importando su tipo, como una sucesión de Bytes, almacenados en dispositivos como Discos, CD, DVDs o unidades de almacenamiento USB.

Un nombre de archivo puede tener hasta 255 caracteres en la mayoría de los sistemas. y contener cualquier carácter excepto '/' o el carácter NULL; sin embargo, algunos caracteres como '&' y ' ' pueden causar problemas por sus significados especiales para su interpretación en la línea de comandos. Los nombres de archivos son sensibles a las mayúsculas y minúsculas.

En los sistemas Unix/Linux, cada usuario tiene un directorio personal, en el cual almacena sus archivos. Dicho directorio se conoce como el hogar (Home) del usuario. Una vez que entres en sesión, cada programa que se ejecutase encuentra situado en un directorio de trabajo.

Tipos de Sistema de Archivos de GNU/Linux Cuando intentas instalar Linux, notaras que ofrece distintos sistemas de archivos como los siguientes:

Ext, Ext2, Ext3, Ext4, JFS, XFS, Btrfs, FAT32, exFAT, NTFS y Swap

Así que, ¿qué son estos sistemas de archivos que ofrece Linux?

- Ext: Antiguo y descontinuado debido a sus limitaciones.
- Ext2: Primer sistema de archivos de Linux que permite 2 Terabytes de datos.

- Ext3: Evolución del Ext2, con actualizaciones y retrocompatibilidad[74].
- Ext4: Es más rápido y permite archivos mucho más grandes con una velocidad significativa[75].
- F2FS: (Flash-Friendly File System) es un sistema de archivos que toma en cuenta las características de los dispositivos de almacenamiento basados en memorias Flash NAND como las unidades de estado sólido (SSD) y tarjetas eMMC y SD.
- JFS: Sistemas de archivos hechos por IBM. Funcionan bien con archivos grandes y pequeños, pero existen reportes de fallas y los archivos se corrompen después de un largo tiempo de uso.
- XFS: Sistema de archivos creado por SGI que funciona lento con archivos pequeños.
- Btrfs: Hecho por Oracle, no es tan estable como Ext en algunas distribuciones, pero es un buen reemplazo, si es necesario.
- FAT32: Establecido en 1996, es robusto, versátil pero anticuado, sólo permite guardar archivos de hasta 4 GB.
- exFAT: Es una actualización de FAT32 para acabar con la limitación de 4 GB por archivo.
- NTFS: Es una alternativa a FAT32 sin sus limitaciones, usada en discos duros y unidades externas.
- Swap: Es un espacio de intercambio que es utilizado para almacenar datos temporales, reduciendo así el uso de la RAM, normalmente es del doble del tamaño de la RAM del equipo.

Otras opciones de sistemas de archivos son: ZFS, ReiserFS, UFS, FFS, HFS, APFS. Algunas de sus características de los sistemas de archivo más usados son:

[74] 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.

[75] Es una muy buena opción para discos de estado sólido SSD, además puedes darte cuenta que cuando intentas instalar cualquier distribución de Linux este es el sistema de archivo por defecto que sugiere Linux.

Sistema	Tamaño máx. volumen	Tamaño máx. archivo
$FAT32$	8 TB	4 GB
$NTFS$	1.845^7 TB	256 TB
$EXT4$	1.153^6 TB	17.5921 TB

En el sistema de archivos de Linux (Ext2/3/4, UFS, ReiserFS, FFS, XFS, Btrfs, etc.) se tiene asociado un elemento en la tabla que guarda a los archivos y directorios dentro del sistema de archivos, que contiene un número entero único. Este número identifica la ubicación del archivo dentro del área de datos llamado *inodo (i-node)*.

Cada *inodo* contiene información de un fichero o directorio. En concreto, en un *inodo* se guarda la siguiente información:

- El identificador del dispositivo que alberga al sistema de archivos.
- El número de *inodo* que identifica al archivo dentro del sistema de archivos.
- La longitud del archivo en Bytes.
- El identificador de usuario del creador o un propietario del archivo con derechos diferenciados.
- El identificador de grupo de un grupo de usuarios con derechos diferenciados.
- El modo de acceso: capacidad de leer, escribir, y ejecutar el archivo por parte del propietario, del grupo y de otros usuarios.
- Las marcas de tiempo con las fechas de última modificación (*mtime*), acceso (*atime*) y de alteración del propio *inodo* (*ctime*).
- El número de enlaces (*Hard Links*), esto es, el número de nombres (entradas de directorio) asociados con este *inodo*.
- Tabla de direccionamiento donde se detallan los bloques del disco duro en que está almacenado el fichero.

podemos conocer la información del *inodo* de un archivo/directorio/enlace usando:

```
$ stat nombre
```

o bien mediante:

```
$ ls -i
```

si conocemos el *inodo* de un archivo y queremos conocer los nombres de archivo de los enlaces, usamos *(por ejemplo 3432343)*:

```
$ find / -inum 3432343
```

y si queremos conocer el número de inodos de disco podemos usar:

```
$ df -i
```

Además están las Dentries que tienen la función de definir la estructura de un directorio, éstas conjuntamente con los inodos, serán los encargados de representar un fichero en la memoria. Notemos que en cada directorio del sistema existen dos entradas especiales:

- La entrada con un punto (.) hace referencia al propio directorio.
- La entrada con dos puntos (..) hace referencia al inodo del directorio que contiene el directorio (el padre).

El área de datos ocupa el resto del disco y es equivalente a la zona de datos en el *FAT*. En esta zona, como su nombre indica, están almacenados los ficheros y directorios de nuestro sistema.

Estructura de Directorios en GNU/Linux Además de los sistemas de archivos que difiere de la de Windows, la estructura de directorios en Linux es distinta, y es necesario conocerla para encontrar ficheros de configuración, instalar ciertos paquetes en el lugar adecuado, localizar las fuentes del Kernel, o la imagen de este, nuestros ficheros personales, entre otros.

De hecho, la Fundación Linux mantiene un estándar de jerarquía del sistema de archivos, este define la estructura de directorios y el contenido de los directorios en las distribuciones Linux. Gracias a este estándar es posible encontrar la misma estructura de directorios en (casi) todas las distribuciones de Linux[76] que a continuación describiremos brevemente:

[76] 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.

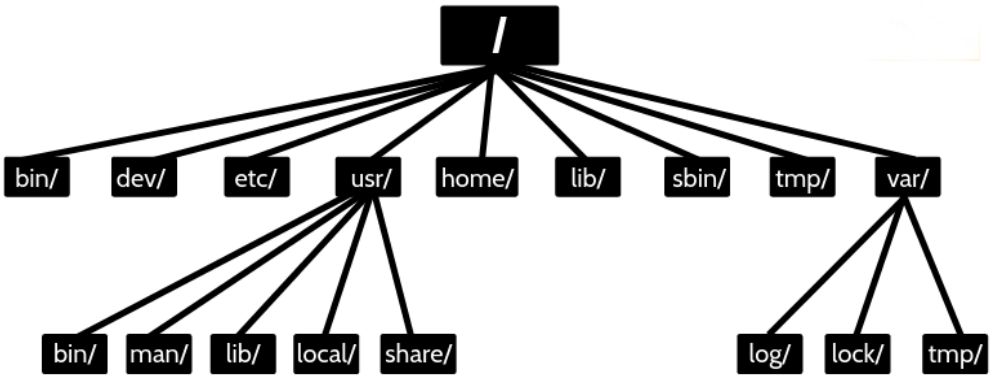

Figura 2: Jerarquía del sistema de archivos de Linux.

