El Uso de Programas de Cómputo en los Cursos de la Carrera de Actuaría en la Facultad de Ciencias, UNAM

Antonio Carrillo Ledesma
Karla Ivonne González Rosas

El Uso de Programas de Cómputo en los Cursos de la Carrera de Actuaría en la Facultad de Ciencias, UNAM

Antonio Carrillo Ledesma y Karla Ivonne González Rosas
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 **5**
1.1 Software Propietario y Libre . . . 5
1.1.1 Software Propietario . . . 6
1.1.2 Software Libre . . . 7
1.2 El Cómputo en la Carrera de Actuaría . . . 9
1.2.1 Cursos que Usan Cómputo . . . 12
1.3 Paquetes de Cómputo de Uso Común . . . 15
1.3.1 Programas de Cálculo Numérico . . . 17
1.3.2 Programas de Estadística . . . 18
1.3.3 Programas Ofimáticos . . . 19
1.3.4 Otros Programas . . . 20
1.4 Agradecimientos . . . 20

2 Sistemas Operativos **21**
2.1 Windows . . . 38
2.2 UNIX y BSD . . . 45
2.3 Apple y sus macOS e iOS . . . 47
2.4 GNU/Linux . . . 52
2.5 Android . . . 66
2.6 Chromebook y Chrome OS . . . 69
2.7 Otros Sistemas Operativos . . . 72

3 Entornos de Desarrollo y Herramientas de Programación **78**
3.1 Java . . . 81
3.2 Python . . . 90
3.3 Julia . . . 106
3.4 C y C++ . . . 109
3.5 Fortran . . . 116
3.6 R . . . 121
3.7 Herramientas de Programación . . . 123
3.7.1 ¿Qué es eso de ASCII, ISO-8859-1 y UTF-8? . . . 129
3.7.2 Uso de Espacios o Tabuladores en Fuentes . . . 133
3.7.3 Comparar Contenido de Fuentes . . . 135
3.7.4 Astyle . . . 136
3.7.5 Compilación y la Optimización del Ejecutable . . . 137
3.7.6 Análisis de Rendimiento y Depuración . . . 142

3.7.7 Mejora del Rendimiento en Python . . . 146
3.7.8 Git . . . 152
3.7.9 GitLab vs GitHub . . . 171
3.7.10 Otras opciones . . . 176
3.8 Programando Desde la Nube . . . 178

4 Programación y Lenguajes de Programación 182
4.1 Paradigmas de Programación . . . 185
4.2 Lenguaje de Programación . . . 188
4.3 Conceptos Transversales . . . 193
4.4 Algo de Programación . . . 200
4.5 Introducción a los Paradigmas de Programación . . . 213
4.6 Errores de Redondeo y de Aritmética . . . 222
4.7 Documentación del Código Fuente . . . 233
4.7.1 Documentar en C, C++ y Java . . . 236
4.7.2 Documentar en Python . . . 243

5 Programación en Paquetes de Cálculo Numérico 248
5.1 Cálculo Numérico con Octave . . . 251
5.2 Trabajando con Octave . . . 252
5.3 Desde la Nube . . . 261

6 Programación en Paquetes de Cálculo Simbólico 262
6.1 Cálculo Simbólico con Maxima . . . 264
6.2 Trabajando con Maxima . . . 266
6.3 Desde la Nube . . . 272

7 Programación en Paquetes Estadísticos 273
7.1 Cálculo Estadístico con R . . . 275
7.2 Trabajando con R . . . 276
7.2.1 R y Python . . . 280
7.3 Desde la Nube . . . 281

8 Paquetes Ofimáticos 282
8.1 Hojas de Cálculo . . . 283
8.2 Bases de Datos . . . 285
8.3 Herramientas de Presentación y Multimedia . . . 287
8.4 Procesamiento de Imágenes . . . 288

8.5 Procesamiento de Textos 289

9 Seguridad, Privacidad y Vigilancia 292
9.1 ¿Qué es la Privacidad y por qué es Importante? 293
9.2 Las Vulnerabilidades y Exposiciones Comunes 300
9.3 Alfabetismo Digital 302
9.4 Amenazas a la Ciberseguridad 304
9.5 Dicen que si no Pagas por un Producto, Entonces el Producto Eres Tú. 311
9.6 Datos que Recopila Google 314
9.7 ¿Cómo me Protejo? 317

10 Consideraciones y Comentarios Finales 322
10.1 El Cómputo en Instituciones Educativas 325
10.2 Integración del Cómputo en Ciencias e Ingenierías 329
10.3 Ventajas, Desventajas y Carencias del Software Libre 330
10.4 Comentarios Finales 331

11 Apéndice A: Software Libre y Propietario 334
11.1 Software Propietario 337
11.2 Software Libre 339
11.3 Seguridad del Software 346
11.4 Tipos de Licencias 349
11.4.1 Licencias Creative Commons 355
11.4.2 Nuevas Licencias para Responder a Nuevas Necesidades 357
11.5 Implicaciones Económico-Políticas del Software Libre 360
11.5.1 Software Libre y la Piratería 360
11.5.2 ¿Cuánto Cuesta el Software Libre? 361
11.5.3 La Nube y el Código Abierto 364
11.5.4 El Código Abierto como Base de la Competitividad . . 367
11.5.5 Software Libre en Empresas y Corporaciones 368
11.6 Código Abierto y las Organizaciones Internacionales 376
11.6.1 Las Naciones Unidas y el Código Abierto 377
11.6.2 La Comisión Europea se Compromete a Liberar Todo el Software que Pueda Beneficiar a la Sociedad 378

12 Apéndice B: Máquinas Virtuales **381**
12.1 Tipos de Máquinas Virtuales 382
12.2 Técnicas de Virtualización 383
12.3 Otras Formas de Virtualización 384
12.4 Aplicaciones de las Máquinas Virtuales de Sistema 386
12.5 Ventajas y Desventajas 388
12.5.1 Ventajas 388
12.5.2 Desventajas 390
12.5.3 Consideraciones Técnicas y Legales de la Virtualización 391
12.6 Máquinas Virtuales en la Educación, Ciencias e Ingeniería . . 392
12.7 ¿Qué Necesito para Crear y Usar una Máquina Virtual? 395
12.8 ¿Cómo Funciona una Máquina Virtual? 397
12.9 Aplicaciones y Paquetes Disponibles 398
12.10Acceso a Datos Desde una Máquina Virtual 404
12.11Desde la Nube 405

13 Bibliografía **408**

1 Introducción

La Carrera de Actuaría prepara a actuarios (véase [1]), estos son profesionistas que estudian, plantean, formulan y aplican modelos de contenido matemático, con el fin de proveer información para la planeación, previsión y la toma de decisiones, para resolver problemas económicos y sociales que involucran riesgos. Los egresados están capacitados para intervenir en ámbitos que van desde el demográfico y financiero hasta el ecológico y administrativo para interactuar con los profesionistas que ahí se desempeñen. Su campo de trabajo está en los sectores públicos o de la administración pública descentralizada, así como en el sector privado en compañías aseguradoras, despachos de consultoría actuarial y estadística, de cómputo e informática y de finanzas.

Por lo anterior, un eje fundamental de desarrollo, es el que se refiere a la formación en cómputo, hoy día, ante los retos que el vertiginoso y dinámico cambio que enfrenta el mundo global en que vivimos, ante las exigencias de la sociedad de la información se requiere el manejo de las Tecnologías de la Información y de la Comunicación (TIC) por ello, el modelo educativo de cualquier Carrera Universitaria y en particular la Carrera de Actuaría en la Facultad de Ciencias de la UNAM, tiene la necesidad imperiosa atender una formación computacional como parte integral de una formación omnilateral de los educandos; por ello, la Facultad de Ciencias cuenta, para lograr este objetivo, con asignaturas, Aulas y Talleres de cómputo para ponerse al día en el manejo de esta importante herramienta. Las Aulas y Talleres de cómputo del Tlahuizcalpan cuentan con equipo y programas actualizados que permiten estar a la vanguardia y que facilitan el trabajo académico en las materias que cursan los estudiantes.

1.1 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.

1.1.1 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, existen 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 apéndice 11.1) 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.

1.1.2 Software Libre

El Software libre (véase apéndice 11.2) son programas de cómputo -el 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.

¿Que es el Software Libre? La definición exacta y sus diversas variantes se plasman en el apéndice 11, pero podemos entender el espíritu a través de los documentos de la fundación para el Software libre (véase [11], [12], [4], [5], [3] y [7]). 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 [11], [10] y [8]). Algunos han conseguido un uso y alta calidad, por ejemplo el compilador GCC (véase [13]), el Kernel de Linux (véase [14]) y el sistema operativo Debian GNU/Linux (véase [15]) y Android (véase [16]). Mientras que otros proyectos han caído en el olvido, pero en la gran mayoría, se tiene copia del código fuente que permitiría a quienes esten 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 ¿que 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 [8] y [9]) -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 [8] y [9]).

En los últimos años, muchos proyectos han pasado de ser simples programas en línea de comandos a complejas aplicaciones multiplataforma -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 ejecura 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.

1.2 El Cómputo en la Carrera de Actuaría

Existe una gran variedad de programas de cómputo que permite automatizar, la cada vez más creciente cantidad de tareas inherentes al desarrollo de las actividades profesionales y de investigación; y en particular en la Carrera de Actuaría.

Es común que las grandes y pequeñas empresas compren programas de cómputo especializados -algunos con un alto costo comercial- para invertir lo menos posible en el desarrollo de herramientas computacionales que satisfagan sus necesidades. Donde es común que una empresa desarrolle más de una versión del programa, donde al menos una versión es libre y generalmente usada como versión de desarrollo y prueba; y las demás versiones sean propietarias y sean comercializadas, dando como servicios adicionales el soporte y capacitación del mismo. La capacitación en el uso de dichos programas es cada vez más, delegada a las formadoras de recursos humanos, entre las que destacan las Universidades.

La capacitación de capital humano especializado en uno o más paquetes de cómputo, requiere por un lado el conocimiento teórico que sustente el funcionamiento del paquete y por otro lado contar con la plataforma computacional adecuada -Hardware y Software- para correr dicho paquete. El creciente costo monetario de la licencia de uso de un paquete de cómputo -y no necesariamente incluye las versiones siguientes del mismo- implica un gasto no justificable para la gran mayoría de las Universidades a nivel mundial.

En las Universidades, no se busca enseñar el uso de uno o más paquetes de cómputo per se, la enseñanza de herramientas de cómputo, es un proceso inherente a la adquisición de conocimiento de las diferentes materias que es necesario cursar para egresar de una carrera universitaria. Así, mediante el uso de uno o más paquetes similares de bajo o nulo costo se le puede enseñar los conocimientos necesarios que le permitirán al educando conocer en poco tiempo las peculiaridades de los paquetes especializados y de alto costo usados en las empresas.

El Software libre tiene miles de proyectos actualmente activos (véase [8] y [9]), estos tratan de satisfacer la gran mayoría de las necesidades de los usuarios noveles y avanzados; y por el uso de dicho Software no es necesario pagar grandes cantidades de dinero. Además, todas las actualizaciones y nuevas versiones de los paquetes por lo general son puestas en la red para que se puedan descargar y ser usadas por cualquier usuario, sin recurrir a gastos onerosos.

Esto queda de manifiesto en uno de los proyectos insignia del Software libre, el Kernel Linux (véase [14]), que en su versión 3.10 cuenta con más de quince millones de líneas de código, que ha sido creado desde 1990 hasta la fecha por miles de programadores distribuidos por todo el mundo e intercomunicados casi exclusivamente por Internet. Este proyecto es la base de diversas distribuciones de sistemas operativos -entre ellos de Android, Ubuntu y Debian GNU/Linux entre otras distribuciones- que corren en aparatos tan heterogéneos como electrodomésticos, teléfonos celulares, tabletas, computadoras portátiles y de escritorio, así como en la mayoría de las supercomputadoras. Además, estos sistemas operativos corren en una gran gama de procesadores, tanto de última generación como en los ya clásicos 80386.

Basándose en los proyectos de Software libre, existen diversas distribuciones de sistemas operativos como Debian GNU/Linux (véase [15]) que integra más de cuarenta mil aplicaciones todas ellas de Software libre que permite correr todos los programas de cómputo de uso común -dado que también soportan la virtualización (véase [42], [43], [41] y [40]), es posible correr otros

sistemas operativos como Windows dentro del propio sistema operativo de Software libre-.

En el caso de Software comercial o propietario como MATLAB (véase [29]), SAS (véase [35]), SPSS (véase [36]), Microsoft Office (véase [18]) entre otros, se ha invertido una gran cantidad de trabajo y recursos económicos para generar interfaces de usuario pulidas y perfectamente integradas, así como, una gran cantidad de ayuda y en algunos casos con un asistente virtual en el mismo paquete para que guíen al usuario en su uso. La documentación integrada del paquete, se complementa con una gran cantidad de ayuda en línea, libros, artículos y páginas blancas -en algunos casos en más de 80 idiomas- que detallan como usar el Software para solucionar una gran gama de problemas de diversos grupos de usuarios, tanto usuarios ocasionales como especializados. En el caso de que el Software sea usado en ambientes universitarios, estos cuentan con guías, tutoriales o centros de entrenamiento que capacitan en cada uno de los tópicos necesarios para dominar el paquete hasta llegar dominar las opciones avanzadas del mismo.

En contraste, en el Software libre, es común que en muchos proyectos incipientes, la documentación, ayuda en línea e interfase gráfica es un aspecto poco logrado. En muchos casos, los entornos de trabajo (IDEs) son dejados a otros proyectos. De tal forma que los creadores -programadores- del paquete se centren en el desarrollo computacional de la infraestructura base del paquete, delegando a otros, las partes que tienen que ver con el usuario final y la documentación del mismo. Por ejemplo el paquete de GNUPlot que se encarga de la visualización de gráficas en dos y tres dimensiones es usado por decenas de paquetes en línea de comandos -por ejemplo Python, Perl, C y C++- que cuando requieren hacer gráficas se la solicitan a dicho paquete, de esta forma se reutiliza lo ya creado y se simplifica con mucho el desarrollo del paquete, permitiendo a los desarrolladores centrarse en las características que necesitan programar para innovar y satisfacer las crecientes necesidades de los usuarios.

Por el incipiente desarrollo de las aplicaciones de Software libre que compiten con el Software privativo o comercial que estamos tan acostumbrados a usar -programas para cálculo numérico y simbólico, estadístico y de ofimática, entre otros, comúnmente usados para la enseñanza y resolución de problemas actuariales-, al menos en un corto plazo, el Software libre parece no ser una opción viable de reemplazo. Pero ya son lo suficientemente maduras para ser tomadas en cuenta, en un, cada vez más competitivo ambiente de trabajo multiplataforma, en donde es necesario que el usuario -estudiante y eventual

profesionista- tenga acceso todos y cada uno de los paquetes de cómputo que se le muestran en clase, así como en las prácticas profesionales, que le permitan adquirir soltura y expertes en el uso de los mismos.

1.2.1 Cursos que Usan Cómputo

En la Facultad de Ciencias, cada semestre, se imparten decenas de cursos -algunos compartidos por otras carreras y otros específicos de la carrera de Actuaría de sus diferentes planes de estudios vigentes 2006 y 2015- y desde hace varios años, crece semestre a semestre el número de cursos[1] que solicitan hacer uso de equipos de cómputo y tener acceso a múltiples versiones de paquetería especializada (véase apéndice 10.1), algunos de estos cursos son (véase [2]):

Primer Semestre

- *Cálculo Diferencial e Integral I*
- *Geometría Analítica I*
- *Problemas Socio-Económicos de México*

Segundo Semestre

- *Cálculo Diferencial e Integral II*
- *Geometría Analítica II*
- *Matemáticas Financieras*
- *Programación I*

Tercer Semestre

- *Cálculo Diferencial e Integral III*
- *Probabilidad I*
- *Programación II*

[1] Los cursos solicitan, desde el uso de un equipo de cómputo y cañón para proyectar, hasta la asignación de una máquina por estudiante que tenga instalado múltiples programas especializados corriendo en más de un sistema operativo.

Cuarto Semestre

- *Cálculo Diferencial e Integral IV*
- *Finanzas I*
- *Matemáticas Actuariales del Seguro de Personas I*
- *Probabilidad II*

Quinto Semestre

- *Estadística I*
- *Finanzas II*
- *Investigación de Operaciones*
- *Matemáticas Actuariales del Seguro de Personas II*

Sexto Semestre

- *Economía I*
- *Estadística II*
- *Matemáticas Actuariales del Seguro de Daños*
- *Procesos Estocásticos*

Séptimo Semestre

- *Análisis Numérico*
- *Demografía I*
- *Estadística III*
- *Seguridad Social*

Octavo Semestre

- *Pensiones Privadas*

- *Teoría del Riesgo*

Optativas

- *Administración de Riesgos*
- *Administración de Riesgos Financieros*
- *Análisis de Regresión*
- *Análisis Multivariado*
- *Bases de Datos*
- *Carteras de Inversión*
- *Demografía II*
- *Econometría I*
- *Econometría II*
- *Estadística Bayesiana*
- *Fianzas*
- *Inteligencia Artificial*
- *Muestreo*
- *Planeación Estratégica*
- *Productos Financieros Derivados I*
- *Productos Financieros Derivados II*
- *Reaseguro*
- *Redes de Computadoras*
- *Seminario de Aplicaciones Actuariales*
- *Seminario de Matemáticas Actuariales Aplicadas*
- *Teoría de Gráficas*
- *Teoría de Juegos en Economía*
- *Valuación de Opciones*

1.3 Paquetes de Cómputo de Uso Común

Dada la diversidad de cursos en la carrera de Actuaría y la gran cantidad de profesores y ayudantes de los mismos, el abanico de paquetes de cómputo solicitados es grande (véase apéndice 10.1), entre los que destacan (véase [2]):

- Java JRE y JDK
- Code Blocks IDE
- NetBeans IDE
- DrJava IDE
- IntelliJ IDEA
- BlueJ IDE
- SciTE
- JetBrains IDE
- SharpDevelop IDE
- Alice
- DFD
- Turbo C IDE
- Developer Studio-Fortran IDE
- Microsoft Visual Studio
- Microsoft Windows SDK
- Compaq Visual Fortran
- Microsoft Office
- Libre Office
- OpenOffice
- MathType

- Scientific WorkPlace
- Microsoft SQL Server
- PostgreSQL
- SPSS
- PSPP
- SAS
- Vensim PLE
- Statgraphics
- GPower
- EViews
- Systat
- Stata
- Statistica
- ITSM2000
- R
- Tinn-R
- RStudio
- Gretl
- MATLAB
- Scilab
- Octave
- FreeMat
- Maple

- Mathematica
- NetLogo
- GeoGebra
- Compresores y descompresores de archivos Winzip, WinRAR, 7-zip
- SSH Secure File Transfer
- PDFCreator
- Adobe Reader
- Navegadores de páginas Web: Internet Explorer, Google Chrome, Mozilla, Konquerror

Estos paquetes pueden ser clasificados de forma burda en:

- Programas de Cálculo Numérico
- Programas de Estadística
- Programas Ofimáticos
- Otros Programas

En el presente trabajo, nos centraremos en los rubros más usados, pero el tipo de programas usados, las plataformas de ejecución soportada y las diferentes versiones de un mismo paquete que solicitan los profesores y ayudantes, crecen constantemente, así mismo, su complejidad y en su caso, el costo monetario de las licencias de uso también se incrementa de manera onerosa.

1.3.1 Programas de Cálculo Numérico

Los paquetes de cálculo numérico, son programas matemáticos que ofrecen un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio. Entre sus prestaciones básicas se hallan:

- Manejo de números reales y complejos

- La manipulación de vectores y matrices tanto reales como complejas
- Manejo de funciones elementales y especiales
- Resolución de problemas de álgebra lineal
- Resolución de ecuaciones no lineales
- La representación de datos y funciones
- La implementación de algoritmos
- Integración de funciones
- Máximos y mínimos de funciones
- Manipulación de polinomios
- Integración de ecuaciones diferenciales
- Graficación de funciones en 2D y 3D
- La comunicación con programas en otros lenguajes y con otros dispositivos Hardware

1.3.2 Programas de Estadística

Los paquetes estadísticos, son programas matemáticos que ofrecen un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio. Entre sus prestaciones básicas destacan:

- Análisis de datos mediante operadores para cálculos sobre arreglos, matrices y/o Tablas
- Tablas Cruzadas
- Reordenamiento de Datos
- Análisis de la Varianza (ANOVA)
- Frecuencias
- Estadística Descriptiva

- Estadística lineal
- Estadística no lineal
- Estadística Biestadística
- Pruebas Estadísticas Clásicas
- Análisis de Serie de Temporales
- Modelos de Regresión
- Clasificación
- Fiabilidad
- Categorías
- Clustering
- Validación de Datos
- Tendencias
- Gráficos y Diagramas

1.3.3 Programas Ofimáticos

En la actualidad, los llamados paquetes ofimáticos, no son otra cosa que programas de cómputo integrado, que permiten automatizar multiples tareas que permiten idear, crear, manipular, transmitir, almacenar información necesaria en una oficina. Entre sus prestaciones básicas destacan:

- Procesamiento de Textos -con formato enriquecido y notación científica-
- Hojas de Cálculo
- Bases de Datos
- Herramientas de Presentación y Multimedia

1.3.4 Otros Programas

Dentro del abanico de programas que son usados en la carrera de Actuaría, y que no tienen cabida en los rubros anteriores, destacan:

- Lenguajes de Programación
- Entornos de Desarrollo Integrados y editores para Programación
- Cálculo Simbólico
- Manipulación de Gráficos
- Navegadores Web, compresores y descompresores de archivos, etc.

1.4 Agradecimientos

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

En la realización de este texto se han revisado -en la mayoría de los casos indicamos la referencia, pero pudimos omitir varias de ellas, por lo cual pedimos una disculpa- múltiples páginas Web, artículos técnicos, libros, entre otros materiales bibliográficos, los más representativos y de libre acceso los ponemos 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.

Este proyecto fue posible gracias al apoyo recibido por la Facultad de Ciencias de la Universidad Nacional Autónoma de México (UNAM) y al tiempo robado a nuestras actividades académicas, principalmente durante el período de confinamiento de los años 2020 a 2022.

2 Sistemas Operativos

Actualmente tenemos 3 grandes sistemas operativos en el mercado[2]:

- Windows
- Unix
- GNU[3]/Linux

De los cuales, sus dignos representantes son: Windows, macOS, iOS, Android, Chrome OS y GNU/Linux con todas sus diferentes distribuciones[4]. Y sin temor a equivocarnos aseguramos que Android es la distribución de GNU/Linux más popular e iOS es el más popular de los UNIX.

¿Qué es un Sistema Operativo? El conjunto de programas informáticos que permiten la administración eficaz de los recursos de una computadora es conocido como sistema operativo o Software de sistema. Estos programas comienzan a trabajar apenas se enciende el equipo, ya que gestionan el Hardware desde los niveles más básicos y permiten además la interacción

[2] Cuotas de mercado de diferentes sistemas operativos:

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

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

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

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

con el usuario. Cabe destacar que los sistemas operativos no funcionan sólo en las computadoras. Por el contrario, este tipo de sistemas se encuentran en la mayoría de los dispositivos electrónicos que utilizan microprocesadores: el Software de sistema posibilita que el dispositivo cumpla con sus funciones -por ejemplo, un teléfono móvil o un reproductor de DVD-.

El sistema operativo cumple con cinco funciones básicas:

- Proporciona la interfaz del usuario -gráfica o de texto-
- La administración de recursos
- La administración de archivos
- La administración de tareas
- El servicio de soporte y utilidades

En cuanto a la interfaz del usuario, el sistema se encarga de que el usuario pueda ejecutar programas, acceder a archivos y realizar otras tareas con la computadora. La administración de recursos permite el control del Hardware, incluyendo los periféricos y la red. El Software de sistema también se encarga de la gestión de archivos, al controlar la creación, la eliminación y el acceso a los mismos, así también, de la administración de las tareas informáticas que ejecutan los usuarios finales. Por último, podemos mencionar que el servicio de soporte se encarga de actualizar las versiones, mejorar la seguridad del sistema, agregar nuevas utilidades, controlar los nuevos periféricos que se agregan a la computadora y corregir los errores del Software.

Tipos de Sistemas Operativos en Función de la Administración de las Tareas Podemos distinguir dos clases de sistemas operativos en función de cómo administran sus tareas, pueden ser:

Sistemas Operativos Monotarea: son sistemas operativos que únicamente cuentan con la capacidad para realizar una tarea al mismo tiempo. Son los sistemas más antiguos, que también llevan aparejados un CPU de menor capacidad. En estos casos, si el equipo está imprimiendo, no atenderá a las nuevas órdenes, ni será capaz de iniciar un nuevo proceso hasta que el anterior haya finalizado.

Sistemas Operativos Multitarea: son los sistemas operativos más modernos, con capacidad para el procesamiento de varias tareas al mismo tiempo. Cuentan con la capacidad para ejecutar varios procesos en uno o más procesadores, por lo que existe la posibilidad de que sean utilizados por varios usuarios al mismo tiempo, y podrían aceptar múltiples conexiones a través de sesiones remotas.

Tipos de Sistemas Operativos en Función de la Administración de los Usuarios También es posible realizar una división de los sistemas operativos en función de la forma en la que se administran los usuarios, como vemos a continuación:

Sistema de Administración Monousuario: sólo pueden gestionar un usuario al mismo tiempo. Así, a pesar de que varios usuarios pueden tener acceso al sistema, solo un usuario puede acceder para realizar y ejecutar operaciones y programas.

Sistemas de Administración Multiusuario: se refiere a todos aquellos sistemas operativos que permiten el empleo de sus procesamientos y servicios al mismo tiempo. Así, el sistema operativo cuenta con la capacidad de satisfacer las necesidades de varios usuarios al mismo tiempo, siendo capaz de gestionar y compartir sus recursos en función del número de usuarios que estén conectados a la vez.

¿Qué Sistema Operativo Usar? ¿Mac o Microsoft? ¿Windows o Linux? ¿Android o iOS? Son preguntas frecuentes que todos nos hemos hecho alguna vez, y es que elegir un sistema operativo, una computadora o un dispositivo móvil no es tan simple. O al menos no lo era años atrás. En la actualidad las diferencias entre sistemas operativos de escritorio son cada vez menos, hasta el punto que prácticamente cualquier servicio Online es compatible con Windows, Mac y GNU/Linux y las principales firmas de Software crean aplicaciones para las tres plataformas principales, salvo excepciones. Lo mismo empieza a ocurrir con el Hardware.

Poco tendremos que decir del sistema operativo de Apple, Mac o iOS (ambos son derivados de Darwin BSD que es un sistema operativo tipo UNIX), ya que son los sistemas operativos más bonitos y que mejores resultados han dado a todos los usuarios que los han probado. Mac es un sistema pensado

para los profesionales de los sectores que necesitan de un equipo de cómputo que sea capaz de todo, como los desarrolladores, programadores, diseñadores, periodistas, fotógrafos, músicos, DJ's y muchos más empleos que se benefician de este sistema operativo.

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

Sin embargo, llegamos a Linux, el gran desconocido por muchos. Un sistema operativo mucho más versátil que Windows y que puede ser igual o más profesional que Mac. Sin embargo, la ventaja que tienen estos dos sistemas operativos, es que vienen ya preparados y configurados para el tipo de mercado al que van dirigidos, pero GNU/Linux no.

Esto es una ventaja y una desventaja al mismo tiempo, ya que si tenemos práctica, podemos hacer que el sistema operativo se adapte a nuestras necesidades sin problemas, pero si no tienes práctica, puede que sea demasiado lo que tienes que configurar.

Cuota de Mercado para los Sistemas Operativos Agosto es uno de los meses en los que miles de compañías analizan el tráfico que les llega de usuarios a sus páginas Web y desde que plataformas llegan, según un informe de International Data Corporation (https://www.idc.com) y The Linux Foundation (https://www.linuxfundation.org) en el año 2024 tenemos:

- En el segmento de los sistemas operativos de escritorio basados en Linux ha subido su cuota de mercado llegando al 3%, esto no parecerá mucho, pero si nos fijamos bien, vemos que Mac tiene un 7.5% -basado en Unix-, Chrome OS -usa el Kernel de Linux- tiene 10.8% y Windows el resto.

- En el segmento de teléfonos inteligentes (SmartPhones) y tabletas basadas en Android -usa el Kernel de Linux- tiene 86 % , iOS tiene 13.9 % -basado en Unix- y menos del 1% el resto de los sistemas operativos.

- En el segmento de servidores se estima que más del 60% de los servidores a nivel mundial usan Linux, 1% usan Unix y el resto Windows. Pero en los principales servidores del mundo (un millón) 96 % usan Linux.

- El 90% de toda la infraestructura Cloud corre usando Linux. Es de destacar que en el servicio de servidores Microsoft Azure, el sistema predominante es Linux.
- En el segmento de supercomputadoras, Linux tiene la cuota más importante del mercado; es utilizado en los Top 500 sistemas de supercómputo de alto desempeño del mundo[5].

Hay que decir, que hoy en día y tal y como están las cosas, no existe un sistema operativo que sea definitivo. Así que la pregunta de si GNU/Linux[6] es mejor que Windows o Mac no tiene sentido, ya que cada sistema operativo tiene sus pros y sus contras.

Pero la disyuntiva sigue ahí. ¿Debemos usar Windows en nuestro equipo de cómputo?, ¿nos conviene pasarnos a Linux?. Hay razones a favor y en contra para todos los gustos.

Número de Líneas del Código Fuente de un Sistema Operativo
Pese a que existen múltiples variantes de cada sistema operativo, se han dado a conocer los números de líneas de código fuente que componen la vertiente más usada de algunos sistemas operativos:

- Microsoft Windows 3.1 (Abril de 1992): 3 millones de líneas ($200 USD en 1992)
- Microsoft Windows 95 (Agosto de 1995): 15 millones de líneas (Home $109.95 y Pro$ 209.95 USD en 1995)
- Microsoft Windows NT 4.0 (Julio 1996): 12 millones de líneas (5 usuarios $809, 10 usuarios $1,129 USD en 1996)

[5] Existe el Ranking de las 500 supercomputadoras más poderosas del mundo (esta se actualiza cada seis meses en junio y noviembre) y puede ser consultada en:
https://top500.org
La cuota de supercomputadoras con GNU/Linux ha sido de: 2012 (94%), 2013 (95%), 2014 (97%), 2015 (97.2%), 2016 (99.6%), 2017 (99.6%), 2018 (100%), 2019 (100%), 2020 (100%).

[6] Los resultados de GNU/Linux son muy satisfactorios para los desarrolladores y partícipes de la comunidad Linux, pero todavía hace falta mucho por hacer para que tenga una cuota significativa en el escritorio y esto sólo será posible si los distribuidores de equipo generan un esquema más agresivo para vender máquinas con Linux preinstalado.

- Microsoft Windows 2000 (Febrero 2000): 29 millones de líneas (Pro $319 USD en 2000)
- Microsoft Windows XP (Octubre 2001): 45 millones de líneas (Home $175 US y Pro $ 255 USD en 2001)
- Microsoft Windows Vista (Enero 2007): 50 millones de líneas (Home Premium $239.99 USD en 2007)
- Microsoft Windows 7 (Octubre 2009): 40 millones de líneas (Home Premium $199.99 USD en 2009)
- Microsoft Windows 8 (Octubre 2012): 60 millones de líneas (Home $119.99 US y Pro $ 199.99 USD en 2013)
- Microsoft Windows 10 (Julio 2015): 60 millones de líneas sin Cortana y 65 millones de líneas con Cortana ($139.99 USD en 2019)
- Sun Solaris (Octubre de 1998) 7.5 millones de líneas
- Red Hat Linux 6.2 (Marzo de 2000): 17 millones de líneas
- Red Hat Linux 7.1 (Abril de 2001): 30 millones líneas
- Red Hat Linux 8.0 (Septiembre de 2002): 50 millones de líneas
- Fedora Core 4 (Mayo de 2005): 76 millones de líneas
- Fedora 9 (Mayo del 2008): 205 millones de líneas
- Debian GNU/Linux 3.0 "Woody" (Julio de 2002): 105,000,000 líneas
- Debian GNU/Linux 3.1 "Sarge" (Junio de 2005); 229,500,000 líneas
- Debian GNU/Linux 7 "Wheezy" (Mayo 2013): 419 millones de líneas
- Debian GNU/Linux 10 "Buster" (Julio 2019): 1,077,110,982 líneas
- Debian GNU/Linux 11 "Bullseye" (Agosto 2021): 1,152,960,944 líneas
- Debian GNU/Linux 12 "bookworm" (Junio 2023): 1,341,564,204 líneas
- Kernel Linux 0.01 (Septiembre 1991): 8,413 líneas

- Kernel Linux 1.0 (Marzo 1994): 176,250 líneas
- Kernel Linux 2.6 (Diciembre 2003): 5,475,685 líneas
- Kernel Linux 4.12 (Julio 2017): 24 millones líneas
- Kernel Linux 5.8 (Agosto 2020): 28 millones líneas
- Kernel Linux 5.14 (Julio 2021): 29.7 millones de líneas

El Kernel o Núcleo Es un componente fundamental de cualquier sistema operativo. Es el encargado de que el Software y el Hardware de cualquier equipo de cómputo puedan trabajar juntos en un mismo sistema, para lo cual administra la memoria de los programas y procesos ejecutados, el tiempo de procesador que utilizan los programas, o se encarga de permitir el acceso y el correcto funcionamiento de periféricos y otros elementos físicos del equipo.

Kernel de Linux el núcleo del sistema operativo Linux/Unix (llamado Kernel) es un programa escrito casi en su totalidad en lenguaje C, con excepción de una parte del manejo de interrupciones, expresada en el lenguaje ensamblador del procesador en el que opera, el Kernel reside permanentemente en memoria y alguna parte de él está ejecutándose en todo momento. Pero es muy común confundir al Kernel de Linux con una distribución como Debian y Ubuntu, Linux solo es un núcleo (hay otros como son FreeBSD, NetBSD y Hurd).

Durante mucho tiempo el núcleo Linux solo funcionaba en la serie de máquinas x86 de Intel, desde el 386 en adelante. Sin embargo, hoy día esto ya no es cierto. El núcleo Linux se ha adaptado a una larga y creciente lista de arquitecturas. Siguiendo esos pasos, la distribución Debian GNU/Linux se ha adaptado a estas plataformas. En general este proceso tiene un comienzo difícil (hay que conseguir que la *libc* y el enlazador dinámico funcionen sin trabas), luego sigue un trabajo relativamente largo y rutinario, de conseguir recompilar todos los paquetes bajo las nuevas arquitecturas.

Debian GNU/Linux es un sistema operativo, no un núcleo (en realidad es más que un SO, ya que incluye miles de aplicaciones). Para probar esta afirmación, aun cuando la mayor parte de adaptaciones se hacen sobre núcleos Linux, también existen adaptaciones basadas en los núcleos FreeBSD, NetBSD y Hurd.

Linux es multiprogramado, dispone de memoria virtual, gestión de memoria, conectividad en red y permite bibliotecas compartidas. Linux es multiplataforma y es portable a cualquier arquitectura siempre y cuando está disponga de una versión de GCC compatible.

La parte de un sistema operativo que se ejecuta sin privilegios o en espacio de usuario es la biblioteca del lenguaje C, que provee el entorno de tiempo de ejecución, y una serie de programas o herramientas que permiten la administración y uso del núcleo y proveer servicios al resto de programas en espacio de usuario, formando junto con el núcleo el sistema operativo.

En un sistema con núcleo monolítico como Linux la biblioteca de lenguaje C (*libc*) consiste en una abstracción de acceso al núcleo. Algunas bibliotecas como la biblioteca de GNU proveen funcionalidad adicional para facilitar la vida del programador y usuario o mejorar el rendimiento de los programas. En un sistema con micronúcleo la biblioteca de lenguaje C puede gestionar sistemas de archivos o controladores además del acceso al núcleo del sistema.

A los sistemas operativos que llevan Linux se les llama de forma genérica distribuciones Linux. Estas consisten en una recopilación de software que incluye el núcleo Linux y el resto de programas necesarios para completar un sistema operativo. Las distribuciones más comunes son de hecho distribuciones GNU/Linux o distribuciones Android. El hecho de que compartan núcleo no significa que sean compatibles entre sí. Una aplicación hecha para GNU/Linux no es compatible con Android sin la labor adicional necesaria para que sea multiplataforma.

Las distribuciones GNU/Linux usan Linux como núcleo junto con el entorno de tiempo de ejecución del Proyecto GNU y una serie de programas y herramientas del mismo que garantizan un sistema funcional mínimo. La mayoría de distribuciones GNU/Linux incluye software adicional como entornos gráficos o navegadores Web así como los programas necesarios para permitirse instalar a sí mismas. Los programas de instalación son aportados por el desarrollador de la distribución. Se les conoce como gestores de paquetes. Los creadores de una distribución también se pueden encargar de añadir configuraciones iniciales de los distintos programas incluidos en la distribución.

Las distribuciones Android incluyen el núcleo Linux junto con el entorno de ejecución y herramientas del proyecto AOSP de Google. Cada fabricante de teléfonos dispone de su propia distribución de Android a la cual modifica, elimina o añade programas extra: interfaces gráficas, tiendas de aplicaciones y clientes de correo electrónico son algunos ejemplos de programas suscepti-

bles de ser añadidos, modificados o eliminados. Además de las distribuciones de los fabricantes de teléfonos existen grupos de programadores independientes que también desarrollan distribuciones de Android. LineageOS y Replicant son dos ejemplos de distribuciones Android independientes.

Los usuarios de Linux/Unix estamos acostumbrados a hablar y oír hablar sobre su Kernel[7], el cual puede actualizarse y manipularse en cualquier distribución. Sin embargo, en un sistema operativo tan centrado en el usuario y la sencillez como Windows, su Kernel es un gran desconocido.

Kernel de Windows en la década de los noventa Microsoft estaba basando sus sistemas operativos en los Kernel Windows 9x, donde el código básico tenía muchas similitudes con MS-DOS. De hecho necesitaba recurrir a él para poder operar. Paralelamente, Microsoft también estaba desarrollando otra versión de su sistema dirigido a los servidores llamada Windows NT.

Ambas versiones de Windows fueron desarrollándose por separado. Windows NT era más bien una jugada a largo plazo, una tecnología para ir desarrollando para los Windows del mañana, y en el año 2000 dieron un nuevo paso en esa dirección. A la versión 5.0 de NT la llamaron Windows 2000, y se convirtió en un interesante participante en el sector empresarial.

Tras ver la buena acogida que tuvo, Microsoft decidió llevar NT al resto de usuarios para que ambas ramificaciones convergien. Lo hicieron en octubre del 2001 con la versión 5.1 de Windows NT, que llegó al mercado con el nombre de Windows XP. Por lo tanto, esta versión marcó un antes y un después no sólo por su gran impacto en el mercado, sino porque era el principio de la aventura del Kernel Windows NT en el mundo de los usuarios comunes.

Desde ese día, todas las versiones de Windows han estado basadas en este Kernel con más de 20 años de edad. La versión 5.1.2600 fue Windows XP, la 6.0.6002 fue Windows Vista, y la 6.1.7601 Windows 7. Antes hubo otros

[7] En el caso de los sistemas derivados de Unix y Linux el Kernel lo podemos encontrar en el directorio /boot/, este directorio incluye todos los ejecutables y archivos que son necesarios en el proceso de arranque del sistema y deben ser utilizados antes que el Kernel empiece a dar las órdenes de ejecución de los diferentes módulos del sistema, aquí también es donde reside el gestor de arranque.

En algunas distribuciones al usar un gestor de volúmenes lógico (Logical Volume Manager, LVM) se genera un esquema de particiones con el directorio boot en una partición aparte.

Windows Server 2008 y 2003, y después llegaron las versiones de NT 6.2.9200 llamada Windows 8, la 6.3.9600 o Windows 8, la NT 10.0, también conocida como Windows 10 y finalmente Windows 11.

La principal característica del Kernel de Windows NT es que es bastante modular, y está basada en dos capas principales, la de usuario y la de Kernel. El sistema utiliza cada una para diferentes tipos de programa. Por ejemplo, las aplicaciones se ejecutan en el modo usuario, y los componentes principales del sistema operativo en el modo Kernel. Mientras, la mayoría de los Drivers suelen usar el modo Kernel, aunque con excepciones.

Es por eso que se refieren a él como Kernel híbrido, pero sobre todo también porque permite tener subsistemas en el espacio del usuario que se comunicaban con el Kernel a través de un mecanismo de intercomunicación de procesos IPC (Interprocess Communication).

Cuando ejecutas una aplicación, está accede al modo usuario, donde Windows crea un proceso específico para la aplicación. Cada aplicación tiene su dirección virtual privada, ninguna puede alterar los datos que pertenecen a otra y tampoco acceder al espacio virtual del propio sistema operativo. Es por lo tanto el modo que menos privilegios otorga, incluso el acceso al Hardware está limitado, y para pedir los servicios del sistema las aplicaciones tienen que recurrir a la interfaz de programación de aplicaciones API (Application Programming Interface) de Windows.

El modo núcleo en cambio es ese en el que el código que se ejecuta en él tiene acceso directo a todo el Hardware y toda la memoria del equipo. Aquí todo el código comparte un mismo espacio virtual, y puede incluso acceder a los espacios de dirección de todos los procesos del modo usuario. Esto es peligroso, ya que si un Driver en el modo Kernel modifica lo que no debe, podría afectar al funcionamiento de todo el sistema operativo.

Este modo núcleo está formado por servicios Executive, como el controlador de Caché, el gestor de comunicación, gestor de E/S, las llamadas de procedimientos locales, o los gestores de energía y memoria entre otros. Estos a su vez están formados por varios módulos que realizan tareas específicas, controladores de núcleo, un núcleo y una capa de abstracción del Hardware HAL (Hardware Abstraction Layer).

Diferencias entre los Kernel de Linux y Windows La principal diferencia entre el Kernel de los sistemas operativos Windows y el de Linux está en su filosofía. El desarrollado por el equipo de Linus Torvalds es de

código abierto y cualquiera puede usarlo y modificarlo, algo que le sirve para estar presente en múltiples sistemas operativos o distribuciones GNU/Linux. El Kernel de Microsoft[8] en cambio es bastante más cerrado, y está hecho por y para el sistema operativo Windows.

En esencia, en Linux adoptaron los principios de modularidad de Unix y decidieron abrir el código y las discusiones técnicas. Gracias a ello, Linux ha creado una comunidad meritocrática de desarrolladores, una en la que todos pueden colaborar y en la que cada cambio que se sugiere se debate con dureza para desechar las peores ideas y quedarse con las mejores. También se halaga a quienes consiguen mejorar las funcionalidades más veteranas.

Mientras, en Windows no funciona así, los responsables del Kernel no ven con buenos ojos que se hagan propuestas que se desvíen del plan de trabajo, y asegura que hay pocos incentivos para mejorar las funcionalidades existentes que no sean prioritarias.

Esto hace, a ojos de ese antiguo desarrollador, que al dársele mayor importancia a cumplir planes que a aceptar cambios que mejoren la calidad del producto, o al no tener tantos programadores sin experiencia, el Kernel de Windows NT siempre esté un paso por detrás en estabilidad y funcionalidades.

A nivel técnico existen similitudes entre ambos. Los dos núcleos controlan el Software del sistema de bajo nivel y las interacciones con el Hardware del ordenador a través de la capa de abstracción de Hardware (HAL). El HAL es un elemento del sistema que funciona como interfaz entre Software y Hardware, y como las API, permite que las aplicaciones sean independientes del Hardware.

Los dos están escritos principalmente en C, y son capaces de manejar

[8]Para conocer la información del Kernel de Windows usando la línea de comandos podemos utilizar el siguiente comando en un cmd shell:

```
systeminfo | findstr /B /C:"OS Name" /C:"OS Version"
```

Y en powershell:

```
Get-CimInstance Win32_OperatingSystem | Select-Object Caption, CSDVersion, ServicePackMajorVersion, BuildNumber | FL
```

o

```
[System.Environment]::OSVersion.Version
```

el almacenamiento en Caché, los controladores de dispositivos, la memoria virtual, los sistemas de archivos, los protocolos de red y las llamadas de sistema. En esencia sus funcionalidades son las mismas, aunque la manera de llevarlas a cabo es diferente.

Así como el Kernel de Windows tiene dos modos, y por lo tanto se le considera híbrido, la gran diferencia es que el de Linux sólo tiene una capa, o sea que es un núcleo monolítico. Eso sí, pese a ser más sencillo en este aspecto, para funcionar correctamente tiene su núcleo dividido en tres subcapas diferentes.

Ambos gestionan los problemas de memoria de forma parecida. Tienen sistemas de "Swapping" para mover un proceso o parte de él temporalmente de la memoria principal a una secundaria de almacenamiento en el caso de que en la principal haya poco espacio. Windows lo hace en los ficheros Pagefile.sys y Swapfile.sys, mientras que Linux lo suele hacer en una partición, aunque también lo puede hacer en uno o varios ficheros o deshabilitarlo.

Por lo tanto, podemos decir que la principal diferencia entre ambos es la manera en que se desarrolla cada uno. Además, el Kernel de Linux es mucho más sencillo, lo cual es bueno para los desarrolladores. Mientras, el de Windows intenta poner una capa de protección en su modo usuario para que los usuarios con menos conocimientos tengan menos posibilidades de dañar el sistema, y su estructura lo hace más estable frente, por ejemplo a fallos del Driver gráfico.

Pero todo esto ya está cambiando, en las últimas versiones de Windows 10 y 11, Microsoft está integrando el Kernel de Linux a su propio Kernel y esto ha permitido usar Linux dentro de Windows de forma nativa gracias al llamado Windows Subsystem for Linux (WSL, WSL2 WSLg), lo cual ha permitido mejorar la estabilidad y desempeño de Windows.

Kernel de Android Durante la última conferencia de Linux Plumbers 2021, Google dio a conocer sobre el éxito de la iniciativa de mover la plataforma Android para usar un Kernel normal de Linux en lugar de usar su propia versión del Kernel, que incluye cambios específicos para la plataforma Android.

Google menciona que dicho cambio de desarrollo es debido a la decisión de pasar después del año 2023 al modelo «Upstream First», que implica el desarrollo de todas las funciones nuevas del Kernel requeridas en la plataforma Android directamente en el Kernel principal de Linux y no en sus ramas separadas (la funcionalidad será primero se promocionará al Kernel principal

y luego se usará en Android, y no al revés).

Para 2023 y 2024, también se planea transferir al núcleo principal de todos los parches adicionales que quedan en la rama del Kernel común de Android.

En cuanto a un futuro próximo, para la plataforma Android 12 prevista para principios de octubre, se ofrecerán compilaciones del Kernel «Generic Kernel Image» (GKI), lo más parecido posible al Kernel 5.10 habitual.

Para estas compilaciones se proporcionará un lanzamiento regular de actualizaciones, que se colocarán en el repositorio ci.android.com. En el Kernel de GKI, las adiciones específicas de Android, así como los controladores relacionados con el Hardware de los fabricantes de equipos originales, se mueven a módulos de Kernel separados.

Esta nueva interfaz, conocida como Kernel Module Kjos, garantizará que la principal diferencia entre la imagen genérica del Kernel de Android (GKI) y la línea principal de Linux, sean solo los ganchos para todos los módulos específicos del proveedor.

Estos módulos no están vinculados a la versión principal del Kernel y se pueden desarrollar por separado, lo que simplifica enormemente el mantenimiento y la transferencia de dispositivos a nuevas ramas del Kernel. Las interfaces necesarias para los fabricantes de dispositivos se implementan en forma de ganchos que le permiten cambiar el comportamiento del Kernel sin realizar cambios en el código.

En total, el Kernel Android 12-5.10 ofrece 194 ganchos comunes, similares a los puntos de seguimiento, y 107 ganchos especializados que le permiten ejecutar controladores en un contexto no atómico. En el Kernel de GKI, los fabricantes de Hardware tienen prohibido aplicar parches específicos al Kernel principal, y los proveedores deben suministrar los componentes para el Hardware de soporte sólo en forma de módulos de Kernel adicionales, en los que se debe garantizar la compatibilidad con el Kernel principal.

Debemos recordar que la plataforma Android desarrolla su propia rama del Kernel: el «Android Common Kernel», sobre la base del cual se forman las compilaciones específicas separadas para cada dispositivo.

Con lo cual, a partir de cada rama de Android, se proporciona a los fabricantes múltiples diseños de Kernel para sus dispositivos. Por ejemplo, Android 11 ofreció una opción de tres núcleos base a la vez: 4.14, 4.19 y 5.4, y para Android 12, se ofrecerán los núcleos base 4.19, 5.4 y 5.10. La variante 5.10 está diseñada como una imagen de Kernel genérica, en la que las capacidades necesarias para los OEM se transfieren al flujo ascendente, se mueven a módulos o se transfieren al Kernel común de Android.

Antes de la llegada de GKI, el Kernel de Android pasó por varias etapas de preparación:

- La primera de ellas era sobre la base de los principales Kernels LTS (3.18, 4.4, 4.9, 4.14, 4.19, 5.4) y de los cuales se creó una bifurcación del «Android Common Kernel», al que se transferían parches específicos para Android (anteriormente, se alcanzaba el tamaño de los cambios varios millones de líneas).

- Después de ello sobre «Android Common Kernel», los fabricantes de Chips como Qualcomm, Samsung y MediaTek forman el SoC Kernel, que incluye complementos para admitir Hardware.

- Finalmente en el «Kernel de SoC», los fabricantes de dispositivos crean el «Kernel de dispositivo», incluidos los cambios relacionados con la compatibilidad con equipos adicionales, pantallas, cámaras, sistemas de sonido, etc.

Este enfoque complicó significativamente la entrega de actualizaciones con la eliminación de vulnerabilidades y la transición a nuevas ramas del Kernel. Si bien Google publica regularmente actualizaciones para su núcleo común de Android, los proveedores a menudo tardan en enviar estas actualizaciones o usan un solo Kernel durante todo el ciclo de vida del dispositivo, generando una alta fragmentación en el ecosistema Android, una obsolescencia anticipada y en el peor de los casos brechas de seguridad.

Otros Kernels GNU/Linux Actualmente existen una gran cantidad de distribuciones de GNU/Linux que vienen muy optimizadas intentando conseguir la mejor desenvoltura de su arquitectura y configuraciones de serie. En el caso de la configuración por omisión de Debian GNU/Linux y Ubuntu, están pensadas para que sean lo más robusta posible y que se use en todas las circunstancias imaginables, por ello están optimizadas de forma muy conservadora para tener un equilibrio entre eficiencia y consumo de energía. Pero es posible agregar uno o más Kernels GNU/Linux generados por terceros que contenga las optimizaciones necesarias para hacer más eficiente y competitivo en cuestiones de gestión y ahorro de recursos del sistema.

Hay varias opciones del Kernel GNU/Linux optimizado (Liquorix viene optimizado para multimedia y Juegos, por otro lado XanMod tiene uno para propósito general, otro aplicaciones críticas en tiempo real y otro más para

cálculos intensivos) de las últimas versiones estable del Kernel. Estos se pueden instalar[9] mediante el uso de los comandos *dpkg* o *apt* (después de agregarlo a nuestro repositorio */etc/apt/sources.list.d/*), de esta forma siempre podremos tener la última versión del Kernel junto con la actualización básica de nuestro sistema GNU/Linux.

Además, sí instalamos cualquiera de los distintos Kernels, siempre podemos seleccionar alguno de los instalados al momento de arrancar nuestro equipo para usarlo de acuerdo a las actividades requeridas en ese momento. Y en caso necesario, es fácil su desinstalación y continuar usando el Kernel que teníamos por defecto.

Por otro lado, existe una versión completamente libre del Kernel de GNU/Linux (Linux-Libre) el cual es un Kernel despojado de elementos de Firmware y controladores que contienen componentes no libres o fragmentos de código cuyo alcance está limitado por el fabricante.

Linux-libre es el núcleo recomendado por la Free Software Foundation y una pieza principal de las distribuciones GNU totalmente libres de fragmentos privativos o Firmwares incluidos en Linux sirven para inicializar los dispositivos o aplicarles parches que solventan fallas del Hardware que no pudieron ser corregidas antes de ser puestos a disposición de los usuarios.

Además, Linux-libre deshabilita las funciones del Kernel para cargar componentes no libres que no forman parte del suministro del Kernel y elimina la mención del uso de componentes no libres de la documentación. El Kernel de Linux-libre se utiliza en distribuciones como Dragora Linux, Trisquel, Dyne, Bolic, gNewSense, Parabola, Musix y Kongoni.

Estabilidad del Kernel La estabilidad de un núcleo no es tan difícil. El Kernel de Unix de Mac o el Kernel de Linux están diseñados de manera diferente pero resuelven el mismo problema. El sistema operativo Windows es igualmente robusto. Eso es evidente.

Pero la estabilidad real del día a día depende de otros factores. Particularmente en los controladores que conectan el sistema operativo al Hardware. Aquí es donde surgen las diferencias.

Apple tiene el momento más fácil, porque solo admiten un pequeño conjunto de Hardware seleccionado. Eso facilita su trabajo, el objetivo es pequeño. Apple ocasionalmente estropea esto, pero en su mayor parte, OS X

[9] Para ver las opciones de optimización del Kernel y como instalarlo ver la página Web de cada proyecto: Liquorix, XanMod, Linux-Libre.

ofrece una notable estabilidad diaria para las aplicaciones de escritorio.

Windows tiene un trabajo mucho más difícil. El sistema operativo admite una gama simplemente gigantesca de Hardware, y los fabricantes de Hardware hacen todo lo posible para proporcionar controladores de alta calidad. Este es el valor real de Windows, como paquete de controladores, es prácticamente imbatible.

Linux también intenta admitir una gran variedad de Hardware. Pero muchos fabricantes de Hardware son absolutamente indiferentes a la compatibilidad con Linux. Por lo tanto, el soporte de Hardware en Linux es mucho más impredecible. Si está ejecutando un servidor en Linux y el sistema solo se comunica con un disco duro y un adaptador de red, es probable que tenga una estabilidad impecable que supere a la industria.

Pero, si instala Linux en una computadora portátil y espera que funcione con la función de reposo / activación, la GPU, la tarjeta de sonido y un montón de extravagantes periféricos, entonces podría alejarse del rango de la estabilidad. Hay algunos controladores de código abierto, pero estos son significativamente peores que las versiones de los fabricantes. Por ejemplo, una GPU puede ser 4 o 5 veces más lenta.

Como usuario de escritorio de Linux, es probable que también instale aplicaciones que hacen cosas interesantes, de diversos proveedores, que pueden no estar del todo de acuerdo con las mejores prácticas.

Las Vulnerabilidades y Exposiciones Comunes El mundo está cada vez más interconectado y, como resultado de esto, la exposición a las vulnerabilidades de seguridad también ha aumentado dramáticamente. Las complejidades de mantener las plataformas de cómputo actuales hacen que sea muy difícil para los desarrolladores cubrir cada punto de entrada potencial. En 2019 hubo un promedio de más de 45 vulnerabilidades y exposiciones comunes registradas por día y estas siguen en aumento año con año.

Las vulnerabilidades y exposiciones comunes (Common Vulnerabilities and Exposures, CVE https://cve.mitre.org) que tienen los distintos sistemas operativos, es una lista de información registrada sobre vulnerabilidades de seguridad conocidas, en la que cada referencia tiene un número de identificación CVE-ID, descripción de la vulnerabilidad, que versiones del Software están afectadas, posible solución al fallo (si existe) o como configurar para mitigar la vulnerabilidad y referencias a publicaciones o entradas de foros o Blogs donde se ha hecho pública la vulnerabilidad o se demues-

tra su explotación. Además suele también mostrarse un enlace directo a la información de la base de datos de vulnerabilidades (https://nvd.nist.gov, https://openssf.org y https://docs.aws.amazon.com/security), en la que pueden conseguirse más detalles de la vulnerabilidad y su valoración.

El CVE-ID ofrece una nomenclatura estándar para identificación de la vulnerabilidad de forma inequívoca que es usada en la mayoría de repositorios de vulnerabilidades. Es definido y es mantenido por The MITRE Corporation (por eso a veces a la lista se la conoce por el nombre MITRE CVE List) con fondos de la National Cyber Security Division del gobierno de los Estados Unidos de América. Forma parte del llamado Security Content Automation Protocol.

Mitos en torno a Linux/Unix Hay varios mitos en torno a Linux/Unix y al Software libre, a saber:

- 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 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 operativo el Software, de modo que no deje de funcionar cuando alguien deja de mantenerlo. Excepto que mantener el Software en Linux/Unix se está convirtiendo en un desafío tan grande que daría igual que fuese privativo. Porque sería complicado hacerlo funcionar en un tiempo razonable, incluso siendo desarrollador, podría costar mucho trabajo y es posible dejar algo sin funcionar en el camino.

- La retrocompatibilidad[10] es un enorme dolor de cabeza, tomar Software hecho para Linux/Unix de hace 10 o 5 años y ejecutarlo en una distribución moderna. Cualquier cosa de mínima complejidad o que use una GUI, simplemente no funciona. Mientras la retrocompatibilidad en Windows es simplemente increíble. En Linux/Unix somos dependientes de los repositorios en línea, y cuando una aplicación depende de ciertas librerías que empiezan a desaparecer de esos repositorios, nos encontramos en una pesadilla. Y mientras más viejo el Software, peor.

2.1 Windows

Microsoft Windows (véase [17]), conocido generalmente como Windows o MS Windows es el nombre de una familia de Software propietario (véase apéndice 11.1) de distribuciones de Software para PC, Smartphone -que perdió cuota de mercado con Android hasta desaparecer-, servidores y sistemas empotrados, desarrollados y vendidos por Microsoft y disponibles para múltiples arquitecturas, tales como x86, x86-64 y ARM.

Desde un punto de vista técnico, no son sistemas operativos, sino que contienen uno (tradicionalmente MS-DOS, o el más actual, cuyo núcleo es Windows NT) junto con una amplia variedad de Software; no obstante, es usual (aunque no necesariamente correcto) denominar al conjunto como sistema operativo en lugar de distribución.

[10] 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 las actualizaciones de seguridad pertinentes.

La versión más reciente de Windows es Windows 11 para equipos personales (que se ofrece como actualización gratuita para los equipos con licencia válida de Windows 10 que cumplan con los requisitos mínimos de Hardware exigidos), Windows Server 2022 para servidores.

Windows 11 tiene al menos siete ediciones con diferente conjunto de características y Hardware previsto, algunas de ellas son: Home, Pro, Pro Education, Pro for Workstations, Enterprise, Education, Mixed Reality. Además habrá un modo SE para hacer frente al Chrome OS de los Chromebooks de Google.

Windows 11 también se puede descargar como una imagen ISO para instalaciones nuevas, ejecución en máquinas virtuales, además de ser la versión de referencia que los fabricantes OEM que usarán para preinstalaciones en equipos nuevos.

Por su parte, Windows 10 tiene al menos doce ediciones con diferente conjunto de características y Hardware previsto, algunas de ellas son: Home, Pro, Enterprise, Enterprise LTBS/LTSC, Education, Mobile, S, Pro for Workstation, Team, Pro Education, loT (Embedded), N y KN. Se espera que cuente con soporte y actualizaciones hasta el 2025.

Todas las ediciones mencionadas tienen la capacidad de utilizar los paquetes de idiomas, lo que permite múltiples idiomas de interfaz de usuario. A pesar de la múltiple cantidad de ediciones, solamente Windows Home y Pro están orientadas para el común de los usuarios y vienen instaladas en equipos nuevos. Las demás ediciones se adquieren mediante otros tipos de compra.

Por su parte, Windows 10 llegó de forma oficial y gratuita a usuarios con licencia genuina de Windows 7, Windows 8.1 y Windows 8 así como a Insiders, siendo la primera versión que buscaba la unificación de dispositivos (escritorio, portátiles, teléfonos inteligentes, tabletas y videoconsolas) bajo una experiencia común, con lo que se esperaba eliminar algunos problemas que se presentaron con Windows 8.1.

Además está Windows PE que es un pequeño sistema operativo usado para instalar, desplegar y reparar Windows 10 en todas sus versiones de escritorio, servidor y para otras ediciones de Windows. En concreto, podemos preparar el disco antes de la instalación, instalar Windows con apps desde un disco local o una red, instalar imágenes de Windows, hacer modificaciones en Windows sin iniciar sesión, recuperar datos perdidos y un largo etcétera.

Seguridad Una de las principales críticas que reciben los sistemas operativos Windows es la debilidad del sistema en lo que a seguridad se refiere y el alto índice de vulnerabilidades críticas. El propio Bill Gates, fundador de Microsoft, ha asegurado en repetidas ocasiones que la seguridad es objetivo primordial para su empresa.

Partiendo de que no existe un sistema completamente libre de errores, las críticas se centran en la lentitud con la que la empresa reacciona ante un problema de seguridad que pueden llegar a meses o incluso años de diferencia desde que se avisa de la vulnerabilidad hasta que se publica la actualización que corrija dicha vulnerabilidad (parche). En algunos casos la falta de respuesta por parte de Microsoft ha provocado que se desarrollen parches que arreglan problemas de seguridad hechos por terceros.

Uno de los pilares en que se basa la seguridad de los productos Windows es la seguridad por ocultación, en general, un aspecto característico del Software propietario que sin embargo parece ser uno de los responsables de la debilidad de este sistema operativo debido a que, la propia seguridad por ocultación, constituye una infracción del uno de los principios de Kerckhoffs, el cual afirma que la seguridad de un sistema reside en su diseño y no en una supuesta ignorancia del diseño por parte del atacante.

Windows 11 Microsoft presentó el 24 de junio del 2021 la siguiente generación de su sistema operativo y que se puede descargar como actualización a partir del 5 de octubre del 2021. Windows 11 con un nuevo diseño que potencia las formas redondeadas y las transparencias, y enfocado a la productividad. Este diseño, que incluye un modo oscuro y uno claro, está pensado para transmitir sensación de calma y sobre todo para que el usuario tenga la información siempre a mano. Esto se aprecia en nuevas funciones, como los Widgets, que pueden personalizar tarjetas de información sobre el tiempo, el tráfico, o información local.

La compañía también ha rediseñado el menú de inicio y la barra de tareas con el acceso a aplicaciones como Teams, que ahora está integrado en Windows. Con Snap Layouts, Windows 11 permite personalizar la apariencia y la disposición de las ventanas. El usuario también podrá personalizar escritorios según el uso, ya sea para el trabajo, para el estudio o el ocio, con configuraciones separadas.

La nueva experiencia con las aplicaciones y son el 'Docking' –cuando el portátil se conecta a un monitor– hacen posible retomar el trabajo, incluso

con varias aplicaciones abiertas, en el mismo lugar donde el usuario lo dejó antes de apartarse del ordenador. Windows 11 también está diseñado para que siempre se sienta igual tanto si se usa en una tableta y su pantalla táctil como si se usa en un ordenador con teclado. Además, mejora la experiencia con el lápiz óptico y teclado táctil en pantalla.

Otro elemento que presenta novedades es Microsoft Store, que ha sido rediseñada para facilitar la búsqueda de aplicaciones, e incorpora también las aplicaciones de Android, que pueden colocarse en la barra de tareas. Con Windows 11 se tendrá una nueva tienda de aplicaciones, una que abrirá sus puertas a todo tipo de aplicaciones y juegos. Esto quiere decir que a partir de ahora la nueva generación de Windows contará con PWA (aplicaciones Web progresivas), UWP (aplicaciones universales) y los programas clásicos Win32 en un mismo lugar.

En cuanto al tema de actualizaciones, Microsoft abandona definitivamente el programa de entrega de actualizaciones semestrales de Windows 10 que definitivamente no pudo concretar con la estabilidad requerida. Windows 11 solo recibirá una actualización anual de características, funciones y soporte de nuevas tecnologías, lo que debería otorgarle tiempo suficiente para desarrollo y pruebas, entregando versiones más pulidas. Las actualizaciones acumulativas de seguridad y corrección de errores si mantendrán el actual ciclo de entrega mensual.

En octubre del 2022 se publicó[11] que Windows 10 se mantiene con una cuota de mercado de 71.29%, mientras que Windows 11 tiene una cuota de 15.44%, esta diferencia deduce por los altos requisitos mínimos de Hardware exigidos -4 GB RAM, TPM 2.0 y 60 GB de Disco-. Por otra parte, las cuotas de mercado de las versiones sin soporte de Windows son: Windows 8.1 de 2.45%, Windows 8 de 0.69%, Windows 7 de 9.61% y Windows XP de 0.39%.

Windows 365 Microsoft presentó en julio del 2021 la nueva versión de Windows en la nube que llevará el nombre de Windows 365. Este nuevo servicio de suscripción, está especialmente dirigido a empresas, que permitirá acceder a nuestra sesión de usuario desde cualquier equipo (el PC, el Mac, la tableta o teléfono Android, etc.), pues el Software y el sistema de

[11] Cuotas de mercado de diferentes sistemas operativos:

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

archivos están alojados en una máquina virtual remota, por lo que la configuración, documentos y herramientas disponibles serán idénticos desde donde accedamos.

Así, desde cualquier navegador (o bien usando la aplicación de Escritorio Remoto de Windows) podremos acceder a un Windows 11/10 disfrutando de una experiencia de arranque casi instantáneo, pero esa no será la única ventaja de esta plataforma. Aún más importante será la posibilidad de contar con varios 'ordenadores' en una misma cuenta, cada uno con distinta potencia (RAM, núcleos de procesador...) y capacidad de almacenamiento contratada, según el trabajo que necesitemos llevar a cabo en cada momento.

Microsoft ya ha confirmado que ofrecerá 12 configuraciones de Hardware distintas para sus equipos virtualizados (iniciando en $130 pesos mexicanos por mes). Así, las empresas podrán 'crear PCs' en cuestión de minutos y asignar cada uno a un empleado, eliminando los inconvenientes que conlleva el hecho de manejar Hardware físico.

Windows 11 SE Microsoft anunció en noviembre del 2021 el lanzamiento de una nueva versión de Windows 11 que hará compañía a las versiones 'Pro' y 'Home': su nombre es Windows 11 SE, llevábamos oyendo rumores al respecto desde junio y desembarca ahora para hacer frente al Chrome OS de los Chromebooks de Google, por lo que se destinará únicamente en portátiles escolares de bajo costo.

El sistema estará optimizado para su uso con MS Edge, MS Office y el resto de servicios de la nube de Microsoft, estará abierto a muchas más aplicaciones de terceros. En palabras de Paige Johnson, directora de marketing educativo de Microsoft, "Windows 11 SE también es compatible con aplicaciones de terceros, incluidas Zoom y Chrome, porque queremos dar a las escuelas la opción de usar lo que funcione mejor para ellas".

Winget Windows Package Manager (*winget*) es el gestor de paquetes de Windows 11 y 10 que permite usar comandos desde la terminal para instalar aplicaciones de forma rápida y sencilla (al más puro estilo de Linux). El único requisito para instalar *winget* es contar con Windows 10 1890 o versión posterior, es posible usar para actualizar una o todas las aplicaciones del sistema usando:

C: winget upgrade –all

también es posible usar *WInstall* interfaz gráfica no oficial de *winget* para elegir programas en un click desde una Web y luego instalarlos en Windows con *winget* con la misma rapidez y automatización.

Microsoft Open Source En Agosto del 2020 presentó la empresa de Redmond el nuevo sitio Microsoft Open Source en que el público puede navegar a través de todo el ecosistema de código abierto que ha estado construyendo en los últimos años. La Web no solo muestra los proyectos Open Source de Microsoft sino que cuenta con secciones para colaborar con la comunidad, descargar herramientas, explorar su código, y hasta encontrar oportunidades de trabajo.

Las dos partes más importantes de este nuevo sitio son las secciones "Get involved" y "Explore proyects". En la primera se puede revisar toda la actividad reciente en los proyectos Open Source de Microsoft alojados en GitHub, y además se cuenta con una larga lista de recursos para aprender a colaborar con proyectos de código abierto, y no necesariamente solo los que mantiene Microsoft.

La segunda sección es la lista de proyectos, y ahí nos encontramos los principales proyectos Open Source mantenidos por los ingenieros de Microsoft y la comunidad. La lista de proyectos es larga y podemos encontrar los proyectos de los empleados de la empresa patrocinados a través de Microsoft FOSS Fund.

Linux Dentro de Windows Desde el 2018 se inició la integración de GNU/Linux en Windows 10, con la actualización de Windows 10 Fall Creator Update con WSL (Windows Subsystem for Linux), se permitía instalar consolas de diversas distribuciones de GNU/Linux como un programa más. Y en el 2020, con la llegada de Windows 10 Build 2020 con WSL2, el cual cuenta con su propio Kernel de Linux que permite instalar de manera casi nativa diversas distribuciones de GNU/Linux con todo el ambiente gráfico permitiendo tener lo mejor de ambos mundos en un mismo equipo -sin hacer uso de programas de virtualización-, incluso es posible ejecutar varias distribuciones de Linux al mismo tiempo en pantalla.

Para usarlo hay que tener todas las actualizaciones de Windows y activar el Subsistema de Windows para Linux (WSL[12]). Reiniciando el sistema, ya podemos usar distribuciones de Linux desde Microsoft Store.

[12] https://docs.microsoft.com/en-us/windows/wsl/install-win10

En el Windows Insider Preview Build 20150 ha incluido soporte para GPU de Intel, AMD y NVIDIA y es compatible con Direct ML (una API de bajo nivel para aprendizaje automático soportado por DirectX 12) permitiendo el uso de las capacidades de computación por GPU de WSL para Linux.

En abril del 2021 se anunció la llegada de WSLg en la que se pueden correr aplicaciones (como gedit, Audacity, etc.) e IDEs con soporte para X11 y Wayland, PulseAudio de Linux con GUI (Graphical User Interface) con soporte para gráficos 3D acelerados por Hardware de forma independiente de la distribución de Linux en la que se instalaron, esta novedad está disponible a partir de la versión Windows Insider Preview Build 21364 y para todos los usuarios en Windows 10 May 2020 Update.

Android Dentro de Windows En el Windows Build 20185 ha incluido soporte para que Windows 10 permite no sólo sincronizar teléfonos Android, sino además mediante "your Phone" permite integrar las aplicaciones, notificaciones, mensajes, fotos, llamadas y otras opciones de teléfonos inteligentes en Android directamente en Windows, ejecutando las aplicaciones sin tener que abrirlas en el teléfono, aunque siguen proviniendo de ahí.

Además en noviembre de 2020, se ha informado que existe la posibilidad de que Microsoft permita la instalación y ejecución de aplicaciones para Android en Windows 11. Con cambios mínimos o directamente sin modificaciones en el código, los desarrolladores podrían enviar a Microsoft Store sus aplicaciones para que sean descargadas e instaladas en PCs, el proyecto tiene el nombre de Latte.

Microsoft Azure Durante los últimos años, Microsoft ha declarado en conferencias magistrales de eventos y en otros lugares que Linux es un sistema operativo de rápido crecimiento que se usa dentro de Microsoft Azure.

