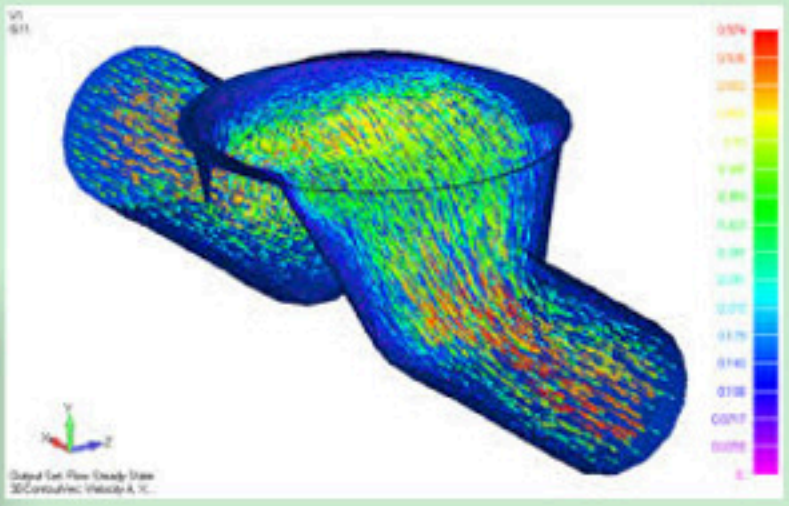
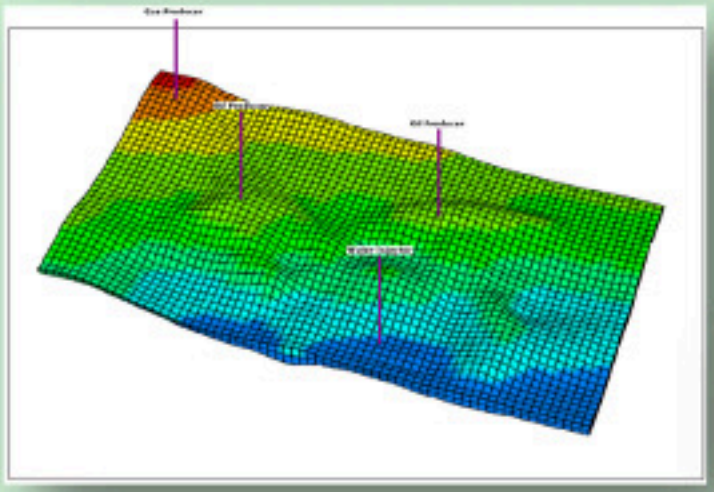
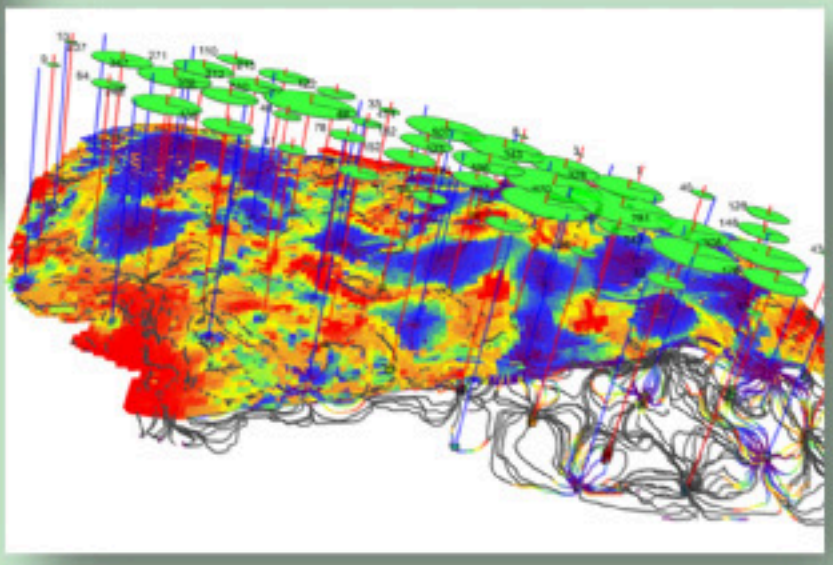
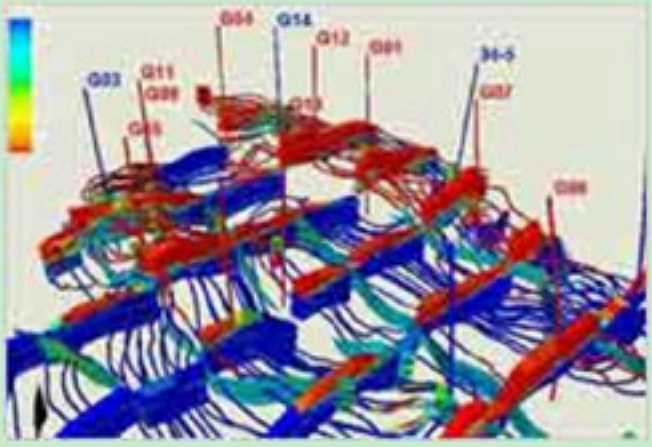
# INTRODUCCIÓN AL CÓMPUTO EN PARALELO










Antonio Carrillo Ledesma

# Introducción al Cómputo en Paralelo

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

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

2025, Versión $1.0\alpha$[1]

[1]El presente trabajo está licenciado bajo un esquema Creative Commons Atribución CompartirIgual (CC-BY-SA) 4.0 Internacional. Los textos que componen el presente trabajo se publican bajo formas de licenciamiento que permiten la copia, la redistribución y la realización de obras derivadas siempre y cuando éstas se distribuyan bajo las mismas licencias libres y se cite la fuente. ¡Copia este libro! ... Compartir no es delito.

# Índice

**1 Introducción** **5**
1.1 Computadoras Actuales . . . . . 8
1.2 ¿Que es Computación Paralela? . . . . . 51
1.3 Agradecimientos . . . . . 57

**2 El Cómputo en Paralelo** **58**
2.1 ¿Que es Computación Paralela? . . . . . 62
2.2 Clasificación Clásica de Flynn . . . . . 68
2.3 Categorías de Computadoras Paralelas . . . . . 71
2.3.1 Equipo Paralelo de Memoria Compartida . . . . . 71
2.3.2 Equipo Paralelo de Memoria Distribuida . . . . . 75
2.3.3 Equipo Paralelo de Memoria Compartida-Distribuida . 76
2.3.4 Cómputo Paralelo en Multihilos . . . . . 82
2.3.5 Cómputo Paralelo en CUDA . . . . . 83
2.4 Métricas de Desempeño . . . . . 88
2.5 Programación de Cómputo de Alto Rendimiento . . . . . 93
2.5.1 Programando con OpenMP para Memoria Compartida 95
2.5.2 Programando con MPI para Memoria Distribuida . . . 98
2.5.3 Esquema de Paralelización Principal-Subordinados . . 103
2.5.4 Opciones de Paralelización Híbridas . . . . . 105
2.6 Programando Desde la Nube . . . . . 107

**3 Entornos de Desarrollo y Herramientas de Programación** **109**
3.1 Java . . . . . 112
3.2 Python . . . . . 121
3.3 Julia . . . . . 138
3.4 C y C++ . . . . . 141
3.5 Fortran . . . . . 148
3.6 R . . . . . 154
3.7 Herramientas de Programación . . . . . 155
3.7.1 ¿Qué es eso de ASCII, ISO-8859-1 y UTF-8? . . . . . 161
3.7.2 Uso de Espacios o Tabuladores en Fuentes . . . . . 165
3.7.3 Comparar Contenido de Fuentes . . . . . 167
3.7.4 Astyle . . . . . 168
3.7.5 Compilación y la Optimización del Ejecutable . . . . . 169
3.7.6 Análisis de Rendimiento y Depuración . . . . . 174

3.7.7 Mejora del Rendimiento en Python . . . 179
3.7.8 Git . . . 184
3.7.9 GitLab vs GitHub . . . 203
3.7.10 Otras opciones . . . 208
3.8 Programando Desde la Nube . . . 210

**4 Programación y Lenguajes de Programación 214**
4.1 Paradigmas de Programación . . . 217
4.2 Lenguaje de Programación . . . 220
4.3 Conceptos Transversales . . . 225
4.4 Algo de Programación . . . 232
4.5 Introducción a los Paradigmas de Programación . . . 245
4.6 Errores de Redondeo y de Aritmética . . . 254
4.7 Documentación del Código Fuente . . . 265
4.7.1 Documentar en C, C++ y Java . . . 268
4.7.2 Documentar en Python . . . 275

**5 Aritmética de Punto Flotante 280**
5.1 Aritmética de Punto Flotante IEEE . . . 285
5.2 Trabajando con Punto Flotante . . . 303
5.3 Aritmética de Baja Precisión . . . 332

**6 Apéndice A: Sistemas Operativos 336**
6.1 Windows . . . 353
6.2 UNIX y BSD . . . 360
6.3 Apple y sus macOS e iOS . . . 362
6.4 GNU/Linux . . . 367
6.5 Android . . . 381
6.6 Chromebook y Chrome OS . . . 384
6.7 Otros Sistemas Operativos . . . 387

**7 Apéndice B: El Sistema Operativo GNU/Linux 393**
7.1 Sistema de Archivos y Estructura de Directorios . . . 400
7.2 Interfaz de Usuario . . . 414
7.2.1 Interfaz Gráfica de Usuario . . . 415
7.2.2 Línea de Comandos y Órdenes . . . 420
7.3 Trabajando en Línea de Comandos . . . 435
7.4 Desde la Nube . . . 474

**8 Apéndice C: Herramientas en Línea de Comandos 479**
8.1 Prompt de la Línea de Comandos . . . 479
8.2 Historia de Comandos . . . 485
8.3 Grabar y Reproducir Contenido de la Terminal . . . 488
8.4 Alias a Comandos . . . 489
8.5 Ayuda de Comandos y Tipo de Archivos . . . 492
8.6 Redireccionando la Entrada y Salida Estándar . . . 496
8.7 Metacarácter o Shell Globbing . . . 504
8.8 Permisos de Archivos y Directorios . . . 509
8.9 Procesos en Primer y Segundo Plano . . . 520
8.10 GNU Parallel . . . 524

**9 Trabajando en Línea de Comandos 526**
9.1 Buscar Archivos . . . 528
9.2 Empaquetadores, Compresores y Descompresores . . . 541
9.3 Copiar Archivos Entre Equipos . . . 570
9.4 Respaldo y Restauración . . . 576
9.5 Incrementando la Seguridad a Nivel Usuario . . . 585
9.6 AntiVurus . . . 601
9.7 Programando en Bash . . . 603
9.8 Algunos Ejemplos en Bash . . . 624

**10 Apéndice D: Software Libre y Propietario 647**
10.1 Software Propietario . . . 650
10.2 Software Libre . . . 652
10.3 Seguridad del Software . . . 659
10.4 Tipos de Licencias . . . 662
10.4.1 Licencias Creative Commons . . . 668
10.4.2 Nuevas Licencias para Responder a Nuevas Necesidades 670
10.5 Implicaciones Económico-Políticas del Software Libre . . . 673
10.5.1 Software Libre y la Piratería . . . 673
10.5.2 ¿Cuánto Cuesta el Software Libre? . . . 674
10.5.3 La Nube y el Código Abierto . . . 677
10.5.4 El Código Abierto como Base de la Competitividad . . . 680
10.5.5 Software Libre en Empresas y Corporaciones . . . 681
10.6 Código Abierto y las Organizaciones Internacionales . . . 689
10.6.1 Las Naciones Unidas y el Código Abierto . . . 690

10.6.2 La Comisión Europea se Compromete a Liberar Todo el Software que Pueda Beneficiar a la Sociedad . . . . . 691

**11 Bibliografía** **694**

# 1 Introducción

La computación paralela es el uso de múltiples recursos computacionales para resolver un problema. Se distingue de la computación secuencial en que varias operaciones pueden ocurrir simultáneamente. Los sistemas de cómputo con procesamiento en paralelo surgen de la necesidad de resolver problemas complejos en un tiempo razonable, utilizando las ventajas de memoria, velocidad de los procesadores, formas de interconexión de estos y distribución de la tarea, a los que en su conjunto denominamos arquitectura en paralelo. Entenderemos por una arquitectura en paralelo a un conjunto de procesadores interconectados capaces de cooperar en la solución de un problema.

Así, para resolver un problema en particular, se usa una arquitectura o combinación de múltiples arquitecturas (topologías), ya que cada una ofrece ventajas y desventajas que tienen que ser sopesadas antes de implementar la solución del problema en una arquitectura en particular. También es necesario conocer los problemas a los que se enfrenta un programador que desea que su programa corra en paralelo eficiente y eficazmente, como son: el partir eficientemente un problema en múltiples tareas y cómo distribuir estas según la arquitectura en particular con que se trabaje.

El cómputo de alto rendimiento (HPC por High-Performance Computing) está formado por un conjunto de computadoras unidas entre sí en forma de Cluster[1], para aumentar su potencia de trabajo y rendimiento.

El paralelismo clásico, o puesto de otra manera, el clásico uso del paralelismo, es el de diseño de programas eficientes en el ámbito científico. La simulación de problemas científicos es un área de gran importancia, los cuales requieren de una gran capacidad de procesamiento y de espacio de memoria, debido a las complejas operaciones que se deben realizar.

Otro uso clásico es el de las gráficas y vídeos de simulaciones generadas por computadora. La generación de fotogramas y videos requiere de una gran cantidad de cálculos matemáticos. Esto supone una tarea muy compleja para un solo procesador, luego es necesario que haya algún tipo de paralelismo, para distribuir la tarea para que esta sea realizada eficiente y eficazmente.

Tradicionalmente, los programas informáticos se han escrito para el cómputo en serie. Para resolver un problema, se construye un algoritmo y se implementa como un flujo en serie de instrucciones. Estas instrucciones se

[1] 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

ejecutan en una unidad central de procesamiento en un ordenador. Sólo puede ejecutarse una instrucción a la vez y un tiempo después de que la instrucción ha terminado, se ejecuta la siguiente.

Actualmente, es una práctica común usar directivas de compilación en equipos paralelos sobre programas escritos de forma secuencial, con la esperanza que sean puestos por el compilador como programas paralelos. Esto en la gran mayoría de los casos genera códigos poco eficientes, pese a que corren en equipos paralelos y pueden usar toda la memoria compartida de dichos equipos, el algoritmo ejecutado continúa siendo secuencial en una gran parte del código.

La computación en paralelo, por el contrario, utiliza simultáneamente múltiples elementos de procesamiento para resolver un problema. Esto se logra mediante la división del problema en partes independientes de modo que cada elemento de procesamiento pueda ejecutar su parte del algoritmo de manera simultánea con los otros. Los elementos de procesamiento son diversos e incluyen recursos tales como una computadora con múltiples procesadores, varios ordenadores en red, Hardware especializado, o cualquier combinación de los anteriores.

Los programas informáticos paralelos son más difíciles de escribir que los secuenciales, porque la concurrencia introduce nuevos tipos de errores de Software, siendo las condiciones de sincronización los más comunes. La comunicación y sincronización entre diferentes subtareas son algunos de los mayores obstáculos para obtener un buen rendimiento del programa paralelo.

**¿Cómo Paralelizo mi programa?** El problema de todos los usuarios de Ciencias e Ingeniería que hacen uso de métodos numéricos y su implementación computacional es: ¿cómo paralelizo mi programa?, esta pregunta no tiene una respuesta simple, depende de muchos factores, por ejemplo: en que lenguaje o paquete está desarrollado el programa, el algoritmo usado para solucionar el problema, a que equipos paralelo tengo acceso y un largo etc. Algunas respuestas ingenuas podrían ser:

- Si uso lenguajes de programación compilables, puedo conseguir un compilador que permita usar directivas de compilación en equipos de memoria compartida sobre programas escritos de forma secuencial, con la esperanza que sean puestos por el compilador como programas paralelos haciendo uso de hilos, OpenMP y optimización del código generado.

- Si uso paquetes como MatLab, Julia o Python, es posible conseguir una versión del paquete que implemente bibliotecas que usen CUDAs, OpenMP o MPI para muchos de los algoritmos más usados en la programación.

- Si mi problema cabe en un sólo equipo, entonces puedo usar dos o más cores para resolver mi problema usando memoria compartida (puedo usar OpenMP, trabajar con hilos o usando paquetes como MatLab o lenguajes como Python, Fortran, C y C++, etc.), pero sólo puedo escalar para usar el máximo número de cores de un equipo (que en la actualidad puede ser del orden de 128 cores, hasta Terabytes de RAM y con almacenamiento de algunas decenas de Terabytes).

- Si mi problema cabe en la GRAM de una GPU, entonces el posible usar una tarjeta gráfica (usando paquetes como MatLab o lenguajes como Python, Fortran, C y C++, etc.), pero sólo puedo escalar a la capacidad de dichas tarjetas gráficas.

- Puedo usar un cluster que usa memoria distribuida-compartida en conjunto con tarjetas gráficas, este tipo de programación requiere una nueva expertes y generalmente implica el uso de paso de mensajes como MPI y rediseño de los algoritmos usados en nuestro programa.

Notemos primero, que no todos los algoritmos son paralelizables. En cualquier caso se tienen que ver los pros y contras de la paralelización para cada caso particular. Pero es importante destacar que existen una gran cantidad de bibliotecas y paquetes que ya paralelizan ciertos algoritmos que son ampliamente usados como la solución de sistemas lineales[2] y no lineales. Además, el tiempo de programación necesario para desarrollar una aplicación paralela eficiente y eficaz para la gran mayoría de los programadores puede ser de semanas o meses en el mejor de los casos. Por ello, es necesario hacer un balance entre las diferentes opciones de paralelización para no invertir un tiempo precioso que puede no justificar dicha inversión económica, de recursos computacionales y sobre todo de tiempo.

[2] Algunas de las bibliotecas más usadas para resolver sistemas lineales usando matrices bandadas y dispersar son PETCs, HYPRE, ATLAS, LAPACK++, LAPACK, EISPACK, LINPACK, BLAS, entre muchas otras alternativas, tanto para implementaciones secuenciales como paralelas y más recientemente para hacer uso de los procesadores CUDA en las GPU de nVidia.

Antes de iniciar con los tópicos del cómputo en paralelo, es necesario conocer cómo está constituida nuestra herramienta de trabajo: la computadora. Los conocimientos básicos de arquitectura de las computadoras nos permite identificar los componentes de Hardware que limitarán nuestros esfuerzos por realizar una paralelización óptima de nuestro programa y en muchos casos descubriremos de la peor manera que los algoritmos usados en nuestro programa no son los más adecuados para paralelizar; y descubriremos con horror que en algunos casos los tiempos de ejecución no mejoran sin importar cuantos equipos adicionales usemos, en el peor de los casos, el tiempo de ejecución se incrementa al aumentar equipos en vez de disminuir.

## 1.1 Computadoras Actuales

La computadora (también conocida como ordenador) actual es una máquina digital programable que ejecuta una serie de comandos para procesar los datos de entrada, obteniendo convenientemente información que posteriormente se envía a las unidades de salida. Una computadora está formada físicamente por numerosos circuitos integrados y varios componentes de apoyo, extensión y accesorios, que en conjunto pueden ejecutar tareas diversas con suma rapidez y bajo el control de un programa (Software).

La constituyen dos partes esenciales, el Hardware, que es su estructura física (circuitos electrónicos, cables, gabinete, teclado, etc.), y el Software, que es su parte intangible (programas, datos, información, documentación, etc.).

Con respecto al Hardware[3], se encuentra compuesto por una serie de dispositivos, clasificados según la función que estos desempeñen. Dicha clasificación se compone de:

[3]En Debian GNU/Linux podemos instalar la aplicación *lshw* para conocer los distintos componentes de la computadora, mediante:

```
# apt install lshw
```

Así, para ver de forma resumida los dispositivos que componen la computadora usamos:

```
# lshw -short
```

Si necesitamos más detalle usamos:

```
# lshw
```

- Los dispositivos de entrada son todos aquellos que permiten la entrada de datos a una computadora. Estos dispositivos (periféricos), son los que permiten al usuario interactuar con la computadora. Ejemplos: teclado, Mouse (ratón), micrófono, Webcam, Scanner, etc.
- Los dispositivos de salida, son todos aquellos que permiten mostrar la información procesada por la computadora. Ejemplos: monitor, impresora, auriculares, altavoces, etc.
- Los dispositivos de comunicación son aquellos que permiten la comunicación entre dos o más computadoras. Ejemplos: Modem, Router, placa de red, Bluetooth, etc.
- Los dispositivos de almacenamiento, son todos aquellos que permiten almacenar datos en el ordenador. Ejemplos: disco duro, Pendrive, Diskette, CD y DVD, etc.
- Los dispositivos de cómputo, son aquellos encargados de realizar las operaciones de control necesarias, sobre el resto de los dispositivos la computadora.

La unidad central de procesamiento (Central Processing Unit CPU) se comunica a través de un conjunto de circuitos o conexiones llamada Bus de datos o canal de datos. El bus conecta la CPU a los dispositivos de almacenamiento, los dispositivos de entrada y los de salida.

En la actualidad, una computadora puede tener una CPU de 32 o de 64[4] bits para describir anchura de registros, Bus de direcciones, Bus de datos o

[4] ¿Por qué se llama x86 a la de 32 bits y x64 a la de 64 bits?

Por pereza. El término adecuado para la arquitectura de 64 bits basada en las tecnologías desarrolladas por Intel y AMD desde 1976 es x86-64, que es como la llamó AMD cuando la desarrolló.

El término "x86" se acuñó porque Intel denominó 8086 a la primera CPU que utilizaba esta microarquitectura. Era una CPU de 16 bits con un conjunto de instrucciones similar al anterior Intel 8085 de 8 bits, por lo que Intel le dio un número más alto. A principios de los años 90, a los desarrollos posteriores de la 8086 se les asignaron números que terminaban en "86": 80186, 80286, 80386 y 80486. En la mayoría de los casos, se omitió la parte "80" y toda la familia de CPU se denominó "x86", por razones obvias. Con lo que habría sido el 80586, Intel eliminó los números porque no podían registrarlos como marca, así que nació la marca "Pentium", que primero significaba quinta generación, pero luego como una simple marca.

Cuando Intel pasó a los 32 bits con el conjunto de instrucciones x86 con el 386 en

instrucciones. Es decir que la cantidad de bits nos permite diferenciar el tipo de CPU que tiene nuestro equipo.

La CPU, el sistema operativo, el Software en general y los Drivers se basan en una misma arquitectura, pero que tenga más o menos bits puede ser significativo en la experiencia del usuario.

En la actualidad, la mayoría de ordenadores cuentan con CPU de 64[5] bits. Ahora bien, todavía hay equipos algo antiguos que cuentan con la arquitectura de 32 bits, por lo que sigue coexistiendo en el mercado y el sector hace desarrollos también pensando en sus capacidades.

Desde el punto de vista funcional es una máquina que posee, al menos,

1985, no cambiaron la marca de nada, pero cuando AMD extendió x86 a 64 bits, lo comercializaron como x86-64. Muchos desarrolladores de Software lo llamaron "AMD64" en su lugar, ya que AMD lo desarrolló. Intel no estaba dispuesto a hacer eso, así que lo llamaron "EMT64" cuando adoptaron las extensiones de 64 bits de AMD después del fracaso de Itanium. Al final, la mayoría de la gente simplemente lo llamó "x64", aunque esto no tiene ningún sentido.

[5] ¿Alguna empresa ha fabricado una CPU de 128 bits?

Si alguien quiere fabricar una CPU de 128 bits, puede utilizar el diseño RISC-V, que admite núcleos de 32, 64 y 128 bits. No hay problema, solo se necesita un diseño adecuado y listo. Muchos dicen que el beneficio de una CPU (núcleo) de 64 bits es la enorme memoria direccionable, 64 bits en comparación con los 32 bits de los núcleos de 32 bits. ¡Pero eso no es cierto!. Todos los núcleos de 64 bits (excepto los muy especiales) tienen un bus de direcciones de 52 o 56 bits. Windows y Linux utilizan internamente un espacio de direccionamiento de a lo más 48 bits. No hay necesidad de memoria física de 64 bits porque hoy en día no es posible tenerla (teóricamente hasta 16 Exabytes).

Algo similar ocurre con los núcleos de 128 bits. Como la memoria no es importante, lo único que queda es la matemática de 128 bits. La pregunta es: ¿quién necesita la matemática de 128 bits? En 1978, el 8086 tenía un coprocesador matemático, el 8087, que admitía tipos de punto flotante de 80 bits. Hoy en día, es similar, pero el tipo máximo admitido sigue siendo de 64 bits que tiene aproximadamente 16 dígitos decimales de precisión con un rango del orden de $1.7 \times 10^{-308}$ a $1.7 \times 10^{308}$, el número de 32 bits que tiene 14 dígitos decimales de precisión con un rango del orden de $3.4 \times 10^{-38}$ a $3.4 \times 10^{38}$. Además, existe el número de 80 bits que tiene 18 dígitos decimales de precisión con un rango del orden de $3.4 \times 10^{-4096}$ a $1.1 \times 10^{4096}$.

En algunos compiladores existen 128 bits de datos, pero no existe matemática de Hardware, sino que se implementa en Software. Se necesitan tipos tan grandes en muy pocas circunstancias.

Lo que importa hoy en día es la paralelización, el procesamiento de múltiples tipos en paralelo. Durante mucho tiempo, x86 ha tenido AVX512, que es un motor matemático de 512 bits (reemplazó al 8087), pero es capaz de trabajar con tipos de 8 a 64 bits en paralelo, por ejemplo, 8 operaciones matemáticas en paralelo utilizando FP64 (punto flotante de 64 bits).

una unidad central de procesamiento, unidad de memoria (Random Access Memory RAM , Read Only Memory ROM y Caché) y dispositivos de entrada/salida (periféricos). Los periféricos de entrada permiten el ingreso de datos, la CPU se encarga de su procesamiento (operaciones aritmético-lógicas) y los dispositivos de salida los comunican a los medios externos. Es así, que la computadora recibe datos, los procesa y emite la información resultante, la que luego puede ser interpretada, almacenada, transmitida a otra máquina o dispositivo o sencillamente impresa; todo ello a criterio de un operador o usuario y bajo el control de un programa de computación.

**CPU** la Unidad Central de Proceso (Central Processing Unit CPU) es aquella parte del procesador que se encarga de ejecutar las diversas acciones que ordenemos al dispositivo que debe llevar a cabo. La CPU es el componente básico dentro de todo dispositivo inteligente, ya que prácticamente cualquier proceso que se ordene al sistema pasa por él. Con el paso del tiempo, además, su eficiencia y calidad ha alcanzado grandes cotas, aunque tecnologías al alza como las NPU han supuesto un mayor salto cualitativo que el de aumentar la potencia bruta de la CPU.

**GPU, para el procesamiento gráfico** la Unidad de Procesamiento Gráfico (Graphics Processing Unit GPU) es el apartado que se dedica a las acciones de mayor peso: las de componente gráfico. De este modo, acciones como la ejecución de videojuegos, o la edición y renderizado de vídeos, se llevan a cabo a través de la GPU del sistema. La calidad del teléfono, ordenador u otro dispositivo inteligente, suele ir supeditada a menudo a la calidad de su GPU, que dependerá de la banda de precio del aparato en cuestión. Eso sí, si los otros componentes no tienen la misma calidad, se puede producir el temido cuello de botella.

En los smartphones, la GPU ya va integrada en los procesadores, pero en los PC's, como AMD y NVIDIA se muestran como marcas especializadas en las GPUs. Cuentan con todo tipo de gamas y también poseen un amplio abanico de precios, un valor siempre directamente proporcional a la calidad y capacidad de sus gráficas, y a su vigencia en el mercado.

**NPU, redes neuronales en tu dispositivo** la Unidad de Procesamiento Neuronal (Neural Processing Unit NPU), a diferencia de la GPU, que cuenta con un funcionamiento paralelo al de la CPU, puede encargarse de

funciones similares a las de la CPU, pero lo hace de un modo mucho más eficiente. Impulsada por Inteligencia Artificial, una NPU[6] es capaz de priorizar procesos para ejecutarlos de un modo exponencialmente más veloz y con un consumo mucho menor. En móviles, se usa especialmente para mejorar el procesado de fotografías, aunque participa en muchos otros procesos.

Además, esta arquitectura todavía tiene años de progreso por delante, a diferencia de la CPU y la GPU, cuyas mejoras ya son de carácter más leve y basadas en aumentar la potencia bruta. La tecnología NPU, en cambio, lleva menos tiempo entre nosotros y todavía tiene mucho margen de mejora para ofrecer un rendimiento cada vez más poderoso.

Desde la llegada de los procesadores de más de un núcleo a PC, como consecuencia de la imposibilidad de hacerlos escalar en potencia solo por velocidad de reloj, la forma de entender los diferentes chips cambió. Han pasado ya dos décadas de dicho cambio y ante la inminente salida de los Chips disgregados o por Chiplets al mercado de masas, no está de más recordar la organización más común en el mundo del Hardware en todo este tiempo.

**¿Qué es un SoC?** las siglas SoC significan System on a Chip y hace referencia a todo chip que tiene la mayoría de componentes integrados en una misma pieza de silicio sin llegar a ser un microcontrolador. Se trata de la pieza de Hardware más usada por el hecho de que a día de hoy todo procesador para PC, teléfono móvil, consola de videojuegos, televisor o incluso servidores, es un SoC y pese a las diferencias entre ellos, todos tienen una organización común.

En realidad, todas las CPU actuales son SoC, ya que se trata de "varias CPU" diseñadas para funcionar alrededor de un elemento de intercomunicación central. Este se encarga de interconectar los diferentes elementos entre sí y de darles acceso a interfaces externas.

Por ejemplo, con la memoria RAM (u otros), a la que está asociado el controlador de memoria que comparten todos sus elementos o a los periféricos, y a los cuales se puede acceder directamente con una serie de interfaces específicas, o a través de un Chipset externo encargado de gestionar los diferentes periféricos y componentes.

[6]Para muestra, la compañía Cerebras con su procesador WSE-3 aglutina 4 billones de transistores, tiene una superficie de 46,225 $mm^2$, integra nada menos que 900,000 núcleos optimizados para IA y es capaz de entrenar hasta 24,000 millones de parámetros, lo que también equivaldría a un rendimiento máximo de IA de 125 petaflops.

En PC, debido a que la comunicación con los periféricos se hace a través del uso de direcciones de memoria de la RAM principal, los componentes relacionados con esta se encuentran subordinados al controlador de memoria. Por lo que son una pieza más conectada a la parte central.

**¿Qué es una APU y en qué se diferencia de un SoC?** las siglas APU significan Accelerated Processor Unit y fue usada por AMD cuando sus CPU e iGPU (o GPU integrada, la veremos a continuación) no traían consigo ningún sistema de gestión de E/S (entrada y salida). Sin embargo, la cosa empezó a cambiar ya con la arquitectura "Carrizo" que fue el nombre clave de los últimos SoC antes del lanzamiento de los AMD Ryzen, donde se incorporaron varias interfaces de periféricos directamente en la CPU.

A día de hoy todo es un SoC, lo que ocurre es que, en sobremesa, las torres tienen tanta conectividad y capacidad de expansión que se suele emplear un Chipset, y lo mismo ocurre en estaciones de trabajo y servidores, pero no en el resto. Y es que si hablamos de un Chip para un PC portátil, una consola o un móvil, entonces al no existir tantas interfaces para periféricos y otros componentes, entonces estás se pueden integrar en un mismo chip.

En realidad, el término APU es más bien comercial de AMD y al día de hoy se utiliza como sinónimo de SoC, pero se puede resumir en que una APU carece de cualquier gestión de periféricos y requiere de un Chip externo para ello. Mientras que un SoC tiene las especificaciones mínimas en ese aspecto, pese a ser también ampliables. Para simplificar la idea, un SoC es una APU más completa.

**¿¡GPU qué es y cuáles son sus características concretas frente a una dGPU?** las siglas iGPU corresponden a integrated GPU, y hace referencia a todo componente de este tipo que se encuentre integrado en una APU o un SoC. Por lo que se trata de procesadores gráficos de potencia limitada que se pueden ver lastrados en velocidad de reloj por el problema del ahogamiento termal (Thermal Throttling) que se produce cuando muchas partes comparten el mismo espacio físico.

Si bien, es posible llegar a ciertos niveles de rendimiento que son aceptables de cara a reproducir las escenas en 3D a tiempo real en los videojuegos, y otras tareas de carácter profesional donde se usa una tarjeta gráfica, estas se ven cuanto menos limitadas:

El hecho de tener que compartir espacio en el mismo Chip con los diferen-

tes núcleos de la CPU produce que esta no pueda alcanzar la misma velocidad de reloj que se alcanzaría siendo un Chip aparte e independiente.

En PC, debido a que como la memoria RAM usa DDR o LPDDR, el ancho de banda es pequeño y hemos de partir del hecho de que el rendimiento de todo Chip gráfico, incluido una iGPU, depende del ancho de banda que se le puede otorgar con dicha memoria RAM.

Los SoC con iGPU actuales tienen un tamaño fijo, definido este por la cantidad de pines soportados por la interfaz con la placa base. Esto limita el tamaño, no solo del Chip, sino también de la gráfica integrada en el mismo.

**iGPU en consolas de videojuegos** al contrario de lo que ocurre con los PC, las consolas de videojuegos no tienen que seguir una serie de normas respecto a sus componentes. Para empezar, no ven el tamaño de sus chips limitados por un estándar de placa base, dado que son productos únicos y exclusivos. Esto les permite tener el tamaño que quieran, incluso más que una CPU convencional para PC, lo que les permite tener una iGPU en su SoC mucho más avanzada.

El otro punto es la memoria utilizada, ya que la de las tarjetas gráficas también se usan como memoria principal. Esta da el ancho de banda necesario para que la iGPU alcance cierto nivel de rendimiento, pero su latencia es mucho más alta que la RAM convencional de PC que está más optimizada en ese aspecto, por lo que el rendimiento de su CPU suele ser más bajo que su equivalente para ordenador.

**¿Qué es una dGPU en un PC o portátil?** las dGPU, o GPU dedicadas, no son otra cosa que las GPU de toda la vida, pero la particularidad es que también son SoC, ya que tiene varios núcleos, especializados en tareas gráficas, alrededor de una interfaz central y compartiendo todos ellos un mismo acceso a memoria.

Sin embargo, carecen de núcleos de CPU en su interior, de ahí a que no se les llame APU o SoC, por el hecho de que de existir estos elementos pasarían a ser una iGPU. Por lo tanto, su principal particularidad de las dGPU es que tienen su propia memoria RAM (SDRAM para ser concretos) la cual suele rodear estos Chips, ya sea en forma de tarjeta gráfica o soldados en la placa de los ordenadores portátiles. A esta la llamamos VRAM y es de uso exclusivo de la dGPU o GPU.

Los equipos de cómputo los podemos clasificar[7] por:

- Equipos móviles: estos equipos buscan un equilibrio entre su capacidad de cómputo versus el rendimiento energético de sus baterías -para operar el mayor tiempo posible sin recargarse- y su peso, entre estos equipos destacan las Laptops, Notebook, Netbook, Ultrabook, tabletas, teléfonos inteligentes, etc.

- Equipos de escritorio: estos equipos al estar permanentemente conectados a la corriente eléctrica pueden tener un mayor número de componentes y disponen de una mejor capacidad disipación de calor por lo que pueden contener una mayor cantidad componentes, como discos, RAM o tarjetas de video y el tamaño, peso o consumo energético no es un inconveniente.

- Servidores: son equipos que suelen atender a múltiples usuarios simultáneamente y disponen de gran cantidad de Cores, RAM, disco y son interconectados por red de alta velocidad con otros servidores para atender las crecientes necesidades de los centros de datos los cuales deben estar permanentemente en operación. Generalmente los equipos son montados en Racks con otras decenas de equipos, por lo que su arquitectura se ve limitada a una moderada generación de calor por parte de sus componentes ya que su sistema de ventilación es por aire para todo el Rack.

- Estaciones de trabajo: son equipos individuales diseñados para atender cargas computacionales intensas, por lo que requieren Hardware más complejo y potente como puede ser múltiples tarjetas de video (con decenas de miles de Cores gráficos) , discos (con cientos de Terabytes), gran cantidad de RAM (pueden llegar a superar el Terabyte) y sistema de refrigeración por aire o líquido, etc.

- Cómputo intensivo: son equipos interconectados por red de alta velocidad con procesadores y tarjetas gráficas dedicadas para el cálculo numérico que soportan cargas intensas por largos periodos de tiempo,

[7] El ordenador del Apollo 11, el Block II, funcionaba a una velocidad de 2 MHz y tenía 2 KB de memoria RAM y 32 KB de memoria ROM. Para ponerlo en contexto, el chip de un cargador USB-C moderno es 563 veces más potente que la computadora que se usó en el Apollo 11, al menos en términos de potencia bruta.

los más comunes son los que forman parte de los Cluster que llegan a tener millones de cores.

**FLOPS** Una medida relativamente objetiva para analizar el rendimiento de un dispositivo suele ser medir sus operaciones de punto flotante por segundo o más conocidas como FLOPS[8]. Hay que tener en cuenta que la medición de FLOPS es muy compleja porque las diferentes operaciones en punto flotante llevan diferentes cantidades de tiempo para ejecutarse. Y no todo el mundo utiliza las mismas operaciones para establecer los cálculos.

Por ejemplo, una división simple como 1/5, toma significativamente menos tiempo que el cálculo del logaritmo de 5. Por eso, se estableció el algoritmo de Linpack como un estándar representativo con el que poder medir todos los sistemas bajo el mismo baremo de FLOPS-.

Es importante señalar que el algoritmo de Linpack utiliza el formato en punto flotante de doble precisión (64 bits[9] que tiene aproximadamente 16 dígitos decimales de precisión con un rango del orden de $1.7 \times 10^{-308}$ a $1.7 \times 10^{308}$). Sin embargo, como veremos la mayoría de los valores que dan los fabricantes son con precisión simple (32 bits que tiene 14 dígitos decimales de precisión con un rango del orden de $3.4 \times 10^{-38}$ a $3.4 \times 10^{38}$). Además,

[8] El Cray-1 fue puesto marcha en 1975 y utilizaba una CPU a 80 MHz y llevaba integrada una unidad SIMD de 64 bits de precisión de punto flotante, lo cual fue un salto de gigante que permitió un salto de los 3 MFLOPS de potencia del CDC 6600 a los 160 MFLOPS en el Cray-1.

[9] ¿Por qué no hay un procesador de 128 bits, si los hay de 64 y 32 bits?

En gran medida, esto se reduce a la cuestión de qué se entiende realmente por "procesador de 64 bits". Al inicio de las computadoras electrónicas, un "procesador de 8 bits" tenía registros de 8 bits, pero la mayoría tenía direccionamiento de 16 bits (y la Unidad de Aritmética y Lógica ALU a veces tenía 4 bits, por lo que cuando se añadían dos registros de 8 bits, la CPU a menudo lo hacía en dos pasos, añadiendo los 4 bits inferiores y luego añadiendo por separado los 4 bits superiores más el acarreo de los 4 inferiores).

Hoy en día, hemos invertido un poco esa situación: un procesador de "64 bits" (más o menos) tiene direccionamiento de 64 bits, pero tiene registros de 128 bits, 256 bits y, en algunos casos, incluso de 512 bits. Pero como los usos de números mayores a 64 bits son bastante raros, esos registros más grandes se usan normalmente para realizar operaciones en una cantidad de operandos más pequeños con una sola instrucción. Por lo tanto, en lugar de una ALU que tiene la mitad del tamaño de un registro y realiza una sola operación en múltiples ciclos de reloj, tenemos una ALU que tiene entre 2 y 4 veces el tamaño de un registro normal y realiza múltiples operaciones en un solo ciclo de reloj.

Dependiendo de cómo prefieras ver las cosas, podrías argumentar razonablemente que una CPU actual de "64 bits" es en realidad una CPU de 128 bits, 256 bits o incluso 512 bits.

los valores que dan los fabricantes suelen ser teóricos y en la práctica suelen ser inferiores debido a otros factores limitantes como la frecuencia de reloj o la velocidad de las memorias ROM y RAM.

Por tanto, aunque todos hemos acabado midiendo el rendimiento en FLOPS, no es una medida absoluta de la potencia del CPU ni de una GPU. Por ejemplo para algunos dispositivos tenemos:

**Móviles** El SoC Snapdragon 821 que monta una GPU Adreno 530 tiene una potencia de 519.2 gigaFLOPS (0.52 TFLOPS), y los Chips Apple A9X del iPad Pro alcanzan los 345.6 gigaFLOPS (0.35 TFLOPS), todos ellos medidos con precisión simple de 32-bits.

**CPU**

- Intel Xeon W-3245: 1.4 TFLOPS
- Intel Core i9-9900X: 1.2 TFLOPS
- AMD Ryzen 9 3950X: 1.1 TFLOPS

Los procesadores de gama media-alta rondan el medio TFLOPS:

- AMD Ryzen 7 3700X: 546.0 GFLOPS - 0.55 TFLOPS
- Intel Core i9-9900: 499.0 GFLOPS - 0.50 TFLOPS
- AMD Ryzen 5 3600X: 461.0 GFLOPS - 0.46 TFLOPS

**Tarjetas Gráficas** Ojo: La tabla está ordenada por los valores en precisión simple (32-bit) primer columna

| GPU | FP32 TFLOPS | FP64 TFLOPS |
|---|---|---|
| TITAN V | 13.8 | 6.9 |
| Radeon RX Vega 64 | 12.7 | 0.8 |
| GeForce GTX 1080 Ti | 11.3 | 0.4 |
| GeForce GTX 1080 | 8.9 | 0.3 |
| Radeon R9 Fury X | 8.6 | 0.5 |
| Radeon HD 7990 | 7.8 | 1.9 |
| GeForce GTX 1070 | 6.5 | 0.2 |
| Radeon RX 480 | 5.8 | 0.4 |

| | | |
|---|---|---|
| GeForce GTX 690 | 5.6 | 0.2 |
| Radeon R9 290X | 5.6 | 0.7 |
| GeForce GTX 780 Ti | 5.3 | 0.2 |
| Radeon HD 6990 | 5.1 | 1.3 |
| GeForce GTX 980 | 4.9 | 0.15 |
| Radeon RX 470 | 4.9 | 0.3 |
| Radeon R9 290 | 4.8 | 0.6 |
| GeForce GTX Titan | 4.7 | 1.5 |
| GeForce GTX 1060 | 4.4 | 0.14 |
| Radeon HD 7970 GHz | 4.3 | 1.1 |
| GeForce GTX 780 | 4.1 | 0.17 |
| Radeon R9 280X | 4.0 | 1.0 |
| Radeon R9 280 | 3.3 | 0.83 |
| GeForce GTX 680 | 3.1 | 0.13 |
| Radeon HD 7950 | 2.9 | 0.71 |

Como podemos ver, las tarjetas gráficas de Nvidia, normalmente, tienen una potencia muy alta en precisión simple, pero muy mala en precisión doble. La precisión simple es la que se usa en los juegos, pero la precisión doble es la que se utiliza en los cálculos complejos científicos y en el minado de muchas criptomonedas.

**Consolas** Todas ellos son en valores de precisión simple (32-bit)

- PlayStation 4: 1.3 TFLOPS
- Xbox One: 1.8 TFLOPS
- PlayStation 4 Pro: 4.2 TFLOPS
- Nintendo Switch: entre 0.4 y 0.5 TFLOPS
- PlayStation 5 promete una GPU con 10.28 TFLOPS
- La Xbox Series X promete una GPU de 12 TFLOPS

**SuperCómputo** Hace pocos se publicó la $64^a$ edición del ranking «Top 500» (noviembre del 2024) que clasifica los Clusters con mayor rendimiento del mundo. Esta lista se basa en la medida del rendimiento de los sistemas

utilizando la prueba de referencia LINPACK, que calcula la velocidad a la que un sistema puede resolver un conjunto de ecuaciones lineales.

Y en esta nueva edición, los Clusters que ocuparon los primeros tres lugares son:

- El sistema El Capitan del Laboratorio Nacional Lawrence Livermore, en California, EE.UU., es el nuevo sistema número 1 del TOP500. El sistema HPE Cray EX255a ha obtenido 1.742 exaflop/s en el benchmark HPL. El Capitan tiene 11,039,616 núcleos y está basado en procesadores AMD EPYC(TM) de $4^a$ generación con 24 núcleos a 1.8 GHz y aceleradores AMD Instinct(TM) MI300A. Utiliza la red Cray Slingshot 11 para la transferencia de datos y alcanza una eficiencia energética de 60.3 Gigaflops/vatio.

- Frontier es ahora el sistema número 2 del TOP500. Este sistema HPE Cray EX fue el primer sistema estadounidense con un rendimiento superior a un exaflop/s. Está instalado en el Laboratorio Nacional Oak Ridge (ORNL) en Tennessee, EE.UU., donde se utiliza para el Departamento de Energía (DOE). Actualmente ha alcanzado 1.353 Exaflop/s utilizando 8,699,904 núcleos. La arquitectura HPE Cray EX combina CPUs AMD EPYC(TM) de tercera generación optimizadas para HPC e IA, con aceleradores AMD Instinct(TM) 250X y una interconexión Slingshot-11.

- Aurora es actualmente el número3 con una puntuación HPL preliminar de 1.012 Exaflop/s. Está instalado en Argonne Leadership Computing Facility, Illinois, EE. UU., donde también se opera para el Departamento de Energía (DOE). Este nuevo sistema de Intel se basa en HPE Cray EX - Intel Exascale Compute Blades. Utiliza procesadores Intel Xeon CPU Max Series, aceleradores Intel Data Center GPU Max Series y una interconexión Slingshot-11.

Para poner en contexto los avances en este campo, en el año 2004 IBM era dueña y señora del mundo de la supercomputación, su espectacular BlueGene/L dominaba la lista TOP 500. Aquel monstruo contaba con 32,768 procesadores PowerPC 440 a 700 MHz y 16 TB de memoria. 20 años después una sola NVIDIA GeForce RTX 4090 con 24 GB de memoria GDDR6X es más potente que esa supercomputadora -lo es al menos en rendimiento

bruto-, BlueGene/L contaba en ese momento con un rendimiento de 70.72 TFLOPS, pero la propia NVIDIA dejaba claro en el lanzamiento de sus RTX 4090 que estas tarjetas gráficas contaban con una potencia de 83 TFLOPS.

Es más, cuatro RTX 4090 con soporte FP8 logran también rivalizar con la supercomputadora más potente de 2009. Y eso sin apretarle las tuercas a las RTX 4090: en noviembre de 2022 es precisamente lo que hicieron en Wccftech y lograron que la RTX 4090 se convirtiera en la primera tarjeta gráfica del mundo en alcanzar los 100 TFLOPS.

Esa comparación es como decimos real en esa potencia de cálculo en bruto, pero también es cierto que en esa y otras supercomputadoras se tenian mecanismos especiales de comunicación entre procesadores o de transferencia de datos, algo para lo que las GPUs actuales, aún siendo sobresalientes, no están tan optimizadas.

**¿Cómo Trabaja una Computadora?** Todas las computadoras sean de uno o más procesadores ejecutan los programas realizando los siguientes pasos:

1. Se lee una instrucción
2. Se decodifica la instrucción
3. Se encuentra cualquier dato asociado que sea necesario para procesar la instrucción
4. Se procesa la instrucción
5. Se escriben los resultados

Esta serie de pasos, simple en apariencia, se complican debido a la jerarquía de memoria RAM, en la que se incluye la memoria Caché, la memoria principal y el almacenamiento no volátil como pueden ser los discos duros o de estado sólido (donde se almacenan las instrucciones y los datos del programa), que son más lentos que el procesador en sí mismo. Con mucha frecuencia, el paso (3) origina un retardo muy largo (en términos de ciclos del procesador) mientras los datos llegan en el bus de la computadora.

Durante muchos años, una de las metas principales del diseño microinformático ha sido la de ejecutar el mayor número posible de instrucciones en paralelo, aumentando así la velocidad efectiva de ejecución de un programa.

No obstante, estas técnicas han podido implementarse en Chips semiconductores cada vez más pequeños a medida que la fabricación de estos fue progresando y avanzando, lo que ha abaratado notablemente su costo.

El procesador es el cerebro de un ordenador. No hay que olvidar otros componentes como la memoria, el almacenamiento o la tarjeta gráfica dedicada, desde luego, pero el procesador está un escalafón por encima en la jerarquía.

Piensa que, si cambiamos el procesador en dos equipos con la misma memoria, almacenamiento o tarjeta gráfica, el comportamiento puede variar notablemente. Sin embargo, para un mismo procesador, los cambios en el resto de componentes no impactan de forma tan directa en la experiencia de uso de un equipo.

**¿Qué es una CPU?** Antes de nada, vamos a definir exactamente lo que es una CPU o un procesador. Como bien indican sus siglas en inglés (Central Processing Unit) es la unidad de procesamiento -puede ser Intel, AMD, ARM, etc- encargada de interpretar las instrucciones de un Hardware haciendo uso de distintas operaciones aritméticas y matemáticas. Características principales de un procesador:

- Frecuencia de reloj. Este primer término hace referencia a la velocidad de reloj que hay dentro del propio procesador. Es un valor que se mide en Mhz o Ghz y es básicamente la cantidad de potencia que alberga la CPU. La mayoría de ellas cuentan con una frecuencia base -para tareas básicas- y otra turbo que se utiliza para procesos más exigentes -con un aumento en el consumo de energía y por ende un aumento en la temperatura del procesador, requiriendo sistemas de disipación de calor eficientes-.

- Consumo energético. Es normal que nos encontremos con CPU 's donde su consumo energético varía notablemente. Es un valor que se muestra en vatios (W) y como es obvio, aquellos procesadores de gama superior, serán más propensos a consumir más energía. Ante esto, es importante contar con un eficiente sistema de enfriamiento además de contar con una fuente de alimentación acorde a la potencia requerida por el procesador, la tarjeta gráfica y sus respectivos sistemas de enfriamiento.

- Número de núcleos. Con el avance de la tecnología, ya es posible encontrar tanto procesadores de Intel como de AMD que cuentan ya con

decenas de núcleos. Estos cores son los encargados de llevar a cabo multitud de tareas de manera simultánea.

- Número de hilos. Si un procesador tiene Hyperthreading en el caso de Intel o SMT (Simultaneous Multi-Threading) en el caso de AMD, significa que cada uno de los núcleos es capaz de realizar dos tareas de manera simultánea, lo que se conoce como hilos de proceso. Por lo tanto, un procesador de cuatro núcleos físicos con Hyperthreading tendría ocho hilos de proceso, y sería capaz de ejecutar ocho órdenes al mismo tiempo -los hilos no tienen las mismas capacidades de un core real y en muchos casos su uso merma el rendimiento del CPU, pero los sistemas operativos los reconocen como si fueran cores reales-[10].

- Memoria Caché. A la hora de "recordar" cualquier tarea, el propio ordenador hace uso de la memoria RAM. Sin embargo no es eficiente este proceso y por tanto es necesario que utilice la memoria Caché de la CPU para paliar esta deficiencia. El Caché se caracteriza porque se llega a ella de forma más rápida y puede ser tipo L1, L2 y L3.

- Zócalo. Es el tipo de conector con pines o Socket al que se conecta la placa base. Por ejemplo, las últimas de Intel suelen tener el Socket LGA 1200, mientras que las de AMD con Ryzen son AM4.

- Red. Si bien la red es un recurso indispensable en un equipo de cómputo, en el caso de equipos paralelos la velocidad de la red es el mayor cuello de botella en cuanto a rendimiento, por ello es necesario usar redes de alto desempeño como las de InfiniBand con un alto costo económico pero de alto desempeño que pueden llegar al orden de cientos de Gigabytes por segundo.

**Nuevos Procesadores** la creciente demanda de dispositivos de cómputo ha generado una gran variedad de procesadores, los podemos clasificar como:

- Procesador compuesto por múltiples núcleos de alta eficiencia -con un consumo energético reducido- que sacrifican potencia de procesamiento

[10] El AMD EPYC 9845 de 160 núcleos y 320 hilos a una frecuencia de 2,00 GHz basada en Zen 5c, este se acompaña de 640 MB de caché L3.

en aras de extender la carga útil de las baterías de los dispositivos móviles.

- Procesador compuesto por múltiples núcleos de alto rendimiento que pueden estar al tope de su capacidad sin generar excesivo calor y son especialmente usados en servidores y en cómputo intensivo.

- Procesadores compuestos por múltiples núcleos de alto rendimiento que pueden ajustar su velocidad de reloj de manera dinámica para tratar cargas de trabajo pesadas por un cierto tiempo -pues generan gran cantidad de calor-, por lo que requieren un sistema eficiente de enfriamiento, son ideales para estaciones de trabajo.

- Procesadores compuestos por múltiples núcleos híbridos que en lugar de tener un único tipo de núcleo multipropósito, estos Chips cuentan con dos grupos de núcleos. El primero de ellos, compuesto por múltiples núcleos de alta eficiencia, se encarga de procesar las tareas más livianas o en segundo plano que deba realizar un procesador, todo ello, con un consumo energético menor. El otro grupo, compuesto por múltiples núcleos de alto rendimiento, sigue una dinámica opuesta, su consumo ener-gético es superior, pero únicamente entran en funcionamiento cuando la tarea en cuestión requiere un extra de procesamiento.

Para gestionar esta división de núcleos híbridos, se ha integrado un "Thread Director", un elemento que se encarga de determinar qué núcleo procesa cada tarea. Las compañías, además, ha modificado cómo funciona la caché de sus procesadores:

- Cada núcleo de rendimiento tiene su propia caché L2.

- Cada cluster de núcleos de eficiencia tiene una "piscina" de memoria L2 común, de la que beben todos los núcleos que sean partícipes.

- Tanto los núcleos de rendimiento como los de eficiencia tienen acceso a una "piscina" de memoria L3 común para todos ellos.

Otros de los cambios que impactarán en el desempeño de las CPUs es el aumento de velocidad y una mayor cantidad de memoria Caché, compatibilidad con memorias DDR6 y con la interfaz PCIe 6.0.

**PCIe** PCI Express (Peripheral Component Interconnect Express), abreviado como PCIe, es una tecnología de conexión de Hardware utilizada para la comunicación de alta velocidad entre diferentes componentes de un equipo informático. Este estándar se ha convertido en la interfaz más habitual para la conexión de tarjetas de expansión, como tarjetas gráficas que sirven para correr juegos, tarjetas de sonido, tarjetas de red y dispositivos de almacenamiento de alta velocidad.

Una de las ventajas destacadas de PCI Express es su arquitectura de canales independientes, que permiten la transferencia simultánea de datos en ambos sentidos. Cada carril tiene una tasa de transferencia específica, medida en gigabits por segundo (Gbps), y la capacidad de un slot PCIe se expresa como el número de carriles que tiene. Esto se traduce en un ancho de banda total mayor, lo que facilita la conexión de dispositivos que requieren altas tasas de transferencia, como las tarjetas gráficas modernas o los dispositivos de almacenamiento de última generación.

Los diferentes tipos de ranuras de PCI Express según su tamaño PCIe X1 carriles 1, pines 18, PCIe x4 carriles 4, pines 32, PCIe x8 carriles 8, pines 49, PCIe x16 Carriles 16, pines 82.

Adicionalmente, también es interesante fijarse en las diferentes versiones que se han ido lanzando desde que PCIe se lanzó al mercado:

- PCIe 1.0 ancho banda 8 GB/s, velocidad de transferencia 2.5 GT/s
- PCIe 2.0 ancho banda 16 GB/s, velocidad de transferencia 5 GT/s
- PCIe 3.0 ancho banda 32 GB/s, velocidad de transferencia 8 GT/s
- PCIe 4.0 ancho banda 64 GB/s, velocidad de transferencia 16 GT/s
- PCIe 5.0 ancho banda 128 GB/s, velocidad de transferencia 32 GT/s
- PCIe 6.0 ancho banda 256 GB/s, velocidad de transferencia 64 GT/s

**Características Arquitectónicas** los procesadores Intel x86 admiten un formato de precisión extendido de 80 bits con un significado de 64 bits, que es compatible con el especificado en el estándar IEEE. Cuando un compilador usa este formato con registros de 80 bits para acumular sumas y productos internos, está trabajando efectivamente con un redondeo unitario de $2^{-64}$ en vez de $2^{-53}$ para precisión doble, dando límites de error más pequeños en un factor de hasta $2^{11} = 2048$.

Algunos procesadores Intel y AMD tienen una operación fusionada de multiplicación y suma (FMA), que calcula una multiplicación y una suma combinadas $x + yz$ con un error de redondeo en lugar de dos. Esto da como resultado una reducción en los límites de error por un factor 2.

Las operaciones FMA de bloques de precisión mixta $D = C + AB$, con matrices $A, B, C$ y $D$ de tamaño fijo, están disponibles en las unidades de procesamiento tensorial de Google, las GPU NVIDIA y en la arquitectura ARMv8-A. Para entradas de precisión media, estos dispositivos pueden producir resultados de calidad de precisión simple, lo que puede proporcionar un aumento significativo en la precisión cuando los bloques FMA se encadenan para formar un producto matricial de dimensión arbitraria.

**Meltdown y Spectre** El tres de enero del 2018 se dio a conocer al público, que 6 meses antes se habían detectado dos distintos fallos en los procesadores de los equipos de cómputo, comunicaciones y redes de internet que usamos. Esto para dar tiempo a los desarrolladores de procesadores y de sistemas operativos de implementar estrategias para mitigar el problema. Estos son problemas de diseño de los procesadores de Intel, AMD, IBM POWER y ARM, esto significa que procesos con privilegios bajos -aquellos que lanzan las aplicaciones de usuarios convencionales- podían acceder a la memoria del Kernel del sistema operativo[11].

Un ataque que explota dicho problema permitiría a un Software malicioso espiar lo que están haciendo otros procesos y también espiar los datos que están en esa memoria en el equipo de cómputo (o dispositivo móvil) atacado. En máquinas y servidores multiusuario, un proceso en una máquina virtual podría indagar en los datos de los procesos de otros procesos en ese servidor compartido.

Ese primer problema, es en realidad solo parte del desastre. Los datos actuales provienen especialmente de un grupo de investigadores de seguridad formados por expertos del llamado Project Zero[12] de Google. Ellos han publicado los detalles de dos ataques (no son los únicos[13]) basados en estos

[11] En GNU/Linux, el Kernel (si usamos una versión actualizada) nos indica las fallas del procesador a las que es vulnerable, usando:

```
$ cat /proc/cpuinfo
$ lscpu
```

[12] https://googleprojectzero.blogspot.com/

[13] Entre las distintas vulnerabilidades detectadas y sus variantes resaltan: Meltdown

fallos de diseño. Los nombres de esos ataques son Meltdown y Spectre. Y en un sitio Web dedicado a describir estas vulnerabilidades destacan que "aunque los programas normalmente no tienen permiso para leer datos de otros programas, un programa malicioso podría explotar Meltdown, Spectre y apropiarse de secretos almacenados en la memoria de otros programas". Como revelan en su estudio, la diferencia fundamental entre ambos es que Meltdown permite acceder a la memoria del sistema, mientras que Spectre permite acceder a la memoria de otras aplicaciones para robar esos datos.

Ya que Meltdown y Spectre son problemas de diseño en los procesadores, no es posible encontrar solución por Hardware para los procesadores existentes y dado que constantemente aparecen nuevas formas de explotar dichos fallos, la única manera de mantener el equipo de cómputo, comunicaciones y redes de internet a salvo es mediante Software que debe implementar las soluciones en los sistemas operativos. En particular en el Kernel de Linux se trabaja en parchar en cada versión del Kernel todos los fallos reportados, por esto y por otra gama de fallos e inseguridades es necesario mantener siempre el sistema operativo y sus aplicaciones actualizadas.

Como se había comentado anteriormente, estos problemas de diseño afectan a todos los procesadores Intel, AMD, IBM POWER y ARM. Eso incluye básicamente a todos los procesadores que están funcionando al día de hoy[14] en nuestros equipos, ya que estos procesadores llevan produciéndose desde 1995. Afecta a una amplia gama de sistemas.

En el momento de hacerse pública su existencia se incluían todos los dispositivos que no utilizasen una versión convenientemente parcheada de IOS, GNU/Linux, MacOS, Android, Windows y Android. Por lo tanto, muchos servidores y servicios en la nube se han visto impactados, así como potencialmente la mayoría de dispositivos inteligentes y sistemas embebidos que utilizan procesadores con arquitectura ARM (dispositivos móviles, televisores

(AC, DE, P, SM, SS, UD, GP, NM, RW, XD, BR, PK, BND), Spectre (PHT, BTB, RSB, STL, SSB, RSRE), PortSmash, Foreshadow, Spoiler, ZombieLoad (1 y 2), Kaiser, RIDL, Plundervolt, LVI, Take a Way, Collide+Probe, Load+Reload, LVI-LFB, MSD, CSME, RYZENFALL (1, 2, 3, 4), FALLOUT (1, 2, 3) , CHIMERA (FW, HW), MASTERKEY (1, 2, 3), SWAPGS, ITLB_Multihit, SRBDS, L1TF, etc. Más información en:

https://cve.mitre.org
https://meltdownattack.com/

[14]Solo en el año 2021 se detectaron 16 vulnerabilidades en procesadores INTEL y 31 en los procesadores AMD.

inteligentes y otros), incluyendo una amplia gama de equipo usado en redes. Se ha considerado que una solución basada únicamente en Software para estas fallas alenta los equipos de cómputo entre un 20 y un 40 por ciento dependiendo de la tarea que realizan y el procesador del equipo.

**Memoria RAM** La memoria RAM (Random Access Memory) o memoria de acceso aleatorio es un componente físico de nuestro ordenador, generalmente instalado sobre la misma placa base. La memoria RAM es extraíble y se puede ampliar mediante módulos de distintas capacidades.

La función de la memoria RAM es la de cargar los datos e instrucciones que se ejecutan en el procesador. Estas instrucciones y datos provienen del sistema operativo, dispositivos de entrada y salida, de discos duros y todo lo que está instalado en el equipo.

En la memoria RAM se almacenan todos los datos e instrucciones de los programas que se están ejecutando, estas son enviadas desde las unidades de almacenamiento antes de su ejecución. De esta forma podremos tener disponibles todos los programas que ejecutamos. Se llama memoria de acceso aleatorio porque se puede leer y escribir en cualquiera de sus posiciones de memoria sin necesidad de respetar un orden secuencial para su acceso.

De forma general existen o han existido dos tipos de memorias RAM. Las de tipo asíncrono, que no cuentan con un reloj para poder sincronizarse con el procesador. Y las de tipo síncrono que son capaces de mantener la sincronización con el procesador para ganar en eficacia y eficiencia en el acceso y almacenamiento de información en ellas. Veamos cuales existen de cada tipo.

**Memorias de Tipo Asíncrono o DRAM** las primeras memorias DRAM (Dinamic RAM) o RAM dinámica eran de tipo asíncrono. Se denomina DRAM por su característica de almacenamiento de información de forma aleatoria y dinámica. Su estructura de transistor y condensador hace que para que un dato quede almacenado dentro una celda de memoria, será necesario alimentar el condensador de forma periódica.

Estas memorias dinámicas eran de tipo asíncrono, por lo que no existía un elemento capaz de sincronizar la frecuencia del procesador con la frecuencia de la propia memoria. Esto provocaba que existiera menor eficiencia en la comunicación entre estos dos elementos.

**Memorias de Tipo Síncrono o SDRAM** a diferencia de las anteriores esta memoria RAM dinámica cuenta con un reloj interno capaz de sincronizar esta con el procesador. De esta forma se mejoran notablemente los tiempos de acceso y la eficiencia de comunicación entre ambos elementos. Actualmente todos nuestras computadoras cuentan con este tipo de memorias operando en ellos. Las principales tipos de memoria son: DDR, DDR2, DDR3, DDR4 y la nueva DDR5. Donde las tasas de transferencia (GB/s) son: DDR (2.1 - 3.2), DDR2 (4.2 - 6.4), DDR3 (8.5 - 14.9), DDR4 (17 - 25.6) y DDR5 (38.4 - 51.2).

Aparte las características propias de cada una de las diferentes memorias DDR, la característica más importante es que, por ejemplo, en la memoria DDR4 cuatro cores pueden acceder simultáneamente a ella y en la DDR5 serán cinco cores.

**Caché L1, L2 y L3** La memoria Caché es otra de las especificaciones importantes de los procesadores, y sirve de manera esencial de la misma manera que la memoria RAM: como almacenamiento temporal de datos. No obstante, dado que la memoria Caché está en el procesador en sí, es mucho más rápida y el procesador puede acceder a ella de manera más eficiente, así que el tamaño de esta memoria puede tener un impacto bastante notable en el rendimiento, especialmente cuando se realizan tareas que demandan un uso intensivo del CPU como en el cómputo de alto desempeño o cómputo científico.

La Caché se divide en diferentes jerarquías de acceso:

- La Caché L1 es el primer sitio donde la CPU buscará información, pero también es la más pequeña y la más rápida, a veces para mayor eficiencia, la Caché L1 se subdivide en L1d (datos) y L1i (instrucciones), actualmente los procesadores modernos en cada core tiene su propio cache de datos e instrucciones.

- La Caché L2 suele ser más grande que la L1 pero es algo más lenta. Sin embargo, por norma general es la que mayor impacto tiene en el rendimiento, este también está incluido en cada core.

- La Caché L3 es mucho más grande que las anteriores, y generalmente se comparte entre todos los núcleos del procesador (a diferencia de las anteriores, que normalmente van ligadas a cada core). Este tercer nivel

es en el que buscará el procesador la información tras no encontrarla en la L1 y L2, por lo que su tiempo de acceso es todavía mayor.

Para poner en contexto la relevancia de la memoria Caché, supongamos que el acceso a los datos de la memoria Caché L1 por el procesador es de dos ciclos de reloj, el acceso a la memoria Caché L2 es de 6 ciclos de reloj, el acceso a la memoria Caché L3 es de 12 ciclos y el acceso a la RAM es de 32 ciclos de reloj.

Además supongamos que la operación suma y resta necesitan de 2 ciclos de reloj para completar la operación una vez que cuente con los datos involucrados en dicha operación, que la multiplicación requiere 4 ciclos de reloj para completar la operación, la división necesita 6 ciclos de reloj para completar la operación y estamos despreciando el tiempo necesario para poner los datos del Caché L1 a los registros del procesador para poder iniciar el cálculo, así también despreciamos el tiempo requerido para sacar el resultado de los registros del procesador al Caché L1.

Esto nos da una idea del número máximo teórico de operaciones básicas que un procesador puede realizar por segundo dependiendo de la velocidad de reloj de la CPU[15].

Si nosotros necesitamos hacer la multiplicación de una matriz $\underline{\underline{A}}$ es de tamaño $n \times n$ por un vector $\underline{u}$ de tamaño $n$ y guardar el resultado en el vector $\underline{f}$ de tamaño $n$. Entonces algunos escenarios son posibles:

1. Si el código del programa cabe en el Caché L1 de instrucciones y la matriz $\underline{\underline{A}}$, los vectores $\underline{u}$ y $\underline{f}$ caben íntegramente en el Caché L1 de datos, entonces el procesador estará siendo utilizado de forma óptima al hacer los cálculos pues no tendrá tiempos muertos por espera de datos.

2. Si el código del programa cabe en el Caché L1 de instrucciones y los vectores $\underline{u}$ y $\underline{f}$ caben íntegramente en el Caché L1 de datos pero la matriz $\underline{\underline{A}}$ está dispersa entre los Cachés L1 y L2, entonces el procesador estará teniendo algunos tiempos muertos mientras carga la parte que necesita de la matriz del Caché L2 a L1 para hacer los cálculos y

[15] Por ejemplo en un procesador AMD Ryzen 9 3900X con 12 Cores (2 Threads por Core) por procesador emulando un total de 24 Cores, corre a una frecuencia base de 3,340 MHz, con una frecuencia mínima de 2,200 MHz y máxima de 4,917 Mhz, con Caché L1d de 384 KiB, L1i de 384 KiB, Caché L2 de 6 MiB y Caché L3 de 64 MiB.

utilizado de forma óptima el procesador mientras no salga del Caché L1.

3. Si el código del programa cabe en el Caché L1 de instrucciones y los vectores $\underline{u}$ y $\underline{f}$ caben íntegramente en el Caché L1 de datos pero la matriz $\underline{\underline{A}}$ está dispersa entre los Cachés L1, L2 y L3, entonces el procesador estará teniendo muchos tiempos muertos mientras carga la parte que necesita de la matriz del Caché L3 y L2 a L1 para hacer los cálculos resultando en mediana eficiencia en el uso del procesador.

4. Si el código del programa cabe en el Caché L1 de instrucciones y los vectores $\underline{u}$ y $\underline{f}$ caben íntegramente en los Cachés L3, L2 y L1 pero los datos de la matriz $\underline{\underline{A}}$ está dispersa entre la RAM y los Cachés L3, L2 y L1, entonces el procesador estará teniendo un exceso de tiempos muertos mientras carga la parte que necesita de la matriz de la RAM a los Cachés L3, L2 y L1 para hacer los cálculos resultando en una gran pérdida de eficiencia en el uso del procesador.

Además, debemos recordar que la computadora moderna nunca dedica el cien por ciento del CPU a un solo programa, ya que los equipos son multitarea[16] y multiusuario[17] por lo que la conmutación de procesos (que se realiza cada cierta cantidad de milisegundos) degrada aún más la eficiencia computacional de los procesos que demandan un uso intensivo de CPU[18].