/ es el directorio principal, la *raíz* o *root*. Contiene el resto de directorios, es decir, todos los demás serían subdirectorios de este (incluso si están en particiones o discos diferentes). Sin duda es el más importante.

/bin es el directorio donde se almacenan los binarios, es decir, los programas que emplea el sistema para labores administrativas como los comandos *cp, echo, grep, mv, rm, ls, kill, xkill, ps, su, tar*, etc.

/sbin la S es de System, y como su nombre indica, aquí se almacenan los binarios o programas que emplea el propio sistema operativo para tareas de arranque, restauración, etc. Por ejemplo, *fsck, mount, mkfs, reboot, swapon.*

/boot es el directorio de arranque, donde está la o las imágenes del Kernel Linux que se cargarán durante el arranque, también directorios y configuración del propio gestor de arranque.

/etc muy importante para el administrador, ya que aquí residen los ficheros de configuración de los componentes del sistema y otros programas instalados.

/dev es un directorio muy especial donde se encuentran los dispositivos de bloques o caracteres, es decir, ficheros que representan la memoria, particiones, discos, dispositivos de Hardware, etc. Ya sabes que en LINUX y

UNIX todo es un archivo, y no unidades como en Windows. Por ejemplo, el disco duro o particiones serían */dev/sda, /dev/hda, /dev/scd*, /dev/fd/, etc.

/proc es otro directorio muy especial, más que un directorio es una interfaz por decirlo de un modo sencillo. Y aquí el sistema nos presenta los procesos[77] 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.

[77] Existen procesos activos y dormidos, procesos huérfanos y procesos zombies.

Los procesos activos son aquellos que están en ejecución en el sistema y los procesos dormidos son aquellos que esperan algún recurso o señal para continuar con su ejecución.

Los procesos huérfanos son aquellos que se siguen ejecutando a pesar que su proceso padre concluyó su operación.

Los procesos zombies es un proceso que ha concluido pero aún están presentes en la tabla de procesos.

/opt es un directorio que almacenará los paquetes o programas instalados en el sistema que son de terceros. Por ejemplo, si instalamos algún antivirus, Chrome, Arduino IDE o ciertos paquetes grandes, suelen instalarse aquí.

/root no hay que confundirlo con /, una cosa es el directorio raíz o *root* y otra muy diferente */root*. En este caso, se puede asemejar a un */home* pero es exclusivo para el usuario *root* o usuario administrador.

/srv almacena ficheros y directorios relativos a servidores que tienes instalados en el sistema, como Web, FTP, CVS, etc.

/sys junto con */dev* y */proc*, es otro de los especiales. Y como */proc*, realmente no almacena nada, sino que es una interfaz también. En este caso, son ficheros virtuales con información del Kernel e incluso, se pueden emplear algunos de sus ficheros para configurar ciertos parámetros del Kernel.

/tmp es el directorio para ficheros temporales de todo tipo. Es empleado por los usuarios para almacenar de forma temporal ciertos ficheros o incluso para almacenar Caché o ciertos ficheros volátiles de navegadores Web, etc. No obstante, hay otro directorio para lo mismo en */var/tmp*.

/var se trata de un directorio con directorios y ficheros que suelen crecer de tamaño, como bases de datos, *logs*, etc. Es precisamente los *logs* o registros del sistema por lo que es más popular este directorio, y allí encontrarás muchísima información de todo lo que ocurre en el sistema: */var/logs/*. Dentro de dicho directorio encontrarás separados por directorios, los *logs* de multitud de Software, incluido el sistema.

/usr son las siglas de *User System Resources*, y actualmente almacena ficheros de solo lectura relativos a utilidades del usuario, como los paquetes que instalamos mediante el gestor de paquetes en nuestra distribución. Dentro hay como una jerarquía de árbol de directorios vistos hasta ahora (casi todos) como si de un segundo nivel se tratase. Vas a encontrar */usr/bin*, */usr/lib*, */usr/sbin*, */usr/src,* etc., que por lo dicho anteriormente y sus nombres, es intuitivo saber lo que almacenan. Por ejemplo en /usr/src es donde permanecerán los ficheros de código fuente.

Ten en cuenta que no todas las distribuciones de Linux siguen este esquema y puede haber pequeñas variaciones, pero si se adaptan al estándar, no tendrás problemas al navegar por la estructura de archivos.

Rutas Absolutas y Relativas cuando se empieza a manejar un intérprete de comandos, una de las cosas que más cuesta es acostumbrarte a encontrar y hacer referencia a elementos del sistema de ficheros. Mientras que en un entorno gráfico tenemos que hacer Clic en carpetas y subcarpetas hasta llegar al elemento deseado, en el intérprete de comandos tendremos que conseguir lo mismo, pero indicando el lugar mediante una cadena de texto compuesta por los nombres de las carpetas que hay que recorrer hasta el lugar donde se encuentra el elemento deseado. Según el sistema cada nombre de carpeta se separa por un carácter especial, que en Linux es la diagonal (/).