Han declarado con orgullo que el 50 % de las máquinas virtuales nuevas que se ejecutaban en Azure ejecutaban Linux. En Octubre del 2022, las estadísticas reportadas señalan:

- Más del 50 % de los núcleos de máquinas virtuales ejecutan Linux en Azure
- Las imágenes basadas en Linux comprenden el 60 % de las imágenes de Azure Marketplace

- Los 100 principales clientes de Microsoft implementan cargas de trabajo de Linux en Azure
- Azure Tuned Kernels proporciona un rendimiento de red un 25 % más rápido
- Microsoft es compatible con todas las principales distribuciones de Linux, como: Red Hat, SUSE, Ubuntu, Oracle Linux, Debian, CentOS, CoreOS y OpenSUSE (Relacionado: Azure también es compatible con FreeBSD)
- Azure ofrece dos servicios de orquestación de Kubernetes administrados con soporte nativo: Azure Kubernetes Service y Azure Red Hat OpenShift

2.2 UNIX y BSD

Unix (véase [**?**]) es un sistema operativo portable, multitarea y multiusuario; desarrollado en 1969 por un grupo de empleados de los laboratorios Bell de AT&T. El sistema, junto con todos los derechos fueron vendidos por AT&T a Novell Inc. Esta vendió posteriormente el Software a Santa Cruz Operation en 1995, y está, a su vez, lo revendió a Caldera Software en 2001, empresa que después se convirtió en el grupo SCO. Sin embargo, Novell siempre argumentó que solo vendió los derechos de uso del Software, pero que retuvo el Copyright sobre "UNIX". En 2010, y tras una larga batalla legal, esta ha pasado nuevamente a ser propiedad de Novell.

Solo los sistemas totalmente compatibles y que se encuentran certificados por la especificación Single UNIX Specification pueden ser denominados "UNIX" (otros reciben la denominación «similar a un sistema Unix»). En ocasiones, suele usarse el término "Unix tradicional" para referirse a Unix o a un sistema operativo que cuenta con las características de UNIX Versión 7 o UNIX System V o UNIX versión 6.

Berkeley Software Distribution o BSD (en español, «distribución de Software Berkeley») (véase [**?**]) fue un sistema operativo derivado de Unix que nace a partir de los aportes realizados a ese sistema por la Universidad de California en Berkeley. En los primeros años del sistema Unix sus creadores, los Laboratorios Bell de la compañía AT&T, autorizaron a la Universidad de Berkeley en California y a otras universidades, a utilizar el código fuente y

adaptarlo a sus necesidades. Durante los años 1970 y 1980 Berkeley utilizó el sistema para sus investigaciones en materia de sistemas operativos.

Cuando AT&T retiró el permiso de uso a la universidad por motivos comerciales, la universidad promovió la creación de una versión inspirada en el sistema Unix utilizando los aportes que ellos habían realizado, permitiendo luego su distribución con fines académicos y al cabo de algún tiempo reduciendo al mínimo las restricciones referente a su copia, distribución o modificación (véase apéndice 11.4).

Algunos sistemas operativos descendientes del sistema desarrollado por Berkeley son SunOS, FreeBSD, NetBSD, OpenBSD, DragonFlyBSD y Mac Big Sur. BSD también ha hecho grandes contribuciones en el campo de los sistemas operativos en general. Además, la licencia permisiva de BSD ha permitido que otros sistemas operativos, tanto libres como propietarios incorporaron código BSD. Por ejemplo, Microsoft Windows ha utilizado código derivado de BSD en su implementación de TCP/IP, y utiliza versiones recompiladas de la línea de comandos BSD para las herramientas de redes. También Darwin, el sistema en el cual está construido Mac Big Sur, el sistema operativo de Apple, está derivado en parte de FreeBSD 5. Otros sistemas basados en Unix comerciales como Solaris también utilizan código BSD.

Algunos proyectos activos descendientes del sistema BSD son:

FreeBSD (https://www.freebsd.org/es/)

Es un sistema operativo para computadoras basadas en las CPU de arquitectura Intel. También funciona con procesadores compatibles como AMD. Está basado en la versión 4.4 BSD-Lite del CSRG (Computer Systems Research Group) y fue escrito en C y C++. Tiene Licencia BSD. Este proyecto ha realizado una gran inversión de tiempo en ajustar el sistema para ofrecer las mejores condiciones de rendimiento con carga real y facilidad de uso al usuario final.

NetBSD (https://www.netbsd.org)

Está basado en un conjunto de aplicaciones Open Source, incluyendo 4.4 BSD-Lite de la Universidad de California en Berkeley, Net/2 (Berkeley Networking Release 2), el sistema gráfico X del MIT y aplicaciones del proyecto GNU. Tiene Licencia BSD. NetBSD ha invertido sus energías en proveer de un sistema operativo estable, multiplataforma, seguro y orientado a la inves-

tigación. Está portado a 56 arquitecturas de Hardware y suele ser el primero en implementar tecnologías nuevas, como IPv6.

OpenBSD (https://www.openbsd.org)
Está basado en 4.4 BSD y es un descendiente de NetBSD. El proyecto tiene el foco puesto de forma particular en la seguridad y criptografía. Los esfuerzos se centran en la portabilidad, cumplimiento de normas, corrección, seguridad y criptografía integrada. Tiene Licencia BSD. La filosofía del proyecto puede ser descrita en tres palabras: "Free, Functional and Secure" (Libre, Funcional y Seguro).

DragonFlyBSD (https://www.dragonflybsd.org)
Tiene como meta ofrecer un alto rendimiento y escalabilidad bajo cualquier entorno, desde computadoras de un solo usuario hasta enormes sistemas de clústeres. DragonFlyBSD tiene varios objetivos técnicos a largo plazo, pero el desarrollo se centra en ofrecer una infraestructura habilitada para SMP que sea fácil de entender, mantener y desarrollar.

2.3 Apple y sus macOS e iOS

Apple a la empresa multinacional estadounidense Apple Inc., dedicada al diseño, la confección y la comercialización de productos electrónicos y de Software, así como de los servicios en línea (a través de Internet) que los atañen. Es considerada como una de las sociedades más apreciables del mundo. Su función principal es la producción de aparatos digitales populares, las marcas más populares de esta compañía son: Mac, iPods, iPads e iPhones. Y cuya gama de productos ha ganado un nicho particular en el área de los Gadgets tecnológicos mediante su estética común, intensa mercadotecnia y pretendida alternatividad respecto a otras empresas hegemónicas como Microsoft.

En la actualidad los principales productos de Apple gozan de una amplia popularidad a nivel mundial, particularmente en lo vinculado con reproductores de música, computadores personales, tabletas, teléfonos, relojes inteligentes, Wearables y toda una red de Software que va desde sistemas operativos, programas de gestión de multimedios (como iTunes) o suites de edición profesional. Se trata de una empresa líder en innovación tecnológica y computarizada.

El sistema de Apple siempre se ha caracterizado por contar con detalles no solo sofisticados, sino también fuera de lo común, tratando de marcar la historia de la tecnología bajo Software con distintas mejoras, asistentes virtuales y muchos elementos que dejan gran satisfacción y que debes conocer.

Realizando un seguimiento de los últimos productos lanzados y presentados por los chicos de la manzanita nos damos cuenta de que sin duda son los mejores creando expectación. En Apple saben que para vender hay que mostrar cosas nuevas e innovar, ya que vivimos en un mundo en el que la tecnología evoluciona constantemente y si no lo haces, te quedas atrás. Pero..., ¿siempre son cosas nuevas las que nos muestran?.

Una de las cosas en las que destaca Apple por encima de sus competidores es en el campo de la investigación y la innovación. Las inversiones que realiza la empresa más valiosa del mundo en investigación son enormes, y no en vano. Porque si algo sabe hacer bien Apple es ir por delante de la competencia. En esto todos estamos de acuerdo, los de Cupertino saben mejor que nadie que innovar significa no tener competencia en el mercado. Sin duda, esto saben aprovecharlo y a veces, de tal manera que ni nos damos cuenta.

Pero en Apple tienen una fea costumbre que analizando un poco las últimas Keynotes nos damos cuenta. Los de Cupertino suelen vender como algo innovador y único cosas que ya hacía antes pero pasaba por alto. ¿Qué significa esto?, que en Apple saben como vender las características de sus productos para que nos parezcan espectaculares.

Las computadoras de Apple tienen un sistema operativo único y otras características que sólo están disponibles en las Macs. El diseño del Hardware de Apple unifica la marca, con productos como el iMac "todo en uno", el iMac original con sus colores brillantes y la gama de computadoras portátiles. Además, las nuevas características del sistema operativo que integran el uso de una computadora Mac, iPad, el teléfono o el iPod con iTunes de Apple y la tienda de aplicaciones, proporcionan a los clientes una experiencia de Apple aerodinámica.

Características de Mac

- Carpetas inteligentes en Finder
- Grabe la actividad de la pantalla macOS en QuickTime
- Activa las esquinas calientes de tu pantalla

- Reproduce música y películas
- Ejecutar Windows con Boot Camp
- Automatiza las tareas repetitivas
- Crea escritorios virtuales
- Ping archivos de forma inalámbrica con AirDrop
- Firmar documentos en Vista previa
- Autocompletar palabras a medida que escribe

Características de iOS

- Notificaciones innovadoras
- Widgets de máxima utilidad
- Puedes borrar las aplicaciones de fábrica
- Siri abre apps de terceros
- Varios idiomas para el teclado
- Reconocimiento facial en sus fotos
- Te permite controlar tu hogar
- 3D touch con muchos usos
- Dispone de mensajería interna: iMessage

Mac OS y macOS Ventura Mac OS (véase [**?**]) -del inglés Macintosh Operating System, en español Sistema Operativo Macintosh- es el nombre del sistema operativo propietario (véase apéndice 11.1) creado por Apple para su línea de computadoras Macintosh, también aplicado retroactivamente a las versiones anteriores a System 7.6, y que apareció por primera vez en System 7.5.1. Es conocido por haber sido uno de los primeros sistemas dirigidos a un gran público al contar con una interfaz gráfica compuesta por la interacción del Mouse con ventanas, íconos y menús.

Debido a la existencia del sistema operativo en los primeros años de su línea Macintosh resultó a favor de que la máquina fuera más agradable al usuario, diferenciándolo de otros sistemas contemporáneos, como MS-DOS, que eran un desafío técnico. El equipo de desarrollo del Mac OS original incluía a Bill Atkinson, Jef Raskin y Andy Hertzfeld.

Este fue el comienzo del Mac OS clásico, desarrollado íntegramente por Apple, cuya primera versión vio la luz en 1985. Su desarrollo se extendería hasta la versión 9 del sistema, lanzada en 1999. A partir de la versión 10 (Mac OS X), el sistema cambió su arquitectura totalmente y comenzó a basarse en BSD Unix, sin embargo su interfaz gráfica mantiene muchos elementos de las versiones anteriores.

Hay una gran variedad de versiones sobre cómo fue desarrollado el Mac OS original y dónde se originaron las ideas subyacentes. Pese a esto, los documentos históricos prueban la existencia de una relación, en sus inicios, entre el proyecto Macintosh y el proyecto Alto de Xerox PARC. Las contribuciones iniciales del Sketchpad de Ivan Sutherland y el On-Line System de Doug Engelbart también fueron significativas.

Versiones Antes de la introducción de los últimos sistemas basados en el microprocesador PowerPC G3, partes significativas del sistema se almacenaban en la memoria física de sólo lectura de la placa base. El propósito inicial de esto fue evitar el uso de la capacidad de almacenamiento limitada de los disquetes de apoyo al sistema, dado que los primeros equipos Macintosh no tenían disco duro. Sólo el modelo Macintosh Classic de 1991, podía ser iniciado desde la memoria ROM.

Esta arquitectura también permitió una interfaz de sistema operativo totalmente gráfica en el nivel más bajo, sin la necesidad de una consola de sólo texto o el modo de comandos de línea. Los errores en tiempo de arranque, como la búsqueda de unidades de disco que no funcionaban, se comunicaban al usuario de manera gráfica, generalmente con un ícono o con mensajes con el tipo de letra Chicago y un "timbre de la muerte" o una serie de pitidos.

Esto contrastaba con los PCs de la época, que mostraban tales mensajes con un tipo de letra monoespaciada sobre un fondo negro, y que requerían el uso del teclado y no de un ratón, para el acceso. Para proporcionar tales detalles en un nivel bajo, Mac OS dependía del Software de la base del sistema grabado en la ROM de la placa base, lo que más tarde ayudó a garantizar que sólo los equipos de Apple o los clones bajo licencia (con el contenido de la

memoria ROM protegido por derechos de autor de Apple, pudieran ejecutar Mac OS).

Mac OS puede ser dividido en tres familias:

- La familia Mac OS Classic, basada en el código propio de Apple Computer.
- El Sistema Operativo Mac OS X, desarrollado a partir de la familia Mac OS Classic y NeXTSTEP, el cual estaba basado en UNIX.
- MacOS Ventura es el reemplazo de macOS Monterrey (que fue el reemplazo de Mac OS X), disponible a partir de junio del 2022, que usando los procesadores de ARM han mostrado un gran desempeño en comparación con equipos INTEL y AMD de gama alta.

Linux Dentro de IOS Es posible tener un Linux completo en IOS además de poder hacer uso de Secure Shell (SSH) a una computadora con Linux. Para la primera forma, se puede ejecutar un sistema virtualizado utilizando Alpine Linux con iSH, que es de código abierto, pero debe instalarse utilizando la aplicación TestFlight propiedad de Apple.

Alternativamente hay aplicaciones de emulador de terminal de código abierto que proporcionan herramientas de código abierto dentro de un entorno restringido. Esta es la opción más limitada -en realidad no nos permite ejecutar Linux, pero estaremos ejecutando herramientas de Linux- pero brindan algunas funciones de línea de comandos. Por ejemplo:

- Sandboxed Shell, con más de 80 comandos e incluye Python 2 y 3, Lua, C, Clang, etc.
- a-Shell, otorga acceso al sistema de archivos e incluye Lua, Python, Tex, Vim, JavaScript, C yC++, junto con Clang y Clang++; y permite instalar paquetes de Python con pip.
- Blink Shell, permite la conexión con servidores.
- iSH, es un Shell Linux que usa *usermode x86* emulación y traducciones de *syscall*.

2.4 GNU/Linux

GNU[13]/Linux (véase [**?**]) también conocido como Linux, es un sistema operativo libre (véase apéndice 11.2) tipo Unix; multiplataforma, multiusuario y multitarea. El sistema es la combinación de varios proyectos, entre los cuales destacan GNU (encabezado por Richard Stallman y la Free Software Foundation) y el núcleo Linux (encabezado por Linus Torvalds). Su desarrollo es uno de los ejemplos más prominentes de Software libre: todo su código fuente puede ser utilizado, modificado y redistribuido libremente por cualquiera, bajo los términos de la GPL (Licencia Pública General de GNU) y otra serie de licencias libres.

A pesar de que «Linux» se denomina en la jerga cotidiana al sistema operativo, este es en realidad solo el Kernel (núcleo) del sistema. La idea de hacer un sistema completo se remonta a mediados de la década de 1980 con el proyecto GNU, así como una gran cantidad de los componentes que se usan hoy en día -además del núcleo-, que van desde los compiladores de GNU hasta entornos de escritorio. Sin embargo, tras la aparición de Linux en la década de 1990 una parte significativa de los medios generales y especializados han utilizado el término «Linux» para referirse a todo. Esto ha sido motivo de polémicas. Cabe señalar que existen derivados de Linux que no tienen componentes GNU -por ejemplo Android-, así como distribuciones de GNU donde Linux está ausente -por ejemplo Debian GNU/Hurd-.

A GNU/Linux se le encuentra normalmente en forma de compendios conocidos como distribuciones o distros, a las cuales se les ha adicionado selecciones de aplicaciones y programas para descargar e instalar las mismas. El propósito de una distribución es ofrecer GNU/Linux como un producto final que el usuario pueda instalar, cumpliendo con las necesidades de un grupo de usuarios o bien del público en general. Algunas de ellas son: Ubuntu, CentOS, Debian, Linux Mint, Arch Linux, Fedora, Red Hat, Oracle, Zorin, MX Linux, Parrot, Manjaro, Elementary, etc.

Algunas de ellas son especialmente conocidas por su uso en servidores de internet y supercomputadoras -donde GNU/Linux tiene la cuota más importante del mercado. Según el informe de International Data Corporation (IDC), GNU/Linux es utilizado por los más poderosos 500 sistemas de super-

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

cómputo de alto desempeño del mundo[14]-, en cuanto a teléfonos inteligentes y tabletas tiene una cuota de 86% y con menor participación, el sistema GNU/Linux también se usa en el segmento de las computadoras de escritorio, portátiles, computadoras de bolsillo, sistemas embebidos, videoconsolas y otros dispositivos.

Creación El proyecto GNU, iniciado en 1983 por Richard Stallman, tiene el objetivo de crear un «sistema de Software compatible con Unix compuesto enteramente de Software libre». El trabajo comenzó en el año 1984. Más tarde, en 1985, Stallman fundó la Free Software Foundation para financiar el desarrollo de GNU, y escribió la Licencia Pública General de GNU en 1989. A principios de la década de 1990, muchos de los programas que se requieren en un sistema operativo -como bibliotecas, compiladores, editores de texto, el Shell Unix, y un sistema de ventanas- ya se encontraban en operación. Sin embargo otros elementos como los controladores de dispositivos y los servicios estaban incompletos.

Linus Torvalds ha declarado que si el núcleo de GNU hubiera estado disponible en el momento (1991), no se habría decidido a escribir su propio núcleo. Aunque no fue liberado hasta 1992 debido a complicaciones legales, el desarrollo de BSD -de los cuales NetBSD, OpenBSD y FreeBSD descienden- es anterior al de Linux. Torvalds también ha declarado que si BSD hubiera estado disponible en ese momento, probablemente no habría creado Linux.

En 1991 Torvalds asistía a la Universidad de Helsinki. Usuario de MINIX y de los programas provenientes de GNU, se mostraba interesado por los sistemas operativos. Comenzó a trabajar en su propio núcleo en ese año, frustrado por la concesión de licencias que utilizaba MINIX, que en ese momento se limitaba a uso educativo.

El núcleo Linux maduró hasta superar a los otros núcleos en desarrollo. Las aplicaciones GNU también reemplazaron todos los componentes de MINIX, porque era ventajoso utilizar el código libre del proyecto GNU con el nuevo sistema operativo. El código GNU con licencia bajo la GPL puede ser reutilizado en otros programas de computadora, siempre y cuando también se liberen bajo la misma licencia o una licencia compatible. Torvalds inició un cambio de su licencia original, que prohibía la redistribución comercial a la GPL. Los desarrolladores de ambas partes trabajaron para integrar com-

[14] Top500.org informó, en su lista de noviembre de 2017 -y así ha continuado hasta ahora-, que las 500 supercomputadoras más potentes del mundo utilizan Linux.

ponentes de GNU con el núcleo Linux, consiguiendo un sistema operativo completamente funcional.

Para darnos una idea del frenético crecimiento del Kernel de Linux, por ejemplo, en la versión 4.10 se añadieron 632,782 líneas de código nuevo y en el Kernel 4.12 se añadieron más 1.2 millones de líneas de código nuevas, teniendo un total de 24,170,860 líneas de código. El número de desarrolladores involucrados fue de 1821 colaboradores y 220 empleados hicieron un promedio de 231 cambios por día, casi 10 cambios por hora, diariamente se añadieron casi 20 mil líneas de código, y casi 800 líneas por hora en dicha versión.

Hay que precisar que, si bien el código alojado en el repositorio del Kernel es cuantioso, sólo una pequeña parte del mismo afectará a nuestras propias instalaciones de GNU/Linux, pues gran parte del código fuente es específico para cada una de las (múltiples) arquitecturas de Hardware compatibles con Linux.

De hecho, a principios de 2018, Greg Kroah-Hartman (responsable de mantenimiento del código), afirmó que "un portátil promedio usa alrededor de 2 millones de líneas de código del Kernel para funcionar correctamente", cuando en aquel momento, el Kernel completo ya contaba con 25 millones de líneas de código (que ya han aumentado a más de 28 millones en la versión 5.8).

GNU/Linux puede funcionar tanto en entorno gráfico como en modo consola. 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, que son un conjunto de programas conformado por ventanas, íconos 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, entre muchos otros.

¿Qué es lo que está llevando a la gente a probar distribuciones de GNU/Linux y a utilizarlas como sistema operativo principal en sus equipos de cómputo? A continuación, vamos a exponer una lista con las razones por las que deberías probar una distribución de GNU/Linux -ya que es una sabia elección- como sistema operativo principal en tu equipo de cómputo:

Software Libre y Código Abierto muchos usuarios de internet no conocen el significado principal del Software libre ni del código abierto. Software libre son esos programas que se automanifiestan, por parte de sus autores, que puede ser copiado, modificado y redistribuido con o sin cambios o mejoras. El concepto de código abierto, es el Software desarrollado y distribuido libremente. Tiene beneficios prácticos ya que si alguien tiene una idea o piensa que puede mejorar el código puede modificarlo sin problemas.

Seguridad no descubrimos el agua tibia diciendo que el sistema operativo de Microsoft es el más atacado por virus y Malware y además, se han descubierto varios virus para Mac OS, unos que llevan ocultos mucho tiempo. Pero con GNU/Linux eso no pasa, ya que es un sistema suficientemente seguro y que no tenemos muchos registros de ataques a esta plataforma.

Aunque hay compañías Linuxeras, como Oracle, Novell, Canonical, Red Hat o SUSE, donde el grueso de distribuciones y Software Linux está mantenido por usuarios y colectivos sin ánimo de lucro. A diferencia de Microsoft y Windows, detrás de Linux no es habitual encontrarnos con una empresa con intereses empresariales, de manera que es más fácil evitar problemas de tipo legal o violaciones de nuestra privacidad o seguridad por parte de quienes han programado esa aplicación o versión de GNU/Linux que usamos. Un ejemplo es la recopilación de datos de uso. A diferencia de los sistemas operativos comerciales, en GNU/Linux no es habitual toparse con este problema.

Es Gratis aunque Mac OS X también es gratuito, está pensado para funcionar solamente en equipos de cómputo Apple. En cuanto a Windows, a pesar de la tendencia, sigue siendo de pago, a pesar de las muchas ofertas que hizo para cambiar de Windows 7 a Windows 10.

Si adquieres una computadora nueva con Windows, el precio incluye la licencia de compra. Por otro lado, todo el mundo sabe que los sistemas operativos de GNU/Linux son totalmente gratuitos y puedes instalarlos en cualquier equipo de cómputo. Las distribuciones más populares puedes descargarlas desde sus páginas oficiales e instalarlas las veces que quieras y en el número de equipos de cómputo que necesites. Además, no tendremos que pagar por utilizar el Software, sin embargo, podremos donar lo que nos plazca al proyecto para que sigan mejorándolo.

Fácil de Utilizar muchos de nosotros hemos utilizado un sistema operativo basado en GNU/Linux y no lo sabíamos. Aeropuertos, estaciones de tren, sistemas de gestión empresarial y ahora en el espacio con SpaceX, etc. Muchos de estos sistemas están basados en GNU/Linux.

Una de las barreras que durante años ha evitado a muchos usar Linux es su complejidad. O al menos lo era cuando la mayoría de tareas debías hacerlas desde la línea de comandos.

En la actualidad, distribuciones GNU/Linux como Ubuntu, Mint, Manjaro, Debian u OpenSUSE ofrecen una interfaz similar a Windows y con todas las herramientas y aplicaciones necesarias para empezar a disfrutar desde el primer día.

Si necesitas un nuevo Software, la mayoría de distribuciones cuentan con su propia tienda de aplicaciones o herramienta de gestión de aplicaciones. Todo está pensado para que cualquiera pueda manejarse sin problemas.

Está claro que existen versiones de GNU/Linux complejas, pero están enfocadas a un público muy concreto. Las distribuciones domésticas cumplen con creces con los requisitos de usuarios amateurs o recién llegados.

Versatilidad configurar un sistema a nuestro gusto, en Windows o en Mac OS X, es algo realmente difícil, pero con los sistemas operativos basados en GNU/Linux se puede tener un sistema operativo totalmente único y totalmente personalizable.

La naturaleza de GNU/Linux y su filosofía de código abierto y libre hace posible que contemos con cientos de versiones diferentes. Esto implica que podamos elegir una versión de GNU/Linux, o distribución, en función de para qué la queremos. ¿Para educación? ¿Para niños? ¿Para uso doméstico? ¿Para gestión de redes? ¿Para temas de seguridad? ¿Para reciclar un PC antiguo? Incluso las hay para arreglar problemas de Windows.

Esta variedad significa que no sólo podemos emplear GNU/Linux en una computadora doméstica. Los ejemplos más claros son Raspberry Pi, Jetson Nano, Pine64 y Arduino[15], son soluciones baratas y diminutas para montar tu propia computadora personal, tu centro multimedia o cualquier artilugio electrónico que desees diseñar. Y para hacerlo funcionar, cuentas con varias distribuciones Linux enfocadas a dicho Hardware.

[15] Son ordenadores del tamaño de una tarjeta de crédito que se conectan a un televisor, un teclado y ratón. Es una placa que soporta varios componentes necesarios en un ordenador común y cuyo precio inicial es de 15 dólares.

Usar GNU/Linux significa que puedes cambiar cualquier elemento de tu sistema operativo. Me refiero a ir más allá de los programas y aplicaciones por defecto. GNU/Linux cuenta con diferentes escritorios y gestores de ventanas, de manera que podemos elegir el que queramos, algo que permiten muchas distribuciones GNU/Linux. Mientras que Windows cuenta con un escritorio por defecto, en GNU/Linux podemos elegir entre: GNOME, KDE, LXQt, LXDE, Xfce, Unity, MATE, Cinnamon, Pantheon, Deepin, Budgie, PIXEL, Enlightenment, Trinity, Moksha, Ukui, etc. En la variedad está el gusto.

Además, cualquier configuración o elemento del sistema operativo es susceptible de ser alterado[16]. La única limitación es que seamos capaces o tengamos los conocimientos adecuados. Pero siempre podemos encontrar en internet un tutorial donde nos explique cómo hacerlo.

Existen distribuciones de Linux de tamaño muy reducido, por ejemplo: BasicLinux ocupa 2.8 MB, requiere un procesador 386 y 3 MB de RAM y cuenta con el escritorio gráfico JWM, Nanolinux ocupa 14 MB, utiliza SLWM como escritorio y cuenta con navegador, procesador de texto, hoja de cálculo, cliente IRC, etc.

Actualizaciones del Sistema Operativo hablando de actualizaciones, sus aplicaciones se actualizan prácticamente al día, en cuanto el desarrollador lanza dicha actualización. Por lo que siempre podemos tener nuestros programas y aplicaciones actualizadas.

Además para los usuarios que así lo requieran existen versiones de soporte a largo plazo (Long-Term Support , LTS) normalmente se asocia con una aplicación o un sistema operativo para el que obtendremos seguridad, mantenimiento y (a veces) actualizaciones de funciones durante un período de tiempo más largo.

Las versiones LTS se consideran las versiones más estables que se someten a pruebas exhaustivas y en su mayoría incluyen años de mejoras en el camino.

[16] BlendOS este prometedor sistema operativo, introduce muchas novedades, empezando porque ahora soporta distintas distribuciones: Arch (el principal), AlmaLinux, Crystal Linux, Debian, Fedora, Kali Linux, Neurodebian Bookworm, Rocky Linux y Ubuntu.

Además de estar disponible en siete entornos gráficos, y que se puede cambiar entre ellos con un sencillo comando. Los entornos en los que está son GNOME, KDE (Plasma), Cinnamon, Xfce, LXQt, MATE y Deepin. El comando para ir cambiando entre los escritorios disponibles es: *sudo system track*. Esta distribución es inmutable, por lo que es difícil que subir de versión estropee algo. Básicamente son imágenes completas a las que se le pueden hacer pequeños retoques, como instalar nuevo software. Pero casi todo va por contenedores.

Es importante tener en cuenta que una versión de Software LTS no implica necesariamente actualizaciones de funciones a menos que haya una versión más reciente de LTS. Sin embargo, obtendrá las correcciones de errores y las correcciones de seguridad necesarias en las actualizaciones de una versión de Soporte a largo plazo.

Se recomienda una versión LTS para consumidores, negocios y empresas listos para la producción porque obtiene años de soporte de Software y sin cambios que rompan el sistema con las actualizaciones. Si observamos una versión que no es LTS para cualquier Software, generalmente es la versión más avanzada con nuevas funciones y un período corto de soporte (por ejemplo, 6-9 meses) en comparación con 3-5 años de soporte en un LTS.

Tiendas de Aplicaciones lo mejor de las distribuciones de GNU/Linux es que tienen una característica en común, sus tiendas de aplicaciones. Ya que vamos a poder instalar cualquier tipo de programa que necesitemos con un Click. Recordamos que esto es algo que Windows está intentando con su propia tienda de aplicaciones, pero no están teniendo muy buenos resultados.

Compatibilidad muchos han experimentado problemas a la hora de actualizar sus sistemas operativos con los programas que tenían instalados. Pero eso con GNU/Linux, no pasa, ya que todas sus actualizaciones tienen retrocompatibilidad a largo plazo dentro de su distribución.

Hoy en día la mayoría de aplicaciones y servicios Online cuentan con versión compatible para cualquier sistema operativo. Siendo más fácil crear una aplicación multiplataforma, por lo que GNU/Linux cuenta con un catálogo de Software que poco o nada tiene que envidiar a Windows o Mac OS X.

En el catálogo destacan las aplicaciones gratuitas y de código abierto, pero también surgen proyectos comerciales, y en la lista se incluyen los juegos, cada vez más presentes en GNU/Linux.

Seguramente hay algún Software no disponible en GNU/Linux, pero es más que probable que encontremos una alternativa o, en su defecto, que podamos ejecutarlo mediante Wine o empleando máquinas virtuales como KVM/QEMU o VirtualBox.

En cuanto al Hardware, la comunidad GNU/Linux ha avanzado mucho en la creación de controladores o Drivers para emplear cualquier dispositivo o componente en GNU/Linux. Podemos encontrarnos con alguna excepción, pero la mayoría de dispositivos cuentan con un controlador compatible por

defecto.

Está en Todas Partes GNU/Linux está presente en la infraestructura de grandes empresas como Amazon, Facebook, Netflix, NASA, SpaceX, el gran colisionador de hadrones o IBM y en el año 2021 llegó a Marte en el sistema operativo del helicóptero que acompaña al rover Perseverance, etc. A nivel de usuario, muchos dispositivos emplean este sistema operativo, bien en alguna de sus versiones o a través de Android, que salvando las distancias, todavía conserva gran parte de su origen Linuxero. Por otro lado, las quinientas principales supercomputadoras emplean Linux como sistema operativo, ya que permite trabajar en todo tipo de entornos y situaciones.

Las grandes empresas de internet hace años que vieron en GNU/Linux una gran oportunidad, y si bien a nivel usuario doméstico no está tan extendido, nunca había sido tan fácil dar el paso. Para hacernos una idea, sólo hay que ver la lista de empresas que apoyan a GNU/Linux a través de The Linux Foundation. Una de las más recientes, la propia Microsoft.

La Comunidad GNU/Linux finalmente, hay que hablar de la fabulosa comunidad de GNU/Linux. Podemos preguntar lo que queramos en sus foros, cambiar el código, enviar tus programas, sin problemas. ¿Trabas en la configuración? Te lo solucionan sin preocupación, ¿consejos sobre Software? Hay cientos de hilos con soluciones. Y nosotros ponemos nuestro granito de arena con este trabajo.

Programas de Windows y macOS en Linux A medida que va pasando el tiempo, las diferencias entre los sistemas operativos se van volviendo irrelevantes. Máquinas virtuales, contenedores y otras tecnologías permiten que podamos utilizar cada día más títulos de nuestros programas preferidos aunque no tenga versión para nuestro sistema operativo.

Wine, la herramienta que actúa como un intérprete entre el núcleo Linux y las aplicaciones Windows ya lleva mucho tiempo entre nosotros. Desde hace poco tiempo, también tenemos una herramienta para los programas de macOS.

Wine para Aplicaciones de Windows 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 gestionar 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 per-

mitir 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.

Darling para Aplicaciones de macOS 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.

Aprender a Usar Linux 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[17] para instalar, una de las más ampliamente usadas es Debian GNU/Linux[18] y sus derivadas como Ubuntu. La comunidad de GNU/Linux te apoya para obtener, instalar y que de una vez por todas puedas usar GNU/Linux en tu computadora.

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 cuál es la que más te conviene:

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

o probar alguna versión Live[19]:

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/

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

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

[19] 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.

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

Ahora, Windows 10 Build 2020 con WSL2[20] (Windows Subsystem for Linux), tiene su propio Kernel de Linux que permite instalar de manera casi nativa diversas distribuciones de GNU/Linux permitiendo tener lo mejor de ambos mundos en un mismo equipo.

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

Sistemas operativos

2.5 Android

Android (véase [16]) es un sistema operativo basado en el núcleo Linux (véase apéndice 11.2). Fue diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes, tabletas y también para relojes inteligentes, televisores y automóviles. Inicialmente fue desarrollado por Android Inc., empresa que Google respaldó económicamente y más tarde, en 2005, compró. Android fue presentado en 2007 junto a la fundación del Open Handset Alliance (un consorcio de compañías de Hardware, Software y telecomunicaciones) para avanzar en los estándares abiertos de los dispositivos móviles. El primer móvil con el sistema operativo Android fue el HTC Dream y se vendió en octubre de 2008. Android es el sistema operativo móvil (SmartPhone y tabletas) más utilizado del mundo, con una cuota de mercado del 86% al año 2020, muy por encima del 13.9% de iOS.

El éxito del sistema operativo lo ha convertido en objeto de litigios sobre patentes en el marco de las llamadas guerras de patentes entre las empresas de teléfonos inteligentes. Según los documentos secretos filtrados en 2013 y 2014,

[20] https://docs.microsoft.com/en-us/windows/wsl/install-win10

el sistema operativo es uno de los objetivos de las agencias de inteligencia internacionales.

La versión básica de Android es conocida como Android Open Source Project (AOSP). El 25 de junio de 2014 en la Conferencia de Desarrolladores Google I/O, Google mostró una evolución de la marca Android, con el fin de unificar tanto el Hardware como el Software y ampliar mercados. El 17 de mayo de 2017, se presentó Android Go. Una versión más ligera del sistema operativo para ayudar a que la mitad del mundo sin Smartphone consiga uno en menos de cinco años. Incluye versiones especiales de sus aplicaciones donde el consumo de datos se reduce al máximo.

Arquitectura del Sistema Android los componentes principales del sistema operativo de Android[21]:

Aplicaciones: las aplicaciones base incluyen un cliente de correo electrónico, programa de SMS, calendario, mapas, navegador, contactos y otros. Todas las aplicaciones están escritas en lenguaje de programación Java.

Marco de trabajo de aplicaciones: los desarrolladores tienen acceso completo a las mismas API del entorno de trabajo usadas por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del Framework). Este mismo mecanismo permite que los componentes sean reemplazados por el usuario.

Bibliotecas: Android incluye un conjunto de bibliotecas de C/C++ usadas por varios componentes del sistema. Estas características se exponen a los desarrolladores a través del marco de trabajo de aplicaciones de Android. Algunas son: System C library (implementación biblioteca C estándar), bibliotecas de medios, bibliotecas de gráficos, 3D y SQLite, entre otras.

[21] Android tiene la base de Linux, por ello en cualquier dispositivo que soporte dicho sistema operativo es posible instalar una aplicación para acceder a la terminal de línea de comandos -por ejemplo ConnectBot-, y en ella podemos correr los comandos de BASH como en un sistema GNU/Linux.

Runtime de Android: Android incluye un conjunto de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje Java. Cada aplicación Android ejecuta su propio proceso, con su propia instancia de la máquina virtual Dalvik. Dalvik ha sido escrito de forma que un dispositivo puede ejecutar múltiples máquinas virtuales de forma eficiente. Dalvik ejecutaba hasta la versión 5.0 archivos en el formato de ejecutable Dalvik (.dex), el cual está optimizado para memoria mínima. La Máquina Virtual está basada en registros y corre clases compiladas por el compilador de Java que han sido transformadas al formato .dex por la herramienta incluida DX. Desde la versión 5.0 utiliza el ART, que se compila totalmente al momento de instalación de la aplicación.

Personalización muchos conocen a Android como el sistema operativo móvil más personalizable. Pero para los que no lo saben, recordamos que está basado en el núcleo de Linux y que muchos desarrolladores están queriendo llevar Android a un sistema operativo de escritorio.

Núcleo Linux: Android depende de Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores. El núcleo también actúa como una capa de abstracción entre el Hardware y el resto del Software.

Android sobre KVM (MicroDroid) En enero de 2021 se anunció por parte de Google que trabajan en MicroDroid, una versión minimalista de Android para máquinas virtuales sobre KVM. No es la primera alternativa a Android, nos encontramos con las capas de personalización Android Go, AOSP o las imágenes GSI. En el caso de MicroDroid, estaríamos ante una limitada imagen de Linux basada en Android, para este proyecto, Google está trabajando en adaptar la máquina virtual de Chrome OS (crosvm), que ya se utiliza para ejecutar aplicaciones de Linux en Chrome OS. De esta forma con MicroDroid podría ejecutar pequeñas máquinas virtuales junto a Android, posiblemente para aplicaciones y uso relacionado con DRM, esta modificación constaría con el mínimo de componentes para iniciar el sistema permitiendo aislar datos entre aplicaciones y sistemas operativos en el mismo dispositivo, así como cambiar instantáneamente entre sistemas operativos.

2.6 Chromebook y Chrome OS

Para entender la razón de ser de los Chromebooks, primero tenemos que entender qué es Chrome OS. Se trata de un sistema operativo creado por Google y diferente a Android. Está basado en el Kernel de Linux, y utiliza Chrome como su interfaz de usuario principal. Esto quiere decir que su aspecto es prácticamente idéntico al de Chrome, pero con algunos añadidos como una barra de tareas, un explorador de archivos y otros elementos presentes en cualquier sistema operativo.

Fue anunciado a mediados del 2009 como un intento de crear un sistema basado en la nube y en aplicaciones Web. Esto hacía que, cuando se estaba conectado a internet se pudieran hacer muchas cosas gracias a herramientas como Google Drive o las aplicaciones de la Chrome Web Store, pero que cuando dejaba de tener internet se limitaba mucho sus funciones.

En cualquier caso, y pese a lo limitado que era en sus primeros años, poco a poco Google lo ha hecho evolucionar. Primero se empezaron a añadir opciones a las aplicaciones de Google para poderse utilizar sin conexión, algo que también benefició a los usuarios que usaran Chrome en otros sistemas operativos.

Pero la evolución más grande fue llegando después. El primer gran paso fue el anuncio de la compatibilidad para ejecutar aplicaciones de Android, y se fue implementando directamente la tienda de aplicaciones Google Play de Android para hacer que la experiencia de instalarlas fuera tan nativa como en Android. Aun así, hay que decir que la llegada de Android a Chrome OS ha sido lenta, y han tardado algunos años en hacer que todo vaya funcionando como debería.

Y a mediados de 2018 se anunció que Google Chrome también podrá utilizar aplicaciones creadas para los sistemas GNU/Linux. Con ello, el catálogo de aplicaciones diseñadas para funcionar sin conexión se multiplica beneficiando a la comunidad de desarrolladores libres, aunque también es de esperar que tarde algunos años en estar todo perfectamente integrado, ya que todavía se están lanzando poco a poco mejoras.

Chrome OS es hoy en día un sistema operativo completo. Tiene lo básico, aplicaciones nativas y compatibilidad con Android, que se une al reproductor de medios, gestor de archivos, configuración de impresoras, etcétera. Además, al igual que el navegador, Chrome OS tiene también una versión libre llamada Chromium OS, que pese a no tener la tecnología nativa de Google sirve para que la comunidad de desarrolladores independientes pueda ayudar a

mejorarlo.

Ahora bien, los Chromebook son equipos de cómputo personales que utilizan como sistema operativo Chrome OS, desarrollado por Google y que, a diferencia de Windows, OS X y Linux, están pensados para utilizarse permanentemente conectados a internet, ya que se basan casi completamente en la nube.

Chromebook Apps también se incluye un reproductor multimedia, y todo se sincroniza permanentemente en la nube. Por ello, si pretendemos utilizar un Chromebook sin conexión a internet, su funcionalidad es más limitada que la de otros equipos de cómputo. De hecho, las aplicaciones se instalan a través de Chrome Web Store, la tienda de aplicaciones integrada en Google Chrome, con lo que algunas de las herramientas más habituales (como Office o Skype, por ejemplo) tendrían que verse reemplazadas por Google Drive y Google Hangouts, aplicaciones nativas de Google.

Chrome Web Store no obstante, también se pueden utilizar de forma local sin recurrir a la red, ya que muchos de los servicios de Google disponen de un modo sin conexión que, una vez volvemos a disponer de internet, se sincronizaran sin problemas.

¿Cómo es un Chromebook? en un Chromebook podemos utilizar dispositivos USB sin problemas, como memorias y discos externos, Webcams, teclados y ratones, y por lo general suelen venir con una cantidad de almacenamiento inferior a lo que estamos acostumbrados (ya que lo que se pretende es que todo esté en la nube, y no en nuestro disco duro local). De hecho, al adquirir uno se nos obsequia con 100 GBytes de espacio en Google Drive.

Igualmente, su precio suele ser bastante asequible (desde 179 dólares o 130 euros) y no requieren de un Hardware potente para funcionar, siendo la ligereza de recursos una de sus mayores bondades. Por su parte, los equipos de cómputo portátiles con Chrome OS son lo que llamamos Chromebook, mientras que si preferimos el formato Mini PC, estaremos ante un Chromebox.

El inicio del sistema es prácticamente instantáneo y todo está listo para funcionar en cuestión de segundos, y dadas sus características, un Chromebook es un equipo ideal para navegar por internet ante todo.

Se accede desde la barra de herramientas en la parte inferior de la pantalla a las aplicaciones que tengamos instaladas, que en realidad se trata de un atajo a las apps que tengamos instaladas en Google Chrome.

Chromebook Integración por supuesto, los Chromebook también son multiusuario, con la ventaja de que con simplemente iniciar sesión con otra cuenta de Gmail todo estará tal y como si lo hubiésemos configurado con ella (aplicaciones, servicios, historial y demás), y por este mismo motivo se complementan a la perfección con otros dispositivos (ya sean equipos de cómputo, Smartphones o Tablets) en los que utilicemos los servicios de Google, gracias a la sincronización en la nube.

Además, los Chromebook también presumen de no necesitar antivirus, pues al almacenarse todo en la nube la seguridad está integrada por defecto y corre por parte de Google.

Microsoft en un Chromebook En el 2020 las empresas Parallels[22] y Google llegaron a un acuerdo para ofrecer a los usuarios la posibilidad de ejecutar aplicaciones Windows en Chrome OS. Ellas aseguran que en Chrome OS la integración será completa: las aplicaciones se ejecutarán cada una en su propia ventana, como las nativas, y no dentro de un Windows virtualizado.

Aunque ninguna de las dos compañías ha ofrecido aún una lista de aplicaciones compatibles con esta función que será lanzada en el 2021, John Solomon (vicepresidente de Chrome OS) ha afirmado que Microsoft Office será una de ellas.

El problema es que, por ahora, estas nuevas funcionalidades no estarán disponibles para todos los usuarios de Chrome OS, sino únicamente para los de Chrome OS Enterprise, la versión empresarial del mismo.

Nota: en últimas fechas han aparecido proyectos que permiten instalar diversas distribuciones de GNU/Linux en los Chromebook, esto es debido a que Google deja de dar soporte a sus equipos después de algunos años de que salieron al mercado, pese a que el equipo es totalmente funcional.

Chrome OS Flex En febrero del 2022, Google anunció su nueva versión de Chrome OS para equipos de cómputo PCs y Macs, la propuesta es Chrome

[22] Empresa (propiedad de Corel desde hace un año) desarrolladora del Software homónimo de virtualización que es especialmente popular entre los usuarios de Mac.

OS Flex y cuya descarga es totalmente gratuita, tiene como propósito atender las necesidades de escuelas y empresas rehusando equipo (procesador Intel o AMD 64 bits, 4 GB RAM, 16 GB de almacenamiento, etc). Esta versión tiene la misma interfaz gráfica y herramientas básicas que se encuentran en Chrome OS; entre ellas el navegador Chrome, se ofrece soporte para sincronización de ajustes y marcadores. Además, si nuestro equipo cumple con las especificaciones básicas, tendremos a nuestra disposición a Google Assistant e integraciones diversas con dispositivos Android.

Es de notar que Chrome OS Flex no tiene soporte para la Play Store o para las aplicaciones de Android y al parecer no hay intención de añadir esta compatibilidad. Tampoco se puede ejecutar Windows en una máquina virtual de Parallels Desktop.

Se puede descargar Chrome OS Flex para crear una unidad USB booteable y la instalación reemplazará al sistema operativo del equipo donde se instale (Mac, Windows o Linux). Pero no está optimizado para sacar provecho de todos los puertos, sensores o accesorios que pueden estar presentes en el equipo. Esto nos proporcionará una experiencia lo más cercana posible a Chrome OS sin comprar un Chromebook pese a sus limitaciones.

2.7 Otros Sistemas Operativos

Sistemas Operativos para PC

1. Fuchsia OS.- Es un sistema operativo versátil y adaptable esta basado en el microkernel Zircon en desarrollo por parte de Google, está disponible desde un repositorio de Git y está ya siendo usado en los Nest Hub, se espera su uso en la domótica que prepara Google como parte del internet de las cosas.

2. Dahlia OS.- Este sistema operativo combina lo mejor de GNU/Linux y Fuchsia OS es moderno, seguro, liviano y receptivo. Se mantiene minimalista al incluir solamente las aplicaciones necesarias, pero es posible agregar todos nuestros favoritos de otros sistemas operativos usando aplicaciones Containers y proporciona una tienda para aplicaciones Flutter de terceros.

3. KataOS.- Es un sistema operativo centrado en la seguridad y los sistemas embebidos que está construido casi enteramente con Rust. No emplea Linux ni Fuchsia, sino el micronúcleo seL4, el cual, según Google,

"pone la seguridad al frente y en el centro". Lo que se pretende con este sistema operativo es proporcionar "una plataforma segura verificable que protege la privacidad del usuario porque es lógicamente imposible que las aplicaciones violen las protecciones de seguridad del Hardware incluidas en el kernel, además de que los componentes del sistema son seguros de forma verificable".

4. ToaruOS.- Es un sistema operativo escrito desde cero y provisto con su propio Kernel, cargador de arranque, biblioteca C estándar, administrador de paquetes, componentes de espacio de usuario y una interfaz gráfica con un administrador de ventanas compuesto. Se inició como un proyecto de investigación en la Universidad de Illinois en el 2010 y a partir del 2012 es desarrollado por la comunidad interesada.

5. Essence, es un sistema operativo con su propio Kernel y escritorio construido desde cero por un entusiasta desde 2017 y se destaca por su original escritorio y pila de gráficos que permite dividir ventanas en pestañas, lo que permite trabajar en una ventana con varios programas a la vez y agrupar aplicaciones en ventanas según las tareas a resolver. El administrador de ventanas funciona al nivel del Kernel del sistema operativo y la interfaz se crea utilizando su propia biblioteca gráfica y un motor de Software vectorial que admite efectos animados complejos completamente vectoriales.

6. eComStation.- Seguro que muchos recuerdan el mítico OS/2 de IBM, sistema operativo que perdura con eComStation, derivado de este adaptado al Hardware moderno. A diferencia de otras alternativas de la lista, este no es gratuito y sus precios comienzan desde 145 dólares para la versión doméstica. Muchas aplicaciones libres como Firefox, OpenOffice o VLC han sido portadas a este sistema operativo.

7. Haiku.- BeOS fue un sistema operativo lanzado en el año 1991 con muy buenas intenciones a nivel de optimización e interfaz. Sin embargo, como les sucedió a muchos otros, terminó sucumbiendo en este complicado mercado. Su legado ha sido continuado por Haiku, un sistema de código abierto que lleva ya años en desarrollo.

8. ReactOS.- Es una alternativa a la arquitectura Windows NT de Microsoft totalmente abierta que no utiliza ningún tipo de código propietario. No obstante, es compatible con muchos de los controladores y

aplicaciones de Windows. Como punto negativo, su desarrollo no es tan rápido como muchos esperarían en un entorno tan cambiante como este.

9. FreeDOS.- Alternativa libre a DOS cuyo desarrollo sigue activo en estos momentos. Se trata de un entorno bastante estable, pero que carece de interfaz gráfica o multitarea. Es compatible a todos los niveles con MS-DOS y sus programas.

10. Solaris.- El sucesor de SunOS, de Sun Microsystems, empezó como una distribución propietaria de UNIX, pero en 2005 fue liberado como OpenSolaris. Más tarde, Oracle compró Sun y le cambió el nombre a Oracle Solaris.

11. Illumos.- Basado en Open Solaris, este proyecto nació por parte de algunos de los ingenieros originales del sistema. En realidad, busca ser una base para crear distribuciones de este sistema operativo. OpenIndiana es una de las más conocidas y utilizadas.

12. DexOS.- Un sistema operativo de 32 Bits escrito para la arquitectura x86 en lenguaje ensamblador. Está diseñado para programadores que desean tener acceso directo al Hardware (incluyendo CPU y gráficos) con un código bien comentado y documentado.

13. Syllable.- Sistema operativo nacido como fork de AtheOS, un clon de AmigaOS, aunque comparte mucho código con Linux. No tiene demasiada utilidad para los usuarios domésticos, aunque es compatible con arquitecturas x86.

14. AROS Research Operating System.- Es otro sistema que implementa en código abierto las APIs de AmigaOS, con cuyos ejecutables es compatible a nivel binario en procesadores de 68k, además de ser compatible a nivel de código con otras arquitecturas como x86 para la que se ofrece de manera nativa. Es portable y puede correr hospedado en Windows, Linux y FreeBSD.

15. MenuetOS.- Llamado también como MeOS, su característica más destacada es que está programado completamente en lenguaje ensamblador. Está diseñado para funcionar en equipos muy básicos aunque soporta

hasta 32 GigaBytes de RAM. Con decir que el sistema cabe en un disquete de 1.44 Megabytes, está dicho todo. Aún así se las arregla para incluir un escritorio gráfico y controladores para teclados, video, audio, USB o impresoras.

16. Visopsys.- Se trara de un sistema gratuito y libre bajo GPL que ha estado en desarrollo desde 1997, como hobby de un solo programador, Andy McLaughlin. Soporta arquitecturas x86, está escrito en C y ensamblador y no se basa en ningún sistema preexistente, si bien utiliza código del kernel Linux, ofrece herramientas comunes de GNU y parte de la interfaz gráfica de usuario como los iconos, resultaran familiares a los usuarios de KDE Plasma.

17. mOS.- Sistema operativo usado en centros de datos y para cómputo de alto rendimiento (High Performance Computing HPC), se basa en el Kernel de Linux pero tiene su propio núcleo ligero LWK, el Kernel gestiona un pequeño número de núcleos de la CPU para asegurarse la compatibilidad y el LWK Kernel gestiona el resto del sistema.

18. KolibriOS.- Es un pequeño sistema operativo poderoso y rápido para PCs. Solamente requiere unos pocos megas de espacio en disco y 8 MB de RAM para funcionar, además de incluir varias aplicaciones básicas.

19. SerenityOS es un sistema operativo Unix con aspecto de Windows de los 90s creado por un único programador como un proyecto terapéutico y está pensado para equipos X86 de escritorio.

20. BlendOS este prometedor sistema operativo Linux, introduce muchas novedades, empezando porque ahora soporta distintas distribuciones: Arch (el principal), AlmaLinux, Crystal Linux, Debian, Fedora, Kali Linux, Neurodebian Bookworm, Rocky Linux y Ubuntu. Además de estar disponible en siete entornos gráficos, y que se puede cambiar entre ellos con un sencillo comando. Los entornos en los que está son GNOME, KDE (Plasma), Cinnamon, Xfce, LXQt, MATE y Deepin. Esta distribución es inmutable, por lo que es difícil que subir de versión estropee algo. Básicamente son imágenes completas a las que se le pueden hacer pequeños retoques, como instalar nuevo Software, pero casi todo va por contenedores.

Sistemas Operativos para móviles

1. PinePhone.- Usa un sistema operativo basado en sistemas operativos de código abierto impulsado por la comunidad Linux, ha sido portado a 16 diferentes distribuciones de Linux y 7 diferentes interfaces gráficas de usuario como: Mobian, Manjaro con interfaz plasma, Ubuntu Touch, postmarketOS, LuneOS, Nemo Mobile, Maemo Leste, Tizen, entre otros. Además la compañía Pine64 es el segundo fabricante de teléfonos (después de OpenMoko) que ofrece el arranque desde una tarjeta microSD, que permite a los usuarios probar múltiples sistemas operativos, antes de instalarse en la memoria Flash interna.

2. HarmonyOS.- Sistema operativo desarrollado por Huawei para reemplazar a Android en sus equipos, es un sistema operativo similar a la idea de Fuchsia OS, con la idea que pueda instalarse tanto en un ordenador, como en un teléfono, tableta, relojes, como en un coche conectado, en donde todos estos dispositivos se conecten entre sí con una sola cuenta, dando así un paso hacia adelante en la utopía de la convergencia.

3. PostmarketOS.- Sistema operativo de Software libre y código abierto en desarrollo principalmente para teléfonos inteligentes y tabletas -es una idea genial, la persecución de tener Linux en los dispositivos Smartphone, como otra alternativa a los sistemas Android e iOS-, haciéndose las primeras pruebas en teléfonos que ya no tienen uso. Distribución basada en Alpine Linux. Puede usar diferentes interfaces de usuario, por ejemplo Plasma Mobile, Hildon, LuneOS UI, MATE, GNOME 3 y XFCE.

4. Plasma Mobile.- Es un sistema en fase de desarrollo por KDE que permite la convergencia con los usuarios de KDE para escritorio.

5. Lomiri.- Sistema operativo basado en Linux que soporta dos sabores: Ubuntu Touch y Manjaro. Ambos basados en Unity 8 que están en constante desarrollo.

6. Windows Phone.- Sistema operativo móvil desarrollado por Microsoft, como sucesor de Windows Mobile. A diferencia de su predecesor fue enfocado en el mercado de consumo en lugar del mercado empresarial.

7. Symbian OS.- Era un sistema operativo que fue producto de la alianza de varias empresas de telefonía móvil, entre las que se encuentran Nokia, Sony Ericsson y otros, el objetivo de Symbian fue crear un sistema operativo para terminales móviles.

8. BlackBerry OS.- Es un sistema operativo móvil desarrollado por Research In Motion para sus dispositivos BlackBerry.- Es multitarea y tiene soporte para diferentes métodos de entrada adoptados por RIM para su uso en computadoras de mano, particularmente la trackwheel, trackball, touchpad y pantallas táctiles.

9. HP webOS.- Se trata de un sistema operativo multitarea para sistemas embebidos basado en Linux, desarrollado por Palm Inc., ahora es propiedad de Hewlett-Packard Company.

10. GrapheneOS.- Es un sistema operativo móvil centrado en la privacidad y la seguridad con compatibilidad con aplicaciones Android, que está desarrollado como un proyecto de código abierto sin ánimo de lucro. Se centra en la investigación y el desarrollo de tecnología de privacidad y seguridad, incluyendo mejoras sustanciales en el Sandboxing, la mitigación de exploits y el modelo de permisos.

11. Sailfish OS.- Es un Sistema Operativo móvil seguro y optimizado para funcionar en Smartphones y tabletas, y también fácilmente adaptable a todo tipo de dispositivos integrados y casos de uso. Es el único Sistema Operativo móvil independiente basado en el código abierto, sin ningún vínculo con las grandes corporaciones, respaldado por unos sólidos derechos de propiedad intelectual, que incluyen todos los derechos de propiedad intelectual y marcas comerciales. En resumen, es una plataforma abierta con un modelo de contribución de código abierto activo.

12. Bada.- Fue un sistema operativo para teléfonos móviles desarrollado por Samsung (Bada «océano» o «mar» en coreano). Diseñado para cubrir teléfonos inteligentes de gama alta como gama baja.

3 Entornos de Desarrollo y Herramientas de Programación

En los últimos años los lenguajes de programación han ido evolucionado en el desarrollo de sistemas o Software, con el objetivo principal de facilitar al usuario las actividades que realiza día con día; por tal motivo, como programador, es importante conocer los conceptos básicos de programación, los tipos de lenguajes que se utilizan para el desarrollo y su funcionamiento para la interpretación de algoritmos, así como para dar solución a los problemas que pudieran presentarse.

En términos generales, un lenguaje de programación es una herramienta que permite desarrollar Software o programas para computadora. Los lenguajes de programación son empleados para diseñar e implementar programas encargados de definir y administrar el comportamiento de los dispositivos físicos y lógicos de una computadora. Lo anterior se logra mediante la creación e implementación de algoritmos de precisión que se utilizan como una forma de comunicación humana con la computadora.

Los 5 lenguajes de programación más populares en la actualidad son Java, C, Python, C++, y #C según el índice de TIOBE[23] actualizado en enero de 2021, por otro lado en la lista de la IEEE a septiembre del 2021 los 10 lenguajes de programación más populares son Python, Java, C, C++, y JavaScript, C#, R, Go, HTML, Swift.

A grandes rasgos, un lenguaje de programación se conforma de una serie de símbolos y reglas de sintaxis y semántica que definen la estructura principal del lenguaje y le dan un significado a sus elementos y expresiones.

Programación es el proceso de análisis, diseño, implementación, prueba y depuración de un algoritmo, a partir de un lenguaje que compila y genera un código fuente ejecutado en la computadora.

La función principal de los lenguajes de programación es escribir programas que permiten la comunicación usuario-máquina. Unos programas especiales (compiladores o intérpretes) convierten las instrucciones escritas en código fuente, en instrucciones escritas en lenguaje máquina.

[23] Pero si vemos los cinco lenguajes más populares a septiembre de 2020 según el índice TIOBE. La lista consta de lenguajes tan conocidos como C, Java, Python, C++ y C#. Salvo Python, los otros cuatro ya estaban en el TOP 5 allá por 2015 y por 2010. Es más, si comparamos el índice de 2020 y 2019, el único cambio es entre C y Java, que pasan de ser segundo y primero a primero y segundo. El resto permanece igual.

Los intérpretes leen la instrucción línea por línea y obtienen el código máquina correspondiente.

En cuanto a los compiladores, traducen los símbolos de un lenguaje de programación a su equivalencia escrita en lenguaje máquina (proceso conocido como compilación). Por último, se obtiene un programa ejecutable.

Para programar, es necesario como mínimo contar con un editor de texto -como *vi*, *nano o micro*- y acceso al compilador o intérprete del lenguaje que nos interese. En Linux se tiene una gran variedad de lenguajes y herramientas de desarrollo -Linux fue hecho por programadores para programadores- que se pueden instalar. Pero, también están los entornos de desarrollo integrado o entorno de desarrollo interactivo -en inglés Integrated Development Environment (IDE)-, estas son aplicaciones informáticas que proporcionan servicios integrales para facilitarle al programador el desarrollo de Software.

Normalmente, un IDE consiste de un editor de código fuente, herramientas de construcción automáticas y un depurador. La mayoría de los IDE tienen auto-completado inteligente de código (IntelliSense). Algunos IDE contienen un compilador, un intérprete, o ambos, tales como NetBeans y Eclipse. El límite entre un IDE y otras partes del entorno de desarrollo de Software más amplio no está bien definido. Muchas veces, a los efectos de simplificar la construcción de la interfaz gráfica de usuario (GUI, por sus siglas en inglés) se integran un sistema controlador de versión y varias herramientas. Muchos IDE modernos también cuentan con un navegador de clases, un buscador de objetos y un diagrama de jerarquía de clases, para su uso con el desarrollo de Software orientado a objetos.

Los IDE están diseñados para maximizar la productividad del programador proporcionando componentes muy unidos con interfaces de usuario similares. Los IDE presentan un único programa en el que se lleva a cabo todo el desarrollo. Generalmente, este programa suele ofrecer muchas características para la creación, modificación, compilación, implementación y depuración de Software. Esto contrasta con el desarrollo de Software utilizando herramientas no relacionadas, como *vi*, GNU Compiler Collection (*gcc*) o *make*.

Uno de los propósitos de los IDE es reducir la configuración necesaria para reconstruir múltiples utilidades de desarrollo, en vez de proveer el mismo conjunto de servicios como una unidad cohesiva. Reduciendo ese tiempo de ajustes, se puede incrementar la productividad de desarrollo, en casos donde aprender a usar un IDE es más rápido que integrar manualmente todas las herramientas por separado.

Una mejor integración de todos los procesos de desarrollo hace posible mejorar la productividad en general, que únicamente ayudando con los ajustes de configuración. Por ejemplo, el código puede ser continuamente armado, mientras es editado, previendo retroalimentación instantánea, como cuando hay errores de sintaxis. Esto puede ayudar a aprender un nuevo lenguaje de programación de una manera más rápida, así como sus librerías asociadas.

Algunos IDE están dedicados específicamente a un lenguaje de programación, permitiendo que las características sean lo más cercanas al paradigma de programación de dicho lenguaje. Por otro lado, existen muchos IDE de múltiples lenguajes tales como *Eclipse, ActiveState Komodo, IntelliJ IDEA, MyEclipse, Oracle JDeveloper, NetBeans, Codenvy y Microsoft Visual Studio.* Por otro lado *Xcode, Xojo y Delphi* están dedicados a un lenguaje cerrado o a un tipo de lenguajes de programación.

Los IDE ofrecen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como *C++, Python, Java, C#, Delphi, Visual Basic,* etc. En algunos lenguajes, un IDE puede funcionar como un sistema en tiempo de ejecución, en donde se permite utilizar el lenguaje de programación en forma interactiva, sin necesidad de trabajo orientado a archivos de texto. Es deseable que un IDE cuente con las siguientes características:

- Multiplataforma
- Soporte para diversos lenguajes de programación
- Integración con Sistemas de Control de Versiones
- Reconocimiento de Sintaxis
- Extensiones y Componentes para el IDE
- Integración con Framework populares
- Depurador
- Importar y Exportar proyectos
- Múltiples idiomas
- Manual de Usuarios y Ayuda

- Componentes
- Editor de texto
- Compilador.
- Intérprete
- Herramientas de automatización
- Depurador
- Posibilidad de ofrecer un sistema de control de versiones
- Factibilidad para ayudar en la construcción de interfaces gráficas de usuarios

Algunos de los más usados son: *Eclipse, Aptana, NetBeans, Sublime Text, Geany, Visual Studio, Brackets, Monodevelop, Komodo, Anjuta, CodeLite, Code::Blocks, PyDev, Eric, PyCharm, PTK, Spyder, Bluefish, Glade, Kdevelop, Emacs, QtCreator, Android SDK, WxFormBuilder,* etc.

3.1 Java

Java (véase [**?**]) es un lenguaje de programación de propósito general, concurrente, orientado a objetos, que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones «escriban el programa una vez y lo ejecuten en cualquier dispositivo (Write Once, Run Anywhere» o WORA)», lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para ejecutarse en otra.

El lenguaje de programación Java fue originalmente desarrollado por James Gosling, de Sun Microsystems (constituida en 1982 y posteriormente adquirida el 27 de enero de 2010 por la compañía Oracle), y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son compiladas a bytecode (clase Java), que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente.

La compañía Sun desarrolló la implementación de referencia original para los compiladores de Java, máquinas virtuales y librerías de clases en 1991, y las publicó por primera vez en 1995. A partir de mayo de 2007, en cumplimiento de las especificaciones del Proceso de la Comunidad Java, Sun volvió a licenciar la mayoría de sus tecnologías de Java bajo la Licencia Pública General de GNU (véase [7]). Otros también han desarrollado implementaciones alternas a estas tecnologías de Sun, tales como el Compilador de Java de GNU y el GNU Classpath.

Orientado a Objetos La primera característica, orientado a objetos (OO), se refiere a un método de programación y al diseño del lenguaje. Aunque hay muchas interpretaciones para OO, una primera idea es diseñar el Software de forma que los distintos tipos de datos que usen, estén unidos a sus operaciones. Así, los datos y el código (funciones o métodos) se combinan en entidades llamadas objetos. Un objeto puede verse como un paquete que contiene el "comportamiento" (el código) y el "estado" (datos). El principio es separar aquello que cambia de las cosas que permanecen inalterables. Frecuentemente, cambiar una estructura de datos implica un cambio en el código que opera sobre los mismos, o viceversa. Esta separación en objetos coherentes e independientes ofrece una base más estable para el diseño de un sistema Software. El objetivo es hacer que grandes proyectos sean fáciles de gestionar y manejar, mejorando como consecuencia su calidad y reduciendo el número de proyectos fallidos.

Otra de las grandes promesas de la programación orientada a objetos es la creación de entidades más genéricas (objetos) que permitan la reutilización del Software entre proyectos, una de las premisas fundamentales de la Ingeniería del Software.

La reutilización del Software ha experimentado resultados dispares, encontrando dos dificultades principales: el diseño de objetos realmente genéricos es pobremente comprendido y falta una metodología para la amplia comunicación de oportunidades de reutilización. Algunas comunidades de código abierto quieren ayudar en este problema dando medios a los desarrolladores para diseminar la información sobre el uso y versatilidad de objetos reutilizables y bibliotecas de objetos.

Independencia de la Plataforma La segunda característica, la independencia de la plataforma, significa que programas escritos en el lenguaje Java

pueden ejecutarse igualmente en cualquier tipo de Hardware. Este es el significado de ser capaz de escribir un programa una vez y que pueda ejecutarse en cualquier dispositivo, tal como reza el axioma de Java, "write once, run anywhere".

Para ello, se compila el código fuente escrito en lenguaje Java, para generar un código conocido como "bytecode" (específicamente Java bytecode), instrucciones máquina simplificadas específicas de la plataforma Java. Esta pieza está "a medio camino" entre el código fuente y el código máquina que entiende el dispositivo destino. El bytecode es ejecutado entonces en la máquina virtual (JVM), un programa escrito en código nativo de la plataforma destino (que es el que entiende su Hardware), que interpreta y ejecuta el código. Además, se suministran bibliotecas adicionales para acceder a las características de cada dispositivo (como los gráficos, ejecución mediante hebras o threads, la interfaz de red) de forma unificada. Se debe tener presente que, aunque hay una etapa explícita de compilación, el bytecode generado es interpretado o convertido a instrucciones máquina del código nativo por «compilación al vuelo JIT (Just In Time)».

Hay implementaciones del compilador de Java que convierten el código fuente directamente en código objeto nativo, como GCJ. Esto elimina la etapa intermedia donde se genera el bytecode, pero la salida de este tipo de compiladores sólo puede ejecutarse en un tipo de arquitectura.

La licencia sobre Java de Sun insiste en que todas las implementaciones sean "compatibles". Esto dio lugar a una disputa legal entre Microsoft y Sun, cuando este último alegó que la implementación de Microsoft no daba soporte a las interfaces RMI y JNI además de haber añadido características "dependientes" de su plataforma. Sun demandó a Microsoft y ganó por daños y perjuicios (unos 20 millones de dólares), así como una orden judicial forzando el acatamiento de la licencia de Sun. Como respuesta, Microsoft no ofrece Java con su versión de sistema operativo, y en recientes versiones de Windows, su navegador Internet Explorer no admite la ejecución de applets sin un «conector (Plugin)» aparte. Sin embargo, Sun y otras fuentes ofrecen versiones gratuitas para distintas versiones de Windows.

Las primeras implementaciones del lenguaje usaban una máquina virtual interpretada para conseguir la portabilidad. Sin embargo, el resultado eran programas que se ejecutaban comparativamente más lentos que aquellos escritos en C o C++. Esto hizo que Java se ganase una reputación de lento en rendimiento. Las implementaciones recientes de la JVM dan lugar a programas que se ejecutan considerablemente más rápido que las versiones antiguas,

empleando diversas técnicas, aunque sigue siendo mucho más lentos que otros lenguajes.

La primera de estas técnicas es simplemente compilar directamente en código nativo como hacen los compiladores tradicionales, eliminando la etapa del bytecode. Esto da lugar a un gran rendimiento en la ejecución, pero tapa el camino a la portabilidad. Otra técnica, conocida como «compilación al vuelo JIT (Just In Time)», convierte el bytecode a código nativo cuando se ejecuta la aplicación. Otras máquinas virtuales más sofisticadas usan una "recompilación dinámica" en la que la VM es capaz de analizar el comportamiento del programa en ejecución y recompila y optimiza las partes críticas. La recompilación dinámica puede lograr mayor grado de optimización que la compilación tradicional (o estática), ya que puede basar su trabajo en el conocimiento que de primera mano tiene sobre el entorno de ejecución y el conjunto de clases cargadas en memoria. La compilación JIT y la recompilación dinámica permiten a los programas Java aprovechar la velocidad de ejecución del código nativo sin por ello perder la ventaja de la portabilidad en ambos.

La portabilidad es técnicamente difícil de lograr, y el éxito de Java en ese campo ha sido dispar. Aunque es de hecho posible escribir programas para la plataforma Java que actúen de forma correcta en múltiples plataformas de distinta arquitectura, el gran número de estas con pequeños errores o inconsistencias llevan a que a veces se parodie el eslogan de Sun, "Write once, run anywhere" cómo "Write once, debug everywhere" (o "Escríbelo una vez, ejecútalo en cualquier parte" por "Escríbelo una vez, depúralo en todas partes").

El concepto de independencia de la plataforma de Java cuenta, sin embargo, con un gran éxito en las aplicaciones en el entorno del servidor, como los Servicios Web, los Servlets, los Java Beans, así como en sistemas empotrados basados en OSGi, usando entornos Java empotrados.

El Recolector de Basura En Java el problema de las fugas de memoria se evita en gran medida gracias a la «recolección automática de basura (o automatic garbage collector)». El programador determina cuándo se crean los objetos y el entorno en «tiempo de ejecución de Java (Java runtime)» es el responsable de gestionar el ciclo de vida de los objetos. El programa, u otros objetos, pueden tener localizado un objeto mediante una referencia a este. Cuando no quedan referencias a un objeto, el recolector de basura de Java

borra el objeto, liberando así la memoria que ocupaba previniendo posibles fugas (ejemplo: un objeto creado y únicamente usado dentro de un método sólo tiene entidad dentro de este; al salir del método el objeto es eliminado). Aun así, es posible que se produzcan fugas de memoria si el código almacena referencias a objetos que ya no son necesarios; es decir, pueden aún ocurrir, pero en un nivel conceptual superior. En definitiva, el recolector de basura de Java permite una fácil creación y eliminación de objetos y mayor seguridad.