[16] Cuentan con la capacidad para ejecutar varios procesos simultáneamente en uno o más procesadores, para ello necesitan hacer uso de la conmutación de tareas, es decir, cada cierto tiempo detiene el programa que está corriendo y guardan sus datos, para poder cargar en memoria otro programa y sus respectivos datos y así reiniciar su ejecución por un período determinado de tiempo, una vez concluido su tiempo de ejecución se reinicia la conmutación de tareas con otro proceso.

[17] 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.

[18] 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

**Last Level Cache** se le llama Last Level Cache siempre al último nivel de Caché de una CPU, existen dos tipos:

- Last Level Cache Estándar.
- Victim Cache.

Una Victim Cache no actúa como la Caché de último nivel de una CPU, sino que en ese caso lo hace el penúltimo nivel y en la Victim Cache acaban los últimos datos descartados de la Caché y que han sido volcados en la RAM, los cuales son copiados en la Victim Cache para poder acceder a ellos más rápido.

**Smart Cache** la Smart Cache (o Caché) es esencialmente L3 pero optimizada por Intel para ser más eficiente a la hora de compartir la información en los núcleos de la CPU. A efectos prácticos, se comporta de igual manera que la Caché L3.

**Disco** Son dispositivos no volátiles (los hay del orden de hasta 32 TB y continuamente incrementan su capacidad[19]), lo que significa que retienen datos

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.

[19]Durante años la tecnología más popular entre los fabricantes ha sido la PMR (Perpendicular Magnetic Recording), también conocida como CMR (Conventional Magnetic Recording). A esta tecnología luego se le sumó la variante SMR (Shingled Magnetic Recording), que lograba aumentar la densidad de grabación, pero lo hacía sacrificando velocidad de transferencia y fiabilidad de las operaciones.

Western Digital ha creado la tecnología ePMR (energy-assisted Perpendicular Magnetic Recording) que permite ofrecer mayores densidades de grabación y, según este fabricante, mejorar la fiabilidad de las escrituras y evitar así los sacrificios que había que hacer con SMR (en los últimos tiempos han aparecido unidades de 20 a 32 TB basadas en dicha tecnología).

Más interesante aún es el sistema de grabación MAMR (Microwave-Assisted Magnetic Recording) que hace uso de microondas para calentar el medio de almacenamiento y así lograr mejorar densidad de grabación y fiabilidad de lecturas y escrituras. Hace años ya prometían que gracias a esta tecnología contaríamos con unidades de 40 TB en 2025, pero parece que dicho logro aún tardará en llegar.

incluso cuando no tienen energía. La información almacenada permanece segura e intacta a menos que el disco duro sea destruido o interferido. La información se almacena o se recupera de manera aleatoria en lugar de acceso secuencial. Esto implica que se puede acceder a los bloques de datos en cualquier momento sin necesidad de pasar por otros bloques de datos.

Actualmente, podemos agrupar los discos duros disponibles en cuatro tipos:

- Parallel Advanced Technology Attachment (PATA)
- Serial ATA (SATA)
- Interfaz de sistema de computadora pequeña (SCSI)
- Adjunto de tecnología avanzada paralela
- Unidades de estado sólido (SSD)

En promedio, las velocidades máximas de los discos actuales son:

- Disco SATA3 de 5,400 RPM, Lectura: 102 MB/s, Escritura: 96 MB/s
- Disco SATA3 de 7,200 RPM, Lectura: 272 MB/S, Escritura: 200 MB/s
- Disco SSD SATA, Lectura 550 MB/s, Escritura 520 MB/s
- Disco SSD NVMe, Lectura 6,600 MB/s, Escritura 5,500 MB/s
- Disco SSD PCI 5.0, Lectura 13,000 MB/s, Escritura 12,000 MB/s
- Unidad Flash USB[20] 2.0, 35 MB/s

Esta otra opción es una alternativa a las microondas, pero en HAMR (Heat-Assisted Magnetic Recording) el proceso de calentar el medio de almacenamiento lo realiza un láser. Toshiba prometió lanzar unidades de más de 32 TB en 2024, mientras que Seagate también quería ofrecer esa capacidad de forma inminente para luego dar el salto a unidades de 40 TB e incluso a los 100 TB que plantean para 2030. Ahí es donde probablemente entre en acción la evolución de HAMR+, que tratará de exprimir aún más la densidad de grabación.

[20]Los colores en los puertos USB son: USB 1.X blanco (12 Mbps), USB 2.X Negro (480 Mbps), USB 3.0 Azul oscuro (5 Gbps), USB 3.1 Azul claro (10 Gbps), USB 3.2 Rojo (20 Gbps). En el caso del USB de color Amarillo, esté es un puerto de carga aún con el dispositivo apagado.

- Unidad Flash USB 3.0 o 3.1 gen 1, 5 Gbit/s
- Unidad Flash USB 3.0 o 3.1 gen 2, 10 Gbit/s
- Unidad Flash USB 3.2 gen 2x2, 20 Gbit/s

**Disco de Estado Sólido SSD** Estos son los últimos avances en tecnología de almacenamiento que tenemos en la industria de las computadoras. Son totalmente diferentes de las otras unidades en que no consisten en partes móviles. Tampoco almacenan datos utilizando magnetismo. En su lugar, hacen uso de la tecnología de memoria flash, circuitos integrados o dispositivos semiconductores para almacenar datos de forma permanente, al menos hasta que se borren. Estas son algunas de sus ventajas.

- Acceso a datos más rápido
- Menos susceptible a los golpes
- Menores tiempos de acceso y latencia
- Menos consumo de energía

Los SSD actuales están disponibles tanto en versiones SATA como en versiones M.2, U.2 y en formato de tarjeta PCI Express 4.0. Los tres últimos hacen uso del protocolo NVMe y la interfaz PCI Express 4.0 x4, lo que les permite superar los 6,600 MB/s de velocidades de lectura y escritura, frente a los 550 MB/s que suelen alcanzar como máximo las unidades SATA. La nueva versión PCI 5.0 ofrece un ancho de banda de 32 GT/s el doble de PCI 4.0, permitiendo discos SSD con 13,000 MS/s de velocidad de lectura secuencial y realizar hasta 2,500K operaciones por segundo de lectura aleatoria y tamaño máximo 15.36 TB a un precio exorbitante.

**Tarjetas microSD** Cuando compramos una tarjeta microSD para ampliar el almacenamiento de nuestro Smartphone, cámara, tableta o cualquier otro dispositivo electrónico la mayoría de la gente normalmente solo se fija en la capacidad de almacenamiento. Ahora bien, ¿qué significan todas esas etiquetas y nombres que llevan adscritas las microSD? ¿Cuál es la diferencia entre una microSDXC y una microSDHC? ¿Es mejor una UHS-I o una UHS-II? ¿Qué quiere decir que una tarjeta es A1 y V30? A continuación, intentamos aclarar toda esta nomenclatura.

Lo primero que tenemos que tener claro es que cuando analizamos una tarjeta micro SD existen multitud de factores que limitan su velocidad y capacidad de almacenamiento. Su rendimiento depende de factores como el tipo de tarjeta que estamos usando, su clase, y otros detalles como el tipo de bus o el número de operaciones que puede realizar por segundo.

**Tipos de tarjetas microSD** actualmente existen 4 generaciones distintas de tarjetas de memoria microSD. Cuanto más modernas sean, mayores velocidades y almacenamiento podrán ofrecer:

- Tarjetas micro SD (Secure Digital): Estas son las memorias de primera generación. Las desarrolló el fabricante SanDisk y fueron las primeras en utilizar el formato de 15 x 11 x 1 milímetros. Su capacidad máxima es de 32 GB.
- Tarjetas micro SDHC (Secure Digital High Capacity): Tarjetas de segunda generación. Cuentan con un bus de datos mejorado que permite alcanzar velocidades superiores, aunque su capacidad máxima sigue siendo de 32 GB.
- Tarjetas micro SDXC (Secure Digital Extended Capacity): Estas micro SD utilizan un sistema de archivos exFAT y su velocidad de transferencia puede llegar hasta los 312 MB/s. Su capacidad de almacenamiento puede llegar hasta los 2 TB y es el tipo de tarjeta más común utilizado a día de hoy.
- Tarjetas micro SDUC: Estas son las tarjetas de memoria más modernas y punteras. Utilizan el sistema de archivos exFAT y permiten almacenar entre 2 TB y 128 TB de datos.

Evidentemente con esto no es suficiente. Si queremos tener una idea aproximada de la velocidad de la micro SD tendremos que fijarnos en aspectos como la clase y tipo de bus que emplea.

La clase es una característica que nos indica la velocidad de transferencia de datos mínima de la tarjeta de memoria. Actualmente hay 4 tipos de clase diferentes:

- Clase 2: Velocidad mínima de 2 MB/s
- Clase 4: Velocidad mínima de 4 MB/s

- Clase 6: Velocidad mínima de 6 MB/s
- Clase 10: Velocidad mínima de 10MB/s

Hoy en día la mayoría de tarjetas micro SD son de clase 10, ya que son capaces de transferir más de 10 MB/s y superan esa cifra fácilmente.

El bus determina la velocidad de la interfaz de la tarjeta de memoria, y nos puede servir como indicativo para conocer la rapidez con la que se pueden leer y escribir los datos:

- Bus estándar: Su velocidad de transferencia alcanza hasta los 12.5 MB/s. Es el tipo de bus utilizado en tarjetas de clase 2, 4 y 6.
- Bus de alta velocidad (High Speed): Se utiliza en tarjetas de clase 10 y alcanza una velocidad de hasta 25 MB/s.
- Bus Ultra High Speed (UHS): Estos son los buses con la interfaz más rápida, y existen varios tipos:
  - UHS-I: Hay dos tipos de buses UHS-I. Por un lado, tenemos el UHS-I clase 1 (U1) que alcanza velocidades de 50 MB/s. Y luego tenemos el UHS-I clase 3 (U3) que llega hasta los 104 MB/s.
  - UHS-II: Alcanza velocidades de transferencia hasta 312 MB/s.
  - UHS-III: Velocidades de transferencia de datos que alcanzan hasta los 624 MB/s.
- SD-Express: Este es el tipo de bus más potente de todos, llegando hasta los 985 MB/s.

Como referencia, te interesará saber que actualmente la mayoría de tarjetas micro SD de gama media utilizan un bus UHS-I de clase 3 (U3) con velocidades de lectura de hasta 104 MB/s.

Otro factor importante es la velocidad de lectura y escritura aleatoria (IOPS) u operaciones por segundo que puede realizar una tarjeta. Este dato determina el rendimiento mínimo en la lectura y escritura aleatoria de la SD:

- Clase de rendimiento de aplicación A1: Las tarjetas A1 tienen una velocidad mínima de lectura aleatoria de 1,500 IOPS, y una velocidad mínima de escritura aleatoria de 500 IOPS.

- Clase de rendimiento de aplicación A2: Las tarjetas A2 ofrecen velocidades superiores, con 4,000 IOPS de lectura y 2,000 IOPS de escritura.

Normalmente con una tarjeta A1 es más que suficiente para tareas del día a día, aunque si necesitamos un rendimiento superior, por ejemplo, para ejecutar aplicaciones desde la SD o jugar a videojuegos, las tarjetas A2 ofrecen un mejor rendimiento.

La velocidad de escritura aleatoria (A1 y A2) es un dato relevante para los Smartphones y tabletas, pero si tenemos una cámara de grabación, una Action camera o un Dron, la característica en la que nos tenemos que fijar es en el Velocidad de escritura secuencial (sistema V) que utiliza (en inglés, Video Speed Class). O dicho de otra forma, en su velocidad de escritura secuencial.

Esta característica nos indica la cantidad de datos que se pueden grabar en la micro SD de forma constante sin bajar de una velocidad mínima. Esto resulta esencial cuando queremos grabar vídeos en alta y ultra-alta definición:

- V30 (Video Speed Class 30): Velocidad de escritura mínima de 30 MB/s
- V60 (Video Speed Class 60): Velocidad de escritura mínima de 60 MB/s
- V90 (Video Speed Class 90): Velocidad de escritura mínima de 90 MB/s

Por ejemplo, si vamos a grabar vídeo en resolución 4K directamente en la tarjeta micro SD, es necesario que la velocidad V sea lo máximo posible, especialmente si vamos a utilizar un amplio BitRate con bajos niveles de compresión, o calidades superiores como el 8K.

**Cintas Magnéticas** En el año 2010, se comunicó que todos los datos utilizados para el proyecto del satélite Nimbus se recuperaron de cintas que en ese momento tenían 46 años. A partir de dicho comunicado se extendió el uso de la cinta magnética para almacenamiento de datos en todo el mundo.

En un mundo altamente digital, la cinta magnética es una de las pocas tecnologías que utiliza señales analógicas para mover parte de los datos, en su esencia, la cinta se parece mucho a un HDD, utiliza materiales de base magnética, pero en este caso, la cinta es literalmente una base de material

generalmente nailon que tiene un revestimiento magnético. Y en lugar de un disco giratorio, la cinta entra, está enhebrada. Puede parecer una cinta VHS, pero es mucho más robusta.

La cinta se mueve linealmente hacia la unidad de cinta y hacia el cartucho de cinta. Para escribir, el cabezal de la cinta toma señales electrónicas y crea un mini campo magnético que puede cambiar la polaridad del material de la película para formar un patrón de ceros y unos. Una vez que los datos se escriben en la cinta, no se pueden cambiar (pero se pueden borrar y reescribir).

La inmutabilidad y las capacidades de encriptación de la cinta, así como la simplicidad de crear un espacio para almacenarla en una bóveda hacen de la cinta un arma clave para asegurar que los datos sobrevivan frente al Ransomware. Uno de los grandes productores de cintas en la actualidad es IBM, los cuales argumentan que esta función hace que la cinta sea el medio ideal para almacenar datos de archivo a los que no es necesario acceder con frecuencia.

La cinta también puede servir como una copia de seguridad y de versiones fuera de línea de archivos importantes o confidenciales que son resistentes a los ataques cibernéticos. Los tipos de datos que permanecen en la cinta abarcan registros financieros, registros médicos, información de identificación personal y documentos que forman parte de una retención legal de múltiples gobiernos.

Una sola cinta mide aproximadamente 3 pulgadas por 3 pulgadas y 3/4 de pulgada de grosor. Es más pequeño que una unidad de disco duro (HDD), pero pesa alrededor de 0.6 kilogramos. Un cartucho puede almacenar 18 Terabytes de datos sin comprimir y 45 Terabytes comprimidos. IBM está trabajando para duplicar esta capacidad en la próxima generación de la tecnología. En cuanto a la velocidad de recuperación, se obtiene un flujo de datos de una unidad de cinta de 1,000 Megabits por segundo, comprimidos.

Una biblioteca de cintas puede variar en tamaño desde algo que puede poner en su escritorio hasta algo que es del tamaño de un refrigerador pequeño (alrededor de 8 pies cuadrados). La pequeña biblioteca del tamaño de un refrigerador tiene capacidad para 1584 cartuchos. IBM promociona que su biblioteca Diamondback será la biblioteca de cintas más densa del mercado. Podrá contener 69 Petabytes de información mientras ocupa menos de 8 pies cuadrados de espacio.

La cinta magnética supera al disco duro y al flash en cuanto a longevidad, costo financiero y costo de huella de carbono, pero pierde en velocidad de

acceso. Las cintas no son recomendables para poner datos de producción en vivo o incluso copias de seguridad, pero son perfectas para cualquier información a la que se acceda con poca frecuencia y que deba conservarse durante mucho tiempo, como registros médicos o datos de archivo.

Es conocido que muchas empresas han usado y seguirán usando cinta magnética en sus operaciones, entre las que destacan las hiperescalas (empresas que han crecido tanto que ofrecen sus propias infraestructuras o tienen datos masivos como resultado de su infraestructura) siempre necesitan muchas formas diferentes de tecnología para manejar la variedad de datos que ingresan a sus sistemas para alimentar una gama de servicios. Entre otras destacan: Bancos, Gobiernos, Milicia y organizaciones como CERN, así como corporaciones como Amazon, Google, Meta, Baidu, Alibaba y Tencent.

**Tarjeta Gráfica** La tarjeta gráfica o tarjeta de vídeo es un componente que viene integrado en la placa base de la computadora o se instala aparte para ampliar sus capacidades. Concretamente, esta tarjeta está dedicada al procesamiento de datos relacionados con el vídeo y las imágenes que se están reproduciendo en la computadora.

**Procesador Gráfico GPU** el corazón de la tarjeta gráfica es la GPU o Unidad de procesamiento gráfico, un circuito muy complejo que integra varios miles de millones de transistores diminutos y puede tener desde uno a miles de núcleos (ya es común encontrar computadoras personales con tarjeta de 10496 cores y 24 GB de GRAM) que tienen capacidad de procesamiento independiente. De la cantidad y capacidad de estos núcleos dependerá la potencia.

Así como los procesadores centrales de las CPU, están diseñados con pocos núcleos pero altas frecuencias de reloj, las GPU tienden al concepto opuesto, contando con grandes cantidades de núcleos con frecuencias de reloj relativamente bajas. Luego tienes la memoria gráfica de acceso aleatorio o GRAM, que son Chips de memoria que almacenan y transportan información entre sí. Esta memoria no es algo que vaya a determinar de forma importante el rendimiento máximo de una tarjeta gráfica, aunque si no es suficiente puede acabar lastrando y limitando la potencia de la CPU.

La idea de usar esa potencia para otros menesteres se denomina GPGPU (General Purpose Computation on Graphics Processing Units) o GPU Computing. En el momento en el que las tarjetas gráficas permiten que se pro-

gramen funciones sobre su Hardware se empieza a hacer uso de GPGPU. Al principio era necesario utilizar los lenguajes enfocados a la visualización en pantalla (como OpenGL) para realizar otros cálculos no relacionados con los gráficos. Esto implicaba el uso de funciones muy poco flexibles, originalmente diseñadas para otros fines, lo que hacía que la labor de programar para tal fin fuese realmente tediosa y complicada.

Para facilitar el empleo de las tarjetas gráficas para cualquier uso no vinculado con los gráficos, NVidia desarrolló toda una tecnología alrededor de la tarjeta, que permitía usar la misma para cualquier tarea: CUDA. ATI, la principal (y actualmente casi única) competidora, un poco más tarde haría lo propio lanzando su propia tecnología: Stream. En un principio las tarjetas gráficas solo trabajaban con aritmética de 32 bits, pero en la actualidad ya se cuenta con aritmética de 64 bit (para lograr esto, muchas tarjetas usan dos de sus cores de 32 bits para emular uno de 64 bits, reduciendo su número de cores útiles a la mitad).

**Procesador Gráfico Integrado** muchos procesadores CPU incorporan una o más GPU en su interior, llamada gráfica integrada (iGPU Integrated Graphics Processing Units o APU Accelerated Processing Unit). Generalmente es muy poco potente, pero lo suficiente para realizar tareas básicas como navegar por Internet, ver vídeos, e incluso para algunos juegos básicos, especialmente en las últimas generaciones puesto que cada vez son más potentes. No obstante, en las últimas generaciones de procesadores cada vez se están introduciendo gráficos integrados más potentes, y ya son capaces de manejar varios monitores, resoluciones 4K e incluso son capaces de mover algunos juegos a una tasa digna de FPS.

**Tarjeta Gráfica** por ejemplo, la tarjeta gráfica de AMD Instinct MI200 cuenta con más de 200,000 cores y 128 GB de HBM2, NVidia GEFORCE RTX 3090 proporciona 10,496 cores y 24 GB de GDDR6x, NVidia A100 cuenta con 80 GB de memoria HBM2 6192 cores y 432 núcleos tensor[21], mientras que la tarjeta NVidia Titan RTX proporciona 130 Tensor TFLOP de rendimiento, 576 núcleos tensores y 24 GB de memoria GDDR6.

[21] Un Tensor core (o núcleos Tensor) calculan la operación de una matriz 4x4 completa, la cual se calcula por reloj. Estos núcleos pueden multiplicar dos matrices FP16 4x4 y sumar la matriz FP32 al acumulador.

Por otra parte, Intel ha desarrollado una aceleradora gráfica Artic Sound-M pensada para centro de datos (especialmente diseñada para juegos en la nube) que utiliza una GPU DG2 Xe-HPG que viene con una configuración de 512 unidades de ejecución lo que equivale a 4,096 Shaders, por ejemplo, esta aceleradora puede manejar hasta 8 Streamings simultáneos de video 4K o más de 30 si el vídeo es en 1080p y cuenta con más de 60 funciones virtualizadas.

En agosto del 2021, se anunció la construcción de la supercomputadora Polaris, acelerado por 2240 GPU NVIDIA A100 Tensor Core, el sistema puede alcanzar casi 1.4 exaflops de rendimiento teórico de IA y aproximadamente 44 petaflops de rendimiento máximo de doble precisión. Polaris, que será construido por Hewlett Packard Enterprise, combinará simulación y aprendizaje automático al abordar cargas de trabajo informáticas de alto rendimiento de inteligencia artificial y con uso intensivo de datos, impulsadas por 560 nodos en total, cada uno con cuatro GPU NVIDIA A100.

**Tipos de Redes Según el Medio Físico** Si bien, nuestros dispositivos de cómputo pueden funcionar sin conexión de red, estos se ven inmediatamente limitados. La red nos permite conectarnos a Internet que es el camino por el cual nos conectamos con el mundo. Las formas de conectar nuestros equipos a Internet en un principio fue exclusivamente por red alámbrica, desde ya hace unos años a la fecha se dispone de conexión a red alámbrica e inalámbrica, pero actualmente nuestros dispositivos cuentan casi exclusivamente con conexión inalámbrica.

**¿Qué es el ancho de banda?** Se trata de la capacidad máxima y la cantidad de datos que se pueden transmitir a través de una conexión (de internet, por ejemplo), en un momento determinado. Algo que debemos tener claro es que el ancho de banda de red es fundamental para la calidad y velocidad de la conexión.

El ancho de banda se mide en *bit/s* o en sus múltiplos *k/bits* o *m/bits* por segundo. Y para la mayoría de los casos, debemos asegurarnos siempre de tener el mayor ancho de banda que nos sea posible, porque de esta manera podremos tener una mejor y más rápida transferencia de datos.

**¿Qué es entonces la velocidad de transmisión?** Este término se puede definir como la velocidad a la que se transmite la información. Cuando un usuario adquiere un paquete con una empresa prestadora de servicios de

internet, recibe, por ejemplo, 10 mbps, 30 mbps, 100 mbps, etc. Y esto se refiere a la cantidad de datos que podemos descargar o subir a la red. Como recomendación, lo indicado es que para que la velocidad pueda existir será necesario tener un ancho de banda igual o superior a la velocidad contratada en el paquete de servicio.

Normalmente vemos en los anuncios de todos los operadores, que estos ofrecen una cantidad cualquiera de megas de navegación; este valor numérico corresponde a la velocidad de descarga únicamente. Para encontrar la velocidad de subida, es necesario acceder a un test que nos revele cuál es el resultado y si lo que nos prometen, es verdad o no.

**¿Qué es la latencia?** Es el tiempo total que transcurre desde que enviamos una información, hasta que la misma llega a un receptor. Su valor de medición se hace en milisegundos, también se conoce como *Ping* y está presente en actividades que realiza cotidianamente como jugar en línea o hacer videollamadas.

**¿Altera la latencia la velocidad de la conexión?** La velocidad de conexión influye en la latencia una vez que pasamos de un rango predeterminado. Poniéndolo en un ejemplo, si tenemos una conexión a Internet de 1 Mbps y la comparamos con una conexión de 100 Mbps, dependiendo del tamaño del paquete se notará una mejora grande en la velocidad. Otros factores que importan a la hora de hablar de latencia son el estar conectado a internet por Wi-fi o un cable, si tiene servicio de fibra óptica o qué tanta distancia hay entre su ordenador y un Router, etc.

La latencia en la conexión también es la suma de otros retardos:

- De procesamiento: se define en el tiempo que tardan los Routers en examinar la cabecera y a su vez, la respuesta en determinar a dónde hay que enviar cualquier paquete haciendo una previa comprobación de sus tablas de enrutamiento.
- De cola: tiempo de espera del paquete para poder ser transmitido a través de un enlace físico. Cabe anotar que no podemos saber previamente si va a haber un retardo de cola o no, ya que este cambia en tiempo real.
- De transmisión: es el tiempo que tarda el paquete en arribar hasta el siguiente nodo o destino final.

- De propagación: es el tiempo que tarda un bit en propagarse desde un punto cualquiera de origen hasta llegar a uno de destino. Su velocidad depende del medio físico por el que se transporte.

El retardo total es la suma de todos los retardos anteriormente enunciados.

Comúnmente se asocia la latencia con la banda de ancha, pero existe una diferencia sustancial entre ambas. Si bien ambas afectan la velocidad de la conexión, la banda ancha permite que se pueda transmitir una gran cantidad de datos, mientras la latencia determina a qué velocidad se transmite esa cantidad de datos.

Si bien, tener red nos permite estar conectados, tenemos una gran limitación por las velocidades de conexión a las que tendremos acceso según el tipo de medio físico que usemos para conectarnos, así como el número de dispositivos con los que compartamos la conexión. Las redes inalámbricas parecen ser omnipresentes, pero las velocidades de interconexión dejan mucho que desear como veremos a continuación.

**Redes alámbricas** se comunica a través de cables de datos (generalmente basada en Ethernet). Los cables de datos, conocidos como cables de red de Ethernet o cables con hilos conductores (CAT5), conectan computadoras y otros dispositivos que forman las redes. Las redes alámbricas son mejores cuando se necesita mover grandes cantidades de datos a altas velocidades, como medios multimedia de calidad profesional.

Ventajas

- Costos relativamente bajos
- Ofrece el máximo rendimiento posible
- Mayor velocidad - cable de Ethernet estándar hasta 1 Gbps[22]
- Mayor rendimiento de Voz sobre IP.
- Mejores estándares Ethernet en la industria.
- Mayor capacidad de ancho de banda por cables.

[22] El término bps se refiere a transmitir Bits por segundo (se requieren 8 Bits para formar un Byte). Por eso el término de 1,000 Mbps es equivalente a 125 MB/s.

- Aplicaciones que utilizan un ancho de banda continúo.

Desventajas

- El costo de instalación siempre ha sido un problema común en este tipo de tecnología, ya que el estudio de instalación, canaletas, conectores, cables y otros suman costos muy elevados en algunas ocasiones.
- El acceso físico es uno de los problemas más comunes dentro de las redes alámbricas. Ya que para llegar a ciertos lugares, es muy complicado el paso de los cables a través de las paredes de concreto u otros obstáculos.
- Dificultad y expectativas de expansión es otro de los problemas más comunes, ya que cuando pensamos tener un número definido de nodos en una oficina, la mayoría del tiempo hay necesidades de construir uno nuevo y ya no tenemos espacio en los Switches instalados.

Hoy en día se puede hacer la siguiente clasificación de las redes de protocolo Ethernet para cable y fibra óptica[23]:

- Ethernet, que alcanza no más de 10 Mbps de velocidad
- Fast Ethernet, que puede trabajar con hasta 100 Mbps
- Gigabit Ethernet, alcanza hasta 1 Gbps (1000 Mbps aprox.)
- 2.5 y 5 Gigabit Ethernet hasta 2.5 Gbps si usamos cableado Cat 5a y nada menos que 5 Gbps con cableado Cat 6

[23] El récord mundial en mayo del 2022 de transmisión en fibra óptica es de 1.02 petabits por segundo enviados a través de 51.7 kilómetros (que podría transmitir hasta 10 millones de canales por segundo de vídeo a resolución 8K), que rompe al récord anterior de 319 terabits por segundo sobre una distancia de 1,800 millas (con el estimado de descarga de 80,000 películas simultáneamente en un segundo).

En 2023 los investigadores de Electronics and Computer Engineering de Aston University en U.K. alcanzaron una velocidad de 301 terabits por segundo (Tbps), equivalente a transferir 1,800 películas 4K a través de Internet en un segundo, utilizando cables de fibra óptica existentes. En comparación, la velocidad media de banda ancha fija en los EE. UU. es de 242,38 megabits por segundo (Mbps), según Speed Test.

Los resultados de la prueba, que se realizaron utilizando el tipo de cables de fibra ya tendidos en el suelo, lograron esta velocidad vertiginosa enviando luz infrarroja a través de hilos tubulares de vidrio, que es como funciona generalmente la banda ancha de fibra óptica. Pero aprovecharon una banda espectral que nunca se ha utilizado en sistemas comerciales, llamada "banda E", utilizando dispositivos nuevos hechos a medida.

- 10 Gigabit Ethernet, que puede alcanzar hasta los 10 GMbps
- 1000GbE para alcanzar la Ethernet Terabit (125 Gbytes)

La categoría del cable o el tipo de fibra óptica determina la velocidad máxima soportada por cada tipo de cable, pero aunque haya cables con la misma velocidad hay otros factores que determinan su usabilidad, como el ancho de banda o la frecuencia. La frecuencia o ancho de banda determina la potencia de la red, a mayor frecuencia mayor ancho de banda y menor perdida de datos. Este factor es importante si vamos a conectar varios equipos al mismo cable de red o vamos a hacer una gran tirada de cable, ya que cuanto más largo sea el cable de red más potencia perderá. Siempre tendrá más velocidad un cable corto que un cable largo, pero si el ancho de banda es amplio tardará más metros en perder potencia y velocidad.

Ethernet es el estándar que domina la gran mayoría de mercado, presente de manera casi exclusiva tanto en mercado doméstico, como en pequeña y mediana empresa representa también un porcentaje muy significativo en grandes centros de datos. Pero existen otros protocolos que se pueden situar en el mismo nivel de calidad que Gigabit Ethernet como InfiniBand.

InfiniBand es un bus serie bidireccional de comunicaciones de alta velocidad en las que las rápidas comunicaciones entre servidores son críticas para el rendimiento, llegando a ofrecer velocidades de hasta 2.0 Gbps netos en cada dirección del enlace en un nodo simple, 4 Gbps netos en un nodo doble y hasta 8 Gbps netos en un nodo quadruple. Estos nodos a su vez se pueden agrupar en grupos de 4 ó 12 enlaces llegando a velocidades de hasta 96 Gbps netos en un grupo de 12 nodos cuádruples. El factor de velocidad neta viene relacionado con que Infiniband de cada 10 bits que transmite 8 de ellos son datos, basándose en la codificación 8B/10B.

Recientemente se han implementado sistemas en los que ya no se utiliza esta codificación 8B/10B sino la 64B/66B que permite mejorar el porcentaje de datos útiles por trama enviada y que ha permitido los nodos FDR-10 (Fourteen Data Rate-10 a 10 Gbps), FDR (Fourteen Data Rate a 13.64 Gbps) y EDR (Enhanced Data Rate a 25 Gbps). Este último en un grupo de 12 nodos proporciona hasta 300 Gbps. Los últimos desarrollos de Gigabit Ethernet, proporcionan hasta 100 Gbps por puerto.

Estas enormes velocidades de conexión hacen que Infiniband sea una conexión con una muy importante presencia en superordenadores y clústers, por ejemplo del top 500 de superordenadores en 2020, 226 están conectados

internamente con Infiniband, 188 lo están con Gigabig Ethernet y el resto con Myrinet, Cray, Fat Tree u otras interconexiones a medida.

Una de las principales ventajas de Infiniband sobre Ethernet es su bajísima latencia, por ejemplo y basándonos en los datos del estudio de Qlogic "Introduction to Ethernet Latency, an explanation to Latency and Latency measurement", la latencia en 10 Gpbs Ethernet se sitúa en 5 microsegundos mientras que la de Infiniband se sitúa por debajo de los 3 microsegundos.

Los sistemas de conmutadores inteligentes InfiniBand de NVIDIA Mellanox ofrecen el mayor rendimiento y densidad de puertos para computación de alto rendimiento (HPC), IA, Web 2.0, Big Data, nubes y centros de datos empresariales. La compatibilidad con configuraciones de 36 a 800 puertos a hasta 200 Gpbs por puerto permite que los clústeres de cómputo y los centros de datos convergentes funcionen a cualquier escala, lo que reduce los costos operativos y la complejidad de la infraestructura.

**Redes inalámbricas:** es una red en la que dos o más terminales (ordenadores, tabletas, teléfonos inteligentes, etc.) se pueden comunicar sin la necesidad de una conexión por cable. Se basan en un enlace que utiliza ondas electromagnéticas (radio e infrarrojo) en lugar de cableado estándar. Permiten que los dispositivos remotos se conecten sin dificultad, ya se encuentren a unos metros de distancia como a varios kilómetros.

Asimismo, la instalación de estas redes no requiere de ningún cambio significativo en la infraestructura existente como pasa con las redes cableadas. Tampoco hay necesidad de agujerear las paredes para pasar cables ni de instalar porta cables o conectores. Esto ha hecho que el uso de esta tecnología se extienda con rapidez.

Tipos de redes inalámbricas

- LAN Inalámbrica: Red de área local inalámbrica. También puede ser una red de área metropolitana inalámbrica.
- GSM (Global System for Mobile Communications): la red GSM es utilizada mayormente por teléfonos celulares.
- D-AMPS (Digital Advanced Mobile Phone Service): está siendo reemplazada por el sistema GSM.
- Fixed Wireless Data: Es un tipo de red inalámbrica de datos que puede ser usada para conectar dos o más edificios juntos para extender o

compartir el ancho de banda de una red sin que exista cableado físico entre los edificios.

- Wi-Fi[24]: es uno de los sistemas más utilizados para la creación de redes inalámbricas en computadoras, permitiendo acceso a recursos remotos como internet e impresoras. Utiliza ondas de radio.

Ventajas

- La instalación de redes inalámbricas suele ser más económica.
- Su instalación también es más sencilla.
- Permiten gran alcance; las redes hogareñas inalámbricas suelen tener hasta 100 metros desde la base transmisora.
- Permite la conexión de gran cantidad de dispositivos móviles. En las redes cableadas mientras más dispositivos haya, más complicado será el entramado de cables.

[24] Notemos que una conexión de WiFi tradicional sin obstáculos con una intensidad de banda de 2.4 GHZ puede tener un alcance máximo de 46 metros y para la banda de 5.0 GHz un alcance de 15 metros. Paro hay muchos objetos que interfieren con la señal del Router del WiFi y el sitio en el que colocamos nuestro dispositivo inalámbrico como computadora o teléfono inteligente:

- Superficies y/o objetos de metal o vidrio blindado
- Refrigeradores, lavadoras y radiadores
- Hornos de microondas, cámaras Web, monitores de bebés y teléfonos inalámbricos
- Paredes y muros
- Dispositivos arquitectónicos que funciones como una jaula de Faraday (es un contenedor recubierto por materiales conductores de electricidad como mallas metálicas, papel aluminio, cajas o cestos de basura de acero que funciona como un blindaje contra los efectos de un campo eléctrico proveniente del exterior).

En caso de que varios dispositivos se conecten a la red, se puede optar por colocar el Router en un punto medio, para que ninguna zona quede sin cobertura. Por otra parte, para una mejor conexión, también procura que el Router se encuentre en una zona elevada, pues esto mejorará el alcance de la señal inalámbrica. Una excelente opción para mejorar la calidad del internet inalámbrico cuando hay obstáculos es utilizar un repetidor de WiFi, este dispositivo es muy útil para amplificar la señal y llevarla a más sitios de nuestra red.

- Posibilidad de conectar nodos a grandes distancias sin cableado, en el caso de las redes inalámbricas corporativas.
- Permiten más libertad en el movimiento de los nodos conectados, algo que puede convertirse en un verdadero problema en las redes cableadas.
- Permite crear una red en áreas complicadas donde, por ejemplo, resulta dificultoso o muy caro conectar cables.

Desventajas

- Calidad de Servicio: La velocidad que posee la red inalámbrica no supera la cableada, ya que esta puede llegar a los 10 Mbps, frente a 100 Mbps que puede alcanzar la cableada. Hay que tomar en cuenta la tasa de error debida a las interferencias.
- Costo: En algunos casos, puede ser más barato cablear una casa/oficina que colocar un servicio de red inalámbrica.
- La señal inalámbrica puede verse afectada e incluso interrumpida por objetos, árboles, paredes, espejos, entre otros.

La velocidad máxima de transmisión inalámbrica de la tecnología 802.11b es de 11 Mbps. Pero la velocidad típica es solo la mitad: entre 1.5 y 5 Mbps dependiendo de si se transmiten muchos archivos pequeños o unos pocos archivos grandes. La velocidad máxima de la tecnología 802.11g es de 54 Mbps. Pero la velocidad típica de esta última tecnología es solo unas 3 veces más rápida que la de 802.11b: entre 5 y 15 Mbps. Resumiendo, las velocidades típicas de los diferentes tipos de red son:

| Estándar | V.Máxima | V.Practica | Frecuencia | Ancho Banda | Alcance |
|---|---|---|---|---|---|
| 802.11 | 2Mbit/s | 1Mbit/s | 2.4Ghz | 22MHz | 330 metros |
| 802.11a(WiFi5) | 54Mbit/s | 22Mbit/s | 5.4Ghz | 20MHz | 390 metros |
| 802.11b | 11Mbit/s | 6Mbit/s | 2.4Ghz | 22MHz | 460 metros |
| 802.11g | 54Mbit/s | 22Mbit/s | 2.4Ghz | 20MHz | 460 metros |
| 802.11n | 600Mbit/s | 100Mbit/s | 2.4Ghzy5.4Ghz | 20 y 40MHz | 820 metros |
| 802.11ac | 6.93Gbps | 100Mbit/s | 5.4Ghz | 80 o hasta 160MHz | Poco alcance, pero sin interferencias |
| 802.11ad | 7.13Gbit/s | Hasta6Gbit/s | 60Ghz | 2MHz | 300 metros |
| 802.11ah | 35.6Mbps | 26.7Mbps | 0.9Ghz | 2MHz | 1000 metros |
| 802.11ax(WiFi6) | 9,6Gbps | 6,9Gbps | 2.4Ghz y 5.4Ghz | 20MHz | 1000 metros |

Como puedes ver, los principales factores que influyen en la calidad de una conexión WiFi, son la frecuencia, el ancho de banda y el alcance total. Considerando que, todo esto junto con la velocidad máxima y la velocidad práctica, se congregan en lo que es cada versión de este tipo de conexión a la red. Además, la conexión se degradará inexorablemente con la cantidad de dispositivos conectados y su consumo de datos.

**Velocidad de los Proveedores** cuando se contrata el servicio de internet, la velocidad de interconexión del mismo depende de cuánto sea el cobro, pero en la mayoría de los casos se tendrá una velocidad de descarga mayor a la velocidad de carga y nuestra red será una intranet (dirección de IP dinámica) compartida con otros miles de usuarios abonados al servicio de internet del proveedor. También es posible contratar una interconexión con velocidades homogéneas para carga y descarga dedicada, el costo del mismo se puede hasta triplicar con respecto a uno no homogéneo.

En caso de requerir una dirección de internet homologada o pública, el costo de contratar el servicio aumenta considerablemente, pero de esta forma nuestros equipos son visibles en el Internet (esto conlleva un aumento de riesgos al estar nuestros equipos más vulnerables a ataques informáticos).

**El Tráfico Global en Internet** El tráfico en Internet a nivel global sigue creciendo de manera vertiginosa. Sólo en 2024 aumentó un 17.2%, según los datos del informe de 2024 Cloudflare Radar Year in Review, publicado por la compañía por quinto año consecutivo, y en el que se repasa información sobre conectividad, seguridad, uso de dispositivos para acceso a la Red y frecuencia de los apagones, entre otras tendencias.

Según este informe, y para sorpresa de prácticamente nadie a estas alturas, los servicios de Internet más populares del mundo son Google, Facebook, Apple, TikTok y AWS. Chrome, con un 65.8% de los internautas como usuarios, es el navegador más utilizado en todo el mundo. WhatsApp sigue siendo la aplicación de mensajería más popular.

React, PHP y JQuery están entre las tecnologías para desarrollar webs más populares. HubSpot, Google y WordPress están entre los proveedores más populares de servicios y plataformas de soporte. Además, Go, ha superado a NodeJS como lenguaje más popular para hacer peticiones de API automatizadas.

Los rastreadores de IA son una gran fuente de tráfico, aunque despier-

tan cada vez más suspicacias por su actividad. Básicamente, se dedican a escanear la web para recopilar grandes cantidades de datos para entrenar modelos grandes de lenguaje. Preocupa bastante el hecho de que algunos recojan datos sin que tengan permiso para hacerlo, frente a los bots «buenos» y verificados, que suelen tener su origen en los motores de búsqueda y son transparentes sobre lo que son y lo que hacen. En este grupo entran GoogleBot, Qualys o BingBot.

Cloudflare, entre otras actividades, se dedica a rastrear el tráfico relacionado con la IA para determinar qué bots son los más agresivos, cuáles tienen el mayor volumen de peticiones y cuáles hacen rastreos de manera regular. Según los investigadores que han realizado el informe, el denominado facebookexternalhit es el que más tráfico de todos ha generado durante el año: un 27.16%. Le siguen Bytespider, de ByteDance, con un 23.35%; Amazonbot, con un 13.34%, ClaudeBot, de Anthropic, con un 8.06%; y GPTBot, con un 5.60%.

Al parecer, el tráfico generado por Bytespider, bot del propietario de TikTok, ha ido descendiendo de manera gradual durante 2024. En las semanas finales del año generaba entre un 80% y un 85% menos de tráfico que al principio. Mientras tanto el de ClaudeBot registró una subida muy pronunciada hacia mitad de 2024, y luego cayó otra vez de forma brusca.

La mayoría de las peticiones Web son todavía de HTTP2, que se lanzó en 2015, y cuenta aún con un 49.6% de las mismas. Un 29.9% todavía son de HTTP original, estandarizado en 1996, mientras que solo un 20.5% son de HTTP3, desplegado en 2022.

Cloudflare también registra, además del tráfico HTTP, las conexiones realizadas a través del protocolo de transmisión TCP, que asegura que hay una transferencia de datos fiable entre dispositivos de red. En 2024, un 20.7% de las conexiones TCP finalizaron de manera inesperada antes de que pudiesen terminar en un intercambio de datos útiles.

Estas anomalías en las conexiones TCP pueden deberse a varios motivos. Entre ellos a los ataques de denegación de servicio (DoS), al escaneado de redes, a las desconexiones de clientes, a la manipulación de las conexiones o a lo que Cloudflare ha llamado «comportamiento poco habitual del cliente».

La mayor parte de las interrupciones de una conexión TCP identificadas por Cloudflare se produjeron en 2024 después de que un servidor recibiese una petición de sincronización, pero antes de que recibiese un reconocimiento de la misma.

En cuanto a la seguridad, Cloudflare ha señalado que sólo un 4.3% de los

mensajes de correo electrónico enviados en 2024 eran maliciosos. En estos había sobre todo enlaces falsos (42.9%) o identidades falsas (35.1%). En el 70% de los casos se utilizaban los dos métodos.

Como datos curiosos en cuanto a seguridad, la compañía asegura que la vulnerabilidad Log4j todavía se usa como método de ataque, y se usa de manera mucho más activa que otras vulnerabilidades comunes. Además, prácticamente todos los mensajes de correo procesados por Cloudflare con dominios .bar, .rest y .uno eran o correos de Spam, o mensajes que, directamente, eran maliciosos.

En 2024, por otro lado, hubo 225 apagones de Internet de gran envergadura. La mayoría se dieron en África, Oriente Medio o India. Más de la mitad fueron apagones ordenados directamente por gobiernos, mientras que otros se debieron a otras causas.

Entre ellas, corte de cables, apagones eléctricos, problemas técnicos o de mantenimiento, episodios climáticos graves o ciberataques. Muchos duraron solo unas pocas horas, mientras que otros, como el experimentado en Bangladesh en julio, duró 10 días. Dentro de estos apagones está también el provocado por el incidente de CrowdStrike el pasado verano.

Otro de los aspectos tratados en el informe es la calidad de la conexión a Internet de los países, con base en su velocidad de subida, la de bajada y su latencia. España, con una velocidad de descarga media de 292.6 Mbps, y de 192.6 Mbps de subida, está a la cabeza en velocidad de conexión. Todos los países desarrollados presentan velocidades de descarga por encima de 200 Mbps de media.

Un 41.3% del tráfico de Internet a nivel mundial procede de dispositivos móviles, con el otro 58.7% producido en ordenadores portátiles y de sobremesa. Eso sí, en alrededor de un centenar de países del mundo, la mayoría del tráfico en 2024 procede de dispositivos móviles. Cuba y Siria tienen el mayor tráfico de dispositivos móviles, un 77%.

Otras áreas de alta demanda de tráfico de dispositivos de este tipo son Oriente Medio, África, Asia-Pacífico y América Central y del Sur. En este aspecto, las mediciones del tráfico son parecidas a las de 2023 y 2022.

**Información de mi Computadora** Si lo que deseamos es un listado detallado del Hardware de nuestro equipo de cómputo, entonces podemos usar cualquiera de estos comandos (que previamente deberemos instalar):

```
$ lscpu
```

```
# lshw
# dmidecode
# hwinfo
```

## 1.2 ¿Que es Computación Paralela?

En el sentido más simple, la computación paralela es el uso simultáneo de múltiples recursos computacionales para resolver un problema computacional:

- Un problema se divide en partes discretas que se pueden resolver simultáneamente
- Cada parte se descompone en una serie de instrucciones
- Las instrucciones de cada parte se ejecutan simultáneamente en diferentes procesadores
- Se emplea un mecanismo global de control/coordinación

**¿Por qué se hace programación paralela?** El hecho de que la programación paralela sea un paradigma da cuenta de que existe una razón por la cual no ha dejado de ser necesaria o no ha sido totalmente automatizable, igualmente hay otras razones interesantes detrás para entender la existencia, actualidad y contemporaneidad de la programación paralela:

- Ley de Moore: Esta ley propuesta por Gordon E. Moore en 1965 dice resumidamente que el número de transistores en un chip determinado se doblaría cada dos años. Esto quiere decir un aumento del rendimiento en los procesadores del alrededor del 50%, esto se traduce en escalar la velocidad de reloj de los procesadores, pero esta ley no es fidedigna desde el 2002 dónde solo ha habido un 20%, lo cuál sigue siendo un aumento considerable, sin embargo, no sería suficiente para que todos los avances en computación que se han logrado hasta el día y las necesidades de procesamiento en crecimiento exponencial puedan satisfacerse totalmente.
- Overclocking infinito: El Overclokcing tiene un límite a pesar de que existiera una refrigeración perpetúa y adecuada del procesador. Esto

es debido a las corrientes parásitas que impiden una velocidad teóricamente infinita a la cual los circuitos pueden cambiar entre estados, o de hecho sus transistores.

- Automatización del paralelismo: Se dice en este paradigma que el éxito es inversamente proporcional al número de cores precisamente porque existen complejidades en el corazón del paralelismo que implican cosas que todavía no se pueden predecir ni con inteligencia artificial, se menciona cuales son las posibles estrategias para atacar un problema de forma paralela, esto da cuenta de que existe una forma prácticamente determinada de abordarlos pero no de automatizarlos, a pesar de que sí existan algunas partes que son automatizables en el proceso.

- Solución en el Hardware: Un diseño adecuado del Hardware permitiría que la paralelización siempre estuviera presente con respecto a los procesadores que se están usando de tal modo que alguno los problemas que son inherentes al paradigma pudieran evitarse. Esto ha resultado imposible hasta la fecha, de hecho, solo diseñar solamente algo tan efectivo y tradicional como se ha hecho en programación secuencial es algo que no existe hasta ahora. Existen algunas aproximaciones como OpenMP y programación por hilos de las que hablaremos más adelante.

**Ventajas**

- Resuelve problemas que no se podrían realizar en una sola CPU
- Resuelve problemas que no se pueden resolver en un tiempo razonable
- Permite ejecutar problemas de un orden y complejidad mayor
- Permite ejecutar código de manera más rápida (aceleración)
- Permite ejecutar en general más problemas
- Obtención de resultados en menos tiempo
- Permite la ejecución de varias instrucciones en simultáneo
- Permite dividir una tarea en partes independientes

**Desventajas**

- Mayor consumo de energía
- Mayor dificultad a la hora de escribir programas
- Dificultad para lograr una buena sincronización y comunicación entre las tareas
- Retardos ocasionados por comunicación entre tareas
- Número de componentes usados es directamente proporcional a los fallos potenciales
- Condiciones de carrera
  - Múltiples procesos se encuentran en condición de carrera si el resultado de los mismos depende del orden de su llegada
  - Si los procesos que están en condición de carrera no son correctamente sincronizados, puede producirse una corrupción de datos

**Paralelismo de Grano Fino, Grano Grueso y Paralelismo Vergonzoso** las aplicaciones a menudo se clasifican según la frecuencia con que sus subtareas se sincronizan o comunican entre sí. Una aplicación muestra un paralelismo de grano fino si sus subtareas deben comunicarse muchas veces por segundo, se considera paralelismo de grano grueso si no se comunican muchas veces por segundo, y es vergonzosamente paralelo si nunca o casi nunca se tienen que comunicar. Aplicaciones vergonzosamente paralelas son consideradas las más fáciles de paralelizar.

El tipo de problemas complejos que aborda el cómputo de alto rendimiento (HPC por High-Performance Computing) no se pueden resolver con una computadora de escritorio y tienen una escala determinada: toma mucho tiempo resolverlos, se necesita una gran cantidad de memoria (RAM), se tienen que realizar muchísimos experimentos parecidos y hay restricciones concretas de tiempo para encontrar resultados. La mayoría de estas temáticas están relacionadas con la innovación en la industria y su aplicación en áreas como energía, medio ambiente, biotecnología, medicina, ingeniería, etc.

A su vez, la evolución del procesador la cantidad de transistores sigue creciendo pero la velocidad de los núcleos individuales (core) casi no aumenta. Podríamos comprar un procesador de 128 núcleos pero el desafío de

aprovechar la potencia de un procesador es cómo distribuimos el tiempo de ejecución de una misma tarea computacional.

De este modo, cada tarea computacional se divide en dos partes: serial y paralelizable. Como todas las aplicaciones tienen una parte secuencial, el tiempo de cómputo de la aplicación paralela está acotado por esa sección serial (Ley de Ahmdahl). Esto implica dos cuestiones fundamentales:

1. Es necesario reducir lo más posible el tiempo de ejecución serial
2. En una tarea computacional existe un máximo de procesadores que conviene poner a trabajar en forma paralela, más allá de ese punto no se justifica seguir paralelizando la tarea.

**¿Qué debe tener en cuenta un centro de HPC para aprovechar al máximo los recursos de cómputo?** Cada centro de HPC tiene que configurar, optimizar aplicaciones y sistemas operativos. Allí el monitoreo es clave, con la escala de las infraestructuras también surgen aplicaciones con problemas de memoria (cuello de botella) y la necesidad de utilizar mejores redes (PARAMNet-3, Infiniband, GigE). Tener poder de cómputo no es barato y solamente con el Hardware no alcanza. El equipo de operaciones de HPC necesita un alto grado de conocimiento y de disciplina para encarar la tarea.

Con el uso del cómputo de alto desempeño es posible por ejemplo reducir el tiempo que nos lleva descubrir nuevos medicamentos de años a meses. Una computadora de alto rendimiento es capaz de resolver este y otros tipos de problemas científicos avanzados mediante simulaciones, modelos y análisis. Estos sistemas abren las puertas de la "Cuarta Revolución Industrial", ya que ayudan a resolver muchas de las problemáticas más importantes del mundo. Los sistemas HPC ya se utilizan para las siguientes tareas:

- Descubrir nuevos componentes de drogas y probar los conocidos para combatir diferentes tipos de cáncer y otras enfermedades
- Simular dinámicas moleculares para crear nuevos materiales, como tejidos balísticos
- Pronosticar cambios climáticos considerables para mejorar la preparación de las comunidades afectadas

Las supercomputadoras representan lo último en sistemas HPC. La definición de supercomputadora depende de diferentes estándares que van cambiando a medida que las capacidades evolucionan. Un solo clúster de supercomputadoras puede incluir decenas de miles de procesadores, y los sistemas más caros y potentes del mundo pueden costar más de US$ 100 millones de dólares.

**Casos de Uso Emergentes** a medida que las tecnologías mejoraron, la computación de alto rendimiento comenzó a implementarse en una gama de capacidades más amplia. Ahora contamos con más potencia de procesamiento y memoria que nunca para solucionar problemas más complejos.

- Aprendizaje automático: como subconjunto de la inteligencia artificial (IA), el aprendizaje automático (AA) se refiere a un sistema que tiene la capacidad de aprender de forma activa por sí mismo, a diferencia de recibir, de forma pasiva, instrucciones para ejecutar. Los sistemas HPC pueden utilizarse en aplicaciones de AA altamente avanzadas donde se analizan grandes cantidades de datos, por ejemplo, investigación del cáncer para detectar el melanoma en imágenes.

- Análisis de grandes conjuntos de datos: se recurre a la comparación rápida y a la correlación de grandes conjuntos de datos para complementar investigaciones y resolver problemas académicos, científicos, financieros, comerciales, gubernamentales, de salud y de seguridad cibernética. Este trabajo requiere un rendimiento masivo y capacidades de cómputo de una potencia enorme. Con un estimado de 50 petabytes de datos al año generados por las misiones, la NASA se apoya en la supercomputación para analizar observaciones y realizar simulaciones con grandes conjuntos de información.

- Modelado avanzado y simulación: al no tener que realizar un montaje físico en las primeras etapas del proceso, el modelado avanzado y la simulación permiten que las empresas ahorren tiempo, materiales y costos de contratación de personal para lanzar sus productos al mercado con mayor rapidez. El modelado y la simulación en HPC se aplican en el descubrimiento y la prueba de fármacos, diseños automotrices y aeroespaciales, pronóstico de sistemas climáticos o meteorológicos, y aplicaciones energéticas.

**Cómo Funciona la Computación de Alto Rendimiento** Existen dos métodos principales para procesar la información en HPC:

- Procesamiento en serie: es el que realizan las unidades de procesamiento central (CPU). Cada núcleo de CPU, por lo general, realiza solo una tarea a la vez. Las CPU son fundamentales para ejecutar diferentes funciones, como sistemas operativos y aplicaciones básicas (por ej., procesamiento de textos, productividad en la oficina).

- Procesamiento en paralelo: es el que se puede realizar mediante varias CPU o unidades de procesamiento de gráficos (GPU). Las GPU, diseñadas originalmente para gráficos independientes, son capaces de realizar diferentes operaciones aritméticas por medio de una matriz de datos (como pixeles de pantalla) de forma simultánea. La capacidad para trabajar en varios planos de datos al mismo tiempo hace que las GPU sean la elección natural para el procesamiento en paralelo en tareas de aplicaciones de aprendizaje automático (AA), como el reconocimiento de objetos en videos.

Para superar los límites de la supercomputación, se necesitan diferentes arquitecturas de sistemas. Para que el procesamiento en paralelo pueda llevarse a cabo, la mayoría de los sistemas HPC integran varios procesadores y módulos de memoria a través de interconexiones con un ancho de banda enorme. Ciertos sistemas HPC combinan varias CPU y GPU, lo que se conoce como computación heterogénea.

La potencia de procesamiento de las computadoras se mide en unidades llamadas "FLOPS"[25] (operaciones de punto flotante por segundo). A principios de 2019, la supercomputadora más potente que existe alcanzó los 143,5 peta Flops ($143 \times 10^{15}$). Este tipo de supercomputadora se llama equipo de petaescala y puede realizar más de mil billones de FLOPS. Por su parte, una computadora de escritorio para juegos de alta gama es más de un millón de veces más lenta y llega apenas a los 200 gigaFLOPS ($1 \times 10^{9}$).

Gracias a los avances tanto en procesamiento como rendimiento, en el año 2020 fuimos testigos de un nuevo salto en la era de la supercomputación: la

[25] "FLOPS" describe una velocidad de procesamiento teórica: para hacer posible esa velocidad es necesario enviar datos a los procesadores de forma continua. Por lo tanto, el procesamiento de los datos se debe tener en cuenta en el diseño del sistema. La memoria del sistema, junto con las interconexiones que unen los nodos de procesamiento entre sí, impactan en la rapidez con la que los datos llegan a los procesadores.

exaescala, que es casi 1000 veces más rápida que la petaescala. Esto significa que un sistema de exaescala realiza $10^{18}$ (o mil millones por mil millones) operaciones por segundo. Necesitaríamos 5 millones de computadoras de escritorio para alcanzar el nivel de rendimiento de procesamiento en supercomputación de 1 exa Flops, suponiendo que cada computadora de escritorio es capaz de realizar 200 gigaFLOPS.

## 1.3 Agradecimientos

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

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

Herramientas
http://132.248.181.216/Herramientas/

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

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. Agradezco los comentarios de la Dra. Ursula Xiomara Iturrarán Viveros (Facultad de Ciencias, UNAM) y del Dr. Josue Tago Pacheco (Facultad de Ingeniería, UNAM).

# 2 El Cómputo en Paralelo

La computación paralela es el uso de múltiples recursos computacionales para resolver un problema. Se distingue de la computación secuencial en que varias operaciones pueden ocurrir simultáneamente. Los sistemas de cómputo con procesamiento en paralelo surgen de la necesidad de resolver problemas complejos en un tiempo razonable, utilizando las ventajas de memoria, velocidad de los procesadores, formas de interconexión de estos y distribución de la tarea, a los que en su conjunto denominamos arquitectura en paralelo. Entenderemos por una arquitectura en paralelo a un conjunto de procesadores interconectados capaces de cooperar en la solución de un problema.

Así, para resolver un problema en particular, se usa una arquitectura o combinación de múltiples arquitecturas (topologías), ya que cada una ofrece ventajas y desventajas que tienen que ser sopesadas antes de implementar la solución del problema en una arquitectura en particular. También es necesario conocer los problemas a los que se enfrenta un programador que desea que su programa corra en paralelo, como son: el partir eficientemente[26] un problema en múltiples subtareas y cómo distribuir eficazmente[27] estas según la arquitectura en particular con que se trabaje. La eficacia difiere de la eficiencia en el sentido de que la eficiencia hace referencia en la mejor utilización de los recursos computacionales (procesadores), en tanto que la eficacia hace referencia en la capacidad para alcanzar un objetivo, aunque en el proceso no se haya hecho el mejor uso de los recursos computacionales.

El cómputo de alto rendimiento (HPC por High-Performance Computing) está formado por un conjunto de computadoras unidas entre sí en forma de Cluster[28], para aumentar su potencia de trabajo y rendimiento. En 2019 la supercomputadora *SUMMIT* de IBM funcionaban a más de 148 peta-Flops[29] (cada uno de ellos equivale a la realización de más de 1000 billones

[26] Es un indicador de la utilización efectiva de los diferentes recursos computacionales (principalmente del uso de los procesadores) en relación al algoritmo dado. Se entiende que la eficacia se da cuando se utilizan menos recursos para lograr un mismo objetivo. O al contrario, cuando se logran más objetivos con los mismos o menos recursos.

[27] La podemos definir como el nivel de consecución de metas y objetivos. La eficacia hace referencia a nuestra capacidad de lograr lo que no proponemos.

[28] 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

[29] "FLOPS" operaciones de punto flotante por segundo. Describe una velocidad de procesamiento teórica: para hacer posible esa velocidad es necesario enviar datos a los

de operaciones por segundo de las pruebas de rendimiento del HPC-AI), mientras que en junio de 2020 *FUGAKU* de Japón superaba los 415.5 peta-Flops y en noviembre del mismo año alcanzó los 2.0 exa-Flops, siendo este el primer equipo en alcanzar valores por arriba de un exa-Flops para cualquier precisión sobre cualquier tipo de Hardware, manteniéndose en el primer lugar del top 500 durante el 2021

El clásico uso del paralelismo, es el de diseño de programas eficientes en el ámbito científico. La simulación de problemas científicos es un área de gran importancia, los cuales requieren de una gran capacidad de procesamiento y de espacio de memoria, debido a las complejas operaciones que se deben realizar.

Otro uso clásico es el de las gráficas y vídeos de simulaciones generadas por computadora. La generación de fotogramas y videos requiere de una gran cantidad de cálculos matemáticos. Esto supone una tarea muy compleja para un solo procesador, luego es necesario que haya algún tipo de paralelismo, para distribuir la tarea para que esta sea realizada eficiente y eficazmente.

Tradicionalmente, los programas informáticos se han escrito para el cómputo en secuencial. Para resolver un problema, se construye un algoritmo y se implementa como un flujo en serie de instrucciones. Estas instrucciones se ejecutan en una unidad central de procesamiento en un ordenador. Sólo puede ejecutarse una instrucción a la vez y un tiempo después de que la instrucción ha terminado, se ejecuta la siguiente.

Actualmente, es una práctica común usar directivas de compilación en equipos paralelos sobre programas escritos de forma secuencial, con la esperanza que sean puestos por el compilador como programas paralelos. Esto en la gran mayoría de los casos genera códigos poco eficientes, pese a que corren en equipos paralelos y pueden usar toda la memoria compartida de dichos equipos, el algoritmo ejecutado continúa siendo secuencial en una gran parte del código.

La computación en paralelo, por el contrario, utiliza simultáneamente múltiples elementos de procesamiento para resolver un problema. Esto se logra mediante la división del problema en partes independientes de modo que cada elemento de procesamiento pueda ejecutar su parte del algoritmo de manera simultánea con los otros. Los elementos de procesamiento son di-

procesadores de forma continua. Por lo tanto, el procesamiento de los datos se debe tener en cuenta en el diseño del sistema. La memoria del sistema, junto con las interconexiones que unen los nodos de procesamiento entre sí, impactan en la rapidez con la que los datos llegan a los procesadores.

versos e incluyen recursos tales como una computadora con múltiples procesadores, varios ordenadores en red, Hardware especializado, o cualquier combinación de los anteriores.

Los programas informáticos paralelos son más difíciles de escribir que los secuenciales, porque la concurrencia introduce nuevos tipos de errores de Software, siendo las condiciones de sincronización las más comunes. La comunicación y sincronización entre diferentes subtareas son algunos de los mayores obstáculos para obtener un buen rendimiento del programa paralelo.

**¿Cómo Paralelizo mi programa?** El problema de todos los usuarios de Ciencias e Ingeniería que hacen uso de métodos numéricos y su implementación computacional es: ¿cómo paralelizo mi programa?, esta pregunta no tiene una respuesta simple, depende de muchos factores, por ejemplo: en que lenguaje o paquete está desarrollado el programa, el algoritmo usado para solucionar el problema, a que equipos paralelo tengo acceso y un largo etc. Algunas respuestas ingenuas podrían ser:

- Si uso lenguajes de programación compilables, puedo conseguir un compilador que permita usar directivas de compilación en equipos de memoria compartida sobre programas escritos de forma secuencial, con la esperanza que sean puestos por el compilador como programas paralelos haciendo uso de hilos, OpenMP y optimización del código generado.

- Si uso paquetes como MatLab, Julia o Python, es posible conseguir una versión del paquete que implemente bibliotecas que usen CUDAs, OpenMP o MPI para muchos de los algoritmos más usados en la programación.

- Si mi problema cabe en un sólo equipo, entonces puedo usar dos o más cores para resolver mi problema usando memoria compartida (puedo usar OpenMP, trabajar con hilos o usando paquetes como MatLab o lenguajes como Python, Julia, Fortran, C y C++, etc.), pero sólo puedo escalar para usar el máximo número de cores de un equipo (que en la actualidad puede ser del orden de 128 cores, hasta Terabytes de RAM y con almacenamiento de algunas decenas de Terabytes).

- Si mi problema cabe en la GRAM de una GPU, entonces el posible usar una tarjeta gráfica (usando paquetes como MatLab o lenguajes como

Python, Julia, Fortran, C y C++, etc.), pero sólo puedo escalar a la capacidad de dichas tarjetas gráficas.

- Puedo usar un cluster que usa memoria distribuida-compartida en conjunto con tarjetas gráficas, este tipo de programación requiere una nueva expertes y generalmente implica el uso de paso de mensajes como MPI y rediseño de los algoritmos usados en nuestro programa.

Notemos primero, que no todos los algoritmos son paralelizables. En cualquier caso se tienen que ver los pros y contras de la paralelización para cada caso particular. Pero es importante destacar que existen una gran cantidad de bibliotecas y paquetes que ya paralelizan ciertos algoritmos que son ampliamente usados como la solución de sistemas lineales[30] y no lineales. Además, el tiempo de programación necesario para desarrollar una aplicación paralela eficiente y eficaz para la gran mayoría de los programadores puede ser de semanas o meses en el mejor de los casos. Por ello, es necesario hacer un balance entre las diferentes opciones de paralelización para no invertir un tiempo precioso que puede no justificar dicha inversión económica, de recursos computacionales y sobre todo de tiempo.

Antes de iniciar con los tópicos del cómputo en paralelo, es necesario conocer cómo está constituida nuestra herramienta de trabajo: la computadora. Los conocimientos básicos de arquitectura de las computadoras nos permite identificar los componentes de Hardware que limitarán nuestros esfuerzos por realizar una paralelización óptima de nuestro programa y en muchos casos descubriremos de la peor manera que los algoritmos usados en nuestro programa no son los más adecuados para paralelizar; y descubriremos con horror que en algunos casos los tiempos de ejecución no mejoran sin importar cuantos equipos adicionales usemos, en el peor de los casos, el tiempo de ejecución se incrementa al aumentar equipos en vez de disminuir.

[30] Algunas de las bibliotecas más usadas para resolver sistemas lineales usando matrices bandadas y dispersar son PETCs, HYPRE, ATLAS, LAPACK++, LAPACK, EISPACK, LINPACK, BLAS, entre muchas otras alternativas, tanto para implementaciones secuenciales como paralelas y más recientemente para hacer uso de los procesadores CUDA en las GPU de nVidia.

## 2.1 ¿Que es Computación Paralela?

En el sentido más simple, la computación paralela es el uso simultáneo de múltiples recursos computacionales para resolver un problema computacional:

- Un problema se divide en partes discretas que se pueden resolver simultáneamente
- Cada parte se descompone en una serie de instrucciones
- Las instrucciones de cada parte se ejecutan simultáneamente en diferentes procesadores
- Se emplea un mecanismo global de control/coordinación

**¿Por qué se hace programación paralela?** El hecho de que la programación paralela sea un paradigma da cuenta de que existe una razón por la cual no ha dejado de ser necesaria o no ha sido totalmente automatizable, igualmente hay otras razones interesantes detrás para entender la existencia, actualidad y contemporaneidad de la programación paralela:

- Ley de Moore: Esta ley propuesta por Gordon E. Moore en 1965 dice resumidamente que el número de transistores en un Chip determinado se doblaría cada dos años. Esto quiere decir un aumento del rendimiento en los procesadores del alrededor del 50%, esto se traduce en escalar la velocidad de reloj de los procesadores, pero esta ley no es fidedigna desde el 2002 dónde solo ha habido un 20%, lo cuál sigue siendo un aumento considerable, sin embargo, no sería suficiente para que todos los avances en computación que se han logrado hasta el día y las necesidades de procesamiento en crecimiento exponencial puedan satisfacerse totalmente.
- Overclocking infinito: El Overclokcing tiene un límite a pesar de que existiera una refrigeración perpetúa y adecuada del procesador. Esto es debido a las corrientes parásitas que impiden una velocidad teóricamente infinita a la cual los circuitos pueden cambiar entre estados, o de hecho sus transistores.

- Automatización del paralelismo: Se dice en este paradigma que el éxito es inversamente proporcional al número de cores precisamente porque existen complejidades en el corazón del paralelismo que implican cosas que todavía no se pueden predecir ni con inteligencia artificial, se menciona cuales son las posibles estrategias para atacar un problema de forma paralela, esto da cuenta de que existe una forma prácticamente determinada de abordarlos pero no de automatizarlos, a pesar de que sí existan algunas partes que son automatizables en el proceso.

- Solución en el Hardware: Un diseño adecuado del Hardware permitiría que la paralelización siempre estuviera presente con respecto a los procesadores que se están usando de tal modo que alguno los problemas que son inherentes al paradigma pudieran evitarse. Esto ha resultado imposible hasta la fecha, de hecho, solo diseñar solamente algo tan efectivo y tradicional como se ha hecho en programación secuencial es algo que no existe hasta ahora. Existen algunas aproximaciones como OpenMP y programación por hilos de las que hablaremos más adelante.

**Ventajas**

- Resuelve problemas que no se podrían realizar en una sola CPU
- Resuelve problemas que no se pueden resolver en un tiempo razonable
- Permite ejecutar problemas de un orden y complejidad mayor
- Permite ejecutar código de manera más rápida (aceleración)
- Permite ejecutar en general más problemas
- Obtención de resultados en menos tiempo
- Permite la ejecución de varias instrucciones en simultáneo
- Permite dividir una tarea en partes independientes

**Desventajas**

- Mayor consumo de energía
- Mayor dificultad a la hora de escribir programas

- Dificultad para lograr una buena sincronización y comunicación entre las tareas
- Retardos ocasionados por comunicación entre tareas
- Número de componentes usados es directamente proporcional a los fallos potenciales
- Condiciones de carrera
  - Múltiples procesos se encuentran en condición de carrera si el resultado de los mismos depende del orden de su llegada
  - Si los procesos que están en condición de carrera no son correctamente sincronizados, puede producirse una corrupción de datos

**Paralelismo de Grano Fino, Grano Grueso y Paralelismo Vergonzoso** las aplicaciones a menudo se clasifican según la frecuencia con que sus subtareas se sincronizan o comunican entre sí. Una aplicación muestra un paralelismo de grano fino si sus subtareas deben comunicarse muchas veces por segundo, se considera paralelismo de grano grueso si no se comunican muchas veces por segundo, y es vergonzosamente paralelo si nunca o casi nunca se tienen que comunicar. Aplicaciones vergonzosamente paralelas son consideradas las más fáciles de paralelizar.