Estas rutas serán usadas por los comandos para saber dónde encontrar los elementos sobre los que se tiene que realizar la acción correspondiente[78]. 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/
```

[78] Por ejemplo, si quiero posicionarme en un directorio determinado, utilizaré el comando cd y para indicar el sitio adonde quiero ir usaré una ruta, por ejemplo cd /home/. El comando cp copia elementos, en este caso necesitaremos dos rutas una para el origen (elemento que quiero copiar) y otra para el destino (elemento nuevo que voy a crear o lugar donde voy a dejar las copias). Por lo tanto podría poner:
cp /etc/passwd /home/copia_passwd.

estas rutas suelen ser bastante largas, pero tienen como ventaja que funcionan siempre, independientemente del lugar desde el que se ejecute la orden[79].

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.

[79] Es muy recomendable utilizar la facilidad que brinda el BASH de completar el nombre de un elemento del sistema de ficheros pulsando la tecla tabulador. Ahorrará mucho tiempo y errores.

- Permisos del grupo.
- Permisos del resto de usuarios (o también llamados "los otros").

Para tener claros estos conceptos, en los sistemas en red siempre existe la figura del administrador, superusuario o root. Este administrador es el encargado de crear y dar de baja a usuarios, así como también, de establecer los privilegios que cada uno de ellos tendrá en el sistema. Estos privilegios se establecen tanto para el directorio de trabajo (Home) de cada usuario como para los directorios y archivos a los que el administrador decida que el usuario pueda acceder.

Permisos del propietario el propietario es aquel usuario que genera o crea un archivo/carpeta dentro de su directorio de trabajo, o en algún otro directorio sobre el que tenga derechos. Cada usuario tiene la potestad de crear, por defecto, los archivos que quiera dentro de su directorio de trabajo. En principio, él y solamente él será el que tenga acceso a la información contenida en los archivos y directorios que hay en su directorio trabajo o Home -bueno, no es del todo cierto esto, ya que el usuario *root* siempre tiene acceso a todos los archivos y directorios del sistema-.

Permisos del grupo lo más normal es que cada usuario pertenezca a un grupo de trabajo. De esta forma, cuando se gestiona un grupo, se gestionan todos los usuarios que pertenecen a éste. Es decir, es más fácil integrar varios usuarios en un grupo al que se le conceden determinados privilegios en el sistema, que asignar los privilegios de forma independiente a cada usuario.

Permisos del resto de usuarios por último, también los privilegios de los archivos contenidos en cualquier directorio, pueden tenerlos otros usuarios que no pertenezcan al grupo de trabajo en el que está integrado el archivo en cuestión. Es decir, a los usuarios que no pertenecen al grupo de trabajo en el que está el archivo, pero que pertenecen a otros grupos de trabajo, se les denomina resto de usuarios del sistema.

¿cómo puedo identificar todo esto? sencillo, abre una terminal y escribe lo siguiente:

$ ls -l

entregará varias salidas como esta:

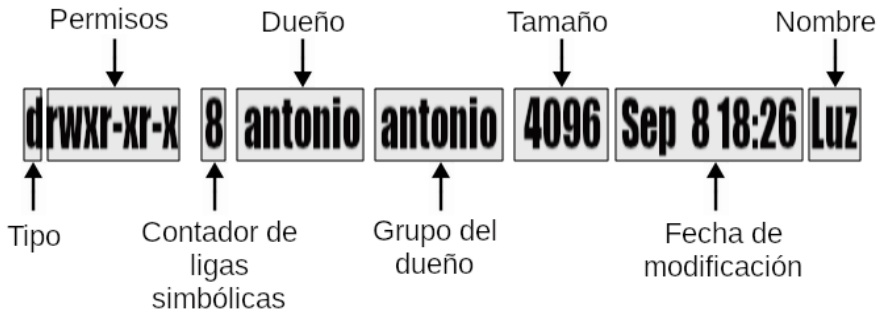

Figura 3: Estructura de permisos en la salida de: ls -l

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

- \- Archivo
- d Directorio
- l Liga simbólica
- s Socket (paso de datos entre dos procesos)
- p Pipe
- c Dispositivo de carácter (comunicación de Hardware)
- b Dispositivo de bloque (para el control de dispositivos)

Los siguientes 9 restantes, representan los permisos del archivo y deben verse en grupos de 3 y representan:

- \- Sin permiso
- r Permiso de lectura
- w Permiso de escritura

- x Permiso de ejecución

Los tres primeros representan los permisos para el propietario del archivo, los tres siguientes son los permisos para el grupo del archivo y los tres últimos son los permisos para el resto del mundo u otros.

Luego viene el contador de ligas simbólicas, el dueño del archivo, grupo al que pertenece, el tamaño en Bytes, la fecha de última modificación y finalmente el nombre del archivo o directorio.

Permisos Especiales Aún hay otro tipo de permisos que hay que considerar. Se trata del Bit de permisos SUID (Set User ID), el Bit de permisos SGID (Set Group ID) y el Bit de permisos de persistencia (Sticky Bit).

Setuid el Bit setuid es asignable a ficheros ejecutables, y permite que cuando un usuario ejecute dicho fichero, el proceso adquiera los permisos del propietario del fichero ejecutado. El ejemplo más claro de fichero ejecutable y con el Bit setuid es:

```
$ su
```

Podemos ver que el Bit está asignado como "s" en:

```
$ ls -l /bin/su
```

Para asignar[80] este Bit a un fichero seria:

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

Y para quitarlo:

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

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

Setgid el Bit setid permite adquirir los privilegios del grupo asignado al fichero, también es asignable a directorios. Esto será muy útil cuando varios usuarios de un mismo grupo necesiten trabajar con recursos dentro de un mismo directorio.

Para asignar este Bit hacemos lo siguiente:

```
$ chmod g+s /carpeta_compartida
```

Y para quitarlo:

```
$ chmod g-s /carpeta_compartida
```

Sticky este Bit suele asignarse en directorios a los que todos los usuarios tienen acceso, y permite evitar que un usuario pueda borrar ficheros/directorios de otro usuario dentro de ese directorio, ya que todos tienen permiso de escritura.

Podemos ver que el Bit está asignado como "t" en el directorio */tmp*.

Para asignar este Bit hacemos lo siguiente:

```
# chmod o+t /tmp
```

Y para quitarlo:

```
# chmod o-t /tmp
```

Entrada y Salida Estándar los procesos pueden abrir archivos a discreción, pero la mayor parte de los procesos esperan a que estén abiertos tres descriptores de archivos (numerados 0, 1 y 2) cuando inician. Estos descriptores se conocen como entrada estándar (0), salida estándar (1) y error estándar (2). Es común que los tres estén abiertos en la terminal del usuario. Así, el programa puede leer lo que el usuario teclea leyendo la entrada estándar, y puede enviar salidas a la pantalla del usuario escribiendo en la salida estándar. El descriptor de archivo de error estándar también está abierto para escritura, y se usa para los mensajes de error.

Standard Input la Entrada estándar, en inglés *standard input* (mejor conocido como *stdin*) es el mecanismo por el cual un usuario le indica a los programas la información que estos deben procesar. Por omisión, el teclado es la entrada estándar. La entrada estándar representa los datos que necesita una aplicación para funcionar, como por ejemplo un archivo de datos o información ingresada desde la terminal y es representado en la terminal como el tipo 0.

Standard Output la Salida estándar, en inglés *standard output* (mejor conocido como *stdout*) es el método por el cual el programa puede comunicarse con el usuario. Por omisión, la salida estándar es la pantalla donde se ejecutaron las instrucciones. La salida estándar es la vía que utilizan las aplicaciones para mostrarte información, allí podemos ver el progreso o simplemente los mensajes que la aplicación quiera darte en determinado momento y es representado en la terminal como el tipo 1.

Standard Error por último existe un flujo conocido como Error estándar, en inglés *standard error output* (mejor conocido como *stderr*) que es utilizado por las instrucciones para desplegar mensajes de error que surjan durante el transcurso de su ejecución. Al igual que *stdout*, el error estándar será la pantalla donde se procesaron las instrucciones. El error estándar es la forma en que los programas te informan sobre los problemas que pueden encontrarse al momento de la ejecución y es representado en la terminal como el tipo 2.

Redirección Mediante Pipe las tuberías (Pipe) unen la salida estándar de un comando con la entrada estándar de otro, es decir, la salida de un comando se emplea como entrada del siguiente. Para ello se emplea el símbolo pipe "|". El uso de tuberías evita la generación constante de archivos intermedios reduciendo el tiempo de procesamiento.

Redirección Hacia el Dispositivo Nulo en GNU/Linux, */dev/null* es un archivo especial al que se envía cualquier información que quiera ser descartada. Aunque al principio no lo parezca, el uso del dispositivo nulo es muy útil.

5.2 Interfaz de Usuario

Tanto el programador como el usuario de Linux manejan principalmente el conjunto de programas de sistemas que se ha escrito y están disponibles para ejecutarse. Estos programas efectúan llamadas al sistema operativo necesarias para apoyar su función, pero las llamadas al sistema en sí están contenidas dentro del programa y no tienen que ser obvias para el usuario.

GNU/Linux puede funcionar tanto en entorno gráfico[81] (Graphical User Interface, GUI) como en modo línea de comandos (Command-Line Interface, CLI) también conocida como consola o *Shell*. La consola es común en distribuciones para servidores, mientras que la interfaz gráfica está orientada al usuario final del hogar, como empresarial.

Los entornos de escritorio pertenecen a la interfaz gráfica, son un conjunto de programas conformado por ventanas, íconos, imágenes y muchas aplicaciones que facilitan el uso de la computadora. Los entornos de escritorio más populares en GNU/Linux son: GNOME, KDE, LXQt, LXDE, Xfce, Unity, MATE, Cinnamon, Pantheon, Deepin, Budgie, PIXEL, Enlightenment, Trinity, Moksha, Ukui, etc. Dependiendo de la distribución se pueden tener uno o más escritorios instalados, por ejemplo en Debian GNU/Linux están disponibles los más usados y si el usuario los instala, puede decidir al iniciar sesión cuál usar.

5.2.1 Interfaz Gráfica de Usuario

La interfaz gráfica de usuario es un tipo de visualización que permite al usuario elegir comandos, iniciar programas y ver listas de archivos y otras opciones utilizando las representaciones visuales (iconos) y las listas de elementos del menú. Las selecciones pueden activarse bien a través del teclado o con el ratón.

Para los autores de aplicaciones, las interfaces gráficas de usuario ofrecen

[81] Un servidor de pantalla en GNU/Linux es un programa que es responsable de la coordinación de entrada y salida de sus clientes, hacia y desde en resto del sistema operativo, y entre el Hardware y el sistema operativo. El servidor de visualización proporciona el marco para un entorno gráfico para que se pueda utilizar el Mouse y el teclado para interactuar con las aplicaciones. El servidor de pantalla se comunica con sus clientes a través del protocolo del servidor de pantalla como: X11, Wayland o Mir. El servidor de visualización es un componente clave en cualquier interfaz gráfica de usuario, específicamente el sistema de ventanas. No debemos confundir el servidor de visualización con el entorno de escritorio. El entorno de escritorio utiliza un servidor de pantalla debajo.

un entorno que se encarga de la comunicación con la computadora. Esto hace que el programador pueda concentrarse en la funcionalidad, ya que no está sujeto a los detalles de la visualización ni a la entrada a través del ratón o del teclado. También permite a los programadores crear programas que realicen de la misma forma las tareas más frecuentes, cómo guardar un archivo, porque la interfaz proporciona mecanismos estándar de control como ventanas y cuadros de diálogo. Otra ventaja es que las aplicaciones escritas para una interfaz gráfica de usuario son independientes de los dispositivos: a medida que la interfaz cambia para permitir el uso de nuevos dispositivos de entrada y salida, como un monitor de pantalla grande o un dispositivo óptico de almacenamiento, las aplicaciones pueden utilizarlos sin necesidad de cambios.

¿Qué es un Entorno de Escritorio? un entorno de escritorio es un conjunto de Software para ofrecer al usuario de una computadora una interacción amigable y cómoda. El Software es una solución completa de interfaz gráfica de usuario, ofrece iconos, barras de herramientas, carpetas, fondos de pantalla y Widgets de escritorio, e integración entre aplicaciones con habilidades como, arrastrar y soltar. En general cada entorno de escritorio se distingue por su aspecto y comportamiento particular, aunque algunos tienden a imitar características de escritorios ya existentes. El primer entorno moderno de escritorio que se comercializó fue desarrollado por Xerox en los años 80. Actualmente el entorno más conocido es el ofrecido por la familia Windows aunque existen otros como los de: Macintosh (Classic y Cocoa) y de código abierto como: GNOME, KDE, LXQt, LXDE, Xfce, Unity, MATE, Cinnamon, Pantheon, Deepin, Budgie, PIXEL, Enlightenment, Trinity, Moksha, Ukui, etc.

¿Qué son los Gestores de Ventanas? un gestor de ventanas o en inglés Windows Manager, es un programa que controla la ubicación y apariencia de las aplicaciones bajo el sistema X Windows. Las computadoras suelen ofrecer una interfaz gráfica de usuario que facilita la interacción con el sistema operativo. Las plataformas Windows y Macintosh ofrecen métodos de visualización y control de las ventanas e interacción con las aplicaciones, estandarizados por sus vendedores. En cambio el sistema gráfico X Windows, popular en el ámbito de sistemas Unix y similares, como GNU/Linux, permite al usuario escoger entre varios gestores según sus gustos o necesidades. Los

gestores de ventanas difieren entre sí de muchas maneras, incluyendo apariencia, consumo de memoria, opciones de personalización, escritorios múltiples o virtuales y similitud con ciertos entornos de escritorio ya existentes. Estos se dividen en 3 tipos, que son los siguientes:

- Stacking: Aquellos que imitan las apariencias y funcionalidades de Windows y Mac OS X, por ende, gestionan las ventanas como pedazos de papel en un escritorio, que pueden ser apiladas unas sobre otras.
- Tiling: Aquellos de tipo "mosaico" donde las ventanas no se superponen, y donde suelen hacerse un uso muy extenso de atajos de teclado, y se obtiene una menor dependencia del uso del ratón.
- Dynamics: Aquellos que permiten alterar dinámicamente el diseño de las ventanas entre mosaicos o flotantes.

Las acciones asociadas al gestor de ventanas suelen ser, abrir, cerrar, minimizar, maximizar, mover, escalar y mantener un listado de las ventanas abiertas. Es también muy común que el gestor de ventanas integre elementos como: el decorador de ventanas, un panel, un visor de escritorios virtuales, iconos y un tapiz.

Entornos de Escritorios más Conocidos:

KDE (https://kde.org)
proyecto que fue iniciado en octubre de 1996 por el programador alemán Matthias Ettrich, quien buscaba crear una interfaz gráfica unificada para sistemas Unix. En sus inicios imitó a CDE (Common Desktop Environment), un entorno de escritorio utilizado por varios Unix. Este es un entorno de escritorio con multitud de aplicaciones e infraestructura de desarrollo para diversos sistemas operativos como GNU/Linux, Mac OS X, Windows, etc. Los principales componentes de Software elaborados por KDE se agrupan bajo el nombre KDE Frameworks, KDE Plasma y KDE Applications. Las aplicaciones KDE están traducidas a aproximadamente 88 idiomas y están construidas con los principios de facilidad de uso y de accesibilidad moderna en mente y funcionan de forma completamente nativa en GNU/Linux, BSD, Solaris, Windows y Mac OS X.

GNOME (https://www.gnome.org)
este proyecto fue iniciado por los programadores mexicanos Miguel de Icaza y Federico Mena, forma parte oficial del proyecto GNU. Nació como una alternativa a KDE bajo el nombre de GNU Network Object Model Environment (Entorno de Modelo de Objeto de Red GNU). Actualmente, GNOME se está traduciendo a 193 idiomas. Está disponible en las principales distribuciones GNU/Linux, incluyendo Fedora, Debian, Ubuntu, Manjaro Linux, Red Hat Enterprise Linux, SUSE Linux Enterprise, CentOS, Oracle Linux, Arch Linux, Gentoo, SteamOS, entre otras. También, se encuentra disponible en Solaris, un importante sistema operativo UNIX y en Sistemas operativos Unix-like como FreeBSD.