Instalación de Java e IDEs Existen diversas versiones de Java para Linux, la más usada es JDK de Oracle pero también está una versión abierta llamada OpenJDK, para instalar por ejemplo OpenJDK 17 en Debian GNU/Linux es necesario hacer:

```
# apt install default-jdk
```

o

```
# apt install openjdk-17-jre openjdk-17-jdk openjdk-17-doc
```

si se desea instalar solo el Run-Time JRE, para ello usamos:

```
# apt install default-jre
```

o

```
# apt install openjdk-17-jre
```

y si hay más de una versión instalada, podemos actualizar la versión por omisión de Java:

```
# update-java-alternatives -s java-1.17.0-openjdk-amd64
```

para conocer la versión instalada usamos:

```
$ java –version
```

Además, se pueden instalar diversas herramientas e IDEs para facilitar la programación en Java, para ello usar:

```
# apt install scite jedit kate gedit nedit emacs medit \
kscope geany geany-plugins editra qtcreator anjuta \
anjuta-extras codelite codelite-plugins tea vim-gtk \
mousepad eric neovim neovim-qt medit kwrite katepart
# apt install eclipse eclipse-cdt eclipse-pydev netbeans \
bluefish bluefish-plugins codeblocks codeblocks-contrib
# apt install fte fte-console fte-terminal nano joe vim \
vim-python-jedi vim-tlib vim-latexsuite vim-nox micro \
neovim kakoune vim-athena jed
# apt install kdiff3 meld diffuse dirdiff kompare numdiff \
colordiff dwdiff wdiff xxdiff tkdiff ndiff ccdiff xxdiff
# apt install alleyoop astyle c2html java2html code2html \
c2html autodia txt2html html2text
# apt install git git-all gitk gitg git-cola git-gui qgit tig \
vim-fugitive git-extras
# apt install mercurial
# apt install subversion rapidsvn
# apt install cvs tkcvs
```

Además, es posible instalar varios editores especializados de las páginas oficiales de sus proyectos:

https://netbeans.apache.org/download/index.html
https://www.eclipse.org/downloads/
http://brackets.io/
https://www.jetbrains.com/idea/download/#section=Linux
https://www.oracle.com/tools/downloads/Jdeveloper-12c-downloads.html
http://www.drjava.org/
https://www.jgrasp.org/
https://www.bluej.org/
http://www.jcreator.com/index.htm
https://codenvy.com/
https://atom.io/
https://www.sublimetext.com/

Compilar y Ejecutar Para compilar el archivo ejemplo.java usamos *javac* en línea de comandos mediante:

```
$ javac ejemplo.java
```

y lo ejecutamos con:

```
$ java ejemplo
```

Monitoreo de Aplicaciones Java Java Development Kit (JDK) provee binarios, herramientas y compiladores para el desarrollo de aplicaciones en Java. Además incluye una poderosa herramienta de monitoreo que es *jconsole*. Podemos ejecutarla en una terminal de Linux usando:

```
$ jconsole
```

al lanzar la aplicación, nos preguntará si deseamos hacer el monitoreo local o de algún equipo conectado en red. Si seleccionamos local, entonces podemos ver el consumo de las aplicaciones que corren en la máquina virtual de Java en tiempo real de CPU, memoria, Threads, Clases, etc.

Además podemos usar el comando *jps* (Java Virtual Machine Process Status) que nos permite conocer cada proceso que corre en la máquina virtual de Java. El uso básico es:

```
$ jps
```

pero podemos pedirle más información usando:

```
$ jps -v
```

esta última nos proporciona el indicador de proceso y el nombre de la clase o archivo *Jar* que se detecte en cada instancia.

Crear y Ejecutar Archivos .jar Un archivo *.jar* (Java ARchive) es un formato de archivo independiente de la plataforma que se utiliza para agregar muchos archivos de clase Java, metadatos y recursos asociados, como texto, imágenes, etc., en un solo archivo para su distribución.

Permite que los tiempos de ejecución de Java implementen de manera eficiente una aplicación completa en un archivo de almacenamiento y brinda muchos beneficios, como seguridad, sus elementos pueden comprimirse, acortar los tiempos de descarga, permite el sellado y control de versiones de paquetes, admite la portabilidad. También es compatible con el empaquetado para extensiones.

Para crear y ejecutar archivos .jar necesitamos hacer lo siguiente:

1. Primero comencemos escribiendo una clase Java simple con un método principal para una aplicación llamada MiApp, con fines de demostración.

   ```
   $ nano MiApp.java
   ```

   Copie y pegue el siguiente código en el archivo MiApp.java.

   ```
   public class MiApp {
       public static void main(String[] args){
           System.out.println("Solo ejecuta MiApp");
       }
   }
   ```

   Grabe el archivo y cierre este.

2 A continuación, necesitamos compilar y empaquetar la clase en un archivo JAR usando las utilidades javac y jar como se muestra:

   ```
   $ javac -d . MiApp.java
   $ ls
   $ jar cvf MiApp.jar MiApp.class
   $ ls
   ```

3 Una vez creado MiApp.jar, ahora podemos ejecutar el archivo usando el comando java como se muestra:

   ```
   $ java -jar MiApp.jar
   no main manifest attribute, in MiApp.jar
   ```

De la salida del comando anterior, encontramos un error. La JVM (Java Virtual Machine) no pudo encontrar nuestro atributo de manifiesto principal, por lo que no pudo ubicar la clase principal que contiene el método principal (public static void main (String [] args)).

El archivo JAR debe tener un manifiesto que contenga una línea con el formato Main-Class: classname que defina la clase con el método principal que sirve como punto de partida de nuestra aplicación.

4 Para corregir el error anterior, necesitaremos actualizar el archivo JAR para incluir un atributo de manifiesto junto con nuestro código. Creemos un archivo MANIFEST.MF:

```
$ nano MANIFEST.MF
```

Copie y pegue la siguiente línea en el archivo MANIFEST.MF:

```
Main-Class: MiApp
```

Guarde el archivo y agreguemos el archivo MANIFEST.MF a nuestro MiApp.jar usando el siguiente comando:

```
$ jar cvmf MANIFEST.MF MiApp.jar MiApp.class
```

5 Finalmente, cuando ejecutamos el archivo JAR nuevamente, debería producir el resultado esperado como se muestra en la salida:

```
$ java -jar MiApp.jar
Solo ejecuta MiApp
```

Para obtener más información, debemos consultar las páginas de manual de los comandos java, javac y jar.

```
$ man java
$ man javac
$ man jar
```

3.2 Python

Python (véase [**?**]) es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License, que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores (véase apéndice 11.2).

Python fue creado a finales de los ochenta por Guido van Rossum en el Centro para las Matemáticas y la Informática (CWI, Centrum Wiskunde & Informatica), en los Países Bajos, como un sucesor del lenguaje de programación ABC, capaz de manejar excepciones e interactuar con el sistema operativo Amoeba. Van Rossum es el principal autor de Python, y su continuo rol central en decidir la dirección de Python es reconocido, refiriéndose a él como Benevolente Dictador Vitalicio (en inglés: Benevolent Dictator for Life, BDFL).

Características y paradigmas Python es un lenguaje de programación multiparadigma. Esto significa que más que forzar a los programadores a adoptar un estilo particular de programación, permite varios estilos: programación orientada a objetos, programación imperativa y programación funcional. Otros paradigmas están soportados mediante el uso de extensiones.

Python usa tipado dinámico y conteo de referencias para la administración de memoria.

Una característica importante de Python es la resolución dinámica de nombres; es decir, lo que enlaza un método y un nombre de variable durante la ejecución del programa (también llamado enlace dinámico de métodos).

Otro objetivo del diseño del lenguaje es la facilidad de extensión. Se pueden escribir nuevos módulos fácilmente en *C* o *C++*. Python puede incluirse en aplicaciones que necesitan una interfaz programable.

Aunque la programación en Python podría considerarse en algunas situaciones hostil a la programación funcional tradicional del Lisp, existen bastantes analogías entre Python y los lenguajes minimalistas de la familia Lisp como puede ser Scheme.

El intérprete de Python estándar incluye un modo interactivo en el cual

se escriben las instrucciones en una especie de intérprete de comandos: las expresiones pueden ser introducidas una a una, pudiendo verse el resultado de su evaluación inmediatamente, lo que da la posibilidad de probar porciones de código en el modo interactivo antes de integrarlo como parte de un programa. Esto resulta útil tanto para las personas que se están familiarizando con el lenguaje como para los programadores más avanzados.

Existen otros programas, tales como IDLE, bpython o IPython, que añaden funcionalidades extra al modo interactivo, como el autocompletado de código y el coloreado de la sintaxis del lenguaje.

Elementos del lenguaje Python fue diseñado para ser leído con facilidad. Una de sus características es el uso de palabras donde otros lenguajes utilizarían símbolos. Por ejemplo, los operadores lógicos: !, || y &&, en Python se escriben; not, or y and, respectivamente. Curiosamente el lenguaje Pascal es junto con COBOL uno de los lenguajes con muy clara sintaxis y ambos son de la década de los 70. La idea del código claro y legible no es algo nuevo.

El contenido de los bloques de código (bucles, funciones, clases, etc.) es delimitado mediante espacios o tabuladores, conocidos como indentación, antes de cada línea de órdenes pertenecientes al bloque. Python se diferencia así de otros lenguajes de programación que mantienen como costumbre declarar los bloques mediante un conjunto de caracteres, normalmente entre llaves {}. Se pueden utilizar tanto espacios como tabuladores para indentar el código, pero se recomienda no mezclarlos.

Debido al significado sintáctico de la indentación, cada instrucción debe estar contenida en una sola línea. No obstante, si por legibilidad se quiere dividir la instrucción en varias líneas, añadiendo una barra invertida: \ al final de una línea, se indica que la instrucción continúa en la siguiente.

Variables Las variables se definen de forma dinámica, lo que significa que no se tiene que especificar cuál es su tipo de antemano y puede tomar distintos valores en otro momento, incluso de un tipo diferente al que tenía previamente. Se usa el símbolo = para asignar valores.

Módulos Existen muchas propiedades que se pueden agregar al lenguaje importando módulos, que son "minicódigos" (la mayoría escritos también en Python) que proveen de ciertas funciones y clases para realizar determinadas tareas. Un ejemplo es el módulo: Tkinter, que permite crear interfaces grá-

ficas basadas en la biblioteca Tk. Otro ejemplo es el módulo: os, que provee acceso a muchas funciones del sistema operativo. Los módulos se agregan a los códigos escribiendo la palabra reservada *import* seguida del nombre del módulo que queramos usar.

Python tiene una gran biblioteca estándar, usada para una diversidad de tareas. Esto viene de la filosofía "pilas incluídas" ("batteries included") en referencia a los módulos de Python[24]. Los módulos de la biblioteca estándar pueden mejorarse por módulos personalizados escritos tanto en *C* como en Python. Debido a la gran variedad de herramientas incluidas en la biblioteca estándar, combinada con la habilidad de usar lenguajes de bajo nivel como *C* y *C++*, los cuales son capaces de interactuar con otras bibliotecas, Python es un lenguaje que combina su clara sintaxis con el inmenso poder de lenguajes menos elegantes.

Algunos Módulos para Python

TensorFlow Models sirve para el aprendizaje automático y aprendizaje profundo. TensorFlow Models es el repositorio de fuente abierta para encontrar muchas bibliotecas y modelos relacionados con el aprendizaje profundo.

Keras es una API de redes neuronales de alto nivel, escrita en Python y es capaz de ejecutarse sobre TensorFlow, CNTK o Theano. Fue desarrollado con un enfoque para permitir la experimentación rápida.

Frasco es un framework ligero de aplicaciones Web WSGI. está diseñado para que el inicio sea rápido y fácil, con la capacidad de escalar hasta aplicaciones complejas. Comenzó como un simple envoltorio alrededor de Werkzeug y Jinja y se ha convertido en uno de los frameworks de aplicación Web Python más populares.

Scikit-learn es un módulo de Python para el aprendizaje automático construido sobre SciPy y distribuido bajo la licencia BSD.

[24] Una lista de módulos disponibles en Python está en su página oficial.
Para la versión 2 en: https://docs.python.org/2/py-modindex.html
Para la versión 3 en: https://docs.python.org/3/py-modindex.html

Zulip es una poderosa aplicación de chat grupal de código abierto que combina la inmediatez del chat en tiempo real con los beneficios de productividad de las conversaciones enhebradas. Zulip es utilizado por proyectos de código abierto, compañías de Fortune 500, cuerpos de grandes estándares y otros que necesitan un sistema de chat en tiempo real que les permita a los usuarios procesar fácilmente cientos o miles de mensajes al día. Con más de 300 colaboradores que fusionan más de 500 commits por mes, Zulip es también el proyecto de chat grupal de código abierto más grande y de más rápido crecimiento.

Django es un framework Web Python de alto nivel que fomenta un desarrollo rápido y un diseño limpio y pragmático de desarrollo Web de código abierto, escrito en Python, que respeta el patrón de diseño conocido como modelo-vista-template. Fue desarrollado en origen para gestionar varias páginas orientadas a noticias de la World Company de Lawrence, Kanzas y fue liberada al público bajo una licencia BSD en julio del 2005.

Rebound es una herramienta de línea de comandos que obtiene instantáneamente los resultados de desbordamiento de pila cuando se produce un error de compilación.

Google Images Download Este es un programa de línea de comando de Python para buscar palabras clave / frases clave en Google Imágenes y opcionalmente descargar imágenes a su computadora. También puede invocar este script desde otro archivo Python.

YouTube-dl es usado para descargar videos de: youtube.com u otras plataformas de video.

System Design Primer este repositorio es una colección organizada de recursos para ayudar a aprender a construir sistemas a escala.

Mask R-CNN es para detección y segmentación de objetos. Esta es una implementación de Mask R-CNN en Python 3, Keras y TensorFlow. El modelo genera cuadros de delimitación y máscaras de segmentación para cada instancia de un objeto en la imagen. Se basa en Feature Pyramid Network (FPN) y ResNet101 backbone.

Face Recognition es usado para reconocer y manipular caras desde Python o desde la línea de comandos con la biblioteca de reconocimiento facial más simple del mundo. Esto también proporciona una herramienta de línea de comandos: face_recognition simple que permite hacer reconocimiento de rostros en una carpeta de imágenes desde la línea de comandos.

Snallygaster Herramienta para buscar archivos secretos en servidores HTTP.

Ansible es un sistema de automatización de TI radicalmente simple. Maneja la administración de configuraciones, la implementación de aplicaciones, el aprovisionamiento en la nube, la ejecución de tareas ad-hoc y la orquestación multinodo, incluida la trivialización de cosas como actualizaciones continuas de tiempo de inactividad cero con balanceadores de carga.

Detectron es el sistema de software de Facebook AI Research que implementa algoritmos de detección de objetos de última generación, incluyendo Mask R-CNN, está escrito en Python y funciona con el marco de aprendizaje profundo Caffe2.

Asciinema registrador de sesión de terminal y el mejor compañero de asciinema.org.

HTTPie es un cliente HTTP de línea de comando. Su objetivo es hacer que la interacción de la CLI con los servicios Web sea lo más amigable posible para los humanos. Proporciona un comando http simple que permite el envío de solicitudes HTTP arbitrarias utilizando una sintaxis simple y natural, y muestra una salida coloreada. HTTPie se puede usar para probar, depurar y, en general, interactuar con servidores HTTP.

You-Get es una pequeña utilidad de línea de comandos para descargar contenidos multimedia (videos, audios, imágenes) desde la Web, en caso de que no haya otra forma práctica de hacerlo.

Sentry es un servicio que ayuda a controlar y corregir fallas en tiempo real. El servidor está en Python, pero contiene una API completa para enviar eventos desde cualquier lenguaje, en cualquier aplicación.

Tornado es un framework Web de Python y una biblioteca de red asíncrona, desarrollada originalmente en FriendFeed. Mediante el uso de E/S de red sin bloqueo, Tornado puede escalar a decenas de miles de conexiones abiertas, lo hace ideal para largos sondeos, WebSockets y otras aplicaciones que requieren una conexión de larga duración para cada usuario.

Magenta es un proyecto de investigación que explora el papel del aprendizaje automático en el proceso de creación de arte y música. Principalmente, esto implica desarrollar nuevos algoritmos de aprendizaje profundo y aprendizaje de refuerzo para generar canciones, imágenes, dibujos y otros materiales. Pero también es una exploración en la construcción de herramientas e interfaces inteligentes que permiten a artistas y músicos ampliar sus procesos utilizando estos modelos.

ZeroNet crea sitios Web descentralizados utilizando Bitcoin Crypto y la red BitTorrent.

Gym OpenAI Gym es un conjunto de herramientas para desarrollar y comparar algoritmos de aprendizaje de refuerzo. Esta es la biblioteca de código abierto de Gym, que le da acceso a un conjunto estandarizado de entornos.

Pandas es un paquete de Python que proporciona estructuras de datos rápidas, flexibles y expresivas diseñadas para que trabajar con datos "relacionales" o "etiquetados" sea fácil e intuitivo. Su objetivo es ser el componente fundamental de alto nivel para hacer un análisis práctico y real de datos en Python. Además, tiene el objetivo más amplio de convertirse en la herramienta de análisis / manipulación de datos de código abierto más potente y flexible disponible en cualquier lenguaje.

Luigi es un paquete de Python que te ayuda a construir tuberías complejas de trabajos por lotes. Maneja la resolución de dependencia, la administración del flujo de trabajo, la visualización, el manejo de fallas, la integración de línea de comando y mucho más.

SpaCy (by Explosion AI) es una biblioteca para el procesamiento avanzado del lenguaje natural en Python y Cython, está basado en las últimas investigaciones y fue diseñado desde el primer día para ser utilizado en productos reales. SpaCy viene con modelos estadísticos precompilados y vectores de palabras, y actualmente admite tokenización para más de 20 lenguajes. Cuenta con el analizador sintáctico más rápido del mundo, modelos de redes neuronales convolucionales para etiquetado, análisis y reconocimiento de una entidad nombrada y fácil integración de aprendizaje profundo.

Theano es una biblioteca de Python que permite definir, optimizar y evaluar expresiones matemáticas que involucran matrices multidimensionales de manera eficiente. Puede usar GPU y realizar una diferenciación simbólica eficiente.

TFlearn es una biblioteca de aprendizaje profundo modular y transparente construida sobre Tensorflow. Fue diseñada para proporcionar una API de nivel superior a TensorFlow con el fin de facilitar y agilizar la experimentación, sin dejar de ser totalmente transparente y compatible con ella.

Kivy es un framework Python de código abierto y plataforma para el desarrollo de aplicaciones que hacen uso de interfaces de usuario innovadoras y multitáctiles. El objetivo es permitir un diseño de interacción rápido y fácil y un prototipado rápido a la vez que hace que su código sea reutilizable.

Mailpile es un cliente de correo electrónico moderno y rápido con características de cifrado y privacidad fáciles de usar. El desarrollo de Mailpile esta financiado por una gran comunidad de patrocinadores y todo el código relacionado con el proyecto es y será lanzado bajo una licencia de Software Libre aprobada por OSI.

Matplotlib es una biblioteca de trazado 2D de Python que produce figuras con calidad de publicación en una variedad de formatos impresos y entornos interactivos en todas las plataformas. Matplotlib se puede utilizar en scripts Python, el shell Python e IPython, así como en servidores de aplicaciones Web y varios toolkits de interfaz gráfica de usuario.

YAPF (by Google) toma el código y lo reformatea con el mejor formato que cumpla con la guía de estilo, incluso si el código original no viola la guía de estilo.

Cookiecutter una utilidad de línea de comandos que crea proyectos desde cookiecutters (plantillas de proyecto), por ejemplo creando un proyecto de paquete Python a partir de una plantilla de proyecto de paquete Python.

HTTP Prompt es un cliente HTTP interactivo de línea de comandos con autocompletado y resaltado de sintaxis, basado en prompt_toolkit y HTTPie.

Speedtest-cli interfaz de línea de comandos para probar el ancho de banda de Internet con speedtest.net: http://www.speedtest.net/

Pattern es un módulo de minería Web para Python. Tiene herramientas para Minería de datos, Procesamiento de lenguaje natural, Aprendizaje automático y Análisis de red.

Gooey (Beta) convierte (casi) cualquier programa de consola Python 2 o 3 en una aplicación GUI con una línea.

Wagtail CMS es un sistema de gestión de contenido creado en Django. Se centra en la experiencia del usuario y ofrece un control preciso para diseñadores y desarrolladores.

Bottle es un micro-Framework WSGI rápido, simple y liviano para Python. Se distribuye como un módulo de archivo único y no tiene dependencias distintas de la biblioteca estándar de Python.

Prophet (by Facebook) es un procedimiento para pronosticar datos de series temporales. Se basa en un modelo aditivo en el que las tendencias no lineales se ajustan a la estacionalidad anual y semanal, más las vacaciones. Funciona mejor con datos de periodicidad diaria con al menos un año de datos históricos. Prophet es robusto para datos faltantes, cambios en la tendencia y grandes valores atípicos.

Falcon es un marco Web de Python confiable y de alto rendimiento para construir Backend de aplicaciones a gran escala y microservicios. Fomenta el estilo arquitectónico REST e intenta hacer lo mínimo posible sin dejar de ser altamente efectivo.

Mopidy es un servidor de música extensible escrito en Python. Mopidy reproduce música desde el disco local, Spotify, SoundCloud, Google Play Music y más. Edita la lista de reproducción desde cualquier teléfono, tableta o computadora usando una gama de clientes MPD y Web.

Hug tiene como objetivo hacer que el desarrollar APIs impulsadas por Python sea lo más simple posible, pero no más simple. Como resultado, simplifica drásticamente el desarrollo de la API de Python.

SymPy es una biblioteca de Python para matemática simbólica.

Visdom es una herramienta flexible para crear, organizar y compartir visualizaciones de datos vivos y enriquecidos. Admite Torch y Numpy.

Pygame es una biblioteca de plataforma cruzada diseñada para facilitar la escritura de software multimedia, como juegos en Python.

Requests es una biblioteca de Python que le permite enviar solicitudes HTTP / 1.1, agregar encabezados, datos de formularios, archivos multiparte y parámetros con simples diccionarios de Python. También le permite acceder a los datos de respuesta de la misma manera.

Statsmodels es un paquete de Python que proporciona un complemento para Scipy para cálculos estadísticos que incluyen estadística descriptiva y estimación e inferencia para modelos estadísticos.

Scrapy es ampliamente utilizada en la biblioteca de raspado Web de Python. Se usa para crear programas de rastreo. Inicialmente, fue diseñado para raspar, como su nombre indica, pero ahora se usa para muchos propósitos, incluida la extracción de datos, las pruebas automatizadas, etc. Scrapy es de código abierto.

PyTorch es una biblioteca de código abierto, básicamente es un reemplazo de la biblioteca Numpy y está equipada con funcionalidades de nivel superior para construir redes neuronales profundas. Se puede usar otro lenguaje como Scipy, Cython y Numpy, que ayudan a extender PyTorch cuando sea necesario. Muchas organizaciones, incluyendo Facebook, Twitter, Nvidia, Uber y otras organizaciones usan Pytorch para la creación rápida de prototipos en investigación y para entrenar modelos de aprendizaje profundo.

Requests es una de las famosas bibliotecas de Python que tiene licencia bajo Apache2 y esta escrita en Python. Esta biblioteca ayuda a los humanos a interactuar con los lenguajes. Con la biblioteca de solicitudes, no es necesario que agregue consultas, cadenas manualmente a las URL ni codificar los datos POST. Se puede enviar solicitudes HTTP al servidor mediante la biblioteca de solicitudes y se puede agregar datos de formularios, contenido como encabezado, archivos en varias partes, etc.

PyFlux es una biblioteca de Python que se usa para predecir y analizar series temporales, está desarrollado por Ross Taylor, esta biblioteca tiene muchas opciones para la interfaz y contiene muchas clases nuevas de tipos de modelos. Pyflux permite a los usuarios implementar muchos modelos modernos de series de tiempo como GARCH y predecir la naturaleza de cómo reaccionará en el futuro.

Zappa es uno de los mejores paquetes de Python creados por Miserlou, es tan fácil de construir e implementar aplicaciones sin servidor en API Gateway y Amazon Web Services Lambda. Dado que AWS maneja la escala horizontal de forma automática, por lo que no habrá tiempo de espera de solicitud. Con Zappa, puede actualizar su código en una sola línea con Zappa.

Arrow es una famosa biblioteca de Python amigable para los humanos que ofrece funciones sensatas como crear, formatear, manipular y convertir fechas, horas y marcas de tiempo. Es compatible con Python 2 y 3 y es una alternativa de fecha y hora, ofrece funciones completas con una interfaz más agradable.

Pendulum es un paquete de Python que se utiliza para manipular fechas y horas, el código seguirá funcionando si se reemplazan todos los elementos de DateTime. Con Pendulum, se puede analizar DateTime y mostrar la fecha y hora con la zona horaria. Básicamente, Pendulum es una versión mejorada de la biblioteca Arrow y tiene todos los métodos útiles como redondear, truncar, convertir, analizar, formatear y aritmética.

Theano es una biblioteca de aprendizaje profundo de Python, que se utiliza para optimizar, definir y evaluar ecuaciones numéricas matemáticas y matrices multidimensionales, está desarrollado por el grupo de aprendizaje automático, por lo que, básicamente, Theano es un compilador de expresión matemática y proporciona una estrecha integración con Numpy y proporciona una optimización rápida y estable.

IPython esta es una de las herramientas de Python más útiles, ya que proporciona una rica arquitectura para el usuario. Esta herramienta permite escribir y ejecutar el código Python en el navegador. Ipython funciona en varios sistemas operativos, incluidos Windows, Mac OS X, Linux y la mayoría de los sistemas operativos Unix. IPython brinda todas las características que obtendrá en el intérprete básico con algunas características adicionales como números, más funciones, funciones de ayuda, edición avanzada, etc.

Imbalanced-learn en un mundo ideal, tendríamos conjuntos de datos perfectamente equilibrados y todos entrenaríamos modelos y seríamos felices. Desafortunadamente, el mundo real no es así, y ciertas tareas favorecen datos muy desequilibrados. Por ejemplo, al predecir el fraude en las transacciones de tarjetas de crédito, es de esperar que la gran mayoría de las transacciones (+ 99.9%) sean realmente legítimas. El entrenamiento ingenuo de algoritmos de ML conducirá a un rendimiento deprimente, por lo que se necesita cuidado adicional al trabajar con estos tipos de conjuntos de datos. Afortunadamente, este es un problema de investigación estudiado y existe una variedad de técnicas. Imbalanced-learn es un paquete de Python que ofrece implementaciones de algunas de esas técnicas, para hacer la vida mucho más fácil. Es compatible con Scikit-learn y es parte de los proyectos Scikit-learn-contrib.

Caffe2 el marco original de Caffe ha sido ampliamente utilizado durante años, y es conocido por su rendimiento incomparable y base de código probado en batalla. Sin embargo, las tendencias recientes en DL hicieron que el marco se estancara en algunas direcciones. Caffe2 es el intento de llevar Caffe al mundo moderno. Admite formación distribuida, implementación (incluso en plataformas móviles), las CPU más nuevas y Hardware compatible con CUDA. Si bien PyTorch puede ser mejor para la investigación, Caffe2 es adecuado para despliegues a gran escala como se ve en Facebook.

Dash es una biblioteca de código abierto para crear aplicaciones Web, especialmente aquellas que hacen un buen uso de la visualización de datos, en Python puro. esta construido sobre Flask, Plotly.js y React, y proporciona abstracciones que te liberan de tener que aprender esos Frameworks y permitirte ser productivo rápidamente. Las aplicaciones se representan en el navegador y responderán para que se puedan usar en dispositivos móviles. No se requiere JavaScript.

Fire es una biblioteca de código abierto que puede generar automáticamente una CLI para cualquier proyecto de Python. La clave aquí es automática: ¡casi no es necesario escribir ningún código o docstrings para construir una CLI!. Para hacer el trabajo, solo se tiene que llamar a un método Fire y pasarlo como se quiera para convertirlo en una CLI: una función, un objeto, una clase, un diccionario, o incluso no pasar ningún tipo de argumento (lo que convertirá todo el código en una CLI).

Flashtext es una biblioteca para búsqueda y reemplazo de palabras en un documento. La belleza de FlashText es que el tiempo de ejecución es el mismo sin importar cuántos términos de búsqueda se tenga, en contraste con la expresión regular en la que el tiempo de ejecución aumentará casi linealmente con el número de términos.

Pipenv con Pipenv, se especifican todas las dependencias en un Pipfile, que normalmente se genera mediante el uso de comandos para agregar, eliminar o actualizar dependencias. La herramienta puede generar un archivo Pipfile.lock, lo que permite que las compilaciones sean deterministas, ayudando a evitar esos errores difíciles de detectar debido a una dependencia poco clara que ni siquiera se cree que es necesaria.

Luminoth las imágenes están en todas partes hoy en día y comprender su contenido puede ser crítico para varias aplicaciones. Afortunadamente, las técnicas de procesamiento de imágenes han avanzado mucho, impulsadas por los avances en DL. Luminoth es un kit de herramientas de código abierto Python para visión artificial, construido con TensorFlow y Sonnet. Actualmente, viene de fábrica y es compatible con la detección de objetos en forma de un modelo llamado Faster R-CNN.

Instalación de Python e IDEs Para instalar Python 3 en Debian GNU/Linux es necesario hacer:

```
# apt install ipython3 python3 idle3 python3-pip \
python3-matplotlib python3-rpy2 python3-numpy spyder3 \
python3-scipy bpython3 python3-pandas python-sklearn \
python-sklearn-docspe python-wxgtk3.0 jython xonsh \
python3-mpi4pypython3-pyqt5 python3-pyqtgraph mypy \
python-wxgtk3.0-dev python3-numba pyflakes3
# apt install flake8 black
```

Para instalar Jupyter (entorno de trabajo orientado a científicos que soporta los lenguajes R y Python):

```
# apt install jupyter-console jupyter-notebook
# pip3 install jupyter
# pip3 install matplotlib
# pip3 install ipywidgets
# jupyter nbextension enable –py –sys-prefix \
widgetsnbextension
```

y podemos instalar PYREPOT usando:

```
# pip install pyreport
```

además podemos instalar editores especializados en Python usando:

```
# apt install eric pyzo pyzo-doc thonny
```

otras opciones se pueden descargar de:

https://www.jetbrains.com/pycharm/
http://www.pydev.org/
https://wingware.com/

También, se pueden instalar diversas herramientas e IDEs para facilitar la programación en Python, para ello usar:

```
# apt install scite jedit kate gedit nedit emacs medit \
kscope geany geany-plugins editra qtcreator anjuta \
anjuta-extras codelite codelite-plugins tea vim-gtk \
mousepad eric neovim neovim-qt medit kwrite katepart
# apt install fte fte-console fte-terminal nano joe vim \
vim-python-jedi vim-tlib vim-latexsuite vim-nox micro \
neovim micro kakoune vim-athena jed
# apt install kdiff3 meld diffuse dirdiff kompare numdiff \
colordiff dwdiff wdiff xxdiff tkdiff ndiff ccdiff xxdiff
# apt install alleyoop astyle c2html java2html code2html \
c2html autodia txt2html html2text moreutils
# apt install git git-all gitk gitg git-cola git-gui qgit tig \
vim-fugitive git-extras
# apt install mercurial
# apt install subversion rapidsvn
# apt install cvs tkcvs
```

Compilar y Ejecutar Para compilar el archivo ejemplo.py en *python3* en línea de comandos:

- para compilarlo y ejecutarlo en *python3* en línea de comandos usamos:

  ```
  $ python3 ejemplo.py
  ```

- en caso de necesitar depurar de forma interactiva un código, podemos usar el módulo *pdb*, por ejemplo:

  ```
  $ pdb ejemplo.py
  ```

- en caso de necesitar hacer una compilación estática podemos usar:

  ```
  $ pyflakes ejemplo.py
  ```

Codificar Según Guía de Estilo PEP8 Python cuenta con herramientas que permiten al programador conocer en que discrepa su código de la guía de estilo de Python PEP8 y en caso de que lo requiera, se puede usar estas para que reformatee nuestro código siguiendo al PEP8.

- en caso querer revisar las discrepancias de nuestro código con respecto a la guía oficial de estilo PEP8 de Python usamos:

  $ flake8 ejemplo.py

- en caso de querer conocer qué cambios se sugiere hacer al código según la guía oficial de estilo PEP8 de Python usamos:

  $ black –check ejemplo.py
  $ black –diff ejemplo.py

- en caso de querer reformatear el código según la guía oficial de estilo PEP8 de Python usamos:

  $ black ejemplo.py
  $ black *.py

Anaconda Por otro lado existe Anaconda, una Suite de código abierto que abarca una serie de aplicaciones, librerías y conceptos diseñados para el desarrollo de la Ciencia de datos con Python. En líneas generales Anaconda Distribu-tion es una distribución de Python que funciona como un gestor de entorno, un gestor de paquetes y que posee una colección de más de 720 paquetes de código abierto. Anaconda Distribution se agrupa en 4 sectores o soluciones tecnológicas; *Anaconda Navigator, Anaconda Project, las librerías de Ciencia de Datos y Conda*. Todas estas se instalan de manera automática y en un procedimiento muy sencillo.

Para más información ver: https://www.anaconda.com/.

También está SageMath, una Suite de código abierto bajo la licencia GPL de Software matemático como: N*umPy, SciPy, matplotlib, Sympy, Maxima, GAP, FLINT, R*, entre otros. Además combina acceso a una poderosa combinación del lenguaje basada en Python o directamente vía interfaces o *Wrappers*. La misión del proyecto es crear una alternativa de Software libre a *Magma, Maple, Mathematica y Matlab*.

Para más información ver: http://www.sagemath.org/.

Instalación de Aplicaciones Usando Pip Pip es un sistema de administración de paquetes que se utiliza para instalar y administrar paquetes de Software escritos en Python. Pip se usa principalmente para instalar paquetes disponibles en Python Package Index (PyPI, página del proyecto: https://pypi.org). Los desarrolladores también pueden usar Pip para instalar módulos y paquetes desarrollados localmente.

Para instalar Pip en Python 2 hacemos:

```
# apt install python-pip
```

y para instalar alguna aplicación para todos los usuarios, por ejemplo *ratarmount*, usamos:

```
# pip2 install ratarmount
```

y para instalar alguna aplicación para el usuario, por ejemplo *ratarmount*, usamos:

```
$ pip2 install –user ratarmount
```

Para instalar Pip en Python 3 hacemos:

```
# apt install python3-venv python3-pip
```

y para instalar alguna aplicación para todos los usuarios, por ejemplo *ratarmount*, usamos:

```
# pip3 install ratarmount
```

y para instalar alguna aplicación para el usuario, por ejemplo *ratarmount*, usamos:

```
$ pip3 install –user ratarmount
```

En caso de instalación para el usuario, para usar la aplicación debemos agregar al PATH:

```
export PATH ="$PATH:/home/$USER/.local/bin"
```

Sin pérdida de generalidad (usando pip2 o pip3), podemos ver los detalles de algún paquete, usando:

pip3 show nombre

Podemos instalar algún paquete, usando:

pip3 install nombre

Podemos actualizar algún paquete, usando:

pip3 install –upgrade nombre

Podemos desinstalar algún paquete, usando:

pip3 uninstall nombre

Podemos listar los paquetes instalados, usando:

pip3 list nombre

Podemos buscar algún paquete, usando:

pip3 search nombre

Podemos listar los paquetes desactualizados, usando:

pip3 list –outdated

3.3 Julia

es un lenguaje de programación homoicónico, multiplataforma y multiparadigma (véase [**?**]) -soporta programación orientada a objetos, por procedimientos, funcional y meta además de multietapas- de tipado dinámico de alto nivel y alto desempeño para la computación genérica, técnica y científica, con una sintaxis similar a la de otros entornos de computación similares, con licencia MIT (véase apéndice 11.2).

Dispone de un compilador avanzado (JIT), mecanismos para la ejecución en paralelo y distribuida, además de una extensa biblioteca de funciones matemáticas. La biblioteca, desarrollada fundamentalmente en Julia, también contiene código desarrollado en C o Fortran, para el álgebra lineal, generación de números aleatorios, procesamiento de señales, y procesamiento

de cadenas. Adicionalmente, la comunidad de desarrolladores de Julia contribuye con la creación y distribución de paquetes externos a través del gestor de paquetes integrado de Julia a un paso acelerado. IJulia es el resultado de la colaboración entre las comunidades de IPython y Julia, provee de una poderosa interfaz gráfica basada en el navegador para Julia.

Algunas características básicas son:

- El despacho múltiple: permite definir el comportamiento de las funciones a través de diversas combinaciones de tipos de argumentos
- Sistema de tipado dinámico: tipos para la documentación, la optimización y el despacho de funciones
- Buen desempeño, acercándose al de lenguajes estáticamente compilados como C
- Gestor de paquetes integrado
- Macros tipo Lisp y otras herramientas para la meta-programación
- Llamar funciones de Python: mediante el paquete PyCall
- Llamar funciones de C directamente: sin necesidad de usar envoltorios u APIs especiales
- Poderosas características de línea de comandos para gestionar otros procesos
- Diseñado para la computación paralela y distribuida
- Corutinas: hilos ligeros
- Los tipos definidos por el usuario son tan rápidos y compactos como los tipos estándar integrados.
- Generación automática de código eficiente y especializado para diferentes tipos de argumentos
- Conversiones y promociones para tipos numéricos y de otros tipos, elegantes y extensibles
- Soporte eficiente para Unicode, incluyendo UTF-8 pero sin limitarse solo a este

- Es de uso general, pero tiene todo lo necesario para hacer ciencia de datos, aprendizaje automático, ecuaciones diferenciales, resolvedores de sistemas lineales, etc.

Julia incluye una terminal interactiva, llamada REPL en donde se puede visualizar automáticamente los resultados de la ejecución del programa o segmento de código. Tanto el compilador justo a tiempo (JIT) basado en LLVM así como el diseño del lenguaje le permiten a Julia acercarse e incluso igualar a menudo el desempeño de C. Julia no le impone al usuario ningún estilo de paralelismo en particular. En vez de esto, le provee con bloques de construcción clave para la computación distribuida, logrando hacer lo suficientemente flexible el soporte de varios estilos de paralelismo y permitiendo que los usuarios añadan más.

Instalación de Julia e IDEs Existen diversas versiones de Julia para Linux, para instalar en Debian GNU/Linux es necesario hacer:

```
# apt install julia julia-doc
```

Además, se pueden instalar diversas herramientas e IDEs para facilitar la programación en Julia, para ello usar:

```
# apt install scite jedit kate gedit nedit emacs medit \
kscope geany geany-plugins editra qtcreator anjuta \
anjuta-extras codelite codelite-plugins tea vim-gtk \
mousepad eric neovim neovim-qt medit kwrite katepart
# apt install eclipse eclipse-cdt eclipse-pydev netbeans \
bluefish bluefish-plugins codeblocks codeblocks-contrib
# apt install fte fte-console fte-terminal nano joe vim \
vim-python-jedi vim-tlib vim-latexsuite vim-nox micro \
neovim micro kakoune vim-athena jed
# apt install kdiff3 meld diffuse dirdiff kompare numdiff \
colordiff dwdiff wdiff xxdiff tkdiff ndiff ccdiff xxdiff
# apt install alleyoop astyle c2html java2html code2html \
c2html autodia txt2html html2text
# apt install git git-all gitk gitg git-cola git-gui qgit tig \
vim-fugitive git-extras
```

```
# apt install mercurial
# apt install subversion rapidsvn
# apt install cvs tkcvs
```

3.4 C y C++

C (véase [**?**]) es un lenguaje de programación originalmente desarrollado por Dennis Ritchie entre 1969 y 1972 en los Laboratorios Bell, como evolución del lenguaje anterior B, a su vez basado en BCPL. Es un lenguaje orientado a la implementación de Sistemas operativos, concretamente Unix, Linux y el Kernel de Linux. *C* es apreciado por la eficiencia del código que produce y es el lenguaje de programación más popular para crear Software de sistemas, aunque también se utiliza para crear aplicaciones.

Se trata de un lenguaje de tipos de datos estáticos, débilmente tipificado, de medio nivel, ya que dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel. Los compiladores suelen ofrecer extensiones al lenguaje que posibilitan mezclar código en ensamblador con código *C* o acceder directamente a memoria o dispositivos periféricos.

Filosofía Uno de los objetivos de diseño del lenguaje *C* es que sólo sean necesarias unas pocas instrucciones en lenguaje máquina para traducir cada elemento del lenguaje, sin que haga falta un soporte intenso en tiempo de ejecución. Es muy posible escribir *C* a bajo nivel de abstracción; de hecho, *C* se usó como intermediario entre diferentes lenguajes.

En parte, a causa de ser relativamente de bajo nivel y tener un modesto conjunto de características, se pueden desarrollar compiladores de *C* fácilmente. En consecuencia, el lenguaje *C* está disponible en un amplio abanico de plataformas (más que cualquier otro lenguaje). Además, a pesar de su naturaleza de bajo nivel, el lenguaje se desarrolló para incentivar la programación independiente de la máquina. Un programa escrito cumpliendo los estándares e intentando que sea portátil puede compilarse en muchos computadores.

C se desarrolló originalmente (conjuntamente con el sistema operativo Unix, con el que ha estado asociado mucho tiempo) por programadores para programadores. Sin embargo, ha alcanzado una popularidad enorme, y se ha usado en contextos muy alejados de la programación de Software de sistemas, para la que se diseñó originalmente.

Propiedades Núcleo del lenguaje simple, con funcionalidades añadidas importantes, como funciones matemáticas y de manejo de archivos, proporcionadas por bibliotecas. Es un lenguaje muy flexible que permite programar con múltiples estilos. Uno de los más empleados es el estructurado "no llevado al extremo", entre sus principales propiedades destacan:

- Un sistema de tipos que impide operaciones sin sentido
- Usa un lenguaje de preprocesado, el preprocesador de *C*, para tareas como definir macros e incluir múltiples archivos de código fuente
- Acceso a memoria de bajo nivel mediante el uso de punteros
- Interrupciones al procesador con uniones
- Un conjunto reducido de palabras clave
- Por defecto, el paso de parámetros a una función se realiza por valor. El paso por referencia se consigue pasando explícitamente a las funciones las direcciones de memoria de dichos parámetros
- Punteros a funciones y variables estáticas, que permiten una forma rudimentaria de encapsulado y polimorfismo
- Tipos de datos agregados (struct) que permiten que datos relacionados (como un empleado, que tiene un id, un nombre y un salario) se combinen y se manipulen como un todo (en una única variable "empleado")

Carencias Aunque la lista de las características útiles de las que carece *C* es larga, éstos factores han sido importantes para su aceptación, porque escribir rápidamente nuevos compiladores para nuevas plataformas, mantiene lo que realmente hace el programa bajo el control directo del programador, y permite implementar la solución más natural para cada plataforma. Esta es la causa de que a menudo *C* sea más eficiente que otros lenguajes. Típicamente, sólo la programación cuidadosa en lenguaje ensamblador produce un código más rápido, pues da control total sobre la máquina, aunque los avances en los compiladores de *C* y la complejidad creciente de los microprocesadores modernos han reducido gradualmente esta diferencia, Algunas carencias son:

- Recolección de basura nativa, sin embargo se encuentran a tal efecto bibliotecas como la "*libgc*" desarrollada por Sun Microsystems, o el Recolector de basura de Boehm
- Soporte para programación orientada a objetos, aunque la implementación original de *C++* fue un preprocesador que traducía código fuente de *C++* a *C*. Véase también la librería *GObjec*t
- Funciones anidadas, aunque *GCC* tiene esta característica como extensión
- Soporte nativo para programación multihilo. Disponible usando librerías como libpthread

Ventajas estas se pueden resumir en:

- Lenguaje muy eficiente puesto que es posible utilizar sus características de bajo nivel para realizar implementaciones óptimas
- A pesar de su bajo nivel es el lenguaje más portado en existencia, habiendo compiladores para casi todos los sistemas de cómputo conocidos
- Proporciona facilidades para realizar programas modulares y/o utilizar código o bibliotecas existentes

Inconvenientes El mayor problema que presenta el lenguaje *C* frente a los lenguajes de tipo de dato dinámico es la gran diferencia en velocidad de desarrollo: es más lento programar en *C*, sobre todo para el principiante. La razón estriba en que el compilador de *C* se limita a traducir código sin apenas añadir nada. La gestión de la memoria es un ejemplo clásico: en *C* el programador ha de reservar y liberar la memoria explícitamente. En otros lenguajes (como *BASIC*, *MATLAB* o *C#*) la memoria es gestionada de forma transparente para el programador. Esto alivia la carga de trabajo humano y en muchas ocasiones evita errores, aunque también supone mayor carga de trabajo para el procesador.

El mantenimiento en algunos casos puede ser más difícil y costoso que con ciertos lenguajes de más alto nivel. El código en *C* se presta a sentencias cortas y enrevesadas de difícil interpretación.

Cabe destacar el contexto y época en la que fue desarrollado *C*. En aquellos tiempos existían muy pocos programadores, los cuales, a su vez, eran prácticamente todos expertos en el área. De esta manera, se asumía que los programadores eran conscientes de sus trabajos y capaces de manejar perfectamente el lenguaje. Por esta razón es muy importante que los recién iniciados adopten buenas prácticas a la hora de escribir en *C* y manejar la memoria, como por ejemplo un uso intensivo de indentación y conocer a fondo todo lo que implica el manejo de punteros y direcciones de memoria.

Aplicabilidad Hecho principalmente para la fluidez de programación en sistemas UNIX. Se usa también para el desarrollo de otros sistemas operativos como Windows o GNU/Linux. Igualmente para aplicaciones de escritorio como GIMP, cuyo principal lenguaje de programación es *C*.

De la misma forma, es muy usado en aplicaciones científicas (para experimentos informáticos, físicos, químicos, matemáticos, entre otros, conocidos como modelos y simuladores), industriales (industria robótica, cibernética, sistemas de información y base de datos para la industria petrolera y petroquímica). Predominan también todo lo que se refiere a simulación de máquinas de manufactura, simulaciones de vuelo (es la más delicada, ya que se tienen que usar demasiados recursos tanto de Hardware como de Software para desarrollar aplicaciones que permitan simular el vuelo real de una aeronave). Se aplica por tanto, en diversas áreas desconocidas por gran parte de los usuarios noveles.

Los equipos de cómputo de finales de los 90 son varios órdenes de magnitud más potentes que las máquinas en que C se desarrolló originalmente. Programas escritos en lenguajes de tipo dinámico y fácil codificación (Ruby, Python, Perl, etc.) que antaño hubieran resultado demasiado lentos, son lo bastante rápidos como para desplazar en uso a *C*. Aun así, se puede seguir encontrando código *C* en grandes desarrollos de animaciones, modelados y escenas en 3D en películas y otras aplicaciones multimedia.

Actualmente, los grandes proyectos de Software se dividen en partes, dentro de un equipo de desarrollo. Aquellas partes que son más "burocráticas" o "de gestión" con los recursos del sistema, se suelen realizar en lenguajes de tipo dinámico o de guión (script), mientras que aquellas partes "críticas", por su necesidad de rapidez de ejecución, se realizan en un lenguaje de tipo compilado, como *C* o *C++*. Si después de hacer la división, las partes críticas no superan un cierto porcentaje del total (aproximadamente el 10%)

entonces todo el desarrollo se realiza con lenguajes dinámicos. Si la parte crítica no llega a cumplir las expectativas del proyecto, se comparan las alternativas de una inversión en nuevo Hardware frente a invertir en el coste de un programador para que reescriba dicha parte crítica.

Ya que muchos programas han sido escritos en el lenguaje *C* existe una gran variedad de bibliotecas disponibles. Muchas bibliotecas son escritas en *C* debido a que *C* genera código objeto rápido; los programadores luego generan interfaces a la biblioteca para que las rutinas puedan ser utilizadas desde lenguajes de mayor nivel, tales como Java, Perl y Python.

C++ (véase [**?**]) es un lenguaje de programación diseñado a mediados de 1979 por Bjarne Stroustrup. La intención de su creación fue extender al lenguaje de programación C mecanismos que permiten la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, *C++* es un lenguaje híbrido.

Posteriormente se añadieron facilidades de programación genérica, que se sumaron a los paradigmas de programación estructurada y programación orientada a objetos. Por esto se suele decir que el *C++* es un lenguaje de programación multiparadigma.

Actualmente existe un estándar, denominado ISO *C++*, al que se han adherido la mayoría de los fabricantes de compiladores más modernos. Existen también algunos intérpretes, tales como ROOT.

Una particularidad de *C++* es la posibilidad de redefinir los operadores, y de poder crear nuevos tipos que se comporten como tipos fundamentales.

El nombre "*C++*" fue propuesto por Rick Mascitti en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico. Antes se había usado el nombre "*C* con clases". En *C++*, la expresión "*C++*" significa "incremento de *C*" y se refiere a que *C++* es una extensión de *C*.

El concepto de clase Los objetos en C++ son abstraídos mediante una clase. Según el paradigma de la programación orientada a objetos un objeto consta de:

- Identidad, que lo diferencia de otros objetos (Nombre que llevará la clase a la que pertenece dicho objeto).
- Métodos o funciones miembro

- Atributos o variables miembro

Diferencias de tipos respecto a C En C++, cualquier tipo de datos que sea declarado completo (fully qualified, en inglés) se convierte en un tipo de datos único. Las condiciones para que un tipo de datos T sea declarado completo son a grandes rasgos las siguientes:

- Es posible al momento de compilación conocer el espacio asociado al tipo de datos (es decir, el compilador debe conocer el resultado de sizeof(T))
- T Tiene al menos un constructor, y un destructor, bien declarados
- Si T es un tipo compuesto, o es una clase derivada, o es la especificación de una plantilla, o cualquier combinación de las anteriores, entonces las dos condiciones establecidas previamente deben aplicar para cada tipo de dato constituyente
- En general, esto significa que cualquier tipo de datos definido haciendo uso de las cabeceras completas, es un tipo de datos completo
- En particular, y a diferencia de lo que ocurría en *C*, los tipos definidos por medio de struct o enum son tipos completos. Como tales, ahora son sujetos a sobrecarga, conversiones implícitas, etcétera

Los tipos enumerados, entonces, ya no son simplemente alias para tipos enteros, sino que son tipos de datos únicos en *C++*. El tipo de datos bool, igualmente, también pasa a ser un tipo de datos único, mientras que en *C* funcionaba en algunos casos como un alias para alguna clase de dato de tipo entero.

Compiladores Uno de los compiladores libres de C++ es el de GNU, el compilador G++ (parte del proyecto GCC, que engloba varios compiladores para distintos lenguajes). Otros compiladores comunes son Intel C++ Compiler, el compilador de Xcode, el compilador de Borland C++, el compilador de CodeWarrior C++, el compilador g++ de Cygwin, el compilador g++ de MinGW, el compilador de Visual C++, Carbide.c++, entre otros.

Instalación de C y C++ e IDEs Existen diversas versiones de C y C++ para Linux, para instalarlos en Debian GNU/Linux es necesario hacer:

```
# apt install build-essential manpages-dev glibc-doc \
glibc-doc-reference gcc-doc-base gcc-doc splint \
c++-annotations-pdf g++ jam ohcount ctags \
cppcheck cccc autoconf automake make cmake scons
```

Además, se pueden instalar diversas herramientas e IDEs para facilitar la programación en C y C++, para ello usar:

```
# apt install scite jedit kate gedit nedit emacs medit \
kscope geany geany-plugins editra qtcreator anjuta \
anjuta-extras codelite codelite-plugins tea vim-gtk \
mousepad eric neovim neovim-qt medit kwrite katepart
# apt install eclipse eclipse-cdt eclipse-pydev netbeans \
bluefish bluefish-plugins codeblocks codeblocks-contrib
# apt install fte fte-console fte-terminal nano joe vim \
vim-python-jedi vim-tlib vim-latexsuite vim-nox micro \
neovim micro kakoune vim-athena jed
# apt install kdiff3 meld diffuse dirdiff kompare numdiff \
colordiff dwdiff wdiff xxdiff tkdiff ndiff ccdiff xxdiff
# apt install alleyoop astyle c2html java2html code2html \
c2html autodia txt2html html2text indent
# apt install git git-all gitk gitg git-cola git-gui qgit tig \
vim-fugitive git-extras
# apt install mercurial
# apt install subversion rapidsvn
# apt install cvs tkcvs
```

Compilar y Ejecutar Para compilar el archivo ejemplo.c en *C* en línea de comandos usamos:

```
$ gcc ejemplo.c -o ejemplo
```

y lo ejecutamos con:

```
$ ./ejemplo
```

Para compilar el archivo ejemplo.cpp en *C++* en línea de comandos usamos:

```
$ g++ ejemplo.cpp -o ejemplo
```

y lo ejecutamos con:

```
$ ./ejemplo
```

3.5 Fortran

Fortran (véase [**?**]) contracción del inglés The IBM Mathematical Formula Translating System, es un lenguaje de programación de alto nivel de propósito general, procedimental e imperativo, que está especialmente adaptado al cálculo numérico y a la computación científica. Desarrollado originalmente por IBM en 1957 para el equipo IBM 704, y usado para aplicaciones científicas y de ingeniería, el Fortran vino a dominar esta área de la programación desde el principio y ha estado en uso continuo por más de medio siglo en áreas de cómputo intensivo tales como la predicción numérica del tiempo, análisis de elementos finitos, dinámica de fluidos computacional, física computacional y química computacional. Es uno de los lenguajes más populares en el área de la computación de alto rendimiento y es el lenguaje usado para programas que evalúan el desempeño (benchmark) y el ranking de los supercomputadores más rápidos del mundo.

El Fortran abarca un linaje de versiones, cada una de las cuales evolucionó para añadir extensiones al lenguaje mientras que usualmente retenía compatibilidad con las versiones previas. Versiones sucesivas han añadido soporte para procesamiento de datos basados en caracteres (Fortran 77), programación de arreglos, programación modular y programación orientada a objetos (Fortran 90/95), y programación genérica (Fortran 2003/2008).

Ventajas e inconvenientes de su sintaxis como fue una primera tentativa de creación de un lenguaje de programación de alto nivel, tiene una sintaxis considerada arcaica por muchos programadores que aprenden lenguajes más modernos. Es difícil escribir un bucle "for", y errores en la escritura de un solo carácter pueden llevar a errores durante el tiempo de ejecución en vez de errores de compilación, en el caso de que no se usen las construcciones más frecuentes. Algunas de las primeras versiones no poseían

facilidades que son consideradas muy útiles, tal como la asignación dinámica de memoria.

Se debe tener en cuenta que la sintaxis de Fortran fue orientada para el uso en trabajos numéricos y científicos. Muchas de sus deficiencias han sido abordadas en revisiones recientes del lenguaje. Por ejemplo, en 1990 se presentó un tercer estándar ANSI conocido como FORTRAN 90, que contenía muchas nuevas características y permitía una programación más estructurada. Una serie de cambios menores se presentaron en 1995 conocido como FORTRAN 95 posee comandos mucho más breves para efectuar operaciones matemáticas con matrices y dispone de tipos y soporta OpenMP. Esto no solo mejora mucho la lectura del programa sino que además aporta información útil al compilador. Actualmente se tienen estándares para las versiones 2003 y 2008 y se continúa trabajando en mejoras al lenguaje.

Por estas razones Fortran prácticamente no se usa fuera de los campos científicos y del análisis numérico, pero permanece como el lenguaje preferido para desarrollar aplicaciones de computación numérica de alto rendimiento.

Características

Tipos de datos soportados:

- Numéricos (enteros, reales, complejos y de doble precisión).
- Boleanos (logical).
- Arreglos.
- Cadenas de caracteres.
- Archivos.

FORTRAN 90/95 ya es estructurado, y no requiere sentencias GOTO. Sólo admite dos ámbitos para las variables: local y global.

Variables y constantes

- Fortran no es sensible a mayúsculas y minúsculas. Los nombres de variables tienen de 6 a 31 caracteres máximo y deben comenzar por una letra. Los blancos son significativos.

- Declaración explícita de variables.
- Enteras (I-N), el resto reales. (se modifica con IMPLICIT).
- Punteros: en los primeros FORTRAN no hay punteros y todas las variables se almacenan en memoria estática. En FORTRAN 90 se declaran INTEGER, POINTER::P.
- Para memoria dinámica ALLOCATE y DEALLOCATE

Tipos de datos

- Arrays, pueden tener hasta 7 dimensiones y se guardan por columnas.
- REAL M(20),N(-5:5)
- DIMENSION I(20,20) (tipo por nomenclatura implícita)
- Cadenas de caracteres, el primer carácter es el 1, el operador // permite concatenar cadenas.
- CHARACTER S*10, T*25
- Almacenamiento de datos. Se usa COMMON para datos compartidos y EQUIVALENCE cuando almacenamos una variable con dos posibles tipos en la misma posición de memoria (como union en C). Se usa DATA para inicializar datos estáticos.
- DATA X/1.0/,Y/3.1416/,K/20/
- Tipos definidos por el usuario, con TYPE <nombre>... END TYPE <nombre>

Control de secuencia el conjunto de estructuras de control es limitado:

- Expresiones, prioridad de operadores
- Enunciados

- ◦ Asignación, cuando se hace entre cadenas hay ajuste de tamaño con blancos o truncamiento.
- ◦ Condicional. Permite IF ELSE IF... Para selección múltiple SELECT CASE CASE.....CASE DEFAULT.... END SELECT
- ◦ Iteración. DO....END DO
- ◦ Nulo, se usa solo para la etiqueta. CONTINUE.

- Control de subprogramas. CALL invoca al subprograma y RETURN devuelve un valor al programa llamante.
- Construcciones propensas a error: GOTO.

Entrada y salida

- Tipos de archivos:
  - ◦ Secuenciales
  - ◦ De acceso directo
- Comandos: READ, WRITE, PRINT, OPEN , CLASE, INQUIRE (propiedades o estado del archivo) REWIND y ENDFILE (para ubicar el puntero del fichero).
- Para el tratamiento de excepciones en las sentencias READ/WRITE se puede introducir la posición de la rutina de dicho tratamiento (ERR=90).

Subprogramas

- Hay tres tipos de subprogramas:

◦ Function, devuelven un solo valor de tipo numérico, lógico o cadena de caracteres.

◦ Subroutine, devuelve valores a través de variables no locales COMMON.

◦ Función de enunciado, permite calcular una sola expresión aritmética o lógica.

◦ FN(X,Y)=SIN(X)**2-COS(Y)**2

- Gestión de almacenamiento.
  - ◦ Las variables son locales o globales (COMMON)
  - ◦ Recursividad: RECURSIVE FUNCTION FACTORIAL(X)
  - ◦ Parámetros de subprograma. Paso por referencia.

Abstracción y encapsulación. Evaluación del lenguaje

- La abstracción es posible mediante los subprogramas y el uso de variables COMMON, aunque su uso es propenso a errores.
- FORTRAN sigue siendo utilizado en el ámbito científico y es muy eficiente realizando cálculos.
  - La estructura del programa suele ser difícil de entender.
  - En FORTRAN 90/95 se incluye la recursividad y la memoria dinámica.
  - Las etiquetas de las sentencias ya no son necesarias, ni el GOTO, pues se ha transformado en un lenguaje estructurado.
  - El aspecto de los programas sigue siendo de procesamiento por lotes

Instalación de Fortran e IDEs Existen diversas versiones de Fortran para Linux, para instalarlos en Debian GNU/Linux es necesario hacer:

```
# apt install gfortran gfortran-doc fortran77-compiler \
fortran95-compiler fortran-compiler cfortran ftc fort77
```

Además, se pueden instalar diversas herramientas e IDEs para facilitar la programación en Fortran, para ello usar:

```
# apt install scite jedit kate gedit nedit emacs medit \
kscope geany geany-plugins editra qtcreator anjuta \
anjuta-extras codelite codelite-plugins tea vim-gtk \
mousepad eric neovim neovim-qt medit kwrite katepart
# apt install fte fte-console fte-terminal nano joe vim \
vim-python-jedi vim-tlib vim-latexsuite vim-nox micro \
neovim micro kakoune vim-athena jed
# apt install kdiff3 meld diffuse dirdiff kompare numdiff \
colordiff dwdiff wdiff xxdiff tkdiff ndiff ccdiff xxdiff
# apt install alleyoop astyle c2html java2html code2html \
c2html autodia txt2html html2text
# apt install git git-all gitk gitg git-cola git-gui qgit tig \
vim-fugitive git-extras
```

```
# apt install mercurial
# apt install subversion rapidsvn
# apt install cvs tkcvs
```

Compilar y Ejecutar Para compilar el archivo ejemplo.f77 en *Fortran 77* en línea de comandos usamos:

```
$ g77 ejemplo.f77 -o ejemplo
```

y lo ejecutamos con:

```
$ ./ejemplo
```

Para compilar el archivo ejemplo.f90 en *Fortran 90/95/2003* en línea de comandos usamos:

```
$ gfortran ejemplo.f90 -o ejemplo
```

y lo ejecutamos con:

```
$ ./ejemplo
```

3.6 R

El paquete R (véase [39]) es un lenguaje y entorno de programación para análisis estadístico y gráfico. Se trata de un proyecto de Software libre, resultado de la implementación GNU del premiado lenguaje S. SPSS, R y S-Plus —versión comercial de S— son, probablemente, los tres lenguajes más utilizados en investigación por la comunidad estadística, siendo además muy populares en el campo de la investigación biomédica, la bioinformática y las matemáticas financieras. A esto contribuye la posibilidad de cargar diferentes bibliotecas o paquetes con finalidades específicas de cálculo o gráfico.

Además, R puede integrarse con distintas bases de datos y existen bibliotecas que facilitan su utilización desde lenguajes de programación interpretados como Perl y Python. R soporta hacer interfase con lenguajes de programación como C, C++ y Fortran.

Otra de las características de R es su capacidad gráfica, que permite generar gráficos con alta calidad. R posee su propio formato para la documentación basado en LaTeX (véase [28]). R también puede usarse como

herramienta de cálculo numérico, campo en el que puede ser tan eficaz como otras herramientas específicas tales como FreeMat, GNU Octave y su equivalente comercial, MATLAB. Se ha desarrollado una interfaz RWeka para interactuar con Weka (véase [66]) que permite leer y escribir ficheros en el formato arff y enriquecer R con los algoritmos de minería de datos de dicha plataforma.

Los ambientes de desarrollo integrado para R existen como proyectos externos, como pueden ser editores —que sólo soportan la sintaxis—, los IDEs (Integrate Development Environments) y los GUI (Graphical User Interfaces) —permiten editar, ejecutar y depurar código desarrollado para R—. Hay más de 20 proyectos activos, tres de los más conocidos son Tinn-R (véase [67]), RStudio (véase [68]) y Rkward de KDE.

Instalación de R Existen diversos paquetes para *R* en Debian GNU/Linux, para instalarlos es necesario hacer:

```
# apt install r-base
# apt install 'r-cran*'
# apt install 'r-bioc*'
```

Editores nativo en Debian GNU/Linux

```
# apt install rkward rkward-data
```

y podemos generar gráficos de datos usando:

```
# apt install rlplot
```

para conocer todos los paquetes asociados, usar:

```
$ apt search ^r-cran
$ apt search ^r-bioc
```

Paquetes internos

```
R2WinBugs mcmcplots devtools Rattle e1071 FactiMineR \
Ellipse xlsx hsaur shiny
```

3.7 Herramientas de Programación

En Linux existe una gran variedad de herramientas para programación, ya que este sistema operativo fue hecho por programadores y para programadores, por ello entre las miles de herramientas, tenemos algunas que son ampliamente usadas, entre las que destacan:

Editores de Terminal

- Diakonos
- Jet
- Joe
- LE
- Mined
- Nano
- Nice Editor (NE)
- Pico
- Setedit
- Vim
- Fte

Editores Sencillos con Interfaz Gráfica

- Gedit
- SciTE
- JEdit
- NEdit
- MEdit

- KScope
- Editra
- Kate
- KWrite
- Leafpad
- Mousepad
- Anjunta
- TEA
- Pluma
- GVim
- Emacs

Editores Avanzados con Interfaz Gráfica

- Atom
- Bluefish
- BlueGriffon
- Brackets
- Geany
- Glade
- Google Web Designer
- KompoZer
- Light Table
- Notepadqq
- Scribes
- Sublime Text

Entornos de Programación Integrado (IDEs)

- Aptana
- Arduino IDE
- Android Studio
- CodeLite
- Code::Blocks
- Eclipse
- Gambas
- JetBrains Suite
- NetBeans
- Ninja-IDE
- Python IDLE
- PyDev
- Postman
- Qt Creator
- Simply Fortran
- Visual Studio Code
- Wing Python IDE
- Spyder
- PyCharm
- Jupyter
- Eric

Kit de Desarrollo de Software

- .Net Core SDK
- Android SDK
- Java JDK

Comparadores de texto y fuentes

- KDiff3
- Meld
- Diffuse
- DirDiff
- kompare
- Numdiff
- colordiff
- wdiff
- xxdiff
- tkdiff
- Ndiff

Otras Herramientas

- Alleyoop
- C2HTML
- Java2HTML
- Code2HTML
- c2html

- AutoDia
- txt2html
- html2text

Programas para control de versiones que permite desarrollo colaborativo de Software:

- Git https://git-scm.com/
- Mercurial https://www.mercurial-scm.org/
- Subversion https://subversion.apache.org/
- Perforce
- Bazaar
- CVS
- LibreSource
- Monotone
- SmartGit
- GitKraken
- Git Cola

Generadores automaticos de documentación que generan salida en PDF, HTML y XML para lenguajes como C++ y Java:

- Doxygen http://www.doxygen.org/
- JavaDoc

Formateador de código fuente para C, C++, Java y C#

- Astyle http://astyle.sourceforge.net

Lenguaje Unificado de Modelado (Unified Modeling Language)[25] forja un lenguaje de modelado visual común semántica y sintácticamente rico para la arquitectura, el diseño y la implementación de sistemas de Software complejos tanto en estructura como en comportamiento:

- UML https://www.uml.org/

Depuradores de programas

- ddd https://www.gnu.org/Software/ddd/
- gdb https://www.gnu.org/Software/gdb/
- kdbg http://www.kdbg.org/

Programas para rastrear errores en la manipulación de memoria y punteros desbordados

- Valgrind http://valgrind.org/
- DUMA http://duma.sourceforge.net/

Programas para hacer análisis de rendimiento[26]

- gprof https://sourceware.org/binutils/docs/gprof/
- Callgrind http://valgrind.org/docs/manual/cl-manual.html
- kCachegrind http://kCachegrind.sourceforge.net/html/Home.html
- time https://www.cyberciti.biz/faq/unix-Linux-time-command-examples-usage-syntax/

[25] Otras opciones son: UML Diagram Generation, Code Generation, Document Generation and Reporting, Scaling, Database Schema Generation, Entity Relationship Diagrams, Data Flow Datagrams, StarUML BOUML, EclipseUML, UML Modeller, Papyrus, Nclass, PlantUML, UMLet, NetBeansIDE, Open ModelSphere, gModeler, RISE, Oracle jdeveloper, Oracle SQL Developer, Dia, Kivio, ArgoUML, Xfig, etc.

[26] Otras opciones son: Splint, cppcheck, Rough Auditing Tool for Security, C y C++ Code Counter, CppNcss, Gnocchi, CUnit, CppUnit, OProfile, Intel VTune, Nemiver, Mudflap, etc.

En este apartado, solo tocaremos las más usadas, pero abunda la documentación de estas y otras importantes herramientas en línea de comandos (véase **??**). Iniciaremos por las de compilar[27] y depurar[28] programas compilables en C, C++, Fortran, entre otros.

3.7.1 ¿Qué es eso de ASCII, ISO-8859-1 y UTF-8?

Los tres estándares representan el esfuerzo informático por brindar un sistema de codificación que permita representar los caracteres que se usan en todos los idiomas. El primer esfuerzo lo hizo *ASCII* y fue para el idioma inglés (128 caracteres), luego ante su insuficiencia para representar otros caracteres como los latinos por ejemplo, nace *ISO-8859-1* (también llamado *LATIN-1* ó *ASCII* extendido) pero debido a que no podía representar caracteres de otros idiomas aparece el estándar Unicode (del cual es parte *UTF-8*).

Un buen detalle a saber es que mientras *ISO-8859-1* usa un byte para representar un carácter, no pasa lo mismo con *UTF-8* que puede usar hasta 4 bytes ya que es de longitud variable. Esto hace que una base de datos en *UTF-8* sea un poco más grande que una en ISO-8859-1. Esto sucede porque -por ejemplo- mientras *ISO-8859-1* usa un byte para representar la letra ñ, *UTF-8* usa dos bytes. Hay un tema más y es que muchas veces cuando vamos a migrar información nos encontramos con caracteres *ISO-8859-1* (los correspondientes a los números 147, 148, 149, 150, 151 y 133) que no pueden verse en un editor *UNIX/LINUX* pero si en un navegador *HTML*.

Unicode es un set de caracteres universal, es decir, un estándar en el que se definen todos los caracteres necesarios para la escritura de la mayoría de los idiomas hablados en la actualidad que se usan en la computadora. Su objetivo es ser, y, en gran medida, ya lo ha logrado, un superconjunto de

[27] Un compilador es un programa informático que traduce un programa que ha sido escrito en un lenguaje de programación a un lenguaje común, usualmente lenguaje de máquina, aunque también puede ser traducido a un código intermedio (bytecode) o a texto y que reúne diversos elementos o fragmentos en una misma unidad, este proceso de traducción se conoce como compilación.

[28] Un depurador (en inglés, debugger), es un programa usado para probar y depurar (eliminar) los errores del programa "objetivo". El código a ser examinado puede alternativamente estar corriendo en un simulador de conjunto de instrucciones (ISS), una técnica que permite gran potencia en su capacidad de detenerse cuando son encontradas condiciones específicas pero será típicamente más lento que ejecutando el código directamente en el apropiado (o el mismo) procesador.

todos los sets de caracteres que se hayan codificado. El texto que aparece en la computadora o en la Web se compone de caracteres. Los caracteres representan letras del abecedario, signos de puntuación y otros símbolos.

En el pasado, distintas organizaciones han recopilado diferentes sets de caracteres y han creado codificaciones específicas para ellos. Un set puede abarcar tan sólo los idiomas de Europa occidental con base en el latín (sin incluir países de la Unión Europea como Bulgaria o Grecia), otro set puede contemplar un idioma específico del Lejano Oriente (como el japonés), y otros pueden ser parte de distintos sets diseñados especialmente para representar otro idioma de algún lugar del mundo.

Lamentablemente, no es posible garantizar que su aplicación particular pueda soportar todas las codificaciones, ni que una determinada codificación pueda soportar todos sus requerimientos para la representación de un cierto idioma. Además, generalmente resulta imposible combinar distintas codificaciones en la misma página Web o en una base de datos, por lo que siempre es muy difícil soportar páginas plurilingües si se aplican enfoques "antiguos" cuando se trata de tareas de codificación.

El Consorcio Unicode proporciona un único y extenso set de caracteres que pretende incluir todos los caracteres necesarios para cualquier sistema de escritura del mundo, incluyendo sistemas ancestrales (como el cuneiforme, el gótico y los jeroglíficos egipcios). Hoy resulta fundamental para la arquitectura de la Web y de los sistemas operativos, y las principales aplicaciones y navegadores Web incluyen soporte para este elemento. En el Estándar Unicode también se describen las propiedades y algoritmos necesarios para trabajar con caracteres. Este enfoque facilita mucho el trabajo con sistemas o páginas plurilingües y responde mucho mejor a las necesidades del usuario que la mayoría de los sistemas de codificación tradicionales.

Sets de caracteres, sets de caracteres codificados y codificaciones un set de caracteres o repertorio comprende el grupo de caracteres que se utilizarían para una finalidad específica, ya sea los necesarios para el soporte de los idiomas de Europa Occidental en la computadora.