El tipo de problemas complejos que aborda el cómputo de alto rendimiento (HPC por High-Performance Computing) no se pueden resolver con una computadora de escritorio y tienen una escala determinada: toma mucho tiempo resolverlos, se necesita una gran cantidad de memoria (RAM), se tienen que realizar muchísimos experimentos parecidos y hay restricciones concretas de tiempo para encontrar resultados. La mayoría de estas temáticas están relacionadas con la innovación en la industria y su aplicación en áreas como energía, medio ambiente, biotecnología, medicina, ingeniería, etc.

A su vez, la evolución del procesador la cantidad de transistores sigue creciendo pero la velocidad de los núcleos individuales (core) casi no aumenta. Podríamos comprar un procesador de 128 núcleos pero el desafío de aprovechar la potencia de un procesador es cómo distribuimos el tiempo de ejecución de una misma tarea computacional.

De este modo, cada tarea computacional se divide en dos partes: serial y paralelizable. Como todas las aplicaciones tienen una parte secuencial, el

tiempo de cómputo de la aplicación paralela está acotado por esa sección serial (Ley de Ahmdahl). Esto implica dos cuestiones fundamentales:

1. Es necesario reducir lo más posible el tiempo de ejecución serial
2. En una tarea computacional existe un máximo de procesadores que conviene poner a trabajar en forma paralela, más allá de ese punto no se justifica seguir paralelizando la tarea.

**¿Qué debe tener en cuenta un centro de HPC para aprovechar al máximo los recursos de cómputo?** Cada centro de HPC tiene que configurar, optimizar aplicaciones y sistemas operativos. Allí el monitoreo es clave, con la escala de las infraestructuras también surgen aplicaciones con problemas de memoria (cuello de botella) y la necesidad de utilizar mejores redes (PARAMNet-3, Infiniband, GigE). Tener poder de cómputo no es barato y solamente con el Hardware no alcanza. El equipo de operaciones de HPC necesita un alto grado de conocimiento y de disciplina para encarar la tarea.

Con el uso del cómputo de alto desempeño es posible por ejemplo reducir el tiempo que nos lleva descubrir nuevos medicamentos de años a meses. Una computadora de alto rendimiento es capaz de resolver este y otros tipos de problemas científicos avanzados mediante simulaciones, modelos y análisis. Estos sistemas abren las puertas de la "Cuarta Revolución Industrial", ya que ayudan a resolver muchas de las problemáticas más importantes del mundo. Los sistemas HPC ya se utilizan para las siguientes tareas:

- Descubrir nuevos componentes de drogas y probar los conocidos para combatir diferentes tipos de cáncer y otras enfermedades
- Simular dinámicas moleculares para crear nuevos materiales, como tejidos balísticos
- Pronosticar cambios climáticos considerables para mejorar la preparación de las comunidades afectadas

Las supercomputadoras representan lo último en sistemas HPC. La definición de supercomputadora depende de diferentes estándares que van cambiando a medida que las capacidades evolucionan. Un solo clúster de supercomputadoras puede incluir decenas de miles de procesadores, y los sistemas más caros y potentes del mundo pueden costar más de US$ 100 millones de dólares.

**Casos de Uso Emergentes** a medida que las tecnologías mejoraron, la computación de alto rendimiento comenzó a implementarse en una gama de capacidades más amplia. Ahora contamos con más potencia de procesamiento y memoria que nunca para solucionar problemas más complejos.

- Aprendizaje automático: como subconjunto de la inteligencia artificial (IA), el aprendizaje automático (AA) se refiere a un sistema que tiene la capacidad de aprender de forma activa por sí mismo, a diferencia de recibir, de forma pasiva, instrucciones para ejecutar. Los sistemas HPC pueden utilizarse en aplicaciones de AA altamente avanzadas donde se analizan grandes cantidades de datos, por ejemplo, investigación del cáncer para detectar el melanoma en imágenes.

- Análisis de grandes conjuntos de datos: se recurre a la comparación rápida y a la correlación de grandes conjuntos de datos para complementar investigaciones y resolver problemas académicos, científicos, financieros, comerciales, gubernamentales, de salud y de seguridad cibernética. Este trabajo requiere un rendimiento masivo y capacidades de cómputo de una potencia enorme. Con un estimado de 50 petabytes de datos al año generados por las misiones, la NASA se apoya en la supercomputación para analizar observaciones y realizar simulaciones con grandes conjuntos de información.

- Modelado avanzado y simulación: al no tener que realizar un montaje físico en las primeras etapas del proceso, el modelado avanzado y la simulación permiten que las empresas ahorren tiempo, materiales y costos de contratación de personal para lanzar sus productos al mercado con mayor rapidez. El modelado y la simulación en HPC se aplican en el descubrimiento y la prueba de fármacos, diseños automotrices y aeroespaciales, pronóstico de sistemas climáticos o meteorológicos, y aplicaciones energéticas.

**Cómo Funciona la Computación de Alto Rendimiento** Existen dos métodos principales para procesar la información en HPC:

- Procesamiento en serie: es el que realizan las unidades de procesamiento central (CPU). Cada núcleo de CPU, por lo general, realiza solo una tarea a la vez. Las CPU son fundamentales para ejecutar diferentes

funciones, como sistemas operativos y aplicaciones básicas (por ej., procesamiento de textos, productividad en la oficina).

- Procesamiento en paralelo: es el que se puede realizar mediante varias CPU o unidades de procesamiento de gráficos (GPU). Las GPU, diseñadas originalmente para gráficos independientes, son capaces de realizar diferentes operaciones aritméticas por medio de una matriz de datos (como píxeles de pantalla) de forma simultánea. La capacidad para trabajar en varios planos de datos al mismo tiempo hace que las GPU sean la elección natural para el procesamiento en paralelo en tareas de aplicaciones de aprendizaje automático (AA), como el reconocimiento de objetos en videos.

Para superar los límites de la supercomputación, se necesitan diferentes arquitecturas de sistemas. Para que el procesamiento en paralelo pueda llevarse a cabo, la mayoría de los sistemas HPC integran varios procesadores y módulos de memoria a través de interconexiones con un ancho de banda enorme. Ciertos sistemas HPC combinan varias CPU y GPU, lo que se conoce como computación heterogénea.

La potencia de procesamiento de las computadoras se mide en unidades llamadas "FLOPS"[31] (operaciones de punto flotante por segundo). A principios de 2019, la supercomputadora más potente que existe alcanzó los 143,5 peta-Flops ($143 \times 10^{15}$). Este tipo de supercomputadora se llama equipo de petaescala y puede realizar más de mil billones de FLOPS. Por su parte, una computadora de escritorio para juegos de alta gama es más de un millón de veces más lenta y llega apenas a los 200 giga-FLOPS ($1 \times 10^{9}$).

Gracias a los avances tanto en procesamiento como rendimiento, en el año 2020 fuimos testigos de un nuevo salto en la era de la supercomputación: la exaescala, que es casi 1000 veces más rápida que la petaescala. Esto significa que un sistema de exaescala realiza $10^{18}$ (o mil millones por mil millones) operaciones por segundo. Necesitaríamos 5 millones de computadoras de escritorio para alcanzar el nivel de rendimiento de procesamiento en supercomputación de 1 exa-Flops, suponiendo que cada computadora de escritorio es capaz de realizar 200 giga-FLOPS.

[31] "FLOPS" describe una velocidad de procesamiento teórica: para hacer posible esa velocidad es necesario enviar datos a los procesadores de forma continua. Por lo tanto, el procesamiento de los datos se debe tener en cuenta en el diseño del sistema. La memoria del sistema, junto con las interconexiones que unen los nodos de procesamiento entre sí, impactan en la rapidez con la que los datos llegan a los procesadores.

**Arquitecturas de Software y Hardware** En las dos siguientes secciones se explican en detalle las dos clasificaciones de computadoras más conocidas en la actualidad. La primera clasificación, es la clasificación clásica de Flynn en dónde se tienen en cuenta sistemas con uno o varios procesadores, la segunda clasificación es moderna en la que sólo se tienen en cuenta los sistemas con más de un procesador.

El objetivo es presentar de una forma clara los tipos de clasificación que existen en la actualidad desde el punto de vista de distintos autores, así como cuáles son las ventajas e inconvenientes que cada uno ostenta, ya que es común que al resolver un problema particular se usen una o más arquitecturas de Hardware interconectadas generalmente por red.

## 2.2 Clasificación Clásica de Flynn

La clasificación clásica de arquitecturas de computadoras que hace alusión a sistemas con uno o varios procesadores fue realizada por Michael J. Flynn y la publicó por primera vez en 1966 y por segunda vez en 1970.

Esta taxonomía se basa en el flujo que siguen los datos dentro de la máquina y de las instrucciones sobre esos datos. Se define como flujo de instrucciones al conjunto de instrucciones secuenciales que son ejecutadas por un único procesador y como flujo de datos al flujo secuencial de datos requeridos por el flujo de instrucciones. Con estas consideraciones, Flynn clasifica los sistemas en cuatro categorías:

**Single Instruction stream, Single Data stream (SISD)** Los sistemas Monoprocesador de este tipo se caracterizan por tener un único flujo de instrucciones sobre un único flujo de datos, es decir, se ejecuta una instrucción detrás de otra. Este es el concepto de arquitectura serie de Von Neumann donde, en cualquier momento, sólo se ejecuta una única instrucción, un ejemplo de estos sistemas son las máquinas secuenciales convencionales.

**Single Instruction stream, Multiple Data stream (SIMD)** Estos sistemas de procesador Matricial tienen un único flujo de instrucciones que operan sobre múltiples flujos de datos. Ejemplos de estos sistemas los tenemos en las máquinas vectoriales con Hardware escalar y vectorial.

El procesamiento es síncrono, la ejecución de las instrucciones sigue siendo secuencial como en el caso anterior, todos los elementos realizan una misma

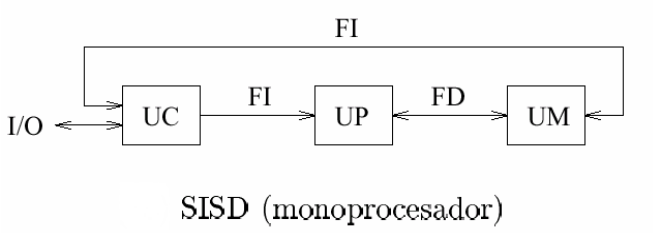



Figura 1: Ejemplo de máquina SISD

instrucción pero sobre una gran cantidad de datos. Por este motivo existirá concurrencia de operación, es decir, esta clasificación es el origen de la máquina paralela.

El funcionamiento de este tipo de sistemas es el siguiente. La unidad de control manda una misma instrucción a todas las unidades de proceso (ALUs). Las unidades de proceso operan sobre datos diferentes pero con la misma instrucción recibida.

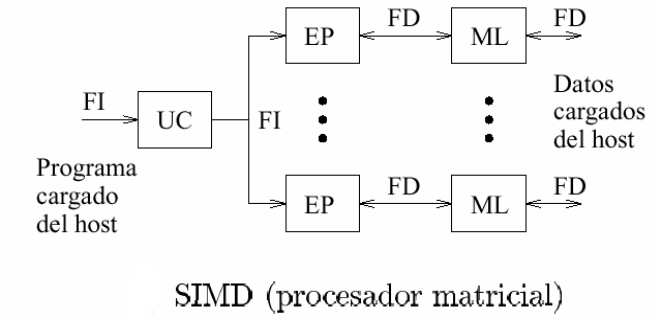



Figura 2: Ejemplo de máquina SIMD

Existen dos alternativas distintas que aparecen después de realizarse esta clasificación:

- Arquitectura Vectorial con segmentación, una CPU única particionada en unidades funcionales independientes trabajando sobre flujos de datos concretos.

- Arquitectura Matricial (matriz de procesadores), varias ALUs idénticas a las que el procesador da instrucciones, asigna una única instrucción pero trabajando sobre diferentes partes del programa.

**Multiple Instruction stream, Single Data stream (MISD)** Sistemas con múltiples instrucciones Array Sistólico que operan sobre un único flujo de datos. Este tipo de sistemas no ha tenido implementación hasta hace poco tiempo. Los sistemas MISD se contemplan de dos maneras distintas:

- Varias instrucciones operando simultáneamente sobre un único dato.
- Varias instrucciones operando sobre un dato que se va convirtiendo en un resultado que será la entrada para la siguiente etapa. Se trabaja de forma segmentada, todas las unidades de proceso pueden trabajar de forma concurrente.

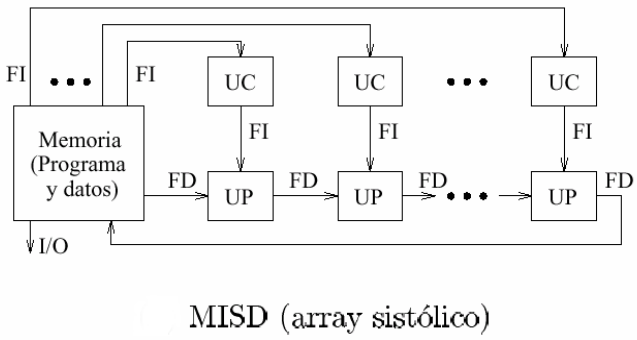



Figura 3: Ejemplo de máquina MISD

**Multiple Instruction stream, Multiple Data stream (MIMD)** Sistemas con un flujo de múltiples instrucciones Multiprocesador que operan sobre múltiples datos. Estos sistemas empezaron a utilizarse antes de la década de los 80s. Son sistemas con memoria compartida que permiten ejecutar varios procesos simultáneamente (sistema multiprocesador).

Cuando las unidades de proceso reciben datos de una memoria no compartida estos sistemas reciben el nombre de MULTIPLE SISD (MSISD). En arquitecturas con varias unidades de control (MISD Y MIMD), existe otro

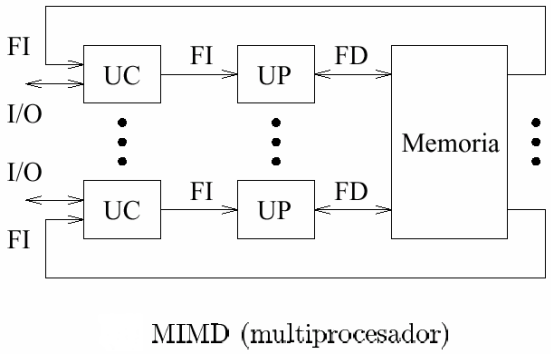



Figura 4: Ejemplo de máquina MIMD

nivel superior con una unidad de control que se encarga de controlar todas las unidades de control del sistema -ejemplo de estos sistemas son las máquinas paralelas actuales-.

## 2.3 Categorías de Computadoras Paralelas

Clasificación moderna que hace alusión única y exclusivamente a los sistemas que tienen más de un procesador (i.e máquinas paralelas). Existen dos tipos de sistemas teniendo en cuenta su acoplamiento:

- Los sistemas fuertemente acoplados son aquellos en los que los procesadores dependen unos de otros.
- Los sistemas débilmente acoplados son aquellos en los que existe poca interacción entre los diferentes procesadores que forman el sistema.

Atendiendo a esta y a otras características, la clasificación moderna divide a los sistemas en dos tipos: Sistemas multiprocesador (fuertemente acoplados) y sistemas multicomputadoras (débilmente acoplados).

### 2.3.1 Equipo Paralelo de Memoria Compartida

Un multiprocesador puede verse como una computadora paralela compuesta por varios procesadores interconectados que comparten un mismo sistema de memoria.

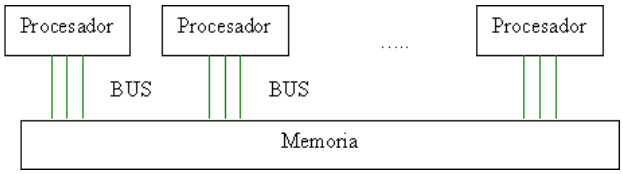



Figura 5: Arquitectura de una computadora paralela con memoria compartida

Los sistemas multiprocesadores son arquitecturas MIMD con memoria compartida. Tienen un único espacio de direcciones para todos los procesadores y los mecanismos de comunicación se basan en el paso de mensajes desde el punto de vista del programador.

Dado que los multiprocesadores comparten diferentes módulos de memoria, pueden acceder a un mismo módulo varios procesadores, a los multiprocesadores también se les llama sistemas de memoria compartida.

Para hacer uso de la memoria compartida por más de un procesador, se requiere hacer uso de técnicas de semáforos que mantienen la integridad de la memoria; esta arquitectura no puede crecer mucho en el número de procesadores interconectados por la saturación rápida del bus o del medio de interconexión.

Dependiendo de la forma en que los procesadores comparten la memoria, se clasifican en sistemas multiprocesador UMA, NUMA, COMA y Pipeline, que explicamos a continuación:

**Uniform Memory Access (UMA)** Sistema multiprocesador con acceso uniforme a memoria. La memoria física es uniformemente compartida por todos los procesadores, esto quiere decir que todos los procesadores tienen el mismo tiempo de acceso a todas las palabras de la memoria. Cada procesador tiene su propia Caché privada y también se comparten los periféricos.

Los multiprocesadores son sistemas fuertemente acoplados (tightly-coupled), dado el alto grado de compartición de los recursos (Hardware o Software) y el alto nivel de interacción entre procesadores, lo que hace que un procesador depende de lo que hace otro. El sistema de interconexión debe ser rápido y puede ser de uno de los siguientes tipos: bus común, red Crossbar[32] y red

[32]Red reconfigurable que permite la conexión de cada entrada con cualquiera de las salidas, es decir, permite cualquier permutación.

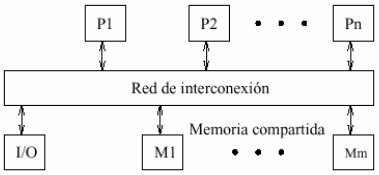



Figura 6: Acceso Uniforme a la memoria UMA

Multietapa. Este modelo es conveniente para aplicaciones de propósito general y de tiempo compartido por varios usuarios, existen dos categorías de sistemas UMA.

- Sistema Simétrico: cuando todos los procesadores tienen el mismo tiempo de acceso a todos los componentes del sistema (incluidos los periféricos), reciben el nombre de sistemas multiprocesador simétrico. Los procesadores tienen el mismo dominio (prioridad) sobre los periféricos y cada procesador tiene la misma capacidad para procesar.

- Sistema Asimétrico: son sistemas con procesador principal y procesadores subordinados, en donde sólo el primero puede ejecutar aplicaciones y dónde el tiempo de acceso para diferentes procesadores no es el mismo. Los procesadores subordinados (Attached) ejecutan código usuario bajo la supervisión del principal, por lo tanto cuando una aplicación es ejecutada en un procesador principal dispondrá de una cierta prioridad.

**Non Uniform Memory Access (NUMA)** Un sistema multiprocesador NUMA es un sistema de memoria compartida donde el tiempo de acceso varía según donde se encuentre localizado el acceso.

El acceso a memoria, por tanto, no es uniforme para diferentes procesadores, existen memorias locales asociadas a cada procesador y estos pueden acceder a datos de su memoria local de una manera más rápida que a las memorias de otros procesadores, debido a que primero debe aceptarse dicho acceso por el procesador del que depende el módulo de memoria local.

Todas las memorias locales conforman la memoria global compartida y físicamente distribuida y accesible por todos los procesadores.

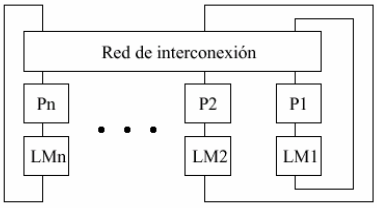



Figura 7: Acceso no uniforme a la memoria NUMA

**Cache Only Memory Access (COMA)** Los sistemas COMA son un caso especial de los sistemas NUMA. Este tipo de sistemas no ha tenido mucha trascendencia, al igual que los sistemas SIMD.

Las memorias distribuidas son memorias Cachés, por este motivo es un sistema muy restringido en cuanto a la capacidad de memoria global. No hay jerarquía de memoria en cada módulo procesador. Todas las Cachés forman un mismo espacio global de direcciones. El acceso a las Cachés remotas se realiza a través de los directorios distribuidos de las Cachés.

Dependiendo de la red de interconexión utilizada, se pueden utilizar jerarquías en los directorios para ayudar a la localización de copias de bloques de Caché.

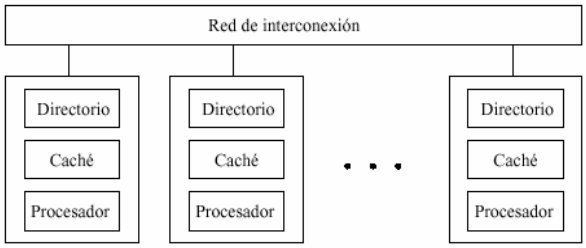



Figura 8: Ejemplo de COMA

**Procesador Vectorial Pipeline** En la actualidad es común encontrar en un solo procesador los denominados Pipeline o Procesador Vectorial Pipeline del tipo MISD. En estos procesadores los vectores fluyen a través de las unidades aritméticas Pipeline.

Las unidades constan de una cascada de etapas de procesamiento compuestas de circuitos que efectúan operaciones aritméticas o lógicas sobre el flujo de datos que pasan a través de ellas, las etapas están separadas por registros de alta velocidad usados para guardar resultados intermedios. Así la información que fluye entre las etapas adyacentes está bajo el control de un reloj que se aplica a todos los registros simultáneamente.

### 2.3.2 Equipo Paralelo de Memoria Distribuida

Los sistemas multicomputadoras (los más comunes son los Clusters) se pueden ver como una computadora paralela en el cual cada procesador tiene su propia memoria local. En estos sistemas la memoria se encuentra distribuida y no compartida como en los sistemas multiprocesador. Los procesadores se comunican a través de paso de mensajes, ya que éstos sólo tienen acceso directo a su memoria local y no a las memorias del resto de los procesadores.

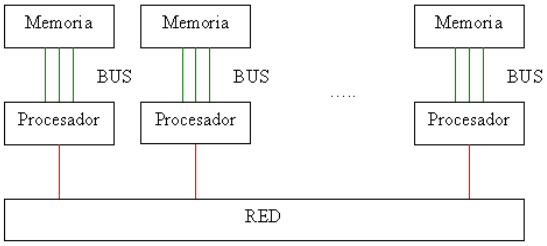



Figura 9: Arquitectura de una computadora paralela con memoria distribuida

La transferencia de los datos se realiza a través de la red de interconexión que conecta un subconjunto de procesadores con otro subconjunto. La transferencia de unos procesadores a otros se realiza por múltiples transferencias entre procesadores conectados dependiendo del establecimiento de dicha red.

Dado que la memoria está distribuida entre los diferentes elementos de proceso, estos sistemas reciben el nombre de distribuidos. Por otra parte, estos sistemas son débilmente acoplados, ya que los módulos funcionan de forma casi independiente unos de otros. Este tipo de memoria distribuida es de acceso lento por ser peticiones a través de la red, pero es una forma muy efectiva de tener acceso a un gran volumen de memoria.

### 2.3.3 Equipo Paralelo de Memoria Compartida-Distribuida

La tendencia actual en las máquinas paralelas es de aprovechar las facilidades de programación que ofrecen los ambientes de memoria compartida y la escalabilidad de los ambientes de memoria distribuida. En este modelo se conectan entre sí módulos de multiprocesadores, pero se mantiene la visión global de la memoria a pesar de que es distribuida.

**Clusters** El desarrollo de sistemas operativos y compiladores del dominio público (Linux y Software GNU), estándares para interfaz de paso de mensajes (Message Passing Interface MPI), conexión universal a periféricos (Peripheral Component Interconnect PCI), entre otros, han hecho posible tomar ventaja de los recursos económicos computacionales de producción masiva (procesadores, discos, redes).

La principal desventaja que presenta a los proveedores de multicomputadoras es que deben satisfacer una amplia gama de usuarios, es decir, deben ser generales. Esto aumenta los costos de diseños y producción de equipos, así como los costos de desarrollo de Software que va con ellos: sistema operativo, compiladores y aplicaciones. Todos estos costos deben ser añadidos cuando se hace una venta. Por supuesto alguien que sólo necesita procesadores y un mecanismo de pase de mensajes no debería pagar por todos estos añadidos que nunca usará. Estos usuarios son los que están impulsando el uso de Clusters principalmente de computadoras personales (PC), cuya arquitectura se muestra a continuación:

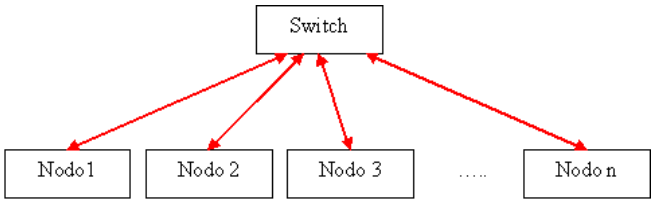



Figura 10: Arquitectura de un cluster

Los Cluster[33] se pueden clasificar en dos tipos según sus características físicas:

[33] 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

- Cluster homogéneo: si todos los procesadores y/o nodos participantes en el equipo paralelo son iguales en capacidad de cómputo -es permitido variar la cantidad de memoria o disco duro en cada procesador-

- Cluster heterogéneo: es aquel en que al menos uno de los procesadores y/o nodos participantes en el equipo paralelo son de distinta capacidad de cómputo

El Clustering no presenta dependencias a nivel de Hardware (no todos los equipos necesitan el mismo Hardware) ni a nivel de Software (no necesitan el mismo sistema operativo). Este tipo de sistemas disponen de una interfaz que permite dirigir el comportamiento de los Clusters. Dicha interfaz es la encargada de la interacción con usuarios y procesos, realizando la división de la carga entre los diversos servidores que compongan el Cluster.

Los Clusters pueden formarse de diversos equipos; los más comunes son los de computadoras personales, pero es creciente el uso de computadoras multiprocesador de más de un procesador de memoria compartida interconectados por red con los demás nodos del mismo tipo, incluso el uso de computadoras multiprocesador de procesadores vectoriales Pipeline. Los Clusters armados con la configuración anterior tienen grandes ventajas para procesamiento paralelo:

- La reciente explosión en redes implica que la mayoría de los componentes necesarios para construir un Cluster son vendidos en altos volúmenes y por lo tanto son económicos. Ahorros adicionales se pueden obtener debido a que sólo se necesitará una tarjeta de vídeo, un monitor y un teclado por Cluster. El mercado de los multiprocesadores es más reducido y más costoso

- El reemplazar un componente defectuoso en un Cluster es relativamente trivial comparado con hacerlo en un multiprocesador, permitiendo una mayor disponibilidad de Clusters cuidadosamente diseñados

Desventajas del uso de Clusters de computadoras personales para procesamiento paralelo:

- Con raras excepciones, los equipos de redes generales producidos masivamente no están diseñados para procesamiento paralelo

y típicamente su latencia es alta y los anchos de banda pequeños comparados con multiprocesadores. Dado que los Clusters explotan tecnología que sea económica, los enlaces en el sistema no son veloces implicando que la comunicación entre componentes debe pasar por un proceso de protocolos de negociación lentos, incrementando seriamente la latencia. En muchos y en el mejor de los casos (debido a costos) se recurre a una red tipo Fast Ethernet restringimiento la escalabilidad del Cluster

- Hay poco soporte de Software para manejar un Cluster como un sistema integrado

- Los procesadores no son tan eficientes como los procesadores usados en los multiprocesadores para manejar múltiples usuarios y/o procesos. Esto hace que el rendimiento de los Clusters se degrade con relativamente pocos usuarios y/o procesos

- Muchas aplicaciones importantes disponibles en multiprocesadores y optimizadas para ciertas arquitecturas, no lo están en Clusters

Sin lugar a duda los Clusters presentan una alternativa importante para varios problemas particulares, no sólo por su economía, si no también porque pueden ser diseñados y ajustados para ciertas aplicaciones. Las aplicaciones que pueden sacar provecho de Clusters son en donde el grado de comunicación entre procesos es de bajo a medio.

**Tipos de Cluster** Básicamente existen tres tipo de Clusters, cada uno de ellos ofrece ventajas y desventajas, el tipo más adecuado para el cómputo científico es del de alto-rendimiento, pero existen aplicaciones científicas que pueden usar más de un tipo al mismo tiempo.

- Alta disponibilidad (Fail-over o High-Availability): este tipo de Cluster está diseñado para mantener uno o varios servicios disponibles incluso a costa de rendimiento, ya que su función principal es que el servicio jamás tenga interrupciones como por ejemplo un servicio de bases de datos de transacciones bancarias

- Alto rendimiento (HPC o High Performance Computing): este tipo de Cluster esta diseñado para obtener el máximo rendimiento de la aplicación utilizada incluso a costa de la disponibilidad del

sistema, es decir el Cluster puede sufrir caídas, este tipo de configuración está orientada a procesos que requieran mucha capacidad de cálculo.

• Balanceo de Carga (Load-Balancing): este tipo de Cluster está diseñado para balancear la carga de trabajo entre varios servidores, lo que permite tener, por ejemplo, un servicio de cálculo intensivo multiusuarios que detecte tiempos muertos del proceso de un usuario para ejecutar en dichos tiempos procesos de otros usuarios.

**Grids** Son cúmulos (grupo de Clusters) de arquitecturas en paralelo interconectados por red, los cuales distribuyen tareas entre los Clusters que lo forman, estos pueden ser homogéneos o heterogéneos en cuanto a los nodos componentes del cúmulo. Este tipo de arquitecturas trata de distribuir cargas de trabajo acorde a las características internas de cada Cluster y las necesidades propias de cada problema, esto se hace a dos niveles, una en la parte de programación en conjunto con el balance de cargas y otra en la parte de Hardware que tiene que ver con las características de cada arquitectura que conforman el cúmulo.

**Balance de Carga** A la hora de diseñar un sistema paralelo con compartición de recursos, es necesario considerar cómo balancear la carga de trabajo. Se entiende este concepto, como la técnica usada para dividir el trabajo a compartir entre varios procesos, equipos de cómputo, u otros recursos. Está muy relacionada con los sistemas multiproceso, que trabajan o pueden trabajar con más de una unidad para llevar a cabo su funcionalidad.

Para evitar los cuellos de botella, el balance de la carga de trabajo se reparte de forma equitativa a través de un algoritmo que estudia las peticiones del sistema y las redirecciona a la mejor opción.

**Balance de Carga por Hardware** presenta las siguientes características:

- A partir de un algoritmo de planificación de procesos -Round Robin, LRU-, examina las peticiones entrantes y selecciona el más apropiado entre los distintos elementos del sistema

- La selección del siguiente libre del sistema está basada en el algoritmo de sustitución y es aleatoria
- La sesión debe de ser mantenida por el desarrollador
- Al ser un proceso de Hardware, es muy rápido

**Balance de Carga por Software** presenta las siguientes características:

- Examinan la solicitud para garantizar que se puede satisfacer la demanda de los usuarios
- Distintas peticiones del mismo usuario son servidas simultáneamente o secuencialmente según se definan
- Más lentos que los balanceadores de Hardware
- Normalmente son soluciones baratas

**Escalabilidad** Se entiende por escalabilidad a la capacidad de adaptación y respuesta de un sistema con respecto al rendimiento del mismo a medida que aumentan de forma significativa la carga computacional del mismo. Aunque parezca un concepto claro, la escalabilidad de un sistema es un aspecto complejo e importante del diseño de sistemas paralelos.

La escalabilidad está íntimamente ligada al diseño de sistemas paralelos, influye en el rendimiento de forma significativa. Si una aplicación está bien diseñada, la escalabilidad no constituye un problema. Analizando la escalabilidad, se deduce de la implementación y del diseño general del sistema. No es atributo del sistema configurable.

La escalabilidad supone un factor crítico en el crecimiento de un sistema paralelo. Si un sistema tiene como objetivo crecer la carga computacional -en el número de usuarios o procesos- manteniendo su rendimiento actual, tiene que evaluar dos posibles opciones:

- Con un Hardware de mayor potencia o
- Con una mejor combinación de Hardware y Software

Se pueden distinguir dos tipos de escalabilidad, vertical y horizontal:

- El escalar verticalmente o escalar hacia arriba, significa el añadir más recursos a un solo nodo en particular dentro de un sistema, tal como el añadir memoria o un disco duro más rápido a una computadora.
- La escalabilidad horizontal, significa agregar más nodos a un sistema, tal como añadir una computadora nueva a un programa de aplicación para espejo.

**Escalabilidad Vertical** El escalar hacia arriba de un sistema viene a significar una migración de todo el sistema a un nuevo Hardware que es más potente y eficaz que el actual. Una vez se ha configurado el sistema futuro, se realizan una serie de validaciones y copias de seguridad y se pone en funcionamiento. Las aplicaciones que estén funcionando bajo la arquitectura Hardware antigua no sufren con la migración, el impacto en el código es mínimo.

Este modelo de escalabilidad vertical tiene un aspecto negativo. Al aumentar la potencia en base a ampliaciones de Hardware, llegará un momento que existirá algún tipo de limitación de Hardware. Además a medida que se invierte en Hardware de muy altas prestaciones, los costos se disparan tanto de forma temporal -ya que si se ha llegado al umbral máximo, hay componentes de Hardware que tardan mucho tiempo en ampliar su potencia de forma significativa- como económicos. Sin embargo a nivel estructural no supone ninguna modificación reseñable, lo que la convierte en una buena opción si los costos anteriores son asumibles.

**Escalabilidad Horizontal** La escalabilidad horizontal consiste en potenciar el rendimiento del sistema paralelo desde un aspecto de mejora global, a diferencia de aumentar la potencia de una única parte del mismo. Este tipo de escalabilidad se basa en la modularidad de su funcionalidad, por ello suele estar conformado por una agrupación de equipos que dan soporte a la funcionalidad completa. Normalmente, en una escalabilidad horizontal se añaden equipos para dar más potencia a la red de trabajo.

Con un entorno de este tipo, es lógico pensar que la potencia de procesamiento es directamente proporcional al número de equipos en la red. El total de la potencia de procesamiento es la suma de la velocidad física de cada equipo transferida por la partición de aplicaciones y datos extendida a través de los nodos.

Si se aplica un modelo de escalabilidad basado en la horizontalidad, no existen limitaciones de crecimiento a priori. Como principal defecto, este modelo de escalabilidad supone una gran modificación en el diseño, lo que conlleva a una gran trabajo de diseño y reimplantación. Si la lógica se ha concebido para un único servidor, es probable que se tenga que estructurar el modelo arquitectónico para soportar este modelo de escalabilidad.

El encargado de cómo realizar el modelo de partición de datos en los diferentes equipos es el desarrollador. Existen dependencias en el acceso a la aplicación. Es conveniente, realizar una análisis de actividad de los usuarios para ir ajustando el funcionamiento del sistema. Con este modelo de escalabilidad, se dispone de un sistema al que se pueden agregar recursos de manera casi infinita y adaptable al crecimiento de cargas de trabajo y nuevos usuarios.

La escalabilidad cuenta como factor crítico en el crecimiento de usuarios. Es mucho más sencillo diseñar un sistema con un número constante de usuarios -por muy alto que sea este- que diseñar un sistema con un número creciente y variable de usuarios. El crecimiento relativo de los números es mucho más importante que los números absolutos.

### 2.3.4 Cómputo Paralelo en Multihilos

En una computadora, sea secuencial o paralela, para aprovechar las capacidades crecientes del procesador, el sistema operativo divide su tiempo de procesamiento entre los distintos procesos, de forma tal que para poder ejecutar a un proceso, el Kernel les asigna a cada uno una prioridad y con ello una fracción del tiempo total de procesamiento, de forma tal que se pueda atender a todos y cada uno de los procesos de manera eficiente.

En particular, en la programación en paralelo usando MPI, cada proceso -que eventualmente puede estar en distinto procesador- se lanza como una copia del programa con datos privados y un identificador del proceso único, de tal forma que cada proceso sólo puede compartir datos con otro proceso mediante paso de mensajes.

Esta forma de lanzar procesos por cada tarea que se desee hacer en paralelo es costosa, por llevar cada una de ellas toda una gama de subprocesos para poderle asignar recursos por parte del sistema operativo. Una forma más eficiente de hacerlo es que un proceso pueda generar bloques de subprocesos que puedan ser ejecutados como parte del proceso (como subtareas), así en el tiempo asignado se pueden atender a más de un subproceso

de manera más eficiente, esto es conocido como programación multihilos.

Los hilos realizarán las distintas tareas necesarias en un proceso. Para hacer que los procesos funcionen de esta manera, se utilizan distintas técnicas que le indican al Kernel cuales son las partes del proceso que pueden ejecutarse simultáneamente y el procesador asignará una fracción de tiempo exclusivo al hilo del tiempo total asignado al proceso.

Los datos pertenecientes al proceso pasan a ser compartidos por los subprocesos lanzados en cada hilo y mediante una técnica de semáforos el Kernel mantiene la integridad de estos. Esta técnica de programación puede ser muy eficiente si no se abusa de este recurso, permitiendo un nivel más de paralelización en cada procesador. Esta forma de paralelización no es exclusiva de equipos multiprocesadores o multicomputadoras, ya que pueden ser implementados a nivel de sistema operativo.

### 2.3.5 Cómputo Paralelo en CUDA

Son las siglas de arquitectura unificada de dispositivos de cómputo (Compute Unified Device Architecture CUDA) que hace referencia a una plataforma de computación en paralelo (que incluye su propia RAM conocida como GRAM para hacer óptima la solicitud de dicho recurso por sus múltiples cores) incluyendo un compilador y un conjunto de herramientas de desarrollo que permiten a los programadores usar una variación del lenguaje de programación C -Por medio de Wrappers se puede usar MatLab, Python, Julia, Fortran y Java en vez de C/C++- para codificar algoritmos en las unidades de procesamiento de gráficos (Graphics Processing Unit GPU).

CUDA intenta explotar las ventajas de las GPU (de decenas a centenas de cores[34]) frente a las CPU (decenas de cores[35]) de propósito general utilizando el paralelismo que ofrecen sus múltiples cores o núcleos, que permiten el lan-

[34]La GRAM de las tarjetas CUDA es relativamente pequeña (a lo más de algunas decenas de Gigabytes), pero está diseñada para que pueda ser compartida por sus cores de forma óptima. Pero si es necesario que múltiples cores de la GPU soliciten datos a la RAM del equipo de cómputo, pueden ocasionar degradación en la eficiencia de la GPU porque la RAM tiene una arquitectura DDR4 o DDR5 (lo que significa que solo puede atender peticiones de datos de sólo 4 o 5 cores simultáneamente).

[35]Los actuales CPUs pueden tener decenas de cores y comparten hasta Terabyates de RAM. Pero la arquitectura actual de la RAM es de tipo DDR4 o DDR5, esto significa, que sólo hay 4 o 5 canales para satisfacer las solicitudes de petición de datos de/hacia los cores, si más cores necesita acceso a la RAM tendrán que esperar que los otros cores concluyan su uso, mermando la eficiencia de la CPU.

zamiento de un altísimo número de hilos simultáneos. Si una aplicación está bien diseñada utilizando numerosos hilos que realizan tareas independientes usando la memoria compartida de la tarjeta (que es lo que hacen las GPU al procesar gráficos, su tarea natural) hará un uso eficiente de este dispositivo.

Por ejemplo, la tarjeta gráfica de NVidia GEFORCE RTX 3090 proporciona 10,496 cores y 24 GB de GDDR6x, mientras que la tarjeta NVidia Titan RTX proporciona 130 Tensor[36] TFLOP de rendimiento, 576 núcleos tensores y 24 GB de memoria GDDR6.

Ahora, miles de desarrolladores, científicos e investigadores están encontrando innumerables aplicaciones prácticas para esta tecnología en campos como el procesamiento de vídeo e imágenes, la biología y la química computacional, la simulación de la dinámica de fluidos, la reconstrucción de imágenes de Tomografía Axial Computarizada TAC, el análisis sísmico o el trazado de rayos, entre otros.

**Procesamiento paralelo con CUDA** Los sistemas informáticos están pasando de realizar el «procesamiento central» en la CPU a realizar «coprocesamiento» repartido entre la CPU y la GPU. Para posibilitar este nuevo paradigma computacional, NVIDIA ha inventado la arquitectura de cálculo paralelo CUDA, que ahora se incluye en las GPUs GeForce, ION Quadro y Tesla GPUs, lo cual representa una base instalada considerable para los desarrolladores de aplicaciones.

CUDA ha sido recibida con entusiasmo por la comunidad científica. Por ejemplo, se está utilizando para acelerar AMBER, un simulador de dinámica molecular empleado por más de 60,000 investigadores del ámbito académico y farmacéutico de todo el mundo para acelerar el descubrimiento de nuevos medicamentos. En el mercado financiero, Numerix y CompatibL introdujeron soporte de CUDA para una nueva aplicación de cálculo de riesgo de contraparte y, como resultado, se ha multiplicado por 18 la velocidad de la aplicación. Cerca de 400 instituciones financieras utilizan Numerix en la actualidad.

Un buen indicador de la excelente acogida de CUDA es la rápida adopción de la GPU Tesla para aplicaciones de GPU Computing. En la actualidad hay más de 700 Clusters de GPUs instalados en compañías publicadas en Fortune

[36] Un Tensor core (o núcleos Tensor) calculan la operación de una matriz 4x4 completa, la cual se calcula por reloj. Estos núcleos pueden multiplicar dos matrices FP16 4x4 y sumar la matriz FP32 al acumulador.

500 de todo el mundo, lo que incluye empresas como Schlumberger y Chevron en el sector energético o BNP Pariba en el sector bancario. Y en el 2021 se puso en funcionamiento el cluster Perlmutter que esta formado por 6,159 NVIDIA A100 Tensor Core GPUs para ciencias astrofísicas y atmosféricas.

Por otra parte, la reciente llegada de los últimos sistemas operativos de Microsoft y Apple está convirtiendo el GPU Computing en una tecnología de uso masivo. En estos nuevos sistemas, la GPU no actúa únicamente como procesador gráfico, sino como procesador paralelo de propósito general accesible para cualquier aplicación.

**Plataforma de Cálculo Paralelo CUDA** proporciona unas cuantas extensiones de MatLab, Fortran, Python, Julia, C, C++, etc. Que permiten implementar el paralelismo en el procesamiento de tareas y datos con diferentes niveles de granularidad. El programador puede expresar ese paralelismo mediante diferentes lenguajes de alto nivel como Fortran, Python, Julia, C y C++ o mediante estándares abiertos como las directivas de OpenACC -que es un estándar de programación para el cómputo en paralelo desarrollado por Cray, CAPS, Nvidia y PGI diseñado para simplificar la programación paralela de sistemas heterogéneos de CPU/GPU-. En la actualidad, la plataforma CUDA se utiliza en miles de aplicaciones aceleradas en la GPU y en miles de artículos de investigación publicados, en las áreas de:

- Bioinformática
- Cálculo financiero
- Dinámica de fluidos computacional (CFD)
- Ciencia de los datos, analítica y bases de datos
- Defensa e Inteligencia
- Procesamiento de imágenes y visión computarizadas
- EDA (diseño automatizado)
- Aprendizaje automático, Inteligencia artificial
- Ciencia de los materiales
- Medios audiovisuales y entretenimiento

- Imágenes médicas
- Dinámica molecular
- Análisis numérico
- Química cuántica
- Exploración sísmica
- Mecánica estructural computacional
- Visualización e interacción de proteínas
- Modelos meteorológicos y climáticos

Algunas bibliotecas aceleradas en la GPU:

- Thrust C++ Template
- cuBLAS
- cuSPARSE
- NPP
- cuFFT

Algunos lenguajes de programación:

- CUDA C/C++, Fortran, Python, Julia
- .NET, Java

Algunos compiladores disponibles:

- OpenACC, paraleliza automáticamente los bucles de código Fortran o C utilizando directivas
- Compilador de autoparalelización de C y Fortran (de PGI) para CUDA C
- Compilador de autoparalelización HMPP de CAPS para CUDA C basado en C y Fortran

- Fortran, Python, Julia, Java, C++ y C
- Compilador de Fortran para CUDA de PGI
- Traductor de Fortran a C para CUDA
- FLAGON: librería de Fortran 95 para cálculo en la GPU
- Interfaz (Wrapper) de Python para CUDA: PyCUDA
- Wrapper de Java
- jCUDA: Java para CUDA
- Vínculos para las librerías BLAS y FFT de CUDA
- JaCUDA
- Integración de .NET para CUDA
- Thrust: librería de plantillas de C++ para CUDA
- CuPP : infraestructura de C++ para CUDA
- Libra: capa de abstracción de C/C++ para CUDA
- F# para CUDA
- Librería ArrayFire para acelerar el desarrollo de aplicaciones de GPU Computing en C, C++, Fortran y Python

Soporte de MATLAB, Mathematica, R, LabView:

- MATLAB, Mathematica, R, LabView
- MathWorks: Librerías MATLAB procesadas con GPUs NVIDIA
- Plugin Jacket basado en CUDA para MATLAB
- GPULib: librería de funciones matemáticas con vínculos para IDL y MATLAB
- Programación con CUDA en Mathematica de Wolfram

- Plugin de Mathematica para CUDA
- Habilitación del GPU Computing en el entorno estadístico de R
- Librería CUDA para LabVIEW de National Instruments
- Formas de usar CUDA desde LabVIEW CVI

Además existen herramientas de productividad y Cluster:

- Soporte de Eclipse para CUDA
- CUDA Occupancy Calculator
- Administrador de Clusters de cálculo para GPUs Tesla
- PBS Works para GPUs Tesla
- Scyld de Penguin Computing

## 2.4 Métricas de Desempeño

Las métricas de desempeño del procesamiento de alguna tarea en paralelo es un factor importante para medir la eficiencia y consumo de recursos al resolver una tarea con un número determinado de procesadores y recursos relacionados de la interconexión de éstos.

Entre las métricas para medir desempeño en las cuales como premisa se mantiene fijo el tamaño del problema, destacan las siguientes:

- Factor de aceleración
- Eficiencia
- Fracción serial

Cada una de ellas mide algo en particular y sólo la combinación de estas dan un panorama general del desempeño del procesamiento en paralelo de un problema en particular en una arquitectura determinada al ser comparada con otras.

**Factor de Aceleración (o Speed-Up)** Se define como el cociente del tiempo que se tarda en completar el cómputo de la tarea usando un sólo procesador entre el tiempo que necesita para realizarlo en $p$ procesadores trabajando en paralelo:

$$s = \frac{T(1)}{T(p)} \tag{2.1}$$

se asume que se usará el mejor algoritmo tanto para un solo procesador como para $p$ procesadores.

Esta métrica en el caso ideal debería de aumentar de forma lineal al aumento del número de procesadores.

**Eficiencia** Se define como el cociente del tiempo que se tarda en completar el cómputo de la tarea usando un solo procesador entre el número de procesadores multiplicado por el tiempo que necesita para realizarlo en $p$ procesadores trabajando en paralelo:

$$e = \frac{T(1)}{pT(p)} = \frac{s}{p}. \tag{2.2}$$

Este valor será cercano a la unidad cuando el Hardware se esté usando de manera eficiente, en caso contrario el Hardware será desaprovechado.

**Fracción serial** Se define como el cociente del tiempo que se tarda en completar el cómputo de la parte secuencial de una tarea entre el tiempo que se tarda en completar el cómputo de la tarea usando un solo procesador:

$$f = \frac{T_s}{T(1)} \tag{2.3}$$

pero usando la ley de Amdahl:

$$T(p) = T_s + \frac{T_p}{p}$$

y reescribiéndola en términos de factor de aceleración, obtenemos la forma operativa del cálculo de la fracción serial que adquiere la forma siguiente:

$$f = \frac{\frac{1}{s} - \frac{1}{p}}{1 - \frac{1}{p}}. \tag{2.4}$$

Esta métrica permite ver las inconsistencias en el balance de cargas, ya que su valor debierá de tender a cero en el caso ideal, por ello un incremento en el valor de $f$ es un aviso de granularidad fina con la correspondiente sobrecarga en los procesos de comunicación.

**Costo o Trabajo** Se define el costo o trabajo de resolver un problema en paralelo como el producto del tiempo de cálculo en paralelo $T_p$ por el número de procesadores usando $p$ y se representa por:

$$C_p = p * T_p.$$

**Aceleración y Eficiencia Relativa** Cuando se trabaja en más de un equipo paralelo -supongamos con $p$ y $p'$ procesadores con $p \geq p'$- es común comparar el desempeño entre dichos equipos. Para ello se define la aceleración relativa como:

$$S_p^{p'} = \frac{T_{p'}}{T_p}$$

para $p \geq p'$, en la cual se espera que:

$$S_p^{p'} \simeq \frac{p}{p'}$$

y eficiencia relativa como:

$$E_p^{p'} = \frac{p'}{p} S_p^{p'} = \frac{p'}{p} \frac{T_{p'}}{T_p}.$$

**Análisis de Rendimiento Usando Métricas** Suponiendo un Cluster con 17 Cores o núcleos[37], se muestra una ejemplificación de las métricas para un problema de ejemplo:

[37] A cada procesador encapsulado se le llama Core o núcleo, logrando que la comunicación entre ellos se realiza de una forma más rápida a través de un bus interno integrado en la propia pastilla de silicio sin tener que recurrir por tanto al bus del sistema mucho más lento. Al contrario del caso de la tecnología HyperThreading que el sistema operativo los reconoce como un core en el equipo, pero estos cores virtuales distan mucho del desempeño de uno real.

| Cores | Tiempo | Aceleración | Eficiencia | Frac. Ser. |
|---|---|---|---|---|
| 1 | 5295 | | | |
| 3 | 2538 | 2.08 | 0.69 | 0.218 |
| 5 | 1391 | 3.80 | 0.76 | 0.078 |
| 9 | 804 | 6.58 | 0.73 | 0.045 |
| 17 | 441 | 12.00 | 0.70 | 0.025 |

Nótese que en todos los casos la fracción serial disminuye sustancialmente con el aumento del número de procesadores, pero la aceleración está por debajo del valor esperado.

Suponiendo un Cluster A con 100 Cores y un Cluster B con 128 Cores, se muestra una ejemplificación de las métricas para un problema de ejemplo en ambos Clusters con los siguientes resultados:

| Cores | Cluster A | Cluster B |
|---|---|---|
| 16 | 9158 seg | ND |
| 32 | 5178 seg | 5937 seg |
| 64 | 3647 seg | 4326 seg |
| 100 | 2661 seg | |
| 128 | | 2818 seg |

Como se muestra en la tabla, en todos los casos el Cluster A usando como máximo 100 Cores obtiene un tiempo de cálculo inferior al que requiere Cluster B usando a lo más los 128 Cores.

Haciendo uso de las métricas de aceleración y eficiencia relativa[38] se tiene que para el Cluster B:

$$S_{128}^{32} = 5937/2818 = 2.10$$

donde lo esperado sería:

$$S_{128}^{32} = 32/128 = 4.00,$$

para el caso de la eficiencia:

$$E_{128}^{32} = (32/128) * (5937/2818) = 0.52.$$

[38] Aceleración relativa es $S_p^{p'} = \frac{T_{p'}}{T_p}$ para $p \geq p'$, en la cual se espera que $S_p^{p'} \simeq \frac{p}{p'}$ y eficiencia relativa es $E_p^{p'} = \frac{p'}{p} S_p^{p'} = \frac{p'}{p} \frac{T_{p'}}{T_p}$.

En el caso del Cluster A se tiene que:

$$S_{100}^{16} = 9158/2661 = 3.44$$

dónde lo esperado sería:

$$S_{100}^{16} = 16/100 = 6.35,$$

para el caso de la eficiencia:

$$E_{100}^{16} = (16/100) * (9158/2661) = 0.55.$$

Haciendo uso del mismo número de Cores base para el Cluster A que para Cluster B, se tiene que:

$$S_{100}^{32} = 5178/2661 = 1.94$$

dónde lo esperado sería:

$$S_{100}^{16} = 32/100 = 3.12,$$

para el caso de la eficiencia:

$$E_{100}^{16} = (32/100) * (5178/2661) = 0.62.$$

De todo lo anterior, se desprende que el Cluster A obtiene valores de una aceleración y eficiencias relativas ligeramente mejores que el Cluster B, pero esto no se refleja en la disminución de casi 6% del tiempo de ejecución y del uso de 28 Cores menos.

Además, el costo computacional:

$$C_p = P * T_p,$$

que para el caso del Cluster B es:

$$C_{128} = 360,704$$

y en Cluster A es:

$$C_{100} = 266,100$$

que representa una disminución de 27%; además de un factor muy importante, el Cluster A tuvo un costo monetario mucho menor con respecto del Cluster B.

## 2.5 Programación de Cómputo de Alto Rendimiento

Hay muchas aplicaciones a las herramientas computacionales, pero nos interesan 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. La solución de estos problemas genera un alto consumo de memoria, espacio de almacenamiento o tiempo de cómputo; por ello nos interesa trabajar en computadoras que nos puedan satisfacer estas demandas.

La computación de alto rendimiento (véase 2) -High performance Computing o HPC en inglés- es la agregación de potencia de cálculo para resolver problemas complejos en Ciencia e Ingeniería o Gestión. Para lograr este objetivo, la computación de alto rendimiento se apoya en tecnologías computacionales como los Clusters, las supercomputadoras o la computación paralela. La mayoría de las ideas actuales de la computación distribuida se han basado en la computación de alto rendimiento.

La computación paralela o de alto rendimiento es una forma de cómputo en la que muchas instrucciones se ejecutan simultáneamente, operando sobre el principio de que problemas grandes, a menudo se pueden dividir en unos más pequeños, que luego son resueltos simultáneamente (en paralelo). Hay varias formas diferentes de computación paralela: paralelismo a nivel de bits, paralelismo a nivel de instrucción, paralelismo de datos y paralelismo de tareas.

El paralelismo se ha empleado durante muchos años, sobre todo en la computación de altas prestaciones, pero el interés en ella ha crecido últimamente debido a las limitaciones físicas que impiden el aumento de la frecuencia de los actuales procesadores comerciales. Como el consumo de energía –y por consiguiente la generación de calor– de las computadoras constituye una preocupación en los últimos años, la computación en paralelo se ha convertido en el paradigma dominante en la arquitectura de computadoras, principalmente en forma de procesadores multinúcleo.

Las computadoras paralelas pueden clasificarse según el nivel de paralelismo que admite su Hardware:

- Equipos con procesadores multinúcleo

- Equipos con multiprocesador (múltiples procesadores multinúcleo en una sola máquina)
- Procesadores masivamente paralelos
- Equipos con una o más tarjetas CUDA (Compute Unified Device Architecture)
- Cluster de equipos multinúcleo, multiprocesador y/o CUDAS
- Cúmulos de Clusters (Grids)

Muchas veces, para acelerar tareas específicas, se utilizan arquitecturas especializadas de computación en paralelo junto a procesadores tradicionales.

Existen múltiples vertientes en el cómputo en paralelo, algunas de ellas son:

- Cómputo en memoria compartida usando OpenMP
- Cómputo en memoria distribuida usando MPI

Como es de esperarse, los programas informáticos paralelos son más complejos y difíciles de escribir que los secuenciales, porque la concurrencia introduce nuevos tipos de errores de Software -por ello existe una creciente gama de herramientas que coadyuvan a mejorar la escritura, depuración y desempeño de los programas en paralelo-, pero la comunicación y sincronización entre diferentes subtareas son algunos de los mayores obstáculos para obtener un buen rendimiento del programa paralelo.

Además, el tiempo de programación necesario para desarrollar una aplicación paralela eficiente y eficaz para la gran mayoría de los programadores puede ser de semanas o meses en el mejor de los casos. Por ello, es necesario hacer un balance entre las diferentes opciones de paralelización para no invertir un tiempo precioso que puede no justificar dicha inversión económica, de recursos computacionales y sobre todo de tiempo.

Actualmente, en muchos centros de cómputo es una práctica común usar directivas de compilación en equipos paralelos sobre programas escritos de forma secuencial, con la esperanza que sean puestos por el compilador como programas paralelos. Esto en la gran mayoría de los casos genera códigos poco eficientes, pese a que corren en equipos paralelos y pueden usar toda

la memoria compartida de dichos equipos, el algoritmo ejecutado continúa siendo secuencial en una gran parte del código.

Si la arquitectura paralela donde se implemente el programa es UMA de acceso simétrico, los datos serán accesados a una velocidad de memoria constante. En caso contrario, al acceder a un conjunto de datos es común que una parte de estos sean locales a un procesador (con un acceso del orden de nanosegundos), pero el resto de los datos deberán ser accesados mediante red (con acceso del orden de milisegundos), siendo esto muy costoso en tiempo de procesamiento.

Por lo anterior, si se cuenta con computadoras con memoria compartida o que tengan interconexión por bus, salvo en casos particulares no será posible explotar éstas características eficientemente. Pero en la medida en que se adecuen los programas para usar bibliotecas y compiladores acordes a las características del equipo disponible -algunos de ellos sólo existen de manera comercial- la eficiencia aumentará de manera importante.

### 2.5.1 Programando con OpenMP para Memoria Compartida

OpenMP es una interfaz de programación de aplicaciones (API) para la programación multiproceso de memoria compartida en múltiples plataformas. Permite añadir concurrencia a los programas escritos en C, C++, Julia, Python y Fortran sobre la base del modelo de ejecución Fork-join. Está disponible en muchas arquitecturas, incluidas las plataformas de Unix, Linux y de Microsoft Windows. Se compone de un conjunto de directivas de compilador, rutinas de biblioteca, y variables de entorno que influyen en el comportamiento en tiempo de ejecución.

Definido conjuntamente por proveedores de Hardware y de Software, OpenMP es un modelo de programación portable y escalable que proporciona a los programadores una interfaz simple y flexible para el desarrollo de aplicaciones paralelas, para plataformas que van desde las computadoras de escritorio hasta supercomputadoras. Una aplicación construida con un modelo de programación paralela híbrido se puede ejecutar en un Cluster de computadoras utilizando OpenMP y MPI, o a través de las extensiones de OpenMP para los sistemas de memoria distribuida.

OpenMP se basa en el modelo *Fork-join*, paradigma que proviene de los sistemas Unix, donde una tarea muy pesada se divide en $K$ hilos (Fork) con menor peso, para luego «recolectar» sus resultados al final y unirlos en un solo resultado (Join).

Cuando se incluye una directiva de compilador OpenMP esto implica que se incluye una sincronización obligatoria en todo el bloque. Es decir, el bloque de código se marcará como paralelo y se lanzarán hilos según las características que nos dé la directiva, y al final de ella habrá una barrera para la sincronización de los diferentes hilos (salvo que implícitamente se indique lo contrario con la directiva Nowait). Este tipo de ejecución se denomina *Fork-join*.

OpenMP también soporta el modelo de paralelismo de tareas. El equipo de hilos del bloque de código paralelo ejecuta las tareas especificadas dentro de dicho bloque. Las tareas permiten un paralelismo asíncrono. Desde la versión 4.0 lanzada en 2013 admite la especificación de dependencias entre tareas, relegando a la biblioteca de tiempo de ejecución de OpenMP el trabajo de planificar las tareas y ponerlas en ejecución. Los hilos de ejecución irán ejecutando las tareas a medida que estas estén disponibles (sus dependencias ya estén satisfechas). El uso de tareas da lugar a sincronización con una granularidad más fina. El uso de barreras no es estrictamente necesario, de manera que los hilos pueden continuar ejecutando tareas disponibles sin necesidad de esperar a que todo el equipo de hilos acabe un bloque paralelo. El uso de tareas con dependencias crea un grafo, pudiéndose aplicar propiedades de grafos a la hora de escoger tareas para su ejecución.

Salvo el uso de implementaciones de Hardware de la biblioteca de tiempo de ejecución OpenMP (p.ej. en una matriz de puertas programables FPGAs), los sobrecostes de las tareas es mayor, este sobrecoste ha de ser amortizado mediante el potencial paralelismo adicional que las tareas exponen.

**Estructura del Programa en C++** Ejemplo de cálculo de Pi usando OpenMP:

```
#include <stdio.h>
// Indica si se carga lo referente a OpenMP
#ifdef _OPENMP
#include <omp.h>
int threads=omp_get_num_threads();
#else
int threads=0;
#endif
#define STEPCOUNTER 1000000000
int main (void)
```

```
{
  long i;
  double pi=0;
  printf("threads %d", threads);
#pragma omp parallel for reduction(+:pi)
  for (i=0; i < STEPCOUNTER; i++)
  {
    pi += 1.0/(i*4.0 +1.0);
    pi -= 1.0/(i*4.0 +3.0);
  }
  pi = pi*4.0;
  printf("PI = %2.16lf ",pi);
  return 0;
}
```

El compilador de OpenMP es el mismo que para los lenguajes C, C++ o Fortran respectivamente (véase 3.7). Por ello, para usarlo en C++ en línea de comandos (véase 7.3), instalamos el compilador g++, mediante:

```
# apt install g++
```

Así, para compilar con g++[39], sin usar OpenMP, usamos:

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

Ejecutar midiendo el tiempo[40]:

```
$ time ./pi
```

[39] Compilar fuentes en C++ solicitando que el ejecutable tenga el nombre *ejemp*:

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

en este caso no se usa ninguna directiva para optimizar el ejecutable generado. Para compilar usando diversas optimizaciones (O1, -O2 o -O3) usar por ejemplo:

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

[40] Los sistemas operativos tipo Linux/Unix cuentan con un comando básico que mide el tiempo de ejecución de cualquier comando, usando:

```
$ time ls
```

Ahora, usando el compilar para OpenMP usamos:

```
$ g++ -o pi -fopenmp pi.cpp
```

Indicar el número de hilos, por ejemplo 2:

```
$ export OMP_NUM_THREADS=2
```

Ejecutar:

```
$ time ./pi
```

### 2.5.2 Programando con MPI para Memoria Distribuida

Para poder intercomunicar dos o más Cores en una o en múltiples computadoras se usa la «interfaz de paso de mensajes (Message Passing Interface MPI)» (véase [16], [17], [20] y [18]), una biblioteca de comunicación para procesamiento en paralelo que puede ser usada desde lenguajes de programación como: C, C++, Pytho, Julia, Fortran. MPI ha sido desarrollado como un estándar para el paso de mensajes y operaciones relacionadas.

Este enfoque es adoptado por usuarios e implementadores de bibliotecas, en el cual se proveen a los programas de procesamiento en paralelo de portabilidad y herramientas necesarias para desarrollar aplicaciones que puedan usar el cómputo paralelo de alto desempeño.

El modelo de paso de mensajes posibilita a un conjunto de procesos que tienen solo memoria local, la comunicación con otros procesos (usando Bus o red) mediante el envío y recepción de mensajes. Por definición el paso de mensajes posibilita transferir datos de la memoria local de un proceso a la memoria local de cualquier otro proceso que lo requiera.

En el modelo de paso de mensajes para equipos paralelos, los procesos se ejecutan en paralelo, teniendo direcciones de memoria separada para cada proceso, la comunicación ocurre cuando una porción de la dirección de memoria de un proceso es copiada mediante el envío de un mensaje dentro de otro proceso en la memoria local mediante la recepción del mismo.

---

pero podemos instalar el paquete que además de medir el tiempo de ejecución nos diga que recursos se usaron el la ejecución del comando indicado, para instalarlo usamos:

```
# apt install time
```

el uso es el mismo del comando interno time.

Las operaciones de envío y recepción de mensajes es cooperativa y ocurre sólo cuando el primer proceso ejecuta una operación de envío y el segundo proceso ejecuta una operación de recepción, los argumentos base de estas funciones son:

- Para el que envía, la dirección de los datos a transmitir y el proceso destino al cual los datos se enviarán.
- Para el que recibe, debe tener la dirección de memoria donde se pondrán los datos recibidos, junto con la dirección del proceso que los envío.

Es decir:
*Send(dir, lg, td, dest, etiq, com)*

$\{dir, lg, td\}$ describe cuántas ocurrencias $lg$ de elementos del tipo de dato $td$ se transmitirán empezando en la dirección de memoria $dir$.

$\{des, etiq, com\}$ describe el identificador $etiq$ de destino $des$ asociado con la comunicación $com$.

*Recv(dir, mlg, td, fuent, etiq, com, st)*

$\{dir, lg, td\}$ describe cuántas ocurrencias $lg$ de elementos del tipo de dato $td$ se transmitirán empezando en la dirección de memoria $dir$.

$\{fuent, etiq, com, est\}$ describe el identificador $etiq$ de la fuente $fuent$ asociado con la comunicación $com$ y el estado $est$.