Xfce (https://www.xfce.org)
es un entorno de escritorio libre para sistemas tipo Unix como GNU/Linux, BSD, Solaris y derivados. Su objetivo es ser rápido y ligero, sin dejar de ser visualmente atractivo y fácil de usar. Consiste en varios componentes empaquetados por separado que en conjunto proporcionan la funcionalidad completa del entorno de escritorio, pero se pueden seleccionar por separado para que el usuario pueda adaptar el ambiente de trabajo a sus necesidades. Puede ser instalado en varias plataformas como: Linux, NetBSD, FreeBSD, OpenBSD, Solaris, Cygwin y MacOS X, sobre x86, PPC, Sparc, Alpha.

LXDE (https://lxde.org)
es un entorno de escritorio libre para Unix y otras plataformas POSIX[82], 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

[82] Significa Portable Operating System Interface. Consiste en una familia de estándares especificadas por la IEEE con el objetivo de facilitar la interoperabilidad de sistemas operativos. Además, POSIX establece las reglas para la portabilidad de programas. Por ejemplo, cuando se desarrolla Software que cumple con los estándares POSIX existe una gran probabilidad de que se podrá utilizar en sistemas operativos del tipo Unix. Si se ignoran tales reglas, es muy posible que el programa o librería funcione bien en un sistema dado pero que no lo haga en otro.

cada uno de ellos se puede utilizar independientemente con muy pocas dependencias. Usa Openbox como gestor de ventanas predeterminado y apunta a ofrecer un escritorio ligero y rápido basado en componentes independientes que pueden ser utilizados en otros entornos.

LXQt (https://lxqt.github.io)
es un entorno de escritorio libre y de código abierto para Linux, resultado de la fusión entre los proyectos LXDE y Razor-qt. LXQt conjuga la filosofía de LXDE con las librerías QT, usa el gestor de ventanas del escritorio Razor-qt, es un escritorio muy liviano y es considerado por muchos como el sucesor de LXDE.

Gestores de Ventanas más Conocidos:

Enlightenment (https://www.enlightenment.org)
también conocido simplemente como E, es un gestor de ventanas X11 ligero para UNIX y GNU/Linux. Uno de sus objetivos es llegar a ser un entorno de escritorio completo. Es muy configurable y muy atractivo visualmente. Durante un tiempo fue el gestor de ventanas de GNOME.

IceWM (https://ice-wm.org)
es un gestor de ventanas para el X Windows System gráfico de infraestructura escrito por Marko Macek. Se ha codificado desde cero en C++ y es liberado bajo GNU. IceWM es ligero y personalizable. Se puede configurar a partir de archivos de texto almacenados en un directorio Home del usuario, haciendo fácil la personalización y copia de configuraciones. Posee soporte oficial para menús de GNOME y KDE previamente disponible como un paquete separado.

Windows Maker (https://www.windowsmaker.org)
es un popular gestor de ventanas para X Windows System diseñado para emular NeXT del GUI como OpenStep compatible, ha sido descrito como "uno de los más útiles y universales gestores de ventanas disponibles. Windows Maker tiene la reputación de ser rápido, eficiente y altamente estable y es muy popular entre las soluciones de código abierto para su uso tanto en nuevas como en viejas máquinas. Como con la mayoría de gestores de ventanas, soporta un montón de temas disponibles.

Fluxbox (http://fluxbox.org)
es un gestor de ventanas X basado en Blackbox 0.61.1.Su objetivo es ser ligero y personalizable, y cuenta con un apoyo mínimo de iconos gráficos. Su interfaz de usuario sólo tiene una barra de tareas y un menú al que se puede acceder pulsando con el botón derecho sobre el escritorio. Todas las configuraciones básicas están controladas por ficheros de texto. Fluxbox puede mostrar algunos Eye Candies: colores, gradientes, bordes y una que otra apariencia básica. Las versiones recientes soportan esquinas redondeadas y elementos gráficos. Fluxbox también tiene varias características de las cuales Blackbox carece, incluyendo ventanas con pestañas y un título configurable.

Openbox (http://openbox.org/new/)
es un famoso gestor de ventanas libre para el sistema de ventanas X, licenciado bajo la GNU General Public License. Openbox fue originalmente derivado de Blackbox 0.65.0, pero ha sido totalmente reescrito en el lenguaje de programación C y desde la versión 3.0 no se basa en ningún código de Blackbox. Su sistema de menú tiene un método para utilizar los menús dinámicos.

Otros existen muchos otros gestores de ventanas que pudiéramos mencionar, tales como: 9wm, Awesome, AfterStep, Scwm, Blackbox, Bspwm, Byobu, Cinnamon, Cwm, Deepin, Dwm, Fvwm, Icewm, Jwm, Kahakai, Lumina, Qtile, Wmii, WindowLab, Ratpoison, Sawfish, Sway, wm2, Wmx, StumpWM, Twm, Waimea, Xmonad, I3, E16.

5.2.2 Línea de Comandos y Órdenes

La mayoría de los usuarios de ordenadores de hoy sólo están familiarizados con la interfaz gráfica de usuario o GUI, los vendedores y los expertos les han enseñado que la interfaz de línea de comandos o CLI es una cosa espantosa del pasado. Es una pena, porque una buena interfaz de línea de comandos es una maravillosa y expresiva forma de comunicarse con el ordenador, muy parecida a lo que el lenguaje escrito es para los seres humanos. Se ha dicho que "las interfaces gráficas de usuario hacen fáciles las tareas fáciles, mientras que las interfaces de línea de comandos hacen posibles las tareas difíciles" y eso es muy cierto aún hoy.

Dado que Linux fue desarrollado desde la familia de sistemas operativos Unix, comparte la misma rica herencia de herramientas de línea de comandos

que Unix. Unix saltó a la fama en los primeros años ochenta (aunque fue desarrollado una década antes), antes de que se extendiera la adopción de las interfaces gráficas de usuario, y por eso, se desarrolló una amplia interfaz de línea de comandos en su lugar. De hecho, una de las razones más potentes para que los primeros que utilizaron Linux lo eligieran sobre, digamos, Windows NT, era la poderosa interfaz de línea de comandos que hacía las "tareas difíciles posibles".

Emuladores de Terminal Cuando utilizamos una interfaz gráfica de usuario, necesitamos otro programa llamado emulador de terminal para interactuar con el Shell. Si buscamos en nuestros menús de escritorio, probablemente encontraremos uno. KDE usa Konsole y GNOME usa Gnome-terminal, aunque es probable que se llame simplemente "Terminal" en nuestro menú. Hay muchos otros emuladores de terminal disponibles para Linux, pero todos hacen básicamente lo mismo; nos dan acceso al Shell.

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

El Intérprete de Órdenes de Consola o Shell es un programa informático, cuya función consiste en interpretar órdenes, y un lenguaje de consola. El más conocido es Bash[83]. Es una Shell de Unix compatible con POSIX[84] y el intérprete de comandos por defecto en la mayoría de las dis-

[83] 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.

[84] Significa Portable Operating System Interface. Consiste en una familia de estándares especificadas por la IEEE con el objetivo de facilitar la interoperabilidad de sistemas operativos. Además, POSIX establece las reglas para la portabilidad de programas. Por ejemplo, cuando se desarrolla Software que cumple con los estándares POSIX existe una gran probabilidad de que se podrá utilizar en sistemas operativos del tipo Unix. Si se ignoran tales reglas, es muy posible que el programa o librería funcione bien en un sistema dado pero que no lo haga en otro.

Sin ir más lejos, examinemos en la última versión de la especificación IEEE 1003 (designación formal de los estándares POSIX). Al buscar la referencia sobre el comando du, podemos ver que solamente las opciones -a, -H, -k, -L, -s, y -x son obligatorias. Otras, tales

tribuciones GNU/Linux, además de Mac OS. También se ha llevado a otros sistemas como Windows y Android.

Algunas alternativas a Bash son: SH (Bourne Shell), TCSH/CSH (C Shell), KSH (Korn Shell), Fish (Friendly Interactive Shell), ZSH (Z Shell), TSCH (TS Shell), Dash (Debian Almquist Shell), DSH (Distributed Shell).

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

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

Sintaxis Estándar de Comandos Se ha definido un estándar para comandos POSIX (Portable Operating System Interfaz) por la IEEE, pero no todos los comandos la siguen, Muchos comandos definidos por POSIX tienen una forma moderna y una sintaxis obsoleta, por compatibilidad con sistemas viejos. Por ejemplo el estándar especifica que las opciones de los comandos (no así, los operandos) pueden aparecer en cualquier orden, pero algunos comandos pueden requerir que las opciones aparezcan en un orden particular y muchas veces esto no está mencionado explícitamente el la página del manual del comando.

Un comando consiste en una sucesión de palabras separadas por espacios en blanco. La primera palabra es el nombre del comando y las palabras subsecuentes son sus argumentos u opciones. Los operandos consisten de las opciones del programa, si hay alguna, seguidas de sus operandos, si hay alguno. Las opciones controlan o modifican lo que el comando hace. Los operadores especifican nombres de trayectorias o con lo que va a trabajar el comando.

Las opciones se especifican con una sucesión de palabras. Cada palabra es un grupo de opciones o una opción argumento. Las opciones generalmente se denotan por letras. Se pueden escribir en cualquier orden y se pueden combinar varias opciones en un solo grupo (siempre y cuando no reciban

como -c y -h, aparecen en la versión de du provista por el proyecto GNU. Esto significa que si utilizamos estas últimas en un Script desarrollado en Linux e intentamos hacerlo funcionar en FreeBSD o AIX, es muy probable que no funcione como esperamos.

ningún argumento). Cada grupo de opciones está precedido por un guión (-). Por ejemplo, los comandos:

```
$ ls -al
$ ls -l -a
```

son equivalentes e indican que el comando *ls* (list archives) sea llamado con las opciones *a* (all files) y *l* (long listing).

Hay otras convenciones comunes para especificar argumentos:

- Dos guiones (- -) indica el final de las opciones. Esto es útil cuando quieres pasar argumentos que empiecen con un - a un comando, por ejemplo:

  ```
  $ rm - - -miArchivo
  ```

es una forma de indicarle que el archivo es *-miArchivo*, también podemos usar:

```
$ rm ./-miArchivo
```

- Un solo guión representa a la entrada estándar en un contexto en que el programa espera una trayectoria. Por ejemplo, el comando:

  ```
  $ diff - miArchivo
  ```

encuentra las diferencias entre la entrada estándar y el archivo *miArchivo*

Metacarácter o Shell Globbing los metacaracteres o Shell Globbing son caracteres que tienen un significado especial en la línea de comandos, para usar estos caracteres como caracteres ordinarios en la línea de comandos, debes marcarlos de manera especial para que el Shell no los interprete. Por ejemplo:

- La diagonal inversa (\) siempre sirve como carácter de escape para uno o más caracteres que le suceden, lo cual le da a esos caracteres un significado especial. Si un carácter es un metacaracter, el significado especial es el carácter mismo. Por ejemplo \\ usualmente significa una solo \ y \$ el signo de pesos. En estos casos, la diagonal inversa le quita su significado a los caracteres.

- Cuando un texto se encierra entre comillas (" "), la mayoría de los de los metacaracteres que estén en dicho texto aon tratados como caracteres normales, excepto por $, que generalmente indica sustituciones a realizar.

- Otra forma de citar muy similar a la anterior, pero más fuerte es usar comillas sencillas (' ') ya que ni siquiera el carácter $ es interpretado

También existen otros metacaracteres que son comodines que el sistema permite usar para especificar los nombres de archivos que satisfacen el filtro especificado a la hora de buscar, eliminar o filtrar nombres de archivo, estos metacaracteres son: *, ?, [ ] y [^].

- ***** Se utiliza para reemplazar cero o más caracteres. Puede ser sustituido por cualquier cadena de caracteres, ejemplo:

  $ ls a*.pdf

- **?** Sustituye un carácter cualquiera, ejemplo:

  $ ls a?chivo.pdf

- [ ] Se usa para definir rangos o conjuntos de caracteres a localizar, para definir los rangos se debe usar el guión -, si son varios caracteres se separan por coma, ejemplo:

  $ ls [Aa]rchivo[0-9].pdf

- [^] o **[!]**Este caso es contrario al anterior, este representa que se busque algo exceptuando lo que se encuentra entre los corchetes, también trabaja con rangos, ejemplo:

  $ls [^A]rchivo.pdf

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