Un set de caracteres codificados es un grupo de caracteres en el que se ha asignado un número exclusivo a cada carácter. Las unidades de un set de caracteres codificados se conocen como puntos de código. El valor de un punto de código representa la ubicación de un carácter en el set de caracteres codificados. Por ejemplo, el punto de código para la letra *á* en el set de

caracteres codificados Unicode es *225* en notación decimal, o *E1* en notación hexadecimal.

La codificación de caracteres refleja la manera en la que el set de caracteres codificados se convierte a bytes para su procesamiento en la computadora. Además, en Unicode existen distintas formas de codificar el mismo carácter. Por ejemplo, la letra *á* se puede representar mediante dos bytes en una codificación y con cuatro bytes, en otra. Los formatos de codificación que se pueden usar con Unicode se denominan *UTF-8, UTF-16 y UTF-32*.

Por todo lo anterior, al programar es necesario tener en cuenta la codificación usada por el editor o IDE que se use para ello y que no todos los editores soportan las mismas codificaciones[29], además puede haber problemas de portabilidad en los archivos entre distintos sistemas operativos. En el código fuente (las instrucciones del programa) no se suele usar caracteres distintos al *ASCII*, pero en las cadenas de visualización o en la documentación es común el uso de caracteres acentuados, es aquí donde hay que tomar una decisión sobre el usar o no dichos caracteres, siempre y cuando el compilador los soporte.

Si siempre se usa el mismo editor y la misma plataforma de desarrollo, no hay razón para no usar caracteres extendidos como los acentos. Pero si se usarán múltiples sistemas operativos y no hay garantía de usar editores que soporten dichos caracteres, entonces existe la posibilidad de perder dichos caracteres o bien pueden generar errores al compilar los archivos por no ser soportados. Por ello una opción para evitar problemas es sólo usar caracteres *ASCII* o tener el cuidado de usar editores que no pierdan dichos caracteres.

En Linux, para verificar la codificación de un archivo se utiliza el comando *file –i* o *–mime*, este comando permite mostrar en pantalla el tipo de archivo y su codificación, usando:

```
$ file -i Car.java
```

El comando *iconv* es utilizado para realizar esta tarea de convertir el código de un texto a otro. La lógica para aplicar correctamente el commando *iconv* es la siguiente:

```
$ iconv options -f from-encoding -t to-encoding inputfile(s) -o
outputfile
```

[29] Dado que los archivos fuente se intercambian entre usuarios y es común el uso de diferentes sistemas operativos, la conversiones de los caracteres entre diferentes formatos puede ser causa de problemas de codificación, perdiéndose dichos caracteres en la conversión.

así, *-f* o *–from-code* significa la entrada de la codificación, y *–t* o *–to-encoding* especifica la salida de la misma. Para enumerar todos los juegos de caracteres podemos usar *-l* o *–list*:

```
$ iconv -l
```

Con todo esto en mente podemos proceder a explicar la codificación de *UTF-8* a *ASCII*. Primero hay que comenzar con conocer las codificaciones de los caracteres en el archivo y luego poder ver el contenido del mismo. Así, se podrán convertir todos los archivos a la codificación *ASCII*. Todo después de haber utilizado el comando *iconv*, para poder verificar lo que contiene la salida del archivo. Para eso hay que hacer lo siguiente:

```
$ file -i input.file
$ cat input.file
$ iconv -f ISO-8859-1 -t UTF-8//TRANSLIT input.file -o
out.file
$ cat out.file
$ file -i out.file
```

Cabe destacar que, si el comando *//IGNORE* se añade a *to-encoding*, los caracteres no pueden ser convertidos y un error se mostrará luego de la conversión. También, si el comando *//TRANSLIT* es añadido a *to-encoding* como en el ejemplo dado *(ASCII//TRANSLIT)*, los caracteres convertidos son transliterados, si es posible, como necesarios.

Esto implicaría que en este evento los caracteres no pueden ser representados como se desea, aunque pueden haber aproximaciones del mismo, inclusive dos. Por lo que, si un carácter no puede ser transliterado, no será reconocido como un objetivo para reemplazo y se mostrará la marca (?) en la salida del archivo.

Algunas veces es necesario convertir el archivo de UTF-8 a ASCII y lo hacemos mediante:

```
$ iconv -f UTF-8 -t ISO-8859-1 prog.c -o progMod.c
```

o mediante:

```
$ iconv -f UTF-8 -t ASCII//TRASLIT prog.c -o progMod.c
```

3.7.2 Uso de Espacios o Tabuladores en Fuentes

En muchos lenguajes de programación la identación es una forma opcional de hacer legible el código para el programador, en otros (como Python) es imprescindible (ya que de ella dependerá su estructura).

¿Qué es la identación? En un lenguaje informático, la identación es lo que a la sangría al lenguaje humano escrito (a nivel formal). Así como para el lenguaje formal, cuando uno redacta una carta, debe respetar ciertas sangrías, los lenguajes informáticos, pueden requerir una identación.

No todos los lenguajes de programación necesitan una identación, aunque sí se estila implementarla, a fin de otorgar mayor legibilidad al código fuente. Una identación depende del lenguaje y estilo de codificación usado, por ejemplo en C, C++ y Java se acostumbra usar tres espacios en blanco. En el caso de Python se suele usar 4 espacios en blanco o un tabulador, que indicará que las instrucciones identadas forman parte de una misma estructura de control.

Los programadores siempre han debatido entre el uso de espacios y tabulaciones para estructurar su código. Los espacios y las tabulaciones son utilizados por los programadores para estructurar el código de una forma determinada. La primera línea de código (sin espacio o tabulación) inicia un "bloque" de contenido. Si las sucesivas líneas de código forman parte de ese mismo bloque (encerrado entre corchetes) o forman nuevos subbloques, estas se van desplazando hacia la derecha para indicar esa subordinación. En caso de formar un bloque completamente nuevo, se mantiene en la misma posición que la línea inmediatamente anterior.

A nivel funcional, la diferencia entre el uso de espacios o tabulaciones es nula. Cuando el código pasa por el compilador antes de ser ejecutado, la máquina interpreta de igual forma ambos formatos. No obstante, sí existen diferencias técnicas que marcan la diferencia entre el uso de tabulaciones y espacios:

- Precisión. Una tabulación no es más que un conjunto de espacios agrupados. Por norma general, este conjunto suele ser de 8 caracteres, pero puede variar. ¿Qué quiere decir esto? Que cuando un mismo fichero de código se abre en dos máquinas diferentes, la apariencia del código puede ser diferente. En cambio, el uso de espacios no conlleva este problema: un espacio siempre ocupa el mismo "espacio" —valga la

redundancia— y asegura que el código se visualiza de la misma forma en todas las máquinas.

- Comodidad. En el caso de las tabulaciones, basta con pulsar la tecla de tabulación una única vez para estructurar correctamente el código. En el caso de los espacios, es necesario pulsar varias veces la misma tecla para lograr la estructura deseada.

- Almacenamiento. El uso de tabulaciones también reduce el tamaño del fichero final, mientras que el uso de espacios lo aumenta. Lo mismo sucedería con el uso de espacios en lugar de saltos de línea.

Entonces, ¿cuál es la más correcta? La realidad es que todo depende de las preferencias personales. Si necesitas optimizar el tamaño de los ficheros al máximo, el uso de espacios se convierte en un sacrilegio. Si, en cambio, tu código debe lucir exactamente igual en múltiples máquinas, el uso de espacios puede ser más conveniente para lograr esa homogeneidad.

Por suerte, existen múltiples editores en la actualidad que trabajan y facilitan la transición entre ambos sistemas. Asimismo, los equipos de desarrollo de software establecen en sus guidelines el uso de espacios o tabulaciones. De esta forma, se evitan conflictos entre los programadores de un mismo proyecto y se alcanza esa homogeneidad tan deseada.

El comando *expand* y *unexpand* (que vienen instalados en los paquetes *GNU Core*) permite convertir tabuladores en espacios y viceversa, según nuestras necesidades o gustos. Estos comandos sacan el resultado de stdin o de los archivos nombrados en la línea de comando. Utilizando la opción *-t* se pueden establecer una o más posiciones de tabulador.

Para ver si se usan espacios o tabuladores en un archivo fuente podemos usar el comando *cat* con la opción *-T* que nos mostrará los caracteres tabulador como *^I*, ejemplo:

```
$ cat -T archivo
```

Para convertir los espacios en tabuladores (un tabulador igual a 8 espacios) usamos:

```
$ unexpand progEsp.c
```

o redireccionando la salida usando:

```
$ unexpand progEsp.c > progTab.c
```

Para convertir los tabuladores en espacios (1 tabulador por ejemplo 4 caracteres) usamos:

```
$ expand -t 4 progTab.c
```

o redireccionando la salida usando:

```
$ expand -t 4 progTab.c > progEsp.c
```

También es posible buscar todos los archivos (digamos *.cpp) y cambiar los tabuladores por 4 espacios (para ello usamos el comando *sponge* que está contenido en el paquete *moreutils*), mediante:

```
$ find . -name '*.cpp' -type f -exec bash -c \
'expand -t 4 "$0" | sponge "$0"' {} \;
```

3.7.3 Comparar Contenido de Fuentes

Cuando se programa es común generar distintas versiones de un mismo archivo, en GNU/Linux se tiene varias herramientas para comparar y combinar cambios. En la línea de comandos el comando *diff* permite ver los cambios entre dos versiones de un archivo y el comando *merge* sirve para combinar cambios. Por otro lado *sdiff* nos permite ver las diferencias entre dos archivos y de forma interactiva combinar cambios.

Pese a que son poderosos estos comandos, en forma gráfica se puede obtener todo su potencial. Algunas de estas opciones son:

```
# apt install kdiff3 meld diffuse dirdiff kompare wdiff  \
numdiff colordiff xxdiff tkdiff ndiff
```

Estos permiten comparar dos o tres versiones de un archivo simultáneamente, y hacerlo con el contenido de una o más carpetas. Cada uno tiene la capacidad de mostrar los cambios y si se requiere hacer la combinación de ellos.

meld nos muestra gráficamente las diferencias entre dos archivos o también, entre todos los archivos de dos directorios utilizando distintos colores, y nos permite editar estos archivos desde el propio programa, actualizando dinámicamente las diferencias. El programa incluye filtros y distintas ayudas para hacer la edición más sencilla, como flechas al lado de los cambios para aplicar cambio en cualquiera de los archivos con un simple clic. Este programa se puede utilizar como un sencillo cliente de control de cambios para Git, CVS, Subversion, etc.

kdiff3 nos muestra gráficamente las diferencias entre tres archivos utilizando distintos colores, y nos permite editar estos archivos desde el propio programa, actualizando dinámicamente las diferencias. El programa incluye filtros y distintas ayudas para hacer la edición más sencilla, como flechas al lado de los cambios para aplicar cambio en cualquiera de los archivos con un simple clic.

3.7.4 Astyle

Para dar uniformidad a la codificación de los programas fuente, se puede usar un formateador automático de código, *Astyle* soporta una gran variedad de lenguajes y opciones, para instalar en Debian GNU/Linux usar:

```
# apt install astyle
```

para formatear los archivos de C, C++, C# usar:

```
$ astyle -s3 -p –style=allman –lineend=Linux *.?pp
```

para Java, una opción es

```
$ astyle -s3 -p –style=java –lineend=Linux *.java
```

Algunos estilos disponibles son:

style=allman style=java style=kr style=stroustrup
style=whitesmith style=vtk style=ratliff style=gnu
style=Linux style=horstmann style=1tbs style=google
style=mozilla style=pico style=lisp

más opciones en:

- http://astyle.sourceforge.net/astyle.html
- https://en.wikipedia.org/wiki/Programming_style
- https://en.wikipedia.org/wiki/Indent_style

3.7.5 Compilación y la Optimización del Ejecutable

Al programar es necesario revisar nuestro código por un compilador y los errores son inherentes al proceso de programación. Los errores de programación responden a diferentes tipos y pueden clasificarse dependiendo de la fase en que se presenten. Algunos tipos de errores son más difíciles de detectar y reparar que otros, veamos entonces:

- Errores de sintaxis
- Advertencias
- Errores de enlazado
- Errores de ejecución
- Errores de diseño

Errores de sintaxis son errores en el código fuente. Pueden deberse a palabras reservadas mal escritas, expresiones erróneas o incompletas, variables que no han sido declaradas, etc. Los errores de sintaxis se detectan en la fase de compilación. El compilador, además de generar el código objeto, nos dará una lista de errores de sintaxis. De hecho nos dará sólo una cosa o la otra, ya que si hay errores no es posible generar un código objeto.

Advertencias además de errores, el compilador puede dar también advertencias (Warnings). Las advertencias son errores, pero no lo suficientemente graves como para impedir la generación del código objeto. No obstante, es importante corregir estos errores la mayoría de las veces, ya que ante un aviso el compilador tiene que tomar decisiones, y estas no tienen porqué coincidir con lo que nosotros pretendemos hacer, ya se basan en las directivas que los creadores del compilador decidieron durante la creación del compilador. Por lo tanto, en ocasiones, ignorar las advertencias puede ocasionar que nuestro programa arroje resultados inesperados o erróneos.

Errores de enlazado el programa enlazador también puede encontrar errores. Normalmente se refieren a funciones que no están definidas en ninguno de los ficheros objetos ni en las bibliotecas. Puede que hayamos olvidado incluir alguna biblioteca, o algún fichero objeto, o puede que hayamos olvidado definir alguna función o variable, o lo hayamos hecho mal.

Errores de ejecución incluso después de obtener un fichero ejecutable, es posible que se produzcan errores durante la ejecución del código. En el caso de los errores de ejecución normalmente no obtendremos mensajes específicos de error o incluso puede que no obtengamos ningún error, sino que simplemente el programa terminará inesperadamente. Estos errores son más difíciles de detectar y corregir (pues se trata de la lógica como tal de nuestra aplicación). Existen herramientas auxiliares para buscar estos errores, son los llamados depuradores (Debuggers). Estos programas permiten detener la ejecución de nuestros programas, inspeccionar variables y ejecutar nuestro programa paso a paso (instrucción a instrucción). Esto resulta útil para detectar excepciones, errores sutiles, y fallos que se presentan dependiendo de circunstancias distintas. Generalmente los errores en tiempo de ejecución se dan por situaciones no consideradas en la aplicación, por ejemplo, que el usuario ingrese una letra en vez de un número y ésto no es controlado.

Errores de diseño finalmente los errores más difíciles de corregir y prevenir. Si nos hemos equivocado al diseñar nuestro algoritmo, no habrá ningún programa que nos pueda ayudar a corregirlos, pues es imposible que un programa pueda determinar qué es lo que tratamos de conseguir o un programa que realice aplicaciones cualquiera por nosotros. Contra estos errores sólo cabe practicar y pensar, realizar pruebas de escritorio, hacerle seguimiento y depuración a la aplicación hasta dar con el problema (una mala asignación, un valor inesperado, olvidar actualizar una variable, etc.), también es útil buscar un poco de ayuda de libros o en sitios y foros especializados.

Compilación y la Optimización del Ejecutable Para usar muchas de estas herramientas (en línea de comandos), primero debemos conocer cómo compilar fuentes[30], sin pérdida de generalidad trabajaremos en C++ solici-

[30] Compilador para C es gcc, para C++ es g++, para Fortran es f77 o f95, etc.

tando que el archivo ejecutable[31] tenga el nombre *ejemp*:

```
$ g++ *.cpp -o ejemp
```

Para ejecutar el programa ya compilado:

```
$ ./ejemp
```

Para compilar y ver todos los avisos usar:

```
$ g++ -pedantic -Wall -Wextra -O *.cpp
```

o de forma alternativa:

```
$ g++ -Weffc++ *.cpp
```

Por otro lado, también podemos hacer una revisión estática del código, por ejemplo en C++ usamos:

```
$ cppcheck –enable=all *.?pp
```

mostrará los avisos de análisis estático del código indicado.

Para conocer el tiempo de ejecución[32] de un programa, podemos usar el comando básico *time*, mediante:

```
$ time ejecutable
```

que entregará información del tipo:

```
$ time ls

real    0m0.004s
user    0m0.001s
sys     0m0.004s
```

[31] Un archivo ejecutable es tradicionalmente un archivo binario con instrucciones en código de máquina cuyo contenido se interpreta por el ordenador como un programa. Además, suele contener llamadas a funciones específicas de un sistema operativo.

[32] El tiempo total de ejecución de un programa (tiempo real) es la suma del tiempo de ejecución del programa del usuario (tiempo de usuario) más el tiempo de ejecución del sistema necesario para soportar la ejecución (tiempo de sistema).

Pero podemos instalar una versión optimizada de este comando que proporciona información adicional, para ello instalar:

```
# apt install time
```

y su ejecución mediante:

```
$ /usr/bin/time ejecutable
```

por ejemplo para el comando *ls*, entrega una salida del tipo:

```
$ /usr/bin/time -v ls
   Command being timed: "ls"
   User time (seconds): 0.00
   System time (seconds): 0.00
   Percent of CPU this job got: 66%
   Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.00
   Average shared text size (kBytes): 0
   Average unshared data size (kBytes): 0
   Average stack size (kBytes): 0
   Average total size (kBytes): 0
   Maximum resident set size (kBytes): 2360
   Average resident set size (kBytes): 0
   Major (requiring I/O) page faults: 0
   Minor (reclaiming a frame) page faults: 110
   Voluntary context switches: 1
   Involuntary context switches: 1
   Swaps: 0
   File system inputs: 0
   File system outputs: 0
   Socket messages sent: 0
   Socket messages received: 0
   Signals delivered: 0
   Page size (Bytes): 4096
   Exit status: 0
```

Además, podemos compilar usando diversas optimizaciones[33] disponibles en todos los compiladores GNU de Linux, estas opciones de optimización están ordenadas, desde no optimizar, a la mejor optimización posible, estas son: –O0, O1, -O2, -O3, -O3 -march=native, -O3 -march=native -fito, -Ofast -march=native.

Para compilar y solicitar la optimización usamos:

```
$ g++ -O1 *.cpp
```

y para ejecutar el programa usamos:

```
$ ./a.out
```

El resultado de las optimizaciones dependen del programa y se puede ver que el rendimiento (tiempo de ejecución) mejora en varios órdenes de magnitud.

Por ejemplo en los siguientes test[34] se obtienen estos rendimientos:
Cryto++ v8.2:

O0 (95), -O2 (660.46), -O3 (712.01), -O3 -march=native (751.56), -O3 -march=native -fito (699.80), -Ofast -march=native (751.01)

LeelaChessZero:

O0 (18,300), -O2 (157,289), -O3 (142,198), -O3 -march=native (136,608), -O3 -march=native -fito (163,773), -Ofast -march=native (157,629)

Himeno Benchmark v3.0:

O0 (597.53), -O2 (4,150.11), -O3 (4,015.86), -O3 -march=native (4,771.42), -O3 -march=native -fito (4,774.03), -Ofast -march=native (5,065.07)

[33] La optimización de código es el conjunto de fases de un compilador que transforma un fragmento de código en otro fragmento con un comportamiento equivalente y se ejecuta de forma más eficiente, es decir, usando menos recursos de cálculo como memoria o tiempo de ejecución.

[34] https://www.phoronix.com/scan.php?page=news_item&px=GCC-10.1-Compiler-Optimizations

C-Ray v1.1:

O0 (113.58), -O2 (69.70), -O3 (38.00), -O3 -march=native (30.46), -O3 -march=native -fito (30.24), -Ofast -march=native (27.13)

Geometric Mean Of All Test Results:

O0 (222.36), -O2 (681.88), -O3 (709.76), -O3 -march=native (735.14), -O3 -march=native -fito (755.97), -Ofast -march=native (758.30)

3.7.6 Análisis de Rendimiento y Depuración

El comando *gprof* produce un perfil de ejecución de programas en C, C++, Pascal, FORTRAN77, etc. El efecto de las rutinas llamadas se incorpora en el perfil de cada llamador. Los datos del perfil se toman del fichero de perfil de grafos de llamada ('gmon.out' por omisión) que es creado por programas que se han compilado con la opción -pg de cc(1), pc(1), y f77(1). La opción -pg también enlaza al programa versiones de las rutinas de biblioteca que están compiladas para la perfilación. *Gprof* lee el fichero objeto dado (el predeterminado es 'a.out') y establece la relación entre su tabla de símbolos y el perfil de grafo de llamadas de 'gmon.out'. Si se especifica más de un fichero de perfil, la salida de gprof muestra la suma de la información de perfilado en los ficheros de perfil dados.

Gprof calcula la cantidad de tiempo empleado en cada rutina. Después, estos tiempos se propagan a lo largo de los vértices del grafo de llamadas. Se descubren los ciclos, y se hace que las llamadas dentro de un ciclo compartan el tiempo del ciclo. El primer listado muestra las funciones clasificadas de acuerdo al tiempo que representan incluyendo el tiempo de sus descendientes en su grafo de llamadas. Debajo de cada entrada de función se muestran sus hijos (directos) del grafo de llamadas, y cómo sus tiempos se propagan a esta función. Un despliegue similar sobre la función muestra cómo el tiempo de esta función y el de sus descendientes se propagan a sus padres (directos) del grafo de llamadas.

También se muestran los ciclos, con una entrada para el ciclo completo y un listado de los miembros del ciclo y sus contribuciones al tiempo y número de llamadas del ciclo. En segundo lugar, se da un perfil plano, similar al producido por *prof*. Este listado de los tiempos de ejecución totales, los números de llamadas, el tiempo en milisegundos que la llamada empleó en la propia rutina, y el tiempo en ms que la llamada empleó en la propia rutina

pero incluyendo sus descendientes. Finalmente, se proporciona un índice a los nombres de función.

Para obtener el análisis de rendimiento, hacemos:

```
$ g++ -g -pg -O0 *.cpp
$ ./a.out
$ gprof -c -z a.out gmon.out > sal.txt
```

el archivo *sal.txt* contiene el análisis de rendimiento detallado. Un ejemplo de esta salida es:

```
Flat profile:
Each sample counts as 0.01 seconds.
% cumulative self self total
time seconds seconds calls s/call s/call name
23.25    0.60    0.60  40656734     0.00     0.00 retorna(int, int)
14.85    0.98    0.38  27627674     0.00     0.00 retoaNumColu(int, int)
12.89    1.31    0.33  91126931     0.00     0.00 Vector::retorna(int)
10.94    1.59    0.28        31     0.01     0.03 ResJacobi::resuelve()
...
```

que permite conocer en que parte del código se consume más tiempo de ejecución.

Depuración con ddd un depurador (en inglés: Debugger) es un programa usado para probar y depurar (eliminar) los errores de otros programas (el programa "objetivo"). El código a ser examinado puede alternativamente estar corriendo en un simulador de conjunto de instrucciones (ISS), una técnica que permite gran potencia en su capacidad de detenerse cuando son encontradas condiciones específicas pero será típicamente algo más lento que ejecutando el código directamente en el apropiado (o el mismo) procesador. Algunos depuradores ofrecen dos modos de operación - la simulación parcial o completa, para limitar este impacto.

Si es un depurador de nivel de fuente o depurador simbólico, comúnmente ahora visto en entornos de desarrollo integrados, cuando el programa "se estrella" o alcanza una condición predefinida, la depuración típicamente muestra la posición en el código original. Si es un depurador de bajo nivel o un depurador de lenguaje de máquina, muestra la línea en el fuente desensamblado (a menos que también tenga acceso en línea al código fuente

original y pueda exhibir la sección apropiada del código del ensamblador o del compilador). Un "estrellamiento" sucede cuando el programa no puede continuar normalmente debido a un error de programación. Por ejemplo, el programa pudo haber intentado usar una instrucción no disponible en la versión actual del CPU o haber intentado tener acceso a memoria protegida o no disponible.

Típicamente, los depuradores también ofrecen funciones más sofisticadas tales como correr un programa paso a paso (un paso o animación del programa), parar el programa (Breaking), es decir, pausar el programa para examinar el estado actual en cierto evento o instrucción especificada por medio de un Breakpoint, y el seguimiento de valores de algunas variables. Algunos depuradores tienen la capacidad de modificar el estado del programa mientras que está corriendo, en vez de simplemente observarlo. También es posible continuar la ejecución en una posición diferente en el programa pasando un estrellamiento o error lógico.

La importancia de un buen depurador no puede ser exagerada. De hecho, la existencia y la calidad de tal herramienta para un lenguaje y una plataforma dadas a menudo puede ser el factor de decisión en su uso, incluso si otro lenguaje/plataforma es más adecuado para la tarea.

Para hacer depuración del código mediante el depurador gráfico *ddd* usar:

```
$ g++ -g -O0 *.cpp
$ ddd ./a.out
```

Puede usarse también los depuradores *xxgdb, gdb, kdbg, nevimer, lldb* cada uno tiene sus pros y contras, depende del usuario cual es el más adecuado para usar.

Depuración con valgrind es un conjunto de herramientas libres que ayuda en la depuración de problemas de memoria y rendimiento de programas.

La herramienta más usada es *Memcheck*. *Memcheck* introduce código de instrumentación en el programa a depurar, lo que le permite realizar un seguimiento del uso de la memoria y detectar los siguientes problemas:

- Uso de memoria no inicializada.
- Lectura/escritura de memoria que ha sido previamente liberada.

- Lectura/escritura fuera de los límites de bloques de memoria dinámica.
- Fugas de memoria.
- Otros.

El precio a pagar es una notable pérdida de rendimiento; los programas se ejecutan entre cinco y veinte veces más lento al usar Valgrind, y su consumo de memoria es mucho mayor. Por ello normalmente no siempre se ejecuta un programa en desarrollo usando Valgrind, sino que se usa en situaciones concretas cuando se está buscando un error determinado se trata de verificar que no haya errores ocultos como los que Memcheck puede detectar.

Valgrind incluye además otras herramientas:

- Addrcheck, versión ligera de Memcheck que se ejecuta más rápido y requiere menos memoria pero que detecta menos tipos de errores.
- Massif, mide el rendimiento del montículo (heap).
- Helgrind, herramienta de detección de condiciones de carrera (race conditions) en código multihilo.
- Cachegrind, mide el rendimiento de la Caché durante la ejecución, de acuerdo a sus características (capacidad, tamaño del bloque de datos, grado de asociatividad, etc.).

Para el rastreo de problemas con la manipulación de memoria y punteros desbordados usamos:

```
$ g++ -g -O0 *.cpp
$ valgrind –tool=memcheck –leak-check=yes –show-reachable=yes
./a.out
```

o analizar la salida usando *kCachegrind*:

```
$ valgrind –tool=callgrind ./a.out
$ kCachegrind profile.callgrind
```

Existen varios paquetes de modo gráfico para *valgrind*, uno de ellos es *alleyoop* y se usa:

```
$ alleyoop ./a.out -v –arg1=foo
```

otro es *kCachegrind*, podemos ver más opciones en:

- http://valgrind.org/
- http://alleyoop.sourceforge.net/usage.html
- http://kCachegrind.sourceforge.net/html/Home.html

3.7.7 Mejora del Rendimiento en Python

Python es un lenguaje interpretado, pero es posible compilar el código para generar Byte Code para el intérprete (no aumenta la velocidad de ejecución). Si se necesita crear un archivo *.pyc* para un módulo que no se importa, se puede usar los módulos *py_compile* y *compile_all* desde el intérprete de Python.

El módulo py_compile puede compilar manualmente cualquier módulo. Una forma de usar la función *py_compile.compile* en ese módulo de forma interactiva es:

```
>>> import py_compile
>>> py_compile.compile('modulo.py')
```

esto escribirá el archivo *modulo*.pyc.

En la línea de comando de Linux es posible compilar todos los archivos en un directorio utilizando el módulo compileall, para ello usar:

```
$ python -m compileall *.py
```

y ejecutar mediante:

```
$ python modulo.pyc
```

También es posible hacer ligera optimización del código generado mediante:

```
$ python –O -m compileall *.py
```

esto generará código Bytecode con extensión *.pyo*, y ejecutar mediante:

```
$ python modulo.pyo
```

Python es un lenguaje razonablemente rápido, pero no es tan rápido como los programas compilados de C o Fortran. Eso es porque se interpreta *CPython*, la implementación estándar. Para ser más precisos, su código de Python se compila en un código de *Bytes* que luego se interpreta. Eso es bueno para aprender, ya que puede ejecutar el código en el *REPL* de Python y ver los resultados de inmediato en lugar de tener que compilar y ejecutar. Pero como los programas de Python no son tan rápidos, los desarrolladores han creado varios compiladores de Python[35] a lo largo de los años, incluidos[36] Numba, *Nuitka, PyPy, Cython, cx_FreezeIron, Pythran, Jython entre otros.*

Comparando Compiladores de Python Alguien ya ha hecho el trabajo de crear un punto de referencia de Python. Opté por *PyStone*, una traducción de un programa en C de Guido van Rossum, el creador de Python (el programa en C era en sí mismo una traducción de un programa *Ada*). Encontré una versión convertida por el desarrollador Christopher Arndt en *Github* que era capaz de probar Python 3. Para dar un sentido de perspectiva, aquí está el rendimiento de *CPython* (es decir, Python estándar) con *Pystone*:

```
Python 2.7.15Rc1 2: 272.647 pystones / second.
Python 3.6.5: 175,817
```

Como puede ver, hay una gran diferencia entre Python 2 y 3 (cuanto más Pystones por segundo, mejor). En los siguientes desgloses, todos los compiladores de Python se compararon con Python 3.

Nuitka Aunque puede seguir las instrucciones en la página de descarga, lo siguiente en Debian GNU/Linux funcionó bien para mí:

```
$ apt install nuitka
```

[35] El rendimiento rápido no es la única razón para compilar; Posiblemente la mayor desventaja de los lenguajes de *Scripting* como Python es que se proporciona de manera implícita su código fuente a los usuarios finales.

[36] Si está interesado en los compiladores de Python en general, tenga en cuenta que hay mucho debate y controversia sobre los "mejores" compiladores y la rapidez general del lenguaje.

adicionalmente Nuitka también puede usar otro compilador de C (además del gcc), así que descargué clang. Puedes instalarlo con esto:

```
$ apt install clang
```

De forma predeterminada, *Nuitka* usa *gcc*, pero un parámetro te permite usar el *clang*, así que lo probé con ambos. El compilador clang es parte de la familia llvm, y está pensado como un reemplazo moderno para *gcc*. Compilar pystone.py con *gcc* fue tan simple como esto (primera línea), o con *clang* (segunda línea), y con la optimización del tiempo de enlace para *gcc* (tercera línea):

```
$ nuitka pystone.py
$ nuitka pystone.py –clang
$ nuitka pystone.py –lto
```

Después de compilar, lo que tomó aproximadamente 10 segundos, ejecuté el pystone.exe desde la terminal con:

```
$ ./pystone.exe 500000
```

Hice 500,000 pases:

Tamaño Ejecución pystones / seg.

1. 223.176 Kb 597,000
2. 195,424 Kb 610,000
3. 194.2 kb 600,000

Estos fueron los promedios de más de 5 corridas. Había cerrado tantos procesos como pude, pero tómo los tiempos con un poco de variación porque había un +/- 5% en los valores de tiempo.

PyPy Guido van Rossum dijo una vez: "Si quieres que tu código se ejecute más rápido, probablemente debas usar PyPy". Para instalarlo en Debian GNU/Linux usar:

```
$ apt install pypy
```

Entonces lo corrí así:

$ pypy pystone.py

El resultado fue una asombrosa cantidad de 1,776,001 pystones por segundo, casi tres veces más rápido que *Nuitka*.

PyPy usa un compilador justo a tiempo y hace algunas cosas muy inteligentes para alcanzar su velocidad. De acuerdo con los puntos de referencia reportados, es 7.6 veces más rápido que el *CPython* en promedio. Puedo creer eso fácilmente. La única (leve) desventaja es que siempre está un poco por detrás de las versiones de Python (es decir, hasta 2.7.13 (no 2.7.15) y 3.5.3 (no 3.6.5)). Producir un exe requiere un poco de trabajo. Tienes que escribir tu Python en un subconjunto llamado *RPython*.

Cython no es solo un compilador para Python; es para un superconjunto de Python que admite la interoperabilidad con C / C++. CPython está escrito en C, por lo que es un lenguaje que generalmente se combina bien con Python.

Configurar las cosas con *Cython* es un poco complicado. No es como *Nuitka*, que acaba de salir de la caja. Primero, debes comenzar con un archivo de Python con una extensión .pyx; ejecuta *Cython* para crear un archivo pystone.c a partir de eso:

$ cython pystone.pyx –embed

No omita el parámetro *-embed*. Se agrega en *main* y eso es necesario. A continuación, compila pystone.c con esta hermosa línea:

$ gcc $ (python3-config –includes) pystone.c -lpython3.6m -o pystone.exe

Si recibe algún error, como "no se puede encontrar la versión -lpython", podría ser el resultado de su versión de Python. Para ver qué versión está instalada, ejecute este comando:

$ pkg-config –cflags python3

Después de todo eso, *Cython* solo dio 228,527 pystones / sec. Sin embargo, *Cython* necesita que hagas un poco de trabajo especificando los tipos de variables. Python es un lenguaje dinámico, por lo que no se especifican los tipos; *Cython* utiliza la compilación estática y el uso de variables de tipo C le permite producir un código mucho mejor optimizado. (La documentación es bastante extensa y requiere lectura).

Tamaño Ejecución pystones / seg.

1. 219.552 Kb 228.527

cx_freeze es un conjunto de Scripts y módulos para "congelar" Scripts de Python en ejecutables, y se pueden encontrar en *Github*.
Lo instalé y creé una carpeta congelada para administrar cosas en:

$ pip3 install cx_Freeze –upgrade

Un problema que encontré con el Script de instalación fue un error que falta "lz". Necesitas tener instalado zlib; ejecuta esto para instalarlo:

$ apt install zlib1g-dev

Después de eso, el comando *cx_Freeze* tomó el Script *pystone.py* y creó una carpeta dist que contenía una carpeta *lib*, un archivo *lib* de 5MB y el archivo de aplicación pystone:

$ cxfreeze pystone.py –target-dir dist

Tamaño Ejecución pystones / seg.

1. 10,216 174,822

No es el rendimiento más rápido, porque es la misma velocidad que *CPython*. (La congelación de Python implica enviar su aplicación en un solo archivo (o carpeta) con los elementos Python necesarios, en lugar de compilar; significa que el destino no requiere Python).

Numba Este es un compilador "justo a tiempo" para Python que optimiza el código que se usa en algoritmos numéricos como son en las matrices, bucles y funciones de NumPy (también da soporte a *Threading*, vectorización *SIMD* y aceleración por *GPUs*: *Nvidia CUDA, AMD ROC*). La forma más común de usar Numba es a través de su colección de decoradores que se pueden aplicar a sus funciones para indicar a Numba que las compile usando el estándar *LLVM*. Cuando se realiza una llamada a una función decorada de Numba, se compila en el código de máquina "justo a tiempo" para su ejecución y todo o parte de su código puede ejecutarse posteriormente a la velocidad de código de máquina nativo. Numba también trabaja bien con Jupiter notebook para computación interactiva y con ejecución distribuida como *Dask* y *Spark*.

Se puede instalar en Debian GNU/Linux mediante:

```
$ apt install python3-numba
```

y se puede descargar mediante *CONDA* paquete de *Anaconda*, usando:

```
$ conda install numba
```

o mediante *PIP* usando:

```
$ pip install numba
```

Dando mejores resultados en la ejecución de múltiples pruebas que *PyPy*, pero no en todos los casos. Por ello, la recomendación es evaluar el rendimiento mediante pruebas en cada caso particular.

Conclusión Una buena opción es *PyPy* por el rendimiento obtenido en código general (y dependiendo del código en cuestión Numba puede ser mejor que *PyPy* en aplicaciones de cómputo científico), la compilación fue muy rápida y produjo los resultados en menos de un segundo después de presionar la tecla *RETURN*. Si requieres un ejecutable, sin embargo, te recomiendo Nuitka; fue una compilación sin complicaciones y se ejecuta más rápido que CPython. Experimenta con estos compiladores de Python y vea cuál funciona mejor para tus necesidades particulares.

3.7.8 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.

¿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.

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[37] 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:

3.7.9 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.

[37] 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
```

3.7.10 Otras opciones

Herramientas para convertir código fuentes en HTML, usando:

```
$ code2html Fuente Salida.html
```

o

```
$ c2html Fuente
```

Para java, usamos:

```
$ java2html Fuentes
```

También podemos convertir código fuente en PDF, usando:

```
$ nl test.cpp | a2ps -1 -l100 -otest.ps ; ps2pdf test.ps
```

el primer comando numera las líneas del fuente, el segundo comando generá del fuente numerado un .PS y el último comando convierte .PS a .PDF

Si se tiene que ejecutar múltiples programas que son independientes uno de otro se puede usar el programa *parallel* para correr N (número de cores del equipo) de ellos al mismo tiempo, por ejemplo si tenemos un archivo Bash con el nombre *mi-Bash* y cuyo contenido es:

```
./a.out 4 5 4 > a1.txt
./a.out 4 5 3 > a2.txt
./a.out 4 5 6 > a3.txt
./a.out 4 5 4 > a4.txt
./a.out 3 5 4 > a5.txt
./a.out 4 6 4 > a6.txt
```

entonces podemos ejecutarlo usando *parallel*, el programa usará el número máximo de cores disponibles:

```
$ parallel -v < mi-Bash
```

si solo se desea usar una determinada cantidad de cores (por ejemplo 3) entonces usamos:

```
$ parallel -v -j 3 < mi-Bash
```

3.8 Programando Desde la Nube

Existen diferentes servicios Web[38] que permiten editar, compilar y ejecutar código de diversos lenguajes y paquetes desde el navegador, esto en aras de que los estudiantes y profesores que cuenten con algún sistema de acceso a red y un navegador puedan programar en los más diversos lenguajes, IDEs y terminales sin hacer instalación alguna en su equipo de cómputo, tableta o teléfono celular.

Algunos ejemplos de estos servicios son:

- https://www.jdoodle.com/
- https://repl.it/
- http://browxy.com
- https://jupyter.org/try
- https://tio.run/
- https://www.compilejava.net/
- http://codepad.org/
- https://code.hackerearth.com/
- https://www.remoteinterview.io/online-c-compiler
- https://ideone.com/
- https://hackide.herokuapp.com/
- https://www.codechef.com/ide
- http://cpp.sh/

[38] Cuando se trabaja desde la Web es recomendable usar el modo Privado o Incógnito para no guardar el historial de navegación, información introducida en los formularios y borrar al cerrar el navegador los datos de los sitios visitados. Pero recuerda que los sitios Web que visitamos sí guardan información de nuestra visita, nuestro proveedor de Internet también guarda constancia de nuestra visita y si descargamos algo, esto no se borra al igual que el historial de descargas, además de las marcas de páginas o favoritos se conservarán al cerrar el navegador.

- https://codebunk.com/
- https://rextester.com/
- https://www.tutorialspoint.com/codingground.htm
- https://www.compileonline.com
- http://pythonfiddle.com/
- https://trinket.io/python
- https://www.pythonanywhere.com/try-ipython/
- https://www.rollapp.com/
- https://godbolt.org/
- https://www.codiva.io/
- https://paiza.io/en
- https://wandbox.org/
- http://coliru.stacked-crooked.com/
- http://quick-bench.com/
- https://cppinsights.io/
- https://ideone.com/
- http://cpp.sh/
- https://ide.geeksforgeeks.org/
- https://www.codechef.com/ide
- https://visualstudio.microsoft.com/services/visual-studio-online/

`Usando Editores Colaborativos`

La escritura colaborativa es una escritura de códigos de programación en la Web hecha por más de una persona simultáneamente.

Algunos ejemplos de estos servicios son:

- http://collabedit.com (edita código, tiene chat, no compila)
- https://gitduck.com/
- https://codeshare.io/
- https://www.tutorialspoint.com/codingground.htm
- http://ideone.com
- https://codebunk.com
- https://visualstudio.microsoft.com/services/visual-studio-online/
- https://ace.c9.io/build/kitchen-sink.html
- https://coderpad.io/
- https://peerpad.net/
- https://aws.amazon.com/cloud9/
- https://codeanywhere.com/
- https://stekpad.com/home/

Algunas de las terminales soportados son:

CentOS, IPython, Lua, MemCached, Mongo DB, MySQL, Node.js, Numpy, Oracle, Octave, PowerShell, PHP, R Programming, Redis, Ruby, SciPy, SymPy, etc.

Algunos de los IDEs soportados son:

Ada (GNAT), Algol68, Angular JS, Assembly, AsciiDoc, AWK, Bash Shell, Befunge, Bootstrap, Brainf**k, C, CSS3, Chipmunk BASIC, Clojure, Cobol, CoffeeScript, ColdFusion, C99 Strict, C++, C++ 0x, C++ 11, C#, Dart, D Programming Language, Embedded C, Erlang, Elixir, Factor, Fantom, Falcon, Fortran-95, Forth,F#, Free Basic, Groovy, GO, Haxe, Haskell, HTML, ilasm, Intercal, Icon, Java, Java 8, Java MySQL, JavaScript, JSP, JQuery, Julia, Korn Shell (ksh), Latex, Lisp, LOLCODE, Lua, Matlab/Octave, Malbolge, Markdown, MathML, Mozart-Oz, Nimrod, Node.JS, Objective-C, OCaml, Pascal, PARI/GP, Pawn, Perl, Perl MySQL, PHP, PHP MySQL, WebView, Pike, Processing.js, p5.js, Prolog, Python-2, Python-3, Python MySQL, Jupyter Notebook, Rexx, reStructure, Ruby, Rust, Scala, R Programming, Scheme, Smalltalk,SML/NJ, Simula, SQLite SQL, Tcl, TeX, Unlambda, VB.NET, Verilog, Whitespace, Ya Basic, etc.

Google Colaboratory Integrante de la G Suite for Education de Google permite a los usuarios que pertenezcan a esta Suite (como gran parte de los estudiantes de la UNAM) tener acceso desde el navegador para escribir y ejecutar código de Python (Jupyter), es posible elegir correr nuestro Notebook en una CPU, GPU o en una TPU de forma gratuita. Tiene algunas restricciones, como por ejemplo que una sesión dura 12 hrs, pasado ese tiempo se limpia nuestro ambiente y perdemos las variables y archivos que tengamos almacenados allí.

Es conveniente para principiantes que requieran experimentar con Machine Learning y Deep Learning pero sin recurrir en costos de procesamiento Cloud. Además el ambiente de trabajo ya viene con muchas librerías instaladas y listas para utilizar (como por ejemplo *Tensorflow, Sclkit-learn, Pytorch, Keras* y *OpenCV*), ahorrándonos el trabajo de configurar nuestro ambiente de trabajo. Podemos importar nuestros archivos y datos desde *Google Drive, GitHub*, etc.

Más información sobre Google Colaboratory en:

https://colab.research.google.com/notebooks/intro.ipynb

4 Programación y Lenguajes de Programación

En la actualidad, la mayoría de nosotros utilizamos computadoras permanentemente: para mandar correos electrónicos, navegar por Internet, chatear, jugar, escribir textos. Las computadoras se usan para actividades tan disímiles como predecir las condiciones meteorológicas de la próxima semana, guardar historias clínicas, diseñar aviones, llevar la contabilidad de las empresas o controlar una fábrica. Y lo interesante aquí (y lo que hace apasionante a la programación) es que el mismo aparato sirve para realizar todas estas actividades: uno no cambia de computadora cuando se cansa de chatear y quiere jugar al solitario.

Muchos definen una computadora moderna como "una máquina que almacena y manipula información bajo el control de un programa que puede cambiar". Aparecen acá dos conceptos que son claves: por un lado se habla de una máquina que almacena información, y por el otro lado, esta máquina está controlada por un programa que puede cambiar.

Una calculadora sencilla, de esas que sólo tienen 10 teclas para los dígitos, una tecla para cada una de las 4 operaciones, un signo igual, encendido y limpiar, también es una máquina que almacena información y que está controlada por un programa. Pero lo que diferencia a esta calculadora de una computadora es que en la calculadora el programa no puede cambiar.

Un programa de computadora es un conjunto de instrucciones paso a paso que le indican a una computadora cómo realizar una tarea dada, y en cada momento uno puede elegir ejecutar un programa de acuerdo a la tarea que quiere realizar.

Las instrucciones se deben escribir en un lenguaje que nuestra computadora entienda. Los lenguajes de programación son lenguajes diseñados especialmente para dar órdenes a una computadora, de manera exacta y no ambigua. Sería muy agradable poder darle las órdenes a la computadora en castellano, pero el problema del castellano, y de las lenguas habladas en general, es su ambigüedad.

El Mito de la Máquina Todopoderosa Muchas veces la gente se imagina que con la computadora se puede hacer cualquier cosa, que no hay tareas imposibles de realizar. Más aún, se imaginan que si bien hubo cosas que eran imposibles de realizar hace 50 años, ya no lo son más, o no lo serán dentro de algunos años, cuando las computadoras crezcan en poder (memoria, velocidad, unidades de almacenamiento), y la computadora se

vuelva una máquina todopoderosa.

Sin embargo eso no es así: existen algunos problemas, llamados no computables que nunca podrán ser resueltos por una computadora digital, por más poderosa que ésta sea. La computabilidad es la rama de la computación que se ocupa de estudiar qué tareas son computables y qué tareas no lo son. De la mano del mito anterior, viene el mito del lenguaje todopoderoso: hay problemas que son no computables porque en realidad se utiliza algún lenguaje que no es el apropiado.

En realidad todas las computadoras pueden resolver los mismos problemas, y eso es independiente del lenguaje de programación que se use. Las soluciones a los problemas computables se pueden escribir en cualquier lenguaje de programación. Eso no significa que no haya lenguajes más adecuados que otros para la resolución de determinados problemas, pero la adecuación está relacionada con temas tales como la elegancia, la velocidad, la facilidad para describir un problema de manera simple, etc., nunca con la capacidad de resolución.

Los problemas no computables no son los únicos escollos que se le presentan a la computación. Hay otros problemas que si bien son computables demandan para su resolución un esfuerzo enorme en tiempo y en memoria. Estos problemas se llaman intratables. El análisis de algoritmos se ocupa de separar los problemas tratables de los intratables, encontrar la solución más barata para resolver un problema dado, y en el caso de los intratables, resolverlos de manera aproximada: no encontramos la verdadera solución porque no nos alcanzan los recursos para eso, pero encontramos una solución bastante buena y que nos insume muchos menos recursos (el orden de las respuestas de Google a una búsqueda es un buen ejemplo de una solución aproximada pero no necesariamente óptima).

En este trabajo veremos problemas no sólo computables sino también tratables. Y aprenderemos a medir los recursos que nos demanda una solución, y empezaremos a buscar la solución menos demandante en cada caso particular. Por ejemplo:

> Dado un número N se quiere calcular N 33. Una solución posible, por supuesto, es hacer el producto N×N×. . .×N , que involucra 32 multiplicaciones. Otra solución, mucho más eficiente es: Calcular N×N . Al resultado anterior multiplicarlo por sí mismo con lo cual ya disponemos de N^4. Al resultado anterior multiplicarlo por sí mismo con lo cual ya disponemos de N^8.

> Al resultado anterior multiplicarlo por sí mismo con lo cual ya disponemos de N^{16}. Al resultado anterior multiplicarlo por sí mismo con lo cual ya disponemos de N^{32}. Al resultado anterior multiplicarlo por N con lo cual conseguimos el resultado deseado con sólo 6 multiplicaciones. Cada una de estas soluciones representa un algoritmo, es decir un método de cálculo, diferente. Para un mismo problema puede haber algoritmos diferentes que lo resuelven, cada uno con un costo distinto en términos de recursos computacionales involucrados.

Hay muchas aplicaciones a las herramientas computacionales que se han programado y podemos programar, pero nos interesan especialmente aquellas que permitan resolver problemas concomitantes en Ciencia e Ingeniería. Muchas de estas aplicaciones caen en lo que comúnmente se llama cómputo científico. La computación científica es el campo de estudio relacionado con la construcción de modelos matemáticos, técnicas numéricas para resolver problemas científicos y de ingeniería; y su respectiva implementación computacional.

Este campo es distinto a las ciencias de la computación y el procesamiento de información, también es diferente a la teoría y experimentación, que son las formas tradicionales de la ciencia y la ingeniería. El enfoque de la computación científica es para ganar entendimiento, principalmente a través del análisis de modelos matemáticos implementados en computadoras.

Los programas de aplicación de la computación científica a menudo modelan cambios en las condiciones del mundo real, tales como el tiempo atmosférico, el flujo de aire alrededor de un avión, el movimiento de las estrellas en una galaxia, el comportamiento de un dispositivo explosivo, entre otros. Estos programas deberían crear una 'malla lógica' en la memoria de la computadora, donde cada ítem corresponda a un área en el espacio y contenga información acerca del espacio relevante para el modelo. Por ejemplo, en modelos para el tiempo atmosférico, cada ítem podría ser un kilómetro cuadrado, con la altitud del suelo, dirección actual del viento, humedad ambiental, temperatura, presión, etc. El programa debería calcular el siguiente estado probable basado en el estado actual, simulado en medidas de tiempo, resolviendo ecuaciones que describen cómo operan los sistemas mediante el uso de un algoritmo[39], y repetir el proceso para calcular el siguiente estado. Este código o programa se escribe en un lenguaje de programación que sigue

[39] Un algoritmo es un conjunto preescrito de instrucciones o reglas bien definidas, orde-

algún paradigma de programación[40], que posteriormente puede ser ejecutado por una unidad central de procesamiento -computadora-.

Así, una parte importante de la programación es el hecho de conocer y usar uno o más paradigmas de programación. Entonces iniciemos delineando lo que es un paradigma de programación. Los paradigmas difieren unos de otros, en los conceptos y la forma de abstraer los elementos involucrados en un problema, así como en los pasos que integran la solución del problema, en otras palabras, el cómputo. Tiene una estrecha relación con la formalización de determinados lenguajes al momento de definirlos -es el estilo de programación empleado-.

4.1 Paradigmas de Programación

Los Tipos más comunes de paradigmas de programación son[41]:

- Programación imperativa o por procedimientos: es el más usado en general, se basa en dar instrucciones a la computadora de como hacer las cosas en forma de algoritmos. La programación imperativa es la más usada y la más antigua, el ejemplo principal es el lenguaje de máquina. Ejemplos de lenguajes puros de este paradigma serían C, BASIC o Pascal.

- Programación orientada a objetos: esta basado en el imperativo, pero encapsula elementos denominados objetos que incluyen tanto variables como funciones. Esta representado por C++, C#, Java o Python entre otros, pero el más representativo sería el Smalltalk que esta completamente orientado a objetos.

- Programación dinámica: esta definido como el proceso de romper problemas en partes pequeñas para analizarlos y resolverlos de forma lo más cercana al óptimo, busca resolver problemas en $O(n)$ sin usar por tanto métodos recursivos. Este paradigma está más basado en el

nadas y finitas que permiten llevar a cabo una actividad mediante pasos sucesivos que no generen dudas a quien deba hacer dicha actividad. Dados un estado inicial y una entrada, siguiendo los pasos sucesivos se llega a un estado final y se obtiene una solución.

[40] Este representa un enfoque particular o filosofía para diseñar soluciones e implementarlas en algún lenguaje de programación.

[41] En general la mayoría son variantes de los dos tipos principales: imperativa y declarativa.

modo de realizar los algoritmos, por lo que se puede usar con cualquier lenguaje imperativo como C, C++, Java o Python.

- Programación dirigida por eventos: la programación dirigida por eventos es un paradigma de programación en el que tanto la estructura como la ejecución de los programas van determinados por los sucesos que ocurran en el sistema, definidos por el usuario o que ellos mismos diseñen, por ejemplo en las interfaces gráficas de usuarios o en la Web.

- Programación declarativa: esta basado en describir el problema declarando propiedades y reglas que deben cumplirse, en lugar de instrucciones. Hay lenguajes para la programación funcional, la programación lógica, o la combinación lógico-funcional. Unos de los primeros lenguajes funcionales fueron Lisp y Prolog.

- Programación funcional: basada en la definición los predicados y es de corte más matemático, está representado por Scheme (una variante de Lisp) o Haskell. Python también representa este paradigma.

- Programación lógica: basado en la definición de relaciones lógicas, está representado por Prolog.

- Programación con restricciones: similar a la lógica usando ecuaciones. Casi todos los lenguajes son variantes del Prolog.

- Programación multiparadigma: es el uso de dos o más paradigmas dentro de un programa. El lenguaje Lisp se considera multiparadigma. Al igual que Python, que es orientado a objetos, reflexivo, imperativo y funcional.

- Programación reactiva: este paradigma se basa en la declaración de una serie de objetos emisores de eventos asíncronos y otra serie de objetos que se «suscriben» a los primeros -es decir, quedan a la escucha de la emisión de eventos de estos- y "reaccionan" a los valores que reciben. Es muy común usar la librería Rx de Microsoft (Acrónimo de Reactive Extensions), disponible para múltiples lenguajes de programación.

El paradigma de programación más utilizado en la actualidad, es el de «orientación a objetos». El núcleo central de este paradigma es la unión de

datos y procesamiento en una entidad llamada «objeto», relacionable a su vez con otras entidades «objeto».

Tradicionalmente, datos y procesamiento se han separado en diferentes áreas del diseño y la implementación de Software. Esto provocó que grandes desarrollos tuvieran problemas de fiabilidad, mantenimiento, adaptación a los cambios y escalabilidad. Con la orientación a objetos y características como el encapsulado, polimorfismo o la herencia, se permitió un avance significativo en el desarrollo de Software a cualquier escala de producción.

Otra parte importante de la programación tiene que ver con el algoritmo a implementar y las estructuras de datos necesarias para soportar los datos del problema, así como su eficiencia. Entonces entenderemos por eficiencia algorítmica para describir aquellas propiedades de los algoritmos que están relacionadas con la cantidad de recursos utilizados por el algoritmo. Un algoritmo debe ser analizado para determinar el uso de los recursos que realiza. La eficiencia algorítmica puede ser vista como análogo a la ingeniería de productividad de un proceso repetitivo o continuo. Con el objetivo de lograr una eficiencia máxima se quiere minimizar el uso de recursos. Sin embargo, varias medidas (e.g. complejidad temporal, complejidad espacial) no pueden ser comparadas directamente, luego, cual de los algoritmos es considerado más eficiente, depende de cual medida de eficiencia se está considerando como prioridad, e.g. la prioridad podría ser obtener la salida del algoritmo lo más rápido posible, o que minimice el uso de la memoria, o alguna otra medida particular.

Entonces, ¿cómo determinar si un algoritmo es mejor que otro?. Algunas pautas pueden ser:

- Facilidad de implementar
- Facilidad de entender
- Facilidad de modificar
- Usa menos memoria
- Menor tiempo de ejecución

Tomando en cuenta todo lo anterior, ¿cómo se hace un programa?

- Hay que tener claro el problema a resolver (*formalización*)

- Se debe planear cómo se quiere resolver o abordar el problema (*análisis y diseño*)
- Se elige uno o más lenguajes de programación y se escriben las instrucciones en ese lenguaje para llevar a cabo esa tarea (*codificación*)
- El texto se compila o interpreta para detectar errores sintácticos y semánticos (*depuración*)
- El programa revisado se prueba con los distintos datos de entrada (*ejecución*)
- Se evalúan los resultados, y de ser necesario se regresa a cualquiera de los pasos anteriores para completar el proceso de corrección (*validación*)

Así, para dar solución a algún problema de nuestro interés, debemos elegir el mejor algoritmo a nuestra disposición, seleccionar[42] el paradigma de programación que nos ofrezca ventajas según su eficiencia algorítmica y con ello usar el lenguaje que nos permita implementar dicho programa usando las técnicas de análisis y diseño que garanticen la calidad de un producto de Software[43].

4.2 Lenguaje de Programación

Entenderemos por un lenguaje de programación, a un lenguaje formal que especifica una serie de instrucciones para que una computadora produzca diversas clases de datos. Los lenguajes de programación pueden usarse para crear programas que pongan en práctica algoritmos específicos que controlen el comportamiento físico y lógico de una computadora. Esta formado por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Al proceso por el cual

[42] Si bien puede seleccionarse la forma pura de estos paradigmas al momento de programar, en la práctica es habitual que se mezclen, dando lugar a la programación multiparadigma o lenguajes de programación multiparadigma.

[43] La ingeniería de Software es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de Software y el estudio de estos enfoques, es decir, el estudio de las aplicaciones de la ingeniería al Software. Integra matemáticas, ciencias de la computación y prácticas cuyos orígenes se encuentran en la ingeniería.

se escribe, se prueba, se depura, se compila (de ser necesario) y se mantiene el código fuente de un programa informático se le llama programación.

Un programa permite especificar de manera precisa sobre qué datos debe operar una computadora, cómo deben ser almacenados o transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Todo esto, a través de un lenguaje que intenta estar relativamente próximo al lenguaje humano o natural. Una característica relevante de los lenguajes de programación es precisamente que más de un programador pueda usar un conjunto común de instrucciones que sean comprendidas entre ellos para realizar la construcción de un programa de forma colaborativa.

Variables son títulos asignados a espacios en memoria para almacenar datos específicos. Son contenedores de datos y por ello se diferencian según el tipo de dato que son capaces de almacenar. En la mayoría de lenguajes de programación se requiere especificar un tipo de variable concreto para guardar un dato específico. A continuación, un listado con los tipos de variables más comunes:

Tipo de dato	Breve descripción
Char	Contienen un único carácter
Int	Contienen un número entero
Float	Contienen un número decimal
String	Contienen cadenas de texto
Boolean	Solo pueden contener[44] verdadero o falso

Condicionales son estructuras de código que indican que, para que cierta parte del programa se ejecute, deben cumplirse ciertas premisas; por ejemplo: que dos valores sean iguales, que un valor exista, que un valor sea mayor que otro. Estos condicionantes por lo general solo se ejecutan una vez a lo largo del programa. Los condicionantes más conocidos y empleados en programación son:

- If: Indica una condición para que se ejecute una parte del programa.
- Else if: Siempre va precedido de un "If" e indica una condición para que se ejecute una parte del programa siempre que no cumpla la condición del if previo y sí se cumpla con la que el "else if" especifique.

[44] En el caso de variables booleanas, el cero es considerado para muchos lenguajes como el literal falso ("False"), mientras que el uno se considera verdadero ("True").

- Else: Siempre precedido de "If" y en ocasiones de "Else If". Indica que debe ejecutarse cuando no se cumplan las condiciones previas.

Bucles son parientes cercanos de los condicionantes, pero ejecutan constantemente un código mientras se cumpla una determinada condición. Los más frecuentes son:

- For: Ejecuta un código mientras una variable se encuentre entre 2 determinados parámetros.
- While: Ejecuta un código mientras que se cumpla la condición que solicita.

Hay que decir que a pesar de que existan distintos tipos de bucles, todos son capaces de realizar exactamente las mismas funciones. El empleo de uno u otro depende, por lo general, del gusto del programador.

Funciones estas se crearon para evitar tener que repetir constantemente fragmentos de código. Una función podría considerarse como una variable que encierra código dentro de sí. Por lo tanto cuando accedemos a dicha variable (la función), en realidad lo que estamos haciendo es ordenar al programa que ejecute un determinado código predefinido anteriormente.

Todos los lenguajes de programación tienen algunos elementos de formación primitivos para la descripción de los datos y de los procesos o transformaciones aplicadas a estos datos -tal como la suma de dos números o la selección de un elemento que forma parte de una colección-. Estos elementos primitivos son definidos por reglas sintácticas y semánticas que describen su estructura y significado respectivamente.

Implementación de un Lenguaje de Programación Es la que provee una manera de que se ejecute un programa para una determinada combinación de Software y Hardware. Existen básicamente dos maneras de implementar un lenguaje:

- Compilación: es el proceso que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, generando un programa equivalente que la máquina será capaz interpretar. Los programas traductores que pueden realizar esta operación se llaman

compiladores. Estos, como los programas ensambladores avanzados, pueden generar muchas líneas de código de máquina por cada proposición del programa fuente.

- Interpretación: es una asignación de significados a las fórmulas bien formadas de un lenguaje formal. Como los lenguajes formales[45] pueden definirse en términos puramente sintácticos, sus fórmulas bien formadas pueden no ser más que cadenas de símbolos sin ningún significado. Una interpretación otorga significado a esas fórmulas.

Para ayudar al programador es común el uso de editores de texto o ambientes integrados de desarrollo que permiten resaltar los elementos de la sintaxis con colores diferentes para facilitar su lectura. A la forma visible de un lenguaje de programación se le conoce como sintaxis. La mayoría de los lenguajes de programación son puramente textuales, es decir, utilizan secuencias de texto que incluyen palabras, números y puntuación, de manera similar a los lenguajes naturales escritos. Por otra parte, hay algunos lenguajes de programación que son más gráficos en su naturaleza, utilizando relaciones visuales entre símbolos para especificar un programa.

La sintaxis de un lenguaje de programación describe las combinaciones posibles de los símbolos que forman un programa sintácticamente correcto. El significado que se le da a una combinación de símbolos es manejado por su semántica -ya sea formal o como parte del código duro[46] de la referencia de implementación-.

[45] En matemáticas, lógica y ciencias de la computación, un lenguaje formal es un lenguaje cuyos símbolos primitivos y reglas para unir esos símbolos están formalmente especificados. Al conjunto de los símbolos primitivos se le llama alfabeto (o vocabulario) del lenguaje, y al conjunto de las reglas se le llama gramática formal (o sintaxis). A una cadena de símbolos formada de acuerdo a la gramática se le llama fórmula bien formada (o palabra) del lenguaje. Estrictamente hablando, un lenguaje formal es idéntico al conjunto de todas sus fórmulas bien formadas. A diferencia de lo que ocurre con el alfabeto (que debe ser un conjunto finito) y con cada fórmula bien formada (que debe tener una longitud también finita), un lenguaje formal puede estar compuesto por un número infinito de fórmulas bien formadas.

[46] Hard-Code, término del mundo de la informática que hace referencia a una mala práctica en el desarrollo de Software que consiste en incrustar datos directamente (a fuego) en el código fuente del programa, en lugar de obtener esos datos de una fuente externa como un fichero de configuración o parámetros de la línea de comandos, o un archivo de recursos.

La sintaxis de los lenguajes de programación es definida generalmente utilizando una combinación de expresiones regulares -para la estructura léxica- y la Notación de Backus-Naur[47] para la estructura gramática.

Objetivos a Cumplir en el Código Generado Para escribir programas que proporcionen los mejores resultados, cabe tener en cuenta una serie de objetivos, entre los que destacan:

- Corrección: Un programa es correcto si hace lo que debe hacer tal y como se estableció en las fases previas a su desarrollo. Para determinar si un programa hace lo que debe, es muy importante especificar claramente qué debe hacer el programa antes de desarrollarlo y, una vez acabado, compararlo con lo que realmente hace.

- Claridad: Es muy importante que el programa sea lo más claro y legible posible, para facilitar así su desarrollo y posterior mantenimiento. Al elaborar un programa se debe intentar que su estructura sea sencilla y coherente, así como cuidar el estilo en la edición; de esta forma se ve facilitado el trabajo del programador, tanto en la fase de creación como en las fases posteriores de corrección de errores, ampliaciones, modificaciones, etc. Fases que pueden ser realizadas incluso por otro programador, con lo cual la claridad es aún más necesaria para que otros programadores puedan continuar el trabajo fácilmente.

- Eficiencia: Se trata de que el programa, además de realizar aquello para lo que fue creado -es decir, que sea correcto-, lo haga gestionando de la mejor forma posible los recursos que utiliza. Normalmente, al hablar de eficiencia de un programa, se suele hacer referencia al tiempo que tarda en realizar la tarea para la que ha sido creado y a la cantidad de memoria que necesita, pero hay otros recursos que también pueden ser de consideración al obtener la eficiencia de un programa, depen-diendo de su naturaleza -espacio en disco que utiliza, tráfico de red que genera, etc.-.

[47] La notación de Backus-Naur, también conocida por sus denominaciones inglesas Backus-Naur form (BNF), Backus-Naur formalism o Backus normal form, es un metalenguaje usado para expresar gramáticas libres de contexto: es decir, una manera formal de describir lenguajes formales.

- Portabilidad: Un programa es portable cuando tiene la capacidad de poder ejecutarse en una plataforma, ya sea Hardware o Software, diferente a aquella en la que se elaboró. La portabilidad es una característica muy deseable para un programa, ya que permite, por ejemplo, a un programa que se ha desarrollado para sistemas GNU/Linux ejecutarse también en la familia de sistemas operativos Windows. Esto permite que el programa pueda llegar a más usuarios más fácilmente.

- Integridad: Un programa tiene integridad si es posible controlar su uso, el grado con que se puede controlar el acceso al Software o a los datos a personas no autorizadas denotará su integridad.

- Facilidad de uso: Un programa tiene facilidad de uso si el esfuerzo requerido es mínimo o moderado para aprender el manejo de una aplicación, trabajar con ella, introducir datos y conseguir resultados.

- Facilidad de mantenimiento: Un programa tiene facilidad de mantenimiento si el esfuerzo requerido para localizar y reparar errores es moderado o mínimo.

- Flexibilidad: Un programa tiene flexibilidad si el esfuerzo requerido para modificar y/o añadir funcionalidades a una aplicación en funcionamiento es moderado o mínimo.

- Reusabilidad: Es el grado en que un programa o porción de este es reusable en otras aplicaciones.

- Interoperatibilidad: Un programa debe poder comunicarse con otras aplicaciones o sistemas informáticos, el esfuerzo necesario para ello medirá su interoperatibilidad.

Existe una gran variedad de lenguajes de programación y su grado de uso depende de diversos factores, entre los que destacan: C, C++, Java y Python.

4.3 Conceptos Transversales

En esta sección comentaremos algunos conceptos transversales a los distintos paradigmas de programación y a los lenguajes que los implementan.

Abstracción Entendiendo un sistema como una abstracción de la realidad, los datos son las entidades que representan cada uno de los aspectos de la realidad que son significativos para el funcionamiento del sistema. Para que tenga sentido como abstracción de la realidad, cada dato implica un determinado valor y requiere de una convención que permita representarlo sin ambigüedad y procesarlo de una manera confiable.

En la misma línea, la lógica del procesamiento de los datos es también una abstracción de los procesos que suceden en la realidad que conforma el dominio de la aplicación. El proceso de abstraerse progresivamente de los detalles y así manejar niveles de abstracción, es el que permite construir sistemas complejos.

Las unidades de Software que realizan la funcionalidad del sistema requieren también de convenciones y criterios de ordenamiento y articulación interna tanto para un funcionamiento confiable y eficiente del sistema, como para que el proceso de construcción y mantenimiento del Software sea de la forma más simple posible.

Modularización, Encapsulamiento y Delegación Una estrategia central de la programación es buscar la manera de organizar y distribuir la funcionalidad de un sistema complejo en unidades más pequeñas de Software con que se responsabilizan de tareas específicas y que interactúan entre ellas. Estas unidades reciben nombres diferentes según cada lenguaje de programación, como rutinas, funciones, procedimientos, métodos, predicados, subprogramas, bloques, entidades, siendo "módulos" una de las más frecuentes y que dan origen al término.

La clave de cada módulo es no conocer el funcionamiento interno de los demás módulos con los que interactúa, sino sólo su interfaz, es decir, la forma en que debe enviarle información adicional en forma de parámetros y cómo va a recibir las respuestas. Esta propiedad recibe diversos nombres, como el de encapsulamiento, ocultación de información o "caja negra". Ante la modificación de una funcionalidad en particular del sistema, en la medida que su implementación esté encapsulada en un módulo, el impacto que produce su cambio no afectará a los otros módulos que interactúan con el.

En concordancia con la distribución de responsabilidades entre las diferentes unidades de Software, la delegación consiste en la invocación que desde un módulo se efectúa a otro módulo, de manera que el que invoca indica de forma explicita qué es lo que pretende y el que es invocado se ocupa de todo lo

necesario para realizarlo. Puede realizarse de numerosas maneras, variando el criterio de distribución de responsabilidades, el modo de evaluación, la forma de paso de parámetros, los tipos de datos que utiliza, de acuerdo a las posibilidades y restricciones de cada lenguaje y paradigma.

Declaratividad La declaratividad, en términos generales, se basa en la separación del conocimiento sobre la definición del problema con la forma de buscar su solución, una separación entre la lógica y el control.

En un programa declarativo se especifican un conjunto de declaraciones, que pueden ser proposiciones, condiciones, restricciones, afirmaciones, o ecuaciones, que caracterizan al problema y describen su solución. A partir de esta información el sistema utiliza mecanismos internos de control, comúnmente llamado "motores", que evalúan y relacionan adecuadamente dichas especificaciones, la manera de obtener la solución. De esta forma, en vez de ser una secuencia de órdenes, un programa es un conjunto de definiciones sobre el dominio del problema.

Basándose en la noción de delegación, la declaratividad plantea como criterio para distribuir las responsabilidades, separar las relacionadas con modelar o definir el conocimiento del problema de aquellas de manipular ese conocimiento para alcanzar un objetivo concreto. En otras palabras, distinguir el "qué" del "cómo".

La declaratividad brinda la posibilidad de usar una misma descripción en múltiples contextos en forma independiente de los motores que se utilicen.

Permite focalizar por un lado en las cuestiones algorítmicas del motor, por ejemplo para trabajar en forma unificada sobre eficiencia, y por otro en la definición del dominio del problema y la funcionalidad de la aplicación en sí.

La noción opuesta, aunque en cierta medida complementaria, de la declaratividad, puede denominarse "proceduralidad". Los programas procedurales se construyen indicando explícitamente la secuencia de ejecución en la que se procesan los datos y obtienen los resultados. Para ello se detalla un conjunto de sentencias, ordenadas mediante estructuras de control como decisiones, iteraciones y secuencias, que conforman "algoritmos".

En un sistema complejo, no se puede hablar de declaratividad o proceduralidad como conceptos excluyentes o totalizantes, sino que coexisten y se relacionan en una permanente tensión. Dependiendo de los lenguajes y de las herramientas que se utilicen, y en particular del diseño del sistema, habrá

partes del sistema que por su sentido o ubicación dentro del sistema global serán más declarativas o procedurales que otras, logrando con ello aprovechar las ventajas respectivas.

Tipos de Datos Un tipo de dato, o como también es llamado, un tipo abstracto de dato, es un conjunto de valores y de operaciones asociadas a ellos. La utilización de diversos tipos de datos permite la agrupación o clasificación del gran volumen y variedad de valores que es necesario representar y operar en un sistema, según sus semejanzas y diferencias.

Tomando como criterio las similitudes en cuanto al contenido de lo que representan, una primera condición es la existencia de un conjunto de valores o entidades homogéneos en su representación. Otra condición es que los elementos del mencionado conjunto se comporten en forma uniforme respecto a una serie de operaciones.

Cada paradigma de programación, y en particular cada lenguaje, tiene su forma de determinar tanto la conformación de cada tipo de dato, con sus valores y operaciones, como la forma en que se relacionan entre sí conformando un sistema de tipo de datos.

En un lenguaje fuertemente tipado, toda variable y parámetro deben ser definidos de un tipo de dato en particular que se mantiene sin cambios durante la ejecución del programa, mientras que en uno débilmente tipado no, sino que pueden asumir valores y tipos de datos diferentes durante la ejecución del programa.

El tipo de dato al que pertenece una entidad determina la operatoria que se puede realizar con el. Una tarea que realizan muchos de los lenguajes de programación como forma de su mecanismo interno es el chequeo del tipo de dato de las entidades del programa. Esta acción se realiza de diferentes maneras, con mayor o menor flexibilidad, y en diferentes momentos, como la compilación o la ejecución del programa, y en otros no se realiza.

De todas maneras, el tipo de datos permite entender qué entidades tienen sentido en un contexto, independientemente de la forma de chequeo o si el tipado es débil o fuerte. Por ejemplo, a un bloque de Software que recibe como argumento una variable, conocer de qué tipo de dato es, le permite saber qué puede hacer con ella.

Estructuras de Datos Los valores atómicos, es decir, aquellos que no pueden ser descompuestos en otros valores, se representan mediante tipos de

datos simples.

En contrapartida, en un tipo de dato compuesto los valores están compuestos a su vez por otros valores, de manera que conforma una estructura de datos. Cada uno de los valores que forman la estructura de datos corresponde a algún tipo de dato que puede ser tanto simple como compuesto.

Su utilidad consiste en que se pueden procesar en su conjunto como una unidad o se pueden descomponer en sus partes y tratarlas en forma independiente. En otras palabras, las estructuras de datos son conjuntos de valores.

Polimorfismo y Software Genérico El polimorfismo es un concepto para el que se pueden encontrar numerosas y diferentes definiciones. En un sentido amplio, más allá de las especificidades de cada paradigma y del alcance que se le dé a la definición del concepto desde la perspectiva teórica desde la que se le aborde, el objetivo general del polimorfismo es construir piezas de Software genéricas que trabajen indistintamente con diferentes tipos de entidades, para otra entidad que requiere interactuar con ellas; en otras palabras, que dichas entidades puedan ser intercambiables. Mirando con mayor detenimiento los mecanismos que se activan y sus consecuencias para el desarrollo de sistemas, se pueden distinguir dos grandes situaciones.

Hay polimorfismo cuando, ante la existencia de dos o más bloques de Software con una misma interfaz, otro bloque de Software cualquiera puede trabajar indistintamente con ellos. Esta noción rescata la existencia de tantas implementaciones como diferentes tipos de entidades con que se interactúe.

Una entidad emisora puede interactuar con cualquiera de las otras entidades de acuerdo a las características de la interfaz común, y la entidad receptora realizará la tarea solicitada de acuerdo a la propia implementación que tenga definida, independientemente de las otras implementaciones que tengan las otras entidades. Consistentemente con la noción de delegación, la entidad emisora se desentiende de la forma en que las otras entidades implementaron sus respuestas, ya sea igual, parecida o totalmente diferente.

Analizando el concepto desde el punto de vista de las entidades que responden a la invocación, el proceso de desarrollo debe contemplar la variedad y especificidad de cada una para responder adecuadamente a lo que se les solicita. Desde el punto de vista de la entidad que invoca, el proceso es transparente, y es en definitiva esta, la que se ve beneficiada por el uso del concepto.

Otra forma de obtener un bloque de Software genérico es ante el caso en

que las entidades con las que el bloque quiere comunicarse, en vez de tener diferentes implementaciones, aún siendo de diferente tipo, sean lo suficientemente similares para compartir una misma y única implementación.

Desde el punto de vista de la entidad que invoca, el proceso continúa siendo transparente y sigue aprovechando los beneficios de haber delegado la tarea y haberse despreocupado de qué tipo de entidad es la receptora. En este caso, el concepto de polimorfismo se relaciona o se basa en la coexistencia de herencia, de variables de tipo de dato o en las formas débiles de declaración y chequeo de tipos de datos, dependiendo de las diferentes herramientas de cada paradigma.

Asignación y Unificación La asignación destructiva es una operación que consiste en cambiar la información representada por una variable, de forma tal que si se consulta su valor antes y después de dicha operación, se obtiene un resultado distinto. Estas asignaciones se realizan repetitivamente sobre la misma celda de memoria, remplazando los valores anteriores.

La asignación determina el estado de una variable, que consiste en el valor que contiene en un momento en particular. La asignación destructiva es la forma más usual de provocar efecto de lado, pero no la única, ya que, dependiendo de los lenguajes, hay otro tipo de instrucciones que también permiten modificar el estado de información de un sistema.

La unificación es un mecanismo por el cual una variable que no tiene valor, asume un valor. Una vez unificada, o "ligada", como también se le dice, una variable no cambia su valor, por lo que no existe la noción de estado. La duración de la unificación está condicionado por el alcance que tienen las variables en cada lenguaje en particular, pudiendo una variable unificarse con varios valores alternativos en diferentes momentos de la ejecución del bloque de Software en el que se encuentran. Se suele denominar como "indeterminada" a una variable sin ligar o no unificada.

Modo de Evaluación Los parámetros que se utilizan en la invocación de un bloque de Software cualquiera pueden ser evaluados en diferentes momentos de acuerdo al modo de evaluación que utilice el lenguaje de programación.

La evaluación ansiosa consiste en que los argumentos son evaluados antes de invocar al bloque de Software y es responsabilidad de la entidad que los invoca.

La evaluación diferida plantea que la evaluación de los argumentos es res-

ponsabilidad del bloque de Software invocado, quien decide el momento en que lo hará. Provoca que se difiera la evaluación de una expresión, permitiendo que en algunos casos, de acuerdo a cómo sea la implementación, no sea necesario evaluarla nunca, con el consiguiente beneficio en términos de eficiencia.

La evaluación diferida es utilizada en cierto tipo de situaciones que serían consideradas erróneas o imposibles de resolver con la evaluación ansiosa, como por ejemplo los bucles o las listas infinitas.

Orden Superior Asumiendo un esquema de un único orden, en un programa existen por un lado datos y por otro los procedimientos -y todo bloque de Software- que trabajan con ellos, de manera tal que los proce-dimientos reciben datos como parámetros y devuelven datos como resultados. La noción de orden superior plantea que los procedimientos son tratados como datos y en consecuencia pueden ser utilizados como parámetros, representados en variables, devueltas como resultados u operados en cálculos más complejos con otros datos. Se denomina de orden superior a los procedimientos que reciben como argumentos a otros procedimientos.

Recursividad La recursividad, entendida como iteración con asignación no destructiva, está relacionada con el principio de inducción. En general, un bloque de Software recursivo se define con al menos un término recursivo, en el que se vuelve a invocar el bloque que se está definiendo, y algún término no recursivo como caso base para detener la recursividad.

4.4 Algo de Programación

Tipos de datos Dependiendo del lenguaje (véase [**?**], [**?**], [**?**] y [**?**]), la implementación del mismo y la arquitectura en la que se compile/ejecute el programa, se tienen distintos tipos de datos, pero los básicos son[48]:

- char, 8 bits, rango de -128 a 127
- unsigned char, 8 bits, rago de 0 a 255
- int, 16 bits, rango de -32,768 a 32,767
- unsigned int, 16 bits, rango de 0 a 65,535
- long int, 32 bits, rango de -2,147,483,648 a 2,147,483,647
- long, 64 bits, rango de -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807
- float, 32 bits, rango de -3.4x10^{38} a 3.4x10^{38}, con 14 dígitos de precisión
- double, 64 bits, rango de -1.7x10^{308} a 1.7x10^{308}, con 16 dígitos de precisión
- long double, 80 bits, 3.4x10^{-4932} a 3.4x10^{4932}, con 18 dígitos de precisión
- boolean, 1 bit, rango de True o False

La conversión entre los diferentes tipos de datos esta en función del lenguaje y existe una perdida de datos si se hace de un tipo de dato que no "quepa" en el rango del otro; estos y otros errores de programación han generado graves errores en el sistema automatizado de control que han desembocado por ejemplo en, misiones fallidas de cohetes espaciales y su carga en numerosas ocasiones[49], la quiebra de la empresa Knight Capital 2012, sobredosis de radioterapia en 1985 a 1987, la desactivación de servidores de Amazon en el 2012, el apagón en el noreste de EE.UU. en el 2003, fallo en el sistema de reserva de vuelos de American Airlines en el 2013, fallo en el sistema de misiles estadounidense Patriot en Dhahran en el 1991, etc.

[48] La precedencia de los operadores básicos son: $=$ el de mayor procedencia, siguen $-$ y $+$ unario, luego $*, /, \%$ y finalmente los operadores $+$ y $-$.

[49] Una de las pérdidas, fue el satélite mexicano Centenario en mayo del 2015.

Condicionales El flujo de programas a menudo tiene que dividirse. Es decir, si una condición se cumple, se hace algo, y si no, se hace otra cosa. Los enunciados condicionales permiten cambiar el comportamiento del programa de acuerdo a una condición dada, la estructura básica de un condicional es:

Pseudocódigo:

```
Si <condición> entonces
  <acción>
```

Ejemplo en C, C++ y Java[50]:

```
if (x > 0) x*= 10;
```

```
if (x > 0) {
  x *= 10;
}
```

Ejemplo en Python:

```
if x > 0:
  x = x * 10
```

La estructura completa del condicional es:

Pseudocódigo:

```
Si <condición> entonces
  <acción>
else
  <acción>
```

Ejemplo en C, C++ y Java:

```
if (x % 2 == 0) x = x * x;
else x = x * 2;
```

```
if (x % 2 == 0) {
  x = x * x;
} else {
  x = x * 2;
}
```

[50] En condicionales, tienen mayor precedencia $<, >, >=, <=$, luego están $==$ y $!=$, finalmente && (AND) y || (OR).

Ejemplo en Python:

```
if x % 2 == 0:
   x = x * x
else:
   x = x * 2
```

La estructura condicional encadenada es:
Pseudocódigo:

```
Si <condición> entonces
   <acción>
else if
   <acción>
else
   <acción>
```

Ejemplo en C, C++ y Java:

```
if (x < y) {
   z = x;
} else if (x > y) {
   z = y;
} else {
   z = 0;
}
```

Ejemplo en Python:

```
if x < y:
   z = x
elif[51] x > y:
   z = y
else:
   z = 0
```

[51] elif es una contracción de "else if", sirve para enlazar varios "else if", sin tener que aumentar las tabulaciones en cada nueva comparación.

Condicionales con operadores lógicos, tenemos dos operadores lógicos el AND (&&) y el OR (||), ejemplos:

Pseudocódigo:

```
Si <condición operador condición> entonces
  <acción>
else
  <acción>
```

Ejemplo en C, C++ y Java:

```
if (x % 2 == 0 || x > 0) x = x * x;
else x = x * 2;
```

```
if (x % 2 == 0 && x > 0) {
  x = x * x;
} else {
  x = x * 2;
}
```

Ejemplo en Python:

```
if x % 2 == 0 and x > 0:
  x = x * x
else:
  x = x * 2
```

Bucle for La implementación de bucles mediante el comando *for* permite ejecutar el código que se encuentre entre su cuerpo mientras una variable se encuentre entre 2 determinados parámetros, la estructura básica es:

Pseudocódigo:

```
for <inicio; mientras; incremento/decremento> entonces
  <acción>
```

Ejemplo en C, C++ y Java:

```
for (x = 0; x < 21; x++) {
  z = z + x;
}
```

Ejemplo en Python:

```
for x in range(0, 21):
  z = z + x
for i in range(1,3):
  if n == 3:
    break
else[52]
  print("no se encontro el numero 3")
```

Bucle for para navegar en por ejemplo Listas Si se tene una lista declarada, es posible hacer un recorrido en sus elementos usando *for*, por ejemplo:

Ejemplo en Java:

```
List<String> lista = new ArrayList<String>();
...
for (String elem : lista) {
    System.out.print(elem);
}
```

Ejemplo en Python:

```
lista = ["uno","dos","tres","cuatro"]
for elem in lista:
    print elem
```

Bucle while La implementación de bucles mediante el comando *while* permite ejecutar el código que se encuentre entre su cuerpo mientras una condición se cumple, la estructura básica es:

Pseudocódigo:

```
while <condición> entonces
  <acción>
```

Ejemplo en C, C++ y Java:

[52] La instrucción *else* sólo se ejecutará si concluye normalmente el ciclo *for*, es decir, sin la interrupción por *break*. En este caso nunca se ejecutara el *print*.

```
while (z < 20) {
  z = z + x;
}
```

Ejemplo en Python:

```
while z < 20:
  z = z + x
else[53]
  print("concluyo")
```

Bucle do-while La implementación de bucles mediante el comando *do-while* permite ejecutar el código que se encuentre entre su cuerpo y después revisa si se cumple la condición para continuar el ciclo, se puede usar *do-while* en C, C++ y Java, pero no esta presente en Python (pero siempre se puede usar un *while* para emularlo). La estructura básica es:

Pseudocódigo:

```
do
  <acción>
while <condición>
```

Ejemplo en C, C++ y Java:

```
do {
  z = z + x;
} while (z < 20);
```

Condicional switch Cuando se requiere hacer un condicional sobre una variable y que esta tenga varias alternativas, una opción para evitar una cadena de sentencias *if-else*, se puede usar *switch* en C, C++ y Java, pero no esta presente en Python (siempre se puede usar *if-elif* para emularlo). La estructura básica es:

Pseudocódigo:

[53] La instrucción *else* sólo se ejecutará si concluye normalmente el ciclo *while*, es decir, sin la interrupción por *break*. En este caso al terminar el *while* se ejecutara el *print*.

```
switch(expresion) {
  case constante:
    <acción>
    break;
  case constante:
    <acción>
    break;
  .
  .
  .
  default:
  <acción>
}
```

son opcionales los *break* y el *default*.

Ejemplo en C, C++ y Java:

```
switch(i) {
  case 1:
    x=23;
  case 2:
    x ++;
    break;
  default:
  x=0;
}
```

Primer ejemplo Como ya se ha hecho costumbre, el primer ejemplo de un programa es: *Hola Mundo*, así que iniciemos con ello creando el archivo correspondiente *hola.ext*[54] en cualquier editor de texto o en un IDE, entonces:
Ejemplo en C:

```
#include <stdio.h>
int main(void) {
  printf("Hello World\n");
  return 0;
}
```

para compilarlo usamos *gcc* en línea de comandos mediante:

```
$ gcc hola.c -o hola
```

y lo ejecutamos con:

```
$ ./hola
```

Ejemplo en C++:

```
#include <iostream>
int main() {
  std::cout << "Hello World!\n";
}
```

para compilarlo usamos *g++* en línea de comandos mediante:

```
$ g++ hola.cpp -o hola
```

y lo ejecutamos con:

```
$ ./hola
```

[54]La extensión depende del lenguaje de programación, para el lenguaje C la extensión es .c, para el lenguaje C++ la extensión es .cpp, para el lenguaje Java la extensión es .java, para el lenguaje Python la extensión es .py.

Ejemplo en Java:

```
class hola {
   public static void main(String[] args) {
      System.out.println("Hello world!");
   }
}
```

para compilarlo usamos *javac* en línea de comandos mediante:

```
$ javac hola.java
```

y lo ejecutamos con:

```
$ java hola
```

Ejemplo en Python 2:

```
print "Hello World\n"
```

para compilarlo y ejecutarlo usamos *python2* en línea de comandos mediante:

```
$ python2 hola.py
```

Ejemplo en Python 3:

```
print ("Hello World\n")
```

para compilarlo y ejecutarlo usamos *python3* en línea de comandos mediante:

```
$ python3 hola.py
```

Un Ejemplo de Cálculo de Primos El algoritmo más simple, para determinar si un número es primo o compuesto, es hacer una serie de divisiones sucesivas del número, con todos los números primos menores que él, si alguna división da como residuo 0 o es divisible con el número entonces es compuesto en caso contrario es primo.

- Iniciamos con el número 4
- Verificamos si es divisible con los primos almacenados (iniciamos con 2 y 3)
- Si es divisible con algún número primo entonces es compuesto, en caso contrario es primo y este se guarda como un nuevo primo
- se incrementa uno al número y se regresa a verificar si es divisible entre los números primos almacenados, hasta encontrar por ejemplo los primeros mil primos

Ejemplo en C y C++:

```
#include <stdio.h>
// NPB Numero de primos a buscar
#define NPB 1000
int main ()
{
  int n, i, np;
  int p[NPB];
  // Guarda los primeros 2 primos
  p[0] = 2;
  p[1] = 3;
  np = 2;
  // Empieza la busqueda de primos a partir del numero 4
  n = 4;
  // Ciclo para buscar los primeros NPB primos
  while (np < NPB)
  {
    for (i = 0; i < np; i++)
    {
      if((n % p[i]) == 0) break;
```

```
    }
    if(i == np)
    {
      p[i] = n;
      np++;
    }
    n++;
  }
  // Visualiza los primos encontrados
  printf("\nVisualiza los primeros %d primos\n", NPB);
  for (i = 0; i < NPB; i++)
    printf("%d\n", p[i]);
  return 0;
}
```

Ejemplo en Java:

```
public class CalPrimos {
  public static void main(String[] args) {
    // NPB Numero de primos a buscar
    int NPB = 1000;
    int n, i, np;
    int p[] = new int[NPB];
    // Guarda los primeros 2 primos
    p[0] = 2;
    p[1] = 3;
    np = 2;
    // Empieza la busqueda de primos a partir del numero 4
    n = 4;
    // Ciclo para buscar los primeros NPB primos
    while (np < NPB) {
      for (i = 0; i < np; i++) {
        if((n % p[i]) == 0) break;
      }
      if (i == np) {
        p[i] = n;
        np++;
      }
```

```
            n++;
          }
          // Visualiza los primos encontrados
          System.out.println("Visualiza los primeros " + NPB + "
primos");
          for (i = 0; i < NPB; i++) System.out.println(p[i]);
        }
      }
```

Nótese que el algoritmo para buscar primos de los lenguajes C, C++ y el de Java son muy similares salvo la declaración de la función inicial y el arreglo que contendrá a los primos. Esto deja patente la cercanía entre dichos lenguajes y por que en este trabajo los presentamos en forma conjunta.

Ejemplo en Python:

```
def criba_Eratostenes(N):
  p = [] # inicializa el arreglo de primos encontrados
  # Guarda los primeros 2 primos
  p.append(2)
  p.append(3)
  np = 2
  # Empieza la busqueda de primos a partir del numero 4
  n = 4
  #Ciclo para buscar los primeros N primos
  while np < N:
    xi = 0
    for i in p:[55]
      xi = xi + 1
      if (n % i) == 0:
        break
    if xi == np:
      p.append(n)
      np = np + 1
    n = n + 1
   # Visualiza los primos encontrados
```

[55] En este código, al usar el *for* sobre los elementos del arreglo de primos, es necesario usar un contador para saber si ya se recorrieron todos los primos existentes y así determinar si es un nuevo primo y agregarlo a la lista.

```
    print("Visualiza los primeros " + str(N) + " primos ")
    for i in range(np):
      print(p[i])
    return p
# Solicita el calculo de los primeros primos
P = criba_Eratostenes(1000)
print(P)
```

Otro ejemplo en Python:

```
def criba_Eratostenes(N):
    p = [] # inicializa el arreglo de primos encontrados
    # Guarda los primeros 2 primos
    p.append(2)
    p.append(3)
    np = 2
    # Empieza la busqueda de primos a partir del numero 4
    n = 4
    #Ciclo para buscar los primeros N primos
    while np < N:
      for i in range(np):[56]
        if (n % p[i]) == 0:
          break
      if i == np-1:
        p.append(n)
        np = np + 1
      n = n + 1
     # Visualiza los primos encontrados
    print("Visualiza los primeros " + str(N) + " primos ")
    for i in range(np):
      print(p[i])
    return p
# Solicita el calculo de los primeros primos
P = criba_Eratostenes(1000)
print(P)
```

[56] Aquí, se hace el recorrido sobre el arreglo de primos usando la indexación sobre sus elementos, usando el número de elementos que se tiene mediante el uso del *for* con un *range*.

4.5 Introducción a los Paradigmas de Programación

La construcción de programas de cómputo -Software- puede involucrar elementos de gran complejidad, que en muchos casos no son tan evidentes como los que se pueden ver en otras Ciencias e Ingenierías. Un avión, una mina, un edificio, una red de ferrocarriles son ejemplos de sistemas complejos de otras Ingenierías, pero el programador construye sistemas cuya complejidad puede parecer que permanece oculta. El usuario siempre supone que en informática todo es muy fácil -"apretar un botón y ya esta"-.

Cuando se inicia uno en la programación, es común el uso de pequeños ejemplos, generalmente se programa usando una estructura secuencial -una instrucción sigue a la otra- en programas cortos[57], cuando los ejemplos crecen se empieza a usar programación estructurada[58] -que usa funciones- para después proseguir con ejemplos más complejos haciendo uso de la programación orientada a objetos -uso de clases y objetos- o formulaciones híbridas de las anteriores[59].

A continuación delinearemos estos paradigmas de programación:

Programación Secuencial es un paradigma de programación en la que una instrucción del código sigue a otra en secuencia también conocido como código espagueti. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del programa.

[57] Los ejemplos no son complejos, suelen estar construidos y mantenidos por una o pocas personas, son códigos de cientos o miles de líneas y tienen un ciclo de vida corto. Además, se puede construir aplicaciones alternativas en un período razonable de tiempo y no se necesitan grandes esfuerzos en anális y diseño.

[58] Al crecer la complejidad del Software a desarrollar, es muy difícil o imposible que un desarrollador o un grupo pequeño de ellos pueda comprender todas las sutilidades de su diseño, para paliar los problemas que conlleva el desarrollo de grandes y complejos sistemas informáticos surge la programación orientada a objetos.

[59] El surgimiento de la programación orientada objetos trata de lidiar con una gran cantidad de requisitos que compiten entre sí, incluso contradiciéndose, tienen desacoplamientos de impedancias entre usuarios del sistema y desarrolladores y es común la modificación de los requisitos con el paso del tiempo pues los usuarios y desarrolladores comienzan a compenetrarse mejor. Así, la programación orientada a objetos permite dirigir un equipo grande de desarrolladores, manejar una gran cantidad de código, usar estándares de desarrollo —al igual que en otras ingenierías— y verificar la fiabilidad de los estándares existentes en el mercado.

Programación Estructurada también llamada Procedimental, es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa de computadora recurriendo únicamente a subrutinas y tres estructuras básicas: secuencia, selección (if y switch) e iteración (bucles for y while); así mismo, se considera innecesario y contraproducente el uso de la instrucción de transferencia incondicional, que podría conducir a código espagueti, mucho más difícil de seguir y de mantener, y fuente de numerosos errores de programación.

Entre las ventajas de la programación estructurada sobre el modelo anterior -hoy llamado despectivamente código espagueti-, cabe citar las siguientes:

- Los programas son más fáciles de entender, pueden ser leídos de forma secuencial y no hay necesidad de tener que rastrear saltos de líneas dentro de los bloques de código para intentar entender la lógica interna.
- La estructura de los programas es clara, puesto que las instrucciones están más ligadas o relacionadas entre sí.
- Se optimiza el esfuerzo en las fases de pruebas y depuración. El seguimiento de los fallos o errores del programa (debugging), y con él su detección y corrección se facilita enormemente.
- Se reducen los costos de mantenimiento. Análogamente a la depuración, durante la fase de mantenimiento, modificar o extender los programas resulta más fácil.
- Los programas son más sencillos y más rápidos de confeccionar.
- Se incrementa el rendimiento de los programadores.

Programación Orientada a Objetos (POO, u OOP según sus siglas en inglés) es un paradigma de programación que viene a innovar la forma de obtener resultados. El surgimiento de la programación orientada objetos trata de lidiar con una gran cantidad de requisitos que compiten entre sí, incluso contradiciéndose, tienen desacoplamientos de impedancias entre usuarios del sistema y desarrolladores y es común la modificación de los requisitos con el paso del tiempo pues los usuarios y desarrolladores comienzan a compenetrarse mejor. Así, la programación orientada a objetos permite

dirigir un equipo grande de desarrolladores, manejar una gran cantidad de código, usar estándares de desarrollo -al igual que en otras ingenierías- y verificar la fiabilidad de los estándares existentes en el mercado.

Los objetos manipulan los datos de entrada para la obtención de datos de salida específicos, donde cada objeto ofrece una funcionalidad especial. Los objetos son entidades que tienen un determinado «estado», «comportamiento (método)» e «identidad»:

- La identidad es una propiedad de un objeto que lo diferencia del resto; dicho con otras palabras, es su identificador -concepto análogo al de identificador de una variable o una constante-.

- Un objeto contiene toda la información que permite definirlo e identificarlo frente a otros objetos pertenecientes a otras clases e incluso frente a objetos de una misma clase, al poder tener valores bien diferenciados en sus atributos. A su vez, los objetos disponen de mecanismos de interacción llamados métodos, que favorecen la comunicación entre ellos. Esta comunicación favorece a su vez el cambio de estado en los propios objetos. Esta característica lleva a tratarlos como unidades indivisibles, en las que no se separa el estado y el comportamiento.

- Los métodos (comportamiento) y atributos (estado) están estrechamente relacionados por la propiedad de conjunto. Esta propiedad destaca que una clase requiere de métodos para poder tratar los atributos con los que cuenta. El programador debe pensar indistintamente en ambos conceptos, sin separar ni darle mayor importancia a alguno de ellos. Hacerlo podría producir el hábito erróneo de crear clases contenedoras de información por un lado y clases con métodos que manejen a las primeras por el otro. De esta manera se estaría realizando una «programación estructurada camuflada» en un lenguaje de POO.

La programación orientada a objetos difiere de la programación estructurada tradicional, en la que los datos y los procedimientos están separados y sin relación, ya que lo único que se busca es el procesamiento de unos datos de entrada para obtener otros de salida. La programación estructurada anima al programador a pensar sobre todo en términos de procedimientos o funciones, y en segundo lugar en las estructuras de datos que esos procedimientos manejan. En la programación estructurada solo se escriben funciones que procesan

datos. Los programadores que emplean POO, en cambio, primero definen objetos para luego enviarles mensajes solicitándoles que realicen sus métodos por sí mismos.

Conceptos fundamentales La programación orientada a objetos es una forma de programar que trata de encontrar solución a los problemas que genera el desarrollo de proyectos de tamaño mediano o grande y/o complejos. Introduce nuevos conceptos, que superan y amplían conceptos antiguos ya conocidos. Entre ellos destacan los siguientes:

- Clase: se puede definir de las propiedades y comportamiento de un tipo de objeto concreto. La instanciación es la lectura de estas definiciones y la creación de un objeto a partir de ella.
- Herencia: Por ejemplo, la herencia de la clase C a la clase D, es la facilidad mediante la cual la clase D hereda en ella cada uno de los atributos y operaciones de C, como si esos atributos y operaciones hubiesen sido definidos por la misma D. Por lo tanto, puede usar los mismos métodos y variables registrados como «públicos (public)» en C. Los componentes registrados como «privados (private)» también se heredan pero se mantienen escondidos al programador y sólo pueden ser accedidos a través de otros métodos públicos. Para poder acceder a un atributo u operación de una clase en cualquiera de sus subclases pero mantenerla oculta para otras clases es necesario registrar los componentes como «protegidos (protected)», de esta manera serán visibles en C y en D pero no en otras clases.
- Objeto: La instancia de una clase. Entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos), los mismos que consecuentemente reaccionan a eventos. Se corresponden con los objetos reales del mundo que nos rodea, o con objetos internos del sistema (del programa).
- Método: Es un algoritmo asociado a un objeto (o a una clase de objetos), cuya ejecución se desencadena tras la recepción de un «mensaje». Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en las propiedades del objeto, o la generación de un «evento» con un nuevo mensaje para otro objeto del sistema.

- Evento: Es un suceso en el sistema -tal como una interacción del usuario con la máquina, o un mensaje enviado por un objeto-. El sistema maneja el evento enviando el mensaje adecuado al objeto pertinente. También se puede definir como evento la reacción que puede desencadenar un objeto; es decir, la acción que genera.

- Atributos: Características que tiene la clase.

- Mensaje: Una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos con ciertos parámetros asociados al evento que lo generó.

- Propiedad o atributo: Contenedor de un tipo de datos asociados a un objeto (o a una clase de objetos), que hace los datos visibles desde fuera del objeto y esto se define como sus características predeterminadas, y cuyo valor puede ser alterado por la ejecución de algún método.

- Estado interno: Es una variable que se declara privada, que puede ser únicamente accedida y alterada por un método del objeto, y que se utiliza para indicar distintas situaciones posibles para el objeto (o clase de objetos). No es visible al programador que maneja una instancia de la clase.

- Componentes de un objeto: Atributos, identidad, relaciones y métodos.

- Identificación de un objeto: Un objeto se representa por medio de una tabla o entidad que esté compuesta por sus atributos y funciones correspondientes.

Características de la POO Existe un acuerdo acerca de qué características contempla la «orientación a objetos». Las características siguientes son las más importantes:

- Abstracción: Denota las características esenciales de un objeto, donde se capturan sus comportamientos. Cada objeto en el sistema sirve como modelo de un «agente» abstracto que puede realizar trabajo, informar y cambiar su estado, y «comunicarse» con otros objetos en el sistema sin revelar «cómo» se implementan estas características. Los procesos, las funciones o los métodos pueden también ser abstraídos,

y, cuando lo están, una variedad de técnicas son requeridas para ampliar una abstracción. El proceso de abstracción permite seleccionar las características relevantes dentro de un conjunto e identificar comportamientos comunes para definir nuevos tipos de entidades en el mundo real. La abstracción es clave en el proceso de análisis y diseño orientado a objetos, ya que mediante ella podemos llegar a armar un conjunto de clases que permitan modelar la realidad o el problema que se quiere atacar.

- Encapsulamiento: Significa reunir todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión (diseño estructurado) de los componentes del sistema.

- Polimorfismo: Comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre; al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando. O, dicho de otro modo, las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del objeto referenciado. Cuando esto ocurre en «tiempo de ejecución» esta última característica se llama asignación tardía o asignación dinámica.

- Herencia: Las clases no se encuentran aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. La herencia organiza y facilita el polimorfismo y el encapsulamiento, permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que volver a implementarlo. Esto suele hacerse habitualmente agrupando los objetos en clases y estas en árboles o enrejados que reflejan un comportamiento común. Cuando un objeto hereda de más de una clase se dice que hay herencia múltiple; siendo de alta complejidad técnica por lo cual suele recurrirse a la herencia virtual para evitar la duplicación de datos.

- Modularidad: Se denomina «modularidad» a la propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos),

cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las partes restantes. Estos módulos se pueden compilar por separado, pero tienen conexiones con otros módulos. Al igual que la encapsulación, los lenguajes soportan la modularidad de diversas formas.

- Principio de ocultación: Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una «interfaz» a otros objetos que específica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas; solamente los propios métodos internos del objeto pueden acceder a su estado. Esto asegura que otros objetos no puedan cambiar el estado interno de un objeto de manera inesperada, eliminando efectos secundarios e interacciones inesperadas.

- Recolección de basura: La recolección de basura (garbage collection) es la técnica por la cual el entorno de objetos se encarga de destruir automáticamente, y por tanto desvincular la memoria asociada, los objetos que hayan quedado sin ninguna referencia a ellos. Esto significa que el programador no debe preocuparse por la asignación o liberación de memoria, ya que el entorno la asignará al crear un nuevo objeto y la liberará cuando nadie lo esté usando.

Muchos de los objetos prediseñados de los lenguajes de programación actuales permiten la agrupación en bibliotecas o librerías, sin embargo, muchos de estos lenguajes permiten al usuario la creación de sus propias bibliotecas. Está basada en varias técnicas, incluyendo herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento.

Recursión o Recursividad Un concepto que siempre le cuesta bastante a los programadores que están empezando es el de recursión o recursividad (se puede decir de las dos maneras). La recursividad consiste en funciones que se llaman a sí mismas, evitando el uso de bucles y otros iteradores.

Uno ejemplo fácil de ver y que se usa a menudo es el cálculo del factorial de un número entero. El factorial de un número se define como ese número multiplicado por el anterior, éste por el anterior, y así sucesivamente hasta

llegar a 1. Así, por ejemplo, el factorial del número 5 sería: 5x4x3x2x1 = 120.

Tomando el factorial como base para un ejemplo, ¿cómo podemos crear una función que calcule el factorial de un número? Bueno, existen multitud de formas. La más obvia quizá sería simplemente usar un bucle determinado para hacerlo, algo así en C:

```
long factorial(int n){
   long res = 1;
   for(int i=n; i>=1; i- -) res = res * i;
   return res;
}
```

Sin embargo hay otra forma de hacerlo sin necesidad de usar ninguna estructura de bucle que es mediante recursividad. Esta versión de la función hace exactamente lo mismo, pero es más corta, más simple y más elegante:

```
long factorial(int n)
{
    long fact;
    if (n <= 1) return 1;
    return n*factorial(n-1);
}
```

Aquí lo que se hace es que la función se llama a sí misma (eso es recursividad), y deja de llamarse cuando se cumple la condición de parada (en este caso que el argumento sea menor o igual que 1 que es lo que hay en el condicional).

Ventajas e inconvenientes ¿Ganamos algo al utilizar recursión en lugar de bucles/iteradores para casos como este?

En este caso concreto del cálculo factorial no, y de hecho es una forma más lenta de hacerlo y ocupa más memoria. Esto no es preocupante en la realidad, pero conviene saberlo. Lo del factorial es solo una forma de explicarlo con un ejemplo sencillo y que sea fácil de entender.

Pero entonces, si no ganamos nada en este caso ¿para qué sirve realmente la recursividad?

Pues para resolver ciertos problemas de manera elegante y eficiente. El ejemplo típico sería el recorrer un árbol de elementos para hacer algo con

todos ellos. Imagínate un sistema de archivos con carpetas y subcarpetas y archivos dentro de estas carpetas, o el árbol de elementos de una página Web donde unos elementos incluyen a su vez otros y no sabes cuántos hay en cada uno. En este tipo de situaciones la manera más eficiente de hacer una función que recorra todos los elementos es mediante recursión. Es decir, se crea una función que recorra todos los elementos hijo del nodo que se le pase y que se llame a sí misma para hacer lo mismo con los subnodos que haya. En el caso del sistema de archivos se le pasaría una carpeta y se llamaría a sí misma por cada subcarpeta que hubiese, y así sucesivamente con todas las demás. Con una sola llamada inicial recorrerá automáticamente toda la estructura del sistema de archivos.

Con eso, y sin necesidad de complicarse, de repente se tiene una función muy poderosa capaz de enumerar cualquier estructura arbitraria por compleja que sea. Ahí es donde se ve el verdadero poder de la recursividad, aunque hay aplicaciones más potentes y más complejas todavía.

Detalles a Tener en Cuenta Otra cosa importante a tener en cuenta es que, cada vez que se hace una llamada a una función desde otra función (aunque sea a sí misma), se crea una nueva entrada en la pila de llamadas del intérprete. Esta tiene un espacio limitado por lo que puede llegar un punto en el que si se hacen demasiadas se sature y se produzca un error. A este error se le denomina "Desbordamiento de pila" o "Stack Overflow". Ahora ya sabemos de donde viene el nombre del famoso sitio para dudas de programadores sin el que la programación moderna no sería posible.

Además, hay que tener mucho cuidado con la condición de parada. Esta se refiere a la condición que se debe comprobar para determinar que ya no se harán más llamadas a la función. Es en ese momento en el que empiezan a devolverse los valores hacia "arriba", retornando a la llamada original.

Si no tienes la condición de parada controlada pueden pasar varias cosas (todas malas), como por ejemplo:

- Que se sature la pila y se produzca un desbordamiento
- Que se ocupe cada vez más memoria
- Que se produzcan desbordamientos de variables al ir acumulando resultados.

4.6 Errores de Redondeo y de Aritmética

La aritmética que realiza una computadora es distinta de la aritmética de nuestros cursos de álgebra o cálculo. En nuestro mundo matemático tradicional consideramos la existencia de números con una cantidad infinita de cifras, en la computadora cada número representable tienen sólo un número finito, fijo de cifras (véase 4.4), los cuales en la mayoría de los casos es satisfactoria y se aprueba sin más, aunque a veces esta discrepancia puede generar problemas.

Un ejemplo de este hecho lo tenemos en el cálculo de raíces de:

$$ax^2 + bx + c = 0$$

cuando $a \neq 0$, donde las raíces se calculan comúnmente con el algoritmo:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \text{ y } x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

o de forma alternativa con el algoritmo que se obtiene mediante la racionalización del numerador:

$$x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}} \text{ y } x_2 = \frac{-2c}{b - \sqrt{b^2 - 4ac}}$$

Otro algoritmo que implementaremos es el método de Newton-Raphson[60] en su forma iterativa:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

en el cual se usa x_0 como una primera aproximación a la raíz buscada y x_n es la aproximación a la raíz después de n iteraciones (sí se converge a ella), donde $f(x) = ax^2 + bx + c$ y $f'(x_i) = 2ax + b$.

Salida del Cálculo de Raíces La implementación computacional se muestra en los programas desarrollados en distintos lenguajes (C, C++, Java y Python) usando diferentes paradigmas de programación (secuencial, procedimental y orientada a objetos), todos ellos generan la siguiente salida:

[60] También podemos usar otros métodos, como el de Newton Raphson Modificado para acelerar la convergencia

$$x_{i+1} = x_i - \frac{f(x_i)f'(x_i)}{[f'(x_i)]^2 - f(x_i)f''(x_i)}$$

que involucra la función $f(x)$, la primera derivada $f'(x)$ y a la segunda derivada $f''(x)$.

Polinomio (1.000000) X^2 + (4.000000)X + (1.000000) = 0

Chicharronera 1
Raiz (-0.2679491924311228), evaluacion raiz: -4.4408920985006262e-16
Raiz (-3.7320508075688772), evaluacion raiz: 0.0000000000000000e+00

Chicharronera 2
Raiz (-0.2679491924311227), evaluacion raiz: 0.0000000000000000e+00
Raiz (-3.7320508075688759), evaluacion raiz: -5.3290705182007514e-15

Metodo Newton-Raphson
Valor inicial aproximado de X1 = -1.2679491924311228
Raiz (-0.2679491924311227), evaluacion raiz: 0.0000000000000000e+00
Valor inicial aproximado de X2 = -4.7320508075688759
Raiz (-3.7320508075688772), evaluacion raiz: 0.0000000000000000e+00

En esta salida se muestra la raíz calculada y su evaluación en la ecuación cuadrática, la cual debería de ser cero al ser una raíz, pero esto no ocurre en general por los errores de redondeo. Además, nótese el impacto de seleccionar el algoritmo numérico adecuado a los objetivos que persigamos en la solución del problema planteado.

En cuanto a la implementación computacional, el paradigma de programación seleccionado depende la complejidad del algoritmo a implementar y si necesitamos reusar el código generado o no.

Si lo que necesitamos implementar computacionalmente es una fórmula o conjunto de ellas que generen un código de decenas de líneas, la implementación secuencial es suficiente, si es menor a una centena de líneas puede ser mejor opción la implementación procedimental y si el proyecto es grande o complejo, seguramente se optará por la programación orientada a objetos o formulaciones híbridas de las anteriores.

En última instancia, lo que se persigue en la programación es generar un código: correcto, claro, eficiente, de fácil uso y mantenimiento, que sea flexible, reusable y en su caso portable.

Ejemplo en C Usando Programación Estructurada: Escribimos el código siguiente en cualquier editor y lo llamamos *cuadratica.c:*

```
#include <stdio.h>
#include <math.h>

// Funcion cuadratica
double f(double x, double a, double b, double c) {
  return (x * x * a + x * b + c);
}

// Derivada de la funcion cuadratica
double df(double x, double a, double b) {
  return (2.0 * x * a + b);
}

// Evalua el valor X en la función cuadratica
void evalua(double x, double a, double b, double c) {
  printf("\nRaiz (%1.16f), evaluacion raiz: %1.16e", x, (a * x * x
+ b * x + c));
}

// Metodo Newton-Raphson
// x = x - f(x)/f'(x)
double metodoNewtonRapson(double x, int ni, double a, double
b, double c) {
   int i;
  for (i = 0; i < ni; i++) {
    x = x - (f(x, a, b, c) / df(x, a, b));
  }
  return x;
}

// Funcion Principal ....
int main() {
   // Coeficientes del polinomio
   double A = 1.0, B = 4.0, C = 1.0;
   // Raices del polinomio
```

```
        double X1, X2, x;
        // Calculo del discriminante
        double d = B * B - 4.0 * A * C;
        // Raices reales
        if (d >= 0.0) {
          printf("\nPolinomio (%f) X^2 + (%f )X + (%f) = 0\n", A,
B, C);
          printf("\nChicharronera 1");
          X1 = (-B + sqrt(d)) / (2.0 * A);
          X2 = (-B - sqrt(d)) / (2.0 * A);
          evalua(X1, A, B, C);
          evalua(X2, A, B, C);
          printf("\n\nChicharronera 2");
          X1 = (-2.0 * C) / (B + sqrt(d));
          X2 = (-2.0 * C) / (B - sqrt(d));;
          evalua(X1, A, B, C);
          evalua(X2, A, B, C);
          // Metodo Newton-Raphson
          printf("\n\nMetodo Newton-Raphson");
          x = X1 - 1.0;
          printf("\nValor inicial aproximado de X1 = %f", x);
          x = metodoNewtonRapson(x, 6, A, B, C);
          evalua(x, A, B, C);
          x = X2 - 1.0;
          printf("\nValor inicial aproximado de X2 = %f", x);
          x = metodoNewtonRapson(x, 6, A, B, C);
          evalua(x, A, B, C);
          printf("\n");
        } else {
          // Raices complejas
         printf("Raices Complejas ...");
        }
        return 0;
    }
```

Ejemplo en C++ Usando Programación Orientada a Objetos Escribimos el código siguiente en cualquier editor y lo llamamos *cuadratica.cpp:*

```
#include <stdio.h>
#include <math.h>
#include<iostream>
using namespace std;
class cuadratica {
 private:
    // Coeficientes del polinomio
    double A, B, C;

  // Funcion cuadratica
  double f(double x, double a, double b, double c) {
    return (x * x * a + x * b + c);
  }

  // Derivada de la funcion cuadratica
  double df(double x, double a, double b) {
    return (2.0 * x * a + b);
  }

  // Evalua el valor X en la función cuadratica
  void evalua(double x, double a, double b, double c) {
    printf("\nRaiz (%1.16f), evaluacion raiz: %1.16e", x, (a * x
* x + b * x + c));
  }

  // Metodo Newton-Raphson
  // x = x - f(x)/f'(x)
  double metodoNewtonRapson(double x, int ni, double a, double
b, double c) {
    int i;
    for (i = 0; i < ni; i++) {
      x = x - (f(x, a, b, c) / df(x, a, b));
    }
    return x;
  }
```

```
public:
  // Constructor de la clase
  cuadratica(double a, double b, double c) {
    A = a;
    B = b;
    C = c;
  }
  // calculo de raices
  void raices() {
    // Raices del polinomio
    double X1, X2, x;
    // Calculo del discriminante
    double d = B * B - 4.0 * A * C;

    // Raices reales
    if (d >= 0.0) {
      printf("\nPolinomio (%f) X^2 + (%f )X + (%f) = 0\n",
A, B, C);
      printf("\nChicharronera 1");
      X1 = (-B + sqrt(d)) / (2.0 * A);
      X2 = (-B - sqrt(d)) / (2.0 * A);
      evalua(X1, A, B, C);
      evalua(X2, A, B, C);

      printf("\n\nChicharronera 2");
      X1 = (-2.0 * C) / (B + sqrt(d));
      X2 = (-2.0 * C) / (B - sqrt(d));;
      evalua(X1, A, B, C);
      evalua(X2, A, B, C);
      // Metodo Newton-Raphson
      printf("\n\nMetodo Newton-Raphson");
      x = X1 - 1.0;
      printf("\nValor inicial aproximado de X1 = %f", x);
      x = metodoNewtonRapson(x, 6, A, B, C);
      evalua(x, A, B, C);
      x = X2 - 1.0;
      printf("\nValor inicial aproximado de X2 = %f", x);
```

```
            x = metodoNewtonRapson(x, 6, A, B, C);
            evalua(x, A, B, C);
            printf("\n");
          } else {
            // Raices complejas
            printf("Raices Complejas ...");
          }
        }
};

int main(void)
{
    cuadratica cu1 = cuadratica(1.0, 4.0, 1.0);
    cu1.raices();
    return 0;
}
```

Ejemplo en Java Usando Programación Orientada a Objetos Escribimos el código siguiente en cualquier editor y lo llamamos *cuadratica.java:*

```
import java.lang.Math;
public class cuadratica {
    // Coeficientes del polinomio
    double A, B, C, d;

    // Constructor de la clase
    public cuadratica(double a, double b, double c) {
        A = a;
        B = b;
        C = c;
    }

    // Funcion cuadratica
    public double f(double x) {
        return (x * x * A + x * B + C);
    }
```

```
        // Evalua el valor X en la función cuadratica
        public void evalua(double x) {
          System.out.println("Raiz (" + x + ") , evaluacion raiz:" +
(A * x * x + B * x + C));
        }

        // Derivada de la funcion cuadratica
        public double df(double x) {
          return (2.0 * x * A + B);
        }

        // Metodo Newton-Raphson
        // x = x - f(x)/f'(x)
        public double metodoNewtonRapson(double x, int ni) {
          for (int i = 0; i < ni; i++) {
            x = x - (f(x) / df(x));
          }
          return x;
        }

        public void raices() {
          // Raices del polinomio
          double X1, X2, x;
          // Calculo del discriminante
          d = B * B - 4.0 * A * C;
          // Raices reales
          if (d >= 0.0) {
            System.out.println("");
            System.out.println("Raices Reales (" + A + ")X^2 + ("
+ B + ")X + (" + C + ") = 0");
            System.out.println("Chicharronera 1");
            X1 = (-B + Math.sqrt(d)) / (2.0 * A);
            X2 = (-B - Math.sqrt(d)) / (2.0 * A);
            evalua(X1);
            evalua(X2);
            System.out.println("");
            System.out.println("Chicharronera 2");
            X1 = (-2.0 * C) / (B + Math.sqrt(d));
```

```
            X2 = (-2.0 * C) / (B - Math.sqrt(d));;
            evalua(X1);
            evalua(X2);
            System.out.println("");
              // Metodo Newton-Raphson
            System.out.println("Newton-Rapson");
            X1 = Math.round(X1);
            System.out.print("Valor inicial aproximado de X1 = " +
X1);
            x = metodoNewtonRapson(X1, 8);
            evalua(x);
            X2 = Math.round(X2);
            System.out.print("Valor inicial aproximado de X2 = " +
X2);
            x = metodoNewtonRapson(X2, 8);
            evalua(x);
          } else {
            // Raices complejas
            System.out.println("Raices Complejas ...");
          }
        }

        // Funcion Principal ....
          public static void main(String[] args) {
          cuadratica cu1 = new cuadratica(1.0, 4.0, 1.0);
          cu1.raices();
        }
    }
```

Ejemplo en Python Usando Programación Orientada a Objetos
Escribimos el código siguiente en cualquier editor y lo llamamos *cuadratica.py:*

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
import math
class Ejemplo:
  # Constuctor
  def __init__(self, a, b ,c):
    self.A = a;
    self.B = b;
    self.C = c;
  # Funcion cuadratica
  def f(self, x, a, b, c):
    return (x * x * a + x * b + c);
  # Derivada de la funcion cuadratica
  def df(self, x, a, b):
    return (2.0 * x * a + b);
  # Evalua el valor X en la funcion cuadratica
  def evalua(self, x, a, b, c):
    print('Raiz (%1.16f), evaluacion raiz: %1.16e' % (x, (a * x *
x + b * x + c)))
  # Metodo Newton-Raphson
  # x = x - f(x)/f'(x)
  def metodo_Newton_Rapson(self, x, ni, a, b, c):

    for i in range(ni):
      x = x - (self.f(x, a, b, c) / self.df(x, a, b))
    return x
  # Calculo de raices
  def raices(self):
    # Calculo del discriminante
    d = self.B * self.B - 4.0 * self.A * self.C

    # Raices reales
    if d >= 0.0:
```

```
            print('\nPolinomio (%f) X^2 + (%f )X + (%f) = 0\n' %
(self.A, self.B, self.C))
            print('\nChicharronera 1')
            X1 = (-self.B + math.sqrt(d)) / (2.0 * self.A)
            X2 = (-self.B - math.sqrt(d)) / (2.0 * self.A)
            self.evalua(X1, self.A, self.B, self.C)
            self.evalua(X2, self.A, self.B, self.C)
            print('\nChicharronera 2')
            X1 = (-2.0 * self.C) / (self.B + math.sqrt(d))
            X2 = (-2.0 * self.C) / (self.B - math.sqrt(d))
            self.evalua(X1, self.A, self.B, self.C)
            self.evalua(X2, self.A, self.B, self.C)
            # Metodo Newton-Raphson
            print("\n\nMetodo Newton-Raphson")
            x = X1 - 1.0;
            print("\nValor inicial aproximado de X1 = %1.16f" % x)
            x = self.metodo_Newton_Rapson(x, 6, self.A, self.B, self.C)
            self.evalua(x, self.A, self.B, self.C);
            x = X2 - 1.0;
            print("\nValor inicial aproximado de X2 = %1.16f" % x);
            x = self.metodo_Newton_Rapson(x, 6, self.A, self.B, self.C)
            self.evalua(x, self.A, self.B, self.C)
            print("\n")
        else:
            # Raices complejas
            print("Raices Complejas ...")

if __name__ == '__main__':
    ej = Ejemplo(1.0, 4.0, 1.0)
    ej.raices()
```

4.7 Documentación del Código Fuente

La documentación dentro del código fuente (véase 3.7.1) tiene como objetivo que los que lo lean, mantengan y reparen el código, lo entiendan. Así que la documentación debe tener los siguientes propósitos:

- Explicar el propósito de cada clase y comportamiento, aunque pueda parecer obvia para tí, puede no ser tan obvia para otras personas.
- Describir los parámetros que espera cada comportamiento, los valores de retorno, y las excepciones que puede lanzar.
- Si la clase o los métodos están fuertemente acoplados con otra clase o forma de llamado, es necesario mencionarlo.
- Los comentarios deben indicar: El **por qué** (razones) y **cómo funciona el código**.
- Las cadenas de documentación deben indicar: **Cómo usar** el código.

Tener cadenas de documentación o comentarios incorrectos es mucho peor que no tenerlos en absoluto. Por ello es nuestro menester mantenerlos actualizados cando se hagan cambios, asegurándonos de que los comentarios y las cadenas de documentación continúan siendo consistentes con el código, y no lo contradicen. Algunas reglas son:

1. Los comentarios no deben duplicar el código.
2. Los buenos comentarios no excusan el código poco claro.
3. Si no puede escribir un comentario claro, es posible que haya un problema con el código.
4. Los comentarios deben disipar la confusión, no causarla.
5. Explique el código unidiomático en los comentarios.
6. Proporcione enlaces a la fuente original del código copiado.
7. Incluya enlaces a referencias externas donde sean más útiles.
8. Agregue comentarios al corregir errores.

9. Utilice comentarios para marcar implementaciones incompletas.

La documentación básica en C, C++ y Java se realiza usando:

- // para comentar dentro de una línea de código:

```
for (int i = 0; i < 10; i ++) // Este ciclo se realiza 10 veces
```

- /* y */ para comentar una o más líneas:

```
/*
  Este ciclo se realiza 10 veces
*/
for (int i = 0; i < 10; i ++) xp += 10 + i;
```

La documentación básica en Python se realiza usando:

- # para comentar dentro de una línea de código:

```
for i in Array: # Este ciclo se realiza tantas veces como elementos en Array
```

- """ y """ para comentar una o más líneas:

```
"""
  Este ciclo se realiza tantas veces como elementos en Array
"""
for i in Array:
  xp = xp + i
```

Además, si se realiza la documentación con cierta estructura, esta se puede utilizar para generar un manual de referencia del código en formatos: *HTML, PDF, PS, o XML* a partir de los fuentes con unos cuantos comandos de texto en unos segundos, pues qué mejor.

Existen varias herramientas para ello, una de ellas es DOXYGEN para códigos de Java, Fortran, C y C++, en Python se puede usar *Docstring* o una cadena de documentación como se verá en la siguientes secciones.

Instalación de Doxygen Para instalar DOXYGEN usar:

```
# apt install doxygen graphviz
```

una vez instalada, hay que generar el archivo de configuración de DOXYGEN, para ello usar:

```
$ doxygen -g
```

de aquí podemos editar el archivo Doxyfile generado según las necesidades de la documentación, un ejemplo de dicha configuración para generar la salida en *HTML, LaTeX y XML* esta en:
Jerarquia de Clases DOXYGEN
Para generar la documentación de los fuentes en la carpeta donde este el archivo de configuración y los archivos fuentes, usar:

```
$ doxygen
```

La documentación generada con DOXYGEN se mostrará en carpetas separadas para cada una de las salidas seleccionadas por ejemplo: *HTML, LaTeX, XML*, etc.
Para ver la documentación generada, usar en la consola:

```
$ cd html
$ xpdf index.html
```

Para generar la documentación en formato *PDF* a partir de la salida de *LaTeX* usar:

```
$ cd latex
$ make pdf
$ xpdf refman.pdf
```

en este caso se supone que se tiene instalado *LaTeX* en la máquina, en caso contrario podemos instalar lo básico usando:

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

y adicionalmente, si se requieren otras opciones instalamos:

```
# apt install texmaker texmacs texmacs-extra-fonts texlive-
  latex-base texlive-latex-recommended myspell-en-us myspell-es
```

4.7.1 Documentar en C, C++ y Java

Hay varios estilos de documentación (véase 3.7.1), aquí ponemos una que es fácil de usar para códigos en C++, pero es lógicamente extensible a lenguajes como Java.

```
#ifndef __test__
#define __test__
/// Descripcion breve de la clase.
/**
 * Descripcion detallada de la clase ...
 *
 * @author Antonio Carrillo
 * @date Winter 2010
 * @version 0.0.1
 * @bug No errors detected
 * @warning No warnings detected
 * @todo Exception handling
 */
class test
{
 private:

 /// Descripcion breve.
 const char *nmClass;
 /**
 * Descripcion corta.
 *
 * Descripcion larga ...
 *
 * 0 = Dirichlet, 1 = Neumann (or Robin)
 */
 int bdType;



 public:
 /**
```

```
* Descipcion breve.
*
* Descripcion detallada ...
*
* Algo de LaTeX ...
*
* \f[
* |I_2|=\left| \int_{0}^T \psi(t)
* \left\{
* u(a,t)-
* \int_{\gamma(t)}^a
* \frac{d\theta}{k(\theta,t)}
* \int_{a}^\theta c(\xi)u_t(\xi,t)\,d\xi
* \right\} dt
* \right|
* \f]
*
*
* @param[out] clas Descripcion del parametro de salida
* @param[in] fun Descripcion del parametro de entrada
*/
test(const char *clas, const char *fun)
{
nameClassFunct(clas, fun);
}

/**
* Descripcion breve.
*
* Descripcion detallada
*
* @param nVert Descripcion del parametro
* @param[in] g Descripcion del parametro
* @param[in] me Descripcion del parametro
* @param[out] values Descripcion del parametro
* @param z Descripcion del parametro
* @return Descripcion de lo que regresa
*/
```

```
    int eval(int nVert, double **g, StdElem *me, double ***values,
double *z);
  };
  /**
   * Descripcion breve de la clase.
   *
   * Descripcion detallada de la clase
   *
   * Otro parrafo de la descripcion ...
   *
   * Algo de formulas con LaTeX
   *
   * \f{eqnarray*}{
   * g &=& \frac{Gm_2}{r^2} \\
   * &=& \frac{(6.673 \times 10^{-11}\,\mbox{m}^3\,\mbox{kg}^{-
1}\,
   * \mbox{s}^{-2})(5.9736 \times 10^{24}\,\mbox{kg})}{(6371.01\,\mbox{km})^2}
\\
   * &=& 9.82066032\,\mbox{m/s}^2
   * \f}
   *
   *
   * Documentacion sobre la cual se basa la clase o archivo(s) que
hagan una descripcion de la
   * misma: Archivo.doc
   *
   * Descripcion breve del ejemplo de uso de esta clase (este archivo
se supone que estara en
   * una carpeta de nombre ./Examples en la posicion actual del
código)
   *
   * Algo de LaTeX
   *
   * La distancia entre \f$(x_1,y_1)\f$ and \f$(x_2,y_2)\f$ is
\f$\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}\f$.
   *
   * @example ExampleText.cpp
   */
```

```
#endif
```

Adicionalmente es deseable que algunos comportamientos o clases tengan información adicional como son: propósito, entradas, salidas, estructuras de datos usadas en entradas y salidas, dependencia de datos o ejecución, restricciones, etc., usando una estructura como la siguiente:

```
/**
* Proposito y Metodo:
* Entradas:
* Salidas:
* Entradas TDS:
* Salidas TDS:
* Dependencias:
* Restricciones y advertencias"
*/
```

Para controlar las versiones se podría usar algo como lo siguiente:

```
/**
* @file release.notes
* @brief Package TkrRecon
* @verbatim
* Coordinator: Leon Rochester
*
* v4r4p8 09-Mar-2002 LSR Remove GFxxxxx and SiRecObjs,
no longer used
* v4r4p7 07-Mar-2002 TU Mainly, add a combo vertexing to
the TkrRecon sequence
* @endverbatim
*/
```

Un ejemplo completo puede ser el siguiente:

```
#ifndef __ErrorControl__
#define __ErrorControl__
#include <new>
using namespace std;
#include <stdlib.h>
```

```
#include "Printf.hpp"
#ifdef USE_HYPRE
#include <mpi.h>
#endif
/// Error Control, this class handles errors for the system RESSIM
/**
* @author Antonio Carrillo and Gerardo Cisneros
* @date Winter 2010
* @version 0.0.2
* @verbatim
Coordinator: Robert Yates
v0.0.1 Junuary 2011 Antonio Carrillo generates the first version of the class
v0.0.2 March 2011 Gerardo Cisneros add HYPRE errors control
Inputs: Name of class and function
Outputs: Exit of program
TDS Inputs: none
TDS Outputs: none
Dependencies: #ifdef USE_HYPRE, MPI package
Restrictions and Caveats: Non exception handling still
@endverbatim
* @bug No errors detected
* @warning No warnings detected
* @todo Exception handling
*/
class ErrorControl {
private:
/// Name of class
const char *nmClass;
/// Name of function generating the error
const char *nmFunction;
public:
/**
* Class Constructor
*/
ErrorControl(void) {
```

```
    nameClassFunct(" ", " ");
    }
    /**
    * Class Constructor
    * @param clas Class name
    */
    ErrorControl(const char *clas) {
    nameClassFunct(clas, " ");
    }
    /**
    * Class Constructor
    * @param clas Class name
    * @param fun Name of function generating the error
    */
    ErrorControl(const char *clas, const char *fun) {
    nameClassFunct(clas, fun);
    }
    /**
    * Name of class and function
    * @param clas Class name
    * @param func Name of function generating the error
    */
    void nameClassFunct(const char * clas, const char *func) {
    nameClass(clas);
    nameFunct(func);
    }
    /**
    * No memory for this request
    * @param var Var name
    */
    void memoryError(const char * var) {
    Afprintf(stderr, "\n\nNo memory for %s request in %s of class
%s\n\n", var, nmFunction, nmClass);
    fatalError(1);
    }
    /**
    * No memory for this request
    * @param var Var name
```

```
    * @param i Index number
    */
    void memoryError(const char * var, int i) {
    Afprintf(stderr, "\n\nNo memory for %s request %d in %s of
  class %s\n\n", var, i, nmFunction, nmClass);
    fatalError(1);
    }
    /**
    * No memory for this request
    * @param var Var name
    * @param func Name of function generating the error
    */
    void memoryError(const char * var, const char *func) {
    Afprintf(stderr, "\n\nNo memory for %s request in %s of class
  %s\n\n", var, func, nmClass);
    fatalError(1);
    }
    /**
    * Fatal error.
    * @param cod Error code
    */
    void fatalError(int cod) {
    Afprintf(stderr, "\nFatal Error\nEnd program\n");
    #ifdef USE_HYPRE
    MPI_Abort(MPI_COMM_WORLD, cod);
    #else
    exit(cod);
    #endif
    }
    /**
    * Fatal error.
    * @param cod Error code
    */
    void fatalError(int cod, const char *txt) {
    Afprintf(stderr, txt);
    Afprintf(stderr, "\nFatal Error\nEnd program\n");
    #ifdef USE_HYPRE
    MPI_Abort(MPI_COMM_WORLD, cod);
```

```
#else
exit(cod);
#endif
}
/**
* Set name of class
* @param clas Class name
*/
void nameClass(const char *clas) {
nmClass = clas;
}
/**
* Set name of function
* @param func Function name
*/
void nameFunct(const char *func) {
nmFunction = func;
}
};
/**
* Error Control, this class handles errors for the system RESSIM
*
* Use of the class ErrorControl for error handling within the
system RESSIM,
* for example in the error control of memory request
*
* @example ExampleErrorControl.cpp
*/
#endif
```

Más detalles sobre los parámetros en la documentación del código fuente para ser usada por DOXYGEN se pueden ver en:
http://www.stack.nl/~dimitri/doxygen/commands.html#cmdparam

4.7.2 Documentar en Python

En Python el uso de acentos y caracteres extendidos esta soportado por la codificación UTF-8 (véase 3.7.1), para ello en las primeras líneas de código es necesario usar:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

siempre y cuando se use un editor o IDE que soporte dicha codificación, en caso contrario los caracteres se perderán.

Para documentar en Python se usa un *Docstring* o cadena de documentación, esta es una cadena de caracteres que se coloca como primer enunciado de un módulo, clase, método o función, y cuyo propósito es explicar su intención. Un ejemplo sencillo (en Python 3), es:

```
def promedio(a, b):
  """Calcula el promedio de dos numeros."""
  return (a + b) / 2
```

Un ejemplo más completo:

```
def formula_cuadratica(a, b, c):
  """Resuelve una ecuación cuadratica.
  Devuelve en una tupla las dos raices que resuelven la
  ecuacion cuadratica:

  ax^2 + bx + c = 0.
  Utiliza la formula general (tambien conocida
  coloquialmente como el "chicharronero").
  Parametros:
  a – coeficiente cuadratico (debe ser distinto de 0)
  b – coeficiente lineal
  c – termino independiente
  Excepciones:
  ValueError – Si (a == 0)
  """
if a == 0:
  raise ValueError(
    'Coeficiente cuadratico no debe ser 0.')
from cmath import sqrt
discriminante = b ** 2 - 4 * a * c
x1 = (-b + sqrt(discriminante)) / (2 * a)
x2 = (-b - sqrt(discriminante)) / (2 * a)
return (x1, x2)
```

La cadena de documentación en el segundo ejemplo es una cadena multilíneas, la cual comienza y termina con triples comillas (""""). Aquí se puede observar el uso de las convenciones establecidas en el PEP 257 (Python Enhancement Proposals):

- La primera línea de la cadena de documentación debe ser una línea de resumen terminada con un punto. Debe ser una breve descripción de la función que indica los efectos de esta como comando. La línea de resumen puede ser utilizada por herramientas automáticas de indexación; es importante que quepa en una sola línea y que este separada del resto del *docstring* por una línea en blanco.

- El resto de la cadena de documentación debe describir el comportamiento de la función, los valores que devuelve, las excepciones que arroja y cualquier otro detalle que consideremos relevante.

- Se recomienda dejar una línea en blanco antes de las triples comillas que cierran la cadena de documentación.

Todos los objetos documentables (módulos, clases, métodos y funciones) cuentan con un atributo *__doc__* el cual contiene su respectivo comentario de documentación. A partir de los ejemplos anteriores podemos inspeccionar la documentación de las funciones *promedio* y *formula_cuadratica* desde el *shell* de Python:

```
>>> promedio.__doc__
'Calcula el promedio de dos numeros.'
>>> formula_cuadratica.__doc__
'Resuelve una ecuación cuadratica.\n\n Devuelve en una
tupla las dos raices que resuelven la\n ecuacion
cuadratica:\n \n ax^2 + bx + c = 0.\n\n
Utiliza la formula general (tambien conocida\n
coloquialmente como el "chicharronero").\n\n
Parametros:\n a – coeficiente cuadratico (debe ser
distinto de 0)\n b – coeficiente lineal\n
c – termino independiente\n\n Excepciones:\n
ValueError – Si (a == 0)\n \n '
```

Sin embargo, si se esta usando el *shell* de Python es mejor usar la función *help()*, dado que la salida producida queda formateada de manera más clara y conveniente:

```
>>> help(promedio)
Help on function promedio in module __main__:
promedio(a, b)
Calcula el promedio de dos numeros.
>>> help(formula_cuadratica)
Help on function formula_cuadratica in module __main__:
formula_cuadratica(a, b, c)
Resuelve una ecuacion cuadratica.
Devuelve en una tupla las dos raices que resuelven la
ecuacion cuadratica:
ax^2 + bx + c = 0.
Utiliza la formula general (tambien conocida
coloquialmente como el "chicharronero").
Parametros:
a – coeficiente cuadratico (debe ser distinto de 0)
b – coeficiente lineal
c – termino independiente
Excepciones:
ValueError – Si (a == 0)
```

Ciertas herramientas, por ejemplo *shells* o editores de código, pueden ayudar a visualizar de manera automática la información contenida en los comentarios de documentación. De esta manera un usuario puede tener a su alcance de manera sencilla toda la información que necesita para poder usar nuestras funciones.

Generando documentación en páginas de HTML Los *Docstrings* se pueden usar también para producir documentación en páginas de HTML que pueden ser consultadas usando un navegador de Web. Para ello se usa el comando *pydoc* desde una terminal. Por ejemplo, si las dos funciones anteriores (*promedio* y *formula_ cuadratica*) se encuentran en un archivo fuente llamado *ejemplos.py*, podemos ejecutar el siguiente comando en una terminal dentro del mismo directorio donde esta el archivo fuente:

```
pydoc -w ejemplos
```

La salida queda en el archivo *ejemplos.html*, y así se visualiza desde un navegador. La documentación de *pydoc* explica otros artilugios que puede hacer esta utilería de Python.

Docstrings vs. Comentarios Un comentario en Python inicia con el símbolo de número (#) y se extiende hasta el final de la línea. En principio los *Docstrings* pudieran parecer similares a los comentarios, pero hay una diferencia pragmática importante: los comentarios son ignorados por el ambiente de ejecución de Python y por herramientas como *pydoc*; esto no es así en el caso de los *Docstrings*.

A un nivel más fundamental hay otra diferencia aún más grande entre los *Docstrings* y los comentarios, y esta tiene que ver con la intención:

- Los *Docstrings* son documentación, y sirven para entender qué hace el código.
- Los comentarios sirven para explicar cómo lo hace.

La documentación es para la gente que usa tu código. Los comentarios son para la gente que necesita entender cómo funciona tu código, posiblemente para extenderlo o darle mantenimiento.

El uso de *Docstrings* en Python facilita la escritura de la documentación técnica de un programa. Escribir una buena documentación requiere de disciplina y tiempo, pero sus beneficios se cosechan cuando alguien -quizás mi futuro yo dentro de seis meses- necesita entender qué hacen nuestros programas. Los *Docstrings* no sustituyen otras buenas prácticas de programación, como son el uso apropiado de comentarios o el empleo de nombres descriptivos para variables y funciones.

5 Programación en Paquetes de Cálculo Numérico

El cálculo numérico permite diseñar e implementar algoritmos para que a través de números y reglas matemáticas se pueden simular procesos matemáticos complejos aplicados a la resolución de problemas de Ciencias e Ingenierías. De esta forma los paquetes de cálculo numérico proporcionan todo el andamiaje para llevar a cabo todos aquellos procedimientos matemáticos susceptibles de expresarse algorítmicamente que permitan su simulación o cálculo en procesos más sencillos empleando números.

Los paquetes de cálculo numérico, son programas matemáticos que ofrecen un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio con un amplio abanico de herramientas numéricas para la lectura, manipulación, análisis y graficación de datos. Entre sus prestaciones básicas se hallan:

- Manejo de números reales y complejos
- La manipulación de vectores y matrices tanto reales como complejas
- Manejo de funciones elementales y especiales
- Resolución de problemas de álgebra lineal
- Resolución de ecuaciones no lineales
- La representación de datos y funciones
- La implementación de algoritmos
- Integración de funciones
- Máximos y mínimos de funciones
- Manipulación de polinomios
- Integración de ecuaciones diferenciales
- Graficación de funciones en 2D y 3D
- La comunicación con programas en otros lenguajes de programación y con otros dispositivos de Hardware.

El programa comercial líder para el cálculo numérico es MATLAB (véase [29]) de la compañía MathWorks, este paquete salió a la venta en el año de 1984 con la versión 1.0 y casi año con año, ha generado nuevas versiones de su paquete y múltiples sistemas de licenciamiento.

La idea detrás de paquetes como MATLAB es la de emplear grupos de subrutinas escritas en principio en el lenguaje de programación FORTRAN como son las librerías LINPACK (véase [44]) y EISPACK (véase [45]) para manipular matrices y vectores y proporcionar un sencillo acceso a dicho Software y así, entre otras cosas, ser usado en resolución de problemas de álgebra lineal, análisis numérico, ecuaciones diferenciales parciales y ordinarias, sin necesidad de escribir programas en lenguajes de bajo nivel.

Estos paquetes, pueden disponer de herramientas adicionales que expanden sus prestaciones, como son el diseño y simulación de sistemas de control. Además, se pueden ampliar las capacidades base de dichos programas con las cajas de herramientas y con los paquetes de bloques. En algunos casos existen versiones para cómputo secuencial y paralelo -tanto en memoria compartida como distribuida, también para usar los múltiples Cores gráficos CUDA (GPUs) de las tarjetas NVIDIA (véase **??**)-.

Los paquetes de cómputo para el Cálculo Numérico más usados actualmente son:

- MATLAB (véase [29]) «abreviatura de MATrix LABoratory (laboratorio de matrices)» es un Software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M). Entre sus prestaciones básicas se hallan: la mani-pulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispo-sitivos Hardware. El paquete MATLAB dispone de dos herramientas adicionales que expanden sus prestaciones, a saber, una plataforma de simulación multidominio Simulink (véase [47]) y un editor de interfaces de usuario GUIDE. Además, se pueden ampliar las capacidades de MATLAB con su caja de herramientas (Tool Box) y las de Simulink con los paquetes de bloques.

- Octave (véase [32]) es un programa de cálculo numérico de licencia GNU (véase [7]), conocido por buscar una sintaxis similar a la de MATLAB, existiendo una gran compatibilidad en las funciones de cálculo numérico.

- FreeMat (véase [31]) es un programa de cálculo numérico de licencia GPL (véase [7]) que proporciona un ambiente libre para el rápido desarrollo de prototipos para la Ciencia y la Ingeniería, además del procesamiento de datos. Es similar a MATLAB y Octave, pero cuenta con una interfaz externa de código en los lenguajes de programación C, C++ y Fortran, incluso distribuye el desarrollo de algoritmos en paralelo con la interfaz de paso de mensajes llamada MPI.

- Scilab (véase [30]) es un programa de cálculo numérico de licencia CeCILL compatible con GPL (véase 11.2), desarrollado principalmente en Francia, que incluye su propia librería para gráficos. Es similar a MATLAB, pero no busca una compatibilidad total, como lo puede hacer FreeMat y Octave. Scilab tiene una herramienta para el diseño y simulación de sistemas de control Scicos (véase [46]) similar a Simulink (véase [47]) de MATLAB.