El conjunto básico de directivas (en nuestro caso sólo se usan estas) en C++ de MPI son (véase [16] y [17]):

| | |
|---|---|
| MPI::Init | Inicializa al MPI |
| MPI::COMM_WORLD.Get_size | Busca el número de procesos existentes |
| MPI::COMM_WORLD.Get_rank | Busca el identificador del proceso |
| MPI::COMM_WORLD.Send | Envía un mensaje |
| MPI::COMM_WORLD.Recv | Recibe un mensaje |
| MPI::Finalize | Termina al MPI |

**Estructura del Programa en C++** Ejemplo de Hola_Mundo en MPI:

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char *argv[])
{
int rank, size;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
printf("Hola! Soy el %d de %d\n", rank, size);
MPI_Finalize();
return 0;
}
```

Otro ejemplo, para realizar una suma en MPI:

```
#include <iostream>
#include <iomanip>
#include <mpi.h>
using namespace std;
int main(int argc, char ** argv){
int mynode, totalnodes;
int sum = 0,startval,endval,accum;
MPI_Status status;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &totalnodes);
MPI_Comm_rank(MPI_COMM_WORLD, &mynode);
startval = 1000*mynode/totalnodes+1;
endval =1000*(mynode+1)/totalnodes;
for(int i=startval;i<=endval;i=i+1) sum = sum + i;
if(mynode!=0)
MPI_Send(&sum, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
else
for(int j=1;j<totalnodes;j=j+1){
MPI_Recv(&accum, 1, MPI_INT, j, 1,
MPI_COMM_WORLD, &status);
sum = sum + accum;
```

```
}
if(mynode == 0)
cout << "The sum from 1 to 1000 is: "<< sum << endl;
MPI_Finalize();
}
```

Existe una gran variedad de compiladores de MPI en línea de comandos (véase 7.3), algunos disponibles en Debian GNU/Linux son instalados mediante:

```
# apt install lam-runtime libmpich-dev mpi-default-dev mpich\
mpi-default-bin openmpi-bin valgrind-mpi xmpi
```

Para compilar y ejecutar es posible usar alguna de estas opciones:

mpic++, mpic++.openmpi, mpiexec.mpich, mpif90.openmpi, mpirun.lam, mpicc, mpicxx, mpiexec.openmpi, mpifort, mpirun.mpich, mpiCC, mpicxx.mpich, mpif77, mpifort.mpich, mpirun.openmpi, mpicc.mpich, mpicxx.openmpi, mpif77.mpich, mpifort.openmpi, mpitask, mpicc.openmpi, mpiexec, mpif77.openmpi, mpimsg, mpivars, mpiCC.openmpi, mpiexec.hydra, mpif90, mpipython, mpichversion, mpipython, mpiexec.lam, mpif90.mpich, mpirun

Por ejemplo, para compilar *ejemp.cpp* en *mpic++* solicitando que el ejecutable tenga el nombre *ejemp*, usamos:

```
$ mpic++ ejemp.cpp -o ejemp
```

en este caso no se uso ninguna opción de optimización en tiempo de compilación (véase 3.7), se puede hacer uso de ellas (-O1, -O2 o -O3), mediante:

```
$ mpic++ -O3 ejemp.cpp -o ejemp
```

para ejecutar el programa ya compilado en 4 procesos y medir el tiempo de ejecución (véase 3.7), usamos:

```
$ time mpirun -np 4 ejemp
```

También podemos compilar *ejemp.c* en *mpicc* solicitando que el ejecutable tenga el nombre *ejemp*:

```
$ mpicc ejemp.cpp -o ejemp
```

en este caso no se uso ninguna opción de optimización en tiempo de compilación, se puede hacer uso de ellas (-O1, -O2 o -O3), mediante:

```
$ mpicc -O3 ejemp.c -o ejemp
```

para ejecutar el programa ya compilado en 4 procesos, usamos:

```
$ mpirun -np 4 ejemp
```

Un último ejemplo, en el caso de usar *mpiCC.mpic* y *lamboot*, entoces es necesario compilar usando:

```
$ mpiCC.mpich -O2 ejemp.cpp -o ejemp -lm
```

iniciar ambiente de ejecución paralelo:

```
$ lamboot -v
```

correr usando 8 procesos por ejemplo:

```
$ mpirun.mpich -np 8 ejemp
```

correr usando 5 procesos segun lista *machines.lolb* por ejemplo:

```
$ mpirun.mpich -machinefile machines.lolb -np 5 ejemp
```

terminar ambiente de ejecución paralelo:

```
$ lamhalt -v
```

**Observación 1** *Para que en la ejecución de MPI no pida la clave de usuario:*
*$ ssh-keygen -t rsa*
*En cada pregunta responder con ENTER, para después copiar usando:*
*$ ssh-copy-id usuario@servidor*
*Ojo: Si continúa pidiendo clave, es que esta instalado rsh o lsh.*

### 2.5.3 Esquema de Paralelización Principal-Subordinados

El esquema de paralelización Principal-Subordinados -también conocido como Maestro-Esclavo-, permite sincronizar por parte del nodo principal las tareas que se realizan en paralelo usando varios nodos subordinados, éste modelo puede ser explotado de manera eficiente si existe poca comunicación entre el principal y el subordinado -entre los subordinados no debe de existir comunicación- y los tiempos consumidos en realizar las tareas asignadas son mayores que los períodos involucrados en las comunicaciones para la asignación de dichas tareas. De esta manera se garantiza que la mayoría de los procesadores estarán trabajando de manera continua y existirán pocos tiempos muertos.

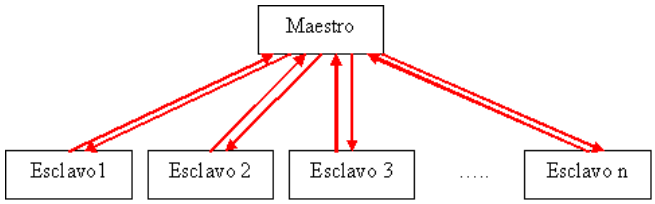



Figura 11: Esquema del Maestro-Esclavo

Donde, tomando en cuenta la implementación en estrella del Cluster, el modelo de paralelismo de MPI (véase 2.5.2) y las necesidades propias de comunicación del programa, el nodo principal tendrá comunicación sólo con cada nodo subordinado y no existirá comunicación entre los nodos subordinados, esto reducirá las comunicaciones y optimizará el paso de mensajes, algunos ejemplos de este esquema se pueden consultar en:

Herramientas
http://132.248.181.216/Herramientas/

Un factor limitante en este esquema es que el nodo principal deberá de atender todas las peticiones hechas por cada uno de los nodos subordinados, esto toma especial relevancia cuando todos o casi todos los nodos subordinados compiten por ser atendidos por el nodo principal.

Se recomienda implementar este esquema en un Cluster heterogéneo en donde el nodo principal sea más poderoso computacionalmente que los nodos subordinados. Si a este esquema se le agrega una red de alta velocidad y de

baja latencia, se le permitirá operar al Cluster en las mejores condiciones posibles, pero este esquema se verá degradado al aumentar el número de nodos subordinados inexorablemente.

Pero hay que ser cuidadosos en cuanto al número de nodos subordinados que se usan en la implementación en tiempo de ejecución versus el rendimiento general del sistema al aumentar estos, algunas observaciones posibles son:

- El esquema Principal-Subordinados programado usando MPI lanza $P$ procesos (uno para el nodo principal y $P-1$ para los nodos subordinados), estos en principio corren en un solo procesador pero pueden ser lanzados en múltiples procesadores usando una directiva de ejecución, de esta manera es posible que en una sola máquina se programe, depure y sea puesto a punto el código usando mallas pequeñas (del orden de cientos de nodos) y cuando este listo puede mandarse a producción en un Cluster.

- El esquema Principal-Subordinados no es eficiente si sólo se usan dos procesadores (uno para el nodo principal y otro para el nodo subordinado), ya que el nodo principal en general no realiza los cálculos pesados y su principal función será la de distribuir tareas; los cálculos serán delegados al nodo subordinado.

**Estructura del Programa Principal-Subordinados** La estructura del programa se realizo para que el nodo principal mande trabajos de manera síncrona a los nodos subordinados. Cuando los nodos subordinados terminan la tarea asignada, avisan al nodo principal para que se les asigne otra tarea (estas tareas son acordes a la etapa correspondiente del método de descomposición de dominio ejecutándose en un instante dado). En la medida de lo posible se trata de mandar paquetes de datos a cada nodo subordinado y que estos regresen también paquetes al nodo principal, a manera de reducir las comunicaciones al mínimo y tratar de mantener siempre ocupados a los nodos subordinados para evitar los tiempos muertos, logrando con ello una granularidad gruesa, ideal para trabajar con Clusters.

La estructura básica del programa bajo el esquema Principal-Subordinados codificada en C++ y usando MPI (véase 2.5.2) es:

```
main(int argc, char *argv[])
```

```
{
   MPI::Init(argc,argv);
   ME_id = MPI::COMM_WORLD.Get_rank();
   MP_np = MPI::COMM_WORLD.Get_size();
   if (ME_id == 0) {
     // Operaciones del Principal
   } else {
     // Operaciones del Subordinado con identificador ME_id
    }
   MPI::Finalize();
}
```

En este único programa se deberá de codificar todas las tareas necesarias para el nodo principal y cada uno de los nodos subordinados, así como las formas de intercomunicación entre ellos usando como distintivo de los distintos procesos a la variable *ME_id*. Para más detalles de esta forma de programación y otras funciones de MPI (véase 2.5.2, [16] y [17]).

El principal factor limitante para el esquema Principal-Subordinados es que se presupone contar con un nodo principal lo suficientemente poderoso para atender simultáneamente las tareas síncronas del método, ya que este distribuye tareas acorde al número de nodos subordinados, estas si son balanceadas, ocasionaran que muchos de los procesadores subordinados terminen aproximadamente al mismo tiempo y el nodo principal tendrá que atender múltiples comunicaciones simultáneamente, degradando su rendimiento al aumentar el número de nodos subordinados que atender.

Para los factores limitantes inherente al propio esquema Principal-Subordinados, es posible implementar algunas operaciones del nodo principal en paralelo, ya sea usando equipos multiprocesador o en más de un nodo distinto a los nodos subordinados.

### 2.5.4 Opciones de Paralelización Híbridas

En la actualidad, casi todos los equipos de cómputo usados en estaciones de trabajo y Clusters cuentan con dos o más Cores, en ellos siempre es posible usar MPI para intercambiar mensajes entre procesos corriendo en

el mismo equipo de cómputo, pero no es un proceso tan eficiente como se puede querer. En estas arquitecturas llamadas de memoria compartida es mejor usar OpenMP o cualquiera de sus variantes para trabajar en paralelo. Por otro lado es común contar con las cada vez más omnipresentes tarjetas NVIDIA, y con los cada vez más numerosos Cores CUDA -que una sola tarjeta NVIDIA TESLA puede tener del orden de miles de ellos- y que en un futuro serán cada vez más numerosos.

Para lograr obtener la mayor eficiencia posible de estos tres niveles de paralelización, se están implementando procesos híbridos (véase [21] y [19]), en donde la intercomunicación de equipos con memoria compartida se realiza mediante MPI y la intercomunicación entre Cores que comparten la misma memoria se realiza con OpenMP, además las operaciones matriciales, vectoriales, etc. se le encargan a los numerosos Cores CUDA de las tarjetas NVIDIA.

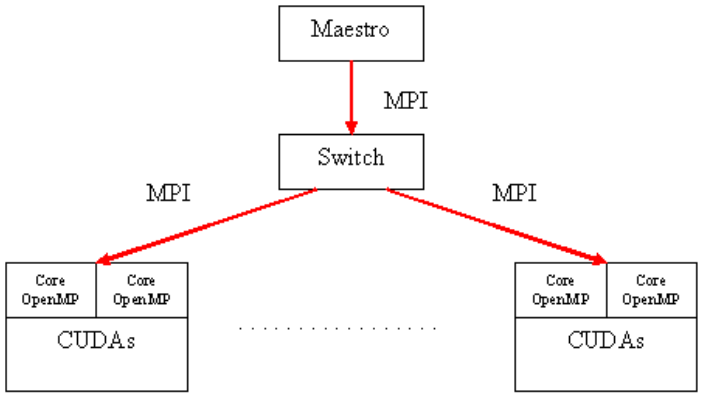



Figura 12: Paralelización Híbrida

Se han desarrollado múltiples programas que sus algoritmos resuelven ecuaciones diferenciales parciales concomitantes en Ciencias e Ingenierías que hacen uso de esta forma integradora de paralelismo. Para ello, la interconexión de equipos de memoria compartida se realizaría mediante MPI y en cada equipo de memoria compartida se manipularían uno o más subdominios mediante OpenMP -ya que cada subdominio es independiente de los demás- y la manipulación de matrices y operaciones entre matrices y vectores que requiere cada subdominio se realizarían en las tarjetas NVIDIA mediante los numerosos Cores CUDA sin salir a la RAM de la computadora.

Permitiendo así, tener una creciente eficiencia de paralelización que optimizan en gran medida los recursos computacionales, ya que todas las matrices y vectores se generarían en la GRAM de la tarjeta NVIDIA. De forma tal que sea reutilizable y que pueda usarse en problemas en los que el número de grados de libertad sea grande, permitiendo hacer uso de equipos de cómputo cada vez más asequibles y de menor costo, pero con una creciente eficiencia computacional que compiten con los grandes equipos de cómputo de alto desempeño.

## 2.6 Programando Desde la Nube

Existen diferentes servicios Web[41] 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.

**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 Machne 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

[41] 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.

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

# 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[42] 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.

[42] 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 [5]) 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 [13]). 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
```

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

Lenguaje Java

## 3.2 Python

Python (véase [6]) 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 10.2).

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 gráficas 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[43]. 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.

[43] 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

**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.

**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
```

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

Lenguaje Python

## 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 10.2).

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
```

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

Lenguaje Julia

## 3.4 C y C++

**C** (véase [7]) 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 [8]) 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
```

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

Lenguajes C y C++

## 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 [**?**]) 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 [**?**]). 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 [**?**]) 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 [**?**]), RStudio (véase [**?**]) 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)[44] 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**[45]

[44] 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.

[45] 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.

- 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/

En este apartado, solo tocaremos las más usadas, pero abunda la documentación de estas y otras importantes herramientas en línea de comandos (véase 7.3). Iniciaremos por las de compilar[46] y depurar[47] 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

[46] 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.

[47] 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.

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 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[48], 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

[48] 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.

*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
```

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

**Aprender a Usar ASTYLE** en la red existen múltiples sitios especializados y una amplia bibliografía para formatear código nosotros hemos seleccionado diversos textos que ponemos a su disposición en:

Sistema Operativo Linux

### 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[49], sin pérdida de generalidad trabajaremos en C++ solicitando que el archivo ejecutable[50] 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
```

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

[50] 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.

mostrará los avisos de análisis estático del código indicado.

Para conocer el tiempo de ejecución[51] 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
```

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
```

[51] 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).

```
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[52] 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[53] se obtienen estos rendimientos:
Cryto++ v8.2:

[52] 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.

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

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)

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.

**Aprender a Usar GPROF** en la red existen múltiples sitios especializados y una amplia bibliografía para optimizar código, nosotros hemos seleccionado diversos textos que ponemos a su disposición en:

GPROF

**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
```

**Aprender a Usar VALGRIND** en la red existen múltiples sitios especializados y una amplia bibliografía para depurar código, nosotros hemos seleccionado diversos textos que ponemos a su disposición en:

VALGRIND

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[54] a lo largo de los años, incluidos[55] 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
```

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.*

[54] 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.

[55] 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.

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/

**Aprender a Usar *Git*** En la red existen múltiples sitios especializados y una amplia bibliografía para aprender a usar *Git*, nosotros hemos seleccionado diversos textos que ponemos a su disposición en:

Git

### 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[56] 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.

[56] Algunos de estos proyectos gratuitos son: Gitlab, Github, Bitbucket, Beanstalk, Launchpad, SourceForge, Phabricator, GitBucket, Gogs, Gitea, Apache Allura, entre otros.

**¿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.

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[57] 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/

[57] 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[58], y repetir el proceso para calcular el siguiente estado. Este código o programa se escribe en un lenguaje de programación que sigue

[58] Un algoritmo es un conjunto preescrito de instrucciones o reglas bien definidas, orde-

algún paradigma de programación[59], 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[60]:

- 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.

[59] Este representa un enfoque particular o filosofía para diseñar soluciones e implementarlas en algún lenguaje de programación.

[60] 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[61] 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[62].

## 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

[61] 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.

[62] 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[63] 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.

[63] 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[64] 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[65] de la referencia de implementación-.

[64] 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.

[65] 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[66] 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.-.

[66] 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 [5], [6], [7] y [8]), 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[67]:

- 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 -3.4x$10^{38}$ a 3.4x$10^{38}$, con 14 dígitos de precisión
- double, 64 bits, rango de -1.7x$10^{308}$ a 1.7x$10^{308}$, con 16 dígitos de precisión
- long double, 80 bits, 3.4x$10^{-4932}$ a 3.4x$10^{4932}$, 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[68], 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.

[67] La precedencia de los operadores básicos son: $=$ el de mayor procedencia, siguen $-$ y $+$ unario, luego $*, /, \%$ y finalmente los operadores $+$ y $-$.

[68] 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[69]:

```
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;
}
```

[69] 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[70] x > y:
   z = y
else:
   z = 0
```

[70] 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[71]
  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:

[71] 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[72]
  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:

[72] 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*[73] 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
```

[73] 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:[74]
      xi = xi + 1
      if (n % i) == 0:
        break
    if xi == np:
      p.append(n)
      np = np + 1
    n = n + 1
   # Visualiza los primos encontrados
```

[74] 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):[75]
        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)
```

[75] 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[76], cuando los ejemplos crecen se empieza a usar programación estructurada[77] -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[78].

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.

[76] 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.

[77] 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.

[78] 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[79] 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:

[79] 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 Aritmética de Punto Flotante

La aritmética de punto flotante es considerada un tema esotérico para muchas personas. Esto es sorprendente porque el punto flotante es omnipresente en los sistemas informáticos. Casi todos los lenguajes de programación tienen un tipo de datos de punto flotante, i.e. los números no enteros como 1.2 ó $1e + 45$.

Los problemas de precisión del punto flotante son una preocupación crítica en la informática y la computación numérica, donde la representación y manipulación de números reales puede conducir a resultados inesperados y erróneos. Estos obstáculos surgen debido alas limitaciones inherentes de la aritmética de punto flotante, que aproxima números reales con una precisión finita.

Esto puede causar problemas importantes en diversas aplicaciones, desde simulaciones científicas hasta cálculos financieros, donde incluso las imprecisiones menores pueden propagarse y amplificarse, dando lugar a errores sustanciales. Comprender estos obstáculos es esencial para que los desarrolladores, ingenieros y científicos implementen algoritmos numéricos sólidos y confiables, garantizando que las matemáticas realizadas por las computadoras sigan siendo confiables y precisas.

**Comprensión de los Errores de Redondeo en Aritmética de Punto Flotante** El meollo de la cuestión es la representación de números de punto flotante. Las computadoras utilizan una cantidad finita de bits para almacenar estos números, y generalmente cumplen con el estándar IEEE 754[80]. Este estándar define cómo se almacenan los números en formato binario, con un número fijo de bits asignados para el signo, el exponente y la mantisa. Si bien esto permite representar una amplia gama de valores, también impone limitaciones a la precisión. No todos los números decimales se pueden representar exactamente en forma binaria, lo que genera pequeñas discrepancias conocidas como errores de redondeo.

Una de las primeras cosas que uno encuentra con sorpresa cuando hace cálculos en una computadora es que por ejemplo, si usamos números muy grandes y seguimos incrementando su valor eventualmente el resultado será

[80] El estándar IEEE 754-2008 define varios tamaños de números de punto flotante: media precisión (binary16), precisión simple (binary32), precisión doble (binary64), precisión cuádruple (binary128), etc., cada uno con su propia especificación.

negativo ... ¿qué pasó? Esto se llama desbordamiento aritmético al intentar crear un valor numérico que está fuera del rango que puede representarse con un número dado de dígitos, ya sea mayor que el máximo o menor que el mínimo valor representable. Algo similar pasa al restar al menor número representable en la máquina, el resultado será positivo y se denomina subdesbordamiento.

Por otro lado, tenemos errores de redondeo, estos ocurren porque el sistema binario no puede representar con precisión ciertas fracciones. Por ejemplo, el número decimal 0.1 no se puede representar exactamente en binario, lo que da como resultado una aproximación, por ejemplo al sumar 0.1 + 0.2 en una computadora usando por ejemplo el lenguaje Python obtenemos:

```
print(0.1+0.2)
0.30000000000000004
```

que no es exactamente lo que esperábamos[81]. Cuando se utilizan tales aproximaciones en los cálculos, los errores pueden acumularse. Este fenómeno es particularmente problemático en procesos iterativos, donde el mismo cálculo se realiza repetidamente y los errores se acumulan con el tiempo.

Estos errores de precisión se vuelven particularmente problemáticos cuando se realizan controles de igualdad. En un mundo ideal, comparar la igualdad de dos números de punto flotante sería sencillo. Sin embargo, debido a los pequeños errores introducidos durante las operaciones aritméticas, dos números que deberían ser iguales pueden no ser exactamente iguales en su representación binaria.

Por ejemplo, el resultado de sumar 0.1 y 0.2 puede no ser exactamente igual a 0.3 debido a la forma en que estos números se representan en binario, por ejemplo:

```
if 0.1 + 0.2 == 0.3:
    print("si")
else:
    print("no")
```

[81] Lo mismo obtenemos si usamos la multiplicación, por ejemplo:

```
print(0.1 * 3)
0.30000000000000004
```

la respuesta será "no". Esta discrepancia puede provocar que fallen las comprobaciones de igualdad, lo que provocará un comportamiento inesperado en los programas.

Para mitigar estos problemas, los desarrolladores suelen utilizar una técnica conocida como "comparación épsilon". En lugar de verificar la igualdad exacta, verifican si la diferencia absoluta entre dos números de punto flotante es menor que un valor pequeño predefinido, conocido como épsilon. Este enfoque reconoce la imprecisión inherente de la aritmética de punto flotante y proporciona una forma más sólida de comparar números. Sin embargo, elegir un valor épsilon apropiado puede resultar complicado, ya que depende del contexto específico y del rango de valores involucrados.

El problema se ve exacerbado por la precisión finita de los números de punto flotante. Al realizar operaciones aritméticas, el resultado a menudo debe redondearse para que se ajuste a los bits disponibles. Este redondeo puede introducir más imprecisiones. Por ejemplo, sumar dos números de punto flotante de magnitudes muy diferentes puede provocar una pérdida de precisión, ya que el número más pequeño puede ignorarse de hecho. Esto se conoce como cancelación catastrófica y puede afectar gravemente a la precisión de los cálculos.

Por ejemplo, ¿qué podría tener de interesante la humilde fórmula cuadrática?. Después de todo, es una fórmula. Simplemente le pones números. Bueno, hay un detalle interesante. Cuando el coeficiente lineal $b$ es grande en relación con los otros coeficientes, la fórmula cuadrática puede dar resultados incorrectos cuando se implementa en aritmética de punto flotante. Eso es cierto, pero veamos qué sucede cuando tenemos $a = c = 1$ y $b = 10e8$ en

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \text{ y } x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \tag{5.1}$$

regresa

$$x_1 = -7.450580596923828e - 09 \text{ y } x_2 = -100000000.0$$

La primera raíz está equivocada en aproximadamente un 25%, aunque la segunda es correcta.

¿Qué pasó? La ecuación cuadrática violó la regla cardinal del análisis numérico: evitar restar números casi iguales. Cuanto más similares sean dos números, más precisión puedes perder al restarlos. En este caso $(b^2 - 4ac)$ es casi igual a $b$. Si evaluamos, obtenemos $1.49e - 8$ cuando sería la respuesta correcta $2.0e - 8$.

Si usamos la otra formula[82]

$$x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}} \text{ y } x_2 = \frac{-2c}{b - \sqrt{b^2 - 4ac}} \tag{5.2}$$

obtenemos

$$x_1 = -1e - 08 \text{ y } x_2 = -134217728.0$$

Entonces, ¿cuál formula cuadrática es mejor? Da la respuesta correcta para la primera raíz, exacta dentro de la precisión de la máquina. Pero ahora la segunda raíz está equivocada en un 34%. ¿Por qué la segunda raíz es incorrecta? La misma razón que antes: ¡restamos dos números casi iguales!

La versión familiar de la fórmula cuadrática calcula correctamente la raíz más grande y la otra versión calcula correctamente la raíz más pequeña. Ninguna versión es mejor en general. No estaríamos ni mejor ni peor usando siempre la nueva fórmula cuadrática que la anterior. Cada uno es mejor cuando evita restar números casi iguales. La solución es utilizar ambas fórmulas cuadráticas, utilizando la apropiada para la raíz que estás intentando calcular.

Otro error común es la suposición de que la aritmética de punto flotante es asociativa y distributiva, como lo es en matemáticas puras. En realidad, el orden de las operaciones puede afectar significativamente el resultado debido a errores de redondeo. Por ejemplo, la expresión (a+ b) + c puede producir un resultado diferente que a + (b + c) cuando se trata de números de punto flotante. Esta no asociatividad puede provocar errores sutiles, especialmente en algoritmos complejos que se basan en cálculos precisos.

Además, la aritmética de punto flotante puede introducir problemas en los algoritmos que requieren comparaciones exactas, como los que se utilizan en la clasificación o la búsqueda. Cuando los números de punto flotante se utilizan como claves en estructuras de datos como tablas hash o árboles de búsqueda binarios, los errores de precisión pueden provocar un comportamiento incorrecto, como no encontrar un elemento existente o insertar incorrectamente un duplicado. Para solucionar este problema, es posible que los desarrolladores necesiten implementar funciones de comparación personalizadas que tengan en cuenta la imprecisión del punto flotante.

Pasando a implicaciones prácticas, estos errores de redondeo pueden tener consecuencias de gran alcance. En las simulaciones científicas, pequeñas im-

[82] Para obtener la segunda fórmula, multiplique el numerador y denominador de $x_1$ por $-b - \sqrt{b^2 - 4ac}$ (y de manera similar para $x_2$).

precisiones pueden dar lugar a predicciones incorrectas, lo que podría socavar la validez de la investigación. En aplicaciones financieras, los errores de redondeo pueden dar lugar a importantes discrepancias monetarias, afectando todo, desde el análisis del mercado de valores hasta las transacciones bancarias. Incluso en aplicaciones cotidianas como la representación de gráficos, los errores de redondeo pueden causar artefactos visuales que restan valor a la experiencia del usuario.

Además, comprender las limitaciones de la aritmética de punto flotante puede ayudar a diseñar sistemas más robustos. Al anticipar dónde es probable que se produzcan errores de redondeo, los desarrolladores pueden implementar controles y equilibrios para detectar y corregir imprecisiones. Por ejemplo, la aritmética de intervalos puede proporcionar límites a los posibles valores de un cálculo, ofreciendo una manera de cuantificar la incertidumbre introducida por los errores de redondeo.

**Estrategias para Mitigar los Problemas de Precisión del Punto Flotante en el Desarrollo de Software** Los problemas de precisión del punto flotante son un desafío común en el desarrollo de Software y a menudo conducen a resultados inesperados y errores sutiles. Estos problemas surgen debido a las limitaciones inherentes a la representación de números reales en formato binario, lo que puede provocar errores de redondeo y pérdida de precisión. Para mitigar estos obstáculos, los desarrolladores deben emplear una variedad de estrategias que garanticen la precisión numérica y la confiabilidad en sus aplicaciones.

Una estrategia eficaz es utilizar tipos de datos de mayor precisión cuando sea necesario. Si bien los tipos de punto flotante estándar como "flotante" y "doble" son suficientes para muchas aplicaciones, es posible que no proporcionen la precisión requerida para cálculos más sensibles. En tales casos, el uso de tipos de precisión extendidos, como "long double" en C++ o bibliotecas de precisión arbitraria como GMP (Biblioteca aritmética de precisión múltiple GNU), puede reducir significativamente el riesgo de pérdida de precisión.

Sin embargo, es importante equilibrar la necesidad de precisión con consideraciones de rendimiento, ya que los tipos de mayor precisión pueden resultar costosos desde el punto de vista computacional. Otro enfoque consiste en implementar algoritmos numéricos que sean inherentemente más estables. Algunos algoritmos son más susceptibles a errores de redondeo que otros y elegir el algoritmo correcto puede marcar una diferencia significativa.

Por ejemplo, al resolver sistemas de ecuaciones lineales, utilizar métodos como la descomposición LU o la descomposición QR puede ser más estable que la eliminación gaussiana. Además, se pueden emplear técnicas de refinamiento iterativo para mejorar la precisión de la solución corrigiendo iterativamente los errores introducidos por la aritmética de punto flotante.

Los desarrolladores deben tener en cuenta el orden de las operaciones en sus cálculos. Las propiedades asociativas y distributivas de la aritmética no siempre se cumplen en la aritmética de punto flotante debido a errores de redondeo. Por lo tanto, reorganizar el orden de las operaciones a veces puede conducir a resultados más precisos. Por ejemplo, al sumar una gran cantidad de números de punto flotante, sumarlos en orden de magnitud ascendente puede minimizar la acumulación de errores de redondeo. Esta técnica, conocida como suma de Kahan, ayuda a preservar la precisión al compensar los pequeños errores que ocurren durante el proceso de suma.

Además de estas estrategias, es fundamental realizar pruebas y validaciones exhaustivas del Software numérico. Las pruebas unitarias deben diseñarse para cubrir una amplia gama de valores de entrada, incluidos casos extremos que probablemente expongan problemas de precisión. Comparar los resultados de los cálculos de punto flotante con soluciones analíticas conocidas o utilizar aritmética de mayor precisión como referencia puede ayudar a identificar discrepancias.

Además, se puede realizar un análisis de sensibilidad para evaluar cómo pequeños cambios en los valores de entrada afectan la salida, proporcionando información sobre la estabilidad y confiabilidad de los algoritmos numéricos.

Por último, la documentación y la comunicación desempeñan un papel fundamental a la hora de mitigar los problemas de precisión del punto flotante. Los desarrolladores deben documentar las limitaciones y suposiciones de sus algoritmos numéricos, así como cualquier fuente potencial de error. Esta información es invaluable para otros desarrolladores que necesiten mantener o ampliar el Software. Además, una comunicación clara con las partes interesadas sobre la precisión esperada del Software puede ayudar a gestionar las expectativas y evitar mal entendidos.

## 5.1 Aritmética de Punto Flotante IEEE

La aritmética de punto flotante es considerada un tema esotérico para muchas personas. Esto es sorprendente porque el punto flotante es omnipresente en los sistemas informáticos. Casi todos los lenguajes de programación tienen

un tipo de datos de punto flotante, i.e. los números no enteros como 1.2 ó $1e + 45$.

Un número de punto flotante de 64 bits (*double* en lenguaje C) tiene aproximadamente 16 dígitos decimales de precisión con un rango del orden de $1.7 \times 10^{-308}$ a $1.7 \times 10^{308}$ de acuerdo con el estándar 754 de IEEE -El estándar IEEE 754-2008 define varios tamaños de números de punto flotante: media precisión (binary16), precisión simple (binary32), precisión doble (binary64), precisión cuádruple (binary128), etc., cada uno con su propia especificación-, la implementación típica de punto flotante[83].

Una de las primeras cosas que uno encuentra con sorpresa cuando hace cálculos en una computadora es que por ejemplo, si usamos números muy grandes y seguimos incrementando su valor eventualmente el resultado será negativo ... ¿qué pasó? Esto se llama desbordamiento aritmético al intentar crear un valor numérico que está fuera del rango que puede representarse con un número dado de dígitos, ya sea mayor que el máximo o menor que el mínimo valor representable. Algo similar pasa al restar al menor número representable en la máquina, el resultado será positivo y se denomina subdesbordamiento.

Las computadoras, desde las PC hasta las supercomputadoras, tienen aceleradores de punto flotante; la mayoría de los compiladores deberán compilar algoritmos de punto flotante de vez en cuando y prácticamente todos los sistemas operativos deben responder a excepciones de punto flotante como el desbordamiento.

Desde ya hace mucho tiempo, los procesadores Intel x86 y todos los procesadores de las siguientes generaciones de todas las marcas admiten un formato de precisión extendido de 80 bits con un significado de 64 bits, que es compatible con el especificado en el estándar IEEE. Cuando un compilador usa este formato con registros de 80 bits para acumular sumas y productos internos, está trabajando efectivamente con un redondeo unitario de $2^{-64}$ en vez de $2^{-53}$ para precisión doble, dando límites de error más pequeños en un factor de hasta $2^{11} = 2048$.

¿Dieciséis lugares decimales es mucho? Casi ninguna cantidad medida se conoce con tanta precisión. Por ejemplo: en la constante en la ley de gravedad de Newton sólo se conoce con seis cifras significativas. En la carga

[83] Además, existe el número de 80 bits (*long double* en lenguaje C) que tiene 18 dígitos decimales de precisión con un rango del orden de $3.4 \times 10^{-4096}$ a $1.1 \times 10^{4096}$. Y no podemos olvidar, el número de 32 bits (*float* en lenguaje C) que tiene 14 dígitos decimales de precisión con un rango del orden de $3.4 \times 10^{-38}$ a $3.4 \times 10^{38}$.

de un electrón se conoce con 11 cifras significativas, mucha más precisión que la constante gravitacional de Newton, pero aún menos que un número de punto flotante[84].

Entonces, ¿cuándo no son suficientes 16 dígitos de precisión? Un área problemática es la resta. Las otras operaciones elementales (suma, multiplicación, división) son muy precisas. Siempre que no se presenten desbordamientos y subdesbordamientos, estas operaciones suelen producir resultados que son correctos hasta el último bit. Pero la resta puede ser desde exacta hasta completamente inexacta. Si dos números concuerdan con $n$ cifras, puede perder hasta $n$ cifras de precisión en su resta. Este problema puede aparecer inesperadamente en medio de otros cálculos.

¿Qué pasa con el desbordamiento o con el subdesbordamiento? ¿Cuándo se necesitan números mayores que $10^{308}$? No tan a menudo, pero en los cálculos de probabilidad, por ejemplo, se usan todo el tiempo a menos que se haga un uso inteligente.

Es común en probabilidad calcular un número de tamaño mediano que es el producto de un número astronómicamente grande y un número infinitesimalmente pequeño. El resultado final encaja perfectamente en una computadora, pero es posible que los números intermedios no se deban a un desbordamiento o un subdesbordamiento. Por ejemplo, el número máximo de punto flotante en la mayoría de las computadoras está entre 170 factorial y 171 factorial. Estos grandes factoriales aparecen frecuentemente en aplicaciones, a menudo en proporciones con otros grandes factoriales.

[84] ¿Cuántos dígitos de $\pi$ necesitamos?

- 3.1415 para diseñar los mejores motores.
- 3.1415926535 para obtener la circunferencia de la Tierra dentro de una fracción de pulgada.
- 3.141592653589793 para los cálculos de navegación interplanetaria de la NASA-JPL.
- 3.14159265358979323846264338327950288 42 para medir el radio del universo con una precisión igual al tamaño de un átomo de hidrógeno.

En el 2022 se calcularon los primero 100 billones de decimales del número $\pi$, para lograr este hito necesitó 157 días, 23 horas, 31 minutos y 7,651 segundos de cálculos, 515 Terabytes de almacenamiento y desplegar un abanico de tecnologías de computación en Compute Engine, un servicio de computación de Google Cloud.

**Anatomía de un Número de Punto Flotante** Un número de punto flotante de 64 bits codifica un número de la forma $\pm p \times 2^e$. El primer bit codifica el signo, 0 para números positivos y 1 para números negativos. Los siguientes 11 bits codifican el exponente $e$, y los últimos 52 bits codifican la precisión $p$.

El exponente se almacena con un sesgo de 1023. Es decir, los exponentes positivos y negativos se almacenan todos en un solo número positivo almacenando $e + 1023$ en lugar de almacenarlos directamente. Once bits pueden representar números enteros desde 0 hasta 2047. Restando el sesgo, esto corresponde a valores de $e$ de $-1023$ a $+1024$. Definimos $e_{min} = -1022$ y $e_{max} = +1023$. Los valores $e_{min} - 1$ y $e_{max} + 1$ están reservados para usos especiales.

Los números de punto flotante se almacenan normalmente en forma normalizada. En base 10, un número está en notación científica normalizada si el significando es $\geq 1$ y $< 10$. Por ejemplo, $3.14 \times 10^2$ está en forma norma-lizada, pero $0.314 \times 10^3$ y $31.4 \times 10^2$ no lo están.

En general, un número en base $\beta$ está en forma normalizada si tiene la forma $p \times \beta^e$ donde $1 \leq p < \beta$. Esto dice que para expresar un número binario, es decir, $\beta = 2$, el primer bit del significado de un número normalizado es siempre 1. Dado que este bit nunca cambia, no es necesario almacenarlo. Por lo tanto, podemos expresar 53 bits de precisión en 52 bits de almacenamiento. En lugar de almacenar el significado directamente, almacenamos $f$, la parte fraccionaria, donde el significado es de la forma $1.f$.

El esquema anterior no explica cómo almacenar 0. Es imposible especificar valores de $f$ y $e$ de modo que $1.f \times 2^e = 0$. El formato de punto flotante hace una excepción a las reglas establecidas anteriormente. Cuando $e = e_{min} - 1$ y $f = 0$, los bits se interpretan como 0. Cuando $e = e_{min} - 1$ y $f \neq 0$, el resultado es un número desnormalizado. Los bits se interpretan como $0.f \times 2^{e_{min}}$. En resumen, el exponente especial reservado debajo de $e_{min}$ se usa para representar 0 y números de punto flotante desnormalizados.

El exponente especial reservado arriba de $e_{max}$ se usa para representar $\infty$ y $NaN$. Si $e = e_{max} + 1$ y $f = 0$, los bits se interpretan como $\infty$. Pero si $e = e_{max} + 1$ y $f \neq 0$, los bits se interpretan como un $NaN$ o "no es un número" (Not a Number).

Dado que el exponente más grande es 1023 y el significativo más grande es $1.f$ donde $f$ tiene 52 unidades, el número de punto flotante (en C y C++, esta

constante se define como $DBL_MAX$ definido en <*float.h*>, en Python[85] en *sys.float_info)* más grande es $2^{1023}(2 - 2^{-52}) = 2^{1024} - 2^{971} \approx 2^{1024} \approx 1.8 \times 10^{308}$. Los números mayores que $2^{1024}$ i.e. $(2 - 2^{52})$ producen un desbordamiento conocido como Overflow.

Dado que el exponente más pequeño es $-1022$, el número normalizado positivo más pequeño es $1.0 \times 2^{-1022} \approx 2.2 \times 10^{-308}$ (en C y C++, esta constante se define como $DBL_MIN$ definido en <*float.h*>, en Python en *sys.float_info)*. Sin embargo, no es el número positivo más pequeño representable como un número de punto flotante, solo el número de punto flotante normalizado más pequeño. Los números más pequeños se pueden expresar en forma desnormalizada, aunque con una pérdida de significado. El número positivo desnormalizado más pequeño ocurre con $f$ que tiene 51 números 0 seguidos de un solo 1. Esto corresponde a $2^{-52} * 2^{-1022} = 2^{-1074} \approx 4.9 \times 10^{-324}$.

Los números que aparecen en los cálculos y tienen magnitud menor que $2^{-1023}$ i.e. $(1 + 2^{-52})$ producen un desbordamiento de la capacidad mínima o subdesbordamiento también conocido como Underflow y, por lo general se igualan a cero.

C y C++ da el nombre de $DBL_EPSILON$ al número positivo más pequeño $\epsilon$ tal que $1 + \epsilon \approx 1$, también llamado la precisión de la máquina. Dado que el significativo tiene 52 bits, está claro que $DBL_EPSILON = 2^{-52} \approx 2.2 \times 10^{-16}$. Por eso decimos que un número de punto flotante tiene entre 15 y 16 cifras significativas (decimales).

**Formatos de Punto Flotante** Se han propuesto varias representaciones diferentes de números reales, pero por mucho la más utilizada es la representación de punto flotante. Las representaciones de punto flotante tienen una base (que siempre se asume que es par) y una precisión $p$. Si $\beta = 10$ y $p = 3$, entonces el número 0.1 se representa como $1.00 \times 10^{-1}$. Si $\beta = 2$ y

[85]

```
import sys
print(sys.float_info)
```

```
sys.float_info(max=1.7976931348623157e+308, max_exp=1024,
max_10_exp=308, min=2.2250738585072014e-308, min_exp=-1021,
min_10_exp=-307, dig=15, mant_dig=53, epsilon=2.220446049250313e-
16, radix=2, rounds=1)
```

$p = 24$, entonces el número decimal 0.1 no se puede representar exactamente, pero es aproximadamente $1.10011001100110011001101 \times 2^{-4}$.

En general, un número de punto flotante se representará como $\pm d.dd...d \times \beta^e$, donde $d.dd...d$ se llama mantisa y tiene $p$ dígitos. De forma más precisa $\pm d_0.d_1d_2...d_{p-1} \times \beta^e$ representa el número

$$\pm \left(d_0 + d_1\beta^{-1} + ... + d_{p-1}\beta^{-(p-1)}\right)\beta^e, (0 \leq d_i < \beta). \tag{5.3}$$

El término número de punto flotante se utilizará para referirse a un número real que se puede representar exactamente en el formato en discusión. Otros dos parámetros asociados con las representaciones de punto flotante son los exponentes más grandes y más pequeños permitidos, $e_{max}$ y $e_{min}$. Dado que hay $\beta^p$ posibles significados, y $e_{max} - e_{min} + 1$ posibles exponentes, un número de punto flotante se puede codificar en

$$\left[\log_2\left(e_{max} - e_{min} + 1\right)\right] + \left[\log_2\left(\beta^p\right)\right] + 1$$

bits, donde el $+1$ final es para el bit de signo. La codificación precisa no es importante por ahora.

Hay dos razones por las que un número real podría no ser exactamente representable como un número de punto flotante. La situación más común se ilustra con el número decimal 0.1. Aunque tiene una representación decimal finita, en binario tiene una representación repetida infinita. Por lo tanto, cuando $\beta = 2$, el número 0.1 se encuentra estrictamente entre dos números de punto flotante y ninguno de ellos lo puede representar exactamente.

Una situación menos común es que un número real esté fuera de rango, es decir, su valor absoluto sea mayor que $\beta \times \beta^{e_{max}}$ o menor que $1.0 \times \beta^{e_{min}}$. La mayor parte de esta sección analiza cuestiones debidas a la primera razón.

Las representaciones de punto flotante no son necesariamente únicas. Por ejemplo, tanto $0.01 \times 10^1$ como $1.00 \times 10^{-1}$ representan 0.1. Si el dígito inicial es distinto de cero ($d_0 \neq 0$ en la Ec. (5.3)), se dice que la representación está normalizada. El número de punto flotante $1.00 \times 10^{-1}$ está normalizado, mientras que $0.01 \times 10^1$ no lo está.

**¿Cuándo es Exacta la Conversión de Base de Punto Flotante de Ida y Vuelta?** Suponga que almacenamos un número de punto flotante en la memoria, lo imprimimos en base 10 legible por humanos y lo regresamos a memoria. ¿Cuándo se puede recuperar exactamente el número original?

Suponga que comenzamos con la base $\beta$ con $p$ lugares de precisión y convertimos a la base $\gamma$ con $q$ lugares de precisión, redondeando al más cercano, luego volvemos a convertir a la base original $\beta$. El teorema de Matula dice que si no hay enteros positivos $i$ y $j$ tales que

$$\beta^i = \gamma^j$$

entonces una condición necesaria y suficiente para que la conversión de ida y vuelta sea exacta (suponiendo que no haya desbordamiento o subdesbordamiento) es que

$$\gamma^{q-1} > \beta^p.$$

En el caso de números de punto flotante (por ejemplo doble en C) tenemos $\beta = 2$ y $p = 53$. (ver Anatomía de un Número de Punto Flotante). Estamos imprimiendo a base $\gamma = 10$. Ninguna potencia positiva de 10 también es una potencia de 2, por lo que se mantiene la condición de Matula en las dos bases.

Si imprimimos $q = 17$ decimales, entonces

$$10^{16} > 2^{53}$$

por lo que la conversión de ida y vuelta será exacta si ambas conversiones se redondean al más cercano. Si $q$ es menor, algunas conversiones de ida y vuelta no serán exactas.

También puede verificar que para un número de punto flotante de precisión simple ($p$ = precisión de 24 bits) necesita $q = 9$ dígitos decimales, y para un número de precisión cuádruple (precisión de $p = 113$ bits) necesita $q = 36$ dígitos decimales[86].

Mirando hacia atrás en el teorema de Matula, claramente necesitamos

$$\gamma^q \geq \beta^p.$$

¿Por qué? Porque el lado derecho es el número de fracciones de base $\beta$ y el lado izquierdo es el número de fracciones de base $\gamma$. No puede tener un mapa

[86] El número de bits asignados para la parte fraccionaria de un número de punto flotante es 1 menos que la precisión: la cifra inicial es siempre 1, por lo que los formatos IEEE ahorran un bit al no almacenar el bit inicial, dejándolo implícito. Entonces, por ejemplo, un doble en C tiene una precisión de 53 bits, pero 52 bits de los 64 bits en un doble se asignan para almacenar la fracción.

uno a uno de un espacio más grande a un espacio más pequeño. Entonces, la desigualdad anterior es necesaria, pero no suficiente. Sin embargo, es casi suficiente. Solo necesitamos una base $\gamma$ con más cifras significativas, es decir, Matula nos dice

$$\gamma^{q-1} > \beta^p$$

es suficiente. En términos de base 2 y base 10, necesitamos al menos 16 decimales para representar 53 bits. Lo sorprendente es que un decimal más es suficiente para garantizar que las conversiones de ida y vuelta sean exactas. No es obvio a priori que cualquier número finito de decimales adicionales sea siempre suficiente, pero de hecho solo uno más es suficiente.

A continuación, se muestra un ejemplo para mostrar que el decimal adicional es necesario. Suponga que $p = 5$. Hay más números de 2 dígitos que números de 5 bits, pero si solo usamos dos dígitos, la conversión de base de ida y vuelta no siempre será exacta. Por ejemplo, el número 17/16 escrito en binario es $1.0001_{dos}$ y tiene cinco bits significativos. El equivalente decimal es $1.0625_{diez}$, que redondeado a dos dígitos significativos es $1.1_{diez}$. Pero el número binario más cercano a $1.1_{diez}$ con 5 bits significativos es $1.0010_{dos} = 1.125_{diez}$. En resumen, redondeando al más cercano da

$$1.0001_{dos} -> 1.1_{diez} -> 1.0010_{dos}$$

y así no volvemos al punto de partida.

**Error de Representación** se refiere al hecho de que la mayoría de las fracciones decimales no pueden representarse exactamente como fracciones binarias (en base 2). Esta es la razón principal de por qué muchos lenguajes de programación (Python, Perl, C, C++, Java, Fortran, y tantos otros) frecuentemente no mostrarán el número decimal exacto que esperas.

¿Por qué es eso? 1/10 no es representable exactamente como una fracción binaria. Casi todas las máquinas de hoy en día usan aritmética de punto flotante: IEEE-754, y casi todas las plataformas mapean los flotantes al «doble precisión» de IEEE-754. Estos «dobles» tienen 53 bits de precisión, por lo tanto en la entrada la computadora intenta convertir 0.1 a la fracción más cercana que puede de la forma $J/(2^N)$ donde $J$ es un entero que contiene exactamente 53 bits. Reescribiendo

$$1/10 \approx J/2^N$$

como

$$J \approx 2^N/10$$

y recordando que $J$ tiene exactamente 53 bits (es $\geq 2^{52}$ pero $< 2^{53}$), el mejor valor para $N$ es 56. O sea, 56 es el único valor para $N$ que deja $J$ con exactamente 53 bits. El mejor valor posible para $J$ es entonces el cociente redondeado, si usamos Python para los cálculos tenemos

```
>>> q, r = divmod(2**56, 10)
>>> r
6
```

Ya que el resto es más que la mitad de 10, la mejor aproximación se obtiene redondeándolo

```
>>> q+1
7205759403792794
```

Por lo tanto la mejor aproximación a 1/10 en doble precisión 754 es

$$7205759403792794/2**56$$

el dividir tanto el numerador como el denominador reduce la fracción a

$$3602879701896397/2**55$$

Notemos que como lo redondeamos, esto es un poquito más grande que 1/10; si no lo hubiéramos redondeado, el cociente hubiese sido un poquito menor que 1/10. ¡Pero no hay caso en que sea exactamente 1/10!

Entonces la computadora nunca «ve» 1/10: lo que ve es la fracción exacta de arriba, la mejor aproximación al flotante doble de 754 que puede obtener

```
>>> 0.1 * 2 ** 55
3602879701896397.0
```

Si multiplicamos esa fracción por $10^{55}$, podemos ver el valor hasta los 55 dígitos decimales

```
>>> 3602879701896397 * 10 ** 55 // 2 ** 55
1000000000000000055511151231257827021181583404541015625
```

lo que significa que el valor exacto almacenado en la computadora es igual al valor decimal

$$0.1000000000000000055511151231257827021181583404541015625.$$

en lugar de mostrar el valor decimal completo, muchos lenguajes, redondean el resultado a 17 dígitos significativos.

**Error de Redondeo** Comprimir infinitos números reales en un número finito de bits requiere una representación aproximada. Aunque hay un número infinito de enteros, en la mayoría de los programas el resultado de los cálculos de números enteros se puede almacenar en 32 bits o 64 bits. Por el contrario, dado cualquier número fijo de bits, la mayoría de los cálculos con números reales producirán cantidades que no se pueden representar exactamente usando tantos bits. Por lo tanto, el resultado de un cálculo de punto flotante a menudo debe redondearse para volver a ajustarse a su representación finita. Este error de redondeo es el rasgo característico del cálculo de punto flotante.

Dado que la mayoría de los cálculos de punto flotante tienen errores de redondeo de todos modos, ¿importa si las operaciones aritméticas básicas introducen un poco más de error de redondeo de lo necesario? Esa pregunta es un tema principal a lo largo de esta sección. La sección Dígitos de Guarda analiza los dígitos de protección, un medio para reducir el error al restar dos números cercanos. IBM consideró que los dígitos de guarda eran lo suficientemente importantes que en 1968 añadió un dígito de guarda al formato de doble precisión en la arquitectura System / 360 (la precisión simple ya tenía un dígito de guarda) y modernizó todas las máquinas existentes en el campo.

El estándar IEEE va más allá de sólo requerir el uso de un dígito de protección. Proporciona un algoritmo para la suma, resta, multiplicación, división y raíz cuadrada y requiere que las implementaciones produzcan el mismo resultado que ese algoritmo. Por lo tanto, cuando un programa se mueve de una máquina a otra, los resultados de las operaciones básicas serán los mismos en todos los bits si ambas máquinas admiten el estándar IEEE. Esto simplifica enormemente la portabilidad de programas. Otros usos de esta especificación precisa se dan en operaciones exactamente redondeadas.

**Error Relativo y Ulps** Dado que el error de redondeo es inherente al cálculo de punto flotante, es importante tener una forma de medir este error. Considere el formato de punto flotante con $\beta = 10$ y $p = 3$, que se utilizará

en esta sección. Si el resultado de un cálculo de punto flotante es $3.12 \times 10^{-2}$, y la respuesta cuando se calcula con precisión infinita es 0.0314, está claro que tiene un error de 2 unidades en el último lugar. De manera similar, si el número real 0.0314159 se representa como $3.14 \times 10^{-2}$, entonces tiene un error de 0.159 unidades en el último lugar.

En general, si el número de punto flotante $dd..d \times \beta^e$ se usa para representar $z$, entonces tiene un error de $|d.d...d - (z/\beta^e)|^{\beta^{p-1}}$ unidades en el último lugar. El término ulps se utilizará como abreviatura de (units in the last place) "unidades en último lugar". Si el resultado de un cálculo es el número de punto flotante más cercano al resultado correcto, aún podría tener un error de hasta 0.5 ulp.

Otra forma de medir la diferencia entre un número de punto flotante y el número real al que se aproxima es el error relativo, que es simplemente la diferencia entre los dos números divididos por el número real. Por ejemplo, el error relativo cometido al aproximar 3.14159 por $3.14 \times 10^0$ es $0.00159/3.141590 \approx 0005$.

Para calcular el error relativo que corresponde a .5 ulp, observe que cuando un número real es aproximado por el número de punto flotante más cercano posible $\overbrace{d.dd...dd}^{p} \times \beta^e$, el error puede ser tan grande como $\overbrace{0.00...00}^{p}\beta' \times \beta^e$, donde $\beta'$ es el dígito $\beta/2$, hay $p$ unidades en el significado del número de punto flotante y $p$ unidades de 0 en el significado del error. Este error es $((\beta/2)\beta^{-p}) \times \beta^e$. Dado que los números de la forma $d.dd...dd \times \beta^e$ tienen todos el mismo error absoluto, pero tienen valores que oscilan entre $\beta^e$ y $\beta \times \beta^e$, el error relativo varía entre $((\beta/2)\beta^{-p}) \times \beta^e/\beta^e$ y $((\beta/2)\beta^{-p}) \times \beta^e/\beta^{e+1}$. Eso es,

$$\frac{1}{2}\beta^{-p} \leq \frac{1}{2}ulp \leq \frac{\beta}{2}\beta^{-p} \tag{5.4}$$

En particular, el error relativo correspondiente a 0.5 ulp puede variar en un factor de $\beta$. Este factor se llama bamboleo. Estableciendo $\epsilon = (\beta/2)\beta^{-p}$ en el mayor de los límites en Ec. (5.4) anterior, podemos decir que cuando un número real se redondea al número de punto flotante más cercano, el error relativo siempre está limitado por $\epsilon$, que se conoce como épsilon de la máquina.

En el ejemplo anterior, el error relativo fue $0.00159/3.14159 \approx 0005$. Para evitar números tan pequeños, el error relativo normalmente se escribe como

factor multiplicado $\epsilon$, que en este caso es $\epsilon = (\beta/2)\beta^{-p} = 5(10)^{-3} = 0.005$. Por lo tanto, el error relativo se expresaría como $(0.00159/3.14159)/0.005)\epsilon \approx 0.1\epsilon$.

Para ilustrar la diferencia entre ulps y error relativo, considere el número real $x = 12.35$. Se aproxima por $\widetilde{x} = 1.24 \times 10^1$. El error es 0.5 ulps, el error relativo es $0.8\epsilon$. A continuación, considere el cálculo $8\widetilde{x}$. El valor exacto es $8x = 98.8$, mientras que el valor calculado es $8\widetilde{x} = 9.92 \times 10^1$. El error ahora es 4.0 ulps, pero el error relativo sigue siendo $0.8\epsilon$. El error medido en ulps es 8 veces mayor, aunque el error relativo es el mismo.

En general, cuando la base es $\beta$, un error relativo fijo expresado en ulps puede oscilar en un factor de hasta $\beta$. Y a la inversa, como muestra la Ec. (5.4) anterior, un error fijo de 0.5 ulps da como resultado un error relativo que puede oscilar por $\beta$.

La forma más natural de medir el error de redondeo es en ulps. Por ejemplo, el redondeo al número de punto flotante más cercano corresponde a un error menor o igual a 0.5 ulp. Sin embargo, al analizar el error de redondeo causado por varias fórmulas, el error relativo es una mejor medida. Dado que se puede sobrestimar el efecto de redondear al número de punto flotante más cercano por el factor de oscilación de $\beta$, las estimaciones de error de las fórmulas serán más estrictas en máquinas con una pequeña $\beta$.

Cuando sólo interesa el orden de magnitud del error de redondeo, ulps y $\epsilon$ pueden usarse indistintamente, ya que difieren como máximo en un factor de $\beta$. Por ejemplo, cuando un número de punto flotante tiene un error de $n$ ulps, eso significa que el número de dígitos contaminados es $\log_\beta n$. Si el error relativo en un cálculo es $n\epsilon$, entonces

$$\text{los dígitos contaminados} \approx \log_\beta n. \tag{5.5}$$

**Dígito de Guarda** Un método para calcular la diferencia entre dos números de punto flotante es calcular la diferencia exactamente y luego redondearla al número de punto flotante más cercano. Esto es muy caro si los operandos difieren mucho en tamaño. Suponiendo que $p = 3, 2.15 \times 10^{12} - 1.25 \times 10^{-5}$ se calcularía como

$x = 2.15 \times 10^{12}$

$y = 0.000000000000000125 \times 10^{12}$

$x - y = 2.149999999999999875 \times 10^{12}$

que se redondea a $2.15 \times 10^{12}$. En lugar de utilizar todos estos dígitos, el Hardware de punto flotante normalmente funciona con un número fijo

de dígitos. Suponga que el número de dígitos que se mantiene es $p$, y que cuando el operando más pequeño se desplaza hacia la derecha, los dígitos simplemente se descartan (en contraposición al redondeo). Entonces $2.15 \times 10^{12} - 1.25 \times 10^{-5}$ se convierte en

$x = 2.15 \times 10^{12}$
$y = 0.00 \times 10^{12}$
$x - y = 2.15 \times 10^{12}$

La respuesta es exactamente la misma que si la diferencia se hubiera calculado exactamente y luego se hubiera redondeado. Tome otro ejemplo: $10.1 - 9.93$. Esto se convierte en

$x = 1.01 \times 10^{1}$
$y = 0 - 99 \times 10^{1}$
$x - y = 0.02 \times 10^{1}$

La respuesta correcta es 0.17, por lo que la diferencia calculada está desviada en 30 ulps y es incorrecta en todos los dígitos. ¿Qué tan grave puede ser el error?.

**Teorema 1** *Usando un formato de punto flotante con parámetros $\beta$ y $p$, y calculando las diferencias usando $p$ dígitos, el error relativo del resultado puede ser tan grande como $\beta - 1$.*

Cuando $\beta = 2$, el error relativo puede ser tan grande como el resultado, y cuando $\beta = 10$, puede ser 9 veces mayor. O para decirlo de otra manera, cuando $\beta = 2$, la Ec. (5.5) muestra que el número de dígitos contaminados es $log_2(1/\epsilon) = log_2(2^p) = p$. Es decir, ¡todos los dígitos $p$ del resultado son incorrectos!. Suponga que se agrega un dígito adicional para protegerse contra esta situación (un dígito de guardia). Es decir, el número más pequeño se trunca a $p+1$ dígitos, y luego el resultado de la resta se redondea a $p$ dígitos. Con un dígito de guarda, el ejemplo anterior se convierte en

$x = 1.010 \times 10^{1}$
$y = 0.993 \times 10^{1}$
$x - y = 0.017 \times 10^{1}$

y la respuesta es exacta. Con un solo dígito de guarda, el error relativo del resultado puede ser mayor que $\epsilon$, como en $110 - 8.59$.

$x = 1.10 \times 10^{2}$
$y = 0.085 \times 10^{2}$
$x - y = 1.015 \times 10^{2}$

Esto se redondea a 102, en comparación con la respuesta correcta de 101.41, para un error relativo de 0.006, que es mayor que $\epsilon = 0.005$. En general, el error relativo del resultado puede ser solo un poco mayor que $\epsilon$. De forma más precisa:

**Teorema 2** *Si $x$ y $y$ son números de punto flotante en un formato con parámetros $\beta$ y $p$, y si la resta se realiza con $p+1$ dígitos (es decir, un dígito de guarda), entonces el error de redondeo relativo en el resultado es menor que $2\epsilon$.*

**Cancelación** La última sección se puede resumir diciendo que sin un dígito de guarda, el error relativo cometido al restar dos cantidades cercanas puede ser muy grande. En otras palabras, la evaluación de cualquier expresión que contenga una resta (o una suma de cantidades con signos opuestos) podría resultar en un error relativo tan grande que todos los dígitos carecen de significado (Teorema (1)). Al restar cantidades cercanas, los dígitos más significativos de los operandos coinciden y se cancelan entre sí. Hay dos tipos de cancelación: catastrófica y benigna.

La cancelación catastrófica ocurre cuando los operandos están sujetos a errores de redondeo. Por ejemplo, en la fórmula cuadrática, aparece la expresión $b^2 - 4ac$. Las cantidades $b^2$ y $4ac$ están sujetas a errores de redondeo ya que son el resultado de multiplicaciones de punto flotante. Suponga que están redondeados al número de punto flotante más cercano y, por lo tanto, tienen una precisión de 0.5 ulp. Cuando se restan, la cancelación puede hacer que muchos de los dígitos precisos desaparezcan, dejando principalmente dígitos contaminados por errores de redondeo. Por tanto, la diferencia puede tener un error de muchos ulps. Por ejemplo, considere $b = 3.34, a = 1.22$ y $c = 2.28$. El valor exacto de $b^2 - 4ac$ es 0.0292. Pero $b^2$ se redondea a 11.2 y $4ac$ se redondea a 11.1, por lo que la respuesta final es 0.1, que es un error de 70 ulps, aunque $11.2 - 11.1$ es exactamente igual a 0.1. La resta no introdujo ningún error, sino que expuso el error introducido en las multiplicaciones anteriores.

La cancelación benigna ocurre al restar cantidades exactamente conocidas. Si $x$ e $y$ no tienen error de redondeo, entonces, según el Teorema (2), si la resta se realiza con un dígito de guarda, la diferencia $x - y$ tiene un error relativo muy pequeño (menos de $2\epsilon$).

Una fórmula que presenta una cancelación catastrófica a veces se puede reorganizar para eliminar el problema. Considere nuevamente la fórmula

cuadrática

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \text{ y } x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \tag{5.6}$$

Cuando $b^2 \gg 4ac$, entonces $b^2 - 4ac$ no implica una cancelación y

$$\sqrt{b^2 - 4ac} \approx |b| .$$

Pero la otra suma (resta) en una de las fórmulas tendrá una cancelación catastrófica. Para evitar esto, multiplique el numerador y denominador de $x_1$ por $-b - \sqrt{b^2 - 4ac}$ (y de manera similar para $x_2$) para obtener

$$x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}} \text{ y } x_2 = \frac{-2c}{b - \sqrt{b^2 - 4ac}} \tag{5.7}$$

Si $b^2 \gg ac$ y $b > 0$, entonces calcular $x_1$ usando la Ec. (5.6) implicará una cancelación. Por lo tanto, use la Ec. (5.7) para calcular $x_1$ y (5.6) para $x_2$. Por otro lado, si $b < 0$, use (5.6) para calcular $x_1$ y (5.7) para $x_2$.

La expresión $x^2 - y^2$ es otra fórmula que presenta una cancelación catastrófica. Es más exacto evaluarlo como

$$(x - y)(x + y)$$

A diferencia de la fórmula cuadrática, esta forma mejorada todavía tiene una resta, pero es una cancelación benigna de cantidades sin error de redondeo, no catastrófica. Según el Teorema (2), el error relativo en $x - y$ es como máximo $2\epsilon$. Lo mismo ocurre con $x + y$. Multiplicar dos cantidades con un pequeño error relativo da como resultado un producto con un pequeño error relativo.

**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 Ec. (5.6) o de forma alternativa con el algoritmo que se obtiene mediante la racionalización del numerador Ec. (5.7).

Otro algoritmo que podemos implementar es el método de Newton-Raphson[87] para buscar raíces, que en su forma iterativa está dado por

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

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** Para resolver el problema, usamos por ejemplo el siguiente código en Python usando programación procedimental:

```
import math
def f(x, a, b, c):
  """ Evalua la Funcion cuadratica """
  return (x * x * a + x * b + c);
def df(x, a, b):
  """ Evalua la derivada de la funcion cuadratica """
  return (2.0 * x * a + b);
def evalua(x, a, b, c):
  """ Evalua el valor X en la función cuadratica """
  print('Raiz (%1.16f), evaluacion raiz: %1.16e' % (x, f(x, a, b, c)))
def metodoNewtonRapson(x, ni, a, b, c):
  """ Metodo Newton-Raphson x = x - f(x)/f'(x) """
  for i in range(ni):
    x = x - (f(x, a, b, c) / df(x, a, b))
  return x
def raices(A, B, C):
  """ Calculo de raices """
```

[87] 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 $f''(x)$.

```
    if A == 0.0:
      print("No es una ecuacion cuadratica")
      exit(1)
    # Calculo del discriminante
    d = B * B - 4.0 * A * C
    # Raices reales
    if d >= 0.0:
      print('\nPolinomio (%f) X^2 + (%f )X + (%f) = 0\n' % (A, B, C))
      print('\nChicharronera 1')
      X1 = (-B + math.sqrt(d)) / (2.0 * A)
      X2 = (-B - math.sqrt(d)) / (2.0 * A)
      evalua(X1, A, B, C)
      evalua(X2, A, B, C)
      print('\nChicharronera 2')
      X1 = (-2.0 * C) / (B + math.sqrt(d))
      X2 = (-2.0 * C) / (B - math.sqrt(d))
      evalua(X1, A, B, C)
      evalua(X2, A, B, C)
      # Metodo Newton-Raphson
      print("\n\nMetodo Newton-Raphson")
      x = X1 - 1.0;
      print("\nValor inicial aproximado de X1 = %1.16f" % x)
      x = metodoNewtonRapson(x, 6, A, B, C)
      evalua(x, A, B, C);
      x = X2 - 1.0;
      print("\nValor inicial aproximado de X2 = %1.16f" % x);
      x = metodoNewtonRapson(x, 6, A, B, C)
      evalua(x, A, B, C)
      print("\n")
    else:
      # Raices complejas
      print("Raices Complejas ...")
if __name__ == '__main__':
    raices(1.0, 4.0, 1.0)
```

y generan la siguiente salida:

```
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. Otras implementaciones computacionales se pueden consultar en las ligas, en las cuales se usan distintos lenguajes (C, C++, Java y Python) y diferentes paradigmas de programación ( secuencial, procedimental y orientada a objetos).

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.

## 5.2 Trabajando con Punto Flotante

Hay varias trampas en las que incluso los programadores muy experimentados caen cuando escriben código que depende de la aritmética de punto flotante. En esta sección explicamos algunas cosas a tener en cuenta al trabajar con números de punto flotante, es decir, tipos de datos: float (32 bits), double (64 bits) o long double (80 bits).

**Problemas de Precisión** Como ya vimos en las secciones precedentes, los números de punto flotante se representan en el Hardware de la computadora en fracciones en base 2 (binario). Por ejemplo, la fracción decimal

$$0.125$$

tiene el valor $1/10+2/100+5/1000$, y de la misma manera la fracción binaria

$$0.001$$

tiene el valor $0/2 + 0/4 + 1/8$. Estas dos fracciones tienen valores idénticos, la única diferencia real es que la primera está escrita en notación fraccional en base 10 y la segunda en base 2.

Desafortunadamente, la mayoría de las fracciones decimales no pueden representarse exactamente como fracciones binarias. Como consecuencia, en general los números de punto flotante decimal que usamos en la computadora son sólo aproximados por los números de punto flotante binario que realmente se guardan en la máquina.

El problema es más fácil de entender primero en base 10. Consideremos la fracción 1/3. Podemos aproximarla como una fracción de base 10 como: 0.3 o, mejor: 0.33 o, mejor: 0.333 y así. No importa cuántos dígitos desees escribir, el resultado nunca será exactamente 1/3, pero será una aproximación cada vez mejor de 1/3.

De la misma manera, no importa cuántos dígitos en base 2 quieras usar, el valor decimal 0.1 no puede representarse exactamente como una fracción en base 2. En base 2, 1/10 es la siguiente fracción que se repite infinitamente

$$0.0001100110011001100110011001100110011001100110011...$$

si nos detenemos en cualquier número finito de bits, y tendremos una aproximación. En la mayoría de las máquinas hoy en día, los double se aproximan

usando una fracción binaria con el numerador usando los primeros 53 bits con el bit más significativo y el denominador como una potencia de dos. En el caso de 1/10, la fracción binaria es

$$3602879701896397/2^{55}$$

que está cerca pero no es exactamente el valor verdadero de 1/10.

La mayoría de los usuarios no son conscientes de esta aproximación por la forma en que se muestran los valores. Varios lenguajes de programación como C, C++, Java y Python solamente muestra una aproximación decimal al valor verdadero decimal de la aproximación binaria almacenada por la máquina. En la mayoría de las máquinas, si fuéramos a imprimir el verdadero valor decimal de la aproximación binaria almacenada para 0.1, debería mostrar

$$0.1000000000000000055511151231257827021181583404541015625$$

esos son más dígitos que lo que la mayoría de la gente encuentra útil, por lo que los lenguajes de programación mantiene manejable la cantidad de dígitos al mostrar en su lugar un valor redondeado

$$1/10$$

como

$$0.1$$

sólo hay que recordar que, a pesar de que el valor mostrado resulta ser exactamente 1/10, el valor almacenado realmente es la fracción binaria más cercana posible. Esto queda de manifiesto cuando hacemos

$$0.1 + 0.1 + 0.1 \text{ ó } 0.1 * 3 \text{ ó } 0.1 + 0.2$$[88]

obtendremos

$$0.30000000000000004$$

[88] Para el caso de $0.1 + 0.2$ podemos hacer un programa que nos de 0.3, usando:

```
double x = 0.1;
double y = 0.1;
double z = (x*10.0 + y*10.0) / 10.0
```

que es distinto al 0.3 que esperábamos.

Es de hacer notar que hay varios números decimales que comparten la misma fracción binaria más aproximada. Por ejemplo, los números

$$0.1, 0.10000000000000001 \text{ y}$$

$$0.1000000000000000055511151231257827021181583404541015625$$

son todos aproximados por $3602879701896397/2^{55}$.

Notemos que esta es la verdadera naturaleza del punto flotante binario: no es un error del lenguaje de programación y tampoco es un error en tu código. Verás lo mismo en todos los lenguajes que soportan la aritmética de punto flotante de tu Hardware (a pesar de que en algunos lenguajes por omisión no muestren la diferencia, o no lo hagan en todos los modos de salida). Para una salida más elegante, quizás quieras usar el formateo de cadenas de texto para generar un número limitado de dígitos significativos.

Ejemplo, el número decimal 9.2 se puede expresar exactamente como una relación de dos enteros decimales 92/10, los cuales se pueden expresar exactamente en binario como $0b1011100/0b1010$. Sin embargo, la misma proporción almacenada como un número de punto flotante nunca es exactamente igual a 9.2:

usando 32 bits obtenemos: 9.1999999809265136
usando 64 bits obtenemos: 9.199999999999999928945

ya que se guarda como la fracción: $5179139571476070/2^{49}$.

Otro ejemplo, si tomamos el caso del número 0.02 y vemos su representación en el lenguaje de programación Python, tenemos que:

```
import decimal
print(decimal.Decimal(0.02))
```

y el resultado es:

0.020000000000000000416333634234433702658861875534057617187 5

Además, tenemos la no representabilidad de $\pi$ (y $\pi/2$), esto significa que un intento de cálculo de $tan(\pi/2)$ no producirá un resultado de infinito, ni

siquiera se desbordará en los formatos habituales de punto flotante. Simplemente no es posible que el Hardware de punto flotante estándar intente calcular $tan(\pi/2)$, porque $\pi/2$ no se puede representar exactamente. Este cálculo en C:

```
doble pi =3.14159265358979323846264338327 95;
tan (pi / 2.0);
```

dará un resultado de $1.633123935319537e+16$. En precisión simple usando tanf, el resultado será $-22877332.0$.

De la misma manera, un intento de cálculo de $sin(\pi)$ no arrojará cero. El resultado será $1.2246467991473532e-16$ en precisión doble.

Si bien la suma y la multiplicación de punto flotante son conmutativas ( $a + b = b + a$ y $a \times b = b \times a$), no son necesariamente asociativas . Es decir, $(a + b) + c$ no es necesariamente igual a $a + (b + c)$. Usando aritmética decimal significativa de 7 dígitos:

$$a = 1234.567, b = 45.67834, c = 0.0004$$

$(a + b) + c$ :

$1234.567 + 45.67834$ 1280.24534 se redondea a 1280.245
$1280.245 + 0.0004$ 1280.2454 se redondea a 1280.245

$a + (b + c)$ :

$45.67834 + 0.0004$ 45.67874
$1234.567 + 45.6787$ 1280.24574 se redondea a 1280.246

Tampoco son necesariamente distributivos. Es decir $(a + b) \times c$ puede no ser lo mismo que $a \times c + b \times c$:

$1234.567*3.333333 = 4115.223$, $1.234567*3.333333 = 4.115223$, $4115.223 + 4.115223 = 4119.338$

pero

$1234.567+1.234567 = 1235.802$, $1235.802*3.333333 = 4119.340$

**Algunos Trucos** Supongamos que necesitamos hacer el siguiente cálculo:

$$4.56 * 100$$

la respuesta es:

$$455.9999999999994$$

que no es la esperada. ¿Qué podemos hacer para obtener lo que esperamos?. Por ejemplo, con Python, podemos usar:

```
from sympy import *
print(nsimplify(4.56 * 100, tolerance=1e-1))
```

que nos dará el 456 esperado.

SymPy es un paquete matemático simbólico para Python, y nsimplify toma un número de punto flotante y trata de simplificarlo como una fracción con un denominador pequeño, la raíz cuadrada de un entero pequeño, una expresión que involucra constantes famosas, etc.

Por ejemplo, supongamos que algún cálculo arrojó 4.242640687119286 y sospechamos que hay algo especial en ese número. Así es como puede probar de dónde vino:

```
from sympy import *
print(nsimplify(4.242640687119286))
```

que nos arrojará:

$$3 * sqrt(2)$$

Tal vez hagamos un cálculo numérico y encontremos una expresión simple para el resultado y eso sugiera una solución analítica. Creo que una aplicación más común de nsimplify podría ser ayudarte a recordar fórmulas medio olvidadas. Por ejemplo, quizás estés oxidado con tus identidades trigonométricas, pero recuerdas que cos(p/6) es algo especial.

```
from sympy import *
print(nsimplify(cos(pi/6)))
```

que nos entregará:

$$sqrt(3)/2$$

O, para tomar un ejemplo más avanzado, supongamos que recordamos vagamente que la función gamma toma valores reconocibles en valores semienteros, pero no recordamos exactamente cómo. Tal vez algo relacionado con $\pi$ o $e$. Puede sugerir que nsimplify incluya expresiones con $\pi$ y $e$ en su búsqueda.