```
$ su antonio
```

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

```
$ su - administracion
```

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

```
$ su
```

El usuario root en GNU/Linux es el usuario que tiene acceso administrativo al sistema. Los usuarios normales no tienen este acceso por razones de seguridad. Sin embargo, en múltiples sistemas derivados de Debian GNU/Linux no incluye el usuario root (por ejemplo en todos los derivados de Ubuntu). En su lugar, se da acceso administrativo a usuarios individuales, que pueden utilizar la aplicación *sudo* para realizar tareas administrativas. La primera cuenta de usuario que creó en su sistema durante la instalación tendrá, de forma predeterminada, acceso a *sudo*.

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

Para usar *sudo* en la línea de comandos, simplemente escriba *sudo* antes del comando que desea ejecutar, *sudo* le pedirá su contraseña[85]:

```
$ sudo apt update
```

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

Trabajando en Línea de Comandos Las órdenes se pueden agrupar en varias categorías; la mayor parte de ellas están orientadas hacia archivos o directorios. Por ejemplo los programas de sistema que manipulan directorios son *mkdir* para crear un directorio nuevo, *rmdir* para eliminar un directorio, *cd* para cambiar el directorio actual a otro y *pwd* para visualizar el nombre de la ruta absoluta del directorio actual (de trabajo).

El programa *ls* lista los nombres de los archivos del directorio actual. Cualquiera de las más de 20 opciones de *ls* puede hacer que se exhiban también las propiedades de los archivos, por ejemplo, la opción *-l* pide un listado largo, que muestra el nombre de cada archivo, su dueño, la protección, su tamaño, la fecha y hora en que se creó. El programa *cp* crea un archivo nuevo que es una copia de uno ya existente. El programa *mv* cambia de lugar un archivo dentro del árbol de directorios. En la mayor parte de los casos, este movimiento sólo requiere un cambio de nombre de archivo, pero si es necesario el archivo se copia en su nueva posición y la copia vieja se elimina. Los archivos se eliminan con el programa *rm*.

Para mostrar el contenido de un archivo en la terminal, un usuario puede ejecutar *cat*. El programa *cat* toma una lista de archivos y los concatena, copiando el resultado en la salida estándar, que normalmente es la terminal. Claro que en la terminal el archivo podría exhibirse con demasiada rapidez como para leerse. El programa *more* exhibe el archivo pantalla por pantalla y hace una pausa hasta que el usuario teclea un carácter para continuar con la siguiente pantalla. El programa *head* exhibe sólo las primeras líneas del archivo; *tail* muestra las últimas líneas.

Estos son algunos de los programas que usa Linux, además hay editores de texto (*ed, sed, emacs, nano, vi*, etc.), compiladores (de C, C++, Java, Python, etc.), formateadores de texto (LaTeX*, troff*, etc.), programas para ordenar (*sort*) y comparar (*cmp, diff*) archivos, buscar patrones (*grep, awk*) y muchas otras actividades.

La ejecución de una orden se efectúa con una o más llamadas al sistema operativo. Por lo regular, el Shell ejecuta una pausa cuando se le pide ejecutar una orden, queda en espera a que ésta se termine de ejecutar. Existe una sintaxis sencilla (un signo & al final de la línea de órdenes) para indicar que el Shell no debe esperar hasta que termine de ejecutarse la orden. Una orden que deja de ejecutarse de esta manera mientras el Shell sigue interpretando órdenes subsecuentes es una orden en segundo plano, o que se ejecuta en segundo plano. Los procesos para los cuales el Shell sí espera, se ejecutan en

primer plano.

El Shell del sistema GNU/Linux ofrece un recurso llamado control de trabajos (y visualizarlos con los comandos *ps* o *top*) implementado especialmente en el núcleo. El control de trabajos permite transferir procesos entre el primer y segundo plano. Los procesos pueden detenerse y reiniciarse según diversas condiciones, como que un trabajo en segundo plano requiera entradas desde la terminal del usuario. Este esquema hace posible la mayor parte del control de procesos que proporcionan las interfaces de ventanas, pero no requiere Hardware especial. Cada ventana se trata como una terminal, y permite a múltiples procesos estar en primer plano (uno por ventana) en cualquier momento. Desde luego pueden haber procesos de segundo plano en cualquiera de las ventanas.

Algunos Comandos para Conocer el Sistema en el que Trabajamos
Siempre es una buena práctica saber que componentes de Hardware se tienen en el equipo en que corremos nuestro GNU/Linux, esto nos puede ayudar a lidiar problemas de compatibilidad cuando se trata de instalar paquetes, controladores en nuestro sistema.

El primer comando a ejecutar es aquel que nos dé información del sistema en el que trabajamos, este es *uname* con la bandera *-a*, que nos dará información de la versión del Kernel, la versión y distribución del sistema operativo y en qué tipo de Hardware es que estamos corriendo nuestro GNU/Linux, para ello usamos:

```
$ uname -a
```

Para conocer las características de nuestro CPU, podemos usar varios comandos, entre ellos está *lscpu*, este nos dará la arquitectura del sistema, el número de CPUs, Cores, el modelo de la familia de CPU, Cache del CPU, Threds, entre otros. Para usarlo escribimos:

```
$ lscpu
```

si sólo queremos conocer el número de cores del equipo, usamos:

```
$ nproc
```

Podemos visualizar la memoria total, usada, libre y Swap (de intercambio) de nuestro equipo, usando:

```
$ free -g
```

Para visualizar información sobre los discos duros, unidades de estado sólido, en nuestro equipo, podemos usar *lsblk* que nos reportará dicha información, para ello escribimos:

```
$ lsblk -a
```

o bien podemos usar el comando *df* o en modo administrador a comando *fdisk*, escribiendo:

```
$ df -h
# fdisk -l
```

Si necesitamos conocer la información sobre los controladores USB y todos los dispositivos que están conectados a ellos, usamos:

```
$ lsusb
```

En caso de necesitar conocer los dispositivos PCI que pueden incluir puertos USB, tarjetas gráficas, adaptadores de red, entre otros, más los dispositivos que están conectados a ellos usamos:

```
$ lspci
```

Si lo que deseamos es un listado detallado del Hardware de nuestro equipo de cómputo, entonces podemos usar cualquiera de estos comandos (que previamente deberemos instalar):

```
# lshw
# dmidecode
# hwinfo
```

Así como podemos conocer las características de nuestro equipo de cómputo, podemos también conocer todos los comandos (-c), alias (-a), comandos internos (-b), palabras reservadas Bahs (-k) y funciones de Bash (-A function) disponibles para el usuario, para ello usamos:

```
$ compgen -c
$ compgen -a
$ compgen -b
$ compgen -k
$ compgen -A function
```

Correr Múltiples Comandos en uno Solo Supongamos que se tienen que ejecutar varios comandos uno tras otro -sin conocer si el comando anterior fue exitoso para ejecutar el nuevo-, para este propósito se usa el separador ";", de esta manera se pueden ejecutar una serie de comandos en una línea, ejemplo:

```
$ mkdir tmp ; ls ; cd tmp ; ls
```

en este ejemplo se crea en directorio, luego visualiza el contenido del directorio, para luego cambiar de directorio y finalmente visualiza el contenido del directorio recién creado.

Si necesitamos ejecutar múltiples comandos en uno solo, sólo si el comando anterior fue exitoso, se usa el separador "&&", ejemplo:

```
$ mkdir tmp && ls && cd tmp && ls
```

Notemos que si ejecutamos:

```
$ mkdir tmp & ls & cd tmp & ls
```

el resultado obtenido en nada se parece a lo que esperamos, ya que los comandos pasarán a ser ejecutados en segundo plano "&" y sin orden alguno, por lo que el resultado no será el deseado.

Ya vimos cómo usar un solo comando para ejecutar varios comandos. Pero, ¿qué debo hacer en caso que el primer comando no se ejecute correctamente?, ¿deseo ejecutar el comando siguiente?. Podemos usar || para que si el primer comando falla se ejecute el segundo, pero si no falla no ejecutará el segundo comando, por ejemplo:

```
$ cd miDirectorio || mkdir miDirectorio && cd miDirectorio
```

También podemos combinar los comandos && y || los cuales se comportarán como el operador ternario de C y C++ (condición? expresión_verdadera; expresión_falsa):

```
$ comando1 && comando2 || comando3
```

por ejemplo:

```
$ [-f archivo.txt ] && echo "Archivo existe" || echo "Archivo
  no existe"
```

podemos combinar ;, && y || para correr múltiples comandos. Como por ejemplo:

```
$ sudo apt update && sudo apt upgrade && sudo apt clean
```

Instalar Paquetes Saber cómo instalar paquetes y programas en Linux y cualquiera de sus distribuciones es uno de los temas más complejos para los nuevos usuarios. El método más común para instalar un nuevo Software en Linux es través de la Terminal. No obstante, los comandos varían según el tipo de gestor de paquetes que posee el sistema.

Uno de los apartados que debes tener en cuenta antes de instalar un paquete o programa en Linux, es conocer qué es un sistema de gestión de paquetes. En términos simples, hace referencia a un conjunto de formatos de archivos y herramientas empleadas para actualizar, instalar o desinstalar algún Software en el sistema operativo[86]. 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[87]. 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

[86] 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*.

[87] Un repositorio es un lugar en la red que siempre está actualizado desde donde nuestra distribución de GNU/Linux puede buscar y descargar fácilmente todo tipo de programas y herramientas para su instalación en nuestro equipo.

o actualizar programas directamente desde el repositorio oficial de Linux, o bien desde un repositorio de terceros. Mientras que el comando *dnf* facilita la administración de programas.

Para Debian GNU/Linux usamos:

```
# apt install paquete
# dpkg -i paquete
```

En Ubuntu y derivados se usa:

```
$ sudo apt install paquete
$ sudo dpkg -i paquete
```

Para openSUSE:

```
$ sudo rpm -Uvh paquete
$ su zypper intall paquete
```

En Fedora se usa:

```
$ sudo rpm -Uvh paquete
$ su -c 'yum install paquete'
# yum install paquete
```

Para Mandriva:

```
$ sudo rpm -Uvh paquete
$ su urpmi *.rpm
```

En Gentoo, FreeBSD se usa:

```
# emerge -vq paquete
```

Para Red Hat, Fedora o CentOS se usa:

```
$ sudo yum install paquete
$ sudo rpm -i paquete
$ sudo dfn install paquete
# yum install paquete
# dfn install paquete
```

En Archn Manjaro Linux se usa:

```
# pacman -s paquete
```

Para paquetes SNAP:

```
$ snap install paquete
```

Instalar Paquetes en Debian y Derivados Una de las funciones del usuario administrador es hacer la instalación, actualización y borrado de paquetes, el comando más usado actualmente es apt (Advanced Packaging Tool, herramienta avanzada de empaquetado que es una interfase del paquete *apt-get*), es un programa de gestión de paquetes *.deb* creado por el proyecto Debian, *apt* simplifica en gran medida la instalación, actualización y eliminación de programas en GNU/Linux. Existen también programas que proporcionan estos mismos servicios como: *apt-get*, *aptitude*, *tasksel* y *dpkg*.

Una vez siendo el usuario *root*, podemos solicitar la descarga de las actualizaciones disponibles, usando:

```
# apt update
```

para conocer los paquetes que se necesitan actualizar, usamos:

```
# apt list –upgradeable
```

para actualizar los paquetes instalados en el sistema[88], 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:

[88] 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
```

6 Bibliografía

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

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

Herramientas
http://132.248.181.216/Herramientas/

Referencias

[1] Proyectos de Software Sourceforge, http://sourceforge.net/ 52, 53

[2] GNU Operating System, http://www.gnu.org/ 52, 280, 290

[3] Google Code, http://code.google.com 53

[4] http://es.wikipedia.org/wiki/Linux 42

[5] The Linux Kernel Archives, http://www.Kernel.org/ 52

[6] Debian el Sistema Operativo Universal, http://www.debian.org 52

[7] http://es.wikipedia.org/wiki/Android 52

[8] http://www.gnu.org/philosophy/free-sw.es.html 52, 280

[9] http://es.wikipedia.org/wiki/Software_libre 52, 280

[10] http://www.hispalinux.es/SoftwareLibre 52, 280

[11] http://es.wikipedia.org/wiki/Software_propietario 278

[12] Diferentes Tipos de Licencias para el Software, http://www.gnu.org/licenses/license-list.html 52, 280, 290

[13] FSF, Free Software Fundation, http://www.fsf.org/ 52, 280, 290

[14] GCC, the GNU Compiler Collection, http://gcc.gnu.org/

52

Declaro terminado este trabajo sufrido, ideado y llevado a cabo entre los años 2015 al 2025, aún y a pesar de impedimentos tales como: la mala suerte, la desventura, el infortunio, la incomprensión, la gripe, el Covid-19, las horas de frío y de calor, la tristeza, la desesperanza, el cansancio, el presente, el pasado y mi futuro, el que dirán, la vergüenza, mis propias incapacidades y limitaciones, mis aversiones, mis temores, mis dudas y en fin, todo aquello que pudiera ser tomado por mi, o por cualquiera, como obstáculo en este tiempo de mentiras, verdades, de incredulidad e ignorancia o negación de la existencia real y física de la mala fe.

Atentamente

Antonio Carrillo Ledesma