- Julia (véase [**?**]) es un lenguaje de programación homoicónico, multiplataforma y multiparadigma de tipado dinámico de alto nivel y alto desempeño para la computación genérica, técnica y científica, con una sintaxis similar a la de otros entornos de computación similares, con licencia MIT (véase 11.2).

  Dispone de un compilador avanzado (JIT), mecanismos para la ejecución en paralelo y distribuida, además de una extensa biblioteca de funciones matemáticas. La biblioteca, desarrollada fundamentalmente en Julia, también contiene código desarrollado en C o Fortran, para el álgebra lineal, generación de números aleatorios, procesamiento de señales, y procesamiento de cadenas. Adicionalmente, la comunidad de desarrolladores de Julia contribuye con la creación y distribución de paquetes externos a través del gestor de paquetes integrado de Julia a un paso acelerado. IJulia es el resultado de la colaboración entre las comunidades de IPython y Julia, provee de una poderosa interfaz gráfica basada en el navegador Web para Julia.

- Scipy (véase [33]) es una librería de herramientas numéricas para Python (véase [34]) con licencia Open Source (véase [7]). En su filosofía no ha tratado de imitar a ninguno de los paquetes anteriores y tiene detrás el respaldo de un auténtico lenguaje de programación orientado a objetos e interpretado, que también puede ser compilado para ganar velocidad

en la ejecución. Este hecho le confiere una gran potencia y la capacidad de beneficiarse de las mejoras del lenguaje base.

Existen otros paquetes que pueden ser usados en el cálculo numérico -estos poseen características que enriquecen las opciones clásicas de los paquetes de cálculo numérico-:

- R (véase [39])
- Maple (véase [63])
- Mathematica (véase [64])
- Maxima (véase [65])

Por otro lado existe Anaconda, una Suite de código abierto que abarca una serie de aplicaciones, librerías y conceptos diseñados para el desarrollo de la Ciencia de Datos con Python. En líneas generales Anaconda Distribution es una distribución de Python que funciona como un gestor de entorno, un gestor de paquetes y que posee una colección de más de 720 paquetes de código abierto. Anaconda Distribution se agrupa en 4 sectores o soluciones tecnológicas, Anaconda Navigator, Anaconda Project, las librerías de Ciencia de Datos y Conda. Todas estas se instalan de manera automática y en un procedimiento muy sencillo. Para más información ver: https://www.anaconda.com/.

También esta SageMath, una Suite de código abierto bajo la licencia GPL de Software matemático como: NumPy, SciPy, Matplotlib, Sympy, Maxima, GAP, FLINT, R, entre otros. Además combina acceso a una poderosa combinación del lenguaje basada en Python o directamente vía interfaces o Wrappers. La misión del proyecto es crear una alternativa de Software libre a Magma, Maple, Mathematica y Matlab. Para más información ver: http://www.sagemath.org/.

5.1 Cálculo Numérico con Octave

El paquete Octave (véase [32]), al mantener una sintaxis similar a la del paquete MATLAB (véase [29]) tiene cientos de características, tan variadas como el segmento de usuarios al que dicho Software está dirigido. Al ser un

paquete tan completo, es difícil que un usuario promedio use las características avanzadas de dicho paquete; esto repercute en que el usuario promedio pague un alto costo por el valor de las licencias de uso sin usar dichas características; y esto contrasta con un vertiginoso desarrollo de nuevas características, que permite a la compañía lanzar una o más versiones por año, cada una con múltiples opciones de licenciamiento, según las necesidades del segmento al que están dirigidas.

Las múltiples características y los miles de usuarios, han creado una comunidad robusta, la que ha logrado una gran cantidad de funciones optimizadas, que han permitido la publicación de decenas libros, cientos de artículos y miles de páginas Web en los cuales se muestra como resolver diversos problemas concomitantes en Ciencias e Ingenierías usando dicho paquete y la interacción con otros lenguajes como son C/C++, Fortran o Java.

Entre las principales aplicaciones de Octave/MATLAB incluyen la de métodos secuenciales y paralelos para resolver problemas de álgebra lineal con matrices (ralas, dispersas y densas), estadística, análisis de Fourier, optimización, integración numérica, resolución de ecuaciones diferenciales ordinarias y parciales, creación de gráficos y visualización de datos. Además de opciones para hacer interpolación y regresión de datos, cálculo de eigenvalores y valores singulares, etc.

Entornos de Programación Uno de los aspectos más agradables de Octave/MATLAB es su entorno de programación, que permite centralizar la información en un entorno de ventanas. El depurador está también incorporado en el editor.

5.2 Trabajando con Octave

GNU Octave es un lenguaje de alto nivel destinado para el cálculo numérico. Provee una interfaz sencilla, orientada a la línea de comandos (consola), que permite la resolución de problemas numéricos, lineales y no lineales, además permite la ejecución de scripts y puede ser usado como lenguaje orientado al procesamiento por lotes.

Octave posee una gran cantidad de herramientas que permiten resolver problemas de álgebra lineal, cálculo de raíces de ecuaciones no lineales, integración de funciones ordinarias, manipulación de polinomios, integración de ecuaciones diferenciales ordinarias y ecuaciones diferenciales algebraicas. Sus funciones también se pueden extender mediante funciones definidas por

el usuario escritas en el lenguaje propio de Octave o usando módulos dinámicamente cargados escritos en lenguajes como C, C++ y Fortran entre otros.

Cálculos básicos Vamos a empezar a trabajar con la tarea más básica que podemos darle a esta herramienta, vamos a utilizarla como una calculadora. A continuación, veamos un ejemplo sencillo de suma, multiplicación y división:

```
octave:1> 45 + 76 * 12.5 / 1.5
```

De igual manera también podemos calcular promedios de la siguiente forma:

```
octave:1> (12+15+20)/3
```

Para calcular potencias utilizamos el símbolo "^" o doble asterisco "**", ejemplo:

```
octave:1> 2^8
```

Octave, como toda calculadora cumple con cierta regla de precedencia para el uso de operadores, esta es: las expresiones se evaluarán de izquierda a derecha, la potencia tendrá el mayor orden de precedencia, seguido de la multiplicación y división, y con la suma y resta como los operadores con menor precedencia. No es lo mismo: "4 + 5 / 4 - 3" que "(4 + 5) / (4 - 3)". En la primera expresión se evaluará primero la división entre 5 y 4 y luego se le sumará (restará) el 4 y el 3, en la segunda expresión primero se evaluarán las operaciones dentro de los paréntesis y luego se dividirá el resultado de estos valores.

Variables Las variables son identificadores que permiten almacenar datos, los cuales pueden cambiar durante la ejecución de un programa. Las variables nos permiten asignarle nombres a los valores para luego poder hacer referencia a estos. No hay límite para la longitud en el nombre de una variable, pero estos deben estar constituidos por una secuencia de letras, undersCores (símbolo de subrayado o guión bajo) o números y sólo pueden empezar con letras o undersCores.

Vectores y Matrices En el área de la computación un vector generalmente es definido como un arreglo, es decir, un conjunto de datos a los cuales se accede por medio de índices. Un vector es la forma más simple de una matriz, podemos decir que es una matriz de una dimensión. En Octave disponemos de una gran variedad de formas para definir vectores y matrices, usualmente lo hacemos encerrando los elementos dentro de corchetes, los elementos separados por espacios o comas (,) definen un vector fila, los elementos separados por el retorno de carro o por punto y coma (;) definen un nuevo vector fila en la matriz, ejemplo:

```
octave:1> a=[1,3,5]
octave:2> b=[1;2;3]
octave:3> e=[a ; b ; 2, 4 6]
```

Para definir la matriz identidad, por ejemplo de tamaño 3, hacemos:

```
octave:1> E=eye(3)
```

a partir de esta matriz, podemos definir un vector de ella, mediante:

```
octave:1> y=E(:,1)
```

ahora podemos hacer la multiplicación matriz vector, mediante:

```
v=E*v
```

Para definir una matriz de unos de tamaño 3, hacemos:

```
octave:1> A=ones(3)
```

podemos calcular la inversa de una matriz, para ello construyamos la matriz a invertir, mediante:

```
octave:1>A=ones(3)+eye(3)
octave:2> C=inv(A)
```

Por ejemplo para resolver un sistema lineal, hacemos:

```
octave:1> A=ones(3)+eye(3)
octave:2> b=A(:,3)
octave:3> x=inv(A)*b
```

o de forma alternativa, podemos hacer eliminación gaussiana si A es cuadrada y hacer:

```
octave:3> x=A\b
```

Podemos calcular la transpuesta a un vector mediante el símbolo ', por ejemplo:

```
octave:1> a=[0 0 1]'
```

Para sustituir los valores de una fila o columna por los de un vector, hacemos:

```
octave:1> A(3,:)=v
octave:2> A(:,2)=w
```

Para intercambiar, por ejemplo las filas 2 y 3 de A, de la siguiente forma:

```
octave:1> A([2 3],:)=A([3 2],:)
```

Algunas otras matrices de uso común son:

- *diag(v)* genera una matriz diagonal con el vector v como diagonal
- *toeplitz(v)* define una matriz simétrica de diagonal constante con v como primera fila y primera columna
- *toeplitz(w,v)* define una matriz simétrica de diagonal constante con w como primera columna y v como primera fila
- *ones(n)* genera una matriz de $n \times n$ con todos los valores iguales a uno
- *zeros(n)* genera una matriz de $n \times n$ con todos los valores iguales a cero
- eye(n) genera una matriz identidad de $n \times n$
- *rand(n)* genera una matriz de $n \times n$ con elementos de valor aleatorio entre 0 y 1 (distribución uniforme)
- *randn(n)* genera una matriz de $n \times n$ cuyos elementos siguen una distribución normal (media 0 y varianza 1)

- *ones(m,n), zeros(m,n), rand(m,n)* generan matrices de $m \times n$
- *ones(size(A)), zeros(size(A)), yes(size(A))* generan matrices de la misma forma que A
- *triu(A)* coloca ceros en todos los elementos por debajo de la diagonal (triangular superior)
- *tril(A)* coloca ceros en todos los elementos por encima de la diagonal (triangular inferior)
- *inv(A)* calcula la inversa de A si es invertible
- *pinv(A)* calcula la pseudoinversa de A
- *det(A)* calcula el determinante (si A es una matriz cuadrada)
- *rank(A)* es el rango (número de pivotes=dimensión del espacio de filas y del espacio de columnas)
- *trace(A)* es la traza, suma de autovalores
- *null(A)* es una matriz cuyas columnas $n-r$ forman una base ortogonal para el espacio nulo de A
- *orth(A)* es la matriz cuyas columnas r forman una base ortogonal para el espacio de columnas de A

Secuencias Una forma sencilla de producir una secuencia de números es utilizando la notación *n:m* donde n es el número inicial y m el final, ejemplo:

```
octave:1> 1:10
```

también podemos usar la notación *p:q:r* para crear una secuencia que inicia en p, finaliza en r con intervalos de q. En el siguiente caso almacenaremos en una variable b una secuencia partiendo de cero y finalizando en diez, con un intervalo de dos entre cada número, ejemplo:

```
octave:1> b=0:2:10
```

Funciones matemáticas Octave incluye una serie de funciones matemáticas y trigonométricas que nos ayudan a simplificar algunos cálculos, la lista completa está disponible en la ayuda del programa[61]. Las funciones pueden contener otras funciones como argumentos:

```
octave:1> sqrt( round( 25/3 ) + 1 )
```

en Octave, se asume que los argumentos de funciones trigonométricas están en radianes por lo que si queremos calcular los grados de alguna función trigonométrica debemos aplicar la siguiente fórmula:

```
octave:1> sin(90*pi/180)
```

Sistema de ecuaciones Para la resolución de sistemas de ecuaciones del tipo $Ax = b$ utilizamos la notación $a\backslash b$, por ejemplo, para calcular el siguiente sistema de ecuaciones de primer grado con dos incógnitas:

$$6\cdot x - 7\cdot y = 5$$
$$8\cdot x - 9\cdot y = 7$$

para resolver, haremos lo siguiente: guardaremos los elementos x e y en una matriz a y la igualdad en un vector b, para finalmente ejecutar el comando $a\backslash b$, de la siguiente forma:

```
octave:1> a=[6, -7;8,-9]
octave:2> b=[5;7]
octave:3> a\b
```

Funciones definidas por el usuario En Octave podemos crear nuestras propias funciones, podemos escribirlas directamente desde la línea de comandos o ejecutarlas desde un archivo externo. Las funciones que creamos desde la línea de comandos deben cumplir con el siguiente formato:

```
function variable_salida = nombre_funcion (argumentos_entrada)
  cuerpo _funcion
endfunction
```

[61] Alguna de ellas son: sin, sinh, asin, asinh, cos, cosh, acos, acosh, tan, tanh, atan2, atanh, sec, sech, asec, asech, csc, csch, acsc, acsch, cot, coth, acot, acoth, exp, log, log10, pow2, sqrt, fix, floor, ceil, round, mod, rem, sig, etc.

en caso de devolver varias variables, estas deben estar encerradas entre corchetes "[ ]":

```
function [salida1,salida2] = nombre_funcion (argumentos_entrada)
  cuerpo _funcion
endfunction
```

Por ejemplo, crearemos una función que calcula *seno(x)* en grados:

```
function s = sind(x)
  %SIND(X) Calcula seno(x) en grados
  s = sin(x*pi/180);
endfunction
```

y ejecutamos:

```
octave:|> sind(45)
```

Para que Octave ejecute un archivo o script, este debe tener extensión *.m* y debe encontrarse en el directorio desde donde estemos ejecutando Octave. También podemos agregar rutas adicionales con el comando *addpath('ruta')*, donde ruta es el camino al directorio que contiene los scripts. Para borrar la ruta usamos *rmpath('ruta')*.

Por ejemplo, el cálculo de las raíces de la ecuación cuadrática (véase 4.6), podemos definir en el archivo *cuadratica.m*, el siguiente código:

Ejemplo en Octave/MatLab:

```
% Calculo de raices de una ecuacion cuadratica
function[x1,x2] = raiz(a,b,c)
    fprintf('Polinomio(%1.6f) X^2 + (%1.6f) X + (%1.6f) =
0\n',a,b,c);
    x1 = 0;
    x2 = 0;
    if a == 0
      fprintf('\nNo es cuadratica\n')
      return
    end
    fprintf('\nChicharronera 1\n')
```

```
    x1 = (-b+sqrt(b*b-4*a*c))/(2*a);
    r = f(a, b, c, x1);
    fprintf('Raiz(%1.16f), evaluacion raiz: (%e)\n',x1,r);
    x2 = (-b-sqrt(b*b-4*a*c))/(2*a);
    r = f(a, b, c, x2);
    fprintf('Raiz(%1.16f), evaluacion raiz: (%e)\n',x2,r);
    fprintf('\nChicharronera 2\n')
    x1 = (-2*c)/(b+sqrt(b*b-4*a*c));
    r = f(a, b, c, x1);
    fprintf('Raiz(%1.16f), evaluacion raiz: (%e)\n',x1,r);
    x2 = (-2*c)/(b-sqrt(b*b-4*a*c));
    r = f(a, b, c, x2);
    fprintf('Raiz(%1.16f), evaluacion raiz: (%e)\n',x2,r);
    fprintf('\nMetodo Newton-Raphson\n')
    x = x1 -1;
    fprintf('Valor inicial aproximado de X1 = %1.16f\n',x);
    for i = 1:10
      x = x - (f(a, b, c, x) / df(a, b, c, x));
    end
    r = f(a, b, c, x);
    fprintf('Raiz (%1.16f), evaluacion raiz: (%e)\n',x,r);
    x = x2 -1;
    fprintf('Valor inicial aproximado de X2 = %1.16f\n',x);
    for i = 1:10
      x = x - (f(a, b, c, x) / df(a, b, c, x));
    end
    r = f(a, b, c, x);
    fprintf('Raiz (%1.16f), evaluacion raiz: (%e)\n',x,r);
end
% Funcion cuadratica
function r = f(a, b, c, x)
    r = x * x * a + x * b + c;
end
% Derivada de la funcion cuadratica
function r = df(a, b, c, x)
    r = 2.0 * x * a + b;
end
```

y podemos ejecutar mediante:

[x1,x2] = cuadratica(1,4,1)

para el polinomio $x^2 + 4x + 1 = 0$, con la salida mostrada en 4.6.

Gráficos 2D en Octave La forma más simple para producir gráficas bidimensionales en Octave se hace a través del comando *plot(x[,y][,fmt])*, donde x y opcionalmente y representan los vectores de coordenadas para cada punto, adicionalmente se le puede especificar una serie de formatos que cambiarán la apariencia de salida de la gráfica, esto incluye el estilo de línea, el color y otras características que explicaremos en los siguientes ejemplos. En este primer ejemplo crearemos un vector aleatorio de 50 elementos que luego graficaremos:

```
octave:1> x = (rand(50,1));
octave:2> plot(x)
```

acto seguido se abre una ventana mostrándonos la gráfica resultante.

Gráficos 3D en Octave Para producir gráficos en 3D disponemos de varias opciones, la más simple es usar el comando *plot3(x,y,z)* donde cada argumento es tomado para convertirse en los vértices del gráfico tridimensional. Si todos los argumentos son vectores de la misma longitud se dibujará una única línea continua. En caso de que todos los argumentos sean matrices cada una de las columnas de las matrices serán tratadas como líneas separadas. En caso de que sólo se le pasen dos argumentos en lugar de tres, *plot3(x,c)*, el segundo argumento 'c' debe ser un número complejo, así, las partes reales e imaginarias de este son usadas como las coordenadas y e z respectivamente. Si sólo se le pasa un argumento, *plot3(c)*, las partes reales e imaginarias de los argumentos son usados como los valores y y z, y se trazan frente a su índice.

El comando *plot3* también acepta los argumentos que permiten modificar el formato de presentación de la gráfica descritos en la sección de gráficas bidimensionales, ejemplo:

```
octave:1> t=0:pi/50:10*pi;
octave:2> plot3(sin(t),cos(t),t)
octave:3> grid on
octave:4> axis square
```

acto seguido se abre una ventana mostrándonos la gráfica resultante.

5.3 Desde la Nube

Existen diferentes servicios Web[62] que permiten editar, compilar y ejecutar código de diversos lenguajes y paquetes desde el navegador, esto en aras de que los estudiantes y profesores que cuenten con algún sistema de acceso a red y un navegador puedan programar en los más diversos lenguajes, IDEs y terminales sin hacer instalación alguna en su equipo de cómputo, tableta o teléfono celular.

Algunos ejemplos de estos servicios son:

- Para Octave https://octave-online.net/
- Para Octave https://rextester.com/l/octave_online_compiler
- Para Octave http://www.compileonline.com/execute_matlab_online.php
- Para Octave http://www.tutorialspoint.com/octave_terminal_online.php
- Para Octave https://www.jdoodle.com/execute-octave-matlab-online
- Para SciLab http://cloud.scilab.in/
- Para SciLab http://hotcalcul.com/on-line-calculator/scilab
- Para Scipy https://www.jdoodle.com
- Para Scipy https://try.jupyter.org/
- Para Scipy http://browxy.com

[62] Cuando se trabaja desde la Web es recomendable usar el modo Privado o Incógnito para no guardar el historial de navegación, información introducida en los formularios y borrar al cerrar el navegador los datos de los sitios visitados. Pero recuerda que los sitios Web que visitamos sí guardan información de nuestra visita, nuestro proveedor de Internet también guarda constancia de nuestra visita y si descargamos algo, esto no se borra al igual que el historial de descargas, además de las marcas de páginas o favoritos se conservarán al cerrar el navegador.

6 Programación en Paquetes de Cálculo Simbólico

En matemáticas y ciencias de la computación, el cálculo simbólico, también conocido como cálculo algebraico o álgebra computacional, es un área científica que se refiere al estudio y desarrollo de algoritmos y Software para la manipulación de expresiones matemáticas y otros objetos matemáticos. Aunque, hablando con propiedad, el álgebra computacional debe ser un subcampo de la computación científica, ellos son considerados generalmente como campos distintos, porque la computación científica se basa generalmente en el análisis numérico con números aproximados en punto flotante; mientras que, el álgebra computacional enfatiza el cálculo exacto con expresiones que contengan variables que no tienen cualquier valor dado y por lo tanto son manipulados como símbolos (de ahí se debe el nombre de cálculo simbólico).

Las aplicaciones de Software que realizan cálculos simbólicos son conocidos como sistemas de álgebra computacional, con el término sistema aludiendo a la complejidad de las principales aplicaciones que incluyen, al menos, un método para representar los datos matemáticos en una computadora, un lenguaje de programación de usuario (por lo general diferente del lenguaje usado para la ejecución), un administrador de memoria, una interfaz de usuario para la entrada/salida de expresiones matemáticas, un gran conjunto de subrutinas para realizar operaciones usuales, como la simplificación de expresiones, la regla de la cadena utilizando diferenciación, factorización de polinomios, integración indefinida, etc.

El álgebra computacional es ampliamente utilizado para experimentar en matemáticas y diseñar las fórmulas que se utilizan en los programas numéricos. También se usa para cálculos científicos completos, cuando los métodos puramente numéricos fallan, como en la criptografía asimétrica o para algunos problemas no lineales.

Los paquetes de cálculo simbólico, son programas matemáticos que ofrecen un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio con un amplio abanico de herramientas simbólicas y numéricas para la lectura, manipulación, análisis y graficación de datos. Entre sus prestaciones básicas se hallan:

- Bibliotecas de funciones matemáticas elementales y especiales
- Matrices y manipulación de datos

- Soporte para números complejos, precisión arbitraria, computación de intervalos aritméticos y simbólicos
- Datos en 2D y 3D, función y visualización geográfica y herramientas de animación
- Solucionadores para sistemas de ecuaciones, ecuaciones diofánticas, ecuaciones diferenciales ordinarias, parciales, diferenciales algebraicas, de retraso, diferenciales estocásticas y relaciones de recurrencia
- Herramientas numéricas y simbólicas para cálculo de variable continua o discreta
- Bibliotecas de Estadística multivariable, incluyendo ajuste, pruebas de hipótesis, y cálculos de probabilidad
- Soporte para datos censurados, datos temporales, series temporales y datos basados en unidades
- Cálculos y simulaciones en procesos aleatorios y colas
- Geometría computacional en 2D, 3D y dimensiones mayores
- Optimización restringida y no restringida, local y global
- Herramientas para problemas combinatorios
- Soporta el desarrollo de cálculos matemáticos de manera simbólica y numérica con precisión arbitraria
- Librerías para funciones matemáticas básicas y avanzadas
- Manejo de números complejos y sus diversas operaciones
- Aritmética, álgebra, operaciones para desarrollo de polinomios multivariados
- Limites, series y sucesiones
- Álgebra diferencial
- Herramientas para la manipulación de matrices incluyendo matrices dispersas

- Sistemas de solución para ecuaciones diferenciales en sus diferentes variedades (ODE, DAE, PDE, DDE)
- Herramientas simbólicas y numéricas para cálculo discreto y continuo, incluye integración definida e indefinida, diferenciación
- Optimización con restricciones y sin restricciones
- Herramientas para la resolución de problemas en el campo de la probabilidad
- Herramientas para el uso de series de tiempo
- Conexión a datos en línea, recopilados para aplicaciones financieras y económicas
- Herramientas para cálculos financieros, incluyendo: bonos, anualidades entre otros
- Cálculos y simulaciones para procesos aleatorios
- Herramientas para el desarrollo de sistemas lineales y no lineales
- Incluye matemáticas discretas

Los programas líderes para el cálculo simbólico son Mathemática y Maple, los costos de sus licencias respectivas son onerosos, pero para la UNAM se tienen licencias estudiantiles disponibles para profesores y estudiantes. Existen otros paquetes como Maxima que tiene licencia GPL y están disponibles para los usuarios en múltiples plataformas.

6.1 Cálculo Simbólico con Maxima

El sistema de álgebra computacional Maxima[63] es un motor de cálculo simbólico escrito en lenguaje Lisp publicado bajo licencia GNU GPL. Cuenta con un amplio conjunto de funciones para hacer manipulación simbólica de polinomios, matrices, funciones racionales, integración, derivación, manejo de gráficos en 2D y 3D, manejo de números de coma flotante muy grandes,

[63] Véase https://es.wikipedia.org/wiki/Maxima

expansión en series de potencias y de Fourier, entre otras funcionalidades. Además tiene un depurador a nivel de fuente para el código de Maxima.

Maxima está basado en el sistema original de Macsyma desarrollado por MIT en los años 70. Es bastante fiable, tiene un buen recolector de basura, por lo que no desperdicia memoria. Viene con cientos de auto pruebas (test-suite). Maxima funciona en modo consola, sin embargo incluye las interfaces gráficas xMaxima y wxMaxima para facilitar su uso.

El editor de texto científico GNU TeXmacs también puede ser usado para facilitar una interfaz gráfica de usuario para Maxima. Otras opciones son, *Imaxima*, y el modo interactivo de *Emacs*. También puede hacer uso de la interfaz gráfica de *SageMath*, que facilita su integración con otras herramientas.

Como está escrito en *Common Lisp*, es fácilmente accesible para la programación, desde la capa inferior de Lisp puede llamarse a *Maxima*. Como la mayoría de sistemas algebraicos, *Maxima* se especializa en operaciones simbólicas. También ofrece capacidades numéricas especiales, como son los números enteros y racionales, los cuales pueden crecer en tamaño sólo limitado por la memoria de la máquina; y números reales en coma flotante, cuya precisión puede ser arbitrariamente larga (*bfloat*). Permite el manejo de expresiones simbólicas y numéricas, y además produce resultados con una alta precisión.

Para cálculos intensivos en reales de coma flotante, Maxima ofrece la posibilidad de generar código en otros lenguajes de programación, como Fortran, que quizá se ejecuten de manera más eficiente.

Maxima es un sistema de propósito general; como tal los cálculos especiales como la factorización de números grandes, la manipulación de polinomios extremadamente grandes, etc. son normalmente realizados de forma más eficiente y rápida en sistemas especializados.

- Números
- Listas, arreglos y matrices
- Transformaciones algebraicas
- Resolución de ecuaciones
- Límites, derivadas e integrales
- Conjuntos

- Vectores y campos
- Gráficos
- Ecuaciones diferenciales
- Probabilidades y análisis de datos
- Interpolación numérica
- Inecuaciones racionales
- Ecuaciones diferenciales ordinarias
- Sistemas de ecuaciones diferenciales ordinarias
- Series de potencias
- Transformada de Laplace
- Ecuaciones recurrentes
- Generación de expresiones en Tex
- Programación

6.2 Trabajando con Maxima

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

Operaciones Aritméticas Para iniciar, podemos usar a Maxima como una calculadora, por ejemplo (34*6)/34, entonces escribimos:

```
(34*6)/34;
```

y el paquete nos dará el resultado, las operaciones soportadas son +,-,*,/ y ^. Además puede trabajar con números enteros tan grandes como se desee, por ejemplo para calcular el factorial de 200, hacemos:

200!;

Maxima devuelve los resultados de forma exacta sin aproximaciones decimales, por ejemplo para resolver la siguiente operación:

$$\left[\left(\frac{3^7}{4+\frac{3}{5}}\right)^{-1}+\frac{7}{9}\right]^3$$

escribimos:

((3^7/(4+3/5)^(-1)+7/9)^3;

y obtenemos el resultado de forma simplificada. Si lo necesitamos podemos pedir el resultado en forma decimal; por ejemplo si queremos la expresión decimal del último resultado, escribimos:

float(%);

Maxima puede trabajar con precisión arbitraria. Para calcular el valor del cociente $\frac{e}{\pi}$ con cien cifras decimales, debemos especificar primero la precisión requerida asignándole a la variable *fpprec* el valor 100 y a continuación realizar el cálculo, solicitando la expresión decimal con una llamada a la función *bfloat*, de esta forma:

fpprec:100$ bfloat(%e / %pi);

aquí se usa el símbolo de *$* como delimitador entre instrucciones.

Factorización Maxima permite factorizar un número como el factorial de doscientos (200!) en sus factores primos, sólo hacemos:

factor(200!);

y nos entregará la lista de los factores primos. Si queremos saber el tiempo de ejecución que lleva hacer algún cálculo, hacemos:

showtime:true$

y especificamos la operación:

factor(2 ^300-1);

y desactivamos la información de tiempo de cálculo mediante:

showtime:false$

Números Complejos Maxima soporta números complejos, la unidad imaginaria $\sqrt{-1}$ se representa mediante el símbolo $\%i$ y soporta operaciones básicas con números complejos, por ejemplo definimos los números $z1$ y $z2$:

```
z1:3+5*%i$ z2:1/2-4*%i$
```

ahora podemos hacer cualquier otra operación, por ejemplo sumarlas:

```
z1+z2;
```

Combinatoria Maxima también soporta combinatoria, por ejemplo el cálculo de permutaciones de 3 elementos, mediante:

```
factorial(3)
```

si queremos generarlas, hacemos uso de la función *permutations*, por ejemplo:

```
permutations([1,2,3]);
```

cuando se trabaja con factoriales, la función *minfactorial* puede ayudar a simplificar algunas expresiones, por ejemplo:

```
minfactorial(n!/(n+2)!);
```

Expresiones Algebraicas Una de las capacidades más destacables de Maxima es su habilidad para manipular expresiones algebraicas, por ejemplo a la variable q le asignaremos un expresión literal:

```
q: (x+3)^5-(x-a)^3+(x+b)^(-1)+(x-1/4)^(-5);
```

se observa que en principio Maxima no realiza ningún cálculo. La función *expand* se encarga de desarrollar las potencias y los productos algebraicos:

```
expand(q);
```

no obstante es posible que no nos interese desplegar toda la expresión, entre otras cosas para evitar una respuesta farragosa y difícil de interpretar; en tal caso podemos utilizar *expand* añadiéndole dos argumentos y operando de la siguiente manera:

```
expand(q,3,2);
```

con los argumentos adicionales indicamos que queremos la expansión de todas aquellas potencias con exponente positivo menor o igual a 3 y de las que teniendo el exponente negativo no excedan en valor absoluto de 2.

Dada una expresión con valores literales, podemos desear sustituir alguna letra por otra expresión; por ejemplo, si queremos hacer los cambios $a = 2$, $b = 2c$ en el último resultado obtenido:

```
%,a=2,b=2*c;
```

De forma más general, la función *subst* substituye subexpresiones enteras. En el siguiente ejemplo, introducimos una expresión algebraica y a continuación sustituimos todos los binomios $x + y$ por la letra k:

```
1/(x+y)-(y+x)/z+(x+y)^2;
subst(k,x+y,%);
```

No obstante, el siguiente resultado nos sugiere que debemos ser precavidos con el uso de esta función, ya que Maxima no siempre interpretará como subexpresión aquella que para nosotros sí lo es:

```
subst(sqrt(k),x+y,(x+y)^2+(x+y));
```

la función *subst* realiza una simple sustitución sintáctica que, si bien es suficiente en la mayor parte de los casos, a veces se queda corta como en la situación anterior; la función *ratsubst* es capaz de utilizar información semántica y obtiene mejores resultados:

```
ratsubst(sqrt(k),x+y,(x+y)^2+(x+y));
```

la operación inversa de la expansión es la factorización. Expandamos y factoricemos sucesivamente un polinomio para comprobar los resultados:

```
expand((a-2)*(b+1)^2*(a+b)^5);
factor(%);
```

el máximo común divisor de un conjunto de polinomios se calcula con la función *gcd* y el mínimo común múltiplo con *lcm:*

```
p1: x^7-4*x^6-7*x^5+8*x^4+11*x^3-4*x^2-5*x;
p2: x^4-2*x^3-4*x^2+2*x+3;
gcd(p1,p2);
load(functs)$
lcm(p1,p2);
```

es posible que deseemos disponer del *mcd* factorizado, por lo que hacemos:

```
factor(%);
```

la instrucción *solve* puede admitir como segundo argumento la incógnita que se pretende calcular, lo que resultará de utilidad cuando en la ecuación aparezcan constantes literales:

```
solve((2-a)/x-3=b*x+1/x,x);
```

Matrices Podemos también hacer uso de matrices, por ejemplo:

```
M : matrix([sqrt(3),2/5],[3,sin(2)]);
```

y resolverla:

```
linsolve_by_lu(M,M.M,'floatfield);
```

Límites Sin más preámbulos, veamos algunos ejemplos de cómo calcular límites con la asistencia de Maxima. En primer lugar vemos que es posible hacer que la variable se aproxime al infinito ($x \to \infty$), o que se aproxime al menos infinito ($x \to -\infty$):

```
limit(1/sqrt(x),x,inf);
```

Derivadas Maxima controla el cálculo de derivadas mediante la instrucción *diff*. A continuación se presentan algunos ejemplos sobre su uso:

```
diff(x^log(a*x),x);
```

la segunda derivada:

```
diff(x^log(a*x),x,2);
```

Integrales En Maxima la función que controla el cálculo de integrales es *integrate*, tanto para las definidas como indefinidas; empecemos por estas últimas:

```
integrate(cos(x)^3/sin(x)^4,x);
```

otro ejemplo sobre la integral definida es:

```
integrate(2*x/((x-1)*(x+2)),x,3,5);
float(%);
```

6.3 Desde la Nube

Existen diferentes servicios Web[64] que permiten editar, compilar y ejecutar código de diversos lenguajes y paquetes desde el navegador, esto en aras de que los estudiantes y profesores que cuenten con algún sistema de acceso a red y un navegador puedan programar en los más diversos lenguajes, IDES y terminales sin hacer instalación alguna en su equipo de cómputo, tableta o teléfono celular.

Algunos ejemplos de estos servicio son:

- https://mathics.angusgriffith.com/ Clon de Mathematica
- http://www.wolframalpha.com/
- http://www.quickmath.com/
- http://maxima-online.org
- http://maxima.cesga.es/

[64] Cuando se trabaja desde la Web es recomendable usar el modo Privado o Incógnito para no guardar el historial de navegación, información introducida en los formularios y borrar al cerrar el navegador los datos de los sitios visitados. Pero recuerda que los sitios Web que visitamos sí guardan información de nuestra visita, nuestro proveedor de Internet también guarda constancia de nuestra visita y si descargamos algo, esto no se borra al igual que el historial de descargas, además de las marcas de páginas o favoritos se conservarán al cerrar el navegador.

7 Programación en Paquetes Estadísticos

Los paquetes estadísticos, son programas matemáticos que ofrecen un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio con un amplio abanico de herramientas de lectura, manipulación, análisis y graficación de datos estadísticos. Entre sus prestaciones básicas destacan:

- Análisis de datos mediante operadores para cálculos sobre arreglos, matrices y/o Tablas
- Tablas Cruzadas
- Reordenamiento de Datos
- Análisis de la Varianza (ANOVA)
- Frecuencias
- Estadística Descriptiva
- Estadística Lineal
- Estadística no Lineal
- Estadística Bioestadística
- Pruebas Estadísticas Clásicas
- Análisis de Serie de Temporales
- Modelos de Regresión
- Clasificación
- Fiabilidad
- Categorías
- Clustering
- Validación de Datos
- Tendencias

- Gráficos y Diagramas

Actualmente, los paquetes estadísticos usados en las carreras de Actuaría, Ciencias de la Computación, Matemáticas, Matemáticas Aplicadas y los cursos de Matemáticas de las demás carreras de la Facultad (véase [2]) son:

- SPSS (véase [36])
- R (véase [39])
- SAS (véase [35])
- PSPP (véase [37])
- EViews (véase [56])
- Gretel (véase [57])
- Stata (véase [55])
- Statgraphics (véase [59])
- Statistica (véase [60])
- Systat (véase [61])
- Vensim (véase [62])
- Maple (véase [63])
- Mathematica (véase [64])
- MATLAB (véase [29])
- FreeMat (véase [31])
- Octave (véase [32])
- Maxima (véase [65])
- Scipy (véase [33])
- Anaconda (véase [**?**])

- SageMath (véase [**?**])

En el presente trabajo nos centraremos en R, pero el resto de los paquetes son muy usados ya sea por sus características, facilidad de uso o la accesibilidad del paquete para los estudiantes. Cabe aclarar, que los paquete SPSS y SAS tienen un alto costo monetario, en el caso de SAS existe una versión estudiantil gratuita y SPPS hay una versión de prueba.

7.1 Cálculo Estadístico con R

El paquete R (véase [39]) es un lenguaje y entorno de programación para análisis estadístico y gráfico. Se trata de un proyecto de Software libre, resultado de la implementación GNU del premiado lenguaje S. SPSS, R y S-Plus -versión comercial de S- son, probablemente, los tres lenguajes más utilizados en investigación por la comunidad estadística, siendo además muy populares en el campo de la investigación biomédica, la bioinformática y las matemáticas financieras. A esto contribuye la posibilidad de cargar diferentes bibliotecas o paquetes con finalidades específicas de cálculo o gráfico.

Además, R puede integrarse con distintas bases de datos y existen bibliotecas que facilitan su uso desde lenguajes de programación interpretados como Perl y Python. R soporta hacer interfase con lenguajes de programación como C, C++ y Fortran.

Otra de las características de R es su capacidad gráfica, que permite generar gráficos con alta calidad. R posee su propio formato para la documentación basado en LaTeX (véase [28]). R también puede usarse como herramienta de cálculo numérico, campo en el que puede ser tan eficaz como otras herramientas específicas tales como FreeMat, GNU Octave y su equivalente comercial, MATLAB. Se ha desarrollado una interfaz RWeka para interactuar con Weka (véase [66]) que permite leer y escribir ficheros en el formato *arff* y enriquecer R con los algoritmos de minería de datos de dicha plataforma.

Los ambientes de desarrollo integrado para R existen como proyectos externos, como pueden ser editores -que sólo soportan la sintaxis-, los IDEs (Integrate Development Environments) y los GUI (Graphical User Interfaces) -permiten editar, ejecutar y depurar código desarrollado para R-. Hay más de 20 proyectos activos, dos de los más conocidos son Tinn-R (véase [67]) y RStudio (véase [68]).

7.2 Trabajando con R

R es un poderoso paquete que puede hacer múltiples operaciones por nosotros, a continuación haremos un viaje por sus capacidades, pero sólo es una muestra de ellas, para más información revise el manual y ejemplos en línea. Veamos algunos aspectos básicos a tener en cuenta:

- R distingue mayúsculas y minúsculas
- Para asignar contenido a un objeto usamos <-. Por ejemplo, x <- *10* asigna a x el valor *10*. En lugar de <- también podemos usar =
- Para ver el contenido de un objeto simplemente escribimos su nombre
- Para usar los comandos escribimos el nombre del comando seguido de sus argumentos entre paréntesis. Por ejemplo, *ls()* da una lista de los objetos en el área de trabajo. Como no usamos argumentos (diferentes a los que el comando tenga por defecto) no escribimos nada en el paréntesis
- Para obtener ayuda usamos el comando help. Por ejemplo, *help(mean)* para obtener ayuda sobre el comando *mean* que calcula la media

El comando c se usa para combinar varios objetos en uno solo. Por ejemplo:

```
x <- c(3, 5, 10)
```

crea el vector $x = (3, 5, 10)$, ahora lo visualizamos mediante:

```
x
```

para calcular, por ejemplo la media de x, hacemos:

```
mean(x)
```

Es habitual encontrar datos en la Web en formato de texto. Antes de poder trabajar con ellos hemos de importarlos a un formato con el que R pueda trabajar. Por ejemplo, en la comunidad de Madrid evalúa anualmente

a los alumnos de sexto de primaria de todos los colegios sobre varias materias. Los datos que vamos a utilizar corresponden a las notas medias obtenidas por los colegios en los años 2009 y 2010 (fuente: diario El País) junto con el tipo de colegio (concertado, privado o público).

Para ello, hay que bajar los datos y generar un fichero de R, el comando *read.table* lee un fichero de datos de texto y crea un fichero con el que R ya puede trabajar. En nuestro ejemplo vamos a usar los siguientes argumentos:

- El camino en el que se encuentra el fichero (entre comillas).
- El argumento *sep* que especifica la separación entre las distintas variables. En este caso, un espacio en blanco.
- El argumento *dec* que especifica cómo se separa en los números la parte entera y la parte decimal. En este caso, mediante una coma.
- El argumento header que especifica si la primera fila del fichero contiene (TRUE) o no (FALSE) los nombres de las variables. En este caso, la primera fila sí contiene los nombres de las variables.

```
notas <- read.table("http://www.uam.es/joser.berrendero/datos/notas.txt",

 sep = " ", dec = ",", header = TRUE)
```

Medidas descriptivas elementales, para ver los nombres de las variables del fichero *notas*, escribimos:

```
names(notas)
```

dentro de este fichero cada variable se identifica usando la sintaxis:

fichero$*variable*

Por ejemplo, las medias de las notas de 2009 y 2010 se obtienen con los comandos:

```
mean(notas$nota09)
```

las desviaciones típicas de las notas de 2009 y 2010 se obtienen con los comandos:

```
sd(notas$nota09)
sd(notas$nota10)
```

el comando *summary* permite calcular algunas medidas descriptivas elementales de todas las variables del fichero simultáneamente:

```
summary(notas)
```

Ya con esta información, ahora podemos generar un histograma de las notas de 2009. La distribución es bastante simétrica aunque muestra una ligera asimetría a la izquierda.

```
hist(notas$nota09)
```

para comparar conjuntos de datos un diagrama de caja puede ser más útil que un histograma. Por ejemplo, para comparar las notas de 2009 con las de 2010:

```
boxplot(notas$nota09, notas$nota10)
```

vemos que en 2010 las notas tienden a ser más bajas. También podemos comparar las notas de 2010 para cada tipo de colegio. Aunque las diferencias no son muy grandes, los centros privados tienden a tener notas más altas que los concertados, y éstos más altas que los públicos.

```
boxplot(notas$nota10 ~notas$tipo)
plot of chunk boxplot2
```

para estudiar si hay relación entre las notas de los dos años podemos representar la nube de puntos correspondiente:

```
plot(notas$nota09, notas$nota10)
```

se observa una relación positiva moderada. Numéricamente, esta relación se puede cuantificar mediante la covarianza y la correlación:

```
cov(notas$nota09, notas$nota10)
cor(notas$nota09, notas$nota10)
```

Además de analizar datos, R se puede usar para llevar a cabo todo tipo de representaciones gráficas y cálculos matemáticos. Por ejemplo, la función $\cos(x\hat{}2)$ se puede representar entre $\left[0, \frac{2}{\pi}\right]$ mediante:

```
curve(cos(x^2), 0, 2 * pi)
```

la integral de la función anterior entre $\left[0, \frac{2}{\pi}\right]$ se calcula con:

```
integrate(function(x) cos(x^2), 0, pi)
```

una derivada de la función se obtiene de la siguiente forma:

```
D(expression(cos(x^2)), "x")
```

para representar la función de densidad de una variable aleatoria normal de media $\mu = 3$ y desviación típica $\sigma = 1$ escribimos:

```
curve(dnorm(x, mean = 3, sd = 1), -1, 7)
```

hemos usado el comando *dnorm*, con los argumentos adecuados, para evaluar la función de densidad normal. Si lo que queremos es generar 1000 números aleatorios que sigan esta distribución y después representar el histograma correspondiente:

```
x <- rnorm(1000, mean = 3, sd = 1)
hist(x)
```

Podemos escribir también nuestras propias funciones. En este apartado vamos a ver un ejemplo muy simple. Se trata de escribir una función que simule los resultados de n tiradas de un dado. Por defecto n. El resultado es un vector que contiene las frecuencias de cada posible resultado. Abrimos el editor (Archivo, Abrir script) y escribimos:

```
Dado <- function(n = 50) {
 # Genera n lanzamientos de un dado y devuelve la tabla de frecuencias
 lanzamientos <- sample(1:6, n, rep = TRUE)
 # Selecciona con reemplazamiento n números en 1,2,...,6
 frecuencias <- table(lanzamientos) # Obtiene la tabla de frecuencias
 return(frecuencias)
}
```

una vez guardado el fichero que contiene la función podemos usar:

$$source(nombredelfichero)$$

para cargar la función en el área de trabajo y poder usarla o, alternativamente, copiar y pegar el código anterior en la consola. Una vez hecho esto, ya podemos usar la función.

```
Dado(1000)
Dado()
```

7.2.1 R y Python

El paquete R es un lenguaje y entorno de programación para análisis estadístico y gráfico poderoso, pero es posible extender su uso al lenguaje de programación de Python, logrando una amalgama aún en proceso de desarrollo que permite tener lo mejor de ambos mundos con muy poco trabajo extra.

Para iniciar, se debe importar las funciones y demás que componen la parte del paquete R para poder usarse en Python, mediante:

```
from R_functions import *
```

y se está listo para usar R en Python, ejemplo:

```
lst=[20,12,16,32,27,65,44,45,22,18]
fivenum(lst)
```

obteniendo:

```
> array([12. , 18.5, 24.5, 41. , 65. ])
```

Un ejemplo más es calcular un diagrama de barras con unas cuantas líneas usando la función *dbinom*:

```
probs=[]
import matplotlib.pyplot as plt
for i in range(11):
  probs.append(dbinom(i,10,0.6))
plt.bar(range(11),height=probs)
plt.grid(True)
plt.show()
```

Por otro lado existe Anaconda, una Suite de código abierto que abarca una serie de aplicaciones, librerías y conceptos diseñados para el desarrollo de la Ciencia de datos con Python. En líneas generales Anaconda Distribution es una distribución de Python que funciona como un gestor de entorno, un gestor de paquetes y que posee una colección de más de 720 paquetes de código abierto. Anaconda Distribution se agrupa en 4 sectores o soluciones tecnológicas, Anaconda Navigator, Anaconda Project, Las librerías de Ciencia de datos y Conda. Todas estas se instalan de manera automática y en un procedimiento muy sencillo. Para más información ver: https://www.anaconda.com/.

También esta SageMath, una Suite de código abierto bajo la licencia GPL de Software matemático como: NumPy, SciPy, matplotlib, Sympy, Maxima, GAP, FLINT, R, entre otros. Además combina acceso a una poderosa combinación del lenguaje basada en Python o directamente vía interfaces o Wrappers. La misión del proyecto es crear una alternativa de Software libre a Magma, Maple, Mathematica y Matlab. Para más información ver: http://www.sagemath.org/.

7.3 Desde la Nube

Existen diferentes servicios Web[65] que permiten editar, compilar y ejecutar código de diversos lenguajes y paquetes desde el navegador, esto en aras de que los estudiantes y profesores que cuenten con algún sistema de acceso a red y un navegador puedan programar en los más diversos lenguajes, IDES y terminales sin hacer instalación alguna en su equipo de cómputo, tableta o teléfono celular.

Algunos ejemplos de estos servicio son:

- Para R https://nclab.com/free-portal/
- Para R https://cdn.datacamp.com/dcl-react-prod/example.html

[65] Cuando se trabaja desde la Web es recomendable usar el modo Privado o Incógnito para no guardar el historial de navegación, información introducida en los formularios y borrar al cerrar el navegador los datos de los sitios visitados. Pero recuerda que los sitios Web que visitamos sí guardan información de nuestra visita, nuestro proveedor de Internet también guarda constancia de nuestra visita y si descargamos algo, esto no se borra al igual que el historial de descargas, además de las marcas de páginas o favoritos se conservarán al cerrar el navegador.

8 Paquetes Ofimáticos

En la actualidad, los llamados paquetes ofimáticos, no son otra cosa que programas de cómputo integrado, que permiten automatizar multiples tareas que ayudan a idear, crear, manipular, transmitir, almacenar información necesaria en una oficina. Entre sus prestaciones básicas destacan:

- Hojas de Cálculo
- Bases de Datos
- Herramientas de Presentación y Multimedia
- Procesamiento de Imágenes
- Procesamiento de Textos

Existe una gran cantidad de paquetes ofimáticos, que van, desde los instalados hasta los asequibles a través de la Web, entre los más comunes tenemos:

- Microsoft Office (véase [18])
- Libre Office (véase [20])
- OpenOffice (véase [19])
- Calligra (véase [22])
- Google Docs (véase [21])
- Lotus Symphony (véase [27])
- Only Office (véase [23])
- WPS Office (véase [24])
- Office Online (véase [25])
- Collabora (véase [26])

8.1 Hojas de Cálculo

Es un Software a través del cual se pueden usar datos numéricos y realizar cálculos automáticos de números que estan en una tabla. También es posible automatizar cálculos complejos al utilizar una gran cantidad de parámetros y al crear tablas llamadas hojas de trabajo.

Las hojas de cálculo permiten a los usuarios elaborar tablas y formatos que incluyan cálculos matemáticos mediante fórmulas, las cuales pueden usar operadores matemáticos como son: + (suma), - (resta), * (multiplicación), / (división) y ^(exponenciación), además de poder utilizar elementos denominados funciones como por ejemplo: Suma(), Promedio(), Buscar(), etc.

Así mismo las hojas de cálculo son útiles para gestionar Listas o Bases de Datos; es decir Ordenar y Filtrar la información. Por lo tanto, la hoja de cálculo es una herramienta multiuso que sirve tanto para actividades de oficina, que implican la organización de grandes cantidades de datos, como para niveles estratégicos y de toma de decisiones al crear representaciones gráficas de la información sintetizada.

Existe una gran variedad de paquetes para el manejo hojas de cálculo -los cuales existen tanto en las plataformas de Windows, Linux, Mac-, entre los que destacan:

- Excel: Paquete de Microsoft Office (véase [18])
- Calc: Paquete Libre Office (véase [20])
- OpenCalc: Paquete OpenOffice (véase [19])
- Spread Sheet: Google Docs (véase [21])
- Sheets: Paquete Calligra (véase [22])

Por otro lado, Microsoft VBA (Visual Basic for Applications) es el lenguaje de macros de Microsoft Visual Basic que se utiliza para programar aplicaciones Windows y que se incluye en varias aplicaciones Microsoft. VBA permite a usuarios y programadores ampliar la funcionalidad de programas de la suite Microsoft Office. Visual Basic para Aplicaciones es un subconjunto casi completo de Visual Basic 5.0 y 6.0.

Microsoft VBA viene integrado en aplicaciones de Microsoft Office, como Word, Excel, Access y PowerPoint. Prácticamente cualquier cosa que se

pueda programar en Visual Basic 5.0 o 6.0 se puede hacer también dentro de un documento de Office, con la limitación que el producto final no se puede compilar separadamente del documento, hoja o base de datos en que fue creado; es decir, se convierte en una macro (mejor conocida como súper macro). Esta macro puede instalarse o distribuirse con sólo copiar el documento, presentación o base de datos.

Su utilidad principal es automatizar tareas cotidianas, así como crear aplicaciones y servicios de bases de datos para el escritorio. Permite acceder a las funcionalidades de un lenguaje orientado a eventos con acceso a la API de Windows.

Al provenir de un lenguaje basado en Basic tiene similitudes con lenguajes incluidos en otros productos de ofimática como Libre Office y Open Office, pero no hay compatibilidad entre productos.

Así también, es común que en los cursos de la carreras de Actuaría, Ciencias de la Computación, Matemáticas, Matemáticas Aplicadas y los cursos de Matemáticas de las demás carreras de la Facultad, se requiera hacer análisis estadísticos de datos, esto se realiza mediante el uso de complementos de Excel -herramientas para análisis de datos y Solver-. En donde al usar estas herramientas, se proporcionan los datos y parámetros para cada análisis y la herramienta utilizará las funciones de macros estadísticas o técnicas correspondientes para realizar los cálculos y mostrar los resultados en una tabla de resultados. Algunas herramientas generan gráficos además de tablas de resultados.

Entre las herramientas para análisis se incluyen:

- Análisis de Fourier
- Correlación
- Covarianza
- Estadística descriptiva
- Generación de números aleatorios
- Histograma
- Jerarquía y percentil
- Media móvil

- Muestreo
- Prueba t
- Prueba t para varianzas de dos muestras
- Prueba z
- Regresión
- Suavización exponencial
- Varianza

El acceso a tablas de Excel usando Visual Basic for Applications, además del uso de los complementos de Excel -herramientas para análisis de datos y Solver- para manejo de datos, ha generalizado el uso de los paquetes de Microsoft Office, esto redunda en el uso de dicha suite en una importante cantidad de cursos dentro de las carreras de la Facultad de Ciencias.

8.2 Bases de Datos

Una parte importante de la Modelación Matemática es trabajar con datos de prueba, lo más cercano posible a la realidad. Ello implica que, es necesario contar con mecanismos para almacenar, editar y consultar una cantidad grande de datos, esto se logra usando las bases de datos.

Existe una gran variedad de paquetes para el manejo de base de datos -los cuales se ejecutan en las plataformas de Windows, Linux, Mac-, entre los que destacan:

- Access en Microsoft Office para Windows (véase [18])
- Microsoft SQL Server (véase [82])
- PostgreSQL (véase [83])
- MySQL (véase [84])
- MongoBD (véase [85])

En donde entendemos a un Sistema de Gestión de Bases de Datos (SGBD) como un conjunto de programas que permiten el almacenamiento, modificación y extracción de la información en una base de datos, además de proporcionar herramientas para añadir, borrar modificar y analizar los datos. Los usuarios pueden acceder a la información usando herramientas específicas de interrogación y de generación de informes, o bien mediante aplicaciones diseñadas para este fin.

Los SGBD también proporcionan métodos para mantener la integridad de los datos, para administrar el acceso de usuarios a los datos y recuperar la información si el sistema se corrompe. Permite presentar la información de la base de datos en varios formatos. La mayoría de los SGBD incluyen un generador de informes. También puede incluir un módulo gráfico que permita presentar la información con gráficos y diagramas.

Hay muchos tipos de SGBD distintos, esto depende de la forma en que se manejen los datos, el tamaño de las bases de datos y en el equipo de cómputo en que se implementen. Los tamaños varían desde los que operan en equipos de cómputo personales y con poca memoria a grandes sistemas que funcionan en mainframes con sistemas de almacenamiento distribuido y redundante.

Generalmente se accede a los datos mediante lenguajes de interrogación[66], son denominados lenguajes de alto nivel que simplifican la tarea de construir las aplicaciones. También simplifican la interrogación y la presentación de la información. Un SGDB permite controlar el acceso a los datos, asegurar su integridad, gestionar el acceso concurrente a ellos, recuperar los datos tras un fallo del sistema y hacer copias de seguridad.

El uso generalizado de los paquetes de Microsoft Office, en particular el acceso a el paquete Access mediante tablas de Excel usando Visual Basic for Applications, además del uso de los complementos de Excel para manejo de datos, hace que sean propicios para que un importante número de cursos dentro de las carreras de la Facultad de Ciencias hagan uso extensivo de dicha suite.

Instalación de Manejadores de Bases de Datos e IDEs en Debian GNU/Linux Existen diversas versiones de paquetes para manejar bases de Datos en Linux, para instalar las más comunes en Debian GNU/Linux es necesario hacer:

[66] Lenguajes tipo SQL (Structured Query Languaje) que es un lenguaje de consulta estructurada que da acceso a un sistema de gestión de bases de datos.

```
# apt install apache2 libapache2-mod-evasive php \
apachetop libapache2-mod-php php-mysql phpmyadmin \
php-gd mysql-common mysql-client mysql-server mytop \
mysql-admin mysql-workbench mysql-workbench-data \
ferret  postgresql postgresql-client postgresql-doc pgcli \
postgresql-contrib sqlite3 sqlite3-doc sqlitebrowser mycli \
mongodb mongodb-clients mongodb-server
```

8.3 Herramientas de Presentación y Multimedia

El programa líder del mercado es Microsoft PowerPoint (véase [18]), este es un programa de presentación desarrollado por la empresa Microsoft para sistemas operativos Microsoft Windows y Mac OS, ampliamente usado en distintos campos como la enseñanza, negocios, etc. Aunque todos los paquetes ofimáticos tienen una herramienta para realizar presentaciones, sólo algunos son altamente compatibles con la desarrollada por Microsoft.

Estos programas se han diseñado para hacer presentaciones con texto esquematizado, animaciones de texto e imágenes prediseñadas o importadas desde imágenes de la computadora. Se le pueden aplicar distintos diseños de fuente, plantilla y animación. Este tipo de presentaciones suelen ser más prácticas que las de los paquetes de edición de textos.

Las herramientas de presentación vienen integrados en los paquetes ofimáticos como un elemento más, que puede aprovechar las ventajas que le ofrecen los demás componentes del paquete para obtener un resultado óptimo.

En el caso de PowerPoint, se puede usar como complemento para la edición de texto científico a MathType (véase [70]) para Microsoft Office, este es un paquete adicional de los más usados en cuanto a la tipografía matemática.

Por otro lado, esta ganando terreno en la Facultad de Ciencias el uso de Beamer (véase [81]), el cual es una clase de LaTeX (véase [28]) para la creación de presentaciones. Este funciona con *pdflatex, dvips, LyX* entre otros. Al estar basado en LaTeX, Beamer es especialmente útil para preparar presentaciones en las que es necesario mostrar gran cantidad de expresiones matemáticas. En los últimos semestres se ha hecho una amplia difusión a los paquetes que usan a LaTeX como base, pues son ampliamente usados por la comunidad científica mundial.

8.4 Procesamiento de Imágenes

Existe una gran cantidad de usos para los programas de edición de imágenes, pero en las carreras de la Facultad de Ciencias, la edición y manipulación de gráficos vectoriales[67] es común. Es por ello que la gran mayoría de los paquetes de edición y manipulación de imágenes no proporcionan las herramientas necesarias para procesar adecuadamente imágenes vectoriales. En caso de proveer dichas herramientas, muchas de ellas son de uso tedioso, pues están diseñadas para uso ocasional.

Para subsanar este hecho, existen herramientas y editores hechos a ex profeso, para permitir la edición y manipulación de gráficos vectoriales en los cuales su procesamiento es una tarea sencilla de realizar. Existe una gran variedad de paquetes para edición de gráficos vectoriales -los cuales existen tanto en las plataformas de Windows, Linux, Mac-, entre los que destacan:

- Adobe InDesign
- Scribus
- Inkscape
- Gravit
- Vectr
- Libre Office Draw
- SK1
- Maya
- Blender
- Autodesk 3Ds Max
- Gimp

[67] Una imagen vectorial es una imagen digital formada por objetos geométricos dependientes, cada uno de ellos definido por atributos matemáticos de forma, posición, etc. Y son completamente diferentes a las imágenes de mapa de bits. El interés principal de los gráficos vectoriales es poder ampliar el tamaño de la imagen a voluntad sin sufrir perdida de calidad que sufren los mapas de bits. Pero todos los ordenadores traducen los gráficos vectoriales a mapas de bits para poderlos representar en la pantalla, impresora, etc.

- Corel Photo-Paint
- Adobe Photoshop
- CorelDRAW
- Adobe Illustrator
- Photoshop

8.5 Procesamiento de Textos

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

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

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

- Editor de ecuaciones integrado en Word en Microsoft Office (véase [18])
- MathType para Word en Microsoft Office para Windows (véase [70])
- Scientific WorkPlace LaTex para Windows (véase [71])
- Gummi LaTeX (véase [72])
- Kile LaTeX (véase [73])
- LED LaTeX (véase [74])
- LyX LaTeX (véase [75])
- Texmaker LaTeX (véase [76])

- TeXnicCenter LaTeX (véase [77])
- TextPad LaTeX (véase [78])
- TeXstudio LaTeX (véase [79])
- WinEdt LaTeX (véase [80])
- Formula de Libre Office (véase [20])
- Math de OpenOffice (véase [19])
- Formula de Calligra (véase [22])

Salvo para los productos de Microsoft Office, el resto de los paquetes tienen una curva de aprendizaje de media a alta, pero en contraste permiten desarrollar textos y gráficos con tipografía científica de alta calidad. En la Facultad de Ciencias, desde hace ya varios años, semestre a semestre se imparten cursos a estudiantes, tesistas y profesores de LaTeX y el manejo de uno o más editores que lo soportan; una cantidad importante de ellos se han impartido en las Aulas y Talleres del Departamento de Matemáticas en el Edificio de Investigación y Docencia Tlahuizcalpan, además de contar con el repositorio oficial de LaTeX (véase [87]) dentro de la Facultad.

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

> # apt install science-typesetting texlive-science texstudio pandoc texmaker inkscape kile gummi texstudio enchant texlive-latex-base texlive-latex-recommended latexila lyx medit texworks texlive-full latexila libreoffice calligra abiword evince gnumeric kexi texlive-extra-utils pdf-viewer msttCorefonts djview4 okular gv zathura diffpdf mupdf pdf-presenter-console evince xpdf okular poppler-utils atril pdftk pdfgrep xpdf-utils pdfcrack qpdf pdfsam pdfshuffler htmldoc pdf2svg pdfmod pdfposter pdfchain pdf2djvu gpdftext catdoc chktex cxref cxref-doc latex2rtf antiword unoconv a2ps bookletimposer qpdfview rst2pdf xchm chm2pdf archmage qpdfview-ps-plugin qpdfview qpdfview-djvu-plugin kchmviewer ispanish wspanish texlive-lang-spanish myspell-es myspell-en-us translate-shell

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

Latex

9 Seguridad, Privacidad y Vigilancia

Ante el constante aumento -explosivo- en el uso de dispositivos conectados a internet como computadoras personales, Laptops, tabletas y teléfonos celulares, expertos en ciberseguridad advierten un entorno propicio para que prosperen los cibercriminales y que, tanto individuos como empresas, se encuentren expuestos a múltiples amenazas de ciberseguridad.

En la actualidad, aproximadamente la mitad de la población mundial accede de algún modo a internet. Con tantos accesos concurrentes a la red de redes, la posible amenaza de seguridad a los sistemas informáticos crece y se complejiza, a pesar de las diversas y especializadas maneras de contrarrestarlas. Por su propia naturaleza, una conexión a internet hace que tu equipo de cómputo quede expuesto a ataques y pueda ser accesible por otro equipo, llegando a tener acceso a tu información.

¿Por qué?

- En muchas ocasiones, los usuarios no cuentan con la suficiente sensibilización sobre su exposición al riesgo, o bien, no están familiarizados con las herramientas y/o capacitación de sus organizaciones para prevenir y enfrentar amenazas de ciberseguridad.
- Los usuarios de internet a menudo utilizan redes Wi-Fi no seguras y usan dispositivos que no están configurados con los controles de políticas de seguridad básicos, lo cual los vuelve excepcionalmente vulnerables a ataques cibernéticos.
- Es común que los usuarios compartan dispositivos (computadoras, tabletas, teléfonos) para educación, trabajo y esparcimiento aumentando con ello su exposición a amenazas de ciberseguridad.
- Los piratas informáticos eligen como blanco la dependencia, cada vez mayor, de las personas con respecto a las herramientas digitales, además de que más tiempo en línea incrementa la potencial exposición de los usuarios a amenazas de ciberseguridad.
- Un error común es creer que los ciberatacantes utilizan únicamente herramientas muy avanzadas y técnicas para Hackear las computadoras o cuentas de las personas. Los atacantes han aprendido que la forma

más sencilla de robar la información de los usuarios, comprometer sus cuentas o infectar los sistemas es simplemente engañar al usuario para que lo hagan por ellos con una técnica denominada ingeniería social.

- Los piratas informáticos son extremadamente creativos al idear formas de aprovecharse de los usuarios y de la tecnología para acceder a contraseñas, redes y datos, a menudo sirviéndose de herramientas de ingeniería social y de temas y tendencias populares para tentar a los usuarios a tener comportamientos inseguros en línea.

Todo lo anterior, ha creado una enorme superficie de exposición a ataques cibernéticos dirigidos a los usuarios, la red, la computadora portátil, el teléfono inteligente, la tableta, etc. con la intención de cometer delitos informáticos. Por ello estos lineamientos generales de ciberseguridad son sugeridos para el uso seguro de redes y dispositivos de telecomunicaciones en apoyo al usuario, contiene recomendaciones sencillas y prácticas para fomentar una buena salud cibernética de los dispositivos utilizados para el trabajo a distancia, educación y diversión; y promover la ciberseguridad entre las personas y sus organizaciones.

La seguridad de la información en la red es más que un problema de protección de datos, y debe estar básicamente orientada en asegurar la información importante de las personas, de las organizaciones y la propiedad intelectual. Los riesgos de la información están presentes cuando confluyen fundamentalmente dos elementos: las amenazas de ataques, y las vulnerabilidades de la tecnología; conceptos íntimamente relacionados donde no es posible ninguna consecuencia sin la presencia conjunta de estos.

9.1 ¿Qué es la Privacidad y por qué es Importante?

Consideramos que la privacidad debería ser un derecho básico y una protección necesaria en la era digital para evitar la victimización y la manipulación. Una sociedad no puede tener libertad sin privacidad. Puede parecer un lujo, pero es importante para el bienestar de una sociedad libre y justa. Muchos gobiernos y empleadores hacen vigilancia[68] activa a sus ciudadanos o empleados para monitorear ideas, discusiones o disensiones no deseadas. Los

[68] ¿Qué es vigilancia? Vigilancia es el seguimiento de las comunicaciones, acciones o movimientos de una persona por un gobierno, empresa, grupo o persona.

¿Cuándo es legal la vigilancia? En general, cuando es selectiva, se basa en indicios suficientes de conducta delictiva y está autorizada por una autoridad estrictamente inde-

infractores son luego procesados o reeducados para alinearse con lo que las autoridades consideran apropiado. Sin el beneficio del anonimato, el deseo de los ciudadanos de expresar sus pensamientos se reprime efectivamente.

En la era digital, la privacidad va más allá del anonimato, al proteger a las personas de la victimización y la manipulación. La sociedad ha adoptado la tecnología para educarse, comunicarse, realizar negocios y formar relaciones. Nuestros puntos de vista y opiniones están fuertemente influenciados por lo que aprendemos de las fuentes de noticias locales, nacionales e internacionales.

Contribuimos en gran medida al panorama digital a través de nuestras acciones y decisiones. Nuestras huellas digitales están en todas partes. Cuentan una historia de a dónde vamos, qué hacemos, quién nos gusta o no, y qué pensamos. Se crean con cada Clic que hacemos y con cada archivo, aplicación y dispositivo que usamos. Cuando esos datos se agregan, pueden proporcionar información tremendamente poderosa sobre una persona o comunidad, lo suficiente como para construir personas complejas y precisas.

Esta información se usa comúnmente para manipular las creencias y comportamientos de las personas. Las compras en línea son un ejemplo perfecto: el Marketing dirigido y la publicidad basada en datos es un gran negocio porque logra que la gente gaste dinero. Todo se reduce a saber lo que las personas hacen, piensan, dicen, consumen y miran[69]. Tener acceso a grandes

pendiente, como un juez.

¿Qué es la vigilancia masiva? La vigilancia masiva indiscriminada es el control de las comunicaciones por internet y telefónicas de un gran número de personas -a veces de países enteros- sin que existan indicios suficientes de conducta delictiva. Este tipo de vigilancia no es legal.

¿Qué datos recogen? Algunos gobiernos almacenan y analizan historiales de navegación, búsquedas en internet, mensajes de correo electrónico, mensajes instantáneos, conversaciones por Webcam y llamadas telefónicas. También reúnen metadatos -datos sobre datos-, como destinatarios de correo electrónico, horas de llamadas y registros de ubicación.

¿Qué hacen con mis datos? Se guardan en grandes centros de datos donde unos algoritmos informáticos pueden hacer búsquedas en ellos y analizarlos. También están a disposición de las autoridades de las agencias de seguridad a través de potentes bases de datos como XKeyscore.

[69]Has notado que si buscas algo en un buscador comercial -como Google, Yahoo, Bing, Ask, Baidu, etc.-, minutos después en tus distintas redes sociales aparecerán mensajes relacionados a tu búsqueda. Esto ocurre porque los buscadores comerciales en cada búsqueda que haces, vídeo o los anuncios que ves o en los que haces Clic comparten tu ubicación, los sitios que visitas, los dispositivos, navegadores y aplicaciones que usas para acceder a los servicios del buscador con fines comerciales.

cantidades de datos privados les brinda a los anunciantes la capacidad de crear mensajes oportunos y significativos que atraigan a las personas a los comportamientos deseados.

Pero si los minoristas pueden hacer que las personas compren cosas que no necesitan, ¿para qué más se pueden usar los datos privados? ¿Qué tal cambiar lo que piensa la gente, a quiénes apoyan, sus opiniones políticas, qué debería convertirse en ley y en qué creer? El uso de información privada se ha aprovechado durante mucho tiempo para promover, vilipendiar o perseguir a diversas religiones, partidos políticos y líderes.

En las últimas décadas, ha cambiado la forma en que los ciudadanos del mundo reciben sus noticias. Los segmentos de noticias y entretenimiento han comenzado a mezclarse, a menudo informando hechos con adornos e historias obstinadas para influir en las opiniones públicas. Cuanta más información privada se conozca, más fácil será influir, convencer, engatusar o amenazar a las personas.

Según informes de Oxford Internet Institute, a pesar de los esfuerzos para combatir la propaganda computacional, el problema ha crecido a gran escala. El mayor crecimiento proviene de propaganda política que es difundida junto con desinformación y noticias basura alrededor de los períodos electorales. Se incrementa el número de campañas políticas a nivel mundial que usan Bots, noticias basura y desinformación para polarizar y manipular a los votantes.

Un velo de privacidad puede proteger tanto los beneficios como los abusos. La tendencia actual es establecer y ampliar los derechos de privacidad en beneficio de los ciudadanos. Esto reduce la victimización, manipulación y explotación digitales al proteger los datos confidenciales y permite actividades que promueven la libertad y la libertad de expresión.

Sin leyes, los gobiernos y las empresas han desarrollado prácticas que aprovechan el poder de recopilar información confidencial y utilizarla en su propio beneficio. Las nuevas leyes de privacidad están cambiando el panorama y muchas empresas éticas reducen sus esfuerzos de cobranza para ser más conservadoras y respetuosas. También muestran flexibilidad en la forma en que tratan, protegen y comparten dichos datos.

Algunos gobiernos y agencias también están reduciendo la recolección, limitando la retención o terminando los programas domésticos que los ciudadanos consideran invasivos. Al mismo tiempo, los organismos encargados

Una opción para nuestra privacidad es usar buscadores que no dejan rastro de nuestras búsquedas como puede ser: DuckDuckGo.

de hacer cumplir la ley quieren conservar la capacidad para detectar e investigar delitos, para proteger la seguridad de los ciudadanos.

También se hace un mal uso de la privacidad. Es la herramienta preferida por quienes cometen delitos y permite que los actos atroces contra otros no se detecten. Puede ocultar actos terribles y permitir una coordinación generalizada entre el fraude, abuso y terror.

Se argumenta que las puertas traseras digitales, las claves maestras y los algoritmos de cifrado que obtienen acceso a los sistemas y la información privada ayudarían en la detección legal de actividades delictivas y en las investigaciones para identificar a los terroristas. Aunque suena como una gran herramienta contra los delincuentes, es una caja de Pandora.

Las puertas traseras y las llaves maestras no limitan el acceso para una investigación específica donde existe una causa probable, sino que permiten una vigilancia generalizada[70] y la recolección de datos de toda una población, incluidos los ciudadanos respetuosos de la ley. Esto viola el derecho de las personas a la privacidad y abre la puerta a la manipulación y el enjuiciamiento político. La capacidad de leer cada texto, correo electrónico, mensaje y conversación en línea para "monitorear" a la población crea un camino claro hacia el abuso. El riesgo de control y explotación es real.