```
from sympy import *
print(nsimplify(gamma(3.5), constants=[pi, E]))
```

obteniendo:

$$15 * sqrt(pi)/8$$

También podemos darle a nsimplify una tolerancia, pidiéndole que encuentre una representación simple dentro de una vecindad del número. Por ejemplo, aquí hay una manera de encontrar aproximaciones a $\pi$.

```
from sympy import *
print(nsimplify(pi, tolerance=1e-5))
```

obteniendo:

$$355/113$$

con una tolerancia más amplia, devolverá una aproximación más simple.

```
from sympy import *
print(nsimplify(pi, tolerance=1e-2))
```

obteneindo:

$$22/7$$

finalmente, aquí hay una aproximación de mayor precisión a $\pi$ que no es exactamente simple:

```
from sympy import *
print(nsimplify(pi, tolerance=1e-7))
```

obteniendo:

$$exp(141/895 + sqrt(780631)/895)$$

**Seno de un Googol** ¿Cómo se evalúa el seno de un número grande en aritmética de punto flotante? ¿Qué significa el resultado?.

**Seno de un billón** Comencemos por encontrar el seno de un billón ($10^{12}$) usando aritmética de punto flotante. Hay un par de maneras de pensar en esto. El número de punto flotante $t = 1.0e12$ solo se puede representar con 15 o 16 cifras decimales significativas (para ser precisos, 53 bits[89]), por lo que podría considerarlo como un representante del intervalo de números que tienen todos la misma representación de punto flotante. Cualquier resultado que sea seno de un número en ese intervalo debe considerarse correcto.

Otra forma de ver esto es decir que $t$ se puede representar exactamente -su representación binaria requiere 42 bits y tenemos 53 bits de precisión significativa disponibles-, por lo que esperamos $sin(t)$ devolver el seno de exactamente un billón, con precisión total.

Resulta que la aritmética del IEEE hace lo último, calculando $sin(1e12)$ correctamente hasta 16 dígitos.

Aquí está el resultado en Python.

```
sin(1.0e12)
-0.6112387023768895
```

y verificado por Mathematica calculando el valor con 20 decimales

```
In:= N[Sin[10^12], 20]
out= -0.61123870237688949819
```

**Reducción de alcance** tenga en cuenta que el resultado anterior no es el que obtendría si primero tomara el resto al dividir por $2\pi$ y luego tomará el seno.

```
sin(1.0e12 % (2*pi))
-0,6112078499756778
```

[89] Un doble IEEE 754 tiene 52 bits significativos, pero estos bits pueden representar 53 bits de precisión porque el primer bit de la parte fraccionaria es siempre 1, por lo que no es necesario representarlo.

Esto tiene sentido. El resultado de dividir un billón por la representación de punto flotante de $2\pi$, 159154943091.89536, es correcto con precisión de punto flotante total. Pero la mayor parte de esa precisión está en la parte entera. La parte fraccionaria solo tiene cinco dígitos de precisión, por lo que debemos esperar que el resultado anterior sea correcto hasta un máximo de cinco dígitos. De hecho, tiene una precisión de cuatro dígitos.

Cuando su procesador calcula, $sin(1e12)$ no toma ingenuamente el resto por $2\pi$ y luego calcula el seno. Si así fuera, obtendríamos cuatro cifras significativas en lugar de 16.

Empezamos diciendo que había dos maneras de ver el problema y, según la primera, devolver sólo cuatro cifras significativas sería bastante razonable. Si pensamos en el valor $t$ como una cantidad medida, medida con la precisión de la aritmética de punto flotante (aunque casi nada se puede medir con tanta precisión), entonces todo lo que deberíamos esperar sería cuatro cifras significativas. Pero las personas que diseñaron la implementación de la función seno en su procesador hicieron más de lo que se esperaba que hicieran, encontrando el seno de un billón con total precisión. Lo hacen utilizando un algoritmo de reducción de rango que conserva mucha más precisión que hacer una división ingenuamente.

**¿Seno de un Googol?** ¿Qué pasa si intentamos tomar el seno de un número ridículamente grande como un Googol ($10^{100}$)? Esto no funcionará porque un Googol no se puede representar exactamente como un número de punto flotante (es decir, como un doble IEEE 754). No es demasiado grande; Los números de punto flotante pueden ser tan grandes como alrededor de $10^{308}$. El problema es que un número tan grande no se puede representar con total precisión. Pero podemos representar $2^{333}$ exactamente, y un Googol está entre $2^{332}$ y $2^{333}$. Y sorprendentemente, la función sinusoidal de Python (o más bien la función sinusoidal integrada en el procesador AMD de mi computadora) devuelve el resultado correcto con total precisión.

```
sin(2**333)
0.9731320373846353
```

¿Cómo sabemos que esto es correcto? Lo verifiqué en Mathematica:

```
In:= sin[2.0^333]
Out= 0.9731320373846353
```

¿Cómo sabemos que Mathematica tiene razón? Podemos hacer una reducción de rango ingenua usando precisión extendida, digamos 200 decimales.

```
In:= p = N[Pi, 200]
In:= theta = x - IntegerPart[x/ (2 p)] 2 p
Out= 1.80312861783346995593841966 89...
```

y verificar que el seno del valor reducido del rango sea correcto.

```
In:= Sin[theta]
Out= 0.97313203738463526...
```

**Interpretación Aritmética de Intervalos** debido a que los números de punto flotante tienen 53 bits de precisión, todos los números reales entre $2^{56} - 2^2$ y $2^{56} + 2^2$ se representan como el número de punto flotante $2^{56}$. Este es un rango de ancho 8, más ancho que $2\pi$, por lo que los senos de los números en este rango cubren los posibles valores de seno, es decir, $[-1, 1]$. Entonces, si piensa en un número de punto flotante como una muestra del conjunto de todos los números reales con la misma representación binaria, cada valor posible de seno es un valor de retorno válido para números mayores que $2^{56}$. Pero si piensa que un número de punto flotante se representa exactamente a sí mismo, entonces tiene sentido preguntar por el seno de números como $2^{333}$ y mayores, hasta los límites de la representación de punto flotante[90].

**No Pruebes por la Igualdad** Cuando se usa Punto Flotante no es recomendable escribir código como el siguiente[91]:

```
double x;
double y;
...
if (x == y) {...}
```

[90] El mayor exponente de un doble IEEE es 1023, y el mayor significado es $2 - 2^{-53}$ (es decir, todos los unos), por lo que el mayor valor posible de un doble es $(2^{53} - 1)2^{1024-53}$ y de hecho la expresión de Python sin((2**53 - 1)*2**(1024-53)) devuelve el valor correcto con [16] cifras significativas.

[91] Sin pérdida de generalidad usamos algún lenguaje de programación en particular para mostrar los ejemplos, pero esto pasa en todos los lenguajes que usan operaciones de Punto Flotante.

La mayoría de las operaciones de punto flotante implican al menos una pequeña pérdida de precisión y, por lo tanto, incluso si dos números son iguales para todos los fines prácticos, es posible que no sean exactamente iguales hasta el último bit, por lo que es probable que la prueba de igualdad falle. Por ejemplo:

```
double x = 10;
double y = sqrt(x);
y *= y;
if (x == y)
cout << "La reaiz cuadrada es exacta\n";
else
cout << x-y << "\n";
```

el código imprime: $-1.778636e-015$, aunque en teoría, elevar al cuadrado debería deshacer una raíz cuadrada, la operación de ida y vuelta es ligeramente inexacta. En la mayoría de los casos, la prueba de igualdad anterior debe escribirse de la siguiente manera:

```
double tolerancia = ...
if (fabs(x - y) < tolerancia) {...}
```

Aquí la tolerancia es un umbral que define lo que está "lo suficientemente cerca" para la igualdad. Esto plantea la pregunta de qué tan cerca está lo suficientemente cerca. Esto no puede responderse en abstracto; tienes que saber algo sobre tu problema particular para saber qué tan cerca está lo suficientemente cerca en tu contexto.

Por ejemplo: ¿hay alguna garantía de que la raíz cuadrada de un cuadrado perfecto sea devuelta exactamente?, por ejemplo si hago $sqrt(81.0) == 9.0$, por lo visto anteriormente, la respuesta es no, pero podríamos cambiar la pregunta por $9.0 * 9.0 == 81.0$, esto funcionará siempre que el cuadrado esté dentro de los límites de la magnitud del punto flotante.

Por otro lado, es posible que las expectativas de las matemáticas no se cumplan en el campo del cálculo de punto flotante. Por ejemplo, se sabe que

$$(x + y)(x - y) = x^2 + y^2$$

y esta otra

$$\sin^2\theta + \cos^2\theta = 1$$

sin embargo, no se puede confiar en estos hechos cuando las cantidades involucradas son el resultado de un cálculo de punto flotante. Además, el error de redondeo puede afectar la convergencia y precisión de los procedimientos numéricos iterativos.

Aún más y sin pérdida de generalidad, si comparamos usando el lenguaje de programación Python:

```
print(0.1 + 0.2 == 0.3)
print(0.2 + 0.2 +0.2 == 0.6)
print(1.2 + 2.4 + 3.6 == 7.2)
print(0.1 + 0.2 <= 0.3)
```

en todos los casos dará un resultado de falso, además:

```
print(10.4 + 20.8 > 31.2)
print(0.8 - 0.1 > 0.7)
```

el resultado será verdadero. Por ello es siempre conveniente ver con que números trabajamos usando algo como:

```
print(format(0.1, ".17g"))
print(format(0.2, ".17g"))
print(format(0.3, ".17g"))
```

así cuando sumemos $0.1 + 0.2$, podemos ver el verdadero resultado:

```
print(print(format(0.1 + 0.2, ".17g"))
```

Teniendo esto en cuenta podemos comparar usando:

```
import math
print(math.isclose(0.1 + 0.2, 0.3)
```

nos dará la respuesta esperada. Podemos ajustar la tolerancia relativa usando:

```
import math
print(math.isclose(0.1 + 0.2, 0.3, rel_tol = 1e-20)
```

que en este caso nos dirá que es falso pues no son iguales en los 20 primeros dígitos solicitados.

Para las comparaciones >= o <=, por ejemplo para $a + b <= c$ se debe usar:

```
a, b , c = 0.1, 0.2, 0.3
print(math.isclose(a + b, c) or (a +c < c))
```

**¿Cómo Compara Python los Flotantes y los Enteros?** veamos los siguientes ejemplos:

```
>>> 9007199254740992 == 9007199254740992.0
True
>>> 9007199254740993 == 9007199254740993.0
False
>>> 9007199254740994 == 9007199254740994.0
True
```

¿Qué fue lo que salió mal?, para desmenuzar lo que pasó, veamos primero:

**Representación IEEE-754 de** 9007199254740992.0 comencemos con el valor del primer escenario: 9007199254740992.0, resulta que este número es en realidad es $2^{53}$. Eso significa que en la representación binaria sería 1 seguido de 53 ceros:

$$10000000000000000000000000000000000000000000000000000.$$

la forma normalizada será $1.0 * 2^{53}$, que se obtendrá desplazando los bits 53 lugares a la derecha, lo que nos dará el exponente 53.

Además, recordemos que la mantisa tiene solo 52 bits de ancho y estamos desplazando nuestro valor 53 bits hacia la derecha, lo que significa que se perderá el bit menos significativo (LSB) de nuestro valor original. Sin embargo, como todos los bits aquí son 0, no hay ningún daño y aún podemos representar 9007199254740992.0 con precisión. Resumamos los componentes de 9007199254740992.0:

Bit de signo: 0

Exponente: $53 + 1023$ (bias) = 1076 (10000110100 en binario)

Mantisa: 0000000000000000000000000000000000000000000000000000

Representación IEEE-754:

0100001101000000000000000000000000000000000000000000000000000000

**Representación IEEE-754 de** 9007199254740993.0 el valor del segundo escenario de prueba es 9007199254740993.0 el que es uno mayor que el valor anterior. Su representación binaria se puede obtener simplemente sumando 1 a la representación binaria del valor anterior:

$$10000000000000000000000000000000000000000000000001.0$$

nuevamente, para convertir esto a la forma normalizada, tendremos que desplazarlo a la derecha 53 bits, lo que nos dará el exponente 53.

Sin embargo, la mantisa tiene sólo 52 bits de ancho. Cuando desplazamos nuestro valor a la derecha en 53 bits, el bit menos significativo (LSB) con el valor 1 se perderá, lo que dará como resultado que la mantisa sea todo ceros. Esto conduce a los siguientes componentes:

Bit de signo: 0

Exponente: $53 + 1023$ (bias) $= 1076$ (10000110100 en binario)

Mantisa: 0000000000000000000000000000000000000000000000000000

Representación IEEE-754:

0100001101000000000000000000000000000000000000000000000000000000

Observe que la representación IEEE-754 de 9007199254740993.0 es la misma que la de 9007199254740992.0. Esto significa que en su representación en memoria del número 9007199254740993.0 en realidad se representa como 9007199254740992.0.

Esto explica por qué Python da el resultado para

$$9007199254740993 == 9007199254740993.0$$

False, porque lo ve como una comparación entre

$$9007199254740993 \text{ y } 9007199254740992.0.$$

**Representación IEEE-754 de** 9007199254740994.0 El número en el tercer y último escenario de prueba es 9007199254740994.0, que es 1 más que el valor anterior. Su representación binaria se puede obtener sumando 1 a la representación binaria de 9007199254740993.0, dándonos:

$$10000000000000000000000000000000000000000000000010.0$$

esto también tiene 54 bits antes del punto binario y requiere un desplazamiento a la derecha de 53 bits para convertirlo a la forma normalizada, lo que nos da el exponente 53.

Observe que esta vez, el segundo bit menos significativo (LSB) tiene el valor 1. Por lo tanto, cuando desplazamos este número a la derecha 53 bits, se convierte en el LSB de la mantisa (que tiene 52 bits de ancho). Los componentes se ven así:

Bit de signo: 0

Exponente: $53 + 1023$ (bias) $= 1076$ (10000110100 en binario)

Mantisa: 0000000000000000000000000000000000000000000000000001

Representación IEEE-754:

0100001101000000000000000000000000000000000000000000000000000001

A diferencia de 9007199254740993.0, la representación IEEE-754 del número 9007199254740994.0 puede representarlo exactamente sin pérdida de precisión. Por tanto, el resultado de

$$9007199254740994 == 9007199254740994.0$$

es True en Python.

El estándar IEEE-754 y la aritmética de punto flotante son inherentemente complejos y comparar números de punto flotante no es sencillo. Lenguajes como C y Java implementan la promoción de tipos implícita, convirtiendo números enteros en dobles y comparándolos poco a poco. Sin embargo, esto aún puede dar lugar a resultados inesperados debido a la pérdida de precisión.

Python es único a este respecto. Tiene números enteros de precisión infinita, lo que hace que la promoción de tipos sea inviable en muchas situaciones. En consecuencia, Python utiliza su algoritmo especializado para comparar estos números, que a su vez tiene casos extremos.

Ya sea que esté utilizando C, Java, Python u otro lenguaje, es recomendable utilizar funciones de biblioteca para comparar valores de punto flotante en lugar de realizar comparaciones directas para evitar posibles errores.

**Cuando Cómputo de Alto Rendimiento no es Alto Rendimiento**
A todo el mundo le importa que los códigos se ejecuten rápidamente en sus computadoras. Las mejoras de Hardware de las últimas décadas lo han hecho posible. Pero, ¿qué tan bien estamos aprovechando las aceleraciones del Hardware?

Considere estos dos ejemplos de código C++. Supongamos aquí n = 10000000.

```
void sub(int* a, int* b) {
    for (int i=0; i<n; ++i)
        a[i] = i + 1;
    for (int i=0; i<n; ++i)
        b[i] = a[i];
}

void sub(int* a, int* b) {
    for (int i=0; i<n; ++i) {
        const int j = i + 1;
        a[i] = j;
        b[i] = j;
    }
}
```

¿Cuál corre más rápido? Ambos son simples y dan resultados idénticos (suponiendo que no haya alias). Sin embargo, en las arquitecturas modernas, dependiendo de la configuración de compilación, una generalmente se ejecutará significativamente más rápido que la otra.

En particular, se esperaría que el fragmento 2 se ejecutara más rápido que el fragmento 1. En el fragmento 1, los elementos de la matriz "a", que es demasiado grande para almacenarse en caché, deben recuperarse de la memoria después de escribirse, pero esto no es necesario para Fragmento 2. La tendencia durante más de dos décadas ha sido que la velocidad de computación de los sistemas recién entregados crezca mucho más rápido que la velocidad de la memoria, y la disparidad es extrema hoy en día. El rendimiento de estos núcleos depende casi por completo de la velocidad del ancho de banda de la memoria. Así, el fragmento 2, una versión de bucle fusionado del fragmento 1, mejora la velocidad al reducir el acceso a la memoria principal.

Es poco probable que bibliotecas como C++ STL ayuden, ya que esta operación es demasiado especializada para esperar que una biblioteca la admita (especialmente la versión de bucle fusionado). Además, el compilador no puede fusionar de forma segura los bucles automáticamente sin instrucciones específicas de que los punteros no tengan alias, y aun así no se garantiza que lo haga.

Afortunadamente, los lenguajes informáticos de alto nivel desde la década de 1950 han elevado el nivel de abstracción de la programación para todos nosotros. Naturalmente, a muchos de nosotros nos gustaría simplemente implementar la lógica empresarial requerida en nuestros códigos y dejar que el compilador y el Hardware hagan el resto. Pero, lamentablemente, no siempre se puede simplemente colocar el código en una computadora y esperar que se ejecute rápidamente. Cada vez más, a medida que el Hardware se vuelve más complejo, prestar atención a la arquitectura subyacente es fundamental para obtener un alto rendimiento.

**Preocúpate más por la Suma y la Resta que por la Multiplicación y la División** Los errores relativos en la multiplicación y división son siempre pequeños. La suma y la resta, por otro lado, pueden resultar en una pérdida completa de precisión. Realmente el problema es la resta; la suma sólo puede ser un problema cuando los dos números que se agregan tienen signos opuestos, por lo que puedes pensar en eso como una resta. Aún así, el código podría escribirse con un "+" que sea realmente una resta.

La resta es un problema cuando los dos números que se restan son casi iguales. Cuanto más casi iguales sean los números, mayor será el potencial de pérdida de precisión. Específicamente, si dos números están de acuerdo con n bits, se pueden perder n bits de precisión en la resta. Esto puede ser más fácil de ver en el extremo: si dos números no son iguales en teoría pero son iguales en su representación de máquina, su diferencia se calculará como cero, 100% de pérdida de precisión.

Aquí hay un ejemplo donde tal pérdida de precisión surge a menudo. La derivada de una función f en un punto $x$ se define como el límite de

$$(f(x+h) - f(x))/h$$

cuando $h$ llega a cero. Entonces, un enfoque natural para calcular la derivada de una función sería evaluar

$$(f(x+h) - f(x))/h$$

para alguna h pequeña. En teoría, cuanto menor es h, mejor se aproxima esta fracción a la derivada. En la práctica, la precisión mejora por un tiempo, pero más allá de cierto punto, valores más pequeños de h resultan en peores aproximaciones a la derivada. A medida que h disminuye, el error de aproximación se reduce pero el error numérico aumenta. Esto se debe a que la resta

$$f(x+h) - f(x)$$

se vuelve problemática. Si toma h lo suficientemente pequeño (después de todo, en teoría, más pequeño es mejor), entonces $f(x+h)$ será igual a $f(x)$ a la precisión de la máquina. Esto significa que todas las derivadas se calcularán como cero, sin importar la función, si solo toma h lo suficientemente pequeño. Aquí hay un ejemplo que calcula la derivada de $sin(x)$ en $x = 1$.

```
cout << std::setprecision(15);
for (int i = 1; i < 20; ++i)
{
double h = pow(10.0, -i);
cout << (sin(1.0+h) - sin(1.0))/h << "\n";
}
cout << "El verdadero resultado es: " << cos(1.0) << "\n";
```

Aquí está la salida del código anterior. Para que la salida sea más fácil de entender, los dígitos después del primer dígito incorrecto se han reemplazado por puntos.

```
0.4.........
0.53........
0.53........
0.5402......
0.5402......
0.540301....
0.5403022...
0.540302302.
0.54030235..
0.5403022...
0.540301....
0.54034.....
0.53........
```

0.544.....
0.55.....
0
0
0
0
El verdadero resultado es: 0.54030230586814

La precisión[92] mejora a medida que h se hace más pequeña hasta que $h = 10^{-8}$. Pasado ese punto, la precisión decae debido a la pérdida de precisión en la resta. Cuando $h = 10^{-16}$ o menor, la salida es exactamente cero porque $sin(1.0 + h)$ es igual a $sin(1.0)$ a la precisión de la máquina. (De hecho, 1 + h equivale a 1 a la precisión de la máquina. Más sobre eso a continuación).

¿Qué haces cuando tu problema requiere resta y va a causar una pérdida de precisión? A veces la pérdida de precisión no es un problema; los dobles comienzan con mucha precisión de sobra. Cuando la precisión es importante, a menudo es posible usar algún truco para cambiar el problema de modo que no requiera resta o no requiera la misma resta con la que comenzaste.

**Los Números de Punto Flotante Tienen Rangos Finitos** Todos saben que los números de punto flotante tienen rangos finitos, pero esta limitación puede aparecer de manera inesperada. Por ejemplo, puede encontrar sorprendente la salida de las siguientes líneas de código

```
float f = 16777216;
cout << f << " " << f+1 << "\n";
```

Este código imprime el valor 16777216 dos veces. ¿Que pasó? De acuerdo con la especificación IEEE para aritmética de punto flotante, un tipo flotante tiene 32 bits de ancho. Veinticuatro de estos bits están dedicados al significado (lo que solía llamarse la mantisa) y el resto al exponente. El número 16777216 es $2^{24}$ y, por lo tanto, a la variable flotante f no le queda precisión para representar $f + 1$. Ocurriría un fenómeno similar para $2^{53}$ si f fuera

[92] Los resultados anteriores se calcularon con Visual C ++ 2008. Cuando se compiló con gcc 4.2.3 en Linux, los resultados fueron los mismos, excepto los últimos cuatro números. Donde VC ++ produjo ceros, gcc produjo números negativos: -0.017 ..., - 0.17 ..., -1.7 ... y 17 ....

del tipo double porque un doble de 64 bits dedica 53 bits al significado. El siguiente código imprime 0 en lugar de 1.

```
x = 9007199254740992; // 2^53
cout << ((x+1) - x) << "\n";
```

También podemos quedarnos sin precisión al agregar números pequeños a números de tamaño moderado. Por ejemplo, el siguiente código imprime "¡Lo siento!" porque DBL_EPSILON (definido en float.h) es el número positivo más pequeño $\epsilon$ tal que $1 + \epsilon! = 1$ cuando se usan tipos dobles.

```
x = 1.0;
y = x + 0.5*DBL_EPSILON;
if (x == y)
cout << "¡Lo siento!\n";
```

De manera similar, la constante FLT_EPSILON es el número positivo más pequeño $\epsilon$ tal que $1 + \epsilon$ no es 1 cuando se usan tipos flotantes.

**¿Por qué los Flotantes IEEE Tienen Dos Ceros: $+0$ y $-0$?** Aquí hay un detalle extraño de la aritmética de punto flotante IEEE: las computadoras tienen dos versiones de 0: cero positivo y cero negativo. La mayoría de las veces, la distinción entre $+0$ y $-0$ no importa, pero de vez en cuando las versiones firmadas del cero son útiles.

Si una cantidad positiva llega a cero, se convierte en $+0$. Y si una cantidad negativa llega a cero, se convierte en $-0$. Podría pensar en $+0$ (respectivamente, $-0$) como el patrón de bits para un número positivo (negativo) demasiado pequeño para representarlo.

El estándar de punto flotante IEEE dice que $1/+0$ debería ser $+infinito$ y $1/-0$ debería ser $-infinito$. Esto tiene sentido si interpreta $+/-0$ como el fantasma de un número que se desbordó dejando solo su signo. El recíproco de un número positivo (negativo) demasiado pequeño para representarlo es un número positivo (negativo) demasiado grande para representarlo.

Para demostrar esto, ejecute el siguiente código C:

```
int main()
{
    double x = 1e-200;
```

```
    double y = 1e-200 * x;
    printf("Reciproco de +0: %g\n", 1/y);
    y = -1e-200*x;
    printf("Reciproco de -0: %g\n", 1/y);
}
```

En Linux con gcc, la salida es:

```
Reciproco de +0: inf
Reciproco de -0: -inf
```

Sin embargo, hay algo acerca de los ceros firmados y las excepciones que no tiene sentido. El informe acertadamente denominado "Lo que todo informático debería saber sobre la aritmética de punto flotante" tiene lo siguiente que decir sobre los ceros con signo.

> En aritmética IEEE, es natural definir $log0 = -\infty$ y $logx$ como un $NaN$ cuando $x < 0$. Suponga que $x$ representa un pequeño número negativo que se ha desbordado a cero. Gracias al cero con signo, $x$ será negativo, por lo que $log$ puede devolver un $NaN$. Sin embargo, si no hubiera un cero con signo, la función logarítmica no podría distinguir un número negativo subdesbordado de 0 y, por lo tanto, tendría que devolver $-\infty$.

Esto implica que $log(-0)$ debe ser $NaN$ y $log(+0)$ debe ser $-\infty$. Eso tiene sentido, pero eso no es lo que sucede en la práctica. La función de $log$ devuelve $-\infty$ para $+0$ y $-0$.

Ejecuté el siguiente código en C:

```
int main()
{
    double x = 1e-200;
    double y = 1e-200 * x;
    printf("Log de +0: %g\n", log(y));
    y = -1e-200*x;
    printf("Log de -0: %g\n", log(y));
}
```

En Linux, el código imprime:

```
Log de +0: -inf
Log de -0: -inf
```

**Use Logaritmos para Evitar Desbordamiento y Subdesbordamiento** Las limitaciones de los números de punto flotante descritos en la sección anterior provienen de tener un número limitado de bits en el significado. El desbordamiento y el subdesbordamiento resultan de tener también un número finito de bits en el exponente. Algunos números son demasiado grandes o demasiado pequeños para almacenarlos en un número de punto flotante.

Muchos problemas parecen requerir calcular un número de tamaño moderado como la razón de dos números enormes. El resultado final puede ser representable como un número de punto flotante aunque los resultados intermedios no lo sean. En este caso, los logaritmos proporcionan una salida. Si desea calcular $M/N$ para grandes números $M$ y $N$, calcule $log(M) - log(N)$ y aplique $exp()$ al resultado. Por ejemplo, las probabilidades a menudo implican proporciones de factoriales, y los factoriales se vuelven astronómicamente grandes rápidamente. Para $N > 170$, $N!$ es mayor que DBL_MAX, el número más grande que puede representarse por un doble (sin precisión extendida). Pero es posible evaluar expresiones como 200!/(190!10!) Sin desbordamiento de la siguiente manera:

x = exp( logFactorial(200) - logFactorial(190) - logFactorial(10) );

Una función logFactorial simple pero ineficiente podría escribirse de la siguiente manera:

```
double logFactorial(int n)
{
    double sum = 0.0;
    for (int i = 2; i <= n; ++i) sum += log((double)i);
    return sum;
}
```

Un mejor enfoque sería utilizar una función de registro gamma si hay una disponible. Consulte Cómo calcular las probabilidades binomiales para obtener más información.

**Las Operaciones Numéricas no Siempre Devuelven Números** Debido a que los números de punto flotante tienen sus limitaciones, a veces las ope-raciones de punto flotante devuelven "infinito" como una forma de decir "el resultado es más grande de lo que puedo manejar". Por ejemplo, el siguiente código imprime 1. # *INF* en Windows e inf en Linux.

```
x = DBL_MAX;
cout << 2*x << "\n";
```

A veces, la barrera para devolver un resultado significativo tiene que ver con la lógica en lugar de la precisión finita. Los tipos de datos de punto flotante representan números reales (a diferencia de los números complejos) y no hay un número real cuyo cuadrado sea $-1$. Eso significa que no hay un número significativo que devolver si el código solicita $sqrt(-2)$, incluso con una precisión infinita. En este caso, las operaciones de punto flotante devuelven *NaN*. Estos son valores de punto flotante que representan códigos de error en lugar de números. Los valores *NaN* se muestran como 1. # *IND* en Windows y *NAN* en Linux.

Una vez que una cadena de operaciones encuentra un *NaN*, todo es un *NaN* de ahí en adelante. Por ejemplo, suponga que tiene un código que equivale a algo como lo siguiente:

```
if (x - x == 0)
// hacer algo
```

¿Qué podría impedir que se ejecute el código que sigue a la instrucción if? Si x es un *NaN*, entonces también lo es $x - x$ y los *NaN* no equivalen a nada. De hecho, los *NaN* ni siquiera se igualan. Eso significa que la expresión $x == x$ se puede usar para probar si $x$ es un número (posiblemente infinito). Para obtener más información sobre infinitos y *NaN*, consulte las excepciones de punto flotante IEEE en C ++.

**Trabajando con Factoriales** Los cálculos de probabilidad a menudo implican tomar la razón de números muy grandes para producir un número de tamaño moderado. El resultado final puede caber dentro de un doble con espacio de sobra, pero los resultados intermedios se desbordarían. Por ejemplo, suponga que necesita calcular el número de formas de seleccionar 10 objetos de un conjunto de 200. ¡Esto es 200!/(190!10!), Aproximadamente $2.2e16$. Pero 200! y 190! desbordaría el rango de un doble.

Hay dos formas de solucionar este problema. Ambos usan la siguiente regla: Use trucos algebraicos para evitar el desbordamiento.

El primer truco es reconocer que

$$200! = 200 * 199 * 198 * ... * 191 * 190!$$

y así

$$200!/(190!10!) = 200 * 199 * 198 * ... * 191/10!$$

esto ciertamente funciona, pero está limitado a factoriales.

Una técnica más general es usar logaritmos para evitar el desbordamiento: tome el logaritmo de la expresión que desea evaluar y luego exponga el resultado. En este ejemplo

$$log(200!/(190!10!)) = Log(200!) - log(190!) - log(10!)$$

si tiene un código que calcula el logaritmo de los factoriales directamente sin calcular primero los factoriales, puede usarlo para encontrar el logaritmo del resultado que desea y luego aplicar la función $exp$.

**Calcular Inverso del Factorial** Dado un número positivo $x$, ¿cómo se puede encontrar un número $n$ tal que $n! = x$, o tal que $n! \approx x$?. El Software matemático tiende a trabajar con la función gamma en lugar de factoriales porque $\Gamma(x)$ extiende $x!$ en números reales, con la relación $\Gamma(x + 1) = x!$. El lado izquierdo a menudo se toma como una definición del lado derecho cuando $x$ no es un número entero positivo.

No sólo preferimos trabajar con la función gamma, sino que es más fácil trabajar con el logaritmo de la función gamma para evitar el desbordamiento.

El siguiente código Python encuentra el inverso del logaritmo de la función gamma utilizando el método de bisección. Este método requiere un límite superior e inferior. Si solo pasamos valores positivos, 0 sirve como límite inferior. Podemos encontrar un límite superior probando potencias de 2 hasta obtener algo lo suficientemente grande.

```
from scipy.special import gammaln
from scipy.optimize import bisect
import math as mt
def inverse_log_gamma(logarg):
    assert(logarg > 0)
    a = 1
    b = 2
    while b < logarg:
        b = b*2
    return bisect(lambda x: gammaln(x) - logarg, a, b)
def inverse_factorial(logarg):
```

```
g = inverse_log_gamma(logarg)
return round(g)-1
```

Tenga en cuenta que la función inverse_factorial toma el logaritmo del valor cuyo inverso del factorial desea calcular. Por ejemplo:

```
print(inverse_factorial(mt.log(mt.factorial(42))))
```
regresa 42.

Trabajar en la escala logarítmica nos permite trabajar con números mucho mayores. El factorial de 171 es mayor que el mayor número de doble precisión de IEEE, por lo que inverse_factorial no podría devolver ningún valor mayor que 171 si pasáramos x en lugar de log x. Pero al tomar log x como argumento, podemos calcular factoriales inversos de números tan grandes que el factorial se desborda.

Por ejemplo, supongamos que queremos encontrar el valor de $m$ tal que $m!$ es el factorial más cercano a $\sqrt{(2024!)}$. Podemos usar el código.

```
print(inverse_factorial(gammaln(2025)/2))
```

para encontrar $m = 1112$ aunque $1112!$ es del orden de $10^{2906}$, mucho mayor que el mayor número de punto flotante representable, que es del orden de $10^{308}$.

Cuando $n! = x$ no tiene una solución exacta, existe alguna opción sobre qué valor de $n$ devolver como inverso del factorial de $x$. El código anterior minimiza la distancia desde $n!$ a $x$ en una escala logarítmica. Es posible que desee modificar el código para minimizar la distancia en una escala lineal o encontrar la $n$ más grande con $n! < x$, o la $n$ más pequeña con $n! > x$ dependiendo de su aplicación.

**Trabajando con Exponenciales y Logaritmos** Supongamos que desea evaluar la función[93]

$$u(z) = \frac{e^z - 1 - z}{z^2}$$

para valores pequeños de $z$, digamos $z = 10^{-8}$.

El código Python

[93] Este ejemplo proviene de Lloyd N. Trefethen and J. A. C. Weideman. The Exponentially Convergent Trapezoid Rule. SIAM Review. Vol. 56, No. 3. pp. 385-458.

```
import numpy as np
def f(z): return (np.exp(z) - 1 - z)/z**2
print(f(1e-8))
```

imprime:

```
-0.607747099184471.
```

Ahora suponga que sospecha dificultades numéricas y calcula su resultado con 50 decimales usando *bc -l*[94] usando:

```
scale=50
z=10^-8
(e(z)-1-z)/z^2
```

imprime:

```
.50000000166666667083333334166666660000000000000000
```

Esto sugiere que el cálculo original era completamente erróneo. ¿Qué está sucediendo?

Para $z$ pequeña,

$$e^z \approx 1 + z$$

y así perdemos precisión al evaluar directamente el numerador en la definición de $u$ (en nuestro ejemplo, perdimos toda precisión).

El teorema del valor medio del análisis complejo dice que el valor de una función analítica en un punto es igual al promedio continuo de los valores sobre un círculo centrado en ese punto. Si aproximamos este promedio tomando el promedio de 16 valores en un círculo de radio 1 alrededor de nuestro punto, obtenemos una precisión total. El código Python

```
def g(z):
    N = 16
    ws = z + np.exp(2j*np.pi*np.arange(N)/N)
    return sum(f(ws))/N
print(g(1e-8))
```

[94] El comando bc (Basic Calculator) de la línea de comandos de Linux/Unix, es una calculadora con grandes posibilidades de uso.

imprime:

0.5000000016666668 + 8.673617379884035e-19j

que se aparta del resultado calculado con *bc* en el decimosexto decimal. En un nivel alto, evitamos dificultades numéricas al promediar puntos alejados de la región difícil.

**Logit inverso** A continuación, veamos el cálculo $f(x) = e^x/(1 + e^x)$. (Los estadísticos llaman a esto la función "logit inverso" porque es el inverso de la función que ellos llaman la función "logit".) El enfoque más directo sería calcular $exp(x)/(1 + exp(x))$. Veamos dónde se puede romper eso.

```
double x = 1000;
double t = exp(x);
double y = t/(1.0 + t);
```

Imprimir y da -1. # *IND*. Esto se debe a que el cálculo de t se desbordó, produciendo un número mayor que el que cabía en un doble. Pero podemos calcular el valor de y fácilmente. Si t es tan grande que no podemos almacenarlo, entonces $1 + t$ es esencialmente lo mismo que $t$ y la relación es muy cercana a 1. Esto sugiere que descubramos qué valores de $x$ son tan grandes que $f(x)$ será igual 1 a la precisión de la máquina, luego solo devuelva 1 para esos valores de $x$ para evitar la posibilidad de desbordamiento. Esta es nuestra siguiente regla: No calcules un resultado que puedas predecir con precisión.

El archivo de encabezado float.h tiene un $DBL_EPSILON$ constante, que es la precisión doble más pequeña que podemos agregar a 1 sin recuperar 1. Un poco de álgebra muestra que si $x$ es más grande que $-log(DBL_EPSILON)$, entonces $f(x)$ será igual a 1 a la precisión de la máquina. Así que aquí hay un fragmento de código para calcular $f(x)$ para valores grandes de $x$:

```
const double x_max = -log(DBL_EPSILON);
if (x > x_max) return 1.0;
```

El código provisto en esta sección calcula siete funciones que aparecen en las estadísticas. Cada uno evita problemas de desbordamiento, subdesbordamiento o pérdida de precisión que podrían ocurrir para grandes argumentos negativos, grandes argumentos positivos o argumentos cercanos a cero:

- LogOnePlusX calcúlese como $log(1 + x)$ como en ejemplo antes visto
- ExpMinusOne calcúlese como $e^x - 1$
- Logit calcúlese como $log(x/(1 - x))$
- LogitInverse calcúlese como $e^x/(1 + e^x)$ como se discutió en el ejemplo último
- LogLogitInverse calcúlese como $log(e^x/(1 + e^x))$
- LogitInverseDifference calcúlese como $LogitInverse(x) - LogitInverse(y)$
- LogOnePlusExpX calcúlese como $log(1 + exp(x))$
- ComplementaryLogLog calcúlese como $log(-log(1 - x))$
- ComplementaryLogLogInverse calcúlese como $1.0 - exp(-exp(x))$

Las soluciones presentadas aquí parecen innecesarias o incluso incorrectas al principio. Si este tipo de código no está bien comentado, alguien lo verá y lo "simplificará" incorrectamente. Estarán orgullosos de todo el desorden innecesario que eliminaron. Y si no prueban valores extremos, su nuevo código parecerá funcionar correctamente. Las respuestas incorrectas y los $NaN$ solo aparecerán más tarde.

**Log (1 + x)** Ahora veamos el ejemplo del cálculo de $log(x+1)$ . Considere el siguiente código:

```
double x = 1e-16;
double y = log(1 + x)/x;
```

En este código $y = 0$, aunque el valor correcto sea igual a 1 para la precisión de la máquina.

**¿Qué salió mal?** los números de doble precisión tienen una precisión de aproximadamente 15 decimales, por lo que $1 + x$ equivale a 1 para la precisión de la máquina. El registro de 1 es cero, por lo que y se establece en cero. Pero para valores pequeños de $x$, $log(1 + x)$ es aproximadamente $x$, por lo que $log(1 + x)/x$ es aproximadamente 1. Eso significa que el código anterior para calcular $log(1 + x)/x$ devuelve un resultado con 100% de error relativo. Si $x$ no es tan pequeño que $1 + x$ es igual a 1 en la máquina, aún podemos tener problemas. Si $x$ es moderadamente pequeño, los bits en $x$ no se pierden totalmente al calcular $1 + x$, pero algunos sí. Cuanto más se acerca $x$ a 0, más bits se pierden. Podemos usar la siguiente regla: Utilice aproximaciones analíticas para evitar la pérdida de precisión.

La forma favorita de aproximación de los analistas numéricos es la serie de potencia. ¡La serie de potencia para

$$log(1 + x) = x + x^2/2! + x^3/3! + ...$$

Para valores pequeños de $x$, simplemente devolver $x$ para $log(1 + x)$ es una mejora. Esto funcionaría bien para los valores más pequeños de $x$, pero para algunos valores no tan pequeños, esto no será lo suficientemente preciso, pero tampoco lo hará directamente el cálculo del $log(1 + x)$.

**La Precisión Arbitraria no es una Panacea** Tener acceso a aritmética de precisión arbitraria no necesariamente hace que desaparezcan las dificultades de cálculo numérico. Aún necesitas saber lo que estás haciendo. Antes de ampliar la precisión, hay que saber hasta dónde ampliarla. Si no lo amplía lo suficiente, sus respuestas no serán lo suficientemente precisas. Si lo extiendes demasiado, desperdicias recursos. Tal vez esté bien desperdiciar recursos, por lo que extiende la precisión más de lo necesario. ¿Pero hasta qué punto es más de lo necesario?

Como ejemplo, considere el problema de encontrar valores de $n$ tales que $tan(n) > n$, uno de los valores es:

$$k = 1428599129020608582548671.$$

Verifiquemos que $tan(k) > k$. Usaremos *bc* porque admite precisión arbitraria. Nuestra $k$ es un número de 25 dígitos, así que digamos a *bc* que queremos trabajar con 30 decimales para tener un poco de espacio[95]. *bc* no

[95] *bc* automáticamente le otorga un poco más de espacio del que solicita, pero le pediremos aún más

tiene una función tangente, pero sí la tiene *s()* para seno y *c()* para coseno, por lo que calcularemos la tangente como seno sobre coseno.

```
$ bc -l
escala = 30
k = 14285991290206085825486 71
s(k)/c(k) - k
```

Esperamos que esto devuelva un valor positivo, pero en su lugar devuelve

-1428599362980017942210629.31...

Entonces, ¿está mal la hipótesis? ¿O es *bc* ignorar nuestra solicitud? Resulta que ninguna de las dos cosas es cierta.

No se puede calcular directamente la tangente de un número grande. Utiliza la reducción de rango para reducirlo al problema de calcular la tangente de un ángulo pequeño donde funcionarán sus algoritmos. Dado que la tangente tiene un período $\pi$, reducimos $k$ mod $\pi$ calculando $k - [k/\pi]\,\pi$. Es decir, restamos tantos múltiplos de $\pi$ como podamos hasta obtener un número entre 0 y $\pi$. Volviendo a *bc*, calculamos

```
pi = 4*a(1)
k/pi
```

esto regresa

45473722616081240284265 6.50000050703322137044486 6152761

y entonces calculamos la tangente de

```
t = 0,50000050703322137044486 6152761*pi
= 1,57079791968674000258827 0178941
```

Como $t$ es ligeramente mayor que $\pi/2$, la tangente será negativa. No podemos tener $tan(t)$ mayor que $k$ porque $tan(t)$ ni siquiera es mayor que 0. Entonces, ¿dónde se estropearon las cosas?.

El cálculo de $\pi$ tuvo una precisión de 30 cifras significativas y el cálculo de $k/\pi$ tuvo una precisión de 30 cifras significativas, dado nuestro valor de $\pi$. Hasta ahora ha funcionado según lo prometido.

El valor calculado de $k/\pi$ tiene una precisión de 29 cifras significativas, 23 antes del decimal y 6 después. Entonces, cuando tomamos la parte fraccionaria, solo tenemos seis cifras significativas y eso no es suficiente. Ahí es donde las cosas van mal. Obtenemos un valor $[k/\pi]$ que es mayor que 0.5 en el séptimo decimal, mientras que el valor exacto es menor que 0.5 en el vigésimo quinto decimal. Necesitábamos $25-6 = 19$ cifras más significativas.

Ésta es la principal dificultad del cálculo en punto flotante: restar números casi iguales pierde precisión. Si dos números coinciden en $m$ decimales, puedes esperar perder $m$ cifras significativas en la resta. El error en la resta será pequeño en relación con los datos de entrada, pero no pequeño en relación con el resultado.

Observe que calculamos $k/\pi$ hasta 29 cifras significativas y, dado ese resultado, calculamos la parte fraccionaria exactamente . Calculamos con precisión $[k/\pi]\,\pi$, pero perdimos precisión cuando calculamos y restamos ese valor de $k$.

Nuestro error al calcular $k-[k/\pi]\,\pi$ fue pequeño en relación con $k$, pero no en relación con el resultado final. Nuestra $k$ es del orden de $10^{25}$ y el error en nuestra resta es del orden de $10^{-7}$, pero el resultado es del orden de 1. No hay ningún error en *bc*. Llevó a cabo todos los cálculos con la precisión anunciada, pero no tuvo suficiente precisión de trabajo para producir el resultado que necesitábamos.

## 5.3 Aritmética de Baja Precisión

La popularidad de la aritmética de baja precisión para el cómputo de alto rendimiento se ha disparado desde el lanzamiento en 2017 de la GPU Nvidia Volta. Los núcleos tensores de media precisión de Volta ofrecieron una enorme ganancia de rendimiento 16 veces mayor que la doble precisión para operaciones clave. Y el rendimiento del Hardware está mejorando aún más: el FP16 con núcleo tensor Nvidia H100 es 58 veces más rápido que el FP64 estándar.

Esta sorprendente aceleración ciertamente llama la atención. Sin embargo, en el cálculo científico, la aritmética de baja precisión suele considerarse insegura para los códigos de modelado y simulación. De hecho, a veces se puede aprovechar una precisión más baja, comúnmente en una configuración de "precisión mixta" en la que sólo partes del cálculo se realizan con baja precisión. Sin embargo, en general, cualquier precisión menor que el doble se considera inadecuada para modelar fenómenos físicos complejos

con fidelidad.

En respuesta, los desarrolladores han creado herramientas para medir la seguridad de la aritmética de precisión reducida en códigos de aplicación. Algunas herramientas pueden incluso identificar qué variables o matrices se pueden reducir de forma segura a una precisión menor sin perder precisión en el resultado final. Sin embargo, el uso de estas herramientas a ciegas, sin el respaldo de algún tipo de proceso de razonamiento, puede resultar peligroso.

Un ejemplo ilustrará esto:

> El método del gradiente conjugado para la resolución y optimización de sistemas lineales y el método de Lanczos, estrechamente relacionado, para la resolución de problemas de valores propios mostraron una gran promesa tras su invención a principios de los años cincuenta. Sin embargo, se consideraban inseguros debido a errores de redondeo catastróficos en la aritmética de punto flotante, que son aún más pronunciados a medida que se reduce la precisión del punto flotante.
>
> No obstante, Chris Paige demostró en su trabajo pionero en la década de 1970 que el error de redondeo, aunque sustancial, no excluye la utilidad de los métodos cuando se utilizan correctamente. El método del gradiente conjugado se ha convertido en un pilar del cómputo científico.

Teniendo en cuenta que ninguna herramienta podría llegar a este hallazgo sin un cuidadoso análisis matemático de los métodos. Una herramienta detectaría inexactitudes en el cálculo pero no podría certificar que estos errores no puedan perjudicar el resultado final.

Algunos podrían proponer en cambio un enfoque puramente basado en datos: simplemente pruebe con baja precisión en algunos casos de prueba; si funciona, utilice baja precisión en producción. Sin embargo, este enfoque está lleno de peligros: es posible que los casos de prueba no capturen todas las situaciones que podrían encontrarse en producción.

Por ejemplo, uno podría probar un código de aerodinámica sólo en regímenes de flujo suaves, pero las series de producción pueden encontrar flujos complejos con gradientes pronunciados, que la aritmética de baja precisión no puede modelar correctamente. Los artículos académicos que prueban métodos y herramientas de baja precisión deben evaluarse rigurosamente en escenarios desafiantes del mundo real como este.

Lamentablemente, los equipos de ciencia computacional frecuentemente no tienen tiempo para evaluar sus códigos para un uso potencial de aritmética de menor precisión. Las herramientas ciertamente podrían ayudar. Además, las bibliotecas que encapsulan métodos de precisión mixta pueden ofrecer beneficios a muchos usuarios. Una gran historia de éxito son los solucionadores lineales densos de precisión mixta, basados en el sólido trabajo teórico de Nick Highnam y sus colegas , que han llegado a bibliotecas como: Lu, Hao; Matheson, Michael; Wang, Feiyi; Joubert, Wayne; Ellis, Austin; Oles, Vladyslav. "OpenMxP-Opensource Mixed Precision Computing,".

Entonces la respuesta final es "depende". Cada nuevo caso debe examinarse cuidadosamente y tomar una decisión basada en alguna combinación de análisis y pruebas.

Sí bien, NVIDIA lidera el mercado de las GPU para inteligencia artificial (IA) con una cuota de mercado aproximada del 80%, pero no es en absoluto la única empresa que tiene en su porfolio chips de vanguardia para IA. La compañía californiana Cerebras posee, de hecho, los procesadores para este escenario de uso más complejos que existen. Su chip WSE-2, por ejemplo, aglutina nada menos que 2.6 billones de transistores contabilizados en la escala numérica larga y $850,000$ núcleos optimizados para IA.

Cerebras entrega a sus clientes estos procesadores integrados en una plataforma para IA conocida como CS-2, y precisamente uno de ellos es la compañía de Emiratos Árabes G42. Esta última está construyendo seis superordenadores para IA capaces de superar la barrera de la exaescala que aglutinan una gran cantidad de sistemas CS-2. Y según la CIA algunas de estas máquinas irán a parar a las grandes tecnológicas chinas. No obstante, esto no es todo. Y es que Cerebras dió a conocer en 2024 un procesador para IA aún más potente que su WSE-2.

**El procesador WSE-3** Cerebras ya tiene listo su procesador WSE-3 (Wafer Scale Engine 3), un producto que, como podemos intuir, está llamado a suceder al también ambicioso WSE-2.

Ambos procesadores se fabrican a partir de una oblea completa de silicio, lo que permite a Cerebras integrar muchos más bloques funcionales y núcleos en la lógica que una GPU convencional como las que fabrican NVIDIA, AMD o Huawei.

Y es que aglutina 4 billones de transistores, tiene una superficie de $46,225$ $mm^2$, integra nada menos que $900,000$ núcleos optimizados para IA y tiene

una potencia de cálculo, según Cerebras, de 125 petaflops.

Según Cerebras su procesador WSE-3 es el doble de potente que el WSE-2. De hecho, de acuerdo con las especificaciones que ha publicado rinde como 62 GPU H100 de NVIDIA trabajando al unísono, y no debemos pasar por alto que este procesador de la compañía liderada por Jensen Huang es el más potente que tiene hasta que se produzca el lanzamiento de la GPU H200.

Sea como sea Cerebras entrega sus procesadores WSE-3 integrados en un superordenador conocido como CS-3 que es capaz de entrenar grandes modelos de IA con hasta 24 billones de parámetros. El mapa de memoria externa de este superordenador oscila entre 1.5 TB y 1.2 PB, un espacio de almacenamiento descomunal que permite almacenar modelos de lenguaje masivos en un único espacio lógico.

Según informan, el chip WSE-3 optimizado para la IA es capaz de entrenar hasta $24,000$ millones de parámetros, lo que también equivaldría a un rendimiento máximo de IA de 125 petaflops.

# 6 Apéndice A: Sistemas Operativos

Actualmente tenemos 3 grandes sistemas operativos en el mercado[96]:

- Windows
- Unix
- GNU[97]/Linux

De los cuales, sus dignos representantes son: Windows, macOS, iOS, Android, Chrome OS y GNU/Linux con todas sus diferentes distribuciones[98]. 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

[96] Cuotas de mercado de diferentes sistemas operativos:

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

[97] 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.

[98] 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** Febrero y Agosto son 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), Statcounter (https://www.statcounter.com) y The Linux Foundation (https://www.linux-fundation.org) en el último año tenemos:

- En el segmento de los sistemas operativos de escritorio basados en Linux ha subido su cuota de mercado llegando al 4%, esto no parecerá mucho, pero si nos fijamos bien, vemos que Mac tiene un 15% -basado en Unix-, Chrome OS -usa el Kernel de Linux- tiene 2% 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[99].

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[100] 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)

[99] 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%).

[100] 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[101], 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

[101] 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[102] 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

[102] 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[103] 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

[103] 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[104] 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.

## 6.1 Windows

Microsoft Windows (véase [1]), conocido generalmente como Windows o MS Windows es el nombre de una familia de Software propietario (véase apéndice 10.1) 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.

[104] 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[105] 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

[105] 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[106]). Reiniciando el sistema, ya podemos usar distribuciones de Linux desde Microsoft Store.

[106] 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

## 6.2 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 10.4).

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.

## 6.3 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 [3]) -del inglés Macintosh Operating System, en español Sistema Operativo Macintosh- es el nombre del sistema operativo propietario (véase apéndice 10.1) 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*.

## 6.4 GNU/Linux

GNU[107]/Linux (véase [2]) también conocido como Linux, es un sistema operativo libre (véase apéndice 10.2) 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-

[107] 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[108]-, 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-

[108] 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[109], 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.

[109] 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[110]. La única limitación es que seamos capaces o ten-gamos 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.

[110] 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[111] para instalar, una de las más ampliamente usadas es Debian GNU/Linux[112] 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[113]:

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/

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

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

[113] 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[114] (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

## 6.5 Android

Android (véase [4]) es un sistema operativo basado en el núcleo Linux (véase apéndice 10.2). 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,

[114] 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[115]:

**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.

[115] 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.

## 6.6 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[116] 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

[116] 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.

## 6.7 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.

# 7 Apéndice B: El Sistema Operativo GNU/Linux

GNU/Linux se ve y se siente muy parecido a cualquier otro sistema UNIX, y de hecho la compatibilidad con UNIX ha sido una importante meta del diseño del proyecto Linux. No obstante, Linux es mucho más joven que la mayor parte de los sistemas UNIX. Su desarrollo se inició en 1991, cuando un estudiante finlandés, Linus Torvalds, escribió y bautizó un pequeño pero autosuficiente núcleo para el procesador 80386, el primer procesador de 32 bits verdadero en la gama de CPU compatibles con el PC de Intel.

En los albores de su desarrollo, el código fuente de Linux se ofrecía gratuitamente en internet. En consecuencia, su historia ha sido una colaboración de muchos usuarios de todo el mundo que se han comunicado casi exclusivamente a través de internet. Desde un núcleo inicial que implementaba parcialmente un subconjunto pequeño de los servicios de UNIX, Linux ha crecido para incluir cada vez más funcionalidades UNIX.

En sus inicios, el desarrollo de Linux giraba en gran medida alrededor del núcleo del sistema operativo central: el ejecutivo privilegiado que administra todos los recursos del sistema e interactúa directamente con el Hardware. Desde luego, se requiere mucho más que este núcleo para producir un sistema operativo completo.

Resulta útil hacer la distinción entre el núcleo (Kernel) de Linux y un sistema Linux: el núcleo en Linux es una identidad de Software totalmente original desarrollada desde cero por la comunidad Linux (suele encontrarse en el directorio */boot* en el sistema de archivos); el sistema Linux, tal como lo conocemos hoy, incluye una multitud de componentes, algunos escritos desde cero, otros tomados en préstamo de otros proyectos o creados en colaboración con otros equipos como el proyecto GNU de la Free Software Foundation.

El sistema Linux básico es un entorno estándar para aplicaciones y programación de los usuarios, pero no obliga a adoptar mecanismos estándar para controlar las funcionalidades disponibles como un todo.

GNU/Linux en sus inicios trabajaba en modo consola o terminal -línea de comandos o *Shell*- usando la interfaz de línea de comando (CLI por Command Line Interface) o interfaz de usuario de terminal (TUI por Terminal User Interface). A medida que Linux ha madurado, se ha hecho necesaria otra capa de funcionalidad encima del sistema Linux: El servidor de Pantalla

Un servidor de pantalla es un programa cuya tarea principal es coordinar la entrada y salida de sus clientes hacia y desde el resto del sistema operativo, el Hardware y entre sí. El servidor de pantalla se comunica con sus clientes a

través del protocolo del servidor de pantalla. Un servidor de visualización es un componente clave en cualquier interfaz gráfica de usuario, específicamente el sistema de ventanas. Es el componente básico de la interfaz gráfica de usuario (GUI) que se encuentra entre la interfaz gráfica y el Kernel.

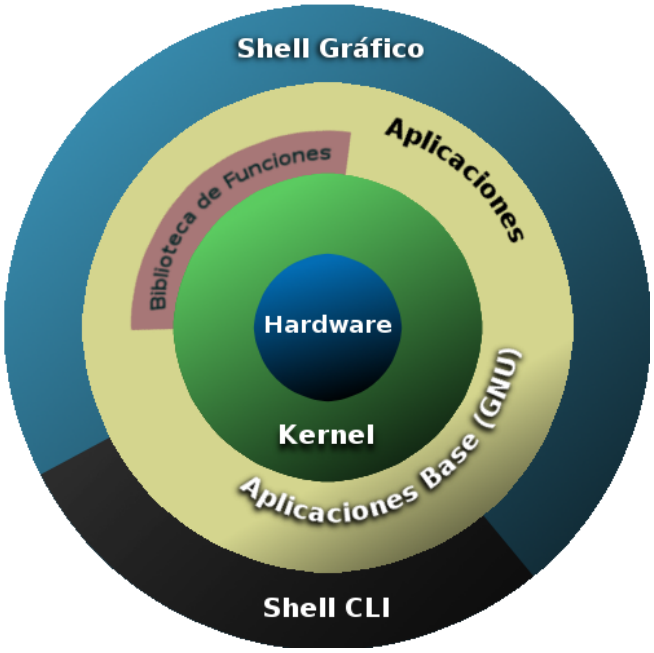



Figura 13: GNU/Linux

Entonces, gracias a un servidor de pantalla, podemos usar la computadora con GUI. Sin él, solo estaría restringido a una interfaz de línea de comandos, sería placentero, pero la GUI también tiene magia. No confundir el servidor de visualización con el entorno de escritorio ( GNOME, KDE, LXQt, LXDE, Xfce, Unity, MATE, Cinnamon, Pantheon, Deepin, Budgie, PIXEL, Enlightenment, Trinity, Moksha, Ukui, etc.), los entornos de escritorio usan uno, pero en una capa por debajo.

El servidor de pantalla se comunica con sus clientes a través del protocolo del servidor de pantalla. Hay tres protocolos principales de servidor de pantalla disponibles: X11, Mir (desarrollado por Canonical) y Wayland.

X Window System, a menudo denominado simplemente X, es el más antiguo de todos (1984), terminó siendo el sistema de ventanas predeterminado para la mayoría de los sistemas operativos similares a UNIX, incluido

GNU/Linux. El servidor X.Org es la implementación gratuita y de código abierto del servidor de visualización X Window System administrado por la Fundación X.Org. Es una aplicación que interactúa con aplicaciones cliente a través del protocolo X11 para dibujar cosas en una pantalla y enviar eventos de entrada como movimientos del mouse, Clics y pulsaciones de teclas. Normalmente, uno iniciaría un servidor X que esperará a que las aplicaciones de los clientes se conecten a él. Xorg se basa en un modelo cliente/servidor y, por lo tanto, permite que los clientes se ejecuten de forma local o remota en una máquina diferente. La mayoría de las funciones que proporciona el protocolo X Server ya no se utilizaban. Casi todo el trabajo que hizo X11 se volvió a delegar a las aplicaciones individuales y al administrador de ventanas. Y, sin embargo, todas esas características antiguas siguen ahí, pesando sobre todas estas aplicaciones, perjudicando el rendimiento y la seguridad. Es por ello y otras cosas que nace Wayland[117].

[117] Wayland fue iniciado por Kristian Hogsberg, un desarrollador de X.Org, como un proyecto personal en 2008. Es un protocolo de comunicación que especifica la comunicación entre un servidor de pantalla y sus clientes. Wayland se desarrolla como un proyecto gratuito y de código abierto impulsado por la comunidad con el objetivo de reemplazar el sistema de ventanas X11 o Xorg, por un sistema de ventanas moderno, seguro y más simple.

En Wayland, el compositor es el servidor de visualización. Compositor, es un administrador de ventanas que proporciona a las aplicaciones un Buffer fuera de la pantalla para cada ventana. El administrador de ventanas compone los búffers de la ventana en una imagen que representa la pantalla y escribe el resultado en la memoria de visualización. El protocolo Wayland permite al compositor enviar los eventos de entrada directamente a los clientes y permite que el cliente envíe el evento de daños directamente al compositor.

La principal ventaja de Wayland sobre X es que comienza desde cero. Una de las principales razones de la complejidad de X es que, a lo largo de los años, su función ha cambiado. Como resultado, hoy en día, X11 actúa en gran medida como un protocolo de comunicaciones "realmente terrible" entre el cliente y el administrador de ventanas. Wayland también es superior cuando se trata de seguridad. Con X11, es posible hacer Keylogging al permitir que cualquier programa exista en segundo plano y lea lo que está sucediendo con otras ventanas abiertas en el área de X11. Con Wayland, esto simplemente no sucederá, ya que cada programa funciona de forma independiente.

Sin embargo, el sistema X Window todavía tiene muchas ventajas sobre Wayland. Aunque Wayland elimina la mayoría de los defectos de diseño del Xorg, tiene sus propios problemas. A pesar de que el proyecto Wayland ha estado activo durante más de diez años, las cosas no son 100% estables, de ahí que aun sigamos usando Xorg. Aún hoy, la mayoría de los videojuegos y las aplicaciones de gráficos intensivos para Linux todavía se escriben para X11. Además, muchos controladores de gráficos de código cerrado, como los de las GPU NVIDIA, aún no ofrecen soporte completo para Wayland. Desde mi punto de vista aún tendremos Xorg para una década más, aunque ya comiencen a salir algunos

Una distribución de GNU/Linux incluye todos los componentes estándar del sistema Linux, más un conjunto de herramientas administrativas que simplifican la instalación inicial y desinstalación de paquetes del sistema.

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

**Hardware en GNU/Linux** El soporte del Hardware en Linux es un asunto complicado, es lo que más problemas da. Linux soporta la mayor parte del Hardware, pero a veces pueden existir problemas[118]:

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

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

[118] Mito: Linux/Unix se puede usar para revivir un equipo de cómputo viejo. La realidad es que si bien, hay múltiples distribuciones de Linux/Unix que corren en una gran cantidad de procesadores antiguos y actuales, los Drivers necesarios para reconocer periféricos como tarjetas gráficas, de red alámbrica e inalámbrica, entre muchos otros, no tienen soporte en Linux/Unix, lo cual hará imposible su uso en Linux/Unix. Esto es cierto en cualquier computadora no importa de cuál generación es el equipo de cómputo. La verdad de todo esto, es que los fabricantes están enfocados en producir Hardware y Drivers que corran en los sistemas operativos con mayor cuota de mercado y por el momento Linux/Unix en equipos personales no son de ellos.

La compatibilidad del Hardware depende en gran medida de la versión de Kernel de GNU/Linux instalado, es de esperarse que en versiones anteriores del Kernel cierto Hardware no se pueda detectar, pero lo contrario también pasa, hay Drivers que solo corren correctamente en versiones anteriores del Kernel y no en las últimas versiones, lo que ocasiona que muchos usuarios se desesperen al tratar de usar sus equipos con GNU/Linux. Y en caso de logar que funcione el Hardware, se fuerza a los usuarios a usar una determinada versión del Kernel (y todas las aplicaciones de la distribución) no actualizable, por la imposibilidad de hacer funcionar el Hardware del equipo en una más moderna con la consiguiente obsolescencia del Software instalado en el equipo.

- si es un dispositivo exclusivo para Windows, como los Winmodems (linmodems.org).

Si el Hardware es "de verdad" (no Winmodems), de marca conocida y actual, casi con toda seguridad estará soportado por Linux.

Los instaladores de Linux reconocen prácticamente todo el Hardware durante la instalación, por lo que la mejor manera de evitar problemas con el Hardware es instalarlo desde el principio. Si añadimos Software para nuestro Hardware posteriormente a la instalación nos costará hacerlo funcionar. ¡Incluso puede ser más rápido instalar el sistema desde cero!.

**¿Cómo puedo saber si mi Hardware está soportado por Linux (antes de comprarlo y cometer un error de forma irreparable)?** Fácil: consultando en internet.

**¿Qué sabe Linux de mi Hardware?** El Kernel se encarga de la gestión del Hardware usando herramientas como *Udev* (sistema de nombrado del Hardware), *Hotplug* (mecanismo de avisos), *Dbus* (comunicaciones entre procesos) o *Hal* (capa de abstracción de Hardware), y mapea todo el Hardware en archivos de dispositivos ubicados en los directorios */dev* y */sys*.

Algunos comando usados para conocer el Hardware son:

- lscpu - Información de procesador
- lshw - Lista de Hardware en Linux
- hwinfo - Información del Hardware en Linux
- lspci - Lista PCI
- lsscsi - Listar dispositivos SCSI
- lsusb - Lista de los buses usb y detalles del dispositivo
- inxi - Script mega Bash para usuarios no técnicos
- lsblk - Lista de dispositivos de bloque
- df - espacio en disco de los sistemas de archivos
- fdisk - Informa y permite modificar las particiones de disco

- mount - Permite montar y desmontar y ver sistema de archivo montados
- free - Información de la memoria RAM y Swap
- lsmem - Lista los rangos de memoria disponible.
- hdparm - Información de disco duro
- lsmod - Información de los módulos de los drivers cargados al Kernel

Archivos del directorio */proc*, contienen información accesible usando el comando *cat:*

- Información de CPU
- Información del Kernel de Linux
- Dispositivos Sata / SCSI
- Particiones

**Componentes de un Sistema GNU/Linux** El sistema GNU/Linux se compone de tres cuerpos principales de código, al igual que la mayor parte de las implementaciones de UNIX.

- El núcleo se encarga de mantener todas las abstracciones importantes del sistema operativo, incluidas cosas tales como memoria virtual y procesos.
- Las bibliotecas del sistema definen un conjunto estándar de funciones a través de las cuales las aplicaciones pueden interactuar con el núcleo y que implementan gran parte de la funcionalidad del sistema operativo que no necesita todos los privilegios del código del núcleo.
- Las utilerías del sistema son programas que realizan tareas de administración especializadas individuales. Algunos programas utilitarios pueden invocarse una sola vez para asignar valores iniciales y configurar

algún aspecto del sistema; otros llamados demonios[119] podrían ejecutarse de forma permanente, realizando tareas tales como responder a conexiones de red entrantes, aceptar solicitudes de ingreso al sistema desde terminales, o actualizar archivos de bitácora.

**Principios de Diseño** Unix y posteriormente Linux se diseñaron como sistemas de tiempo compartido. La interfaz estándar con el usuario (el Shell) es sencilla y puede ser sustituida por otra si se desea[120]. El sistema de archivos es un árbol invertido con múltiples niveles, que permite a los usuarios crear sus propios subdirectorios. Cada archivo de datos de usuario es tan solo una secuencia de Bytes.

El sistema UNIX/Linux fue diseñado por programadores para programadores; por ello, siempre ha sido interactivo, y las funciones para desarrollar programas siempre han tenido prioridad. Tales recursos incluyen a los programas *make, gcc, git*, etc.

Los archivos de disco y los dispositivos de Entrada/Salida (E/S) se tratan de la manera más similar posible. Así, la dependencia de dispositivos y las peculiaridades se mantienen en el núcleo hasta donde es posible; aún en el núcleo, la mayor parte de ellas están confinadas a los Drivers de los dispositivos.

Un archivo en Unix/Linux es una secuencia de Bytes. Los diferentes programas esperan distintos niveles de estructura, pero el núcleo no impone ninguna estructura a los archivos. Por ejemplo, la convención para los archivos de texto es líneas de caracteres *ASCII* separadas por un solo carácter de nueva línea (que es el carácter de salto de línea en *ASCII*), pero el núcleo nada sabe de esta convención.

Los archivos se organizan en directorios en estructura de árbol. Los directorios también son archivos que contienen información sobre cómo encontrar otros archivos. Un nombre de camino, trayectoria o ruta de un archivo es una cadena de texto que identifica un archivo especificando una ruta a través

[119] En sistemas UNIX/LINUX se conoce como demonio o Daemon (Disk And Execution Monitor) a un proceso que se ejecuta en segundo plano del sistema operativo, se ejecuta en todo momento y no posee interacción directa con el usuario, también se le conoce genéricamente como servicio o proceso, del cual no percibimos su ejecución. Un demonio realiza una operación específica en tiempos predefinidos o en respuesta a ciertos eventos del sistema.

[120] Algunos de los distintos tipos de Shell son: Shell Bourne, Shell Zsh, Shell C, Shell Korn, Shell Bourne-Again (mejor conocido como Bash, Bourne again shell), etc.

de la estructura de directorios hasta el archivo. Sintácticamente, una trayectoria consiste en nombres de archivos individuales separados por el carácter diagonal. Por ejemplo */usr/local/fuente*, la primera diagonal indica la raíz del árbol de directorios, llamado directorio *raíz* o *root*. El siguiente elemento, *usr*, es un subdirectorio de la raíz, *local* es un subdirectorio de *usr* y *fuente* es un archivo o directorio que está en el directorio *local*. No es posible determinar a partir de la sintaxis del nombre de una trayectoria si la *fuente* es un archivo ordinario o un directorio.

Un archivo puede conocerse por más de un nombre en uno o más directorios. Tales nombres múltiples se denominan enlaces, también se manejan enlaces simbólicos, que son archivos que contienen el nombre de una ruta de otro archivo o directorio. Las dos clases de enlaces también se conocen como enlaces duros y enlaces blandos. Los enlaces blandos (simbólicos), a diferencia de los duros pueden apuntar a directorios y pueden cruzar fronteras de sistemas de archivos (apuntar a otros sistemas de archivos) y el sistema operativo trata igualmente todos los enlaces.

El nombre del archivo *"."* en un directorio es un enlace duro al directorio mismo. El nombre de archivo *".."* es un enlace al directorio padre. Por tanto, si el directorio actual es */usr/jlp/programa*, entonces *../bin/wdf* se refiere a */usr/jpl/bin/wdf*.

Los dispositivos de Hardware tienen nombres en el sistema de archivos. El núcleo sabe que estos archivos especiales de dispositivos o archivos especiales son interfaces con dispositivos, pero de todos modos el usuario accede a ellos prácticamente con las mismas llamadas al sistema que otros archivos.

## 7.1 Sistema de Archivos y Estructura de Directorios

El Sistema de Archivos de Unix/Linux o cualquier sistema de archivos, generalmente es una capa bajo el sistema operativo la cual maneja el posicionamiento de tus datos en el almacenamiento, sin este el sistema no puede saber dónde empieza y termina un archivo.

Éste consiste en un conjunto de archivos, cada uno de los cuales tiene un nombre, hay tres clases de archivos:

- Archivos regulares, que contienen información.
- Directorios o carpetas, que contienen un conjunto de archivos. Los conjuntos de archivos se identifican por el nombre del directorio que los agrupa.

- Archivos especiales FIFO (First In First Out), algunas veces llamados Pipes, que proveen un canal para comunicación entre procesos independientes y que tienen un nombre asociado.

Como la mayoría de los sistemas operativos modernos, Unix/Linux organiza su sistema de archivos como una jerarquía de directorios. La jerarquía de directorios en un árbol invertido, un directorio especial, root '/', es la raíz de la jerarquía. Un directorio puede contener subdirectorios, archivos regulares y archivos especiales. Unix/Linux trata a los subdirectorios como un tipo particular de archivo. Además Unix/Linux ve a un archivo, no importando su tipo, como una sucesión de Bytes, almacenados en dispositivos como Discos, CD, DVDs o unidades de almacenamiento USB.

Un nombre de archivo puede tener hasta 255 caracteres en la mayoría de los sistemas. y contener cualquier carácter excepto '/' o el carácter NULL; sin embargo, algunos caracteres como '&' y ' ' pueden causar problemas por sus significados especiales para su interpretación en la línea de comandos. Los nombres de archivos son sensibles a las mayúsculas y minúsculas.

En los sistemas Unix/Linux, cada usuario tiene un directorio personal, en el cual almacena sus archivos. Dicho directorio se conoce como el hogar (Home) del usuario. Una vez que entres en sesión, cada programa que se ejecutase encuentra situado en un directorio de trabajo.

**Tipos de Sistema de Archivos de GNU/Linux** Cuando intentas instalar Linux, notaras que ofrece distintos sistemas de archivos como los siguientes:

> Ext, Ext2, Ext3, Ext4, JFS, XFS, Btrfs, FAT32, exFAT, NTFS y Swap

Así que, ¿qué son estos sistemas de archivos que ofrece Linux?

- Ext: Antiguo y descontinuado debido a sus limitaciones.
- Ext2: Primer sistema de archivos de Linux que permite 2 Terabytes de datos.
- Ext3: Evolución del Ext2, con actualizaciones y retrocompatibilidad[121].

[121] El único problema que tiene es que los servidores no utilizan este tipo de sistema de archivos debido a que no soporta recuperación de archivos o Snapshots del disco.

- Ext4: Es más rápido y permite archivos mucho más grandes con una velocidad significativa[122].
- F2FS: (Flash-Friendly File System) es un sistema de archivos que toma en cuenta las características de los dispositivos de almacenamiento basados en memorias Flash NAND como las unidades de estado sólido (SSD) y tarjetas eMMC y SD.
- JFS: Sistemas de archivos hechos por IBM. Funcionan bien con archivos grandes y pequeños, pero existen reportes de fallas y los archivos se corrompen después de un largo tiempo de uso.
- XFS: Sistema de archivos creado por SGI que funciona lento con archivos pequeños.
- Btrfs: Hecho por Oracle, no es tan estable como Ext en algunas distribuciones, pero es un buen reemplazo, si es necesario.
- FAT32: Establecido en 1996, es robusto, versátil pero anticuado, sólo permite guardar archivos de hasta 4 GB.
- exFAT: Es una actualización de FAT32 para acabar con la limitación de 4 GB por archivo.
- NTFS: Es una alternativa a FAT32 sin sus limitaciones, usada en discos duros y unidades externas.
- Swap: Es un espacio de intercambio que es utilizado para almacenar datos temporales, reduciendo así el uso de la RAM, normalmente es del doble del tamaño de la RAM del equipo.

Otras opciones de sistemas de archivos son: ZFS, ReiserFS, UFS, FFS, HFS, APFS. Algunas de sus características de los sistemas de archivo más usados son:

| Sistema | Tamaño máx. volumen | Tamaño máx. archivo |
|---|---|---|
| $FAT32$ | 8 TB | 4 GB |
| $NTFS$ | $1.845^{7}$ TB | 256 TB |
| $EXT4$ | $1.153^{6}$ TB | 17.5921 TB |

[122]Es una muy buena opción para discos de estado sólido SSD, además puedes darte cuenta que cuando intentas instalar cualquier distribución de Linux este es el sistema de archivo por defecto que sugiere Linux.

En el sistema de archivos de Linux (Ext2/3/4, UFS, ReiserFS, FFS, XFS, Btrfs, etc.) se tiene asociado un elemento en la tabla que guarda a los archivos y directorios dentro del sistema de archivos, que contiene un número entero único. Este número identifica la ubicación del archivo dentro del área de datos llamado *inodo (i-node)*.

Cada *inodo* contiene información de un fichero o directorio. En concreto, en un *inodo* se guarda la siguiente información:

- El identificador del dispositivo que alberga al sistema de archivos.
- El número de *inodo* que identifica al archivo dentro del sistema de archivos.
- La longitud del archivo en Bytes.
- El identificador de usuario del creador o un propietario del archivo con derechos diferenciados.
- El identificador de grupo de un grupo de usuarios con derechos diferenciados.
- El modo de acceso: capacidad de leer, escribir, y ejecutar el archivo por parte del propietario, del grupo y de otros usuarios.
- Las marcas de tiempo con las fechas de última modificación (*mtime*), acceso (*atime*) y de alteración del propio *inodo* (*ctime*).
- El número de enlaces (*Hard Links*), esto es, el número de nombres (entradas de directorio) asociados con este *inodo*.
- Tabla de direccionamiento donde se detallan los bloques del disco duro en que está almacenado el fichero.

podemos conocer la información del *inodo* de un archivo/directorio/enlace usando:

```
$ stat nombre
```

o bien mediante:

```
$ ls -i
```

si conocemos el *inodo* de un archivo y queremos conocer los nombres de archivo de los enlaces, usamos *(por ejemplo 3432343)*:

```
$ find / -inum 3432343
```

y si queremos conocer el número de inodos de disco podemos usar:

```
$ df -i
```

Además están las Dentries que tienen la función de definir la estructura de un directorio, éstas conjuntamente con los inodos, serán los encargados de representar un fichero en la memoria. Notemos que en cada directorio del sistema existen dos entradas especiales:

- La entrada con un punto (.) hace referencia al propio directorio.
- La entrada con dos puntos (..) hace referencia al inodo del directorio que contiene el directorio (el padre).

El área de datos ocupa el resto del disco y es equivalente a la zona de datos en el *FAT*. En esta zona, como su nombre indica, están almacenados los ficheros y directorios de nuestro sistema.

**Estructura de Directorios en GNU/Linux** Además de los sistemas de archivos que difiere de la de Windows, la estructura de directorios en Linux es distinta, y es necesario conocerla para encontrar ficheros de configuración, instalar ciertos paquetes en el lugar adecuado, localizar las fuentes del Kernel, o la imagen de este, nuestros ficheros personales, entre otros.

De hecho, la Fundación Linux mantiene un estándar de jerarquía del sistema de archivos, este define la estructura de directorios y el contenido de los directorios en las distribuciones Linux. Gracias a este estándar es posible encontrar la misma estructura de directorios en (casi) todas las distribuciones de Linux[123] que a continuación describiremos brevemente:

/ es el directorio principal, la *raíz* o *root*. Contiene el resto de directorios, es decir, todos los demás serían subdirectorios de este (incluso si están en particiones o discos diferentes). Sin duda es el más importante.

[123]Recordemos que Linux se basa en UNIX y, por tanto, toma prestada su jerarquía de sistema de archivos de UNIX. Encontramos una estructura similar en sistemas operativos similares a UNIX, como BSD y MacOS.

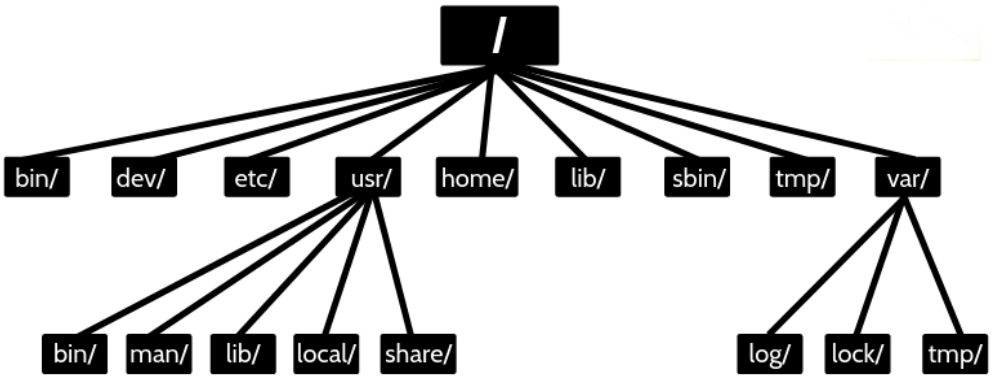



Figura 14: Jerarquía del sistema de archivos de Linux.

**/bin** es el directorio donde se almacenan los binarios, es decir, los programas que emplea el sistema para labores administrativas como los comandos *cp, echo, grep, mv, rm, ls, kill, xkill, ps, su, tar*, etc.

**/sbin** la S es de System, y como su nombre indica, aquí se almacenan los binarios o programas que emplea el propio sistema operativo para tareas de arranque, restauración, etc. Por ejemplo, *fsck, mount, mkfs, reboot, swapon.*

**/boot** es el directorio de arranque, donde está la o las imágenes del Kernel Linux que se cargarán durante el arranque, también directorios y configuración del propio gestor de arranque.

**/etc** muy importante para el administrador, ya que aquí residen los ficheros de configuración de los componentes del sistema y otros programas instalados.

**/dev** es un directorio muy especial donde se encuentran los dispositivos de bloques o caracteres, es decir, ficheros que representan la memoria, particiones, discos, dispositivos de Hardware, etc. Ya sabes que en LINUX y UNIX todo es un archivo, y no unidades como en Windows. Por ejemplo, el disco duro o particiones serían */dev/sda, /dev/hda, /dev/scd*, /dev/fd/, etc.

**/proc** es otro directorio muy especial, más que un directorio es una interfaz por decirlo de un modo sencillo. Y aquí el sistema nos presenta los procesos[124] como directorios numerados con el identificador de procesos *PID* (Process ID). Dentro de cada uno de ellos estaría toda la información necesaria para la ejecución de cada proceso en marcha. Además, encontrarás ficheros de los que extraer información importante, como *cpuinfo, meminfo*, etc. Es precisamente de estos ficheros de los que extraen información algunos comandos que usamos habitualmente, como por ejemplo, cuando hacemos uso de *free* para consultar la memoria disponible, este comando realmente estaría mostrando el contenido de */proc/meminfo* de una forma ordenada.

**/media o /mnt** son los directorios donde se establecen generalmente los puntos de montaje. Es decir, cuando insertamos algún medio extraíble o recurso de red compartido, etc., que hayamos montado, estaría aquí si lo hemos puesto como punto de montaje. El primero es más específico para medios que se montan de una forma temporal.

**/home** es el directorio para los usuarios estándar. Por ejemplo, aquí se almacenan dentro de directorios separados (uno para cada usuario con su nombre), los ficheros personales. Por ejemplo, */home/antonio* sería mi directorio personal.

**/lib o /lib64** es donde se alojan las bibliotecas necesarias para los programas ejecutables presentes en el sistema. En /lib64 estarían las de las aplicaciones de 64 bits y en */lib* estarían las aplicaciones de 32 bits.

**/opt** es un directorio que almacenará los paquetes o programas instalados en el sistema que son de terceros. Por ejemplo, si instalamos algún antivirus, Chrome, Arduino IDE o ciertos paquetes grandes, suelen instalarse aquí.

[124] Existen procesos activos y dormidos, procesos huérfanos y procesos zombies.

Los procesos activos son aquellos que están en ejecución en el sistema y los procesos dormidos son aquellos que esperan algún recurso o señal para continuar con su ejecución.

Los procesos huérfanos son aquellos que se siguen ejecutando a pesar que su proceso padre concluyó su operación.

Los procesos zombies es un proceso que ha concluido pero aún están presentes en la tabla de procesos.

**/root** no hay que confundirlo con /, una cosa es el directorio raíz o *root* y otra muy diferente */root*. En este caso, se puede asemejar a un */home* pero es exclusivo para el usuario *root* o usuario administrador.

**/srv** almacena ficheros y directorios relativos a servidores que tienes instalados en el sistema, como Web, FTP, CVS, etc.

**/sys** junto con */dev* y */proc*, es otro de los especiales. Y como */proc*, realmente no almacena nada, sino que es una interfaz también. En este caso, son ficheros virtuales con información del Kernel e incluso, se pueden emplear algunos de sus ficheros para configurar ciertos parámetros del Kernel.

**/tmp** es el directorio para ficheros temporales de todo tipo. Es empleado por los usuarios para almacenar de forma temporal ciertos ficheros o incluso para almacenar Caché o ciertos ficheros volátiles de navegadores Web, etc. No obstante, hay otro directorio para lo mismo en */var/tmp*.

**/var** se trata de un directorio con directorios y ficheros que suelen crecer de tamaño, como bases de datos, *logs*, etc. Es precisamente los *logs* o registros del sistema por lo que es más popular este directorio, y allí encontrarás muchísima información de todo lo que ocurre en el sistema: */var/logs/*. Dentro de dicho directorio encontrarás separados por directorios, los *logs* de multitud de Software, incluido el sistema.

**/usr** son las siglas de *User System Resources*, y actualmente almacena ficheros de solo lectura relativos a utilidades del usuario, como los paquetes que instalamos mediante el gestor de paquetes en nuestra distribución. Dentro hay como una jerarquía de árbol de directorios vistos hasta ahora (casi todos) como si de un segundo nivel se tratase. Vas a encontrar */usr/bin*, */usr/lib*, */usr/sbin*, */usr/src,* etc., que por lo dicho anteriormente y sus nombres, es intuitivo saber lo que almacenan. Por ejemplo en /usr/src es donde permanecerán los ficheros de código fuente.

Ten en cuenta que no todas las distribuciones de Linux siguen este esquema y puede haber pequeñas variaciones, pero si se adaptan al estándar, no tendrás problemas al navegar por la estructura de archivos.

**Rutas Absolutas y Relativas** cuando se empieza a manejar un intérprete de comandos, una de las cosas que más cuesta es acostumbrarte a encontrar y hacer referencia a elementos del sistema de ficheros. Mientras que en un entorno gráfico tenemos que hacer Clic en carpetas y subcarpetas hasta llegar al elemento deseado, en el intérprete de comandos tendremos que conseguir lo mismo, pero indicando el lugar mediante una cadena de texto compuesta por los nombres de las carpetas que hay que recorrer hasta el lugar donde se encuentra el elemento deseado. Según el sistema cada nombre de carpeta se separa por un carácter especial, que en Linux es la diagonal ( / ).

Estas rutas serán usadas por los comandos para saber dónde encontrar los elementos sobre los que se tiene que realizar la acción correspondiente[125]. Hay dos formas de utilizar rutas, una es de forma absoluta y la otra de forma relativa. Vamos a explicar la diferencia a continuación:

**Rutas Absolutas** el sistema de ficheros es una estructura jerárquica que en el caso de Linux tiene una raíz que se indica cuando se pone solamente el carácter diagonal / . En la raíz están los directorios principales del sistema que a su vez tendrán subdirectorios en su interior. Cuando quiero indicar dónde se encuentra un elemento usando una ruta absoluta, tendré que indicar todos los directorios por los que hay que pasar empezando desde la raíz del sistema. O lo que es lo mismo, siempre empezarán por /. Veamos algunos ejemplos:

```
/etc/apt/sources.list
/var/log/syslog
/home/alumno/.bashrc
/usr/bin/
```

estas rutas suelen ser bastante largas, pero tienen como ventaja que funcionan siempre, independientemente del lugar desde el que se ejecute la orden[126].

[125] Por ejemplo, si quiero posicionarme en un directorio determinado, utilizaré el comando cd y para indicar el sitio adonde quiero ir usaré una ruta, por ejemplo cd /home/. El comando cp copia elementos, en este caso necesitaremos dos rutas una para el origen (elemento que quiero copiar) y otra para el destino (elemento nuevo que voy a crear o lugar donde voy a dejar las copias). Por lo tanto podría poner:
cp /etc/passwd /home/copia_passwd.

[126] Es muy recomendable utilizar la facilidad que brinda el BASH de completar el nombre de un elemento del sistema de ficheros pulsando la tecla tabulador. Ahorrará mucho tiempo y errores.

**Rutas Relativas** las rutas relativas indican el camino para encontrar un elemento, pero basándonos en el directorio desde el que se ejecuta la orden. Son mucho más cortas que las absolutas, pero para saber si son correctas o no, tenemos que saber siempre desde dónde se han utilizado.

Un atajo fundamental para la construcción de rutas relativas es conocer que al escribir .. en la ruta hace referencia al directorio padre. Por lo tanto si ejecuto:

```
$ cd ..
```

estoy dando la orden de cambiar de directorio al padre del actual, es decir, al que está justo antes en la estructura jerárquica. El único elemento que no tiene padre es la propia raíz del sistema ( / ).

Las rutas relativas harán referencia a un elemento que se encuentre en el directorio desde el que ejecutamos la orden, o usará los dos puntos para ascender a directorios superiores. Siempre que sean correctos, podemos combinarlos de la forma que necesitemos separando cada directorio por una diagonal. Por ejemplo, una ruta correcta podría ser: ../../fotos/personales/

**Permisos de Archivos y Directorios** GNU/Linux, al ser un sistema diseñado fundamentalmente para trabajo en red, la seguridad de la información que almacenamos en nuestros equipos (y no se diga en los servidores) es fundamental, ya que muchos usuarios tendrán o podrán tener acceso a parte de los recursos de Software (tanto a aplicaciones, como a información) y Hardware que están gestionados en estos equipos de cómputo. ¿Ahora podemos ver la necesidad de un sistema de permisos?

En GNU/Linux, los permisos o derechos que los usuarios pueden tener sobre determinados archivos contenidos en él se establecen en tres niveles claramente diferenciados. Estos tres niveles son los siguientes:

- Permisos del propietario.
- Permisos del grupo.
- Permisos del resto de usuarios (o también llamados "los otros").

Para tener claros estos conceptos, en los sistemas en red siempre existe la figura del administrador, superusuario o root. Este administrador es el encargado de crear y dar de baja a usuarios, así como también, de establecer

los privilegios que cada uno de ellos tendrá en el sistema. Estos privilegios se establecen tanto para el directorio de trabajo (Home) de cada usuario como para los directorios y archivos a los que el administrador decida que el usuario pueda acceder.

**Permisos del propietario** el propietario es aquel usuario que genera o crea un archivo/carpeta dentro de su directorio de trabajo, o en algún otro directorio sobre el que tenga derechos. Cada usuario tiene la potestad de crear, por defecto, los archivos que quiera dentro de su directorio de trabajo. En principio, él y solamente él será el que tenga acceso a la información contenida en los archivos y directorios que hay en su directorio trabajo o Home -bueno, no es del todo cierto esto, ya que el usuario *root* siempre tiene acceso a todos los archivos y directorios del sistema-.

**Permisos del grupo** lo más normal es que cada usuario pertenezca a un grupo de trabajo. De esta forma, cuando se gestiona un grupo, se gestionan todos los usuarios que pertenecen a éste. Es decir, es más fácil integrar varios usuarios en un grupo al que se le conceden determinados privilegios en el sistema, que asignar los privilegios de forma independiente a cada usuario.

**Permisos del resto de usuarios** por último, también los privilegios de los archivos contenidos en cualquier directorio, pueden tenerlos otros usuarios que no pertenezcan al grupo de trabajo en el que está integrado el archivo en cuestión. Es decir, a los usuarios que no pertenecen al grupo de trabajo en el que está el archivo, pero que pertenecen a otros grupos de trabajo, se les denomina resto de usuarios del sistema.

**¿cómo puedo identificar todo esto?** sencillo, abre una terminal y escribe lo siguiente:

```
$ ls -l
```

entregará varias salidas como esta:

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

- \- Archivo

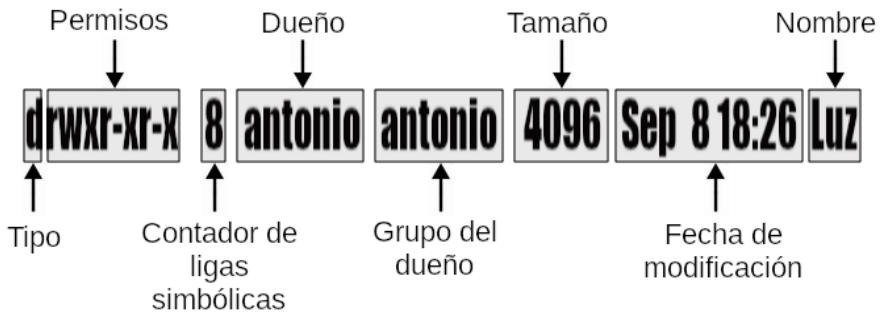



Figura 15: Estructura de permisos en la salida de: ls -l

- d Directorio
- l Liga simbólica
- s Socket (paso de datos entre dos procesos)
- p Pipe
- c Dispositivo de carácter (comunicación de Hardware)
- b Dispositivo de bloque (para el control de dispositivos)

Los siguientes 9 restantes, representan los permisos del archivo y deben verse en grupos de 3 y representan:

- - Sin permiso
- r Permiso de lectura
- w Permiso de escritura
- x Permiso de ejecución

Los tres primeros representan los permisos para el propietario del archivo, los tres siguientes son los permisos para el grupo del archivo y los tres últimos son los permisos para el resto del mundo u otros.

Luego viene el contador de ligas simbólicas, el dueño del archivo, grupo al que pertenece, el tamaño en Bytes, la fecha de última modificación y finalmente el nombre del archivo o directorio.

**Permisos Especiales** Aún hay otro tipo de permisos que hay que considerar. Se trata del Bit de permisos SUID (Set User ID), el Bit de permisos SGID (Set Group ID) y el Bit de permisos de persistencia (Sticky Bit).

**Setuid** el Bit Setuid es asignable a ficheros ejecutables, y permite que cuando un usuario ejecute dicho fichero, el proceso adquiera los permisos del propietario del fichero ejecutado. El ejemplo más claro de fichero ejecutable y con el Bit Setuid es:

```
$ su
```

Podemos ver que el Bit está asignado como "s" en:

```
$ ls -l /bin/su
```

Para asignar[127] este Bit a un fichero sería:

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

Y para quitarlo:

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

**Setgid** el Bit Setid permite adquirir los privilegios del grupo asignado al fichero, también es asignable a directorios. Esto será muy útil cuando varios usuarios de un mismo grupo necesiten trabajar con recursos dentro de un mismo directorio.

Para asignar este Bit hacemos lo siguiente:

```
$ chmod g+s /carpeta_compartida
```

Y para quitarlo:

```
$ chmod g-s /carpeta_compartida
```

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

**Sticky** este Bit suele asignarse en directorios a los que todos los usuarios tienen acceso, y permite evitar que un usuario pueda borrar ficheros/directorios de otro usuario dentro de ese directorio, ya que todos tienen permiso de escritura.

Podemos ver que el Bit está asignado como "t" en el directorio */tmp*.

Para asignar este Bit hacemos lo siguiente:

```
# chmod o+t /tmp
```

Y para quitarlo:

```
# chmod o-t /tmp
```

**Entrada y Salida Estándar** los procesos pueden abrir archivos a discreción, pero la mayor parte de los procesos esperan a que estén abiertos tres descriptores de archivos (numerados 0, 1 y 2) cuando inician. Estos descriptores se conocen como entrada estándar (0), salida estándar (1) y error estándar (2). Es común que los tres estén abiertos en la terminal del usuario. Así, el programa puede leer lo que el usuario teclea leyendo la entrada estándar, y puede enviar salidas a la pantalla del usuario escribiendo en la salida estándar. El descriptor de archivo de error estándar también está abierto para escritura, y se usa para los mensajes de error.

**Standard Input** la Entrada estándar, en inglés *standard input* (mejor conocido como *stdin*) es el mecanismo por el cual un usuario le indica a los programas la información que estos deben procesar. Por omisión, el teclado es la entrada estándar. La entrada estándar representa los datos que necesita una aplicación para funcionar, como por ejemplo un archivo de datos o información ingresada desde la terminal y es representado en la terminal como el tipo 0.

**Standard Output** la Salida estándar, en inglés *standard output* (mejor conocido como *stdout*) es el método por el cual el programa puede comunicarse con el usuario. Por omisión, la salida estándar es la pantalla donde se ejecutaron las instrucciones. La salida estándar es la vía que utilizan las aplicaciones para mostrarte información, allí podemos ver el progreso o simplemente los mensajes que la aplicación quiera darte en determinado momento y es representado en la terminal como el tipo 1.

**Standard Error** por último existe un flujo conocido como Error estándar, en inglés *standard error output* (mejor conocido como *stderr*) que es utilizado por las instrucciones para desplegar mensajes de error que surjan durante el transcurso de su ejecución. Al igual que *stdout*, el error estándar será la pantalla donde se procesaron las instrucciones. El error estándar es la forma en que los programas te informan sobre los problemas que pueden encontrarse al momento de la ejecución y es representado en la terminal como el tipo 2.

**Redirección Mediante Pipe** las tuberías (Pipe) unen la salida estándar de un comando con la entrada estándar de otro, es decir, la salida de un comando se emplea como entrada del siguiente. Para ello se emplea el símbolo pipe "|". El uso de tuberías evita la generación constante de archivos intermedios reduciendo el tiempo de procesamiento.

**Redirección Hacia el Dispositivo Nulo** en GNU/Linux, */dev/null* es un archivo especial al que se envía cualquier información que quiera ser descartada. Aunque al principio no lo parezca, el uso del dispositivo nulo es muy útil.

## 7.2 Interfaz de Usuario

Tanto el programador como el usuario de Linux manejan principalmente el conjunto de programas de sistemas que se han escrito y están disponibles para ejecutarse. Estos programas efectúan llamadas al sistema operativo necesarias para apoyar su función, pero las llamadas al sistema en sí están contenidas dentro del programa y no tienen que ser obvias para el usuario.

GNU/Linux puede funcionar tanto en entorno gráfico[128] (Graphical User Interface, GUI) como en modo línea de comandos (Command-Line Interface,

[128] Un servidor de pantalla en GNU/Linux es un programa que es responsable de la coordinación de entrada y salida de sus clientes, hacia y desde en resto del sistema operativo, y entre el Hardware y el sistema operativo. El servidor de visualización proporciona el marco para un entorno gráfico para que se pueda utilizar el Mouse y el teclado para interactuar con las aplicaciones. El servidor de pantalla se comunica con sus clientes a través del protocolo del servidor de pantalla como: X11, Wayland o Mir. El servidor de visualización es un componente clave en cualquier interfaz gráfica de usuario, específicamente el sistema de ventanas. No debemos confundir el servidor de visualización con el entorno de escritorio. El entorno de escritorio utiliza un servidor de pantalla debajo.

CLI) también conocida como consola o *Shell*. La consola es común en distribuciones para servidores, mientras que la interfaz gráfica está orientada al usuario final del hogar, como empresarial.

Los entornos de escritorio pertenecen a la interfaz gráfica, son un conjunto de programas conformado por ventanas, íconos, imágenes y muchas aplicaciones que facilitan el uso de la computadora. Los entornos de escritorio más populares en GNU/Linux son: GNOME, KDE, LXQt, LXDE, Xfce, Unity, MATE, Cinnamon, Pantheon, Deepin, Budgie, PIXEL, Enlightenment, Trinity, Moksha, Ukui, etc. Dependiendo de la distribución se pueden tener uno o más escritorios instalados, por ejemplo en Debian GNU/Linux están disponibles los más usados y si el usuario los instala, puede decidir al iniciar sesión cuál usar.

### 7.2.1 Interfaz Gráfica de Usuario

La interfaz gráfica de usuario es un tipo de visualización que permite al usuario elegir comandos, iniciar programas y ver listas de archivos y otras opciones utilizando las representaciones visuales (iconos) y las listas de elementos del menú. Las selecciones pueden activarse bien a través del teclado o con el ratón.

Para los autores de aplicaciones, las interfaces gráficas de usuario ofrecen un entorno que se encarga de la comunicación con la computadora. Esto hace que el programador pueda concentrarse en la funcionalidad, ya que no está sujeto a los detalles de la visualización ni a la entrada a través del ratón o del teclado. También permite a los programadores crear programas que realicen de la misma forma las tareas más frecuentes, cómo guardar un archivo, porque la interfaz proporciona mecanismos estándar de control como ventanas y cuadros de diálogo. Otra ventaja es que las aplicaciones escritas para una interfaz gráfica de usuario son independientes de los dispositivos: a medida que la interfaz cambia para permitir el uso de nuevos dispositivos de entrada y salida, como un monitor de pantalla grande o un dispositivo óptico de almacenamiento, las aplicaciones pueden utilizarlos sin necesidad de cambios.

**¿Qué es un Entorno de Escritorio?** un entorno de escritorio es un conjunto de Software para ofrecer al usuario de una computadora una interacción amigable y cómoda. El Software es una solución completa de interfaz gráfica de usuario, ofrece iconos, barras de herramientas, carpetas, fondos de

pantalla y Widgets de escritorio, e integración entre aplicaciones con habilidades como, arrastrar y soltar. En general cada entorno de escritorio se distingue por su aspecto y comportamiento particular, aunque algunos tienden a imitar características de escritorios ya existentes. El primer entorno moderno de escritorio que se comercializó fue desarrollado por Xerox en los años 80. Actualmente el entorno más conocido es el ofrecido por la familia Windows aunque existen otros como los de: Macintosh (Classic y Cocoa) y de código abierto como: GNOME, KDE, LXQt, LXDE, Xfce, Unity, MATE, Cinnamon, Pantheon, Deepin, Budgie, PIXEL, Enlightenment, Trinity, Moksha, Ukui, etc.

**¿Qué son los Gestores de Ventanas?** un gestor de ventanas o en inglés Windows Manager, es un programa que controla la ubicación y apariencia de las aplicaciones bajo el sistema X Windows. Las computadoras suelen ofrecer una interfaz gráfica de usuario que facilita la interacción con el sistema operativo. Las plataformas Windows y Macintosh ofrecen métodos de visualización y control de las ventanas e interacción con las aplicaciones, estandarizados por sus vendedores. En cambio el sistema gráfico X Windows, popular en el ámbito de sistemas Unix y similares, como GNU/Linux, permite al usuario escoger entre varios gestores según sus gustos o necesidades. Los gestores de ventanas difieren entre sí de muchas maneras, incluyendo apariencia, consumo de memoria, opciones de personalización, escritorios múltiples o virtuales y similitud con ciertos entornos de escritorio ya existentes. Estos se dividen en 3 tipos, que son los siguientes:

- Stacking: Aquellos que imitan las apariencias y funcionalidades de Windows y Mac OS X, por ende, gestionan las ventanas como pedazos de papel en un escritorio, que pueden ser apiladas unas sobre otras.
- Tiling: Aquellos de tipo "mosaico" donde las ventanas no se superponen, y donde suelen hacerse un uso muy extenso de atajos de teclado, y se obtiene una menor dependencia del uso del ratón.
- Dynamics: Aquellos que permiten alterar dinámicamente el diseño de las ventanas entre mosaicos o flotantes.

Las acciones asociadas al gestor de ventanas suelen ser, abrir, cerrar, minimizar, maximizar, mover, escalar y mantener un listado de las ventanas abiertas. Es también muy común que el gestor de ventanas integre elementos

como: el decorador de ventanas, un panel, un visor de escritorios virtuales, iconos y un tapiz.

**Entornos de Escritorios más Conocidos:**

**KDE** (https://kde.org)
proyecto que fue iniciado en octubre de 1996 por el programador alemán Matthias Ettrich, quien buscaba crear una interfaz gráfica unificada para sistemas Unix. En sus inicios imitó a CDE (Common Desktop Environment), un entorno de escritorio utilizado por varios Unix. Este es un entorno de escritorio con multitud de aplicaciones e infraestructura de desarrollo para diversos sistemas operativos como GNU/Linux, Mac OS X, Windows, etc. Los principales componentes de Software elaborados por KDE se agrupan bajo el nombre KDE Frameworks, KDE Plasma y KDE Applications. Las aplicaciones KDE están traducidas a aproximadamente 88 idiomas y están construidas con los principios de facilidad de uso y de accesibilidad moderna en mente y funcionan de forma completamente nativa en GNU/Linux, BSD, Solaris, Windows y Mac OS X.

**GNOME** (https://www.gnome.org)
este proyecto fue iniciado por los programadores mexicanos Miguel de Icaza y Federico Mena, forma parte oficial del proyecto GNU. Nació como una alternativa a KDE bajo el nombre de GNU Network Object Model Environment (Entorno de Modelo de Objeto de Red GNU). Actualmente, GNOME se está traduciendo a 193 idiomas. Está disponible en las principales distribuciones GNU/Linux, incluyendo Fedora, Debian, Ubuntu, Manjaro Linux, Red Hat Enterprise Linux, SUSE Linux Enterprise, CentOS, Oracle Linux, Arch Linux, Gentoo, SteamOS, entre otras. También, se encuentra disponible en Solaris, un importante sistema operativo UNIX y en Sistemas operativos Unix-like como FreeBSD.

**Xfce** (https://www.xfce.org)
es un entorno de escritorio libre para sistemas tipo Unix como GNU/Linux, BSD, Solaris y derivados. Su objetivo es ser rápido y ligero, sin dejar de ser visualmente atractivo y fácil de usar. Consiste en varios componentes empaquetados por separado que en conjunto proporcionan la funcionalidad completa del entorno de escritorio, pero se pueden seleccionar por separado

para que el usuario pueda adaptar el ambiente de trabajo a sus necesidades. Puede ser instalado en varias plataformas como: Linux, NetBSD, FreeBSD, OpenBSD, Solaris, Cygwin y MacOS X, sobre x86, PPC, Sparc, Alpha.

**LXDE** (https://lxde.org)
es un entorno de escritorio libre para Unix y otras plataformas POSIX[129], como Linux o BSD. El nombre corresponde a "Lightweight X11 Desktop Environment", que en español significa Entorno de escritorio X11 ligero. Es un proyecto que apunta a entregar un nuevo entorno de escritorio ligero y rápido. No está diseñado para ser tan complejo como KDE o GNOME, pero es bastante usable y ligero, y mantiene una baja utilización de recursos y energía. A diferencia de otros ambientes de escritorio, los componentes no se integran firmemente. Al contrario, los componentes son independientes, y cada uno de ellos se puede utilizar independientemente con muy pocas dependencias. Usa Openbox como gestor de ventanas predeterminado y apunta a ofrecer un escritorio ligero y rápido basado en componentes independientes que pueden ser utilizados en otros entornos.

**LXQt** (https://lxqt.github.io)
es un entorno de escritorio libre y de código abierto para Linux, resultado de la fusión entre los proyectos LXDE y Razor-qt. LXQt conjuga la filosofía de LXDE con las librerías QT, usa el gestor de ventanas del escritorio Razor-qt, es un escritorio muy liviano y es considerado por muchos como el sucesor de LXDE.

**Gestores de Ventanas más Conocidos:**

**Enlightenment** (https://www.enlightenment.org)
también conocido simplemente como E, es un gestor de ventanas X11 ligero para UNIX y GNU/Linux. Uno de sus objetivos es llegar a ser un

[129] Significa Portable Operating System Interface. Consiste en una familia de estándares especificadas por la IEEE con el objetivo de facilitar la interoperabilidad de sistemas operativos. Además, POSIX establece las reglas para la portabilidad de programas. Por ejemplo, cuando se desarrolla Software que cumple con los estándares POSIX existe una gran probabilidad de que se podrá utilizar en sistemas operativos del tipo Unix. Si se ignoran tales reglas, es muy posible que el programa o librería funcione bien en un sistema dado pero que no lo haga en otro.

entorno de escritorio completo. Es muy configurable y muy atractivo visualmente. Durante un tiempo fue el gestor de ventanas de GNOME.

**IceWM** (https://ice-wm.org)
es un gestor de ventanas para el X Windows System gráfico de infraestructura escrito por Marko Macek. Se ha codificado desde cero en C++ y es liberado bajo GNU. IceWM es ligero y personalizable. Se puede configurar a partir de archivos de texto almacenados en un directorio Home del usuario, haciendo fácil la personalización y copia de configuraciones. Posee soporte oficial para menús de GNOME y KDE previamente disponible como un paquete separado.

**Windows Maker** (https://www.windowsmaker.org)
es un popular gestor de ventanas para X Windows System diseñado para emular NeXT del GUI como OpenStep compatible, ha sido descrito como "uno de los más útiles y universales gestores de ventanas disponibles. Windows Maker tiene la reputación de ser rápido, eficiente y altamente estable y es muy popular entre las soluciones de código abierto para su uso tanto en nuevas como en viejas máquinas. Como con la mayoría de gestores de ventanas, soporta un montón de temas disponibles.

**Fluxbox** (http://fluxbox.org)
es un gestor de ventanas X basado en Blackbox 0.61.1.Su objetivo es ser ligero y personalizable, y cuenta con un apoyo mínimo de iconos gráficos. Su interfaz de usuario sólo tiene una barra de tareas y un menú al que se puede acceder pulsando con el botón derecho sobre el escritorio. Todas las configuraciones básicas están controladas por ficheros de texto. Fluxbox puede mostrar algunos Eye Candies: colores, gradientes, bordes y una que otra apariencia básica. Las versiones recientes soportan esquinas redondeadas y elementos gráficos. Fluxbox también tiene varias características de las cuales Blackbox carece, incluyendo ventanas con pestañas y un título configurable.

**Openbox** (http://openbox.org/new/)
es un famoso gestor de ventanas libre para el sistema de ventanas X, licenciado bajo la GNU General Public License. Openbox fue originalmente derivado de Blackbox 0.65.0, pero ha sido totalmente reescrito en el lenguaje de programación C y desde la versión 3.0 no se basa en ningún código de

Blackbox. Su sistema de menú tiene un método para utilizar los menús dinámicos.

**Otros** existen muchos otros gestores de ventanas que pudiéramos mencionar, tales como: 9wm, Awesome, AfterStep, Scwm, Blackbox, Bspwm, Byobu, Cinnamon, Cwm, Deepin, Dwm, Fvwm, Icewm, Jwm, Kahakai, Lumina, Qtile, Wmii, WindowLab, Ratpoison, Sawfish, Sway, wm2, Wmx, StumpWM, Twm, Waimea, Xmonad, I3, E16.

### 7.2.2 Línea de Comandos y Órdenes

La mayoría de los usuarios de ordenadores de hoy sólo están familiarizados con la interfaz gráfica de usuario o GUI, los vendedores y los expertos les han enseñado que la interfaz de línea de comandos o CLI es una cosa espantosa del pasado. Es una pena, porque una buena interfaz de línea de comandos es una maravillosa y expresiva forma de comunicarse con el ordenador, muy parecida a lo que el lenguaje escrito es para los seres humanos. Se ha dicho que "las interfaces gráficas de usuario hacen fáciles las tareas fáciles, mientras que las interfaces de línea de comandos hacen posibles las tareas difíciles" y eso es muy cierto aún hoy.

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

**Emuladores de Terminal** Cuando utilizamos una interfaz gráfica de usuario, necesitamos otro programa llamado emulador de terminal para interactuar con el Shell. Si buscamos en nuestros menús de escritorio, probablemente encontraremos uno. KDE usa Konsole y GNOME usa Gnome-terminal, aunque es probable que se llame simplemente "Terminal" en nuestro menú. Hay muchos otros emuladores de terminal disponibles para Linux, pero todos hacen básicamente lo mismo; nos dan acceso al Shell.

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

**El Intérprete de Órdenes de Consola** o Shell es un programa informático, cuya función consiste en interpretar órdenes, y un lenguaje de consola. El más conocido es Bash[130]. Es una Shell de Unix compatible con POSIX[131] y el intérprete de comandos por defecto en la mayoría de las distribuciones GNU/Linux, además de Mac OS. También se ha llevado a otros sistemas como Windows y Android.

Algunas alternativas a Bash son: SH (Bourne Shell), TCSH/CSH (C Shell), KSH (Korn Shell), Fish (Friendly Interactive Shell), ZSH (Z Shell), TSCH (TS Shell), Dash (Debian Almquist Shell), DSH (Distributed Shell)[132].

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

- $ para usuario sin privilegios

[130] Su nombre es un acrónimo de Bourne-again Shell ("Shell Bourne otra vez"), haciendo un juego de palabras (Born-Again significa "nacido de nuevo") sobre la Bourne Shell (sh), que fue uno de los primeros intérpretes importantes de Unix.

[131] Significa Portable Operating System Interface. Consiste en una familia de estándares especificadas por la IEEE con el objetivo de facilitar la interoperabilidad de sistemas operativos. Además, POSIX establece las reglas para la portabilidad de programas. Por ejemplo, cuando se desarrolla Software que cumple con los estándares POSIX existe una gran probabilidad de que se podrá utilizar en sistemas operativos del tipo Unix. Si se ignoran tales reglas, es muy posible que el programa o librería funcione bien en un sistema dado pero que no lo haga en otro.

Sin ir más lejos, examinemos en la última versión de la especificación IEEE 1003 (designación formal de los estándares POSIX). Al buscar la referencia sobre el comando du, podemos ver que solamente las opciones -a, -H, -k, -L, -s, y -x son obligatorias. Otras, tales como -c y -h, aparecen en la versión de du provista por el proyecto GNU. Esto significa que si utilizamos estas últimas en un Script desarrollado en Linux e intentamos hacerlo funcionar en FreeBSD o AIX, es muy probable que no funcione como esperamos.

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

```
$ echo $0
```

- # para el administrador, conocido como *root*

**Sintaxis Estándar de Comandos** Se ha definido un estándar para comandos POSIX (Portable Operating System Interfaz) por la IEEE, pero no todos los comandos la siguen, Muchos comandos definidos por POSIX tienen una forma moderna y una sintaxis obsoleta, por compatibilidad con sistemas viejos. Por ejemplo el estándar especifica que las opciones de los comandos (no así, los operandos) pueden aparecer en cualquier orden, pero algunos comandos pueden requerir que las opciones aparezcan en un orden particular y muchas veces esto no está mencionado explícitamente el la página del manual del comando.

Un comando consiste en una sucesión de palabras separadas por espacios en blanco. La primera palabra es el nombre del comando y las palabras subsecuentes son sus argumentos u opciones. Los operandos consisten de las opciones del programa, si hay alguna, seguidas de sus operandos, si hay alguno. Las opciones controlan o modifican lo que el comando hace. Los operadores especifican nombres de trayectorias o con lo que va a trabajar el comando.

Las opciones se especifican con una sucesión de palabras. Cada palabra es un grupo de opciones o una opción argumento. Las opciones generalmente se denotan por letras. Se pueden escribir en cualquier orden y se pueden combinar varias opciones en un solo grupo (siempre y cuando no reciban ningún argumento). Cada grupo de opciones está precedido por un guión (-). Por ejemplo, los comandos:

```
$ ls -al
$ ls -l -a
```

son equivalentes e indican que el comando *ls* (list archives) sea llamado con las opciones *a* (all files) y *l* (long listing).

Hay otras convenciones comunes para especificar argumentos:

- Dos guiones (- -) indica el final de las opciones. Esto es útil cuando quieres pasar argumentos que empiecen con un - a un comando, por ejemplo:

  ```
  $ rm - - -miArchivo
  ```

es una forma de indicarle que el archivo es *-miArchivo*, también podemos usar:

```
$ rm ./-miArchivo
```

- Un solo guión representa a la entrada estándar en un contexto en que el programa espera una trayectoria. Por ejemplo, el comando:

```
$ diff - miArchivo
```

encuentra las diferencias entre la entrada estándar y el archivo *miArchivo*

**Metacarácter o Shell Globbing** los metacaracteres o Shell Globbing son caracteres que tienen un significado especial en la línea de comandos, para usar estos caracteres como caracteres ordinarios en la línea de comandos, debes marcarlos de manera especial para que el Shell no los interprete. Por ejemplo:

- La diagonal inversa (\) siempre sirve como carácter de escape para uno o más caracteres que le suceden, lo cual le da a esos caracteres un significado especial. Si un carácter es un metacaracter, el significado especial es el carácter mismo. Por ejemplo \\ usualmente significa una solo \ y \$ el signo de pesos. En estos casos, la diagonal inversa le quita su significado a los caracteres.

- Cuando un texto se encierra entre comillas (" "), la mayoría de los de los metacaracteres que estén en dicho texto aon tratados como caracteres normales, excepto por $, que generalmente indica sustituciones a realizar.

- Otra forma de citar muy similar a la anterior, pero más fuerte es usar comillas sencillas (' ') ya que ni siquiera el carácter $ es interpretado

También existen otros metacaracteres que son comodines que el sistema permite usar para especificar los nombres de archivos que satisfacen el filtro especificado a la hora de buscar, eliminar o filtrar nombres de archivo, estos metacaracteres son: *, ?, [ ] y [^][133].

[133] Véase también el uso de las secuencias (véase 8.7).

- ***** Se utiliza para reemplazar cero o más caracteres. Puede ser sustituido por cualquier cadena de caracteres, ejemplo:

  $ ls a*.pdf

- **?** Sustituye un carácter cualquiera, ejemplo:

  $ ls a?chivo.pdf

- [ ] Se usa para definir rangos o conjuntos de caracteres a localizar, para definir los rangos se debe usar el guión -, si son varios caracteres se separan por coma, ejemplo:

  $ ls [Aa]rchivo[0-9].pdf

- [^] o [**!**]Este caso es contrario al anterior, este representa que se busque algo exceptuando lo que se encuentra entre los corchetes, también trabaja con rangos, ejemplo:

  $ls [^A]rchivo.pdf

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

$ su antonio

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

$ su - administracion

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

$ su

El usuario root en GNU/Linux es el usuario que tiene acceso administrativo al sistema. Los usuarios normales no tienen este acceso por razones de seguridad. Sin embargo, en múltiples sistemas derivados de Debian GNU/Linux no incluye el usuario root (por ejemplo en todos los derivados de Ubuntu). En su lugar, se da acceso administrativo a usuarios individuales, que pueden utilizar la aplicación *sudo* para realizar tareas administrativas. La primera cuenta de usuario que creó en su sistema durante la instalación tendrá, de forma predeterminada, acceso a *sudo*.

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

Para usar *sudo* en la línea de comandos, simplemente escriba *sudo* antes del comando que desea ejecutar, *sudo* le pedirá su contraseña[134]:

```
$ sudo apt update
```

**Trabajando en Línea de Comandos** Las órdenes se pueden agrupar en varias categorías; la mayor parte de ellas están orientadas hacia archivos o directorios. Por ejemplo los programas de sistema que manipulan directorios son *mkdir* para crear un directorio nuevo, *rmdir* para eliminar un directorio, *cd* para cambiar el directorio actual a otro y *pwd* para visualizar el nombre de la ruta absoluta del directorio actual (de trabajo).

El programa *ls* lista los nombres de los archivos del directorio actual. Cualquiera de las más de 20 opciones de *ls* puede hacer que se exhiban también las propiedades de los archivos, por ejemplo, la opción *-l* pide un listado largo, que muestra el nombre de cada archivo, su dueño, la protección, su tamaño, la fecha y hora en que se creó. El programa *cp* crea un archivo nuevo que es una copia de uno ya existente. El programa *mv* cambia de lugar un archivo dentro del árbol de directorios. En la mayor parte de los casos, este movimiento sólo requiere un cambio de nombre de archivo, pero si es necesario el archivo se copia en su nueva posición y la copia vieja se elimina. Los archivos se eliminan con el programa *rm*.

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

Para mostrar el contenido de un archivo en la terminal, un usuario puede ejecutar *cat*. El programa *cat* toma una lista de archivos y los concatena, copiando el resultado en la salida estándar, que normalmente es la terminal. Claro que en la terminal el archivo podría exhibirse con demasiada rapidez como para leerse. El programa *more* exhibe el archivo pantalla por pantalla y hace una pausa hasta que el usuario teclea un carácter para continuar con la siguiente pantalla. El programa *head* exhibe sólo las primeras líneas del archivo; *tail* muestra las últimas líneas.

Estos son algunos de los programas que usa Linux, además hay editores de texto (*ed, sed, emacs, nano, vi*, etc.), compiladores (de C, C++, Java, Python, etc.), formateadores de texto (LaTeX, *troff*, etc.), programas para ordenar (*sort*) y comparar (*cmp, diff*) archivos, buscar patrones (*grep, awk*) y muchas otras actividades.

La ejecución de una orden se efectúa con una o más llamadas al sistema operativo. Por lo regular, el Shell ejecuta una pausa cuando se le pide ejecutar una orden, queda en espera a que ésta se termine de ejecutar. Existe una sintaxis sencilla (un signo & al final de la línea de órdenes) para indicar que el Shell no debe esperar hasta que termine de ejecutarse la orden. Una orden que deja de ejecutarse de esta manera mientras el Shell sigue interpretando órdenes subsecuentes es una orden en segundo plano, o que se ejecuta en segundo plano. Los procesos para los cuales el Shell sí espera, se ejecutan en primer plano.

El Shell del sistema GNU/Linux ofrece un recurso llamado control de trabajos (y visualizarlos con los comandos *ps* o *top*) implementado especialmente en el núcleo. El control de trabajos permite transferir procesos entre el primer y segundo plano. Los procesos pueden detenerse y reiniciarse según diversas condiciones, como que un trabajo en segundo plano requiera entradas desde la terminal del usuario. Este esquema hace posible la mayor parte del control de procesos que proporcionan las interfaces de ventanas, pero no requiere Hardware especial. Cada ventana se trata como una terminal, y permite a múltiples procesos estar en primer plano (uno por ventana) en cualquier momento. Desde luego pueden haber procesos de segundo plano en cualquiera de las ventanas.

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

- Terminal

- Línea de comandos
- Shell
- Prompt

Esto es lo que debes saber al respecto:

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

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

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

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

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

**Algunos Comandos para Conocer el Sistema en el que Trabajamos**

Siempre es una buena práctica saber que componentes de Hardware se tienen en el equipo en que corremos nuestro GNU/Linux, esto nos puede ayudar a lidiar problemas de compatibilidad cuando se trata de instalar paquetes, controladores en nuestro sistema.

El primer comando a ejecutar es aquel que nos dé información del sistema en el que trabajamos, este es *uname* con la bandera *-a*, que nos dará información de la versión del Kernel, la versión y distribución del sistema operativo y en qué tipo de Hardware es que estamos corriendo nuestro GNU/Linux, para ello usamos:

```
$ uname -a
```

Para conocer las características de nuestro CPU, podemos usar varios comandos, entre ellos está *lscpu*, este nos dará la arquitectura del sistema, el número de CPUs, Cores, el modelo de la familia de CPU, Cache del CPU, Threds, entre otros. Para usarlo escribimos:

```
$ lscpu
```

si sólo queremos conocer el número de cores del equipo, usamos:

```
$ nproc
```

Podemos visualizar la memoria total, usada, libre y Swap (de intercambio) de nuestro equipo, usando:

```
$ free -g
```

Para visualizar información sobre los discos duros, unidades de estado sólido, en nuestro equipo, podemos usar *lsblk* que nos reportará dicha información, para ello escribimos:

```
$ lsblk -a
```

o bien podemos usar el comando *df* o en modo administrador a comando *fdisk*, escribiendo:

```
$ df -h
# fdisk -l
```

Si necesitamos conocer la información sobre los controladores USB y todos los dispositivos que están conectados a ellos, usamos:

```
$ lsusb
```

En caso de necesitar conocer los dispositivos PCI que pueden incluir puertos USB, tarjetas gráficas, adaptadores de red, entre otros, más los dispositivos que están conectados a ellos usamos:

```
$ lspci
```

Si lo que deseamos es un listado detallado del Hardware de nuestro equipo de cómputo, entonces podemos usar cualquiera de estos comandos (que previamente deberemos instalar):

```
# lshw
# dmidecode
# hwinfo
```

Así como podemos conocer las características de nuestro equipo de cómputo, podemos también conocer todos los comandos (-c), alias (-a), comandos internos (-b), palabras reservadas Bahs (-k) y funciones de Bash (-A function) disponibles para el usuario, para ello usamos:

```
$ compgen -c
$ compgen -a
$ compgen -b
$ compgen -k
$ compgen -A function
```

**Correr Múltiples Comandos en uno Solo** Supongamos que se tienen que ejecutar varios comandos uno tras otro -sin conocer si el comando anterior fue exitoso para ejecutar el nuevo-, para este propósito se usa el separador ";", de esta manera se pueden ejecutar una serie de comandos en una línea, ejemplo:

```
$ mkdir tmp ; ls ; cd tmp ; ls
```

en este ejemplo se crea en directorio, luego visualiza el contenido del directorio, para luego cambiar de directorio y finalmente visualiza el contenido del directorio recién creado.

Si necesitamos ejecutar múltiples comandos en uno solo, sólo si el comando anterior fue exitoso, se usa el separador "&&", ejemplo:

```
$ mkdir tmp && ls && cd tmp && ls
```

Notemos que si ejecutamos:

```
$ mkdir tmp & ls & cd tmp & ls
```

el resultado obtenido en nada se parece a lo que esperamos, ya que los comandos pasarán a ser ejecutados en segundo plano "&" y sin orden alguno, por lo que el resultado no será el deseado.

Ya vimos cómo usar un solo comando para ejecutar varios comandos. Pero, ¿qué debo hacer en caso que el primer comando no se ejecute correctamente?, ¿deseo ejecutar el comando siguiente?. Podemos usar || para que si el primer comando falla se ejecute el segundo, pero si no falla no ejecutará el segundo comando, por ejemplo:

```
$ cd miDirectorio || mkdir miDirectorio && cd miDirectorio
```

También podemos combinar los comandos && y || los cuales se comportarán como el operador ternario de C y C++ (condición? expresión_verdadera; expresión_falsa):

```
$ comando1 && comando2 || comando3
```

por ejemplo:

```
$ [-f archivo.txt ] && echo "Archivo existe" || echo "Archivo
no existe"
```

podemos combinar ;, && y || para correr múltiples comandos. Como por ejemplo:

```
$ sudo apt update && sudo apt upgrade && sudo apt clean
```

**Instalar Paquetes** Saber cómo instalar paquetes y programas en Linux y cualquiera de sus distribuciones es uno de los temas más complejos para los nuevos usuarios. El método más común para instalar un nuevo Software en Linux es a través de la Terminal. No obstante, los comandos varían según el tipo de gestor de paquetes que posee el sistema.

Uno de los apartados que debes tener en cuenta antes de instalar un paquete o programa en Linux, es conocer qué es un sistema de gestión de paquetes. En términos simples, hace referencia a un conjunto de formatos de archivos y herramientas empleadas para actualizar, instalar o desinstalar

algún Software en el sistema operativo[135]. En la actualidad, se divide en dos grandes sistemas de gestión de paquetes: Debian y Red Hat.

Distribuciones como Fedora, Mandriva, SuSE, CentOS, entre otros emplean el sistema de gestión de Red Hat, que se caracteriza por utilizar la extensión de archivos (*.rpm*). Por otro lado, las distribuciones como Ubuntu, Peppermint, Linux Mint, y muchos otros hacen uso del sistema *dpkg* de Debian, el cual se representa con la extensión (*.deb*) en sus archivos. Ambos sistemas de gestión trabajan de forma diferente para mantenerse activos en el dispositivo y, a su vez, gestionar las descargas.

Los procesos de instalación entre un sistema de gestión y otro difieren en diversos aspectos, especialmente en los comandos empleados. Por ello es fundamental repasar cuáles son los comandos utilizados para instalar un Software en Linux según el sistema de gestión de paquetes. En el caso de Debian GNU/Linux , puedes hacer uso del programa *apt-get* para instalar el programa deseado desde un repositorio[136]. Puedes escribir únicamente *apt* o utilizar *apt-get* en el comando.

Por otro lado, también utilizar el programa *dpkg* para la instalación de Software con extensión (*.deb*). Por otro lado, los sistemas basados en (*.rpm*) hacen uso de los comandos esenciales para las distribuciones de Linux Red Hat. Estos son *yum* y *dnf*. En el caso de *yum*, el usuario puede instalar o actualizar programas directamente desde el repositorio oficial de Linux, o bien desde un repositorio de terceros. Mientras que el comando *dnf* facilita la administración de programas.

Para Debian GNU/Linux usamos:

```
# apt install paquete
# dpkg -i paquete
```

En Ubuntu y derivados se usa:

[135] El usuario root en GNU/Linux es el usuario que tiene acceso administrativo al sistema, los usuarios normales no tienen este acceso por razones de seguridad. Sin embargo, en múltiples sistemas no incluye el usuario root. En su lugar, se da acceso administrativo a usuarios individuales, que pueden utilizar la aplicación *sudo* para realizar tareas administrativas. La primera cuenta de usuario que se creó en su sistema durante la instalación tendrá de forma predeterminada acceso a *sudo*.

[136] Un repositorio es un lugar en la red que siempre está actualizado desde donde nuestra distribución de GNU/Linux puede buscar y descargar fácilmente todo tipo de programas y herramientas para su instalación en nuestro equipo.

```
$ sudo apt install paquete
$ sudo dpkg -i paquete
```

Para openSUSE:

```
$ sudo rpm -Uvh paquete
$ su zypper intall paquete
```

En Fedora se usa:

```
$ sudo rpm -Uvh paquete
$ su -c 'yum install paquete'
# yum install paquete
```

Para Mandriva:

```
$ sudo rpm -Uvh paquete
$ su urpmi *.rpm
```

En Gentoo, FreeBSD se usa:

```
# emerge -vq paquete
```

Para Red Hat, Fedora o CentOS se usa:

```
$ sudo yum install paquete
$ sudo rpm -i paquete
$ sudo dfn install paquete
# yum install paquete
# dfn install paquete
```

En Archn Manjaro Linux se usa:

```
# pacman -s paquete
```

Para paquetes SNAP:

```
$ snap install paquete
```

**Instalar Paquetes en Debian y Derivados** Una de las funciones del usuario administrador es hacer la instalación, actualización y borrado de paquetes, el comando más usado actualmente es apt (Advanced Packaging Tool, herramienta avanzada de empaquetado que es una interfase del paquete *apt-get*), es un programa de gestión de paquetes *.deb* creado por el proyecto Debian, *apt* simplifica en gran medida la instalación, actualización y eliminación de programas en GNU/Linux. Existen también programas que proporcionan estos mismos servicios como: *apt-get*, *aptitude*, *tasksel* y *dpkg*.

Una vez siendo el usuario *root*, podemos solicitar la descarga de las actualizaciones disponibles, usando:

```
# apt update
```

para conocer los paquetes que se necesitan actualizar, usamos:

```
# apt list –upgradeable
```

para actualizar los paquetes instalados en el sistema[137], usamos:

```
# apt upgrade
```

algunas veces ciertos paquetes ya no se usarán al actualizar el sistema, para borrarlos usamos:

```
# apt autoremove
```

para buscar paquetes que podemos instalar, usamos:

```
# apt search nombre
```

para conocer las características de un paquete a instalar, usamos:

```
# apt show nombre
```

para instalar uno o más paquetes, usamos:

[137] Podemos hacer la actualización en un solo comando, usando:

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

```
# apt install paquete
```

para remover uno o más paquetes, usamos:

```
# apt purge paquete
```

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

```
# apt clean
```

podemos conocer todos los paquetes *.deb* instalados en el sistema, usando:

```
$ apt list –installed
```

o

```
$ dpkg -l
```

Existen otro tipo de paquetes para GNU/Linux que son multiplataforma como son: *Snap, Flatpak, Pyhon 3*. Para conocer si tenemos alguno de ellos instalados usamos:
para conocer todos los paquetes *Snap* instalados en el sistema, usamos:

```
$ snap list
```

para conocer todos los paquetes *Flatpak* instalados en el sistema, usamos:

```
$ flatpak list
```

para conocer todos los paquetes *python3* instalados en el sistema, usamos:

```
$ pip3 list
```

## 7.3 Trabajando en Línea de Comandos

Linux es un potente sistema operativo visual y de línea de comandos[138]. En esta última se tiene una potente herramienta, en ella se encuentran desde los comandos básicos hasta los más avanzados[139], algunos de ellos son:

Para manipulación de archivos y directorios[140]

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

Comandos generales

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

Administración y Permisos

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

Búsqueda

> *find, grep, locate*

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

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

[140] Existe la opción de gestionar ficheros comprimidos, gracias a los comandos *zgrep, zegrep, zless, zmore, zdiff, zcmp, zcat*. Como te puedes dar cuenta la función de cada uno de estos comandos será la misma que su homónimos sin «z» (*grep, egrep, less, more, diff, cmp, cat*) pero para ficheros comprimidos con *gzip* y extensión *.gz*. De forma análoga para archivos comprimidos usando *bz2*, están los comandos *bzgrep, bzegrep, bzless, bzmore, bzdiff, bzcmp, bzcat* y para archivos comprimidos usando *xz*, están los comandos *xzgrep, xzegrep, xzless, xzmore, xzdiff, xzcmp, xzcat*.

Respaldo

*tar, gzip, bzip2, zip, unp*

Varios

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

Para monitorear el desempeño

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

A continuación detallamos el uso de varios de estos comandos que se ejecutan en la línea de comandos de GNU/Linux o Terminal[141]. Hay que recalcar que cada comando tiene una gran variedad de opciones, pero la descripción completa de cada comando y opciones de este, se escapa de nuestros fines, por ello si se necesita conocer la referencia completa de dichos comandos hay varias maneras de obtenerla, entre otras haciendo uso de *man, help, info o whatis* aplicado al comando de nuestro interés, por ejemplo:

```
$ man ls
```

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

### Manipulación de Archivos y Directorios

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

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

el comando ls tiene varias opciones que permiten organizar la salida, lo que resulta particularmente útil cuando es muy grande. Por ejemplo, puedes usar *-a* para mostrar los archivos ocultos, *-i* para mostrar el inodo[142], *-l* para mostrar los usuarios, permisos, tamaño en Bytes y la fecha de los archivos; *-S* ordena por tamaño, *-R* recursivo, *-r* en orden inverso, *-t* ordenados por fecha de modificación, *-h* muestra el tamaño en unidades fáciles de leer -como KB, MB o GB-, *-F* les adiciona una / al final del nombre de los directorios, *-d* sólo muestra directorios, -X ordenar por extensión, *-n* muestra *UID* y *UIG* de los archivos y directorios. Así como para todos los comandos Linux, estas opciones pueden combinarse, terminando en algo como:

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

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

```
$ ls -lt
```

o los más recientes al final, usamos:

```
$ ls -ltr
```

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

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

que nos muestre aquellos archivos que coincidan con un patrón:

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

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

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

```
$ pwd
```

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

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

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

```
$ cd ..
```

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

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

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

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

o podemos pedir que cree múltiples directorios simultáneamente:

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

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

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

o podemos pedir que borre múltiples directorios vacíos simultáneamente:

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

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

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

en la sintaxis siempre se especifica primero el origen y luego el destino. Si indicamos un nombre de destino diferente, *cp* copiará el archivo o directorio con el nuevo nombre. El comando también cuenta con la opción *-r* que copia no sólo el directorio especificado sino todos sus directorios internos de forma recursiva. Suponiendo que deseamos hacer una copia del directorio */home/ejercicios* que a su vez tiene las carpetas *ejercicio1* y *ejercicio2* en su interior, en lugar de ejecutar un comando para cada carpeta, ejecutamos:

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

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

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

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

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

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

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

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

**rename** este comando[143] permite renombrar un grupo de archivos[144], por ejemplo deseamos renombrar todos los archivos con extensión .htm y reemplazarlos con la extensión .html, entonces usamos:

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

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

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

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

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

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

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

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

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

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

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

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

```
# apt install rename
```

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

```
# apt install mmv
```

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

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

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

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

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

**ln** permite crear enlaces a los archivos, tanto duros (Hard Links) como simbólicos *-s* (Soft Links). En pocas palabras, un enlace simbólico es como un acceso directo en Windows o un alias en Mac OS X mientras que un enlace duro es un nombre diferente para la misma información en disco. Todos los archivos que apuntan a un mismo enlace duro, comparten el *inodo* -este es un registro en el disco que contiene la información del archivo como su propietario, tamaño, fecha de creación, ubicación, entre otros-.

Para crear un enlace duro usamos:

```
$ ln archivo_origen nombre_enlace
```

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

```
$ ls -li archivo
```

para conocer todos los archivos que apuntan a un mismo enlace duro cuyo *inodo* conozcamos, usamos:

```
$ find / -inum <inodo>
```

y para conocer a todos los archivos que comparten inodo, usamos:

```
$ find . -type f -printf '%10i %p\n' | sort | uniq -w 11 -d -D |
less
```

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

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

Para crear un enlace simbólico, hacemos:

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

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

```
$ readlink nombre_enlace
```

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

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

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

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

y el comando umount para desmontar:

```
# umount nombre_enlace
```

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

```
$ df -i
```

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

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

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

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

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

```
$ unlik nombre_enlace
```

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

```
$ cat prueba.txt
```

podemos pedir que enumere cada línea del archivo mediante:

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

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

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

```
$ tac prueba.txt
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
$ wc archivo.txt
```

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

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

```
$ tail archivo.txt
```

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

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

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

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

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

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

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

```
$ head archivo.txt
```

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

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

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

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

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

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

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

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

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

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

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

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

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

$ head -C5 archivo.txt

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

$ more archivo.txt

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

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

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

$ less archivo.txt

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

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

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

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

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

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

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

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

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

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

**tr** se encarga de sustituir un delimitador por otro de uno o más ficheros (o entrada estándar), por ejemplo:
Se sustituye el delimitador ":" por un tabulador:

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

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

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

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

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

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

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

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

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

podemos borrar los caracteres indicados (minúsculas), ejemplo:

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

o cambiar las vocales de mayúsculas a minúsculas, ejemplo:

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

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

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

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

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

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

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

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

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

**nano** Es un pequeño editor de texto que esta disponible en casi todas las distribuciones actuales de GNU/Linux[145], funciona con un menú en la parte de inferior que se activa con la tecla *Ctrl* (por ejemplo para grabar se usa *Ctrl-o* y para salir *Ctrl-x*).

```
$ nano archivo.txt
```

### Comandos Generales

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

```
$ man clear
```

**help** proporciona ayuda de los comandos, con frecuencia puede sustituir al comando *man*, por ejemplo, para conocer la lista de comandos que soporta:

```
$ help
```

**info** proporciona ayuda de los comandos al igual que *man* y *help*, su uso es similar:

```
$ info mkdir
```

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

```
$ whatis ls
```

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

```
# apt install micro
```

y su uso es similar:

```
$ micro archivo.txt
```

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

```
$ clear
```

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

```
$ w
```

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

```
$ time ls
```

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

```
$ whoami
```

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

```
$ date
```

es posible formatear la salida del comando mediante:

```
$ date + "Weekday: %A Month: %B"[146]
$ date + "Week: %V Year: %y"
$ date -d last-week[147]
```

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

```
$ date - -help
```

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

**cal** muestra el calendario[148] del mes actual, con -y nos muestra el calendario del año completo, con -jy nos muestra el calendario con el número de día del año y con -A $n$ y -B $m$ nos muestra el calendario de $m$ meses antes de la fecha actual y $n$ meses después, ejemplo:

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

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

```
$ uptime
```

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

```
$ uname -a
```

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

```
$ df -hT
```

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

```
# apt install ncal
```

y se usa de forma similar a *cal*.

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

```
$ du -sh
```

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

```
$ free
```

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

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

```
$ bc -l
```

al escribir, por ejemplo:

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

mostrará el resultado con 100 dígitos de precisión.

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

```
$ history
```

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

```
$ history -c
```

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

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

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

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

### Permisos

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

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

donde *opciones* nos permite entre otras cosas, cambiar los permisos recursivamente para un directorio con *-R*, el *modo* son los permisos de lectura, escritura y ejecución representados en notación octal[149] y *archivo* es el nombre del directorio o archivo que queremos modificar[150].

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

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

[149]

| Octal | Binario | Modo Archivo |
|---|---|---|
| 0 | 000 | - - - |
| 1 | 001 | - - x |
| 2 | 010 | - w - |
| 3 | 011 | - w x |
| 4 | 100 | r - - |
| 5 | 101 | r w - |
| 6 | 110 | r w - |
| 7 | 111 | r w x |

[150] También están disponibles los comandos *getfacl* y *setfacl* pertenecientes a *Access Control List* (ACLs) que es un método más flexible para establecer permisos a usuarios y grupos sobre archivos y directorios.

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

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

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

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

$ chown -R pedro /home/ejercicios

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

$ chown -R pedro:pedro /home/ejercicios

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

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

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

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

$ chgrp -R pedro /home/ejercicios

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

$ su antonio

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

$ su

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

$ useradd [opciones] <nombre-usuario>

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

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

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

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

$ usermod [opciones] <nombre-usuario>

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

$ usermod -l violeta andrea

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

$ deluser -r violeta

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

$ passwd antonio

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

$ passwd -S antonio

**lsattr** permite listar los atributos[151] asignados a los ficheros y directorios, para ver los atributos del directorio actual usamos:

```
$ lsattr -a
```

o de forma recursiva

```
$ lsattr -Ra
```

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

```
# chattr +i archivo
```

o quitar el permiso:

```
# chattr -i archivo
```

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

```
# chattr +A archivo
```

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

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

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

```
# chattr +c archivo
```

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

```
# chattr +u archivo
```

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

```
# chattr +e archivo
```

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

```
# chattr +a archivo
```

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

```
$ id usuario
```

### Búsqueda

**find** permite buscar[152] dentro del sistema de archivos un directorio o archivos que concuerden con el patrón dado, por ejemplo:

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

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

y podemos usarlo mediante:

```
$ fdfind debian
```

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

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

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

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

Además, podemos hacer uso de expresiones regulares como:

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

**grep** permite buscar[153] en archivos un determinado patrón[154], mostrando la línea que lo contiene, por ejemplo:

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

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

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

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

si requerimos seguir sobre todos los enlaces simbólicos, usamos:

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

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

```
# apt install ripgrep
```

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

```
$ rg -tpy foobar
```

$ grep -R chmod *.txt

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

$ grep -l main *.java

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

$ grep -i error *.log

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

$ grep -v error *.log

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

$ grep 'error\|fatal' *.log

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

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

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

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

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

$ grep –color -E "error|fatal" *.log

regresa las líneas que no tengan la cadena indicada:

$ grep -v // Ejemplo.java

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

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

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

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

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

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

o líneas en blanco usando:

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

También podemos indicar que busque usando rango de caracteres mediante[155]:

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

**locate** permite buscar archivos o directorios cuyo nombre coincida con el patrón dado[156], por ejemplo:

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

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

```
# apt install mlocate
```

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

```
# updatedb
```

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

```
$ locate *dir2*
```

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

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

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

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

o solo visualizar el número de ocurrencias encontradas, ejemplo:

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

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

**Respaldo**

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

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

para restaurar usar:

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

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

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

o en un solo paso, usamos:

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

para restaurar se usar:

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

o alternativamente podemos usar:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
$ zipcloak archivo.zip
```

Podemos ver la información detallada del archivo comprimido, mediante:

```
$ zipdetails archivo.zip
```

y descomprimir mediante:

```
$ unzip archivo.zip
```

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

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

si al descompactar existe algún error, es posible recuperar parte de los archivos mediante:

```
$ zip -F archive.zip -O archive-fixed.zip
```

o usar *-FF*, después usar:

```
$ jar xvf archive-fixed.zip
```

otra alternativa es usar:

```
$ bsdtar xf archivo.zip
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.zip; do unzip "$z"; done
```

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

```
$ unp archivo.compactado
```

### Varios

**file** determina el tipo de un archivo y te imprime en pantalla el resultado. No hace falta que el archivo tenga una extensión para que *file* determine su tipo, pues la aplicación ejecuta una serie de pruebas sobre el mismo para tratar de clasificarlo.

```
$ file un_archivo_de_texto.txt
```

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

```
$ stat archivo
```

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

```
$ type ls
```

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

```
$ ps -ejH
```

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

```
$ ps -au
```

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

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

si queremos adicionar datos al archivo previamente creado usamos:

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

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

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

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

```
$ kill 3477
```

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

```
$ kill -1 2434
```

y lo reiniciamos usando:

```
$ kill -1 2434
```

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

```
$ killall firefox-esr
```

o podemos finalizar la sesión de un usuario, usando:

```
# killall -u antonio
```

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

```
$ pgrep firefox
```

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

```
$ pwdx 4534
```

o

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

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

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

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

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

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

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

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

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

```
$ sort archivo.txt
```

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

```
$ ls -al | sort -k 4 -n
```

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

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

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

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

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

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

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

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

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

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

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

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

**md5sum** genera la suma de verificación md5 (Compute and Check MD5 Message Digest) o Hash usada para verificar la integridad de los archivos, esta puede haber cambiado como resultado de una transferencia de archivos defectuosa, un error en disco o una interferencia maliciosa, ejemplo[157]:

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

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

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

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

```
$ watch free
```

de ser necesario, podemos quitarle el título, usando:

```
$ watch -t free
```

podemos indicar el intervalo de ejecución usando:

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

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

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

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

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

[157] Otras opciones son:

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

**Monitorear el Desempeño** Existen múltiples herramientas para ser usadas en línea de comandos y ambiente gráfico que permiten monitorear el desempeño y uso de una computadora con GNU/Linux, estas se pueden usar para administrar el sistema y las comunicaciones por red, estos comandos[158] están disponibles en todas las distribuciones de GNU/Linux y son normalmente usados para determinar problemas de desempeño en nuestro sistema de cómputo.

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

```
$ lscpu
```

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

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

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

```
$ free
```

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

```
top, vmstat, dstat
```

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

```
$ top
```

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

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

Otras variantes de este comando son:

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

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

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

```
$ whowatch
```

Otras opciones son:

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

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

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

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

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

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

$ vmstat 1

otras opciones son:

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

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

$ netstat

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

netstat -ltnp | grep -w ':80'

otras opciones son:

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

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

$ iotop

otras opciones son:

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

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

```
$ iostat
```

Otras opciones son:

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

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

```
$ lsof
```

Lista todos los procesos que tiene abierto el archivo:

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

Lista todos los archivos abiertos por el usuario:

```
$ lsof -u usuario
```

también se pueden indicar múltiples usuarios:

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

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

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

Lista todos los archivos abiertos en un directorio:

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

Lista todos los archivos abiertos por un identificador de proceso:

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

también podemos especificar múltiples identificadores de proceso:

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

Lista todos los archivos abiertos por un comando:

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

Busca archivos abiertos por un usuario, comando o proceso:

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

Lista las conexiones y puertos de red abiertos:

```
$ lsof -i
```

si usamos IPV4 o IPV6 podemos ver esos puertos abiertos:

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

podemos pedirle que nos muestre puertos tcp o udp:

```
$ lsof -i tcp
```

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

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

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

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

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

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

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

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

También podemos matar toda la actividad de un usuario particular:

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

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

```
$ lsusb
```

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

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

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

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

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

```
$ nmcli
```

## 7.4 Desde la Nube

Existen diferentes servicios Web[159] 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

[159] 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[160].

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

Algunas versiones listas para usar son:

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

### Terminales de Linux en la Web

- https://www.tutorialspoint.com/execute_bash_online.php
- http://www.webminal.org/
- https://bellard.org/jslinux/
- https://codeanywhere.com/

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

- https://copy.sh/v86/
- https://www.masswerk.at/jsuix/
- https://linuxcontainers.org/lxd/try-it/
- http://cb.vu/

**Editores BAHS en la Web**

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

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

https://termux.com

El paquete cuenta con una página *Wiki* en:

https://wiki.termux.com

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

**Usar Linux en Formato Live** Linux es uno de los sistemas operativos pioneros en ejecutar de forma autónoma o sin instalar en la computadora, existen diferentes distribuciones Live -descargables para formato CD, DVD, USB- de sistemas operativos y múltiples aplicaciones almacenados en un medio extraíble, que pueden ejecutarse directamente en una computadora, estos se descargan de la Web generalmente en formato ISO[161], una de las listas más completas de versiones Live esta en:

https://livecdlist.com

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

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

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

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

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

y usar mediante:

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

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

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

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

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

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

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

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

**Distribuciones de Sistemas Operativos** Existen diversos sitios Web que están enfocados a explorar detalladamente cada distribución actual o antigua, a un nivel técnico acompañado de grandes y útiles análisis técnicos sobre los mismos, lo que facilita el aprendizaje puntual sobre que distribución usar o empezar a usar sin tanta incertidumbre.

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

# 8 Apéndice C: Herramientas en Línea de Comandos

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

## 8.1 Prompt de la Línea de Comandos

Si acostumbras trabajar en la línea de comandos, muy posiblemente uses el Shell Bash (Bourne Again Shell, derivado del Bourne Shell de Unix), con un simple echo de la variable $SHELL puedes determinarlo: echo $SHELL. Si es el caso, entonces tu Prompt[162] debe de verse parecido a este:

antonio@miMaquina:~$

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

antonio@miMaquina:~$ cd /etc

antonio@miMaquina:etc$

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

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

antonio@miMaquina;~$ echo $PS1

\u@\h:\W\$

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

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

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

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

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

Conociendo lo anterior podemos ahora tener un nuevo Prompt:

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

obteniendo:

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

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

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

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

En cuanto a los códigos de color son los siguientes

- 0 Negro
- 1 Rojo
- 2 Verde

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

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

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

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

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

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

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

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

Aquí mostramos otra forma de códigos de color para Bash:

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

- 1;37 Blanco

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

\[ comienza un secuencia de caracteres no imprimibles

\] termina un secuencia de caracteres no imprimibles

Por ejemplo:

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

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

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

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

obteniendo para el usuario antonio, algo como:

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

y para el usuario root, algo como:

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

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

## 8.2 Historia de Comandos

Cada vez que se entra en la terminal y se trabaja, esta es guardada en la historia de comandos, podemos acceder al historial usando las flechas para moverse entre los comandos tecleados o las teclas *Ctrl-r* para buscar mediante una cadena al comando tecleado.

El comando que nos permite ver la historia de comandos teclados es *history*, podemos usarlo mediante:

```
$ history
```

esto nos permite ver los comandos tecleados (según la configuración de *history* en el sistema guarda los últimos mil comandos) y podemos reejecutar alguno usando ! y el número de comando en el historial, por ejemplo:

```
$ !20
```

en caso de que necesitemos ejecutar alguno de los últimos comandos ejecutados, podemos usar !-N, por ejemplo el penúltimo usamos:

```
$ history !-2
```

también podemos borrar un determinado comando del historial usando su número en history:

```
$ history -d 20
```

y podemos borrar la historia mediante:

```
$ history -c
```

pero esto no impide que se siga grabando si continuamos trabajando en la terminal, para borrarla y que no guarde nada de lo que hagamos, usamos:

```
$ set +o history
$ history -cw
```

o forzar el borrado del historial y salir de sesión, mediante:

```
$ cat /dev/null > ~/.bash_history && history -wc && exit
```

Por otro lado, podemos cambiar el formato de visualización para conocer la fecha y hora del comando tecleado, mediante:

$export HISTTIMEFORMAT="%d/%m/%y %T "

otra opción es:

$export HISTTIMEFORMAT="%h/%d  - %H:%M:%S "

y ahora podemos ver el historial en el formato solicitado usando:

$ history

si queremos que sea de forma permanente, debemos agregarlo en ~/.bashrc. Además podemos configurar en ~/.bashrc algunos otros aspectos como:

- cambiar el tamaño máximo del archivo de history, mediante:

  HISTFILESIZE=50000

- el número de comando a recordar, usando:

  HISTSIZE=10000

  si lo ponemos en cero, se deshabilita el guardado de la historial:

  HISTSIZE=0

- decirle a history que no guarde duplicados, usando:

  HISTCONTROL=ignoredups

- que no guarde los comandos que inician con espacios, usando:

  HISTCONTROL=ignorespace

- que no guarde duplicados e ignore los que inician con espacio:

  HISTCONTROL=ignoreboth

- cambiar el nombre del archivo en que se guarda la historia, que por omisión es ~/.bash_history, usando:

  HISTFILE=nombre

- además podemos ignorar ciertos patrones o comandos de la historia, para ello usamos:

  HISTIGNORE="history"

de este modo *history* no aparecerá, pero si algo como «*history* | *less*» y similares, así que podemos usar un comodín para evitarlo:

  HISTIGNORE="history*"

o podemos añadir más comando separándolos por «:», por ejemplo:

  HISTIGNORE="history*:echo*:ps*"

- con la configuración por defecto de *history*, usar varias sesiones de forma simultánea supone un problema. Por un lado, el histórico de comandos se guarda cuando finalizamos la sesión, es decir, al hacer un «*exit*» en Bash. Por defecto, al guardar el histórico de sesión, los contenidos del archivo se sobreescriben así que es posible que los comandos ejecutados en una de las sesiones no queden almacenados en el histórico. Para solucionar este problema, podemos decir que añada los comandos en el archivo *history* en lugar de sobreescribirlos:

  shopt -s histappend

- además, podemos especificar que los comandos se almacenen en el archivo de *history* al momento de ser ejecutados en lugar de al finalizar la sesión. De este modo, sesiones simultáneas podrán visualizar en el histórico sus respectivas ejecuciones de comandos[163]:

  PROMPT_COMMAND="history -a"

[163] Debemos verificar antes que la variable de entorno PROMPT_COMMAND no tiene ningún valor asignado, así no lo perderemos al establecer esta configuración. En caso de que lo tenga, podemos concatenar ambos valores separados por «;».

Si queremos ver los cambios se apliquen inmediatamente, usamos:

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

Si deseamos conocer cuáles son los comandos usados y cuantas veces los hemos usado, escribimos:

```
$ history | awk '{print $2}' | sort | uniq -c | sort -nr
```

## 8.3 Grabar y Reproducir Contenido de la Terminal

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

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

La sintaxis básica para iniciar a grabar es:

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

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

```
$ exit
```

Si deseamos seguir grabando sobre un archivo ya existente usamos:

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

Para reproducir lo grabado usamos:

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

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

```
# apt install asciinema
```

Para iniciar la grabación usamos:

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

para reproducirlo, usamos:

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

## 8.4 Alias a Comandos

Me gustaría crear un alias para el comando rm para tener un mensaje de confirmación antes de ejecutar este comando. Entonces podemos crear un alias como este:

```
$ alias rm='rm -i'
```

este es un alias temporal y dura hasta que cierras la terminal. Para guardar el alias de forma permanente, es necesario editar el archivo ~/.bashrc y agregar mi alias allí.

La estructura de alias es la siguiente:

```
alias name=value
alias name="command"
alias name="command arg1 arg2"
alias name="/path/to/Script"
alias name="/path/to/Script.pl arg1"
```

y podemos eliminarlos mediante:

```
unalias aliasname
```

Alias útiles

```
alias ls="ls –color=auto"
alias ll="ls -la"
alias l.="ls -d .* –color=auto"
alias la="ls -Ah"
alias l="ls -Cfh"
alias dir="dir –color=auto"
alias vdir="vdir -color=auto"
alias cd..="cd .."
alias ..="cd .."
alias ...="cd ../../../"
alias ....="cd ../../../../"
alias .....="cd ../../../../"
alias .4="cd ../../../../"
alias .5="cd ../../../../.."
alias grep="grep –color=auto"
alias egrep="egrep –color=auto"
alias fgrep="fgrep -color=auto"
alias diff="colordiff"
alias bc="bc -l"
alias mkdir="mkdir -pv"
alias df="df -H"
alias du="du -ch"
```

Confirmación de la acción

```
alias rm="rm -I –preserve-root"
alias mv="mv -i"
alias cp="cp -i"
alias ln="ln -i"
alias chown=2chown –preserve-root"
alias chmod="chmod –preserve-root"
alias chgrp="chgrp –preserve-root"
```

También podemos crear algunos comando nuevos, como este, que es una combinación de cd y ls, escribiendo en ~/.bashrc, lo siguiente:

```
cs() {
    cd "$@" && ls -a;
}
```

**Alias a directorios** también podemos hacer aliar a directorios como por ejemplo:

```
alias desk="cd ~/Desktop"
```

pero podemos hacer una función en ~/.bashrc. que guarde los directorios que más usamos, en el archivo ~/.directoriosGuardados, mediante:

```
# Permite guardar una trayectoria del árbol de directorios
guarda() {
    printf "$(pwd)\n" >> ~/.directoriosGuardados;
}
```

y en donde nos interese guardar la trayectoria usamos:

```
$ guarda
```

y con la función goto, nos permita movernos entre las diferentes trayectorias guardadas:

```
    # Permite cambiar entre las trayectorias guardadas por la
función: guarda
    function goto
    {
        local foo=$(sort ~/.directoriosGuardados | nl -w1)
        local REPLY
        echo -e "\n0\tSalir\n$foo\n"
        read -p "Introduzca el numero de directorio: "
        if [[ $REPLY =~^[0-"$(echo "$foo" | wc -l)"]$ ]]; then
            [[ $REPLY = "0" ]] && return
            cd "$(grep ^$REPLY <<<$foo | cut -f2)" && echo
"Ahora en: $(pwd)"
        else
            echo "No existe tal numero"
        fi
    }
```

de esta forma, podremos guardar múltiples trayectorias y cuando deseemos un cambio a una nueva trayectoria usamos:

```
$ goto
```

que nos mostrará las diferentes opciones guardadas y seleccionar usando el número que le corresponda o 0 para salir.

## 8.5 Ayuda de Comandos y Tipo de Archivos

Muchos comandos que podemos llegar a utilizar se pueden clasificar en categorías de acuerdo a su origen. Algunos están incorporados en el Shell, mientras otros provienen de un determinado paquete que hayamos instalado. También existe la posibilidad de que un comando sea en realidad un *alias* de otro comando con sus opciones.

***type*** permite identificar el comando o comandos pasados como parámetro indicando la trayectoria si es comando externo o si es comando interno al Shell. Uso del comando type:

$ type [*opciones*] comando o comandos

Algunas opciones del comando *type* son:

- -P muestra la trayectoria completa del comando
- -p retorna el nombre del archivo en disco al cual pertenece o nada si no hay archivo
- -a muestra la mayor información posible del comando
- -t retorna el tipo de comando, no la trayectoria

Por ejemplo para el comando creado en la función *cs* de la sección anterior

$ type cs

desplegará:

```
cs is a function
cs() {
    cd "$@" && ls -a;
}
```

**man** muestra la documentación completa de todos los comandos, su sintaxis es:

$ man [*opciones*] [sección] comando

para conocer todas las páginas disponibles del sistema, usamos:

$ man -k .

por ejemplo, para *clear*:

$ man clear

para una versión corta de la ayuda, usamos:

$ man -f clear

también podemos buscar ayuda de algo que tiene que ver con alguna palabra, por ejemplo algo con "sound", usamos:

$ man -k sound

además podemos usar grep en conjunción con man para una ayuda rápida sobre alguna opción del comando buscado, por ejemplo:

$ man ls | grep – –g

El comando *man* tiene diversas secciones donde se puede solicitar que busque la información de un comando para evitar ambigüedades, estas son:

1. *bin*: Binarios esenciales para el funcionamiento del sistema
2. *sys*: Llamadas al sistema
3. *lib*: Funciones de las bibliotecas
4. *dev*: Archivos de dispositivos
5. *etc*: Archivos de configuración
6. *games*: Juegos

7. *mis*: Miscelánea
8. *sbin*: Binarios esenciales para administración y mantenimiento del sistema
9. *boot*: Información del Kernel del sistema

por ejemplo para conocer *man* en la sección 7, usamos:

$ man 7 man

También es común el uso de *–help* en el comando que necesitamos conocer su uso, por ejemplo:

$ clear –help

**help** proporciona ayuda de los comandos, con frecuencia puede sustituir al comando *man*. Por ejemplo, para conocer la lista de comandos que soporta:

$ help

**info** proporciona ayuda de los comandos al igual que *man* y *help*, su uso es similar:

$ info mkdir

**pinfo** navegador de visualización de información de un comando del sistema, ejemplo:

$ pinfo mkdir

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

$ whatis ls

**apropos** busca en las páginas del manual para la palabra clave o expresión regular que le pasemos como parámetro, ejemplo:

$ apropos chmod

**whereis** localiza el archivo binario, sus fuentes y las páginas de manual del comando, ejemplo:

```
$ whereis info
```

**which** sirve para averiguar dónde se encuentra instalado uno o más comandos y para ello busca en los directorios del sistema, ejemplo:

```
$ which chmod ls
```

si se quiere saber todos los lugares en donde esta el comando usamos:

```
$ which -a sync
```

otra opción es:

```
$ command -v sync
```

**file** determina el tipo de archivo y muestra el resultado. No hace falta que el archivo tenga una extensión para que *file* determine su tipo, pues la aplicación ejecuta una serie de pruebas sobre el mismo para tratar de clasificarlo, ejemplo:

```
$ file un_archivo_de_texto.txt
```

Si se tiene un archivo que contiene una lista de trayectorias a diferentes archivos, podemos pedirle a file que nos de información de dichos archivos, usando:

```
$ file -f listaArchivos.txt
```

Si se tiene un archivo compactado, podemos pedirle a file que nos indique con que fue compactado, usando:

```
$ file -z archivoCompactado
```

## 8.6 Redireccionando la Entrada y Salida Estándar

Cuando se abre un archivo en Linux, a cada archivo se le asignará un número entero y esta información se almacena en el Kernel. De esta forma el Kernel sabe qué archivos se abren y qué proceso los ha abierto. El número entero asignado es lo que llamamos un descriptor de archivo (en breve FD).

Los procesos pueden abrir archivos a discreción, pero la mayor parte de los procesos esperan a que estén abiertos tres descriptores de archivos (nume-rados 0, 1 y 2) cuando inician. Estos descriptores se conocen como entrada estándar (FD 0 Standar Input "Keyboard"), salida estándar (FD 1 Standard Output "Display Terminal") y error estándar (FD 2 Standard Error "Display Terminal"). Es común que los tres estén abiertos en la terminal del usuario. Así, el programa puede leer lo que el usuario teclea leyendo la entrada estándar, y puede enviar salidas a la pantalla del usuario escribiendo en la salida estándar. El descriptor de archivo de error estándar también esta abierto para escritura, y se usa para los mensajes de error. Podemos ver estos descriptores usando:

```
$ ls -al /dev/std*
```

**Standard input** la entrada estándar, en inglés *standard input* (mejor conocido como *stdin*) es el mecanismo por el cual un usuario le indica a los programas la información que estos deben procesar. Por omisión, el teclado es la entrada estándar. La entrada estándar representa los datos que necesita una aplicación para funcionar, como por ejemplo un archivo de datos o información ingresada desde la terminal y es representado en la terminal como el tipo 0.

**Standard output** la salida estándar, en inglés *standard output* (mejor conocido como *stdout*) es el método por el cual el programa puede comunicarse con el usuario. Por omisión, la salida estándar es la pantalla donde se ejecutaron las instrucciones. La salida estándar es la vía que utilizan las aplicaciones para mostrarte información, allí podemos ver el progreso o simplemente los mensajes que la aplicación quiera darte en determinado momento y es representado en la terminal como el tipo 1.

**Standard error** por último existe un flujo conocido como error estándar, en inglés *standard error output* (mejor conocido como *stderr*) que es

utilizado por las instrucciones para desplegar mensajes de error que surjan durante el transcurso de su ejecución. Al igual que *stdout*, el error estándar será la pantalla donde se procesaron las instrucciones. El error estándar es la forma en que los programas te informan sobre los problemas que pueden encontrarse al momento de la ejecución y es representado en la terminal como el tipo 2.

**Operadores de redirección** a modo de resumen, indicamos las posibles formas de direccionamiento y los símbolos que se utilizan para lograrlo:

- $>$ Redirecciona stdout hacía un archivo, crea archivo si no existe, sobreescribe si existe.
- $>>$ Redirecciona stdout hacía un archivo, crea archivo si no existe, concatena si existe.
- $<$ Redirecciona stdin desde un archivo. El contenido de un archivo es la entrada del comando.
- 2$>$ Redirecciona stderr hacía un archivo, crea archivo si no existe, sobreescribe si existe.
- 2$>>$ Redirecciona stderr hacía un archivo, crea archivo si no existe, concatena si existe.
- $p>\&q$ Fusiona la salida del flujo $p$ con el flujo $q$ $(p, q \in [0, 1, 2])$.
- $p<\&q$ Fusiona la entrada del flujo $p$ con el flujo $q$ $(p, q \in [0, 1, 2])$.

**Otros redireccionamientos que no utilizan descritopres**

- $<<$ Conocido como HERE-DOCUMENT o HereDoc
- $<<<$ Conocido como HERE-STRING

Todos estos tipos son representados físicamente como archivos en el sistema, todo en GNU/Linux son archivos. Así, una redirección consiste en trasladar la información de un tipo a otro, por ejemplo de la salida estándar a la entrada estándar o del error estándar a la salida estándar. Esto lo logramos usando el símbolo >. Por ejemplo, para redireccionar la salida de un comando y volcarla a un archivo bastaría con ejecutar:

```
$ ls -la > archivo.txt
```

Sin embargo, cada vez que ejecutemos el comando, el contenido del archivo *archivo.txt* será reemplazado por la salida del comando ls. Si queremos agregar la salida del comando al archivo, en lugar de reemplazarla, entonces ejecutamos:

```
$ ls -la >> archivo.txt
```

Utilizar el comando touch para crear un archivo vació, es una práctica común que también puede realizarse con el operador de redireccionamiento >, mediante:

```
$ > documento1
```

Lo interesante es que, además de la salida estándar, también podemos redireccionar el error estándar y la entrada estándar. Si queremos forzar a que un programa nos imprima en pantalla los errores que consiga durante su ejecución podemos redireccionar el error estándar hacia la salida estándar. Eso lo logramos ejecutando:

```
$ programa 2>&1
```

¿Recuerdan que líneas arriba se comentó que GNU/Linux identifica a cada tipo con un número? Bueno, aquí es donde esos números cobran sentido. El tipo 2 es el error estándar y el tipo 1 es la salida estándar. En los ejemplos previos no tuvimos la necesidad de especificar el tipo 1 porque la terminal lo asume pero pudimos expresarlos explícitamente de la siguiente manera:

```
$ ls -la 1> archivo.txt
$ ls -la 1>> archivo.txt
```

Podemos, por ejemplo, contar las líneas que tiene un archivo redireccionando la entrada estándar de wc hacía un archivo de texto. Así:

```
$ wc < archivo.txt
```

También podemos redireccionar la entrada y salida en el mismo comando, por ejemplo:

```
$ sort < lista_desordenada.txt > lista_ordenada.txt
```

o podemos redireccionar la salida de errores y la salida al mismo archivo, por ejemplo:

```
$ comando > todo.txt 2>&1
```

la otra opción (menos común) es utilizar 1>&2, que indicaría redirecciona la salida del comando stdout hacía donde stderr apunte.

```
$ comando 2> errores.txt 1>&2
```

o redireccionar la salida y salida de error a distintos archivos, por ejemplo::

```
$ comando > salida.txt 2> errores.txt
```

Otra opción para guardar la salida de un comando es usar *logsave*, por ejemplo

```
$ logsave salida.txt ls -al
```

si necesitamos adicionar otras salidas usamos:

```
$ logsave -a salida.txt date
```

siendo posible redireccionar el error, usando:

```
$ logsave salida.txt ls -al /dev/null 2>&1
```

**Utilizando << y <<<** Es cuando un bloque de texto puede ser redireccionado a un comando o archivo de una manera interactiva. El Here Document funciona indicando un DELIMITADOR que no es más que una palabra o cadena cualquiera que cierra el bloque de texto que se desea redireccionar. Veámoslo con ejemplos:

```
$ wc << fin

> uno dos tres ⟵ de manera interactiva el usuario teclea el
bloque de texto, el delimitador es la palabra "fin""
> cuatro cinco
> seis
> fin
3 6 31
```

El comando wc recibe (stdin) un bloque de texto teclado por el usuario. Indicando el fin del bloque de texto con el delimitador fin. Cuando se recibe este, entonces wc ejecuta su función, al recibir por completo su entrada a examinar, en este caso 3 líneas, 6 palabras, 31 caracteres.

```
$ cat << TERMINA > documento1

uno dos
tres
cuatro cinco
TERMINA

$
$ cat documento1

uno dos
tres
cuatro cinco
```

En este caso el comando *cat* recibe el redireccionamiento de hereDoc con del delimitador TERMINA, cuando encuentra el delimitador deja de recibir entradas y direcciona lo ingresado al archivo "documento1". Si se quisiera agregar o concatenar a "documento1" se utilizará cat << TERMINA >> documento1

La diferencia entre << HERE DOCUMENT y <<< HERE STRING es que en HERE STRING no se pasa un delimitador, sino que se pasa una cadena que es interpretada por el comando al que se le redirecciona como un argumento(s). Este argumento puede ser una variable de Shell que puede expandirse. Es decir, HERE STRING se compone de una cadena o string de posibles argumentos o variables:

```
$ bc <<< 5*5
```

25

```
$ sed 's/hola/Hola/' <<< "hola mundo"
```

Hola mundo

**Redirección mediante pipe** las tuberías (pipe) unen la salida estándar de un comando con la entrada estándar de otro, es decir, la salida de un comando se emplea como entrada del siguiente. Para ello se emplea el símbolo pipe "|".

La utilización de tuberías evita la generación constante de archivos intermedios reduciendo el tiempo de procesamiento.

El siguiente ejemplo lista todos los procesos en ejecución en el sistema mediante el comando "ps -ef". Utilizando el pipe, el resultado viaja como entrada hacia el comando "grep" que se quedará solo con aquellas líneas donde aparezca la palabra "antonio". Este nuevo resultado del comando "grep" se envía como entrada del comando "sort" que se encargará de ordenar el resultado y mostrarlo por la pantalla de la terminal. Es decir que el último comando es el que hace uso de la salida estándar.

```
$ ps -ef | grep antonio| sort
```

En este otro ejemplo, el comando "cat" abre el archivo "nombres.dat" y su contenido lo envía, utilizando el pipe, como entrada del comando "cut" el cual recortará las líneas entre los caracteres 10 y 80. Este nuevo resultado, se envía al comando "sort" como entrada de información y haciendo uso de la opción "-r" lo ordenará en orden inverso al orden por defecto y luego muestra el resultado por la pantalla de la terminal.

```
$ cat nombres.dat | cut -c10-80 | sort -r
```

**Redirección mediante el comando tee** existe un comando que permite desviar una copia o bifurcación de la salida de un comando hacia un archivo sin alterar la entrada del siguiente en una tubería. Se trata del comando "tee"

Para entenderlo mejor, en el siguiente ejemplo, se almacena en en el archivo "conectados.dat" el resultado del comando "who" y, sin que este resultado sufra ninguna alteración por parte del comando "tee", se pasa mediante un segundo pipe hacia el comando "wc", quien se encargará de contar las líneas del resultado que produjo el comando "tee".

```
$ who | tee conectados.dat | wc -l
```

otros ejemplos son:

```
$ ps auxww | tee salida.log
$ ls -alh | tee archivo.txt | grep python
$ ls -alh | grep python | tee archivo.txt
$ ls -alh | grep python | tee archivo1.txt archivo2.txt archivo3.txt
$ ls -alh 2>&1 | tee archivo.txt
```

**Redirección hacia el dispositivo nulo** como se comentó, la salida estándar está asociada al descriptor de archivos 1 y la salida de errores está asociada al descriptor de archivos 2. Si bien para ambas salidas se utiliza el mismo dispositivo físico que es la pantalla de la terminal, ambos se gestionan de manera independiente. Son dos archivos diferentes.

Debes tener en claro que un comando puede generar una salida positiva, que sería aquello que se espera que haga, y eso va a parar a la salida estándar. Pero también puede generar un mensaje de error, y eso va a parar a la salida

de errores. En ambos casos, el efecto es verlo reflejado en la pantalla de la terminal, pero internamente están en dos archivos independientes.

Otra cosa a tener en claro es que cuando utilizas las redirecciones de ">" (símbolo mayor) o "|" (pipe), lo que viaja a través de ellas es el resultado de la ejecución de un comando que hubiera ido normalmente a la salida estándar. Los mensajes de errores no viajan a través de las redirecciones a menos que fusiones las salidas.

Pero algo interesante para contar, es que en ocasiones podría no interesarte que los mensajes de errores se muestren en la pantalla de la terminal por lo que es posible reasignar el descriptor de archivos 2 de la salida de errores estándar hacia un dispositivo que no exista físicamente. A este dispositivo se lo conoce como "dispositivo nulo" y su descriptor está en el directorio /dev.

En el siguiente ejemplo, con el comando "cat" intentaremos abrir el archivo "noexiste.txt" el cual no existe físicamente. En condiciones normales, verías en la pantalla de la terminal el mensaje que indica que "cat" no puede abrir el archivo "noexiste.txt". Este mensaje es un mensaje de error y por consiguiente fue a parar a la salida de errores estándar, en consecuencia lo ves en la pantalla de la terminal.

Pero, mediante la redirección "2>/dev/null", le indicamos al intérprete de comandos que está haciendo la ejecución del comando que todo lo que vaya a parar al descriptor de archivos 2, sea redireccionado hacia el dispositivo nulo "*/dev/nul*l". Y dado que este dispositivo no existe, entonces el mensaje de error se pierde y no se muestra ni almacena en ningún lado.

```
$ cat noexiste.txt 2>/dev/null
```

También podemos hacer algo muy común en la administración de sistemas, descartar el error estándar de un proceso. Para eso ejecutamos:

```
$ programa 2> /dev/null
```

O incluso descartar su salida estándar:

```
$ programa > /dev/null
```

En GNU/Linux, */dev/null* es un archivo especial al que se envía cualquier información que quiera ser descartada. Aunque al principio no lo parezca, el uso del dispositivo nulo es muy útil.

## 8.7 Metacarácter o Shell Globbing

los metacaracteres o Shell Globbing son caracteres que tienen un significado especial en la línea de comandos, para usar estos caracteres como caracteres ordinarios en la línea de comandos, debes marcarlos de manera especial para que el Shell no los interprete. Por ejemplo:

- La diagonal inversa (\) siempre sirve como carácter de escape para uno o más caracteres que le suceden, lo cual le da a esos caracteres un significado especial. Si un carácter es un metacaracter, el significado especial es el carácter mismo. Por ejemplo \\ usualmente significa una solo \ y \$ el signo de pesos. En estos casos, la diagonal inversa le quita su significado a los caracteres.

- Cuando un texto se encierra entre comillas (" "), la mayoría de los de los metacaracteres que estén en dicho texto son tratados como caracteres normales, excepto por $, que generalmente indica sustituciones a realizar.

- Otra forma de citar muy similar a la anterior, pero más fuerte es usar comillas sencillas (' ') ya que ni siquiera el carácter $ es interpretado

También existen otros metacaracteres que son comodines que el sistema permite usar para especificar los nombres de archivos que satisfacen el filtro especificado a la hora de buscar, eliminar o filtrar nombres de archivo, estos metacaracteres son: *, ?, [ ] y [^][164].

- ***** Se utiliza para reemplazar cero o más caracteres. Puede ser sustituido por cualquier cadena de caracteres, ejemplos:

  Muestra el contenido de las carpetas que contengan archivos de extensión *txt*:

  ```
  $ ls *.txt
  ```

  Lista todos los archivos que se llamen *archivo* sin importar su extensión:

  ```
  $ ls archivo.*
  ```

[164] Véase también el uso de las secuencias (véase 8.7).

Muestra todos los archivos con extensión *jpg* y que su nombre tenga al final "*chivo*":

$ ls *chivo.jpg

- **?** Sustituye un carácter cualquiera, ejemplos:

  Muestra todos los archivos empiecen con letras o números pero que luego de ellos tengan los valores "*b4ts.txt*":

  $ ls ?b4ts.txt

  Muestra todos los archivos que inicien con *ab*, siga cualquier letra, número o carácter y finalice con *ts.txt*:

  $ ls ab?ts.txt

  Muestra todos los archivos de tres letras que en medio tenga una letra *i*:

  $ ls ?i?

- [ ] Se usa para definir rangos o conjuntos de caracteres a localizar, para definir los rangos se debe usar el guión -, si son varios caracteres se separan por coma, ejemplos:

  Muestra todos los archivos que comiencen por *z* o *v* sin importar la extensión:

  $ ls [zv]*

  Muestra todos los archivos que comiencen por z o v y terminen con la extensión *.txt*:

  $ ls [zv]*.txt

  Lista todos los archivos de cualquier extensión que tengan los rangos establecidos entre los corchetes:

  $ ls archivo[12].*

Muestra la lista de todos los archivos que cumplan con el rango de "a-f" sin importar la extensión o el nombre:

$ ls [a-f]*

Muestra la lista de todos los archivos que inicien con cualquier cosa, pero que terminen con una letra mayúscula:

$ ls *[A-Z]

Muestra la lista de todos los archivos que inicien con una letra minúscula, tenga después una letra mayúscula, continué con cualquier carácter, después tenga una letra a, b, c-f, z y siga con cualquier cantidad de caracteres:

$ ls [a-z][A-Z]?[a,b,c-f,z]*

- [^] Este caso es contrario al anterior, este representa que se busque algo exceptuando lo que se encuentra entre los corchetes, también trabaja con rangos.

  Muestra los archivos que no empiecen con una letra minúscula pero que tengan extensión *.txt*:

  $ ls [^a-z]*.txt

- [**!**] Este caso igual al anterior, este representa que se busque algo exceptuando lo que se encuentra entre los corchetes, también trabaja con rangos.

  Muestra los archivos que no empiecen con una letra minúscula pero que tengan extensión *.txt*:

  $ ls [!a-z]*.txt

**Secuencias** Como parte del Bash podemos usar el generador de secuencias, como por ejemplo:

```
$ echo {1..10}
```

que muestra la secuencia de los números de 1 a 10, si ahora probamos:

```
$ echo {1..10..2}
```

visualizará los números $1, 3, 5, 7, 9$. Es decir, iniciará con el primer número en la secuencia y terminará en el segundo de la secuencia con incrementos del último número de la secuencia. También podemos hacerlo en orden inverso mediante:

```
$ echo {10..1..2}
```

que nos entregará los números $10, 8, 6, 4, 2$. También podemos usar relleno con ceros, por ejemplo:

```
$ echo {000..121..2}
```

el cual imprimirá los números de 0 a 121 con saltos de dos en dos, como: 000 002 004 006 ... 050 052 054 ... 116 118 120. Y si necesitamos números negativos, podemos usar algo como:

```
$ echo {3..-4}
```

también podemos usarlos para trabajar secuencias con letras, por ejemplo:

```
$ echo {a..z}
```

imprimirá las letras a hasta la z, o este otro ejemplo:

```
$ echo {n..z} {a..m}
```

imprimirá las letras $n$ a la $z$, y a continuación de $a$ a la $m$.

Con este generador de secuencias numéricas podemos aplicar a comando vistos en la sección anterior, como por ejemplo:

```
$ mkdir {2009..2019}_Facturas
```

que creará los directorios de 2009_Facturas, hasta 2019_Facturas. También lo podemos usar para borrar archivos, como por ejemplo:

```
$ rm cuadros_{043..61..3}
```

Este generador de secuencias podemos usarlo también en modo texto, por ejemplo:

```
$ touch archivo_{a..z}.txt
```

creará el archivo *archivo_a.txt* hasta *archivo_z.txt*. También es posible usar algo como {Z..a} pero generará caracteres no alfanuméricos. Otros usos son:

```
$ touch {blahg, splurg, mmmf}_file.txt
```

creará los archivos *blahg_file.txt, splurg_file.txt* y *mmmf_file.txt*.
Si queremos hacer:

```
$ cp -v file1.txt file1.txt.bak
```

lo podemos lograr, usando:

```
$ cp -v file1.txt{,.bak}
```

Otros ejemplos:

```
$ cp archivo{,.bak}
$ mv proyecto/{nuevo/viejo}/dir/otrodir
$ rm d{01..20}/archivo
$ touch {a,b,c}.{rs,cpp}
$ git add {main,x{1,2}}.rs
$ ls -l ~/{Downloads,Pictures}/*.{pdf,png}
$ mkdir -p ~/test/{etc/x1,lib,usr/{x2,x3},bin,tmp/{Y1,Y2,Y3/z},opt,var}
```

## 8.8 Permisos de Archivos y Directorios

GNU/Linux, al ser un sistema diseñado fundamentalmente para trabajo en red, la seguridad de la información que almacenemos en nuestros equipos (y no se diga en los servidores) es fundamental, ya que muchos usuarios tendrán o podrán tener acceso a parte de los recursos de Software (tanto aplicaciones como información) y Hardware que están gestionados en estos equipos de cómputo. ¿Ahora podemos ver por qué la necesidad de un sistema de permisos?

En GNU/Linux, los permisos o derechos que los usuarios pueden tener sobre determinados archivos contenidos en él se establecen en tres niveles claramente diferenciados. Estos tres niveles son los siguientes:

- Permisos del propietario.
- Permisos del grupo.
- Permisos del resto de usuarios (o también llamados "los otros").

Para tener claros estos conceptos, en los sistemas en red siempre existe la figura del administrador, superusuario o root. Este administrador es el encargado de crear y dar de baja a usuarios, así como también, de establecer los privilegios que cada uno de ellos tendrá en el sistema. Estos privilegios se establecen tanto para el directorio de trabajo (Home) de cada usuario como para los directorios y archivos a los que el administrador decida que el usuario pueda acceder.

**Permisos del propietario** el propietario (user ID, UID) es aquel usuario que genera o crea un archivo/carpeta dentro de su directorio de trabajo, o en algún otro directorio sobre el que tenga derechos. Cada usuario tiene la potestad de crear, por defecto, los archivos que quiera dentro de su directorio de trabajo. En principio, él y solamente él será el que tenga acceso a la información contenida en los archivos y directorios que hay en su directorio trabajo o Home -bueno, no es del todo cierto esto, ya que el usuario *root* siempre tiene acceso a todos los archivos y directorios del sistema-.

**Permisos del grupo** lo más normal es que cada usuario pertenezca a un grupo de trabajo (group ID, GID). De esta forma, cuando se gestiona un grupo, se gestionan todos los usuarios que pertenecen a éste. Es decir, es más

fácil integrar varios usuarios en un grupo al que se le conceden determinados privilegios en el sistema, que asignar los privilegios de forma independiente a cada usuario.

**Permisos del resto de usuarios** por último, también los privilegios de los archivos contenidos en cualquier directorio, pueden tenerlos otros usuarios que no pertenezcan al grupo de trabajo en el que está integrado el archivo en cuestión. Es decir, a los usuarios que no pertenecen al grupo de trabajo en el que está el archivo, pero que pertenecen a otros grupos de trabajo, se les denomina resto de usuarios del sistema.

**¿cómo puedo identificar todo esto?** sencillo, abre una terminal y realiza lo siguiente:

```
$ id
```

el comando *id* nos muestra nuestro UID, GID y los grupos a los que pertenecemos, podemos usar:

```
$ id antonio
```

para que nos muestra el UID, GID y los grupos a los que pertenece el usuario antonio.

Para conocer los permisos de nuestros archivos, podemos usar:

```
$ ls -l
```

entregará una salida como esta:

```
-rwxrwxr– 1 antonio ventas 9090 sep 9 14:10 presentacion
-rw-rw-r– 1 antonio antonio 2825990 sep 7 16:36 reporte1
drwxr-xr-x 2 antonio antonio 4096 ago 27 11:41 videos
```

Veamos por partes el listado, tomando como ejemplo la primera línea. La primera columna (-rwxrwxr–) es el tipo de archivo y sus permisos, la siguiente columna (1) es el número de enlaces al archivo, la tercera columna (antonio) representa al propietario del archivo, la cuarta columna (ventas) representa al grupo al que pertenece al archivo y las siguientes son el tamaño,

la fecha y hora de última modificación y por último el nombre del archivo o directorio.

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

- \- Archivo
- d Directorio
- b Archivo de bloques especiales (Archivos especiales de dispositivo)
- c Archivo de caracteres especiales (Dispositivo tty, impresora...)
- l Archivo de vínculo o enlace (soft/symbolic link)
- p Archivo especial de cauce (pipe o tubería)

Los siguientes 9 restantes, representan los permisos del archivo y deben verse en grupos de 3 y representan:

- \- Sin permiso
- r Permiso de lectura
- w Permiso de escritura
- x Permiso de ejecución

Los tres primeros representan los permisos para el propietario del archivo. Los tres siguientes son los permisos para el grupo del archivo y los tres últimos son los permisos para el resto del mundo u otros.

Las nueve posiciones de permisos son en realidad un bit que o está encendido (mostrado con su letra correspondiente) o está apagado (mostrado con un guión -), así que, por ejemplo, permisos como rwxrw-r–, indicaría que los permisos del propietario (rwx) puede leer, escribir y ejecutar el archivo, el grupo (o sea los usuarios que estén en mismo grupo del archivo) (rw-) podrá leer y escribir pero no ejecutar el archivo, y cualquier otro usuario del sistema (r–), solo podrá leer el archivo, ya que los otros dos bits de lectura y ejecución no se encuentran encendidos o activados.

Permisos para archivos:

- Lectura: permite, fundamentalmente, visualizar el contenido del archivo.
- Escritura: permite modificar el contenido del archivo.
- Ejecución: permite ejecutar el archivo como si de un programa ejecutable se tratase.

Permisos para directorios:

- Lectura: Permite saber qué archivos y directorios contiene el directorio que tiene este permiso.
- Escritura: permite crear archivos en el directorio, bien sean archivos ordinarios o nuevos directorios. Se pueden borrar directorios, copiar archivos en el directorio, mover, cambiar el nombre, etc.
- Ejecución: permite situarse sobre el directorio para poder examinar su contenido, copiar archivos de o hacia él. Si además se dispone de los permisos de escritura y lectura, se podrán realizar todas las operaciones posibles sobre archivos y directorios.

**Observación 2** *Si no se dispone del permiso de ejecución, no podremos acceder a dicho directorio (aunque utilicemos el comando "cd"), ya que esta acción será denegada. También permite delimitar el uso de un directorio como parte de una ruta (como cuando pasamos la ruta de un archivo que se encuentra en dicho directorio como referencia. Supongamos que queremos copiar el archivo "X.ogg" el cual se encuentra en la carpeta "/home/antonio/Z" -para lo cual la carpeta "Z" no tiene permiso de ejecución-, haríamos lo siguiente:*

*$ cp /home/antonio/Z/X.ogg /home/antonio/Y/*

*obteniendo con esto un mensaje de error diciéndonos que no tenemos los permisos suficientes para acceder al archivo. Si el permiso de ejecución de un directorio está desactivado, se podrá ver su contenido (si se cuenta con permiso de lectura), pero no se podrá acceder a ninguno de los objetos contenidos en él, pues para ello este directorio es parte del camino necesario para resolver la ubicación de sus objetos.*

**Permisos en formato numérico octal** La combinación de valores de cada grupo de los usuarios forma un número octal, el bit x es 20 es decir 1, el bit w es 21 es decir 2, el bit r es 22 es decir 4, tenemos entonces:

- r = 4
- w = 2
- x = 1

La combinación de bits encendidos o apagados en cada grupo da ocho posibles combinaciones de valores, es decir la suma de los bits encendidos:

| | | |
|---|---|---|
| $---$ | $= 0$ | no se tiene ningún permiso |
| $--x$ | $= 1$ | solo permiso de ejecución |
| $-w-$ | $= 2$ | solo permiso de escritura |
| $-wx$ | $= 3$ | permisos de escritura y ejecución |
| $r--$ | $= 4$ | solo permiso de lectura |
| $r-x$ | $= 5$ | permisos de lectura y ejecución |
| $rw-$ | $= 6$ | permisos de lectura y escritura |
| $rwx$ | $= 7$ | todos los permisos establecidos |

Cuando se combinan los permisos del usuario, grupo y otros, se obtienen un número de tres cifras que conforman los permisos del archivo o del directorio. Esto es más fácil visualizarlo con algunos ejemplos:

**Estableciendo Permisos con el Comando chmod** habiendo entendido lo anterior, es ahora fácil cambiar los permisos de cualquier archivo o directorio, usando el comando chmod (change mode), cuya sintaxis es la siguiente:

chmod [opciones] permisos archivo[s], algunos ejemplos:

```
$ chmod 755 reporte1
$ chmod 511 respaldo.sh
$ chmod 700 julio*
$ chmod 644 *
```

Los ejemplos anterior establecen los permisos correspondientes que el usuario propietario desea establecer, el tercer ejemplo (chmod 700 julio*) cambiará los permisos a todos los archivos que empiecen con julio (julio01, julio02, julio_respaldo, etc.) debido al carácter '*' que es parte de las expresiones regulares que el Shell acepta, e indica lo que sea. El último ejemplo por lo tanto cambiará los permisos a los archivos dentro del directorio actual.

Una opción común cuando se desea cambiar todo un árbol de directorios, es decir, varios directorios anidados y sus archivos correspondientes, es usar la opción -R, de recursividad:

```
$ chmod -R 755 respaldos/*
```

Esto cambiará los permisos a 755 (rwxr-xr-x) del directorio respaldos y de todos los subdirectorios y archivos que estén contenidos dentro de este.

**Estableciendo Permisos en Modo Simbólico** Otra manera popular de establecer los permisos de un archivo o directorio es a través de identificadores del bit (r,w, o x) de los permisos, como ya se vio anteriormente, pero ahora identificando además lo siguiente:

- al usuario con la letra u
- al grupo con la letra g
- a otros usuarios con la letra o
- y cuando nos referimos a todos (usuario, grupo, otros) con la letra a (all, todos en inglés)
- el signo + para establecer el permiso
- el signo - para eliminar o quitar el permiso

La sintaxis es muy simple:

```
$ chmod augo[+|-]rwx[,...] archivo[s]
```

así por ejemplo, si queremos que otros tengan permiso de escritura sería chmod o+w archivo, todos los usuarios con permisos de ejecución chmod a+x archivo.

Si quieres prevenirte de modificar un archivo importante, simplemente quita el permiso de escritura en tu «archivo» con el comando chmod

```
$ chmod -w tuArchivo
```

si quieres hacer un Script ejecutable, escribe

```
$ chmod +x tuScript
```

si quieres remover o agregar todos los atributos a la vez

```
$ chmod -rwx archivo
$ chmod +rwx archivo
```

también puedes usar el signo = (igual) para establecer los permisos en una combinación exacta, este comando remueve los permisos de escritura y ejecución dejando solo el de lectura

```
$ chmod =r archivo
```

En este modo de establecer permisos, solo hay que tomar en cuenta que partiendo de los permisos ya establecidos se agregan o se quitan a los ya existentes

**Cambiando Propietario y Grupo** Volviendo a mostrar el listado al inicio de esta sección:

```
$> ls -l
```

```
-rwxrwxr– 1 antonio ventas 9090 sep 9 14:10 presentacion
-rw-rw-r– 1 antonio antonio 2825990 sep 7 16:36 reporte1
drwxr-xr-x 2 antonio antonio 4096 ago 27 11:41 videos
```

Vemos en la tercera y cuarta columna al usuario propietario del archivo y al grupo al que pertenece, es posible cambiar estos valores a través de los comandos chown (Change Owner, cambiar propietario) y chgrp (Change Group, cambiar grupo). La sintaxis es muy sencilla:

```
$ chown usuario archivo[s]
```

y

```
$ chgrp grupo archivo[s].
```

por ejemplo:

```
# ls -l presentacion
```

```
-rwxrwxr– 1 antonio ventas 9090 sep 9 14:10 presentacion
```

```
# chown juan presentacion
# ls -l presentacion
```

```
-rwxrwxr– 1 juan ventas 9090 sep 9 14:10 presentacion
```

```
# chgrp gerentes presentacion
# ls -l presentacion
```

```
-rwxrwxr– 1 juan gerentes 9090 sep 9 14:10 presentacion
```

Solo el usuario root puede cambiar usuarios y grupos a su voluntad sobre cualquier usuario, queda claro que habiendo ingresado al sistema como usuario normal, solo podrá hacer cambios de grupos, y eso solo a los que pertenezca.

Una manera rápida para el usuario *root* de cambiar usuario y grupo al mismo tiempo, es con el mismo comando chown de la siguiente manera:

```
# chown juan.gerentes presentacion
```

o en vez de punto, con : puntos

```
# chown juan:gerentes presentacion
```

así, cambiará el usuario.grupo en una sola instrucción.

Además, al igual que con chmod, también es posible utilizar la opción -R para recursividad.

**Bits SUID, SGID y de Persistencia (Sticky Bit)** Aún hay otro tipo de permisos que hay que considerar. Se trata del Bit de permisos *SUID* (Set User ID), el Bit de permisos *SGID* (Set Group ID) y el Bit de permisos de persistencia (Sticky Bit). Para entender los dos primeros el *SUID* y el *SGID* veamos los permisos para un comando de uso común a todos los usuarios, que es el comando passwd, que como se sabe sirve para cambiar la contraseña del usuario, y puede ser invocado por cualquier usuario para cambiar su propia contraseña, si vemos sus permisos observaremos un nuevo tipo de permiso:

```
# ls -l /usr/bin/passwd
```

```
-r-s–x–x 1 root root 21944 feb 12 2020 /usr/bin/passwd
```

**SUID** en vez de la 'x' en el grupo del usuario encontramos ahora una 's' (suid). passwd es un comando propiedad de root, pero sin embargo debe de poder ser ejecutado por otros usuarios, no solo por root. Es aquí donde interviene el Bit SUID, donde al activarlo obliga al archivo ejecutable binario a ejecutarse como si lo hubiera lanzado el usuario propietario y no realmente quien lo lanzó o ejecutó. Es decir, es poder invocar un comando propiedad de otro usuario (generalmente de root) como si uno fuera el propietario.

**SGID** el Bit SGID funciona exactamente igual que el anterior solo que aplica al grupo del archivo. Es decir si el usuario pertenece al grupo 'ventas' y existe un binario llamado 'reporte' que su grupo es 'ventas' y tiene el Bit SGID activado, entonces el usuario que pertenezca al grupo 'ventas' podrá ejecutarlo. También se muestra como una 's' en vez del Bit 'x' en los permisos del grupo.

**STICKY BIT (Bit de persistencia)** este Bit se aplica para directorios como en el caso de /tmp y se indica con una 't':

```
# ls -ld /tmp

drwxrwxrwt 24 root root 4096 feb 25 18:14 /tmp
```

Puede apreciarse la 't' en vez de la 'x' en los permisos de otros. Lo que hace el Bit de persistencia en directorios compartidos por varios usuarios, es que el sólo el propietario del archivo pueda eliminarlo del directorio. Es decir cualquier otro usuario va a poder leer el contenido de un archivo o ejecutarlo si fuera un binario, pero sólo el propietario original podrá eliminarlo o modificarlo. Si no se tuviera el Sticky Bit activado, entonces en estas carpetas públicas, cualquiera podría eliminar o modificar los archivos de cualquier otro usuario.

**Estableciendo Permisos Especiales** Para cambiar este tipo de Bit se utiliza el mismo comando *chmod* pero agregando un número octal (1 al 7) extra al principio de los permisos, ejemplo:

```
# ls -l /usr/prog

-r-x–x–x 24 root root 4096 sep 25 18:14 prog
```

```
# chmod 4511 /usr/prog
# ls -l /usr/prog
```

-r-s–x–x 24 root root 4096 sep 25 18:14 prog

Nótese que el valor extra es el '4' y los demás permisos se dejan como se quieran los permisos para el archivo. Es decir, los permisos originales en este ejemplo eran 511 (r-x–x–x), y al cambiarlos a 4511, se cambió el Bit SUID reemplazando el Bit 'x' del usuario por 's'.

Los posibles valores serían los siguientes:

| | | |
|---|---|---|
| - - - - - - - - - | = 0 | Predeterminado, sin permisos especiales. No se requiere indicar. |
| - - - - - - - - t | = 1 | Bit de persistencia, Sticky Bit |
| - - - - - s - - - | = 2 | Bit Sgid de grupo |
| - - - - - s - - t | = 3 | Bit Sgid y Sticky |
| - - s - - - - - - | = 4 | Bit Suid |
| - - s - - - - - t | = 5 | Bit Suid y Sticky |
| - - s - - s - - - | = 6 | Bit Suid y Sgid |
| - - s - - s - - t | = 7 | Bit Suid, Sgid y Sticky |

**Observación 3** *Algo sumamente delicado y que se tiene que tomar muy en cuenta es lo que decidas establecer con permisos de Bit SUID y SGID, ya que recuerda que al establecerlos de esta manera, cualquier usuario podrá ejecutarlos como si fueran el propietario original de ese programa. Y esto puede tener consecuencias de seguridad severas en tu sistema. Siempre considera y reconsidera si conviene que un usuario normal ejecute aplicaciones propias de root a través del cambio de Bits SUID o SGID. Mejores alternativas pueden ser los comandos sudo y su.*

**Permisos preestablecidos con umask** El comando umask establece un permiso de archivo y directorio para todos los archivos y directorios que se creen, para conocer su valor podemos teclear:

```
$ umask
```

y nos regresa su valor, por ejemplo: 0022

En formato simbólico usamos la opción -S:

```
$ umask -S
```

y nos regresa su valor, por ejemplo: u=rwx,g=rx,o=rx

lo anterior indica que un directorio se creará con los permisos 755 y los archivos con los permisos 644. Esto se logra restando de 777 el valor de umask (777-022) y (666-022) respectivamente. El primer valor de umask corresponde para valores de Sticky Bit, *GUID* o *SUID*, que por omisión es 0.

Para establecer el valor de la máscara, simplemente se usa el mismo comando *umask* seguido del valor de máscara que se desee:

```
$ umask 0002
```

para dejarlo fijo en la sesión, entonces conviene agregarlo a *.bash_rc* o *.bash_profile* de nuestro directorio de inicio. Algunos ejemplos son:

| *umask* | *Permisos* | *en Archivos* | *Permisos* | *en Directorios* |
|---|---|---|---|---|
| 000 | 666 | (rw-rw-rw-) | 777 | (rwxrwxrwx) |
| 002 | 664 | (rw-rw-r- -) | 775 | (rwxrwxr-x) |
| 022 | 644 | (rw-r- -r- -) | 755 | (rwxr-xr-x) |
| 027 | 640 | (rw-r- - - - -) | 750 | (rwxr-x- - -) |
| 077 | 600 | (rw- - - - - - -) | 700 | (rwx- - - - - -) |
| 277 | 400 | (r- - - - - - - - ) | 500 | (r-x- - - - - -) |

**Archivos y Fechas** En sistemas similares a UNIX como GNU/Linux, todo se considera un archivo, y toda la información sobre un archivo (metadatos o atributos del archivo como la hora de creación, la última modificación, etc) excepto en contenido del archivo real se almacena en un inodo y Linux identifica todos y cada uno de los archivos por su número de inodo que no sea el nombre de archivo legible por humanos.

Además, el programa *stat* de GNU/Linux es una utilidad útil para mostrar archivos o el estado del sistema de archivos. Muestra información como el número de inodo, la hora de creación del archivo, la última modificación de datos, el último acceso, el último cambio de estado y mucho más. Podemos combinar *stat* y *debugfs* para obtener toda la información de un archivo, mediante:

```
$ stat archivo
```

no regresa el inodo del archivo en cuestión (supongamos 12), y usando *df* encontramos la partición en la que reside el archivo (supongamos /dev/sda1), combinando esto entonces:

```
# debugfs -R 'stat <12>' /dev/sda2
```

nos mostrará toda la información de dicho archivo.

## 8.9 Procesos en Primer y Segundo Plano

La ejecución de una orden se efectúa con una o más llamadas al sistema operativo. Por lo regular, el Shell ejecuta una pausa cuando se le pide ejecutar una orden, queda en espera a que ésta termine se de ejecutar. Existe una sintaxis sencilla (un signo & al final de la línea de órdenes) para indicar que el Shell no debe esperar hasta que termine de ejecutarse la orden. Una orden que deja de ejecutándose de esta manera mientras el Shell sigue interpretando órdenes subsecuentes es una orden en segundo plano (Background), o que se ejecuta en segundo plano. Los procesos para los cuales el Shell sí espera se ejecutan en primer plano (Foreground).

El Shell del sistema GNU/Linux ofrece un recurso llamado control de trabajos (y visualizarlos con los comandos *ps* o *top*) implementado especialmente en el núcleo. El control de trabajos permite transferir procesos entre el primer y segundo plano. Los procesos pueden detenerse y reiniciarse según diversas condiciones, como que un trabajo en segundo plano requiera entradas desde la terminal del usuario. Este esquema hace posible la mayor parte del control de procesos que proporcionan las interfaces de ventanas, pero no requiere Hardware especial. Cada ventana se trata como una terminal, y permite a múltiples procesos estar en primer plano (uno por ventana) en cualquier momento. Desde luego pueden haber procesos de segundo plano en cualquiera de las ventanas.

En GNU/Linux podemos ejecutar procesos en primer plano (Foreground) o bien en segundo plano (Background). Un programa en primer plano lanzado desde un terminal monopoliza dicho terminal, por lo que en principio, no podremos ejecutar ningún otro programa a la vez (veremos más adelante como se puede hacer). Por el contrario un programa en segundo plano una vez iniciado, deja de monopolizar el terminal desde el que se lanzó, y este

nos vuelve a mostrar el Prompt[165].

El comando *bg* que es la abreviatura de la palabra background por tanto recibe como parámetro un número de proceso en estado *Stopped* y hace que ésta reanude su ejecución pero en segundo plano es decir, sin bloquear el teclado para el Shell.

De igual forma que hemos pasado un proceso a ejecutar a un segundo plano, el Shell nos permite recuperar la ejecución en primer plano de un proceso que estaba en segundo plano. Para ello se utiliza el comando *fg* (abreviatura de la palabra Foreground). Como parámetro recibe al igual que *bg* el número de un proceso prefijado con %.

¿Como podemos lanzar otro programa desde un terminal con otro programa en ejecución en Foreground?

Pulsamos *CTRL-z* con lo que pausamos el programa en ejecución y Foreground, ojo lo pausamos con lo cual dejará de funcionar, y ya podremos lanzar otro programa, por ejemplo *ls* o podemos hacer una prueba lanzamos *gimp* y comprobamos que podemos operar con él, luego pulsamos *CTRL-z* y vemos cómo dejamos de poder trabajar con *gimp*).

Ahora queremos volver a poner en funcionamiento a *gimp* y así poder volver a utilizar gimp o si queremos devolverlo a Foreground escribiremos *fg*, o si queremos devolverlo a background escribiremos *bg* (esta sería la opción más lógica).

En el caso de que tengamos más de un programa detenido deberemos indicarle tanto a *fg* como a *bg* el identificador de tarea sobre el que actuarán, este ID podemos obtenerlo con el comando *jobs*.

Tenemos a la mano muchas herramientas en el Shell que nos permiten trabajar con los procesos, saber sus PID, sus estamos, sus nombre, sus usuarios, la cantidad de recursos que consumen. Podemos mencionar algunas:

- ps
- pstree

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

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

- pmap
- top
- kill
- jobs
- disown

Para interrumpir un proceso que se está ejecutando en segundo plano podemos hacerlo de dos formas. Mediante el comando *fg* la pasamos a ejecutar al primer plano y luego con *Ctrl-c*. Pero también podemos conseguir el mismo efecto sin utilizar el comando *fg* sino simplemente con el comando *kill* seguido del número de proceso prefijado con el símbolo % que nos da el comando *jobs* y podemos usar *disown* para que el comando continúe activo después de cerrar la terminal.

**Nohup y &** Cuando se cierra una sesión en GNU/Linux, el sistema manda una señal para matar todos los procesos que esté lanzando nuestro usuario (Señal *SIGHUP*). Por lo tanto, aunque les pongamos el & al final, morirán. Para evitar que esto suceda, hay que ejecutar el Script o comando poniendo el comando *nohup* delante y lo que va a hacer es ignorar la señal de *SIGHUP* y redirigir toda la información correspondiente a un fichero llamado nohup.out

```
$ nohup ./Script.sh &
```

esta opción es muy útil cuando no hemos de interactuar con el proceso, puesto que al quedar el fichero de log, se puede ver el resultado con toda la información al respecto.

Los Scripts son indispensable en Linux y en algunas ocasiones puede ser necesario ejecutarlo en segundo plano, ya sea porque tarde mucho en finalizar o porque el programa tiene que ejecutarse de forma indefinida y al mismo tiempo se quieren analizar sus entradas/salidas en tiempo real, o cuando, en el caso de conexiones remotas, por el motivo que sea, se pueda producir una desconexión.

Una buena práctica sería redireccionar stdin, stdout y stderr. Básicamente, por dos razones: rastrear la salida de nuestro Script en caso de producirse algún error, y evitar problemas al terminar nuestra sesión *ssh*, si es que la ejecutamos en un servidor remoto, por ejemplo:

```
$ nohup ./Script.sh > foo.out 2> foo.err < /dev/null &
```

esta opción es muy útil cuando necesitamos ejecutar un proceso largo y no nos interesa saber nada de él hasta que finalice. Podemos desconectarnos irnos, regresar al día siguiente y analizar el resultado del proceso, como alternativa podemos también usar:

```
$ nohup ./Script.sh > salida.txt 2>&1 &
```

**Limitar el Tiempo de Ejecución** Algunas veces es necesario limitar el tiempo máximo de ejecución de un comando o Script, para ello disponemos de *timeout*, su formato es el siguiente:

```
$ timeout [opciones] duración comando [argumentos]
```

la duración la podemos especificar en segundos (s), minutos (m), horas (h) o días (d), ejemplos:

```
$ timeout 8s ping 127.0.0.1
$ timeout 2h comando argumentos
```

Podemos darle un período de gracia antes de forzar su terminación, ejemplo:

```
$ timeout -k=5 2m comando argumentos
$ timeout -k=5 -s SIGKILL 2m comando argumentos
```

en este caso se envía una señal *KILL* si el comando todavía se está ejecutando tanto tiempo después de que se envió la señal inicial.

También podemos pedir que corra en segundo plano el comando, mediante:

```
$ timeout –foreground 30s ssh -t usuario@servidor top
```

**Limitar uso de CPU a un programa** Algunas veces es necesario limitar el uso de la CPU (expresado en porcentaje de uso) a un proceso o programa para ello usamos a *cpulimit*, para instalarlo usamos:

```
# apt install cpulimit
```

Si queremos limitar el uso del CPU de un programa en ejecución usamos:

```
# cpulimit -l 50 -e apache2
```

Si conocemos el PID de nuestro proceso, podemos usarlo para limitar el uso del CPU (digamos al 30%), usando:

```
$ cpulimit -l 30 -p 4546 &
```

o bien, podemos lanzar el proceso limitando el uso del CPU, mediante:

```
$ cpulimit -l 30 ./miprograma &
```

## 8.10 GNU Parallel

¿Alguna vez tuviste la extraña sensación de que tu computadora no es tan rápida como debería ser? Solía sentirme así, y luego encontré GNU Parallel. GNU Parallel es una herramienta de Shell que permite la ejecución de trabajos en paralelo.

Un trabajo puede ser un único comando o entrada de un archivo que contiene elementos tales como una lista de comandos, una lista de archivos, una lista de Hosts, una lista de usuarios, una lista de URL o una lista de tablas. GNU Parallel también puede tomar información de un comando con pipes (|).

Si te encuentras en una situación en la que necesitas ejecutar varios comandos al mismo tiempo (por ejemplo, en los servidores Linux de un centro de datos), ¿qué haces? GNU Parallel es una alternativa interesante que debes conocer.

Cuando se ejecutan comandos en Linux, ya sea uno a la vez en la línea de comandos o desde un Script de Bash, los comandos se ejecutan en secuencia. El primer comando se ejecuta, seguido por el segundo, seguido por el tercero. Es cierto, el tiempo entre los comandos es tan minúsculo, que el ojo humano

no se daría cuenta. Pero para algunos casos, puede que no sea el medio más eficiente para ejecutar comandos.

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

```
# apt install parallel
```

Por ejemplo, para cambiar el formato de los .jpg a .png podríamos hacer lo siguiente:

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

o

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

Con eso conseguimos que los comandos *find* o *ls* busquen todos los ficheros .jpg en el directorio actual con cualquier nombre y le pase todos los resultados a *parallel* mediante la pipe, que luego transmitirá de uno a uno al comando convert para convertirlos a png. Es decir, va a realizar convert nombre1.jpg nombre1.png, convert nombre2.jpg nombre2.png, y así sucesivamente...

O directamente usamos *parallel* para ello:

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

ahora, utilizaremos zenity para presentar una barra de progreso de manera gráfica. ¡Porque, que sea terminal no significa que no sea visual!

```
$ find -maxdepth 1 -name ‘*.jpg’ | parallel –bar convert {}
convertidas/{/.}.png 2> >(zenity –progress –auto-kill)
```

# 9 Trabajando en Línea de Comandos

La mayoría de los usuarios de ordenadores de hoy sólo están familiarizados con la interfaz gráfica de usuario o GUI (del inglés Graphical User Interface) y los vendedores y los expertos les han enseñado que la interfaz de línea de comandos o CLI (del inglés Command Line Interface) es una cosa espantosa del pasado. Es una pena, porque una buena interfaz de línea de comandos es una maravillosa y expresiva forma de comunicarse con el ordenador, muy parecida a lo que el lenguaje escrito es para los seres humanos. Se ha dicho que "las interfaces gráficas de usuario hacen fáciles las tareas fáciles, mientras que las interfaces de línea de comandos hacen posibles las tareas difíciles" y eso es muy cierto aún hoy.

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

**Emuladores de Terminal** cuando utilizamos una interfaz gráfica de usuario, necesitamos otro programa llamado emulador de terminal para interactuar con el Shell. Si buscamos en nuestros menús de escritorio, probablemente encontraremos uno. KDE usa Konsole y GNOME usa Gnome-terminal, aunque es probable que se llame simplemente "Terminal" en nuestro menú. Hay muchos otros emuladores de terminal disponibles para Linux, pero todos hacen básicamente lo mismo; nos dan acceso al Shell.

**Trabajar con la línea de comandos** no es una tarea tan desalentadora como muchos pudieran pensar. No se requieren conocimientos especiales para usar la línea de comandos, pues es un programa como otro cualquiera. La mayoría de las acciones realizadas en Linux pueden llevarse a cabo usando la línea de comandos. Aunque existen herramientas gráficas para la mayoría de programas, a veces esto no es suficiente. Entonces es cuando la línea de comandos cobra su utilidad.

La terminal es llamada a menudo la línea de comandos, el "Prompt", o el "Shell". Antes, éste era el único método por el cual el usuario interactuaba con lel ordenador; sin embargo, muchos usuarios de Linux encuentran más rápido el uso de la terminal que un método gráfico e incluso hoy día tiene algunas ventajas. Aquí aprenderemos cómo usar la terminal.