Incluso para aquellos que no tienen ninguna objeción a que su gobierno tenga acceso, debemos considerar el hecho de que tales puertas traseras y

[70]El 5 de junio de 2013, el exanalista de la CIA y de la NSA Edward Snowden decidió revelar la existencia de programas de vigilancia sobre las comunicaciones de millones de ciudadanos de todo el mundo. A través de The Guardian y The Washington Post, supimos que, en nombre de la seguridad y sin ningún control judicial, la NSA y el gobierno británico habían rastreado e-mails, llamadas telefónicas y mensajes encriptados. Empresas como Facebook, Google y Microsoft habían sido obligadas a entregar información de sus clientes por órdenes secretas de la NSA. Esta misma agencia grabó, almacenó y analizó los 'metadatos' de las llamadas y de los mensajes de texto enviados en México, Kenia y Filipinas. Incluso llegaron a espiar el móvil de la canciller alemana Angela Merkel.

Snowden hizo las filtraciones a la prensa desde Hong Kong y actualmente vive en Rusia, donde se le concedió asilo. No puede volver a su país porque está acusado de revelar información clasificada a personas no autorizadas y de robo de propiedad del gobierno federal. Para unos es un héroe y para otros un traidor, gracias a las revelaciones de Snowden la opinión pública es más consciente de su derecho a la privacidad y ha reaccionado oponiéndose al espionaje masivo.

Aunque queda un largo camino para asegurar que los Gobiernos no invadan la vida privada de las personas, los tribunales han declarado ilegales algunos aspectos de estos programas y las empresas tecnológicas han tenido que posicionarse ante un escándalo capaz de dañar seriamente su reputación.

claves maestras serían buscadas por ciberdelincuentes y otros actores del estado-nación. Ningún sistema es infalible. Con el tiempo, los delincuentes encontrarían y utilizarían estas herramientas en detrimento de la comunidad digital mundial. Algunas puertas traseras podrían valer decenas de miles de millones de dólares para el comprador adecuado, ya que podrían desbloquear un poder inimaginable para apoderarse de la riqueza, afectar a la gente, dañar naciones, socavar la independencia y reprimir el pensamiento libre.

Proteger la privacidad no se trata de ocultar información. Se trata de la capacidad de liberarse de influencias no deseadas, de la tiranía y de comunicarse con los demás de formas que desafíen el statu quo. La privacidad protege a las personas, pero también los cimientos de una sociedad libre. La privacidad no es un tema fácil y no existe una solución perfecta. Es una situación dinámica y seguirá cambiando con el sentimiento público.

Todos quieren cierto nivel de discreción, confidencialidad y espacio. Nadie quiere que se expongan sus contraseñas, finanzas familiares, detalles de relaciones personales, historial médico, ubicación, compras y discusiones privadas. Las personas tampoco disfrutan de verse inundadas de Spam, Phishing y llamadas de ventas incesantes. La privacidad no se trata necesariamente de ocultar algo, sino de limitar la información a quienes tienen derecho a saber.

Muy poca privacidad puede socavar la libertad de expresión, la libertad y la denuncia de victimizaciones. También empodera a entidades poderosas para manipular el mundo digital de las personas para coaccionarlas, manipularlas y victimizarlas. Demasiada privacidad puede permitir que los actores criminales prosperen y se escondan de las autoridades. Debe lograrse un equilibrio[71].

[71] Los gobiernos nos enfrentan a un dilema falso: seguridad o libertad. En un estado de derecho, donde las leyes equilibran ambos conceptos, las personas son inocentes hasta que se demuestra lo contrario y tienen derecho a que se respete su vida privada. Por tanto, antes de violar estos derechos, los gobiernos deben de tener indicios de que se está cometiendo un delito. No pueden buscar pruebas aleatoriamente en nuestras comunicaciones privadas antes de que se cometa ese delito.

Varios países dan "carta blanca" a la vigilancia indiscriminada en sus leyes. En Francia se permite interceptar masivamente comunicaciones, retener información durante largos períodos de tiempo y se ha eliminado la autorización judicial previa. También Reino Unido ha introducido en su legislación mayores poderes de espionaje. Polonia ha otorgado poderes de vigilancia incompatibles con respecto a la privacidad a la policía y o otras agencias. Otros países como China o Rusia también vigilan internet con total desprecio a la intimidad de las personas.

Algunos gobiernos y empresas tecnológicas quieren que aceptemos que no tenemos derechos cuando estamos en internet. Que cuando usamos el teléfono o entramos en nuestra cuenta de correo electrónico, todo lo que hacemos o decimos les pertenece. No permitiríamos este grado de intrusión en nuestra vida fuera de internet, así que no debemos permitirlo dentro.

Cuando los gobiernos no protegen los derechos humanos, sólo la acción de la gente común y corriente puede hacer que las cosas cambien y que los responsables de los abusos rindan cuentas. Muchas organizaciones trabajan para que los gobiernos prohíban la vigilancia masiva y el intercambio ilegal de información confidencial. Esta será una larga lucha no exenta de dificultades, pero cuyo impulso va creciendo con el tiempo.

¿Qué son los datos biométricos? en general entenderemos por datos biométricos a las características físicas, fisiológicas, morfológicas, de comportamiento y rasgos de personalidad distintivas de cada persona que permite distinguir ciertas singularidades e identificar al individuo en cuestión.

El uso de datos biométricos no es nuevo e incluso es algo en lo que constantemente se ven envueltos las redes sociales[72], aunque los fines de uso son diversos. Recientemente Texas demandó a Meta, dueña de Facebook, por almacenar millones de estos datos y comercializarla con terceros.

Las Amenazas a los Usuarios en un entorno dinámico de interconectividad, pueden venir de cualquier parte, sea interna o externa, e íntimamente relacionada con el entorno de las organizaciones. Las vulnerabilidades son una debilidad en la tecnología o en los procesos asociados con la información, y como tal, se consideran características propias de los sistemas o de la infraestructura que los soporta.

Las personas o usuarios de internet que emplean las tecnologías para vulnerar los sistemas en la red, robar información y/o infectarlos con comportamientos dudosos, son comúnmente conocidos como *Crackers*.

[72] Por ejemplo en el caso de X antes Twitter, podrá hacerse de tu foto de perfil, así como otros datos que incluyen historial de empleo, educación, preferencias de empleo, capacidades y habilidades, así como búsqueda de trabajo, datos que utilizaría para ofrecer servicios similares a los de Linkedln, "recomendarte potenciales trabajos, compartir con potenciales empleadores cuando solicitas un trabajo, permitir que los empleadores encuentren potenciales candidatos y mostrarte publicidad más relevante".

El término Hacker[73] en el sentido más filosófico tiende a promover una conciencia colectiva de la libertad del conocimiento y la justicia social, por lo que muchas veces se los encuentra en situaciones de activismo (llamado en este caso Hacktivismo) en pos de dicha ideología. Su forma de actuar, por lo general, determina su clasificación en:

- *Hacker* de Sombrero Blanco (White Hat): éticos, expertos en seguridad informática, especializados en realizar test de intrusión y evaluaciones de seguridad.
- *Hacker* de Sombrero Negro (Black Hat): también conocidos como *Crackers*, vulneran los sistemas de información con fines maliciosos.
- *Hacker* de Sombrero Gris (Grey Hat): en ocasiones vulneran la ley, y de forma general no atacan malintencionadamente o con intereses personales, sino que sus motivaciones se relacionan a protestas o desafíos personales.
- *Hacker* de Sombrero Verde (Green Hat): son novatos que pueden evolucionar y buscan convertirse en expertos.
- *Hacker* de Sombrero Azul (Blue Hat): son personas expertas que se les paga para probar Software en busca de errores antes de que éste salga al mercado.

Para una entidad, la fuga de información provocada por el actuar de algunos de estos usuarios de la red, puede ocurrir deliberadamente como resultado de una acción intencional de algún empleado descontento, como consecuencia de un ciberataque, o inadvertidamente, por un colaborador desprevenido víctima de un Software malicioso.

Una de las principales amenazas para los dispositivos tecnológicos utilizados para el trabajo y estudio a distancia es el Malware, también conocido como Software o código malicioso. Éste se define como cualquier programa informático que se coloca de forma oculta en un dispositivo, con la intención de comprometer la confidencialidad, integridad o disponibilidad de los datos, las aplicaciones o el sistema operativo.

[73] Existen algunos otros términos en el ciberespacio, por ejemplo: Newbie, que significa principiante; Lammer, persona que presume tener conocimientos que realmente no posee; Phreaker, Hacker orientado a los sistemas telefónicos; Script Kiddie, quien utiliza programas creados por terceros sin conocer su funcionamiento.

Los tipos más comunes de amenazas de Malware incluyen Virus, gusanos, troyanos, Rootkits y Spyware. Las amenazas de Malware pueden infectar cualquier dispositivo por medio del correo electrónico, los sitios Web, las descargas y el uso compartido de archivos, el Software punto a punto y la mensajería instantánea.

Además, existen amenazas relacionadas con la ingeniería social como el Phishing, Smishing y Vishing, por medio de los cuales los atacantes intentan engañar a las personas para que revelen información confidencial o realicen ciertas acciones, como descargar y ejecutar archivos que parecen ser benignos, pero que en realidad son maliciosos.

9.2 Las Vulnerabilidades y Exposiciones Comunes

De manera global, la última década ha sido testigo del cambio de paradigma en que los atacantes buscan explotar vulnerabilidades dentro de las organizaciones y las infraestructuras de redes. Con el fin de contrarrestar estos ataques, las políticas de seguridad persiguen constantemente aprender de ellos para estar preparados lo mejor posible, y en este sentido, intentar garantizar confianza y tranquilidad a los usuarios de la red, sobre el empleo de sus datos, finanzas y propiedades intelectuales.

Los ataques de seguridad vienen en muchas formas y usan varios puntos de entrada. Cada tipo de ataque viene en varios tipos, ya que generalmente hay más de una forma en que se pueden configurar o camuflar en función de la experiencia, los recursos y la determinación del pirata informático.

Las Vulnerabilidades y Exposiciones Comunes (Common Vulnerabilities and Exposures, CVE[74]) que tienen los distintos sistemas operativos (productos), es una lista de información registrada sobre vulnerabilidades de seguridad conocidas, en la que cada referencia tiene un número de identificación llamada CVE-ID, descripción de la vulnerabilidad, que versiones del Software están afectadas, posible solución al fallo (si existe) o como confi-

[74] https://www.cvedetails.com/
https://www.cvedetails.com/top-50-products.php
https://thebestvpn.com/vulnerability-alerts/
Hay que notar que en la base de datos se diferencian distintos productos de Windows (como Windows 7.0 u 8.1) pero no se hace lo mismo con los demás productos (por ejemplo para Debian GNU/Linux están los errores desde 1999 a la fecha) con lo que es engañosa la comparación del número vulnerabilidades para un sistema operativo con tantas versiones generadas en dicho tiempo con otro con un tiempo de vida corto.

gurar para mitigar la vulnerabilidad y referencias a publicaciones o entradas de foros o Blogs donde se ha hecho pública la vulnerabilidad o se demuestra su explotación. Además suele también mostrarse un enlace directo a la información de la base de datos de vulnerabilidades (https://nvd.nist.gov, https://openssf.org y https://docs.aws.amazon.com/security), en la que pueden conseguirse más detalles de la vulnerabilidad y su valoración.

El mundo está cada vez más interconectado y, como resultado de esto, la exposición a las vulnerabilidades de seguridad también ha aumentado dramáticamente. Las complejidades de mantener las plataformas de cómputo actuales hacen que sea muy difícil para los desarrolladores cubrir cada punto de entrada potencial. En 2019 hubo un promedio de más de 45 vulnerabilidades y exposiciones comunes registradas por día y estas siguen en aumento año con año.

El CVE-ID ofrece una nomenclatura estándar para identificación de la vulnerabilidad de forma inequívoca que es usada en la mayoría de repositorios de vulnerabilidades.

Es definido y es mantenido por The MITRE Corporation (por eso a veces a la lista se la conoce por el nombre MITRE CVE List) con fondos de la National Cyber Security Division del gobierno de los Estados Unidos de América. Forma parte del llamado Security Content Automation Protocol.

Por otro lado, como parte de las amplias revelaciones sobre vigilancia masiva filtradas en 2013, 2014 y años posteriores, se descubrió que las agencias de inteligencia estadounidenses y británicas, la Agencia de Seguridad Nacional (NSA, por sus siglas en inglés) y el Cuartel General de Comunicaciones del Gobierno (GCHQ, por sus siglas en inglés), respectivamente, tienen acceso a los datos de los usuarios de dispositivos Android. Estas agencias son capaces de leer casi toda la información del teléfono como los SMS, geolocalización, correos, notas o mensajes.

Documentos filtrados en enero de 2014, revelaron que las agencias interceptan información personal a través de internet, redes sociales y aplicaciones populares, como Angry Birds, que recopilan información para temas comerciales y de publicidad. Además, según The Guardian, el GCHQ tiene una wiki con guías de las diferentes aplicaciones y redes de publicidad para saber los diferentes datos que pueden ser interceptados. Una semana después de salir esta información a la luz, el desarrollador finlandés Rovio, anunció que estaba reconsiderando sus relaciones con las distintas plataformas publicitarias y exhortó a la industria en general a hacer lo mismo.

Las informaciones revelaron que las agencias realizan un esfuerzo adi-

cional para interceptar búsquedas en Google Maps desde Android y otros teléfonos inteligentes para recopilar ubicaciones de forma masiva. La NSA y el GCHQ insistieron en que estas actividades cumplen con las leyes nacionales e internacionales, aunque The Guardian afirmó que «las últimas revelaciones podrían sumarse a la creciente preocupación pública acerca de cómo se acumula y utiliza la información, especialmente para aquellos fuera de Estados Unidos de Norte América, que gozan de menos protección en temas de privacidad que los estadounidenses».

9.3 Alfabetismo Digital

Según la organización Common Sense Media, el alfabetismo digital es la capacidad de encontrar, identificar, evaluar y usar la información encontrada en medios digitales de manera efectiva. Básicamente, es la misma definición tradicional de alfabetismo, pero adaptada a la era digital y a fuentes no tradicionales de información.

El anuario del 2016 de la UNESCO en "Alfabetización mediática e informacional para los objetivos de desarrollo sostenible" hace referencia a las "Cinco leyes de la alfabetización mediática e informacional":

1. La información, la comunicación, las bibliotecas, los medios de comunicación, la tecnología, el internet y otras fuentes de información se encuentran en la misma categoría. Ninguna es más relevante que la otra ni debe ser tratada como tal.

2. Cada ciudadano es un creador de información o conocimiento y tiene un mensaje. Todas las personas deben estar facultadas para acceder a nueva información y expresarse.

3. La información, el conocimiento y los mensajes no siempre están exentos de valores o prejuicios. Cualquier conceptualización, uso y aplicación de alfabetismo digital debe presentar este hecho de manera transparente y comprensible para todos los ciudadanos.

4. Todo ciudadano desea conocer y comprender información, conocimientos y mensajes nuevos, así como comunicarse; y sus derechos nunca deben ser comprometidos.

5. El alfabetismo digital es un proceso dinámico de experiencias vividas. Se considera completa cuando incluye conocimientos, habilidades y actitudes, cuando abarca el acceso, la evaluación, el uso, la producción y la comunicación de información, de contenido mediático y tecnológico.

De igual manera, en el anuario de la UNESCO se exponen 10 habilidades que deben desarrollarse para lograr la alfabetización digital, o como lo define el anuario, alfabetización mediática e informacional. Estas habilidades son:

- Interactuar con información referente a los medios y la tecnología.
- Ser capaz de aplicar habilidades técnicas de comunicación de información para procesar la información y producir contenido mediático.
- Utilizar, de manera ética y responsable la información y comunicar su comprensión o conocimiento adquirido a una audiencia o lectores en una forma y medio apropiados.
- Extraer y organizar información y contenidos.
- Evaluar de forma crítica la información y el contenido presentado en los medios y otras fuentes de información, incluyendo medios en línea, en términos de autoridad, credibilidad, propósito y posibles riesgos.
- Localizar y acceder a información de contenido relevante.
- Sintetizar las ideas extraídas del contenido.
- Comprender las condiciones bajo las cuales se pueden cumplir esas ideas o funciones.
- Comprender el papel y las funciones de los medios, incluyendo medios en línea, en la sociedad y su desarrollo.
- Reconocer y articular la necesidad de información y de los medios.

Internet ha sido una herramienta que ha transformado y definido la comunicación en el siglo XXI. A través de sus múltiples interfaces, internet ha tenido éxito, para que tanto individuos como organizaciones se conecten, se comuniquen e intercambien información. Las plataformas tecnológicas y las redes sociales han acelerado la velocidad a través de la cual los usuarios

pueden acceder y recuperar información, simplificando el proceso en el que las noticias se difunden, actualizan e incluso se comunican. Hoy en día, es prácticamente instantáneo darse cuenta de un evento de noticias sin que sea necesariamente comunicado por medios tradicionales como los periódicos o la radio.

La facilidad a través de la cual las personas ahora pueden comunicarse ha traído una sensación de democratización a la libertad de expresión. Transformar la libertad de expresión ha posibilitado nuevas capacidades para crear y editar contenido, generando nuevas oportunidades para el periodismo alternativo; nuevas capacidades de organización y movilización (que apoyan en gran medida otros derechos, como la libertad de asociación); y nuevas posibilidades para innovar y generar desarrollo económico (apoyando los derechos sociales y económicos).

Sin embargo, esta facilidad en el intercambio y la creación de información también presenta desafíos tanto para las organizaciones como para las personas que son usuarias de dichas redes, tanto como fuente, como usuario final. Aunque estos desafíos varían en escala, todos son igualmente significativos. Algunos de los más destacados incluyen el desafío a la calidad superficial de la información, la susceptibilidad a la información errónea y la exposición a ataques cibernéticos. Por lo tanto, la necesidad de mitigar y mantener la integridad de esta información se ha convertido en un área de trabajo creciente para muchas organizaciones públicas y privadas.

En las siguientes secciones se ofrece una variedad de técnicas y mejores prácticas para mitigar y contrarrestar los desafíos mencionados anteriormente. Sin embargo, es importante tener en cuenta, al leer estas recomendaciones, la posición que representas o con la que te asocias. Algunas de las recomendaciones pueden no ser aplicables a una figura pública, como políticos, activistas u otros actores cuyas mejores prácticas en las redes sociales están sujetas a un mayor escrutinio. En este sentido, el ejercicio de los derechos de expresión, reunión y protesta se debe respetar, y debe ser respetado, en el ámbito digital al tiempo que se garanticen prácticas más seguras de internet.

9.4 Amenazas a la Ciberseguridad

La aparición de vulnerabilidades en los sistemas operativos y los métodos de encubrimiento de los atacantes, lo convierten en una práctica en aumento. Algunos de los principales ataques en la red, hacia donde dirigen su mirada

los *Hackers* para vulnerar la seguridad, pueden definirse como:

- Shoulder Surfing: es una técnica mediante la que el ciberdelincuente consigue nuestra información mirando "por encima del hombro" desde una posición cercana, mientras utilizamos los dispositivos sin darnos cuenta.

- Dumpster Diving: se le conoce como el proceso de buscar en nuestra basura para obtener información útil sobre nuestra persona o empresa que luego puede utilizarse contra nosotros para otro tipo de ataques.

- Malware: el término se refiere de forma genérica a cualquier Software malicioso que tiene por objetivo infiltrarse en un sistema para dañarlo. Comúnmente se asocian como tipos de Malware a los virus, gusanos y troyanos.

- Virus: es un código que infecta los archivos del sistema mediante un programa maligno, pero para ello necesita que el usuario lo ejecute directamente. Una vez activo, se disemina por todo el sistema a donde el equipo o cuenta de usuario tenga acceso, desde dispositivos de Hardware hasta unidades virtuales o ubicaciones remotas en una red.

- Gusanos: es un programa que, una vez infectado el equipo, realiza copias de sí mismo y las difunde por la red. A diferencia del virus, no necesita la intervención del usuario, ya que pueden transmitirse utilizando las redes o el correo electrónico. Son difíciles de detectar, pues su objetivo es difundirse e infectar a otros equipos, y no afectan inicialmente el funcionamiento normal del sistema. Su uso principal es el de la creación de redes zombies (Botnets), utilizadas para ejecutar acciones de forma remota como ataque de denegación de servicio (DoS) a otro sistema.

- Troyanos: similares a los virus, sin embargo, mientras que este último es destructivo por sí mismo, el troyano lo que busca es abrir una puerta trasera (Backdoor) para favorecer la entrada de otros programas maliciosos. Su misión es precisamente pasar desapercibido e ingresar a los sistemas sin que sea detectado como una amenaza potencial. No se propagan a sí mismos y suelen estar integrados en archivos ejecutables aparentemente inofensivos.

- Spyware: es un programa espía, cuyo objetivo es recopilar información de un equipo y transmitirla a una entidad externa sin el consentimiento del propietario. Su trabajo suele ser silencioso, sin dar muestras de su funcionamiento, llegando incluso a instalar otros programas sin que se perciban. Las consecuencias de su infección incluyen, además, pérdida considerable del rendimiento del sistema y dificultad para conectarse a internet.
- AdWare: su función principal es la de mostrar publicidad. Aunque su intención no es la de dañar equipos, es considerado por algunos una clase de Spyware, ya que puede llegar a recopilar y transmitir datos para estudiar el comportamiento de los usuarios y orientar mejor el tipo de publicidad.
- Ransomware[75]: este es uno de los más sofisticados y modernos ataques, ya que lo que hace es secuestrar datos (encriptándolos) y pedir un

[75] El Ransomware se ha convertido en la principal amenaza para la ciberseguridad mundial. Desde que el troyano WannaCry afectó en la primavera de 2017 al menos a 200.000 equipos y servidores de 150 países, poniendo 'contra las cuerdas' a importantes empresas, el uso de este tipo de ataques informáticos no ha dejado de aumentar y son cada vez más numerosos, sofisticados, peligrosos y masivos.

Si en sus inicios los atacantes se conformaban infectando ordenadores de consumo a cambio de unos pocos dólares, todos los informes apuntan que los ciberdelincuentes están enfocando su ámbito de actuación preferente al segmento empresarial, organizaciones, administraciones e infraestructuras públicas con Malware tan peligroso como 'CryptoWall', 'Babuk', 'Black Kingdom', 'Ryuk' o 'CryptoLocker' que destacan por su alto nivel de código y su capacidad de control de ordenadores y redes mediante el cifrado de archivos.

El número de víctimas por Ransomware es ya interminable. La última conocida (abril 2021) ha sido la cadena de tiendas Phone House, pero la lista de los últimos meses es amplísima: la multinacional Acer; el estudio CD Projekt Red; la asociación deportiva NBA; el líder en cámaras Canon; el desarrollador japonés Capcom; la firma de seguros Mapfre; municipios y hospitales estadounidenses o la infraestructura del SEPE, el Servicio Público de Empleo Estatal de España que gestiona subsidios y maneja datos personales de millones de desempleados.

Solo es un ejemplo de afectados y, además, se sospecha que otras empresas han recibido ataques, aunque han preferido no divulgarlos públicamente ante la pérdida reputacional que suponen estos incidentes que son una lacra que parece imparable. Realmente, no hay sistema operativo, plataforma, dispositivo o red informática que esté a salvo, porque el Ransomware emplea cualquier tipo de vulnerabilidad, tipo de Malware o de ataque para secuestrar los equipos. Y no en todas las ocasiones es detectado a tiempo por los sistemas de seguridad y Software antimalware.

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

rescate por ellos. Normalmente, se solicita una transferencia en dinero electrónico (Bitcoins), para evitar el rastreo y localización. Este tipo de ciberataque va en aumento y es uno de los más temidos en la actualidad.

- Stalkerware: se trata de un programa que permite el seguimiento y monitorización de la actividad del usuario en un dispositivo móvil como teléfono, tableta o computadora. El problema de este tipo de programas es que no han sido creados puntualmente para el espionaje y el acoso, sino para el intercambio de datos entre dispositivos de manera más sencilla. Y qué sí se instalan sin el consentimiento de la persona, permite espiar sus comunicaciones y toda su actividad gracias a las funcionalidades y sensores incorporados en el dispositivo.

- Escaneo de Puertos: técnica empleada para auditar dispositivos y redes con el fin de conocer qué puertos están abiertos o cerrados, los servicios que son ofrecidos, así como comprobar la existencia de algún cortafuegos (Firewall), la arquitectura de la red, o el sistema operativo, entre otros aspectos. Su empleo permite al atacante realizar un análisis preliminar del sistema y sus vulnerabilidades, con miras a algún otro tipo de ataque, pues cada puerto abierto en un dispositivo, es una potencial puerta de entrada al mismo.

- Phishing: no es un Software, se trata más bien de diversas técnicas de suplantación de identidad para obtener datos privados de las víctimas, como contraseñas o datos bancarios. Los medios más utilizados son el correo electrónico, mensajería o llamadas telefónicas, y se hacen pasar por alguna entidad u organización conocida, solicitando datos confidenciales, para posteriormente ser utilizado por terceros en su beneficio.

- Whaling: es un método para simular ocupar cargos de nivel superior en una organización con el objeto de conseguir información confidencial u obtener acceso a sistemas informáticos con fines delictivos.

- El Smishing: ocurre cuando se recibe un mensaje de texto corto (SMS)

parte o a todo el equipo apoderándose de los archivos con un cifrado fuerte y exigiendo una cantidad de dinero como "rescate" para liberarlos, el mejor (y casi único) de los consejos en ciberseguridad es la prevención con las copias de seguridad como máximo exponente. La opción de pagar el rescate para recuperar el acceso a los archivos es muy negativa para la industria ya que retroalimenta aún más esta amenaza.

en el teléfono celular, por medio del cual se solicita al usuario llamar a un número de teléfono o ir a un sitio Web.

- El Vishing: es la estafa que se produce mediante una llamada telefónica que busca engañar, suplantando la identidad de una persona o entidad para solicitar información privada o realizar alguna acción en contra de la víctima.

- Juice-jacking o Video-jacking: son los nombres que se han puesto a los procesos por los cuales, a través de un puerto (generalmente USB) nos conectamos a un puerto Hackeado, de esta forma el ciberdelincuente puede instalar, grabar datos, tomar video o sacar datos de nuestros dispositivos móviles. Esto se ha vuelto común en los puertos de recarga de energía públicos a través del puerto USB. Si se hará uso de este tipo de servicios, es recomendable adquirir un dispositivo del tipo PortaPow de carga rápida con adaptador USB que inhabilitan los pines de datos permitiendo sólo la carga del dispositivo.

- Keylogger (registrador de teclas): es un Software malicioso o dispositivo en Hardware -generalmente conectado al teclado- que se encarga de registrar las pulsaciones que se realizan en el teclado, para posteriormente usarlas para robar información privada.

- Criptomining es un Malware diseñado para la extracción de Criptomonedas en nuestros dispositivos. Si el usuario accesa a un sitio Web que esté infectado, el Malware se puede descargar de forma inadvertida a través de una descarga automática y nuestro dispositivo comenzará a desenterrar una moneda criptográfica seleccionada para los Hackers; la infección será notoria por un uso intensivo de nuestro dispositivo.

- Botnets (Redes de robots): Son computadoras o dispositivos conectados a la red (teléfonos inteligentes, tabletas, etc.) infectados y controlados remotamente, que se comportan como robots (Bots) o zombies, quedando incorporados a redes distribuidas, las cuales envían de forma masiva mensajes de correo Spam o código malicioso, con el objetivo de atacar otros sistemas o dejarlos fuera de servicio.

- Denegación de Servicios: tiene como objetivo inhabilitar el uso de un sistema o computadora, con el fin de bloquear el servicio para el que está destinado. Los servidores Web poseen la capacidad de resolver un

número determinado de peticiones o conexiones de usuarios de forma simultánea, en caso de superar ese número, comienzan a ralentizarse o incluso bloquearse y desconectarse de la red. Existen dos técnicas para este ataque: la denegación de servicio o DoS (Denial of Service) y la denegación de servicio distribuido o DDoS (Distributed Denial of Service); la diferencia entre ambos es el número de equipos de cómputo que realizan el ataque. En el primero, las peticiones masivas al servicio se realizan desde una misma máquina o dirección IP, consumiendo así los recursos que ofrece el servicio hasta que no tiene capacidad de respuesta y comienza a rechazar peticiones (denegar el servicio); en el segundo, las peticiones o conexiones se realizan empleando un gran número de computadoras o direcciones IP, todas al mismo tiempo y hacia el mismo servicio objeto del ataque, de forma general, las computadoras que lo realizan se encuentran infestadas, formando parte de una Botnet, y comportándose como zombis.

- Ataque MITM (Man In The Middle): conocido como "hombre en el medio", ocurre cuando una comunicación entre dos sistemas es interceptada por una entidad externa simulando una falsa identidad. En este sentido, el atacante tiene control total de la información que se intercambia, pudiendo manipularla a voluntad, sin que el emisor y el receptor lo perciban rápidamente. Es común que se realice empleando redes WI-FI públicas y abiertas, y es muy peligroso ya que se puede obtener información sensible de las víctimas, y es difícil identificarlo si no se poseen los mínimos conocimientos sobre el tema.

- Rootkit: es un tipo de Malware diseñado para infectar una PC, el cual permite instalar diferentes herramientas que dan acceso remoto al equipo de cómputo. Este Malware se oculta en la máquina, dentro del sistema operativo y sortea obstáculos como aplicaciones antimalware o algunos productos de seguridad. El Rootkit contiene diferentes herramientas maliciosas como un módulo para robar los números de tarjeta o cuentas bancarias, un Bot para ataques y otras funciones que pueden desactivar el Software de seguridad.

- SIM Swapping: es la duplicación del SIM de un número telefónico que permite a un atacante usurpe nuestra identidad, pudiendo autentificarse por medio de SMS en diversos servicios que usen la autentificación de dos pasos, incluyendo los servicios bancarios.

- SIM Jacker: es el envió de un mensaje malicioso al dispositivo destino, que si se abre el enlace adjunto, el dispositivo inteligente queda comprometido y se puede extraer toda la información del mismo incluyendo la ubicación.
- 0-Day (día cero): es una nueva vulnerabilidad para la cual no se han creado parches o revisiones, y que se emplea para llevar a cabo un ataque. El nombre se debe a que no existe ninguna revisión para mitigar el aprovechamiento de la(s) vulnerabilidad(es), estas pueden ser utilizadas para qué troyanos, Rootkits, virus, gusanos y otros Malwares se propaguen e infecten más equipos.
- IP Spoofing: el ciberdelincuente consigue falsear su dirección IP y hacerla pasar por una dirección válida en la cual confiamos, de este modo, consigue saltarse las restricciones y puede hacernos conectar a una Web maliciosa.
- Web Spoofing: consiste en la suplantación de una página Web real por otra falsa. La Web falsa es una copia del diseño original, llegando incluso a utilizar una URL muy similar para que demos nuestras credenciales de acceso.
- Email Spoofing: consiste en suplantar la dirección de correo de una persona o entidad de confianza para solicitarnos datos o que descarguemos archivos maliciosos.
- DNS Spoofing: a través de programas maliciosos específicos y aprovechándose de las vulnerabilidades en las medidas de protección, los atacantes consiguen infectar y acceder a nuestro Router suplantando la DNS (Domain Name System). Así cuando tratamos de acceder a una determinada Web desde el navegador, este nos lleva a otra Web elegida por el atacante.
- Sniffing: se trata de una técnica utilizada para escuchar todo lo que se transmite dentro de una red, de esta forma se monitorea el tráfico y se puede capturar información que viaja de forma no cifrada para analizarla y hacerse de nuestros datos.
- Ataque de inyección de SQL (Structured Query Language): es aprovechar una o más vulnerabilidades de servidores SQL que hace que se divulgue información confidencial de la base de datos que de otra manera

no haría al inyectar código SQL malicioso. Esto representa un riesgo enorme si la base de datos almacena información de identificación personal, como números de tarjetas de crédito, información personal y contraseñas.

- Baiting o Gancho: se sirve de algún medio físico como una unidad USB infectada que el atacante deja al alcance de los usuarios, para que cuando este lo introduzca en su equipo se infecte. También puede usar anuncios en Webs con la que promociona cursos o premios que nos inciten a compartir datos o descargar Software malicioso.

En general, para poder llevar a cabo alguno de estos ataques, los intrusos deben disponer de los medios técnicos, los conocimientos y las herramientas adecuadas, deben contar con una determinada motivación o finalidad, y se tiene que dar además una oportunidad que facilite el desarrollo del ataque, como podría ser un fallo en la seguridad del sistema informático elegido.

9.5 Dicen que si no Pagas por un Producto, Entonces el Producto Eres Tú.

¿Cuál es el modelo de negocio de empresas tecnológicas como Facebook? usar Google, Facebook, Messenger, Instagram y WhatsApp es completamente gratis. Y pese a ello, por ejemplo Facebook saca cada vez más dinero por usuario. Esto es posible gracias al uso que Facebook hace de nuestros datos. Teóricamente, y escándalos por fallos de seguridad al margen, Facebook no vende nuestros datos a terceros, sino que vende a terceros el acceso a nosotros gracias al uso de nuestros datos.

Así, las empresas tecnológicas (no importa si pagamos o no por un servicio o producto) van detectando nuestros gustos e intereses en base al dispositivo desde el que accedemos, las páginas que seguimos, nuestro historial de navegación y otros factores, y crea un perfil sobre cada uno de nosotros. Luego vende espacios de nuestro Feed a las empresas que busquen gente como nosotros (franja de edad determinada, localización, aficiones...). Y ya se está planteando formas de ir más allá mediante acuerdos con terceros, como uno con la banca para poder saber el dinero que tenemos en nuestra cuenta.

Esta red publicitaria que tan optimizada está merece el aplauso por su logro técnico, pero quizás no sea tan plausible si tenemos en cuenta ese rastreo

tan agresivo y la escasa preocupación de las empresas por la privacidad de sus usuarios.

Por otro lado, el uso generalizado de las redes sociales entraña algunos riesgos que, siguiendo recomendaciones básicas, se pueden evitar. Como cualquier comunidad frecuentada por miles de usuarios (o, como sucede a veces con las redes sociales, por miles de millones), se deben conocer los mecanismos de control y de seguridad para poder utilizarlos con fiabilidad para mantener nuestra privacidad y es por eso que el usuario tiene que ser especialmente cuidadoso con el uso que hace de la red social, a continuación daremos algunas recomendaciones:

- No necesitamos informar de todo y a todos, y menos con información sensible. Por ello, cuanta menos información pongamos, mejor.
- A menudo publicamos notas o mensajes en el muro de un amigo. Esto también es visible para otros usuarios de Facebook o de la red social que sea.
- Se dice que una foto vale más que mil palabras. Estamos dando información sobre nuestros hábitos, nuestros movimientos, a posibles atacantes.
- Hay que asegurarse de que nuestro contenido en las redes sociales sea visible solo para amigos y familiares.
- Google, Twitter, Facebook e Instagram usan el geoetiquetado mediante el GPS para ayudar a los usuarios a marcar la ubicación donde se hizo una foto, vídeo y ayudar a que el perfil sea más «social». Cualquier usuario puede interpretar fácilmente información como nuestro estado económico, estilo de vida, lugares frecuentes y la rutina diaria a través de los medios con etiquetas geográficas.
- En una red social lo normal es que cada usuario se identifique con su nombre y apellido real y que aporte datos personales, como si estudia o trabaja, con quién se relaciona o en qué ciudad vive.Esto hace que su exposición pública sea mucho mayor que antes, esto implica la pérdida del anonimato algo que antes era común y habitual en internet.
- En las redes sociales, una vez que se pulsa el botón de "publicar", esa información es enviada a los contactos del usuario. Eso significa que si

más adelante el usuario se arrepiente de lo dicho, publicado o mostrado y trata de borrarlo, solo conseguirá eliminarlo de su propio perfil pero no de internet.

- Quizás lo más importante de estos consejos para mantener la seguridad en redes sociales es el sentido común en lo que publicamos y comentamos en ellas.

Acuerdos de Privacidad la lectura de los acuerdos de privacidad[76] orientará al usuario sobre qué datos se comparten o no, y también se ofrece la opción de seleccionar o anular las opciones de privacidad, seguridad o administrativas escogidas para proteger la cuenta y el dispositivo.

Al registrarte en una cuenta de redes sociales, por defecto, toda la información anotada en un perfil se hace pública, lo que significa que cualquier persona puede acceder al contenido que hayas registrado en una cuenta. Sin embargo, las necesidades y preferencias de privacidad varían de persona a persona. Mientras que algunos usuarios prefieren tener una mayor exposición y así poder promocionar su contenido en redes sociales, otros prefieren incluir muy poca o ninguna información.

Para lograr una mayor protección del usuario y su información, es importante evaluar en qué medida la persona está dispuesta a incluir información personal en su perfil. Por consiguiente, ten en cuenta lo siguiente al:

- Seleccionar un nombre de usuario: el nombre de usuario es el "nombre digital" que una persona se asigna a sí misma o a su organización para ser identificada en línea. Si existe la preferencia de no ser fácilmente identificada en ninguna plataforma, pero poder continuar usando estas redes, la persona puede asignar y usar un seudónimo que puede estar relacionado o no con esa persona. Además, la persona puede cambiar su nombre de usuario en cualquier momento simplemente ingresando

[76] Si de verdad leyéramos los términos y condiciones de uso de plataformas Online seleccionadas (datos de abril 2020), tardaríamos (con una velocidad de 240 palabras por minuto) aproximadamente:

Microsoft 1:03:30 s (15,260 palabras), Spotify 35:48 s (8,600 palabras), Tik Tok 31:24 s (7,459 palabras), Apple 30:30 s (7,314 palabras), Zoom 30:12 s (7,243 palabras), Tinder 25:54 s (6,215 palabras), Uber 25:36 s (5,658 palabras), Twitter 25:30 s (5,633 palabras), LinkedIn 18:06 s (4,346 palabras), Facebook 17:12 s (4,132 palabras), Amazon 14:12 s (3,416 palabras), YouTube 13:42 s (3,308 palabras, Netflix 11:00 s (2,628 palabras), Instagram 9:42 s (2,451 palabras).

a la configuración de su (s) cuenta (s). El nombre de usuario no tiene que ser coherente en todas las redes sociales; estas pueden variar según las preferencias en cada una.

- Incluir una imagen en la cuenta: el usuario tiene la opción de personalizar una cuenta con la inclusión de una foto del perfil. Cuando un usuario prefiere no ser identificado, se sugiere elegir una imagen en la que no pueda ser identificado y cambiarla cuando sea necesario. Ten en cuenta que cuando se usa la misma imagen en todas las redes sociales, la simple búsqueda de imágenes puede llevar a otras cuentas.

- Incluir una ubicación: cuando se activan los servicios de ubicación en la plataforma de redes sociales, estos permiten a los usuarios rastrear el origen de cualquier actividad de medios en línea. Es importante tener en cuenta que una vez que se activa esta función, permanecerá activa hasta que se elija deshabilitarla en la configuración de privacidad. A pesar de que se permitía que esta característica estuviera activa en el pasado, las plataformas tienen la funcionalidad de deshabilitar la ubicación de cualquier contenido que se haya publicado en sus cuentas.

Sin embargo, aunque un usuario active o desactive la función de compartir la ubicación, potencialmente, la ubicación de un usuario podrá ser descubierta por el contenido que comparta o las imágenes que haya elegido para compartir.

9.6 Datos que Recopila Google

Desde el escándalo de Cambridge Analytica[77] en 2018, poco a poco todos nos hemos ido haciendo más conscientes de nuestra privacidad, y cómo las grandes empresas recopilan nuestros datos casi sin que nos demos cuenta. Ya no solo Facebook y las redes sociales, ya que hay otras empresas como Google que pueden recopilar muchos más datos sobre lo que hacemos cada día.

En parte, esto se debe a que es una empresa que ofrece una gran cantidad de servicios gratuitos que utilizamos casi sin pensar, y en otra, porque es la dueña de todo lo que buscamos en la red (si usamos sus productos y servicios).

[77] Se le acusa de tratar de influenciar los resultados de las elecciones presidenciales de 2016 en los Estados Unidos de Norteamérica.

Para hacerme una mejor idea de hasta dónde llega, podemos descargar todos nuestros datos y sí los revisamos es seguro que tendremos una desagradable sorpresa, que nos llevan a la conclusión de que Google sabe mucho más sobre nosotros que la propia Facebook, como por ejemplo, por dónde he viajado y qué aplicaciones utilizo y cuándo.

Algunos de estos datos los puedes encontrar fácilmente en algunas páginas de datos. Pero para otros, hemos utilizado la herramienta Google Takeout para descargar una copia de seguridad de todo lo que tienen sobre nosotros. Ya es bastante significativo que esta copia ocupe varios Gigas. Algo que debes saber es que puedes configurarlo para que muchos de los datos que Google tiene sobre ti se autodestruyan cuando haya pasado determinado tiempo.

Una cosa que hay que tener en cuenta es que Google no tiene por qué estar vendiendo o revisando minuciosamente estos datos, aunque tampoco hay manera de asegurarse de que no lo hagan con alguno. Sin embargo, en algunos casos nos parece excesivo incluso el simple hecho de que recopilen algunos de ellos. Para que sepas de lo que estamos hablando, aquí tienes una lista del tipo de datos personales que he visto que Google recopila.

Lo primero que llama la atención al mirar los datos de Google, es que sabe por dónde te has estado moviendo. Por ejemplo, tiene una función de cronología en la que puedes ver todos los movimientos que ha ido registrando sobre ti cuando tienes habilitado el historial de ubicaciones de Google Maps. Este suele estar activado por defecto en tus dispositivos, lo que quiere decir que si no cambias la configuración deliberadamente seguirá todos tus pasos a través de tu móvil, tableta o cualquier otro dispositivo con el que estés identificado con tu cuenta de Google.

Todos estos datos Google te los va presentando de forma ordenada y cronológica, mostrándote a qué hora has salido de un sitio, el medio de transporte por el que has ido y qué paradas has hecho. Sorprende la minuciosidad con la que Google lo registra todo. Además, en el caso de que te hayas ido de viaje sacando muchas fotos, Google también va a saber dónde y cuándo sacaste las fotografías con tu móvil, y todo te lo va a mostrar de forma ordenada viendo cómo hiciste el camino y dónde hiciste las fotos. Incluso distingue los trayectos que hiciste en coche y los que has hecho a pie. Esto lo hace utilizando los metadatos de tus fotos y cruzándolos con la información de ubicaciones que recopila.

Y más allá de esta herramienta, en la copia de seguridad de Google también puedes encontrar otras sorpresas. Por una parte, hay una carpeta con un historial de ubicaciones, en la que puedes ver las coordenadas por las que

te has movido en los últimos días y la manera en la que lo has hecho. Es como lo de Google Maps, pero con el código en bruto.

También guarda las actividades que hayas registrado utilizando Google Fit. Aquí puedes encontrar dos carpetas diferentes, una con todos los ejercicios que has hecho utilizando esta aplicación, con horas y coordenadas, y otra con un archivo diario en el que se puede ver todos los movimientos agregados en cada momento. Da igual que lleves años sin utilizar la aplicación guarda los datos obtenidos de aplicaciones de terceros al vincularlas con el servicio.

Google también tiene una sección Mi Actividad en la que recoge y almacena todas las búsquedas que haces en Internet, así como el contenido que consumes dentro de portales pertenecientes a Google. Aquí no solo vas a encontrar las búsquedas hechas a través del buscador, sino las búsquedas realizadas en Chrome y tu Android.

La verdad es que al mirar mi cronología no he podido evitar sentirme un poco incómodo, sobre todo porque todos los datos están perfectamente organizados y van acompañados de enlaces. De esta manera, puedes saber que has buscado determinados términos y volver a pulsar sobre ellos para ver los resultados de búsqueda, pero también puedes saber qué perfiles de Twitter o Facebook has visitado o las páginas a las que has entrado.

Además de esto, Google también recopila cuando utilizas otras plataformas como Stadia. Incluso si simplemente pulsas en el botón de Discover de la aplicación de Google para ver los temas que la propia Google considera importantes para ti, va recopilando cuáles son los temas sobre los que te ha mostrado información cada vez.

Es más, dentro de algunas páginas también te dice en qué secciones has navegado. Esto se vuelve un poco más preocupante con el tema de los foros, ya que te dice el título de cada hilo que has visitado dentro de un mismo foro, e incluso te ofrece enlaces para volver a él.

La Web también registra cuántas veces utilizas cada aplicación móvil y lo que es peor, todos estos datos siguen apareciendo aunque borres el historial de Chrome. He probado borrar las páginas a las que he entrado hoy con el navegador, y estas seguían apareciendo en la página de My Activity.

¿Y qué implicaciones tiene esto? Pues no sólo que Google sabe exactamente todo lo que haces en Internet si utilizas sus productos, sino que cualquiera que se ponga frente a tu computadora, tableta o teléfono inteligente (si tiene acceso a tu cuenta) va a poder saberlo, ya que al entrar a My Activity Google no pide que confirme mi identidad. Así que cualquiera puede con relativa facilidad tener acceso a esta información.

En resumen Google tiene acceso a:

- Tu nombre, tu dirección, tu edad, tu correo electrónico. Tu modelo de teléfono, tu proveedor de telefonía celular, tu plan y tu consumo telefónico y de internet.
- Las palabras que usas con más frecuencia dentro de tus correos electrónicos. Todos los correos que hayas escrito o recibido, incluido Spam. Los nombres de tus contactos y sus direcciones y teléfonos.
- Las fotografías que tomas con tu teléfono Android, aunque las hayas borrado y aunque no las subas nunca a ninguna red social. Los sitios a los que vas, dentro y fuera del país; la fecha en la que fuiste y la ruta que tomaste. Qué tan rápido llegaste. La tarjeta de crédito o débito que usas para pagar.
- Todos los sitios de internet que has visitado en Google, con qué frecuencia y lo que viste dentro de cada uno. En qué idioma buscas. A qué hora navegas. Con quién has hablado vía Hangouts. Qué videos te gustan. Qué música oyes, etc.

Para tratar de minimizar nuestra huella digital, si soy usuario de los servicios de Google puedo desactivar todos los servicios de rastreo a los que me da opción en su página de "Administrar tu cuenta de Google", no es notoria la degradación del servicio por dichas desactivaciones. Pero esto no impide que Google y servicios de terceros recolecten y transmitan nuestros datos, solo no se almacenarán en nuestra cuenta, ni tendremos acceso a los mismos.

También puedo prescindir de los servicios de Google usando otras alternativas no tan invasivas como describimos más adelante en este texto.

9.7 ¿Cómo me Protejo?

Algunas normas básicas de ciberseguridad para nuestra seguridad, tener privacidad y evitar la vigilancia son:

- Usar contraseñas largas, robustas (incorporar mayúsculas, minúsculas, números y símbolos) y diferentes para cada servicio que lo solicite, para facilitar el manejo de las contraseñas es recomendable el uso de gestores de contraseñas.

- Cifrar Dispositivos, Discos y Unidades de Respaldo para mantener la confidencialidad de nuestra información.
- Mantener actualizado el sistema operativo y las aplicaciones (sólo instalar las necesarias) de nuestros dispositivos además de usar mecanismos que coadyuven en la seguridad como cortafuegos, antivirus, entre otras aplicaciones de seguridad.
- Generar respaldos periódicos -compactados y cifrados- de nuestra información y garantizar que es posible restituir nuestros datos de dichos respaldos (una buena estrategia de respaldo es la siguiente: mantener tres copias de cualquier fichero importante -una principal y dos respaldos-, mantener los ficheros en dos tipos distintos de almacenamiento para protegerlos ante distintos riesgos y almacenar una copia de seguridad fuera de nuestra casa u oficina).
- Para ciertas actividades en la red o el uso de programas poco confiables es recomendable el uso de máquinas virtuales que limitan los posibles daños por programas maliciosos.
- Al navegar en internet es necesario hacerlo de forma segura para no exponernos de forma innecesaria, mediante un uso seguro y aceptable de las herramientas en la Nube.
- Conocer las medidas mínimas de protección para una navegación segura en internet.
- Conocer las técnicas usadas en ataques de inteligencia social, para no ser víctimas de la ciberdelincuencia.
- Al hacer uso de videoconferencias conocer las políticas de privacidad y las medidas de seguridad para no exponernos a ciberacoso y pérdida de datos.
- Hacer un uso correcto de los dispositivos personales cuando estos son usados para el trabajo.
- Uso de escritorios remotos y virtuales como una forma de mitigar los riesgos cuando se trabaja de forma remota.

Entre otras tantas cosas que el usuario actual de las Tecnologías de la Información y la Comunicación (TIC) debe dominar. Además, debemos conocer algunas recomendaciones útiles para reducir nuestra huella en internet, como:

- No compartir nuestro correo electrónico y número telefónico, pese a que es habitual utilizarlos para registrarnos en sitios Web, si los compartimos libremente por internet nos exponemos a ser víctimas de Spam, Phishing y todo tipo de ciberataques basados en ingeniería social.
- El uso de cuentas de correo alternativas nos permite registrarnos en diferentes sitios Web sin utilizar datos y/o cuentas personales o de trabajo.
- Usar cuentas de correo seguro, anónimo y bidireccional como el servicio de Tutanota, Mailfence o ProtonMail. O usar el modo confidencial -en el cual podemos establecer fecha de vencimiento y contraseña para cada correo- al enviar correos desde cuentas Google.
- Acceder a las opciones del navegador para eliminar cada cierto tiempo la información almacenada en formas de Cookies, Caché o el historial de navegación.
- El uso de modo privado o incógnito[78] en el navegador nos permite no almacenar información sobre las páginas Web visitadas ni se guardan las Cookies, ya que se eliminarán al salir del navegador.
- Evitar revelar información sensible en redes sociales sobre nosotros, familia y conocidos, en medida de lo posible debemos vigilar lo que publicamos, quién puede verlo y configurar las opciones de privacidad.

[78] En Chrome y otros navegadores Web, el modo incógnito -del cual Google sostiene que este modo- está diseñado para evitar el almacenamiento local de datos, y realmente no protege nuestra privacidad, aunque otros usuarios en el mismo dispositivo no verán la actividad en modo incógnito, los sitios Web y servicios, incluidos los de Google, aún pueden recopilar datos. La actividad, como descargas, favoritos y elementos de la lista de reproducción, se almacenará.

Cabe mencionar que ningún navegador ofrece el 100% de privacidad ni anonimato al usuario, lo que si, es que entre los diferentes navegadores existentes, podremos encontrar navegador web (como por ejemplo Brave) que ofrecen capas adicionales de protección de los datos del usuario, pero esto no los vuelve 100% eficientes en ello, pues incluso navegadores como Tor (para la Dark Web) tiene sus fallos.

- El uso de Webs seguras (https y certificados digitales) aseguran que la información que intercambiamos con ellas, como datos bancarios o contraseñas viajen de forma cifrada.
- Usar navegadores especializados (Tor) y lugares de búsqueda de información (DuckDuckGo) que nos permitan una navegación segura al no guardar lo que buscamos, ni los sitios donde accedemos.
- El uso del GPS en nuestros dispositivos móviles es una gran herramienta en nuestra vida digital, pero debemos tenerlo activado sólo cuando sea necesario, ya que muchas aplicaciones comparte nuestra ubicación en tiempo real sin nuestro conocimiento y alguien puede conocer donde vivimos, los lugares que frecuentamos o cuando no estamos en casa.
- Al tomar fotografías o vídeos desactivar el guardado de datos *Exif* (Exchangeable Image File) también conocidos como metadatos: datos sobre la cámara, datos sobre la fotografía y datos sobre el origen como ubicación por GPS, etc.
- Nunca tomar vídeos o fotos comprometedoras, de carácter íntimo o sexual y menos publicarlas, ya que suponen una gran amenaza a nuestra seguridad y pueden tener consecuencias muy graves, como la exposición pública, extorsión o el ciberacoso.
- No compartir fotos, vídeos o audios de menores.
- No debemos compartir vídeos, fotos o audios de terceros sin su aprobación, en especial los que contienen conversaciones privadas que difundan datos personales o información que podría considerarse como revelación de secretos y que la otra persona preferiría no difundir.
- Al emitir opiniones, quejas o comentarios subidos de tono u ofensivos en medios electrónicos nos expone a ser atacados o censurados y en ciertos casos podrían ameritar acciones legales.
- No publicar documentos personales en forma de fotos, vídeos o archivos (*.Docx*, *.PDF*, etc) ya que con su revelación nos expone a una suplantación de identidad y al uso de nuestros datos de forma fraudulenta.

- Cumplir las normas mínimas de protección de dispositivos personales en redes corporativas.
- Hacer uso correcto de escritorios remotos y virtuales en equipos personales para el trabajo remoto.

10 Consideraciones y Comentarios Finales

Los paquetes comerciales -de Software privativo- en general proveen un ambiente integrado de trabajo ideal para las empresas de todo tipo, pero además puede ser usado en la preparación de estudiantes para aplicar sus conocimientos al egresar en las diversas áreas de las carreras universitarias, esto les permite laborar en empresas pequeñas, medianas y grandes con un mínimo de capacitación técnica adicional.

En un mercado tan competitivo como el actual, las organizaciones actuales focalizan sus recursos en las estrategias más adecuadas para conducir a la compañía hacia el éxito. Los paquetes comerciales y los incipientes paquetes de Software libre pueden ayudar a conseguir este objetivo, completando la inversión ya realizada en sistemas operacionales.

Pero el hecho de que las organizaciones actuales, manejan una gran cantidad de información, la cual puede o no estar dispersa en sus múltiples sistemas operacionales, requiere usar paquetes que tengan integrado el manejo de las grandes bases de datos distribuidas o centralizadas, esta integración ofrece beneficios adicionales.

Por otro lado, notemos que, una vez que un producto de Software libre ha empezado a circular, rápidamente está disponible a un costo muy bajo. Al mismo tiempo, su utilidad no decrece. El Software libre, en general, podría ser considerado un bien de uso inagotable, tomando en cuenta que su costo marginal es pequeño y que no es un bien sujeto a rivalidad (la posesión del bien por un agente económico no impide que otro lo posea).

Puesto que el Software libre permite el libre uso, modificación y redistribución, a menudo encuentra un hogar entre usuarios para los cuales el coste del Software no libre es a veces prohibitivo, o como alternativa a la piratería (véase 11.5). También es sencillo modificarlo localmente, lo que permite que sean posibles los esfuerzos de traducción a idiomas que no son necesariamente rentables comercialmente.

La mayoría del Software libre se produce por equipos internacionales que cooperan a través de la libre asociación. Los equipos están típicamente compuestos por individuos con una amplia variedad de motivaciones, y pueden provenir tanto del sector privado, del sector voluntario o del sector público.

En México el Software libre nació en las Universidades y los Centros de Investigación. Es por eso que, desde hace cuatro décadas, los estudiantes y los profesores usan Software libre para fines didácticos y de investigación. Las universidades suelen optar por el uso de Software libre en vez de utilizar

Software privativo porque satisface de una mejor manera sus necesidades de cómputo, dada su naturaleza de apertura del código y la libertad de compartir los resultados obtenidos. De forma colateral, no se tienen gastos adicionales derivados del pago de licenciamientos.

Computólogos, físicos, químicos, matemáticos, otros profesionistas y científicos utilizan Software libre como herramienta de investigación y creación. Un claro ejemplo de ello es la llamada Delta Metropolitana, que es una red de supercomputadoras que están en varios puntos de la Ciudad de México, en el CINESTAV, el IPN, la UAM y la UNAM. Esa red de supercómputo utiliza Software libre para consolidar sus recursos, hacer investigación y generar conocimiento.

Por otro lado, dadas las características del Software de código cerrado, un usuario común ignora absolutamente el contenido del mismo y por tanto si existe dentro de las líneas del código alguna amenaza contra su equipo o su información, el usuario no sólo tiene prohibido el intentar eliminar o cambiar esa parte del código sino que puede ser perseguido por la ley por el hecho de intentar conocer si existe tal amenaza en dicho Software.

Además, en una sociedad de la información, el Software se ha convertido en una herramienta importante de productividad y una licencia de Software privativo constituye un acuerdo o contrato entre dos sujetos jurídicos que voluntariamente acuerdan las condiciones de uso de un programa, pero el costo económico de dicha licencia es cada vez más alto y en el caso de instituciones educativas, gubernamentales y sociedades civiles es en la mayoría de los casos -por decir lo menos- prohibitivo.

Hace un tiempo, en varios periódicos de circulación nacional (véase [93]) fue publicado el siguiente anuncio:

> El Instituto Mexicano de la Propiedad Industrial (IMPI) anunció que en las próximas semanas dará inicio una serie de clausuras a negocios que utilicen Software licenciado de manera ilegal; esto como parte del acuerdo que tiene la dependencia con The Software Alliance (BSA) desde el 2002, el cual busca fomentar el uso de programas informáticos legales y disminuir el índice de piratería en el país.
>
> De acuerdo a la BSA, el porcentaje de Software ilegal utilizado en el territorio aún se ubica en un 56 por ciento, cifra considerablemente menor a lo visto en el 2005, cuando el número ascendía

> a más del 65 por ciento. Sin embargo, México continúa siendo uno de los mayores compradores de piratería a nivel mundial, y lo que se busca con este tipo de acciones en el 2013 es disminuir, al menos, un punto porcentual.
>
> "Si como consecuencia de una visita de inspección completa se encuentra la existencia de Software ilegal, se procede a la sanción. En el 2012 incrementaron hasta un 200% las sanciones por el uso ilegal de Software", dijo Kiyoshi Tsuru, director general en México de la BSA.
>
> Aquí es donde algunos se preguntarán, ¿y qué autoridad tiene The Software Alliance para ejecutar estas determinaciones? Pese a que cuenta con el apoyo de empresas como Microsoft, Apple, Autodesk, Adobe, Aveva, AVG, CISCO, Dell, Hewlett Packard, IBM, SAP y Symantec, lo cierto es que la BSA no puede clausurar legalmente ningún negocio. La verdadera autoridad llega en su acuerdo con el IMPI, el cual sí tiene las facultades para aplicar sanciones.

Además, la UNAM, desde junio 9 del 2009 firmó un acuerdo (véase [94]):

> Con el objetivo de fomentar la cultura de la legalidad en lo que se refiere al uso del Software entre los estudiantes, la Universidad Nacional Autónoma de México y la Business Software Alliance (BSA) firmaron un convenio general de colaboración.
>
> Mediante este acuerdo, se promoverá el uso ético de las tecnologías de la información y comunicación, y se compartirán conocimientos en materia de propiedad intelectual y Software, a fin de apoyar el desarrollo y explotación de bienes digitales en la UNAM, así como definir programas para contribuir al avance de un mundo digital seguro, informaron ambas organizaciones en un comunicado.
>
> El secretario general de la máxima casa de estudios, Sergio M. Alcocer Martínez de Castro, reconoció que la UNAM necesita hacer un esfuerzo en el ámbito institucional y en cada una de las entidades que la conforman, para brindar educación a sus alumnos, profesores y trabajadores en este campo.

> "Se pretende", destacó, "que el convenio sea operativo y que se desarrolle en cada una de las entidades con la impartición de cursos y capacitación y en una rendición de cuentas para que el Software que se utilice sea legal".
>
> El funcionario reconoció a los miembros de Business Software Alliance en Latinoamérica, y apuntó que la Universidad Nacional hará lo necesario para facilitar el uso responsable, ético y seguro del Software.
>
> Informó también que ambas partes se reunirán seis meses después del inicio de este convenio de colaboración para analizar los avances.
>
> En tanto, el director General de BSA-México, Kiyoshi Tsuru Alberú, resaltó que con la firma de este convenio podrán impulsar un plan conjunto relacionado con alta tecnología, ética y legalidad "Estamos seguros que en el mediano plazo se tendrán resultados importantes y se podrá hacer la diferencia", señaló.
>
> Por su parte, el abogado general, Luis Raúl González Pérez, comentó que la UNAM busca difundir estos valores entre su comunidad, y llegar especialmente a los estudiantes que inician el bachillerato, porque desde edad temprana es importante fomentar la cultura de la legalidad.

Ante este escenario, una alternativa viable podría ser optar por el Software libre, aunque, pese a su incipiente desarrollo es seguro que en un futuro podría alcanzar a suplir todas las necesidades básicas de los usuarios, dejando la adquisición de paquetes especializados sólo para los cursos avanzados que justifique el uso de Software privativo.

10.1 El Cómputo en Instituciones Educativas

Hace algunos años la disposición de un equipo de cómputo por cada estudiante era algo difícil de satisfacer para las instituciones educativas. Ahora, las cosas son distintas, cada vez más estudiantes disponen y tienen acceso a dispositivos de cómputo -computadoras de escritorio, portátiles, tabletas, y teléfonos inteligentes- que en principio pareciera que permitirían satisfacer la creciente demanda de recursos computacionales de los estudiantes.

Pero una computadora requiere de un sistema operativo además de los diversos paquetes de Software -que estén disponibles para esa versión del sistema operativo- que permitan resolver los problemas para los cuales usa el equipo de cómputo. Aquí es donde empiezan los problemas para los usuarios de equipos de cómputo, puesto que hay una gran cantidad de equipos de cómputo con diversas tecnologías y recursos que soportan alguna versión de sistema operativo acorde a los recursos computacionales del equipo adquirido que no necesariamente soportan a todos y cada uno de los programas de cómputo que el usuario requiere.

Ante la creciente necesidad de programas de cómputo podríamos pensar en que cada usuario que requiera hacer uso de ellos tenga acceso a un equipo de cómputo adecuado, conjuntamente con el sistema operativo que lo soporte. Pero esto dista mucho de la realidad, puesto que la gran mayoría de los usuarios no pueden hacer esos gastos y menos una institución educativa y sus respectivos estudiantes.

¿Entonces qué opciones tenemos para satisfacer la creciente demanda de recursos computacionales?

- Por un lado, si ya disponemos de un equipo de cómputo con su respectivo sistema operativo, entonces hacer uso de sólo aquellos programas de cómputo que nuestro equipo soporte, teniendo cuidado de no instalar programas de cómputo antagonistas.

- Otra opción es, si ya disponemos de un equipo de cómputo, entonces tener dos o más versiones de sistema operativo que permitan instalar una mayor diversidad de programas de cómputo y tener el cuidado de no instalar programas de cómputo incompatibles. Así, dependiendo de nuestras necesidades podemos hacer uso de uno u otro sistema operativo y sus respectivos programas.

- La opción más viable, es una que conjugue las dos anteriores. Pero además, podríamos emular Hardware del que no disponemos mediante el uso de máquinas virtuales, escritorios remotos y virtuales que nos permitirían en un sólo equipo de cómputo usar simultáneamente diversos sistemas operativos para distintas arquitecturas y sus respectivos programas que ahora es posible instalar en las máquinas virtuales programas de cómputo incompatibles de forma aislada unos de otros.

Usando esta última opción es posible satisfacer en un sólo equipo de cómputo una gran variedad de necesidades computacionales. Esto permite que a nivel de usuario (estudiante, ayudante y profesor) o institución educativa, el equipo de cómputo usando Software de virtualización pueda proporcionar un marco que permita satisfacer las diversas y crecientes necesidades computacionales. Pero hay que notar que aún esta opción no está exenta de problemas legales y técnicos, pero en principio es una opción viable para la gran mayoría de los usuarios y la institución educativa.

Tomando esto en cuenta, es viable tener una cantidad adecuada de paquetes de cómputo, que permitieran satisfacer las necesidades especializadas de la gran mayoría de los cursos y estos estar instalados en aquellos espacios en los cuales se asignarían los cursos, además de las áreas comunes de cómputo en la que los estudiantes requiriesen hacer uso de dichos paquetes. Además, de proporcionar un mecanismo para que los profesores y ayudantes que requieran enseñar algo con alguna versión privativa que no se disponga, sea implementada -en medida de lo posible- en los paquetes disponibles.

Pero hay que hacer notar, que no todas aquellas funciones que hace una versión particular de un paquete, es posible hacerlas con otras versiones o paquetes alternativos. Esto es muy común con ciertas actividades especializadas -al hacer cálculo simbólico, cálculo numérico, manejo de datos y trabajar en entornos de desarrollo-. Ello implicaría, por un lado restringir el Software instalado en los equipos de cómputo o por el otro instalar todas y cada una de las solicitudes de Software, aún cuando se requiera más de una versión de un paquete particular.

El restringir el Software instalado, impediría al profesor -que así lo requiera por la libertad de cátedra- enseñar aquello que considera que es necesario -en particular el manejo de uno o más paquetes especializados de cómputo- para proporcionar las herramientas básicas a sus alumnos y que estos deben de dominar para aprobar su curso.

En el caso de dar flexibilidad, para que cada profesor solicite la instalación del paquete o los paquetes que requiera para sus cursos, implica que el Software solicitado puede o no contar con licencia adecuada de uso. Así, se estaría permitiendo que se tenga instalado Software del que se viola la licencia de uso.

En cuanto a tener la lista definitiva de Software que usarán todos y cada uno de los profesores o ayudantes de los cursos asignados a un espacio es difícil tener antes del inicio del curso -por la constante evolución del Software y las cambiantes necesidades de la enseñanza-, además de depender de la forma de

asignación de estos en los laboratorios y talleres de cómputo. En cuanto a la solicitud para hacer la instalación correspondiente, se requiere tener certeza de en qué espacio serán asignados todos y cada uno de los cursos.

Por ello se han buscado opciones[79] -no siempre las más adecuadas o lícitas (véase 11.5)- para que sin importar en qué espacio sea asignado el curso -siempre y cuando el equipo de cómputo lo soporte- se tenga desde los primeros días de uso del espacio el paquete solicitado y en casos excepcionales el tiempo de espera sea menor a unos horas o días sin importar la plataforma -Windows o Linux- o el tipo de Software solicitado -libre o privativo-.

Por ejemplo, se puede optar por la virtualización[80], usando como sistema operativo base Debian GNU/Linux estable, instalando como paquete de virtualización a KVM/QEMU. Aquí, se montarían las múltiples máquinas virtuales que serían ejecutadas según las necesidades del usuario -para cualquier versión de Windows, Linux u otro sistema operativo de cualquier arquitectura de Hardware soportada por QEMU-. Para controlar la actualización de las máquinas virtuales sin que se requiera intervención del usuario, se usaría RSYNC tunelizado mediante SSH que sincronizaría las máquinas virtuales y la configuración del equipo base de forma remota.

Para tener la flexibilidad anteriormente comentada, es necesario poder contar con distintas versiones de sistemas operativos, de cada una de las versiones -en caso de Windows, tener independientemente los Service Pack-. De tal forma que sea posible instalar cada versión de Software solicitada en la plataforma adecuada, teniendo en cuenta que muchas versiones del Software son mutuamente excluyentes para ser instaladas en una misma versión del sistema operativo simultáneamente.

Por todo lo anterior, el uso de máquinas virtuales -que permiten tener múltiples versiones de sistemas operativos independientemente, así como de una versión particular tener por separado cada una de ellas con los respectivos Service Pack- es una opción viable para proporcionar el servicio de

[79] En el caso que el equipo sólo tenga un sistema operativo sin virtualización, es necesario esperar a que las asignaciones de los cursos y sus respectivas peticiones de uso de paquetes de cómputo estén completas, para entonces proceder a realizar instalación del Software que no sean antagónicos. Nótese que, por lo general, los cursos requieren el uso de los equipos de cómputo y el Software solicitado de forma inmediata, por lo cual esperar tiempo (días) para tener acceso al mismo no es una opción viable.

[80] Una vez creada la máquina virtual, esta es un archivo que puede ser copiado o descargado de la red, por ello el usuario -estudiante, ayudante o profesor- puede llevarse la máquina virtual para hacer uso de ella en el equipo al que tenga acceso, teniendo como único requisito tener instalado el programa de virtualización.

instalación centralizada de los diversos paquetes de cómputo solicitados por los profesores de las diversas carreras universitarias. Esta opción minimiza los tiempos de espera para la instalación de un paquete en particular y agiliza las prestaciones a todos y cada uno de los grupos que se atienden semestralmente en los cientos de equipos en los laboratorios y talleres de cómputo.

10.2 Integración del Cómputo en Ciencias e Ingenierías

El uso de programas de cómputo está integrado a las carreras de Ciencias e Ingenierías desde hace mucho tiempo, pero la gran mayoría se realiza con productos propietarios, lo cual no representa ningún problema técnico, pero sí un problema para la institución y estudiantes, ya que las versiones actualmente usadas, no son del todo compatibles entre sí, ello implica que se requiere o tener la última versión del producto o diferentes versiones del mismo para trabajos cotidianos en una misma computadora.

El uso de programas de cómputo de Software libre está cada día más integrado al uso cotidiano que hacen profesores, ayudantes y estudiantes en la carreras de Ciencias e Ingenierías, pero todavía para el Sistema Operativo Windows, así como para paquetes de uso común, no ha sido posible encontrar un adecuado reemplazo, los más comunes son MATLAB, Mathematica, Maple, SPSS, SAS y Microsoft Office.

Para las Universidades, el contar con las licencias necesarias para que cada máquina a la que los alumnos tienen acceso cuente con una, es en extremo prohibitivo por el costo. Esto mismo sucede en el caso de los estudiantes, pues el costo de una sola licencia para uso académicos es onerosa más si consideramos la diversidad de programas requeridos para una sola meteria y esto pasa con cada uno de los cursos de la carrera.

Es por ello que el uso de herramientas de Software libre se visualiza como un reemplazo natural a los paquetes propietarios, pero la realidad dista de ser tan simple. Ya que, actualmente no es posible obtener las características mínimas en Software libre para que puedan ser un reemplazo real de los paquetes de propietarios. Este hecho ha ocasionado que existe un uso cada vez más generalizado entre profesores y alumnos a usar Software sin la licencia respectiva (véase 11.5).

por ejemplo, en la UNAM, a través de la Dirección General de Cómputo y de Tecnologías de la Información y Comunicación se dispone de un restringido número de paquetes y versiones que son puestos a disposición de la comunidad universitaria para usar en los equipos personales sin aparente costo para el

usuario final -pero el costo de dichos paquetes son deducidos por la empresa como una donación, lo cual sí implica un costo real que se deduce en el ejercicio fiscal de la empresa donante y éste repercute en los ingresos que el gobierno no recaudará por motivo de impuestos-.

10.3 Ventajas, Desventajas y Carencias del Software Libre

Notemos que la ventaja de tener múltiples herramientas para realizar operaciones elementales y avanzadas de paquetes de cálculo numérico, simbólico, estadístico y ofimático es en sí misma una gran ventaja. Para los centros universitarios y usuarios ocasionales, las herramientas de Software libre son una herramienta invaluable, en el caso de empresas que requieren usar opciones avanzadas o generadas por terceros, los paquetes propietarios destacan como herramientas de trabajo óptimas. Pero para todos los casos, hay que destacar:

- Funcionalidades básicas: Todos los paquetes implementan las funcionalidades básicas, ya que todos los paquetes llevan años desarrollándose.
- Funcionalidades avanzadas: Por mucho, los paquetes propietarios tienen implementadas cientos de funciones avanzadas que pueden ser muy útiles para usuarios avanzados, pero rara vez son usados por los usuarios noveles o cotidianos.
- Fiabilidad: En los paquetes en desarrollo son comunes las caídas del programa, pero en los de Software propietario se destaca por ser más fiable que los demás.
- Información: El Software propietario son paquetes con una abundante bibliografía y la propia ayuda del programa.
- Facilidad de Manejo: Ninguno de los programas presenta grandes dificultades a la hora de su uso. Pero en menor o mayor medida, todos los paquetes del Software libre presentan entornos de desarrollo funcional, pero perfectible.
- Costo: El costo de las diversas versiones de Software propietario suele ser prohibitivo para instituciones educativas y usuarios ocasionales, en

el caso del Software libre, los paquetes se pueden descargar de la red sin más costo que el acceso a Internet y los medios de instalación cuando son requeridos.

El Software libre es aún joven, en los miles de proyectos actuales se está trabajando a diario en mejorar la parte computacional de los algoritmos involucrados en el paquete, haciendo y puliendo interfaces gráficas, generando ayuda en línea así como la documentación necesaria para que usuarios noveles y avanzados usen la mayor cantidad de opciones programadas en los paquetes.

Para muestra de este maravilloso avance, tomemos el proyecto del Kernel de Linux y su uso en los sistemas operativos Android, Ubuntu, Debian GNU/Linux, que actualmente se ejecuta en millones de equipos y contiene miles de aplicaciones y están soportados por una gran cantidad de usuarios y empresas comerciales. Estos han logrado desplazar a muchos de sus competidores por sus múltiples bondades y bajo costo de desarrollo, al reusar miles de aplicaciones ya existentes que usan Software libre y permitir desarrollar otro tanto de aplicaciones bajo una plataforma que se ejecutan en los más diversos procesadores.

Así también, en los últimos años, muchos proyectos han pasados de ser simples programas en línea de comandos a complejas aplicaciones multiplataforma -ejecutan en distintos sistemas operativos como son Windows, Linux y Mac- con ambientes gráficos multimedia que en muchos casos han superado a sus contrapartes comerciales, por ejemplo los navegadores Web tipo FireFox y la suite ofimática tipo LibreOffice, entre muchos otros.

10.4 Comentarios Finales

A diferencia de otros paquetes, SPSS, SAS, Microsoft Office, etc. Ofrecen soluciones en forma de una suite completa para la gestión de información para encontrar el llamado poder del conocimiento, pero el costo de las versiones completas y aún las educativas es prohibitivo para la gran mayoría de las instituciones educativas, en particular para la UNAM. Por ello, el resto de los paquetes propietarios y libres ofrecen una ventaja competitiva, al permitir al profesor y sus estudiantes contar con versiones completas y funcionales en las que pueden ser aplicados los conocimientos adquiridos en los diversos cursos de la carrera.

Por otro lado, para reforzar la apropiación del Software libre por parte de la comunidad de la UNAM, es necesario proporcionar a la comunidad

demostraciones y cursos cortos de las herramientas de Software libre, iniciando con mostrar el uso de sistemas operativos libres basados en Linux. Ello es posible haciendo uso de los sistemas llamados Live[81], ya que cada alumno puede probar y usar el sistema operativo en conjunto con cientos de herramientas libres, sin la necesidad de instalar Software en la máquina que utilice para practicar. Cuando el alumno se sienta cómodo con el sistema, es posible ayudarlo a instalar mediante tutoriales en línea y/o presenciales el sistema en su equipo de cómputo.

Lo mismo es posible hacer, al preparar demostraciones del Software que puede reemplazar paquetes muy difundidos en la comunidad como son: MATLAB, Mathematica, Maple, SPSS, SAS y Microsoft Office. Estos cursos no necesariamente se centrarían en las similitudes o diferencias entre paquetes libres y propietarios, más bien, para cautivar a usuarios noveles y futuros ayudantes a dar cursos completos de las herramientas libres mostrando su

[81] Un Live CD/DVD o USB, más genéricamente Live Distro, es un sistema operativo almacenado en un medio extraíble, tradicionalmente un CD/DVD o USB (de ahí sus nombres), que puede ejecutarse directamente en una computadora.

En la historia más reciente de Linux, las llamadas distribuciones Live Distro se han vuelto muy populares porque le permiten probar una distribución de Linux sin siquiera instalarla en el equipo. Esto es excelente porque no tiene todas las molestias de volver a particionar el disco o instalarlo sobre su sistema operativo (Windows/Mac OS). Simplemente puede colocar el CD/DVD o USB para una distribución en vivo e iniciar la computadora desde ahí. Por lo general, obtiene la mayor parte de la funcionalidad principal de la distribución, por lo que realmente puede evaluar si la distribución es para usted antes de elegir instalarla de verdad.

Normalmente, una versión Live viene acompañado de un par de aplicaciones. Algunos Live CD/DVD o USB incluyen una herramienta que permite instalarlos en el disco duro. Otra característica es que por lo general no se efectúan cambios en la computadora utilizada.

Para usar una versión Live es necesario obtener uno (muchos de ellos distribuyen libremente una imagen ISO que puede bajarse de Internet y grabarse en disco/USB) y configurar la computadora para que arranque desde la unidad lectora, reiniciando luego la computadora con el disco en la lectora o USB, con lo que el que el sistema Live se iniciará manualmente.

Uno de los mayores inconvenientes de este sistema es el mal uso de una gran cantidad de memoria RAM, una parte para su uso habitual y otra para funcionar como el disco virtual del sistema. En el arranque, se le pueden dar distintos parámetros para adaptar el sistema a la computadora, como la resolución de pantalla o para activar o desactivar la búsqueda automática de determinado Hardware.

Otro inconveniente es el rendimiento de la Live Distro, pues la velocidad de transferencia de las unidades lectoras CD/DVD o USB es muy inferior a la de los discos duros. Una vez instalada en la computadora se apreciará la velocidad real de la distribución.

aplicabilidad en diferentes ramas de las matemáticas aplicadas.

Para realizar dichos cursos, se cuenta con todos los recursos necesarios. Por un lado, se dispone de laboratorios y talleres con Software libre instalado en los equipos de cómputo, además, se pueden usar los sistemas "Live" que pueden ser proporcionados en DVDs o en unidades flash USB. Estas últimas, proporcionan mejor rendimiento, pueden ser actualizadas y reutilizados tantas veces como sea necesario para conocer uno o más sistemas operativos. Estos sistemas "Live" pueden ser generados por el propio usuario, usando las decenas de paquetes disponibles en Windows o Linux que generan sistemas "Live" a partir de las imágenes ISO bajadas de la red -por ejemplo, de sistemas operativos como Ubuntu, Debian, etc.-.

De esta forma, se puede coadyuvar a que alumnos, ayudantes y profesores conozcan el mundo del Software libre, para que con el tiempo se adopte su uso, sin dejar de lado, el proporcionar cuando sea necesario, cursos de Software privativo pero siempre teniendo en cuenta que se puede -en medida de lo posible- trabajar con paquetes alternativos, como los que proporciona el Software libre.

Además, el Software libre ofrece una ventaja competitiva, al permitirle al profesor y sus estudiantes contar con versiones completas y funcionales en las que pueden ser aplicados los conocimientos adquiridos en los diversos cursos de las carreras de Ciencias e Ingenierías, dejando el manejo especializado de paquetes a cursos avanzados o para cuando el educando realice sus prácticas profesionales. De esta forma se pueden preparar a los estudiantes para aplicar sus conocimientos al egresar en diversas áreas de la carreras de Ciencias e Ingenierías y con pocos conocimientos técnicos adicionales puedan laborar en pequeñas, medianas y grandes empresas.

11 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[82], 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.

[82] 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.

11.1 Software Propietario

No existe consenso sobre el término a utilizar para referirse al opuesto del Software libre. La expresión «Software propietario (Proprietary Software)» (véase [6]), 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.

11.2 Software Libre

La definición de Software libre (véase [11], [12], [4], [5], [3] y [7]) 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 [11]), 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[83]; 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

[83] ¿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.

11.3 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.

11.4 Tipos de Licencias

Tanto la Open Source Initiative como la Free Software Foundation mantienen en sus páginas Web (véase [11], [12], y [7]) 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.

11.4.1 Licencias Creative Commons

Las Licencias[84] 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:

[84] 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[85], 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.

[85] 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.

11.4.2 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.

11.5 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-.

11.5.1 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.

11.5.2 ¿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.

11.5.3 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.

11.5.4 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.

11.5.5 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.

11.6 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.

11.6.1 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.

11.6.2 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.

12 Apéndice B: Máquinas Virtuales

Entendamos por una máquina virtual a un programa de cómputo (véase [42], [43], [41] y [40]) que simula a una computadora, en la cual se puede instalar y usar otros sistemas operativos de forma simultánea como si fuese una computadora real sobre nuestro sistema operativo huésped[86].

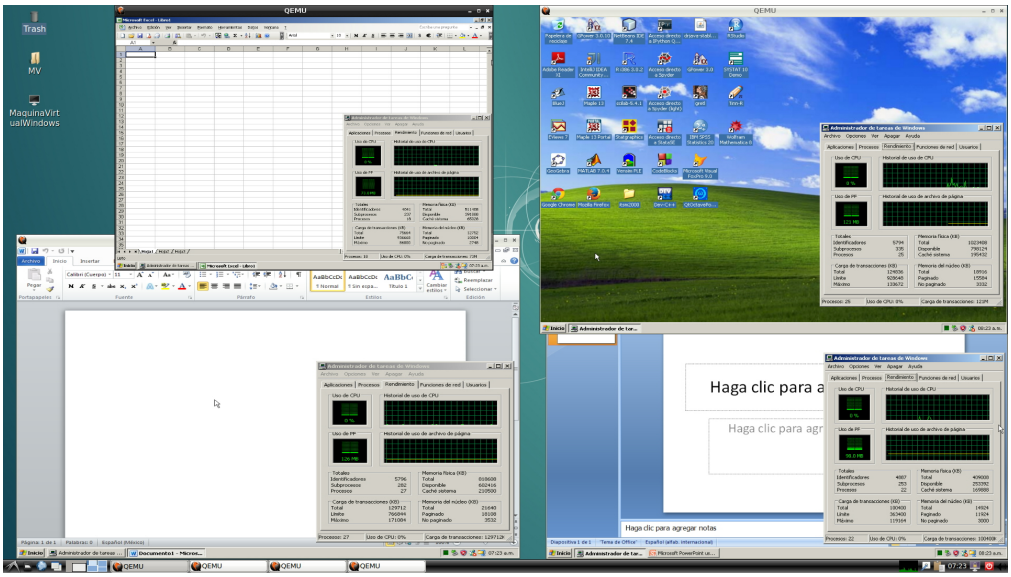

Figura 1: Sobre un equipo AMD de gama baja y 4 GB de RAM, usando como sistema operativo huésped un Linux Debian estable, se ejecutan 4 máquinas virtuales (mediante KVM) de Windows XP con diferentes aplicaciones y dentro de cada una de ellas se muestra la RAM asignada, la usada en ese momento, el uso de CPU de cada máquina virtual, entre otros datos.

Una característica esencial de las máquinas virtuales es que los procesos que ejecutan están limitados por los recursos y abstracciones proporcionados por ellas. Estos procesos no pueden escaparse de esta "computadora virtual". Uno de los usos más extendidos de las máquinas virtuales es ejecutar sistemas operativos nuevos u obsoletos adicionales a nuestro sistema habitual.

[86] Tal y como puede verse reflejado en la definición de máquina virtual, en este texto nos estamos focalizando en las máquinas virtuales de sistema. Existen otro tipo de máquinas virtuales, como por ejemplo las máquinas virtuales de proceso o los emuladores.

De esta forma podemos ejecutar uno o más sistemas operativos -Linux, Mac OS, Windows XP, 7 ó 8- desde nuestro sistema operativo habitual -GNU/Linux, Unix o Windows- sin necesidad de instalarlo directamente en nuestra computadora y sin la preocupación de que se desconfigure el sistema operativo huésped o a las vulnerabilidades del sistema virtualizado, ya que podemos aislarlo para evitar que se dañe.

12.1 Tipos de Máquinas Virtuales

Las máquinas virtuales se pueden clasificar en dos grandes categorías según su funcionalidad y su grado de equivalencia a una verdadera máquina:

- Máquinas virtuales de sistema (en inglés System Virtual Machine). También llamadas máquinas virtuales de Hardware[87], permiten a la má-quina física subyacente compartirse entre varias máquinas virtuales, cada una ejecutando su propio sistema operativo[88]. A la capa de Software que permite la virtualización se le llama monitor de máquina virtual o Hypervisor. Un monitor de máquina virtual puede ejecutarse o bien directamente sobre el Hardware o bien sobre un sistema operativo ("Host Operating System").

- Máquinas virtuales de proceso (en inglés Process Virtual Machine). A veces llamada "máquina virtual de aplicación", se ejecuta como un proceso normal dentro de un sistema operativo y soporta un solo proceso. La máquina se inicia automáticamente cuando se lanza el proceso que se desea ejecutar y se detiene para cuando éste finaliza. Su objetivo es el de proporcionar un entorno de ejecución independiente de la plataforma de Hardware y del sistema operativo, que oculte los detalles de la plataforma subyacente y permita que un programa se ejecute siempre de la misma forma sobre cualquier plataforma.

[87] La virtualización puede se por Software o con apoyo mediante el Hardware, en este último caso se obtienen varios órdenes de magnitud de rendimiento que por Software.

[88] Que sus componentes sean virtuales no quiere decir necesariamente que no existan, por ejemplo una máquina virtual puede tener unos recursos reservados de 2 GB de RAM y 20 GB de disco que se obtienen del equipo donde este ejecutando la máquina virtual. Otros dispositivos podrían realmente ser inexistentes físicamente como por ejemplo un CD-ROM que en verdad es el contenido de una imagen ISO en vez de un lector de CD de verdad.

12.2 Técnicas de Virtualización

Las Máquinas Virtuales ya tienen más de 60 años de vida su origen en los primeros días de la informática en la década de 1960, cuando el tiempo compartido para los usuarios de la computadora central era un medio de separar el Software de un sistema anfitrión físico. La máquina virtual se definió a principios de los 70 como "un duplicado eficiente y aislado de una máquina de computación real".

Las máquinas virtuales tal como las conocemos hoy en día han cobrado fuerza en los últimos 20 años a medida que las empresas adoptaron la virtualización de servidores para utilizar la potencia de computación de sus servidores físicos de manera más eficiente, reduciendo la necesidad de servidores físicos y ahorrando así espacio en el centro de datos. Debido a que las aplicaciones con diferentes requisitos de sistema operativo podían funcionar en un solo Host físico, no se requería un Hardware de servidor diferente para cada una.

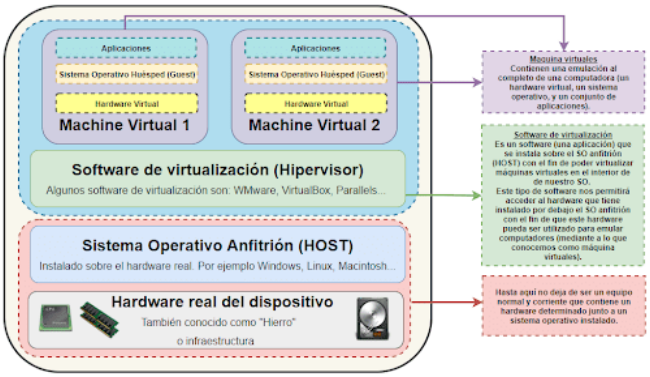

Figura 2: Qué es una Máquina Virtual

Básicamente se reconocen tres tipos de virtualización, algunas de las cuales son usadas actualmente en la gran mayoría de los sistemas operativos, ge-neralmente sin que el usuario esté consciente de que usa virtualización[89], estos son:

[89] El ejemplo más común y omnipresente es la máquina virtual del lenguaje de programación de JAVA o .Net Framework.

Emulación del Hardware Subyacente (ejecución nativa) Esta técnica se suele llamar virtualización completa -Full Virtualization- del Hardware, y se puede implementar usando un Hipervisor de Tipo I o de Tipo II:

1. Monitor de tipo I, se ejecuta directamente sobre el Hardware.
2. Monitor de tipo II, se ejecuta sobre otro sistema operativo.

Cada máquina virtual puede ejecutar cualquier sistema operativo soportado por el Hardware subyacente. Así los usuarios pueden ejecutar dos o más sistemas operativos distintos simultáneamente en computadoras "privadas" virtuales. Actualmente tanto Intel como AMD han introducido prestaciones a sus procesadores x86_64 para permitir la virtualización de Hardware.

Emulación de un Sistema no Nativo Las máquinas virtuales también pueden actuar como emuladores de Hardware, permitiendo que aplicaciones y sistemas operativos concebidos para otras arquitecturas de procesador se puedan ejecutar sobre un Hardware que en teoría no soportan. Esta técnica permite que cualquier computadora pueda ejecutar Software escrito para la máquina virtual. Sólo la máquina virtual en sí misma debe ser portada a cada una de las plataformas de Hardware.

Virtualización a Nivel de Sistema Operativo Esta técnica consiste en dividir una computadora en varios compartimentos independientes de manera que en cada compartimento podamos instalar un servidor. A estos compartimentos se les llama "entornos virtuales". Desde el punto de vista del usuario, el sistema en su conjunto actúa como si realmente existiesen varios servidores ejecutándose en varias máquinas distintas.

12.3 Otras Formas de Virtualización

El éxito de las máquinas virtuales en la virtualización de servidores llevó a aplicar la virtualización a otras áreas, como son el almacenamiento, las redes o las computadoras de escritorio. Lo más probable es que, si hay un tipo de Hardware que se está utilizando en el centro de datos, se esté explorando el concepto de virtualizarlo.

Virtualización de escritorios Implementar los escritorios como un servicio gestionado permite a las organizaciones de tecnología de la información responder más rápido a las necesidades cambiantes del entorno de trabajo y a las nuevas oportunidades. Los escritorios y las aplicaciones virtualizados también pueden distribuirse de forma rápida y sencilla a sucursales, empleados subcontratados o en otros países y trabajadores móviles que utilizan dispositivos móviles como tabletas iPad y Android.

Virtualización de redes las empresas han explorado opciones de red como servicio y la virtualización de funciones de red -NFV, Network Functions Virtualization-, que utiliza servidores de productos básicos para sustituir los aparatos de red especializados y permitir servicios más flexibles y escalables. Esto difiere un poco de la red definida por los programas informáticos, que separa el plano de control de la red del plano de reenvío para permitir un aprovisionamiento más automatizado y una gestión de los recursos de la red basada en políticas.

Una tercera tecnología, las funciones de red virtual, son servicios basados en programas informáticos que pueden ejecutarse en un entorno NFV, incluidos procesos como el enrutamiento, el cortafuegos, el equilibrio de la carga, la aceleración de la WAN y el cifrado.

Máquinas Virtuales y Contenedores El crecimiento de las máquinas virtuales ha dado lugar a un mayor desarrollo de tecnologías como los contenedores, que llevan el concepto un paso más allá y está ganando atractivo entre los desarrolladores de aplicaciones Web. En el entorno de un contenedor, una sola aplicación, junto con sus dependencias, puede ser virtualizada. Con muchos menos gastos generales que una máquina virtual, un contenedor sólo incluye binarios, bibliotecas y aplicaciones.

Mientras que algunos piensan que el desarrollo de contenedores puede "matar" a la máquina virtual, hay suficientes capacidades y beneficios de las máquinas virtuales que mantienen la tecnología en movimiento. Por ejemplo, las máquinas virtuales siguen siendo útiles cuando se ejecutan múltiples aplicaciones juntas o cuando se ejecutan aplicaciones heredadas en sistemas operativos más antiguos.

Además, algunos opinan que los contenedores son menos seguros que los hipervisores de las máquinas virtuales porque los contenedores tienen un solo sistema operativo que las aplicaciones comparten, mientras que las máquinas

virtuales pueden aislar la aplicación y el sistema operativo.

Máquinas Virtuales, 5G y Edge Computing Las máquinas virtuales son vistas como parte de nuevas tecnologías como el 5G y el Edge Computing. Por ejemplo, los proveedores de infraestructura de escritorio virtual -VDI, Virtual Desktop Infrastructure- como Microsoft, VMware y Citrix están buscando formas de extender sus sistemas VDI a los empleados que ahora trabajan en casa a causa de la pandemia.

Como muchas otras tecnologías que se utilizan hoy en día, éstas no se habrían desarrollado si no fuera por los conceptos originales de máquinas virtuales introducidos hace décadas.

12.4 Aplicaciones de las Máquinas Virtuales de Sistema

Cada uno de los sistemas operativos que virtualizamos -con su propio sistema operativo llamado sistema operativo «invitado (Guest)»- es independiente de los otros sistemas operativos. De este modo, en caso que una de las máquinas virtuales deje de funcionar, el resto seguirá funcionando. Una máquina virtual dispone de todos los elementos de un equipo de cómputo real, de disco duro, memoria RAM, unidad de CD o DVD, tarjeta de red, tarjeta de vídeo, etc., pero a diferencia de un equipo de cómputo real estos elementos en vez de ser físicos son virtuales. Así, una vez instalado un sistema operativo en la máquina virtual, podemos usar el sistema operativo virtualizado del mismo modo que lo usaríamos si lo hubiéramos instalado en nuestro equipo de cómputo.

Varios sistemas operativos distintos pueden coexistir sobre la misma computadora trabajando simultáneamente, en sólido aislamiento el uno del otro, por ejemplo para probar un sistema operativo nuevo sin necesidad de instalarlo directamente. La máquina virtual puede proporcionar una arquitectura de instrucciones que sea algo distinta de la verdadera máquina, es decir, podemos simular Hardware. Además, todos los elementos de una máquina virtual se encapsulan en un conjunto pequeño de archivos -en KVM/QEMU es solo un archivo-, esto permite que podamos pasar un sistema operativo virtual de un equipo de cómputo a otro y realizar copias de seguridad de forma fácil y rápida.

La gran mayoría de los manejadores de máquinas virtuales -KVM, Vir-

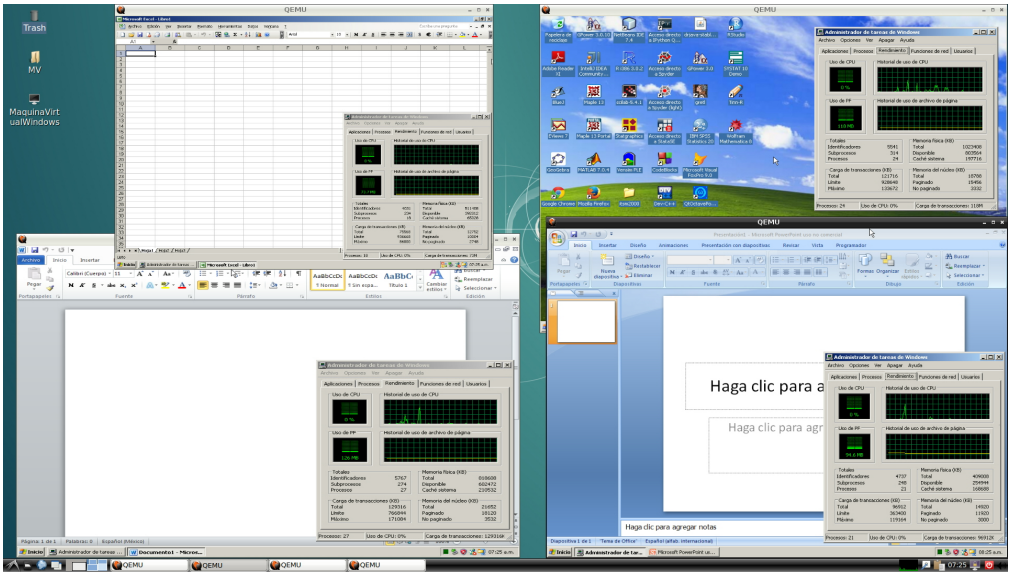

Figura 3: Al poder correr diferentes sistemas operativos y/o versiones del mismo en donde podemos instalar diversas aplicaciones antagónicas que no podrían coexistir en un sólo sistema operativo, nos permite ampliar el uso de nuestro equipo de cómputo.

tualBox o VMWare- permiten instalar prácticamente cualquier sistema operativo -por ejemplo Linux, Android, Mac OS X, Windows, Chrome OS, etc.- . Sin embargo existen otros manejadores de máquinas virtuales -Virtual PC, Hiper-V o Parallels- que están principalmente destinados a virtualizar Windows.

La virtualización es una excelente opción hoy en día, ya que las máquinas actuales -Laptops, Desktops y servidores- en la mayoría de los casos están siendo "subutilizados" -estos cuentan con una gran capacidad de cómputo, disco duro y memoria RAM- ya que no se utilizan todos los recursos todo el tiempo, teniendo un uso promedio que oscila entre 30% a 60% de su capacidad total. Permitiendo así, ejecutar varias máquinas virtuales en un sólo equipo físico aumentando el porcentaje de uso de los recursos de cómputo disponibles -en el caso de virtualizar servidores, a este proceso se le conoce como consolidación de servidores-. Así, la consolidación de servidores contribuye a reducir el coste total de las instalaciones necesarias para mantener los servicios, permitiendo un ahorro considerable de los costos asociados -

energía, mantenimiento, espacio, administración, etc.-, esto se hace patente en la «computación en la nube (Cloud Computing)» muy en boga actualmente.

12.5 Ventajas y Desventajas

Como toda tecnología, la virtualización tiene ventajas y desventajas, las cuales deben de ser sopesadas en cada ámbito de implementación. Lo que es un hecho que permite en un mismo equipo de cómputo ejecutar más de un sistema operativo o distintas versiones del mismo.

Pero queda claro que uno de los inconvenientes de las máquinas virtuales, es que agregan gran complejidad al sistema en tiempo de ejecución. Esto tiene como efecto la ralentización del sistema, es decir, el programa no alcanzará la misma velocidad de ejecución que si se instalase directamente en el sistema operativo «anfitrión (Host)» o directamente sobre la plataforma de Hardware, sin embargo, a menudo la flexibilidad que ofrecen compensa esta pérdida de eficiencia.

Si la virtualización es por Hardware, la velocidad de ejecución es más que aceptable para la mayoría de los casos, por ejemplo, en el caso de usar KVM/QEMU soporta máquinas virtuales de hasta 255 CPUs y 4 TB de RAM, y el rendimiento de aplicaciones como Oracle, SAP, LAMP, MS Exchange sobre máquinas virtuales puede oscilar entre el 95% y el 135% comparado con su ejecución en servidores físicos. Además, se ha conseguido ejecutar hasta 600 máquinas virtuales en un solo servidor físico.

12.5.1 Ventajas

Además de permitir ejecutar múltiples sistemas operativos, diferentes versiones de un mismo sistema pero con diferente Software que en principio puede ser incompatible entre sí. Para usuarios de Windows, el hecho en sí, de no tener porque lidiar con problemas derivados de virus y antivirus le confiere una gran ventaja desde el punto de vista administrativo y del usuario final. Además, permite una administración centralizada, ya que todas las máquinas virtuales tendrían la misma configuración y paquetes sin importar el Hardware subyacente en las que se ejecute el sistema operativo huésped.

En el caso de instituciones educativas de cualquier nivel académico, es común que en un mismo equipo de cómputo sea necesario ejecutar por un lado diferentes versiones de sistemas operativos -por ejemplo Linux, Windows XP,

Windows 7, etc.- y por otro lado, en un sistema operativo, ejecutar diferentes versiones de un mismo paquete -generalmente no se pueden tener instalados simultáneamente más de una versión-.

Las máquinas virtuales son una verdadera opción para coexistir simultáneamente diferentes versiones de sistemas operativos y en un mismo sistema máquinas virtuales ejecutando las diversas versiones de un mismo programa de cómputo, además se pueden configurar para que al momento de iniciarlas siempre se ejecuten a partir de una configuración e instalación base, de tal forma que al ser lanzadas, el usuario pueda instalar, configurar e inclusive dañar la máquina virtual, pero al reiniciarse la máquina virtual en una nueva sesión, se regresa a la configuración de la versión base, de esta forma no hay posibilidad de infección de virus entre diversos lanzamientos de sesiones de la máquina virtual, la actualización es centralizada y se puede hacer por red, sin intervención del usuario.

Por ello, es una opción viable y común tener en una máquina un sistema huésped como Debian GNU/Linux Estable y dentro de él, un grupo de máquinas virtuales de Windows -Windows XP, Windows 7, etc.-, en los que cada máquina virtual tenga instalado Software agrupado por las características del sistema operativo necesario para ejecutar a todas las aplicaciones seleccionadas -por ejemplo agrupados por la versión de Service Pack-.

Por otro lado, si se desconfigura un sistema operativo virtualizado es sumamente fácil de restaurar si lo comparamos con una máquina real. Si tomamos las precauciones necesarias podemos restaurar el estado que tenía un sistema operativo virtualizado, de forma fácil y rápida. Si hablamos del entorno empresarial, la virtualización de sistemas operativos supone un ahorro económico y de espacio considerable. Ya que mediante el uso de la virtualización evitamos la inversión en multitud de equipos físicos, esto supone un ahorro importante en mantenimiento, en consumo energético, espacio y procesos administrativos.

Por otro lado, mediante la virtualización y el balanceo dinámico podemos incrementar las tasas de prestación de servicios de un servidor del siguiente modo. Si disponemos de un servidor Web podemos asignar recursos adicionales al servidor, como por ejemplo memoria RAM y CPU en los picos de carga para evitar que el servidor se caiga y de este modo incrementar la tasa de eficiencia. Una vez finalizado el pico de carga podemos desviar los recursos aplicados al servidor Web a otra necesidad que tengamos. Por lo tanto, aparte de mejorar la tasa de servicio se pueden optimizar los recursos.

Si estamos usando una máquina virtual en un entorno de producción,

podemos ampliar los recursos de un sistema operativo o servidor de una forma muy sencilla, tan solo tenemos que acceder al Software de virtualización y asignar más recursos. Además, es fácil crear un entorno para realizar pruebas de todo tipo aislado del resto del sistema. Así, las máquinas virtuales y la virtualización permiten usar un solo servicio por servidor virtualizado de forma sencilla, de este modo aunque se caiga uno de los servidores virtualizados los otros seguirán funcionando.

En resumen, la virtualización permite ofrecer un servicio más rápido, sencillo a usuarios (académicos, estudiantes, clientes, etc.) y es un pilar que debe ser considerado en una escuela, universidad o compañía en su proceso de transformación o consolidación, permitiendo escalonar y ser creativos a la hora de atender las necesidades crecientes y cambiantes de los usuarios; y contar con servicios agregados, ágiles y adaptables a los constantes cambios de tecnología de Hardware y Software permitiendo escalar a la hiperconvergencia hacia la nube.

12.5.2 Desventajas

Entre las principales desventajas de virtualizar sistemas propietarios[90] como Windows (véase 11.1) -no así los sistemas libres como Debian GNU/Linux (véase 11.2)- es que se puede violar el sistema de licenciamiento (véase 11.5) del Software instalado en las máquinas virtuales, esto es especialmente importante cuando se usa en más de una máquina, pues la licencia usada para la instalación es violada cuando se tiene más de una copia de la máquina virtual o se ejecutan múltiples instancias de la máquina virtual.

En el caso de Windows XP Home, no se infringe la licencia mientras se cuente con número de licencias igual al máximo número de máquinas virtuales lanzadas simultáneamente. Para otras versiones del sistema operativo Windows como es Windows XP Profesional, la virtualización se maneja con licencias adicionales a la del sistema operativo original y se debe de contar con tantas licencias como el máximo número de máquinas virtuales lanzadas simultáneamente. Además, es necesario contar con el tipo de licencia adecuada para virtualizar a todos y cada uno de los paquetes de cómputo instalados en cada máquina virtual y en las instancias para el número de máquinas

[90] Según la Free Software Foundation (véase [11]), 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. Así, un Software que no es libre, es llamado «Software privativo o propietario».

virtuales lanzadas simultáneamente en uno o más equipos.

Para usar una máquina virtual en condiciones favorables, necesitamos un equipo de cómputo potente. Debemos tener en cuenta que si usamos dos sistemas operativos de forma simultánea estamos empleando hasta el doble de recursos. No obstante, cualquier equipo de cómputo doméstico de gama baja actual dispone de los recursos suficientes para usar una o más máquinas virtuales.

Los sistemas operativos y los programas se ejecutarán con mayor lentitud en las máquinas virtuales. Esto es debido a que las máquinas virtuales no pueden sacar un rendimiento ideal del Hardware que tenemos en nuestro equipo. Cuanto más potente sea nuestro equipo de cómputo menos se notará la pérdida de rendimiento.

Si tenemos un problema -de Hardware o Software- en el equipo de cómputo que aloja el sistema operativo anfitrión puede caerse el servicio en la totalidad de máquinas virtuales. Por lo tanto el equipo de cómputo que hace funcionar la máquina virtual es una parte crítica del proceso de virtualización.

A pesar de los inconvenientes que se citan en este apartado, bajo nuestro punto de vista, la virtualización y las máquinas virtuales proporcionan unas ventajas y una flexibilidad que compensan claramente los inconvenientes que acabamos de citar.

12.5.3 Consideraciones Técnicas y Legales de la Virtualización

Como se mostrará en la siguiente sección, virtualizar sistemas operativos -Linux, Unix, Windows entre otros- no representa ningún problema técnico, pero no es el caso en cuanto a las implicaciones legales de hacer la virtualización que involucra el almacenamiento, distribución y el número de veces que se ejecuta simultáneamente una máquina virtual en uno o múltiples equipos, ya que en general, la máquina virtual está contenida en un sólo archivo que facilita su distribución y almacenamiento, violando de esta forma la licencia de algunos sistemas operativos y/o programas instalados en el mismo.

En el caso de virtualizar cualquier sistema operativo libre como Debian GNU/Linux (véase 11.2), el tipo de licencia que tiene, permite y alienta su uso para cualquier fin que uno desee, por ello no hay ningún problema en virtualizarlo, no así el caso de hacerlo en sistemas operativos propietarios tipo Windows, la licencia (véase 11.1) restringe su uso a un sólo equipo de cómputo y en muchos casos prohíbe expresamente su virtualización. Además hay que

tomar en cuenta el resto del Software instalado en el sistema operativo, ya que estos también tienen sus propias restricciones legales a su uso y número de veces que se puede ejecutar simultáneamente un paquete dado.

Esto es especialmente importante cuando se usa en más de una máquina física, la misma máquina virtual, pues la licencia usada para la instalación es violada cuando se tiene más de una copia de la máquina virtual o se ejecutan múltiples instancias de la máquina virtual, esta violación de licencia es suficiente para ser sujeto a multas o incluso cárcel por dicho ilícito (véase 11.5).

Por otro lado, cada vez que se adquiere una licencia de uso de algún Software que no caduque -la cual implica un alto costo monetario-, esta pueda seguir siendo usada en una máquina virtual con una versión tal vez obsoleta del sistema operativo que la soporte, pero corriendo en un sistema huésped moderno y protegido en Hardware de última generación de forma lícita y con el consiguiente ahorro económico.

12.6 Máquinas Virtuales en la Educación, Ciencias e Ingeniería

Como hemos visto en las secciones anteriores, el uso de las máquinas virtuales es variado, flexible y permite ser usado en diversos ámbitos de la educación, del desarrollo y prueba de programas de cómputo y en general, en Ciencias e Ingeniería. Algunas de las utilidades y beneficios que podemos sacar de una máquina virtual son los siguientes:

- Para aprender a instalar, configurar y usar diversos sistemas operativos, además de poder probar diversas opciones de configuración en ellos. El proceso de instalación de la máquina virtual no requiere crear particiones adicionales en nuestro disco ni alterar la configuración de la máquina anfitriona y podemos trasladar la máquina virtual a uno o más equipos de cómputo que la soporte.

- Para usar un Software que no está disponible en nuestro sistema operativo habitual. Por ejemplo, si somos usuarios de Linux y queremos usar Photoshop, lo podemos hacer a través de una máquina virtual.

- En ocasiones tenemos que usar Software que únicamente se puede ejecutar en sistemas operativos obsoletos -Windows 98 por ejemplo-,

podemos crear una máquina virtual con dicho sistema y usar el Software de forma aislada sin preocuparnos de sus vulnerabilidades.

- Podemos experimentar en el sistema operativo que corre dentro de la máquina virtual haciendo cosas que no nos atreveríamos a realizar con nuestro sistema operativo habitual, como por ejemplo, instalar Software no seguro que consideramos sospechoso, etc.

- En muchos casos se quiere aprender a instalar, administrar y usar equipo al que no tenemos acceso como un equipo multiCore, con tarjeta CUDA instalada o un Cluster de múltiples nodos multiCore. Esto es posible hacer mediante el uso de máquinas virtuales en un equipo de gama media.

- Si se hace el adecuado aislamiento de una máquina virtual en la que se instale alguna versión de Windows, esta puede ser inmune a los virus y no requiere el uso de antivirus.

- En el caso de instituciones educativas de cualquier nivel académico, es común que en un mismo equipo de cómputo sea necesario ejecutar por un lado diferentes versiones de sistemas operativos -Linux, Windows XP, Windows 7, etc.- y por otro lado, en un sistema operativo, ejecutar diferentes versiones de un mismo paquete -generalmente no se puede tener instalada simultáneamente más de una versión- esto se logra con máquinas virtualizadas ad hoc coexistiendo en una misma máquina física.

- Podemos crear/simular una red de equipos de cómputo con tan solo un equipo de cómputo. Esta red de equipos de cómputo virtualizados la podemos usar con fines formativos y de este modo adquirir pericia sobre administración de redes.

- Si eres un desarrollador de Software puedes revisar si el programa que estas desarrollando funciona correctamente en varios sistemas operativos y/o navegadores de Web.

- Podemos usar las máquinas virtuales para hacer SandBox[91] con el fin

[91] Un sistema de aislamiento de procesos o entorno aislado, a menudo usando como medida de seguridad para ejecutar programas con seguridad y de manera separada del sistema anfitrión.

de ejecutar aplicaciones maliciosas o abrir correos sospechosos en un ambiente controlado y seguro.

- Para probar versiones Alfa, Beta y Release Candidate de ciertos programas y sistemas operativos.
- Para montar un servidor Web, un servidor VPN, un servidor de correo o cualquier otro tipo de servidor.
- Para probar multitud de programas en Windows y evitar que se ensucie el registro mediante las instalaciones y desinstalaciones de los programas
- Consolidar servidores, i.e. lo que ahora hacen varias máquinas, se pueden poner en un solo equipo físico dentro de varias máquinas virtuales independientes o interactuando entre ellas según se requiera.
- Mantenimiento y pruebas de aplicaciones sin necesidad de adaptar nuevas versiones del sistema operativo.
- Aumentar la disponibilidad al reducir tiempo de parada y mantenimiento. Ya que la máquina virtual está hecha, se pueden poner a trabajar una o más copias en un equipo o en múltiples máquinas físicas en cuestión de segundos, permitiendo la continuidad de un negocio o servicio y de recuperación ante desastres.
- Reducir costos de administración ya que se reducen y agilizan las políticas de respaldo y recuperación, además de optimizar los recursos disponibles permitiendo escalabilidad al crecer con contención de costos, mejorando la eficiencia energética al usar un menor número de equipos de cómputo.
- Permite incursionar en la estrategia de nube híbrida proactiva creando un conjunto de marcos de decisión en la nube y procesos para evaluar las oportunidades de computación en la nube en función de las necesidades y cargas de trabajo de los usuarios, por ejemplo el uso de supercómputo rentado.
- Establecer las habilidades, herramientas y procesos para un entorno dinámico e híbrido al asociarse los educadores y los especialistas en tecnologías de información para realizar un inventario de habilidades

y competencias, e identificar oportunidades de capacitación y áreas de vulnerabilidad potencial.

12.7 ¿Qué Necesito para Crear y Usar una Máquina Virtual?

Actualmente la virtualización de un sistema operativo se puede implementar por Software o por Hardware, lo único que precisamos para poder usar una máquina virtual es un equipo de cómputo e instalar y configurar el programa manejador de la máquina virtual. Cuanto más potente y actual sea el equipo de cómputo del que dispongamos, mejor experiencia obtendremos trabajando con sistemas operativos virtualizados.

Algunos de los puntos importantes para obtener un rendimiento óptimo del sistema operativo virtualizado son los siguientes:

- Preferentemente disponer de un procesador que disponga de capacidad de virtualización por Hardware (Intel VTx/AMD-V). Casi cualquier equipo de cómputo actual dispone de un procesador apto para virtualizar sistemas operativos por Hardware.
- Disponer de espacio suficiente en el disco duro[92], es preferible disponer de un disco de estado sólido (SSD) por sus velocidades de lectura-escritura.
- Necesitamos disponer de memoria RAM suficiente y adecuada[93]. Cuanta más memoria RAM y cuanto más rápida sea, mejores resultados de virtualización obtendremos.
- Sin duda el hecho de tener una buena tarjeta gráfica también ayudará a disponer de una mejor experiencia de virtualización.

Para empezar, debemos decidir qué manejador de máquinas virtuales usar, si trabajamos en Debian GNU/Linux, podemos usar QEMU/KVM y lo instalamos mediante:

[92] Una máquina virtual con Windows XP ocupa por lo menos 2 GB en disco y una con Windows 7 ocupa por lo menos 4 GB en disco.

[93] La cantidad de memoria RAM ideal dependerá del sistema operativo que queremos virtualizar y del número de sistemas operativos que queramos virtualizar de forma simultánea. Si tan solo queremos virtualizar un sistema operativo con 2 o 3 GB de RAM debería ser suficiente.

```
# apt install qemu-kvm qemu qemu-utils
```

QEMU/KVM soporta emulación de IA-32 (x86) PC, AMD64 PC, MIPS R4000, Sun's SPARC sun4m, Sun's SPARC sun4u, ARM development boards (Integrator/CP y Versatile/PB), SH4 SHIX board, PowerPC (PReP y Power Macintosh), y arquitecturas ETRAX CRIS.

Ejemplo 1 *Si hemos descargado la imagen ISO de algún sistema operativo, por ejemplo la de Knoppix*[94]*, la podemos usar mediante:*

```
$ kvm -m 1024 -cdrom KNOPPIX_ V8.2-2018-05-10-EN.iso
```

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

Ejemplo 2 *Instalación y uso de una máquina virtual para Debian GNU/Linux estable a partir del archivo ISO*[95]*, para ello, primero necesitamos:*

Generar el disco virtual que la contendrá, por ejemplo de 10 GB con el nombre debianStable.img mediante:

```
$ qemu-img create -f qcow2 debianStable.img 10G
```

Después, instalar la imagen de Debian estable en el disco virtual generado en el paso anterior, solicitando a KVM que una vez terminada la instalación no haga el reinicio de la máquina virtual, esto mediante:

```
$ kvm -no-reboot -boot d -cdrom debian-802-i386-netinst.iso \
-hda debianStable.img -m 400
```

Ahora podemos usar la máquina virtual mediante:

```
$ kvm -hda debianStable.img -m 800
```

Ejemplo 3 *Instalación y uso de una máquina virtual para Windows XP, en este caso necesitamos:*

Crear el disco virtual, por ejemplo de 10 GB mediante:

```
$ qemu-img create -f qcow2 WindowsXP.img 10G
```

Hacer la instalación básica a partir del ISO, mediante:

```
$ kvm -no-reboot -boot d -hda WindowsXP.img -m 400 \
-localtime -cdrom es_ winxp_ pro_ with_ sp2.iso
```

Y concluir la instalación mediante:

```
$ kvm -no-reboot -boot c -hda WindowsXP.img -m 400 \
```

[94] Knoppix es una versión Live ampliamente conocida y completa de GNU/LINUX, se puede descargar de: https://www.knopper.net/knoppix/

[95] Diversas imágenes ISO del proyecto Linux Debian se pueden descargar de: https://www.debian.org/CD/

-localtime -cdrom es_winxp_pro_with_sp2.iso
Ahora podemos usar la máquina virtual mediante:
$ kvm -boot c -hda WindowsXP.img -m 400 -localtime

12.8 ¿Cómo Funciona una Máquina Virtual?

Explicar el funcionamiento a detalle de una máquina virtual es engorroso y está fuera del alcance del propósito de este texto. No obstante a grandes rasgos podemos decir que una máquina virtual es un Software que mediante una capa de virtualización[96] se comunica con el Hardware que tenemos disponible en nuestro equipo de cómputo consiguiendo de este modo emular la totalidad de componentes de un equipo de cómputo real. De este modo la máquina virtual será capaz de emular un disco duro, una memoria RAM, una tarjeta de red, un procesador, etc.

Una vez que sabemos esto, cuando abrimos una máquina virtual, como por ejemplo Virtualbox (véase [88]), nos encontramos con un entorno gráfico (en el caso de usar QEMU/KVM se usa la línea de comandos para crear y usar las máquinas virtuales, pero también se cuentan con entornos gráficos que usan la línea de comandos internamente) que nos permitirá configurar y asignar recursos a cada uno de los componentes físicos que emula la máquina virtual. En prácticamente la totalidad de máquinas virtuales debemos definir detalles del siguiente tipo:

- Tipo de procesador a usar
- Espacio que queramos asignar al disco duro.
- Memoria RAM que queremos asignar a la máquina virtual.
- La memoria de nuestra tarjeta gráfica.
- La configuración de red.
- etc.

[96] La capa de virtualización es un sistema de archivos propietario y una capa de abstracción de servicio del Kernel que garantiza el aislamiento y seguridad de los recursos entre distintos contenedores. La capa de virtualización hace que cada uno de los contenedores aparezca como servidor autónomo. Finalmente, el contenedor aloja la aplicación o carga de trabajo.

Una vez configurados estos parámetros habremos creado una máquina virtual para instalar un sistema operativo, de este modo tan solo tendremos que abrir la máquina virtual que se acaba de crear e instalar el sistema operativo tal y como si se tratará de un equipo de cómputo real.

12.9 Aplicaciones y Paquetes Disponibles

A continuación mencionaremos algunas de las aplicaciones más conocidas y universalmente disponibles y/o usadas en los Sistemas Operativos GNU Linux/BSD, tanto en el ámbito personal, es decir, distribuciones usadas para fines particulares (hogar), como en el ámbito profesional, es decir, en el área de servidores de organizaciones y empresas.

Es importante destacar que, en este listado no se incluirán aquellas tecnologías de virtualización que vienen como una solución integrada, todo en uno o llave en mano, tales como *Promox*.

VirtualBox Software desarrollado por Oracle multiplataforma, capaz de virtualizar prácticamente la totalidad de sistemas operativos con arquitectura x86/amd64. La base de este Software dispone de una licencia GPL2, mientras que el Pack de extensiones que añaden funcionalidades están bajo licencia privativa. Virtualbox es gratuito para un uso no comercial.

Es un Hipervisor de Tipo 2 multiplataforma, es decir, solo debe y puede ser ejecutado (instalado) en cualquier Host (Ordenador) con alguna de las versiones vigentes o antiguas de Sistemas Operativos Windows, Linux, Macintosh, Solaris, OpenSolaris, OS/2 y OpenBSD.

Posee un continuo y progresivo ciclo de desarrollo con lanzamientos frecuentes, que la convierten en una excelente alternativa a otras soluciones parecidas, pero con una muy apreciable cantidad de características y funciones, sistemas operativos invitados compatibles y plataformas en las que se puede ejecutar.

En la mayoría de las distribuciones GNU Linux/BSD se encuentra dicha aplicación incluida en los repositorios, por lo que, con la siguiente orden de comando suele instalarse en todas ellas:

```
# apt install virtualbox
```

Vale destacar para VirtualBox que, al usar esta aplicación siempre es ideal la instalación de las «Guest Additions» y el «Extension Pack». Por ende,

para esto y otras formas de instalación lo ideal es visitar el enlace oficial de VirtualBox.

Vmware Workstation Player Software privativo multiplataforma desarrollado por EMC Corporation y que es utilizado ampliamente en el entorno profesional en las áreas del Cloud Computing entre muchas otras. Al igual que Virtualbox, esta máquina virtual nos permite virtualizar infinidad de sistemas operativos. Vmware dispone de muchas soluciones de virtualización y prácticamente todas son de pago, no obstante Vmware Workstation Player es totalmente gratuita para un uso no comercial.

Parallels aunque se trata de una máquina virtual multiplataforma, acostumbra a ser usado por los usuarios del sistema operativo OS X de Apple que desean virtualizar el sistema operativo Windows. Esta máquina virtual es de pago y únicamente puede virtualizar los sistemas operativos Windows y Mac OS. Quien quiera probar Parallels lo puede hacer descargando la versión de prueba.

Windows Virtual PC Software gratuito y privativo propiedad de Microsoft que se puede usar tanto en Windows como en Mac OS. Virtual PC está destinado únicamente a virtualizar sistemas operativos Windows.

Virtualización: GNOME Boxes es una aplicación nativa del entorno de Escritorio GNOME, que se utiliza para acceder a sistemas remotos o virtuales. Boxes o Cajas, utiliza las tecnologías de virtualización de *QEMU*, *KVM* y *Libvirt*.

Además, requiere que la CPU sea compatible con algún tipo de virtualización asistida por Hardware (Intel VT-x o AMD-V, por ejemplo); por lo tanto, GNOME Boxes no funciona en las CPUs con procesador Intel Pentium/Celeron, ya que, carecen de esta característica.

En la mayoría de las distribuciones GNU Linux/BSD se encuentra dicha aplicación incluida en los repositorios, por lo que, con la siguiente orden de comando suele instalarse en todas ellas:

```
# apt install gnome-boxes
```

Vale destacar para GNOME Boxes que, es una herramienta muy sencilla dirigida a usuarios novatos, permite descargar desde la aplicación diversas

imágenes y no incorpora demasiadas opciones de configuración que suelen ser muy conocidas y usadas en otras, tales como VirtualBox.

Virt-Manager es una interfaz de usuario de escritorio para la administración de un Gestor de Máquinas Virtuales a través de *Libvirt*. Está dirigida principalmente a las máquinas virtuales gestionadas por *KVM*, pero también maneja las gestionadas por *Xen* y *LXC*.

Virt-Manager presenta una vista resumida de los dominios en ejecución, su rendimiento en vivo y estadísticas de utilización de recursos. Los asistentes permiten la creación de nuevos dominios, y la configuración y ajuste de la asignación de recursos de un dominio y el Hardware virtual. Un visor de cliente *VNC* y *SPICE* integrado presenta una consola gráfica completa para el dominio invitado.

En la mayoría de las distribuciones GNU Linux/BSD se encuentra dicha aplicación incluida en los repositorios, por lo que, con la siguiente orden de comando suele instalarse en todas ellas:

```
# apt install virt-manager
```

Vale destacar para Virt-Manager que, es también es una herramienta sencilla, aunque mucho más completa que GNOME Boxes, por lo que se puede considerar para usuarios medios o avanzados de primer nivel, dado que, fácilmente es capaz de permitir la gestión de todo el ciclo de vida de las máquinas virtuales existentes.

QEMU / KVM es un emulador y virtualizador de máquinas genérico y de código abierto, capaz de ejecutar sistemas operativos y programas hechos para una máquina en una máquina diferente con muy buen rendimiento, y capaz de lograr un rendimiento casi nativo ejecutando el código del huésped directamente en la CPU del Host.

KVM es una solución de virtualización completa para Linux en Hardware x86 que contiene extensiones de virtualización (Intel VT o AMD-V) que consiste en un módulo de Kernel cargable, que proporciona la infraestructura de virtualización del núcleo y un módulo específico del procesador. Y actualmente funciona inmerso dentro de QEMU.

En la mayoría de las distribuciones GNU Linux/BSD se encuentra dicha aplicación incluida en los repositorios, por lo que, con la siguiente orden de comando suele instalarse en todas ellas:

```
# apt install qemu-kvm
```

además, en caso necesario podemos hacer uso de una interfaz gráfica para trabajar con QEMU/KVM, tenemos varias opciones, algunas de ellas son:

```
# apt install qemu-system-gui
# apt install aqemu
# apt install qemuctl
# apt install virt-manager
# apt install gnome-boxes
```

Vale destacar que QEMU/KVM es también es una herramienta muy completa, ya que no solo emula (x86, x86-AMD64, MIPS, Arm, PowerPC, SPARC, etc.) sino que virtualiza, a diferencias de otros iguales de avanzadas como WMWare, que solo permite virtualizar. Para conocer las opciones de emulación usamos:

```
$ apt search qemu-system-
```

Cockpit es una interfase Web Open Source que permite manejar máquinas virtuales de KVM que provee acceso a sistemas Linux permitiendo la administración, manejo y monitoreo a través de una interfaz gráfica intuitiva, para usarlo hay que instalar primero KVM y luego hacer:

```
# apt install cockpit cockpit-machines
```

Librerías y Paquetes relacionados Estos últimos 3 paquetes mencionados suelen instalar otros conexos (relacionados) como dependencias, por lo que, en caso de ser necesario, pueden instalar los mismos, junto a sus dependencias y otros paquetes útiles necesarios, como:

> gnome-boxes virt-manager virt-goodies virt-sandbox virt-top virt-viewer virtinst libvirt-clients libvirt-daemon libvirt-daemon-system qemu qemu-kvm qemu-utils qemu-system qemu-system-gui qemu-block-extra freerdp2-x11 bridge-utils ovirt-guest-agent systemd-container

Otros en caso de querer instalar otras tecnologías de virtualización disponibles sobre Linux/BSD se puede optar por:

Xen instalándolo con la orden de comando siguiente:

```
# apt install xen-system-amd64 xen-utils-4.11 xen-tools
```

LXC instalándolo con la orden de comando siguiente:

```
# apt install lxc
```

Docker instalándolo con la orden de comando siguiente:

```
# apt install docker-ce docker-ce-cli containerd.io
```

Diferencias entre KVM y QEMU Cuando comenzamos en el mundo de la virtualización, la opción más recurrente para comenzar es KVM. Después nos damos cuenta de que se comienza a usar también otro componente muy a menudo, QEMU y siempre hay muchas preguntas en torno a cómo funciona KVM y QEMU o cual es la diferencia entre ellos.

Sobre KVM es un Software de código abierto y significa Kernel Virtual Machine (Máquina Virtual basada en el Kernel) es una solución de virtua-lización para Linux en hardware x86 que contiene extensiones de virtualización (Intel VT o AMD-V). KVM forma parte del Kernel de Linux desde la versión 2.6.20. Específicamente, con KVM podemos convertir a Linux en un hipervisor para que nuestro Host ejecute entornos virtuales, es decir, máquinas virtuales. Cada máquina virtual se implementa como un proceso regular de Linux

KVM ha jugado un papel clave en el entorno de virtualización de código abierto basado en Linux. De hecho, KVM es el único hipervisor para todos los productos de virtualización de Red Hat, tanto para RHOSP - Red Hat Openstack Platform, como para Red Hat Virtualization o abreviado, RHV.

KVM en general es 2 cosas, un módulo del Kernel pero también KVM es es un Fork del ejecutable de QEMU (más adelante hablaremos de eso). Entonces como mencionamos, KVM es un módulo Kernel que permite el uso de tecnologías de extensiones de virtualización Intel o AMD. En pocas palabras, estas extensiones permiten que múltiples sistemas operativos compartan una CPU física sin interferir entre ellos. Por otro lado, no resuelven compartir todos los dispositivos de Hardware, para esto KVM requiere una lógica extra y aquí es dónde comenzamos a hablar de QEMU.

Sobre QEMU es un emulador de procesadores basado en la traducción dinámica de binarios, es decir, realiza la conversión del código binario de la arquitectura fuente o Host, en código entendible por la arquitectura huésped o la máquina virtual.

El resultado de usar QEMU es poder ejecutar el código original como si se estuviera ejecutando en la máquina emulada. Por ejemplo, se podría ejecutar código escrito para el procesador ARM en su máquina basada en Intel.

QEMU es capaz de emular varias plataformas de Hardware diferentes, incluida la x86, plataformas PowerPC, sistemas basados en ARM y también sistemas SUN SPARC. Además del Hardware básico de estos sistemas, QEMU también proporciona emulación de varios módulos adicionales, como tarjetas gráficas, tarjetas de sonido, dispositivos de red, dispositivos de almacenamiento y controladores, dispositivos serie/paralelo/USB y dispositivos de memoria. Esto significa que en muchos casos las computadoras pueden ser completa y totalmente emuladas y utilizadas para ejecutar su Software original.

¿Cómo trabajan juntos? en el Hardware real, el sistema operativo traduce las instrucciones de los programas para que sean ejecutadas por el CPU físico. En una máquina virtual es lo mismo, pero la diferencia es que el CPU está virtualizado por el hipervisor y el hipervisor tiene que traducir las instrucciones del CPU virtual y convertirlo en instrucciones para el CPU físico. Esta traducción tiene una gran sobrecarga de rendimiento.

Para minimizar esta sobrecarga, los procesadores admiten extensiones de virtualización. Intel soporta una tecnología llamada VT-X y el equivalente AMD es AMD-V. Con el uso de estos, una rebanada de CPU físico se puede asignar directamente al CPU virtual. Así que las instrucciones de la CPU virtual se pueden ejecutar directamente en la rebanada del CPU físico. Evitando así la traducción que tendría que hacer el hipervisor.

KVM es el módulo del Kernel de Linux que permite esta asignación de CPU físico para CPU virtual. Esta asignación proporciona la aceleración de Hardware para la máquina virtual y aumenta su rendimiento. De hecho QEMU utiliza esta aceleración cuando el tipo de virtualización es elegido como KVM.

Los desarrolladores de KVM aprovecharon la arquitectura QEMU y básicamente crearon un nuevo modelo de CPU en QEMU. Este nuevo tipo de modelo tiene una lógica específica de KVM. Así las llamadas al sistema que

se harían de forma nativa pasan por del módulo KVM para que la ejecución se ejecute de forma nativa en la CPU, mientras que QEMU se utiliza para proporcionar el resto funcionalidad (emular los dispositivos).

Al trabajar juntos, KVM accede directamente al CPU físico y a la memoria, a su vez QEMU emula los recursos de Hardware, como el disco duro, video, USB, etc. Hoy, cuando las personas se refieren al hipervisor KVM, en realidad se refieren a la combinación QEMU/KVM.

KVM necesita QEMU (emulador) para la funcionalidad completa como hipervisor. QEMU es autosuficiente y KVM es realmente un módulo del Kernel de Linux para la explotación de extensiones VT que actúa como controlador de las capacidades de CPU físicas.

Así puedes notar entonces que QEMU necesita a KVM para aumentar su rendimiento... Por otro, lado KVM por sí solo no puede proporcionar la solución de virtualización completa, necesita de QEMU.

12.10 Acceso a Datos Desde una Máquina Virtual

Para acceder a los datos almacenados en máquinas virtuales, disponemos de las siguientes opciones:

a) Mediante el uso de algún navegador Web, se puede acceder a su cuenta de correo electrónico y al almacenamiento en la nube como *Google Drive, Dropbox, HubiC, pCloud, MediaFire, FlipDrive, Mega*, entre otros.

b) En el sistema operativo Linux, se puede acceder a cualquier servidor de internet mediante los protocolos *SSH*, *SAMBA* o montar un sistema de archivos mediante *NFS* o *SSHFS*, entre otros.

c) En cualquier sistema operativo podemos usar algún navegador gráfico de *FTP, FTPS o SFTP* como *FileZilla, WinSCP, PSCP, PSFTP, FireFTP, CoreFTP*, entre muchos otros, para transportar archivos y carpetas.

d) En las máquinas virtuales de Windows usamos el protocolo *SAMBA*, para tener acceso a este, hay que conectarse a una unidad de red dentro del explorador de archivos de Windows.

e) En Linux, por ejemplo con *PCManFM, Dolphin, Nautilus, Thunar, Konqueror*, entre otros, podemos acceder a una máquina que tenga un servidor de la siguiente forma:

1) Para acceder a un servidor *SAMBA*, escribir la ruta de archivos en el manejador de archivos:

$ smb://estud@192.168.13.230/estud/

2) Para acceder a un servidor *SSH*, escribir la ruta de archivos en el manejador de archivos:

$ sftp://usuario@192.168.13.230/home/usuario/

En línea de comandos, podemos:

3) Montar con *SSHFS* un directorio de otra máquina con servidor *SSH*:

$ sshfs usuario@192.168.13.230:/home/usuario/ /home/algun/lugar

4) Montar con mount un directorio de otra máquina con servidor *NFS*:

mount 10.0.2.2:/directorio ./punto_montaje

5) Usar *SCP* y *SFTP* de *SSH* para transferir archivos, para copiar un archivo, usamos:

$ scp archivo.dat usuario@192.168.13.230:~/Datos/

para copiar un subdirectorio, usamos:

$ scp -r Directorio usuario@192.168.13.230:.

para copiar un archivo de una máquina remota a nuestra máquina, usamos:

$ scp usuario@192.168.13.230:/home/usuario/archivo .

6) Montar con CURLFTPFS un directorio de otra máquina con servidor *FTP*:

curlftpfs -o allow_other usuario:password@192.168.13.230:puerto /home/algun/lugar

12.11 Desde la Nube

Existen diferentes servicios Web[97] que permiten instalar, configurar y usar cientos de sistemas operativos Linux y Unix -máquinas virtuales usando servicios Web en Debian GNU/Linux y QEMU- desde el navegador, esto en

[97] Cuando se trabaja desde la Web es recomendable usar el modo Privado o Incógnito para no guardar el historial de navegación, información introducida en los formularios y borrar al cerrar el navegador los datos de los sitios visitados. Pero recuerda que los sitios Web que visitamos sí guardan información de nuestra visita, nuestro proveedor de internet también guarda constancia de nuestra visita y si descargamos algo, esto no se borra al igual que el historial de descargas, además de las marcas de páginas o favoritos se conservarán al cerrar el navegador.

aras de que los usuarios que cuenten con algún sistema de acceso a red y un navegador puedan usar, configurar e instalar algún sistema operativo y su respectiva paquetería sin hacer instalación alguna en su equipo de cómputo, tableta o teléfono celular[98].

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

Algunas versiones listas para usar son:

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

Usar Linux en Formato Live Linux es uno de los sistemas operativos pioneros en ejecutar de forma autónoma o sin instalar en la computadora, existen diferentes distribuciones Live -descargables para formato CD, DVD, USB[99]- de sistemas operativos y múltiples aplicaciones almacenados en un medio extraíble, que pueden ejecutarse directamente en una computadora,

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

[99] Para generar un dispositivo USB con la imagen contenida en un archivo ISO podemos usar el Software ETCHER, descargable para Linux, Windows y Mac OS desde https://etcher.io/.

estos se descargan de la Web generalmente en formato ISO[100], una de las listas más completas de versiones Live está en:

https://livecdlist.com

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

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

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

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

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

y usar mediante:

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

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

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

https://www.osboxes.org
https://www.virtualbox.org

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

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

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

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

13 Bibliografía

Este texto es una recopilación de múltiples fuentes, nuestra aportación —si es que podemos llamarla así— es plasmarlo en este documento, en el que tratamos de dar coherencia a nuestra visión de los temas desarrollados.

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

Herramientas
http://132.248.181.216/Herramientas/

Referencias

[1] Carrera de Actuaría, Facultad de Ciencias, UNAM, http://www.fciencias.unam.mx/licenciatura/resumen/101 5

[2] Aulas y Talleres en el Tlahuizcalpan de las Carreras de Matemáticas de la Facultad de Ciencias, UNAM, http://www.matematicas.unam.mx/tlahuiz/ 12, 15, 274

[3] http://www.gnu.org/philosophy/free-sw.es.html 7, 339

[4] http://es.wikipedia.org/wiki/Software_libre 7, 339

[5] http://www.hispalinux.es/SoftwareLibre 7, 339

[6] http://es.wikipedia.org/wiki/Software_propietario 337

[7] Diferentes Tipos de Licencias para el Software, http://www.gnu.org/licenses/license-list.html 7, 82, 249, 250, 339, 349

[8] Proyectos de Software Sourceforge, http://sourceforge.net/ 8, 9, 10

[9] Google Code, http://code.google.com 8, 9, 10

[10] Software proyecto GNU, http://www.gnu.org/Software/Software.es.html 8

[11] FSF, Free Software Fundation, http://www.fsf.org/ 7, 8, 339, 349, 390

[12] GNU Operating System, http://www.gnu.org/ 7, 339, 349

[13] GCC, the GNU Compiler Collection, http://gcc.gnu.org/ 8

[14] The Linux Kernel Archives, http://www.kernel.org/ 8, 10

[15] Debian el Sistema Operativo Universal, http://www.debian.org 8, 10

[16] Android, http://www.android.com/ 8, 66

[17] Microsoft Windows, http://windows.microsoft.com 38

[18] Microsoft Office, http://office.microsoft.com/ 11, 282, 283, 285, 287, 289

[19] OPEN OFFICE, Apache OpenOffice, http://www.openoffice.org 282, 283, 290

[20] LibreOffice the Document Foundation, http://www.libreoffice.org 282, 283, 290

[21] Google Docs, http://docs.google.com/ 282, 283

[22] CALLIGRA The Integrated Work Applications Suit, http://www.calligra.org/ 282, 283, 290

[23] OnlyOffice, https://www.onlyoffice.com/ 282

[24] WPS Office, https://www.wps.com/ 282

[25] Office Online, https://www.office.com/ 282

[26] Collabora, https://www.collaboraoffice.com/ 282

[27] Lotus Sympony, http://www-03.ibm.com/Software/lotus/symphony/home.nsf/home 282

[28] LaTeX, A Document Preparation System, http://www.latex-project.org/ 121, 275, 287

[29] MATLAB, MathWorks, http://www.mathworks.com/products/matlab/ 11, 249, 251, 274

[30] SCILAB, Scilab Open Source for Numerical Computation, http://www.scilab.org/ 250

[31] FreeMat, FreeMat Opén Source for rapid engineering and scientific prototyping and data processing, http://freemat.sourceforget.net/ 250, 274

[32] OCTAVE, GNU Octave, http://www.gnu.org/Software/octave/ 249, 251, 274

[33] SCIPY Open Source Library of Scientific Tools, http://www.scipy.org/ 250, 274

[34] Python Programming Language, http://www.python.org/ 250

[35] SAS, Business Analytics and Business Intelligence Software, http://www.sas.com/ 11, 274

[36] IBM SPSS Software, http://www-01.ibm.com/Software/analytics/spss/ 11, 274

[37] GNU PSPP, http://www.gnu.org/Software/pspp/ 274

[38] PSPPIRE Data Editor for PSPP, http://www.softpedia.com/get/Office-tools/Other-Office-Tools/PSPP.shtml

[39] The R Proyect for Statistical Computing, http://www.r-project.or/ 121, 251, 274, 275

[40] QEMU, http://wiki.qemu.org/Main_Page 10, 381

[41] KVM, http://www.linux-kvm.org/page/Main_Page 10, 381

[42] máquinas Virtuales, http://es.wikipedia.org/wiki/M%C3%A1quina_virtual 10, 381

[43] Algunos usos de máquinas Virtuales, http://www.configurarequipos.com/doc747.html 10, 381

[44] LINPACK, http://www.netlib.org/linpack/ 249

[45] EISPACK, http://www.netlib.org/eispack/ 249

[46] Scicos Block Diagram Modeler/Simulator, http://www.scicos.org/ 250

[47] Simulink Simulation and Model-Based Design, http://www.mathworks.com/products/simulink/ 249, 250

[48] KOctave, http://sourceforge.net/projects/koctave/

[49] QTOctave, http://www.ohloh.net/p/qtoctave

[50] IDLE is the Python IDE, http://docs.python.org/2/library/idle.html

[51] IPython Interactive Computing, http://ipython.org/

[52] Appcelerator PyDEV, http://pydev.org/

[53] Eclipse, http://www.eclipse.org/

[54] The Eric Python IDE, http://eric-ide.python-projects.org/

[55] MATLAB to Scilab conversion tips, http://help.scilab.org/docs/5.4.0/en_-US/m2sci_doc.html 274

[56] EViews, http://www.eviews.com/home.html 274

[57] Gretel, http://gretl.sourceforge.net/ 274

[58] Stata, http://www.stata.com/

[59] Statgraphics, http://statgraphics.softonic.com/ 274

[60] Statistica, http://www.statsoft.com/support/download/statistica-Software-updates/ 274

[61] Systat, http://www.systat.com/ 274

[62] Vensim, http://vensim.com/vensim-Software/ 274

[63] Maple, http://www.maplesoft.com/ 251, 274

[64] Mathematica, http://www.wolfram.com/mathematica/ 251, 274

[65] Maxima, http://maxima.sourceforge.net 251, 274

[66] RWeka, http://cran.r-project.org/Web/packages/RWeka/index.html 122, 275

[67] Tinn-R Edit code and run it in R, http://www.sciviews.org/Tinn-R/ 122, 275

[68] RStudio Software, Education, and Services for the R community, http://www.rstudio.com/ 122, 275

[69] SAS grows analytics market share, http://www.bizjournals.com/triangle-/news/2011/06/13/sas-grows-analytics-market-share.html

[70] Mathtype, http://www.dessci.com/en/products/mathtype/ 287, 289

[71] Scientific WorkPlace, http://www.mackichan.com/ 289

[72] Gummi LaTeX Editor, http://dev.midnightcoding.org/projects/gummi 289

[73] Kile LaTeX Editor, http://kile.sourceforge.net/ 289

[74] Led LaTex Editor, http://www.latexeditor.org/ 289

[75] Lyx LaTeX Editor, http://www.lyx.org/ 289

[76] Texmaker LaTeX Editor, http://www.xm1math.net/texmaker/ 289

[77] TeXnicCenter LaTeX Editor, http://www.texniccenter.org/ 290

[78] TextPad LaTeX Editor, http://www.textpad.com/ 290

[79] TeXstudio LaTeX Editor, http://texstudio.sourceforge.net/ 290

[80] WinEdt LaTeX Editor, http://www.winedt.com/ 290

[81] LaTeX Beamer Class, https://bitbucket.org/rivanvx/beamer/wiki/Home 287

[82] Microsoft SQL Server, http://www.microsoft.com/en-us/sqlserver/default.aspx 285

[83] PostgreSQL, http://www.postgresql.org/ 285

[84] MySQL Oracle, http://www.mysql.com/ 285

[85] MongoDB, http://www.mongodb.org/ 285

[86] Knoppix is a Live CD/ DVD based on Debian GNU/Linux, http://knoppix.net

[87] Repositorio de LaTeX en la Facultad de Ciencias, UNAM, http://tezcatl.fciencias.unam.mx/tex-archive/ 290

[88] Oracle MV VirtualBox, https://www.virtualbox.org 397

[89] VMware, https://www.vmware.com

[90] Virtual PC, https://www.microsoft.com/es-mx/download/details.aspx?id=3702

[91] Hyper-V, https://msdn.microsoft.com/es-es/library/mt16937(v=ws.11).aspx

[92] Parallels, https://www.parallels.com

[93] El economista, http://eleconomista.com.mx/tecnociencia/2013/01/22/clausuraran-negocios-mexico-uso-ilegal-Software 323

[94] PCworld, http://www.pcworld.com.mx/UNAM-y-BSA-promueven-el-uso-de-software-legal/

324

Declaramos 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, la migraña, las horas de frío y calor, la tristeza, la desesperanza, el cansancio, el presente, el pasado y nuestro futuro, el que dirán, la vergüenza, nuestras propias incapacidades y limitaciones, nuestras aversiones, nuestros temores, nuestras dudas y en fin, todo aquello que pudiera ser tomado por nosotros, 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
Karla Ivonne González Rosas