**El Intérprete de Órdenes de Consola** o Shell es un programa informático, cuya función consiste en interpretar órdenes, y un lenguaje de consola. El más conocido es Bash[166]. Es una Shell de Unix compatible con POSIX y el intérprete de comandos por defecto en la mayoría de las distribuciones GNU/Linux, además de Mac OS. También se ha llevado a otros sistemas como Windows y Android.

Algunas alternativas a Bash son:

- SH (Bourne Shell) fue desarrollado por Stephen Bourne y es un Shell que se encuentra dentro de la jerarquía de archivos de Unix en /bin/sh.
- TCSH/CSH (C Shell) ha sido desarrollado para proporcionar una interfaz de usuario. Gracias a este Shell podremos ejecutar comandos y ejecutar múltiples programas desde la consola del sistema.
- KSH (Korn Shell) su desarrollo principal fue la interpretación de órdenes a través de línea de comandos.
- Fish (Friendly Interactive Shell) no está basado en sh y posee una sintaxis de línea de comandos única que está diseñada para ser más amigable con los usuarios que están iniciando en el mundo Shell.
- ZSH (Z Shell) ha sido un Shell diseñado en 1990 influenciado por Bash, Ksh y Tcsh. Zsh es un Shell popular gracias a sus características de desempeño y funcionalidades a la hora de ejecutar comandos.
- TSCH (TS Shell) es una versión mejorada de CSH , la cual ofrece múltiples usos ya que es un lenguaje de comandos que puede ser usado tanto como un Shell de inicio de sesión interactivo como un procesador de comandos Shell.

[166] Su nombre es un acrónimo de Bourne-again Shell ("Shell Bourne otra vez"), haciendo un juego de palabras (Born-again significa "nacido de nuevo") sobre la Bourne Shell (sh), que fue uno de los primeros intérpretes importantes de Unix.

- DaSH (Debian Almquist Shell) es un intérprete de comandos de Unix compatible con el estándar sh de POSIX, mucho más ligero y rápido que otros como Bash pero con menos características.
- Busybox, integra más de doscientas utilidades en un solo ejecutable, usa Almquist Shell y es ideal para dispositivos Linux integrados.
- DSH (Distributed Shell) es un pequeño programa que nos permitirá ejecutar un comando en varias máquinas de una sola vez.

Podemos conocer que interprete de comandos usamos, para ello escribimos:

```
$ ps -p $$
```

Si deseamos cambiar de intérprete a por ejemplo *zsh*, podemos instalarlo usando:

```
# apt install zsh
```

y debemos indicarle al sistema que queremos utilizar *zsh* como Shell por defecto usando:

```
chsh -s $(which zsh)
```

La sintaxis de órdenes de Bash es un superconjunto de instrucciones basadas en la sintaxis del intérprete Bourne. La especificación definitiva de la sintaxis de órdenes de Bash, puede encontrarse en el Bash Reference Manual distribuido por el proyecto GNU. Esta sección destaca algunas de sus características únicas.

## 9.1 Buscar Archivos

La búsqueda de archivos de GNU/Linux se puede hacer con por lo menos dos comandos diferentes para buscar archivos: find (que traduce encontrar) y locate (que traduce ubicar).

**Usando el comando find** El comando más común utilizado para encontrar y filtrar archivos es a través del comando *find*. El diseño básico de este comando es el siguiente:

find <directorio inicio> <opciones> <termino búsqueda>

El argumento <*directorio inicio*> es el punto de origen de donde deseas iniciar la búsqueda. Esto es útil si tienes una idea aproximada de dónde podría estar ubicado el archivo deseado, ya que hace más específica la búsqueda. La mayoría de las veces, sin embargo, querrás buscar el archivo en todo el sistema. Puedes hacer esto reemplazando tu ruta con una barra " / ", que es el símbolo del directorio raíz. A veces es posible que quieras iniciar la búsqueda desde el directorio de trabajo actual, es decir, el directorio donde está abierto el terminal. Esto se puede hacer con el argumento punto " . ". Para averiguar tu directorio actual, usa el comando pwd. Finalmente, para comenzar la búsqueda de archivos desde tu carpeta de inicio, usa el símbolo " ~".

El segundo argumento es el filtro que deseas usar para buscar tu archivo. Este podría ser el nombre, tipo, fecha de creación o de modificación del archivo, etc. El tercer argumento es una continuación del segundo, donde se especificará el término de búsqueda relevante.

**Búsqueda por Nombre** por supuesto, el método más común y obvio para buscar un archivo es usar su nombre. Para ejecutar una consulta de búsqueda simple usando el nombre del archivo, usa el comando find de la siguiente manera:

```
$ find . -name "archivo"
```

En el comando anterior, usamos la opción *-name* y buscamos un archivo llamado *archivo*. Ten presente que comenzamos la búsqueda en nuestro directorio actual.

Una cosa importante para recordar cuando se utiliza el estándar es el argumento *-name*, que busca términos distinguiendo entre mayúsculas y minúsculas en GNU/Linux. Por lo tanto, si conoces el nombre del archivo, pero no estás seguro de su las mayúsculas y minúsculas, usa el comando find de esta manera:

```
$ find . -iname "archivo"
```

o bien para buscar todos los archivos con extensión .zip, .tar, .tar.gz, entre otros, usamos:

```
$ find . -type f \( -name "*.zip" -o -name "*.tar*" \)
```

Otra forma de utilizar la opción name es buscar todos los archivos sin una palabra clave en el nombre. En GNU/Linux, puedes hacer esto de dos maneras. El primer método implica el uso de la palabra clave *-not* de la siguiente manera:

```
$ find . -not -name "archivo"
```

También podemos usar *" ! "*[167], aunque debe estar precedido por el identificador de escape para que GNU/Linux sepa que esta es parte del comando de búsqueda y no una independiente.

```
$ find . \! -name "archivo"
```

También puede buscar varios archivos con un formato común como *.txt*, lo cual podría ser útil en algunos casos:

```
$ find . -name "*.txt"
```

esto listará todos los archivos de texto comenzando con la carpeta actual.

Algunas veces es necesario acotar la profundidad de la búsqueda y limitarla, digamos al actual directorio, por ejemplo:

```
$ find . -maxdepth 1 archivo
```

Si deseas buscar un determinado archivo por nombre y eliminarlo, usamos el argumento *-delete* después del nombre del archivo:

```
find . -name "archivo" -delete
```

otra forma es usar:

```
$ find . -name "archivo" | xargs rm -f
```

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

también podemos tomar la salida del comando y hacer algo con ella, como:

```
$ find . -name 'log*' | xargs wc
```

Si al realizar una búsqueda se generan errores (por no tener acceso a archivos o directorios por ejemplo), estos pueden ser redireccionados, mediante:

```
$ find / -type f 2> /dev/null
```

**Búsqueda por Tipo** para la mayoría de los usuarios, basta con saber cómo encontrar archivos por sus nombres. Sin embargo, siempre es útil conocer todas las herramientas que se ofrecen para aprovechar GNU/Linux al máximo.

Aquí es donde entra en juego el argumento -type. GNU/Linux ofrece a los usuarios las siguientes opciones para buscar archivos por tipo:

- f - Archivo normal
- d - Directorio o carpeta
- l - Enlace simbólico
- c - Dispositivos de caracteres
- b - Dispositivos de bloque

Un ejemplo simple del uso del tipo de archivo para la búsqueda se puede ver a continuación:

```
$ find / -type d
```

Esto mostrará una lista de todos los directorios presentes en el sistema de archivos, al haber comenzado la búsqueda desde nuestro directorio raíz con el símbolo de barra inclinada. Para encontrar las ligas simbólicas que hay desde nuestra actual trayectoria, usamos:

```
$ find . -type l
```

También puedes concatenar las opciones -type y -name para hacer tus búsquedas aún más específicas:

```
$ find / -type f -name "archivo"
```

esto buscará archivos llamados *archivo*, excluyendo directorios o enlaces.
Otro ejemplo es buscar archivos, directorios y enlaces simbólicos, mediante:

```
$ find /tmp -type f,d,l
```

o

```
$ find /tmp \( -type f -o -type d -o -type l \)
```

Para busca todas las ligas simbólicas y saber a donde apuntan, usamos:

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

y si queremos conocer las ligas rotas, usamos:

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

Algunas veces es necesario acotar la profundidad de la búsqueda y limitarla, digamos al actual directorio, por ejemplo:

```
$ find . -maxdepth 1 -type f -newer archivo
```

**Búsqueda por Fecha** si quieres buscar archivos en función de su fecha de acceso y las registros de fecha de modificación, GNU/Linux te ofrece las herramientas para hacerlo. Hay 3 registros de tiempo de los cuales Linux realiza seguimiento en los archivos:

- Tiempo de acceso (-atime) - Fecha más reciente en que el archivo fue leído o escrito.
- Tiempo de modificación (-mtime) - Fecha más reciente en que se modificó el archivo.
- Hora de cambio (-ctime) - Fecha más reciente en que se actualizaron los metadatos del archivo.

Esta opción debe usarse con un número. Este número especifica el número de días desde que se accedió, modificó o cambió el archivo. La forma más sencilla de buscar archivos por tiempo es:

```
$ find / -atime 1
```

Esto encontrará todos los archivos a los que se accedió hace un día desde el momento actual. Esto significa que se listarán todos los archivos que fueron leídos y/o escritos desde el día anterior.

Podemos hacer que nuestras consultas sean más precisas con los signos más (+) y menos (-) precediendo al número de días. Por ejemplo:

```
$ find / -mtime +2
```

Esto listará todos los archivos que tienen un tiempo de modificación de más de dos días.

Para buscar todos los archivos cuyos metadatos de *inodo* se actualizaron hace menos de un día, ejecuta lo siguiente:

```
$find / -ctime -1
```

Finalmente, hay algunos argumentos adicionales además de estos 3 que también están relacionados con las búsquedas por fecha. El argumento -*<time-deScriptor>min* busca cuántos minutos han pasado. Se puede usar así:

```
$find / -mmin -1
```

Esto buscará archivos que se modificaron hace menos de un minuto. Además, el argumento *-newer* se puede usar para comparar la antigüedad de dos archivos y encontrar el más reciente.

Para buscar archivos **.log* de más de 90 días, podemos usar:

```
$ find / -name "*.log" -type f -mtime +90 -ls
```

y los podemos borrar usando:

```
$ find /app/logs/ -name "*.log" -type f -mtime +90 -delete
```

Podemos buscar archivos no vacíos de más de 4 días en el directorio actual sin extensión *.gz* y comprimirlos:

```
$ find . -mtime +4 ! -name '*.gz' ! -empty -execdir gzip -v9
{} +
```

**Búsqueda por tamaño** al igual que GNU/Linux te brinda la opción de buscar archivos basados en registros de tiempo, también te permite hacer lo mismo con los tamaños. La sintaxis básica para buscar archivos por tamaño es:

$ find <directorio inicio > -size <tamaño> <unidad tamaño>

Puedes especificar las siguientes unidades de tamaño:

- c - Bytes
- k - kiloBytes
- M - MegaBytes
- G - GigaBytes
- b - Trozos de 512 Bytes

Un ejemplo simple de cómo usar el comando find para los tamaños de archivo es el siguiente:

$ find / -size 10M

Esto buscará en tu sistema archivos que tengan exactamente 10 MegaBytes de tamaño. Al igual que cuando buscaste en función del tiempo, puedes filtrar aún más tus búsquedas con los signos más y menos:

$ find / -size +5G

El comando anterior listará todos los archivos de tu disco que tengan más de 5 GigaBytes de tamaño. De manera similar, puede lograr esto con el signo menos para indicar "menor que" en tus consultas.

Además podemos pedir que la búsqueda nos entregue los datos de los archivos encontrados usando el comando *ls -l*, por ejemplo:

$ find . -size +1000c -exec ls -l {} \;

o si queremos buscar y borrar, usamos:

$ find / -type f -mtime +30 -sise +1M -exec rm -rf {} \;

o bien:

```
$ find / type f -mtime +30 -size +1M -delete
```

Por último, podemos buscar ficheros vacíos, mediante:

```
$ find . -type f -empty
$ find . -size 0c
```

o directorios vacíos, usamos:

```
$ find . -type d -empty
```

Algunos ejemplos útiles son:

```
$ find . -printf '%s %p\n' | sort -nr | head -10
$ find . -type d -printf '%s %p\n' | sort -nr | head -10
$ find . -type f -printf '%s %p\n' | sort -nr | head -10
$ find .  -iname "*.mp4" -printf '%s %p\n' | sort -nr | head
-10
$ find . -type d -empty -print0 | xargs -0 rmdir
$ find . -type f -empty -print0 | xargs -0 rm
```

**Búsqueda por Propiedad** La jerarquía de privilegios de Linux también se puede utilizar al buscar archivos. GNU/Linux te da la capacidad de especificar tus búsquedas según la propiedad del archivo, así como los permisos otorgados a diferentes usuarios.

Para buscar archivos de un determinado propietario, se debe ejecutar el siguiente comando:

```
$ find / -user usr
```

Esto devolverá una lista de todos los archivos que posee el usuario llamado *usr*. Similar a los nombres de usuario, también podemos buscar archivos a través de nombres de grupo:

```
$ find / -group otro
```

Esto buscará aquellos archivos que son propiedad del grupo llamado *otro*.

**Búsqueda por permisos** Los usuarios que desean buscar archivos basados en los permisos de los archivos[168] pueden hacerlo usando la opción *-perm* del comando *find*. Por ejemplo:

```
$ find / -perm 644
```

En GNU/Linux, 644 corresponde a permisos de lectura y escritura. Lo que significa que este comando buscará todos los archivos que solo tienen permisos de lectura y escritura. Otras opciones son:

```
$ find . -perm 1551
$ find . -perm /u=r
$ find . -perm /a=x
```

o si deseamos buscar los que no tienen esos permisos, para ello usamos *-not*, por ejemplo:

```
$ find . -not -perm 777
```

También podemos buscar ficheros/directorios que tengan activos los bits *SUID*, *SGID* y *Sticky Bit* que también podrían representar problemas de seguridad. Pero para que busque cualquiera sin importar el resto de los permisos, debemos usar / para obviar los permisos estándar, para ello usamos respectivamente:

```
$ find . -perm /4000
$ find . -perm /2000
$ find . -perm /1000
```

[168]

| Octal | Binario | Modo Archivo |
|---|---|---|
| 0 | 000 | - - - |
| 1 | 001 | - - x |
| 2 | 010 | - w - |
| 3 | 011 | - w x |
| 4 | 100 | r - - |
| 5 | 101 | r w - |
| 6 | 110 | r w - |
| 7 | 111 | r w x |

El siguiente ejemplo busca archivos que sean de lectura para todos (-perm -444 o -perm -a+r) y que tenga al menos un conjunto de bits de escritura ( -perm /222 o -perm /a+w) pero no son ejecutables para cualquiera (\! -perm /111 o \! -perm /a+x):

```
$ find . -perm -444 -perm /222 \! -perm /111
```

o

```
$ find . -perm -a+r -perm /a+w \! -perm /a+x
```

Por último, podemos cambiar todos los permisos digamos 744 encontrados a 644, usando:

```
$ find . -perm -744 -exec chmod -R 644 {} \;
```

**Búsqueda por inodo** el comando *ln* permite crear enlaces a los archivos, tanto duros (Hard Links) como simbólicos *-s* (Soft Links). Todos los archivos que apuntan a un mismo enlace duro, comparten el *inodo* -este es un registro en el disco que contiene la información del archivo como su propietario, tamaño, fecha de creación y ubicación-.

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

```
$ ls -li archivo
```

y para conocer todos los archivos que apuntan a un mismo enlace duro cuyo *inodo* conozcamos, usamos:.

```
$ find / -inum <inodo>
```

y para conocer a todos los archivos que comparten inodo, usamos:

```
$ find . -type f -printf '%10i %p\n' | sort | uniq -w 11 -d -D |
less
```

**Búsqueda usando expresiones regulares** esto nos ofrece una potencia considerable. Por ejemplo si deseamos buscar un nombre como:

archivo01_01.txt, archivo02_03.txt, etc.

podemos utilizar:

```
$ find . -regex './archivo0[1-2]_0[1-3].*'
```

**Otras opciones útiles** Además de todos estos métodos de búsqueda de archivos, hay otras opciones útiles que uno debería recordar. Por ejemplo, para buscar archivos y carpetas vacíos en el sistema, usamos:

```
$ find / -empty
```

del mismo modo, para buscar todos los ejecutables guardados en tu disco, utiliza la opción *-exec:*

```
$ find / -exec
```

para buscar archivos leíbles, usamos:

```
$ find / -read
```

Hay veces que necesitamos quitar espacios en el nombre de los archivo, por ejemplo:

```
$ find . -name "* *mp3" -exec rename 's/\ /-/g' {} \;
```

remplazará en todos los archivos **.mp3* los espacios en el nombre.
Si necesitamos buscar en todos nuestros archivos *.java* aquellos que contengan por ejemplo la palabra *interface*, podemos usar:

```
$ find . -name "*.java" - type f -exec grep -l interface {} \;
```

Si necesitamos buscar archivos y filtrarlos por nombre de archivo, además de canalizar el resultado a algún otro comando (por ejemplo *wc* para contar palabras y usamos *tr* para sustituir el carácter nulo por nuevas líneas), el comando queda como:

```
$ find . -name ".md" | grep "notas/2020" | \
tr '\n' '\0' | xargs -0 wc -w
```

El siguiente ejemplo copia el contenido de la actual trayectoria a /dest-dir, pero omite archivos y directorios llamados .snapshot (y cualquier cosa dentro de ellos). También omite archivos o directorios cuyo nombre termina en ~, pero no sus contenidos.

```
    $ find . -name .snapshot -prune -o \( \! -name '*~' -print0 \)
\
    | cpio -pmd0 /dest-dir
```

la construcción -prune -o \( \! -name '*~' -print0 \)| es bastante común. La idea aquí es que la expresión antes de -prune coincide con las cosas que se deben podar, sin embargo, el -prune acción en si misma se devuelve verdadero, por lo que el siguiente -o asegura que el lado derecho es evaluado solo para aquellos directorios que no fueron podados. La expresión en el lado derecho de -o está entre paréntesis solo por claridad. Hace hincapié en que la acción -print0 tiene lugar solo para las cosas que no tenían -prune aplicado a ellos. Otro ejemplo es:

```
$ find repo/ \( -exec test -d '{}'/.svn \; -or \
-exec test -d {}/.git \; -or -exec test -d {}/CVS \; \) \
-print -prune
```

en este ejemplo -prune evita el descenso innecesario a los directorios que ya han sido descubiertos.

Los siguientes ejemplos permiten copiar los archivos (por ejemplo *.mp3) pero preservan la estructura de directorios que existía en su lugar de origen, veamos:

```
$ find . -name '*.mp3' -exec cp –parents \{\} ~/OtroDirectorio \;
$ find . -name '*.mp3' | cpio -pdm ~/OtrDirectorio
$ find ~/miDirectorio -name '*.mp3' -exec cp –parents \{\} ~/OtrDirectorio \;
$ rsync -a -m –include '*/' –include '*.mp3' –exclude '*' ~/miDirectorio/ ~/OtrDirectorio
$ rsync -a –prune-empty-dirs –include '*/' –include '*.mp3' –exclude '*' ~/miDirectorio/ ~/OtrDirectorio
```

el primer y segundo ejemplo busca a partir de la actual trayectoria y los copia en *~/OtroDirectorio*, en los demás ejemplos, inicia la búsqueda en *~/midirectorio/* y los copia en *~/OtroDirectorio*.

Para eliminar múltiples archivos del tipo que buscamos, tenemos dos opciones:

```
$ find . -name "archivo" -exec rm -rf {} \;
```

o

```
$ find . -type f -name "archivo" -exec rm -rf {} \;
```

la diferencia entre ambos comandos indicados arriba es que el primero localiza y borra archivos y directorios, y el segundo sólo archivos:

**Usando el comando Locate** En este punto, podrías cuestionar la necesidad de tener una alternativa para el comando *find*. Después de todo, hace todo lo necesario para buscar archivos. Sin embargo, locate es una alternativa útil, ya que es más rápido que find para buscar en todo el sistema.

Por defecto, GNU/Linux no viene con el comando locate preinstalado. Para obtener el paquete, ejecuta los siguientes comandos en tu terminal:

```
# apt install mlocate
```

Ahora que has adquirido locate, puedes usarlo para buscar archivos así:

```
$ locate archivo
```

Puedes usar el argumento *-b* para una búsqueda más específica. Solo buscará el "nombre base" del archivo, enumerando efectivamente solo los archivos que tienen el término de búsqueda en lugar de devolver los directorios que conducen a los archivos.

```
$ locate -b archivo
```

Otros argumentos disponibles incluyen:

-e muestra entradas de archivos existentes en el momento en que se ejecuta el comando locate.

-q inhabilita la visualización de errores encontrados en el proceso de búsqueda.

-c muestra la cantidad de archivos que coinciden, en lugar de los nombres de los archivos

El comando locate estándar a veces puede mostrar archivos que han sido eliminados. Esto se debe a que el comando locate busca en la base de datos principal del sistema operativo GNU/Linux. Si esa base de datos no se actualiza, incluso los archivos eliminados pueden aparecer en los resultados de búsqueda. Puedes actualizar manualmente la base de datos ejecutando lo siguiente:

```
# updatedb
```

## 9.2 Empaquetadores, Compresores y Descompresores

La herramienta de empaquetado por excelencia en Linux y Unix es *tar.* Simplemente se trata de meter o guardar todos los archivos o directorios que necesitemos en un mismo archivo. Una forma sencilla de transportar archivos de un sitio a otro, ya sea en nuestra máquina o entre diferentes máquinas. Además *tar*, no solo te permite empaquetar esos archivos y directorios, sino que además te da la posibilidad de comprimirlos para de esta manera que ocupen menos espacio, y sean más fáciles de transportar.

Existen otros programas para manejar de manera óptima y fácil la agrupación y la compactación de un conjunto de archivos, como *gzip, bz2, xz, zip, lha, arj, rar, etc.* La mayoría de los programas para comprimir y descomprimir son secuenciales (solo usan un core), pero hay otros que trabajan en paralelo, que permiten usar dos o más cores logrando un alto desempeño como: pigz, plzip, pbzip2, lrzip, lbzip2, pixz.

Para instalar los programas más comunes en Debian GNU/Linux usar:

```
# apt install arj bzip2 clzip cpio dtrx gzip kgb \
lbzip2 lhasa lrzip lzip lziprecover lzop ncompress \
p7zip-full p7zip-rar pax pbzip2 pigz pixz plzip \
rar rarcrack tarlz unace unace-nonfree unar unp \
unrar unzip xz-utils zip zpaq zstd zutils \
archivemount
```

Podemos conocer los programas que disponemos en nuestro equipo para compresión usando:

```
$ apropos compress
```

**Lo Básico Empaquetar y Desempaquetar** Empecemos por lo más básico, que es empaquetar y desempaquetar usando el comando *tar.* Lo distingo de comprimir y descomprimir, porque esto lo veremos más adelante, ya que se trata de dos conceptos distintos.

Para que sea más sencillo, creamos un conjunto de archivos con los que trabajaré a continuación. Para ello ejecuta lo siguiente:

```
$ touch archivo{1..20}.txt
```

**Empaquetando Archivos** se han creado 20 archivos que vamos a empaquetar a continuación. Para ello, ejecutamos lo siguiente[169]:

```
$ tar -cvf empaquetado.tar *.txt
```

ahora tenemos todos los archivos en un mismo paquete.

**¿Que son esas opciones que he puesto?** -c es para crear un archivo
-v muestra el progreso del comando
-f para indicar el archivo que se va a crear en este caso[170]

**Desempaquetando Archivos** ahora que ya tenemos claro como empaquetar archivos, es necesario que se puedan desempaquetar. Es tan importante una operación como otra. Así, para desempaquetar, tenemos que utilizar la opción *-x*, de extraer. Así, para desempaquetar, ejecuta la siguiente instrucción:

```
$ tar -xvf empaquetado.tar
```

**Extraer un Archivo de un Paquete** para extraer un o algunos archivos del paquete ejecutamos:

```
$ tar -xvf empaquetado.tar archivo1 archivo2
```

**Extraer un grupo de Archivos de un Paquete** si necesitamos extraer un grupo de archivos podemos usar comodines, por ejemplo:

```
$ tar -xvf empaquetado.tar –wildcards '*.php'
```

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

[170] Tienes que tener en cuenta que -f tiene que preceder al nombre del archivo que vas a crear, ya sea que lo utilices por separado o junto con otras opciones. Quiero decir, que -cvf empaquetado.tar funcionará, pero en cambio -fcv empaquetado.tar arrojará un error, esto es algo que debemos tener en cuenta.

**Listando el Contenido** para conocer el contenido del archivo *.tar* usamos:

$ tar -tvf empaquetado.tar

Si no indicamos la opción -v solo mostrará los archivos empaquetados. Utilizando esta opción, además muestra otra información como el propietario, los permisos o la fecha.

**Verificar un Archivo Empaquetado** para verificar el contenido de un archivo empaquetado, usamos:

$ tar -tvfW empaquetado.tar

**Quitando un Archivo del Paquete** Una vez creado un determinado paquete, es posible que necesitemos quitar un archivo, es decir, que el archivo no pertenezca a ese nuevo paquete. Para hacer esto, debemos tener en cuenta que el paquete no puede estar comprimido. En caso de que esté comprimido no funcionará. Así, para borrar un archivo de un paquete ejecutamos:,

$ tar -f empaquetado.tar –delete archivo12.txt

**Añadir un Archivo a un Paquete** otra situación común que nos encontramos con más o menos frecuencia es cuando ya creamos el paquete, y dejamos un archivo fuera. En este caso, hay varias opciones, desde empaquetar de nuevo, a simplemente añadir ese archivo al paquete. Así, por ejemplo, si lo que queremos es simplemente añadir el archivo al paquete recurrimos a esta opción:

$ tar -f empaquetado.tar –append archivo12.txt

pero, ¿y si el archivo ya existe y simplemente lo queremos modificar? En este caso, usamos la opción:

$ tar -f empaquetado.tar –update archivo12.md

No solo esto, sino que también podemos comparar si un archivo que tenemos empaquetado es distinto del que no está empaquetado. Lo cierto es que lo vamos a comparar por fecha y por tamaño, únicamente, pero por lo menos ya sabemos si se ha modificado. Para hacer esto, ejecutamos:

$ tar -f empaquetado.tar –diff file12.txt

Esto devolverá un resultado como:

archivo12.txt: La fecha de modificación es distinta

archivo12.txt: El tamaño es distinto

**Excluir Archivos o Directorios** es común al respaldar excluir algunos archivos o directorios de nuestro respaldo, para ello podemos usar la opción *–exclude* en el que podemos indicar archivo que deseamos excluir del empaquetado, por ejemplo:

$ tar -cvf empaquetado.tar *.txt –exclude='archivo.txt'

o podemos indicar el directorio a excluir:

$ tar -cvf empaquetado.tar *.txt –exclude=carpeta/ignoraEsta

**Cifrar y Descifrar** es posible cifrar[171] el contenido que deseamos empaquetar usando OpenSSL para generar un archivo con extensión *.tar.gz*, por ejemplo, para cifrar el contenido del actual directorio usamos:

$ tar -czf - * | openssl enc -e -aes256 -out archivoSeguro.tar.gz

y para hacer el proceso de descifrado usamos:

$ openssl enc -d -aes256 -in archivoSeguro.tar.gz | tar xz -C prueba

el cual será extraído en el subdirectorio *prueba*.

[171] El cifrado consiste en ocultar y mantener en secreto la información cifrandola, el cifrado garantiza que la información sin formato cuando se almacena o envía por la red sea ilegible. El descifrado permite volver hacer legible la información. Existen distintas formas de cifrado: cifrado simétrico y asimétrico.

**Comprimir y Descomprimir** Hasta el momento todo lo memos hecho empaquetando archivos, pero si lo que requerimos es reducir el tamaño total ocupado, tenemos que comprimir esos paquetes. Respecto a la compresión hay distintas opciones:

- gzip es la compresión por defecto, y la puedes declarar utilizando la opción -z.
- bzip2 en este caso, tienes que utilizar la opción -j.

Ahora necesitamos combinar estas dos opciones con todo lo que hemos visto hasta el momento. Así para empaquetar y comprimir utilizamos:

```
$ tar -cvzf empaquetado.tar.gz *.txt
```

Para el caso de descomprimir:

```
$ tar -xvzf empaquetado.tar.gz
```

Para ver el contenido del archivo:

```
$ tar -tvf empaquetado.tar.gz
```

Para revisar la integridad de un archivo usamos:

```
$ gunzip -t empaquetado.tar.gz
```

también podemos integrar lo visto anteriormente en un programa Bash como el mostrado en (véase 9.8).

Ahora bien, existe una diferencia en cuanto a las posibilidades que ofrece un archivo empaquetado y otro comprimido. Esta diferencia radica en las operaciones que podemos efectuar. Así, con un archivo comprimido, no podremos actualizarlo, al menos en primera instancia. Es decir, no podemos utilizar las opciones –delete, –append o –update.

Una solución es desempaquetar, operar y comprimir, como por ejemplo:

```
$ gunzip empaquetado.tar.gz; tar -f empaquetado.tar –delete
archivo5.txt; gzip empaquetado.tar
```

Estas son algunas de las opciones que permite hacer *tar*. Y digo bien, simplemente algunas de las opciones, y sin lugar a dudas las más habituales. Y es que *tar* es toda una navaja suiza que permite empaquetar archivos y directorios de decenas de formas posibles, no solo lo que hemos visto hasta el momento. También, se puede especificar directorios, excluir archivos por nombre o incluso por patrón, y mucho más.

**Cifrar y Descifrar Archivos** es posible cifrar archivos usando criptografía de clave pública o sólo una clave de cifrado simétrico. La clave que se usa para el cifrado simétrico deriva de la contraseña dada en el momento de cifrar el documento.

GnuPG es un paquete completo que permite generar un par de claves públicas, intercambiar y comprobar la autenticidad de claves, cifrar y descifrar archivos, etc. Para instalar el paquete GnuPG, usamos:

```
# apt install gnupg
```

**Cifrar un Archivo** para cifrar un archivo se usa el programa *gpg* con la opción *-c* la cual nos permitirá indicar la clave de cifrado y su confirmación, por ejemplo:

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

el archivo resultante será: archivo.tar.pgp. Es necesario escoger claves de cifrado robustas, una opción para generarlas es GnuPG, para generar una clave aleatoria de 32 caracteres, usamos:

```
$ gpg –gen-random –armor 1 32
```

**Descifrar un Archivo** para descifrar un archivo cifrado con GnuPG, usamos el comando *gpg* al cual se le indica el nombre del archivo cifrado, por ejemplo:

```
$ gpg archivo.tar.pgp
```

el archivo resultante será: archivo.tar.

**Cifrar y Descifrar Usando tar** podemos generar un archivo compactado y cifrarlo en una mismo comando mediante:

```
$ tar -cvzf - directorio | gpg -c > archivo.tar.gz.gpg
```

también podemos indicar la clave de cifrado -no es seguro pues otro usuario podría verla- mediante:

```
$ tar -cvzf - directorio | gpg -c –passphrase miclave > archivo.tar.gz.gpg
```

y para descifrar usamos:

```
$ gpg -d archivo.tar.gz.gpg | tar -xvzf -
```

**Cifrar y Descifrar Usando ccrypt** este comando permite cifrar usando *AES256*, no viene instalado por omisión en el sistema, pero lo podemos instalar usando:

```
# apt install ccrypt
```

Para cifrar un archivo *archivo.tar* usamos:

```
$ ccrypt archivo.tar
```

esto nos generará un archivo *archivo.tar.cpt*, para descifrarlo usamos:

```
$ ccrypt -d archivo.tar.cpt
```

que nos retornará a nuestro archivo original.

**Cifrar y Descifrar Usando age** este comando permite cifrar usando una clave pública o frase, no viene instalado por omisión en el sistema, pero lo podemos instalar usando:

```
# apt install age
```

Para generar una clave pública en el archivo llave.txt usamos:

```
$ age-keygen -o llave.txt
```

Para cifrar un archivo *archivo.tar* con clave pública usamos:

```
$ tar cvz archivos | age -r llave.txt > archivo.tar.gz.age
```

para descifrarlo usamos:

```
$ age –decrypt -i llave.txt > archivo.tar.gz
```

que nos retornará a nuestro archivo original.
Para cifrar un archivo *archivo.tar* con frase usamos:

```
$ age –passphrase –output archivo.tar.gz.age archivo.tar.gz
```

para descifrarlo usamos:

```
$ age –decrypt –output archivo.tar.gz archivo.tar.gz.age
```

que nos retornará a nuestro archivo original.

**Respaldos Incrementales** Un respaldo incremental es aquel que almacena solamente las diferencias con respecto al respaldo anterior, tar posee dos opciones que nos van a servir para trabajar con respaldos incrementales:

- «-listed-incremental=file», o «-g file», que permite especificar un archivo de respaldo incremental.
- «-incremental» o «-G», que permite analizar un respaldo incremental y saber qué archivos del total están presentes en dicho respaldo.

El archivo de respaldo incremental almacenará metadatos (.snar) que ayudará a la herramienta tar a determinar qué archivos han cambiado desde el último incremental, cuáles se han agregado, o cuales se han eliminado, de modo que tar podrá generar el siguiente respaldo incremental solamente de los cambios del anterior. Por ejemplo:
Creamos el respaldo completo (la base para los respaldos incrementales)[172]:

```
$ tar -cvzf Respaldos/bkp0.tgz -g Respaldos/incremental.snar
datos/
```

Supongamos ahora que realizamos el respaldo incremental del siguiente día, usando;

```
$ tar -cvzf Respaldos/bkp1.tgz -g Respaldos/incremental.snar
datos/
```

de esta manera realizaremos todos los respaldos incrementales necesarios.
Veamos, antes de proceder a restaurar todo, cómo podemos analizar qué contiene cada respaldo incremental. Esto podemos llevarlo a cabo mediante la opción «-incremental» del comando tar, a la que debemos también agregar dos modificadores «-verbose» para ver la información completa. En mi caso he usado «-G» en vez de «-incremental», y «-vv» en vez de «-verbose -verbose»:

```
$ tar -tvvGf Respaldos/bkp1.tgz
```

[172] Aquí usamos para los ejemplos de los respaldos comprimidos con gzip, pero el comportamiento es exactamente el mismo al utilizar bzip2, xz, lmza o cualquier otro formato soportado.

como puede verse en la salida, delante de cada nombre de archivo tenemos un «Y» o un «N». «Y» significa que el archivo está presente en dicho respaldo, y «N» que no lo está. El resultado es consistente con los cambios que hemos realizado sobre los archivos.

Supongamos ahora que perdemos todos los datos y que necesitamos recuperarlos desde el respaldo completo y los incrementales. Para extraer todos los archivos en el mismo estado exacto en el que estaban a la hora de realizar el último respaldo incremental, debemos usar la opción «-extract» o «-x» junto con la opción «-listed-incremental». Como la información de respaldo incremental está almacenada dentro del Tarball, es decir, el Tarball contiene solo los cambios respecto del último incremental, puede pasarse cualquier argumento al «-listed-incremental» o «-g». Usualmente se utiliza /dev/null, o se usa directamente «-incremental» o «-G».

Suponiendo que el directorio datos no existe, vamos a proceder a restaurar todos los respaldo, desde el primero, completo, uno a uno hasta el último incremental disponible, para regenerar el directorio original y su contenido:

```
$ tar xvGf Respaldos/bkp0.tgz
$ tar xvGf Respaldos/bkp1.tgz
$ tar xvGf Respaldos/bkpn.tgz
```

de esta forma se puede restaurar paso por paso todos los respaldo que antes hicimos, y ahora tenemos dentro del directorio *datos/* toda la información de nuevo. Y sí, los archivos tienen el contenido original modificado tal y como quedaron en el último respaldo incremental.

**Zips de la Muerte** Cuando descomprimimos un archivo debemos tener cuidado, ya que el resultado suele ocupar bastante más y puede dejarnos sin espacio en el ordenador. El ratio máximo de compresión de la mayoría de zips está marcado en 1032 a uno, aunque en muchas ocasiones tampoco se alcanza. Sin embargo, desde 1996 se tiene constancia de la existencia de las llamadas "bombas zip" o lo que se conoce popularmente como "el zip de la muerte". Un archivo que sobrepasa este límite de descompresión e inunda nuestro ordenador con miles de millones de Bytes.

Una bomba zip es un archivo comprimido como cualquier otro. Su potencial peligro está en que es muy fácil de esconder y compartir. Pero al descomprimir es cuando colapsa el ordenador al liberar una gran cantidad de datos.

El zip de la muerte más conocido es 42.zip (https://unforgettable.dk), un archivo con un récord de compresión de 106.000 millones a uno. Su autor es desconocido, pero el ratio es impresionante. Estamos hablando de un archivo que pesa únicamente 42 KiloBytes (de ahí el nombre) y que al descomprimirse puede alcanzar los 4.3 GigaBytes. Pero la clave está en que el archivo contiene cinco capas, cada una con 16 archivos. Lo que en total una vez liberados todos ocupan un total de 4.5 PetaBytes.

David Fifield (https://www.bamSoftware.com/hacks/zipbomb/) nos ofrece tres archivos de ejemplo (uno de ellos no recursivo que establece el ratio en 28 millones a uno). El primero donde pasamos de 42kB a 5.5GB, un segundo de 10MB a 281TB y un tercero de 46MB a 4.5 PetaBytes, este último únicamente compatible con el formato Zip64.

**Los antivirus sí las detectan** afortunadamente para la seguridad de los usuarios, esta técnica de la bomba zip es bastante conocida desde hace muchos años. Pese a que el archivo comprimido pasa desapercibido en el sistema, una vez vamos a descomprimirlo es cuando los antivirus lo detectan como un posible peligro y nos advierten antes de iniciar el proceso.

**También los hay en formato TAR** sin tratarse de ningún gusano o virus, otro ataque similar es la 'bomba fork' o 'wabbit'. En este caso, es un archivo que crea copias de sí mismo. Una técnica de la que los usuarios de Linux tampoco se libran, ya que también funciona con archivos TAR.

**Trabajando con Archivos Compactados** Existe la opción de gestionar ficheros comprimidos, gracias a los comandos *zgrep, zegrep, zless, zmore, zdiff, zcmp, zcat* entre otros. Como te puedes dar cuenta la función de cada uno de estos comandos será la misma que su homónimos sin «z» (*grep, egrep, less, more, diff, cmp, cat*) pero para ficheros comprimidos con *gzip* y extensión *.gz*. A saber:

- zcat: Mostar por pantalla el contenido de un fichero comprimido.

  $ zcat archivo.gz | more

  en el caso de necesitar conocer las propiedades del archivo compactado, usamos:

```
$ zcat -l archivo.gz
```

y en el caso de que el comando *zcat* nos muestre avisos, los podemos callar usando:

```
$ zcat -q archivo.gz
```

- zless y zmore: Comandos para examinar ficheros de texto plano o comprimidos de forma paginada por la pantalla.

```
$ zless archivo.gz
```

- zgrep y zegrep: Comando para realizar búsquedas de expresiones regulares en ficheros comprimidos.

```
$ zgrep -i "cadena" archivo.gz
```

- zdiff y zcmp: Comando para comparar ficheros comprimidos.

```
$ zdiff archivo1.gz archivo2.gz
```

De forma análoga para archivos comprimidos usando *bz2*, están los comandos *bzgrep, bzegrep, bzless, bzmore, bzdiff, bzcmp, bzcat* y para archivos comprimidos usando *xz*, están los comandos *xzgrep, xzegrep, xzless, xzmore, xzdiff, xzcmp, xzcat*.

**Desempaquetar o Descomprimir** El uso básico para desempaquetar o descomprimir según la extensión del archivo es el siguiente:

- *.tar Desempaquetar usando: tar xf archivo.tar
- *.tar.bz2 Descomprimir usando: tar xjf archivo.tar.bz
- *.tbz Descomprimir usando: tar xjf archivo.tbz
- *.tbz2 Descomprimir usando: tar xjf archivo.tbz2
- *.tgz Descomprimir usando: tar xzf archivo.tgz
- *.tar.gz Descomprimir usando: tar xzf archivo.tar.gz

- *.tar.lz Descomprimir usando: tar -xf archivo.lz
- *.tar.xz Descomprimir usando: tar -xf archivo.tar.xz
- *.xz Descomprimir usando: tar Jxf archivo.xz
- *.gz Descomprimir usando: gunzip archivo.gz
- *.bz2 Descomprimir usando: bunzip2 archivo.bz2
- *.zip Descomprimir usando: unzip archivo.zip
- *.Z Descomprimir usando: uncompress archivo.Z
- *.7z Descomprimir usando: 7z e archivo.7z
- *.zst Descomprimir usando: zstd -d archivo.zst
- *.lha Descomprimir usando: lha x archivo.lha
- *.arj Descomprimir usando: arj x archivo.arj
- *.zoo Descomprimir usando: zoo x archivo.zoo
- *.lz Descomprimir usando: lzip -d archivo.lz
- *.cpio Desempaquetar usando: cpio -idv < archivo.cpio
- *.rar Descompactar usando: rar x archivo.rar
- * Descompactar[173] usando: unp archivoCompactado
- * Descompactar usando: dtrx archivoCompactado
- * Descompactar usando: unar archivoCompactado

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

```
# apt install unp
```

para descompactar, usamos:

```
$ unp archivoCompactado
```

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

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

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

o usamos:

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

o usamos:

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

para restaurar, usamos:

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

o usamos:

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

o usamos:

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

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

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

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

```
$ gzip ficheros
```

y para descompactarlo usamos:

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

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

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

o

```
$gunzip fichero.gz
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

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

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

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

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

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

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

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

o

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

Para descomprimir usamos:

$ pigz -d archivo.iso.gz

o

$ unpigz archivo.iso.gz

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

$ pigz -9 -k -p3 archivo.iso

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

$ pzip -k -k archivo.iso

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

$ pzip -k -K archivo.iso

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

$ bzip2 ficheros

y para descompactarlo usamos:

$ bzip2 -d fichero.bz2

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

$ bzcat -dc fichero.bz2

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

$ bunzip2 fichero.bz2

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

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

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

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

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

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

podemos descomprimir usando:

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

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

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

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
$ zipcloak archivo.zip
```

Podemos ver la información detallada del archivo comprimido, usando:

```
$ zipdetails archivo.zip
```

Y descomprimir usando:

$ unzip archivo.zip

para descompactar un solo directorio usamos:

$ unzip -d directorio archivo.zip

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

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

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

$ unzip -P 2GAXTW archivo.zip

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

$ unzip -q archivo.zip

para indicar que no se extraigan algunos archivos, usamos:

$ unzip archivo.zip -x *.eps

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

$ unzip -o archivo.zip

para no sobreescribir los archivos al descomprimir, usamos:

$ unzip -n arvhivo.zip

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

$ unzip -u archivo.zip

para actualizar archivos pero no crear nuevos, usamos:

$ unzip -f archivo.zip

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

$ unzip -l archivo.zip

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

$ unzip -v archivo.zip

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

$ unzip -t archivo.zip

Si al descompactar existe algún error, es posible recuperar parte de los archivos mediante:

$ zip -F archive.zip -O archive-fixed.zip

o usar *-FF*, después usar:

$ jar xvf archive-fixed.zip

otra alternativa es usar:

$ bsdtar xf archivo.zip

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

$ for z in *.zip; do unzip "$z"; done

**Formato *7z*** para respaldar y compactar un grupo de archivos y/o directorios en formato *7-zip*, usamos:

$ 7z a archivo.7z ficheros

también podemos pedir que use una clave (password$$) para cifrar el archivo:

$ 7z a archivo.7z ficheros -ppassword$$

para restaurar usamos:

$ 7z e archivo.7z

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

$ for z in *.7z; do 7z e "$z"; done

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

```
$ zstd ficheros -o archivo.zst
```

para restaurar usamos:

```
$ zstd -d archivo.zst
```

o usamos:

```
$ unzstd archivo.zst
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.zst; do unzstd "$z"; done
```

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

```
$ lha a archivo.lha ficheros
```

para restaurar usamos:

```
$ lha x archivo.lha
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.lha; do lha x "$z"; done
```

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

```
$ arj a archivo.arj ficheros
```

para restaurar usamos:

```
$ arj x archivo.arj
$ unarj archivo.arj
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.arj; do arj x "$z"; done
```

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

```
$ zoo a archivo.zoo ficheros
```

para restaurar usamos:

```
$ zoo x archivo.zoo
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.zoo; do zoo x "$z"; done
```

**Formato lz y tlz** para compactar un *archivo* y remplazarlo por *archivo.lz*, usamos:

```
$ lzip -v archivo
```

para compactar todo un dispositivo (por ejemplo */dev/sdc*), usamos:

```
$ lzip -c /dev/sdc > archivo.lz
```

para restaurar usamos:

```
$ tar -xf archivo.lz
$ lzip -d archivo.lz
```

o extraer un archivo multivolumen de un archivo tar

```
$ lzip -cd archivo.tar.lz | tar -xf -
```

Podemos usar *tarlz* que es una combinación de *tar* y *lzip* que trabaja en paralelo para hacer la compresión, por ejemplo de los archivos *a*, *b*, y *c*:

```
$ tarlz -cf archivo.tar.lz a b c
```

y podemos adicionarle los archivos *d* y *e*, usando:

```
$ tarlz -rf archivo.tar.lz d e
```

Para extraer los archivos podemos usar:

```
$ tarlz -xf archivo.tar.lz
```

También podemos usar la versión plzip que trabaja en paralelo. Para compactar un *archivo* y remplazarlo por *archivo.lz*, usamos:

```
$ plzip -v archivo
```

para compactar todo un dispositivo (por ejemplo */dev/sdc*), usamos:

```
$ plzip -c /dev/sdc > archivo.lz
```

y para restaurar usamos:

```
$ plzip -d archivo.lz
```

**Formato cpio** para empaquetar archivos (por ejemplo*.txt) en un archivo .cpio, usamos:

```
$ ls *.txt | cpio -ov > archivo.cpio
```

para desempaquetar, usamos:

```
$ cpio -idv < archivo.cpio
```

**Formato *rar*** para respaldar y compactar un grupo de archivos y/o directorios al formato *rar*, usamos:

```
$ rar a archivo.rar ficheros
```

para restaurar usamos:

```
$ rar x archivo.rar
$ unrar archivo.rar
```

En algunos casos, archivos *rar* de Windows no es posible descomprimirlos correctamente en Linux, para descomprimirlos podemos descargar utilerías GNU para Win32 de:

http://unxutils.sourceforge.net/

entre ellas, descargar *unrar* (es un sólo archivo zip) de la dirección:

http://sourceforge.net/projects/unxutils/

ahora usando Wine es posible descomprimir los archivos desde Linux mediante:

```
$ wine unrar.exe e archivo.rar
```

Para descomprimir archivos *rar* o *zip* rotos, usar:

```
$ unrar e -kb -y nombreArchivo.rar
```

o usamos:

```
$ bsdtar xf nombreArchivo.zip
```

Si se requiere descomprimir múltiples archivos empaquetados podemos usar:

```
$ for z in *.rar; do unrar e "$z"; done
```

**UNP para descompactar casi de cualquier formato** existe un programa llamado *unp* (basado en un Scrip de Perl) que permite descomprimir de casi cualquier formato, para instalarlo hacemos:

```
# apt install unp
```

Es un extractor universal, su uso es de lo más sencillo, a saber:

```
$ unp archivoCompactado
```

Parte de los formatos soportados y por que programas se muestra en la siguiente lista (generada usando: *unp -s*):

7z (p7zip or p7zip-full), ace (unace), ar,deb (binutils), arj (arj), bz2 (bzip2), cab (cabextract), chm (libchm-bin or archmage), cpio,afio (cpio or afio), dat (tnef), dms (xdms), exe (orange or unzip or unrar or unarj or lha), gz (gzip), cpio,afio (cpio or afio), dat (tnef), dms (xdms), exe (orange or unzip or unrar or unarj or lha), gz (gzip), hqx (macutils), lha,lzh (lha), lz(lzip), lzma (xz-utils or lzma), lzo (lzop), lzx (unlzx), mbox (formail and mpack), pmd (ppmd), rar (rar or unrar or unrar-free), rpm (rpm2cpio and cpio), sea,sea.bin (macutils), shar (sharutils), tar (tar), tar.bz2,tbz2 (tar with bzip2), tar.lzip (tar with lzip), tar.lzop,tzo (tar with lzop), tar.xz,txz (tar with xz-utils), tar.z (tar with compress), tgz,tar.gz (tar with gzip), uu (sharutils), xz (xz-utils), zip,cbz,cbr,jar,war,ear,xpi,adf (unzip), zoo (zoo).

**DTRX para descompactar de múltiples formatos** esta herramienta llamada *dtrx* (Do The Right Extraction), es una aplicación de código abierto que simplifica el proceso de extracción de archivos comprimidos. Para su instalación hacemos:

```
# apt install dtrx
```

Y ya estamos listos para comenzar a usarlo en la terminal. Dtrx soporta los formatos comunes de compresión como: tar, bz2, zip, cpio, deb, rpm, gem, 7z, cab, lzh, rar, gz, bz2, lzma, xz, además de binarios deb, gem (ruby), archivos de Installshield e incluso algunos tipos de exe. También descomprime archivos comprimidos con gzip, bzip2, lzma, xz, y otros compresores.

Para descomprimir usamos:

```
$ dtrx archivo.zip
```

Podemos hacer un preview del contenido de un archivo lanzandolo con el parámetro -l:

```
$ dtrx -l archivo.zip
```

Si tenemos un archivo comprimido que a su vez incluye múltiples ficheros, al ejecutar *dtrx* nos va a dar 5 opciones diferentes, para hacer la cosa más o menos automática:

-a siempre extraer los archivos incluidos durante esta sesión

-o extraer los archivos incluidos esta vez.

-n elegir no extraer los archivos incluidos por esta vez.

-v nunca extraer los archivos incluidos durante esta sesión.

-l listar los archivos incluidos.

Si queremos que trabaje de forma recursiva, lo podemos hacer añadiendo la opción -r:

```
$ dtrx -r archivo.tar.gz
```

Además con el parámetro -m podemos extraer los metadatos de archivos *deb* y *gem*:

```
$ dtrx -m archivo
```

**UNAR para descompactar de múltiples formatos** esta herramienta llamada *unar*, es una aplicación de código abierto que simplifica el proceso de extracción de archivos comprimidos. Para su instalación hacemos:

```
# apt install unar
```

Y ya estamos listos para comenzar a usarlo en la terminal, *unar* soporta los formatos comunes de compresión como: zip, rar, 7z, tar, gzip, bzip2, lzma, xz, cab, msi, nsis, exe, iso, bin, arj, arc, pakm, acem, lzh, adf, entre otros.

Para descomprimir usamos:

```
$ unar archivo.zip
```

**Otras opciones** cuando se tiene una lista de archivos de distintas trayectorias o es resultado de una búsqueda, para compactar es preferible usar *afio*. Podemos instalarlo usando:

```
# apt install afio
```

Para compactar, digamos todos los archivos **.?pp* (programas de C++) usar:

```
$ find . -name *.?pp | afio -o -Z fuentes
```

para descompactarlos, usar:

```
$ afio -i -Z fuentes
```

Si se desea compactar usando GZIP, usar:

```
$ cat lista | afio -o -Z -G 9 fuentes
```

Si se desea ver el listado de archivos que contiene fuentes, usar:

```
$ afio -t fuentes
```

Si se desea compactar y mandar a otra máquina usar:

```
$ find . -name *.?pp | afio -o -Z user@servidor%ssh:fuentes
```

Como el uso de *afio* no necesita extensión en el archivo, para descompactarlo de cualquier formato es recomendable usar *unp*, este escoge el mejor método para el archivo en cuestión:

```
$ unp archivoCompactado
```

**Respaldo y/o Restauración Remoto** es posible usar *ssh*[174] en conjunción con *tar o dd*[175] para hacer respaldos y/o restauraciones de forma remota, por ejemplo:

```
$ ssh user@maq tar czf - /dir1/ > /dest/arch.tar.gz
$ ssh user@maq 'cd /dir1/ && tar -cf - file | gzip -9' >arch.tar.gz
$ tar zcvf - /dir | ssh user@maquina "cat > /dest/arch.tar.gz"
$ tar zcf - /dir/ | gpg -e | ssh user@maq 'cat - > arch.tar.gz.gpg'
$ ssh user@maq 'tar zcf - /some/dir' | tar zxf -
$ ssh user@maq 'tar czf - /home/user' | tar xvzf - -C /home/user
$ cat arch.tar.gz | ssh user@maq "tar zxvf -"
$ cat arch.tar.gz | ssh user@maq "cd /dest/; tar zxvf -"
# dd if=/dev/sdvf | ssh user@maq 'dd of=arch.img'
$ ssh root@maq 'dd if=arch.img' | dd of=/dev/sdvf
$ tar cvzf - /dir | ssh root@maq "dd of=/dest/arch.tar.gz"
$ tar cvjf - * | ssh user@maq "(cd /dest/; tar xjf -)"
$ tar cvzf - dir/ | ssh user@maq "cat > /dest/arch.tgz"
$ tar cvzf - /dir | ssh user@maq "dd of=/dest/arch.tar.gz"
$ ssh user@maq "cat /dest/arch.tar.gz" | tar xvzf -
$ tar cvjf - * | ssh root@maq "(cd /dest/; tar xjf - )"
```

Es posible usar *nc*[176] en conjunción con *tar* y *pv* para hacer respaldos y/o restauraciones de forma remota, supongamos que estamos en el IP 192.168.13.230. podemos mandar un archivo grande (digamos un *.iso)* y usar el comando *pv* para ver el progreso de la transferencia usando:

$ tar -zcf - Archivo.iso | pv | nc -l -p 5555 -q 5

y para recibirlo en el otro equipo usamos:

```
$ nc 192.168.13.230 5555 | pv | tar -zxf -
```

Además, ponemos también usar *nc* en conjunción con *tar* y *gpg* para hacer respaldos y/o restauraciones de forma remota, supongamos que estamos en el IP 192.168.13.230, podemos generar un archivo compactado, cifrarlo y enviarlo a otro equipo en una mismo comando mediante:

[174] ssh o Secure Shell, es un protocolo de administración remota que le permite al usuario controlar y modificar servidores remotos a través de un mecanismo de autentificación.

[175] dd o Data Duplicator permite copiar y convertir datos -en Linux cualquier dispositivo y partición se trata y gestiona como un archivo- de archivos a bajo nivel.

[176] El comando *netcat* (*nc*) es una utilidad que permite escribir y leer datos a través de conexiones de red usando los protocolos TCP y UDP

```
$ tar -cvzf - directorio | gpg -c | nc -l 6666
```

y para recibirlo en el otro equipo usamos:

```
$ nc 192.168.13.230 6666 | gpg -d | tar -xvzf -
```

Siempre es mejor opción usar *scp*, pero *nc* cumplirá con su cometido.

**Montar Archivos Compactados con Archivemount** ¿Alguna vez haz necesitado mirar el contenido de un fichero .tar, .tar.gz, entre otros, pero sin tener que descomprimirlo? Tal vez te gustaría extraer solo unos pocos ficheros, puede que lo que te interese es trabajar con una carpeta a la que le modificamos archivos, sin tener que archivar esta carpeta cada cierto tiempo.

En este caso, tenemos un sistema de ficheros bastante interesante que se llama *archivemount*, y que nos permite ver un fichero .tar, .tar.gz, entre otros. Como si de una carpeta local más se tratara. Al desmontar este sistema de fichero, se crea el mismo fichero de nuevo con los cambios hechos, de forma automática, y con previa copia del anterior.

Archivemount es un sistema de archivos basado en FUSE para variantes de Unix, incluido Linux. Su propósito es montar archivos (es decir, tar, tar.gz, etc.) en un punto de montaje donde se puede leer o escribir como con cualquier otro sistema de archivos. Esto hace que el acceso al contenido del archivo, que puede estar comprimido, sea transparente para otros programas, sin descomprimirlos.

Para instalarlo usamos:

```
# apt install archivemount
```

Para usarlo, necesitamos un directorio cualquiera donde montar nuestro *archivo.tar.gz*, por ejemplo *mnt*, entonces hacemos:

```
$ archivemount archivo.tar.gz mnt
```

ahora podemos ingresar al directorio *mnt* y trabajar con los archivos que tenia nuestro *archivo.tar.gz*, podemos visualizar el contenido, agregar o borrar archivos y al concluir para guardar los cambios necesitamos desmontarlo usando:

```
$ fusermount -u mnt
```

Esto generará un nuevo *archivo.tar.gz* con los cambios y el archivo anterior se guardará en *archivo.tar.gz.orig*.

**Nota**: si el archivo es de varios GigaBytes, entonces el montado y acceso a los archivos puede ser un proceso lento, para ello se desarrollo *ratarmount* (aplicación en Python), que permite montar un archivo *.tar, tar.gz*, entre otros, en modo de solo lectura, pero es notoriamente rápido para archivos compactados de varios GigaBytes de tamaño.

**Montar Archivos Compactados con Ratarmount** permite montar para su lectura un archivos .tar, tar.gz, entre otros. Esta es una aplicación de Python que se puede instalar para todos los usuario usando *pip3*, para ello usamos:

```
# pip3 install ratarmount
```

o instalar a nivel de usuario usando *pip3*, para ello usamos:

```
$ pip3 install ratarmount
```

usando instalación a nivel de usuario, para montar (la primera vez creará el índice que permitirá el acceso rápido a los archivos), usamos:

```
$ ~/.local/bin/ratarmount archivo.tar.gz mnt
```

una vez generado el índice (este proceso es algo lento), podemos ingresar al directorio *mnt* y trabajar con los archivos que tiene nuestro *archivo.tar.gz*, podemos visualizar el contenido y copiar archivos con una velocidad de acceso alta. Al concluir su uso, necesitamos desmontarlo usando:

```
$ fusermount -u mnt
```

como parte del proceso de acceso a nuestros archivo, se genera un archivo tmpXXXX, el cual podemos borrar usando:

```
$ rm tmp*
```

para más información del proyecto, podemos consultar:

https://github.com/mxmlnkn/ratarmount

## 9.3 Copiar Archivos Entre Equipos

**scp** permite transferir archivos y/o directorios de una máquina a otra de forma cifrada usando *SSH* (Secure Shell) que es un protocolo de administración remota que le permite al usuario controlar y modificar servidores remotos a través de un mecanismo de autentificación[177]. El comando *scp* (Secure Copy Protocol) tiene una sintaxis similar al del comando *cp*, con la salvedad que es necesario indicar el usuario, la máquina y el subdirectorio de trabajo del archivo y/o directorio para el destino, fuente o ambos.

Por ejemplo, si se desea transmitir un archivo a una máquina *192.168.13.230* con usuario *antonio*, en el directorio *~/Datos/* estando en sesión en otra máquina, se usa la siguiente sintaxis:

```
$ scp archivo.dat antonio@192.168.13.230:~/Datos/
```

Si se desea transmitir un subdirectorio a la máquina *192.168.13.230*, en el directorio *home* del usuario (denotado con .), se usa la siguiente sintaxis:

```
$ scp -r Directorio antonio@192.168.13.230:.
```

también podemos decirle que excluya algunos archivos (*.mp3) de la copia, usando:

```
$ scp -r !(*.mp3) Directorio antonio@192.168.13.230:.
```

Si se desea copiar un archivo de una máquina remota a nuestra máquina, usamos:

```
$ scp antonio@192.168.13.230:~/archivo ~/destino/
```

o de forma alternativa usamos (. indica el directorio donde el usuario se encuentra):

```
$ scp antonio@192.168.13.230:~/archivo .
```

Si se desea copiar de una máquina remota a otra máquina remota, usamos:

[177] En Android podemos hacer uso de SSH/SFTP mediante aplicaciones como: Termius, JuiceSSH, Mobile SSH, Advanced Client app, ConnectBot.

```
$ scp user1@HOST1:~/archivo user2@HOST2:~/
```

Si se desea transferir múltiples archivos podemos usar:

```
$ scp file1.txt file2.txt user@HOST:/home/user/
```

o de forma alternativa usamos (. indica el directorio donde el usuario se encuentra):

```
$ scp user@Host:/home/user/\{file1.txt,file2.txt\} .
```

En el caso que se quiera limitar el ancho de banda en la transmisión de archivos por *scp*, usar:

```
$ scp -l 400 user@server:/home/user/* .
```

En el caso de que se desee usar otro puerto distinto al de omisión (22) usar:

```
$ scp -P 4455 file.txt user@HOST:/home/user/file.txt
```

En el caso de querer incrementar la velocidad de transferencia en el uso de *scp*, la opción más viable es el cambiar el cifrado usada por omisión por otras como *3des-cbc, aes128-cbc, aes192-cbc, aes256-cbc, aes-128-ctr, aes192-ctr, eas256-ctr, arcfour256, arcfour, blowfish-cbc y cast128-cbc* mediante:

```
$ scp -c blowfish user@server:/home/user/file .
```

o de forma alternativa usamos:

```
$ scp -c arcfour256 user@HOST:/home/user/file .
```

Si adicionalmente se quiere compactar para reducir el tiempo de transferencia, usamos:

```
$ scp -C SourceFile user@HOST:/home/user/TargetFile
```

Si se desea que no se muestre información de la transferencia de los archivos al usar *scp* usar:

$ scp -q SourceFile user@HOST:/home/user/TargetFile

o si desea ver más información en la transferencia usar:

$ scp -v SourceFile user@HOST:/home/user/TargetFile

Si se instala *sshpass*, entonces hacemos:

$ sshpass -p "your_password" scp -r backup_user@target_ip:/home/ /backup/$name

**rsync** es una aplicación libre y multiplataforma que ofrece transmisión eficiente de datos incrementales, que opera también con datos comprimidos y cifrados. Mediante una técnica de Delta Encoding, que permite sincronizar archivos y directorios entre dos máquinas de una red o entre dos ubicaciones en una misma máquina, minimizando el volumen de datos transferidos. Es ideal para trabaja en varias máquinas en las que se desea tener sincronizada una o más carpetas, para instalar usamos:

# apt install rsync

La sintaxis básica es:

$ rsync [Opciones[178]] Origen [Origen]... Destino

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

$ rsync ~/Datos/ ~/Respaldo/

[178] Algunas opciones son: -r recursivo, -b backups, -R relativo, -u actualiza, -p muestra el progreso, -c comprime, -p preserva permisos, -v muestra lo que hace, -q trabaja en modo silencioso, -l preserva ligas simbólicas, -H preserva enlaces duros, -t preserva tiempos de modificación, etc.

Si queremos saber que hará el comando pero sin hacer la operación indicada, podemos usar la opción –dry-run, por ejemplo:

$ rsync –dry-run ~/Datos/ ~/Respaldo/

hay varias opciones que podemos usar en *rsync*, ejemplo de ellas es:

$ rsync –verbose –recursive –links –hard-links –times –delete –stats ~/Datos/ ~/Respaldo/

en este caso se sincronizaría el contenido de ~/Datos con *~/Respaldo*, pero lo hará mostrando lo que transfiere, de forma recursiva, conservará enlaces simbólicos y sus tiempos, borrará los archivos que estén en el destino pero no en el origen y al terminar mostrará las estadísticas de la transferencia.

Para hacer la transmisión cifrada entre equipos, podemos usar *rsync* conjuntamente con *ssh*, supongamos que se esta en la máquina y quiere tener sincronizada la carpeta ~/Datos con mas de un equipo, mediante *ssh* usando un puerto *343*, en la máquina *192.168.13.230* y usuario *antonio*, usar:

$ rsync –partial –recursive –links –hard-links –times –verbose –delete –stats ~/Datos/ -e 'ssh -p 343' antonio@192.168.13.230: ~/Respaldo/

por supuesto esto puede hacerse en cualquier dirección, i.e. de la máquina remota a nuestra máquina o viceversa, ejemplo:

$ rsync –partial –recursive –links –hard-links –times –verbose –delete –stats -e 'ssh -p 343' antonio@192.168.13.230: ~/Respaldo/ ~/Datos/

Los siguientes ejemplos permiten copiar los archivos (por ejemplo *.mp3) pero preservan la estructura de directorios que existía en su lugar de origen:

$ rsync -a -m –include '*/' –include '*.mp3' –exclude '*' ~/miDirectorio/ ~/OtrDirectorio

$ rsync -a –prune-empty-dirs –include '*/' –include '*.mp3' –exclude '*' ~/miDirectorio/ ~/OtrDirectorio

inicia la búsqueda en *~/midirectorio/* y los copia en *~/OtroDirectorio*.

Algunas veces necesitamos cambiar la prioridad de ejecución de un proceso *rsync* para evitar saturar el sistema, para ello podemos usar *ionice* que cambiara la prioridad de los comandos en ejecución, en este caso para todos los procesos de rsync se ejecutarán solo cuando la carga en el sistema sea ligera, mediante:

```
$ pgrep rsync | xargs ionice -c3 -p
```

Entre las opciones para excluir archivos o directorios en el uso de rsync, tenemos:

```
$ rsync -av –exclude 'archivo.txt' /fuente/ /destino/
$ rsync -av –exclude 'miDirectorio' /fuente/ /destino/
$ rsync -av –exclude '*.mp3' /fuente/ /destino/
$ rsync -av –exclude '*.mp3' –exclude '*.txt' /fuente/ /destino/
$ rsync -av –exclude={'*.mp3', '*.txt'} /fuente/ /destino/
```

si tenemos un *archivo.txt* con el contenido a excluir, podemos usar:

```
$ rsync -av –exclude-form={'archivo.txt'} /fuente/ /destino/
```

podemos excluir por el tamaño de los archivos, ejemplos:

```
$ rsync -av –max-size=500m /fuente/ /destino/
$ rsync -av –min-size=1m /fuente/ /destino/
```

**pssh** permite transferir o copiar archivos a múltiples servidores en Linux con un mismo comando:

- pscp - es una utilería para copiar archivos en paralelo a múltiples equipos
- prsync - es una utilería para transferir de forma eficiente archivos entre múltiples equipos en paralelo

- pnuke - permite concluir procesos en múltiples equipos en paralelo
- pslurp - permite copiar archivos de múltiples equipos a un equipo central en paralelo

Si creamos un archivo *Hosts.txt*, con los *IPs* como el siguiente:

192.168.0.3:22
192.168.0.9:22

podemos usar para copiar un archivo a múltiples servidores:

```
$ pscp -h Hosts.txt -l USR -Av wine-1.7.55.tar.bz2 /tmp/
```

o de forma alternativa usamos:

```
$ pscp.pssh -h Hosts.txt -l USR -Av wine-1.7.55.tar.bz2 /tmp/
```

donde:

-h indica que se lean los *IPs* del archivo indicado
-l se indica el usuario a usar en todos los equipos.
-A solicita el password para ser enviado a *ssh*
-v visualiza las operaciones y mensajes que genera el comando

Podemos copiar directorios a múltiples servidores, usando:

```
$ pscp -h myscpHosts.txt -l USR -Av -r Android\ Games/
/tmp/
```

o de forma alternativa:

```
$ pscp.pssh -h myscpHosts.txt -l USR -Av -r Android\ Games/
/tmp/
```

**nc** el comando *netcat* (*nc*) es una utilidad que permite escribir y leer datos a través de conexiones de red usando los protocolos TCP y UDP. Supongamos que estamos en el IP 192.168.13.230. Podemos generar un archivo compactado, cifrarlo y enviarlo a otro equipo en una mismo comando mediante:

```
$ tar -cvzf - directorio | gpg -c | nc -l 6666
```

y para recibirlo en el otro equipo usamos:

```
$ nc 192.168.13.230 6666 | gpg -d | tar -xvzf -
```

Siempre es mejor opción usar *scp*, pero *nc* cumplirá con su cometido.

## 9.4 Respaldo y Restauración

Una copia de seguridad (respaldo, copia de respaldo en inglés backup y data backup) es una copia de los datos originales que se realiza con el fin de disponer de un medio para recuperarlos en caso de su pérdida. Las copias de seguridad son útiles ante distintos eventos y usos: recuperar los sistemas informáticos y los datos de una catástrofe informática, natural o ataque; restaurar una pequeña cantidad de archivos que pueden haberse eliminado accidentalmente, corrompido, infectado por un virus informático u otras causas; guardar información histórica de forma más económica que los discos duros y además permitiendo el traslado a ubicaciones distintas de la de los datos originales; etc.

El proceso de copia de seguridad se complementa con otro conocido como restauración de los datos, que es la acción de leer y grabar en la ubicación original u otra alternativa los datos requeridos. La pérdida de datos es muy común en algún momento.

Ya que los sistemas de respaldo contienen por lo menos una copia de todos los datos que vale la pena salvar, se deben de tenerse en cuenta los requerimientos de almacenamiento. La organización del espacio de almacenamiento y la administración del proceso de efectuar la copia de seguridad son tareas a las que debemos dedicar tiempo y tener la certeza que están bien hechas. Para brindar una estructura de almacenamiento adecuada y segura existen muchos tipos diferentes de dispositivos físicos y en la nube para almacenar datos que son útiles para hacer copias de seguridad, cada uno con

sus ventajas y desventajas a tener en cuenta para elegirlos, como duplicidad, seguridad en los datos y facilidad de traslado.

Antes de que los datos sean enviados a su lugar de almacenamiento se lo debe seleccionar, extraer y manipular. Se han desarrollado muchas técnicas diferentes para optimizar el procedimiento de efectuar los respaldos. Estos procedimientos incluyen entre otras optimizaciones para trabajar con archivos abiertos y fuentes de datos en uso y también incluyen procesos de compresión, cifrado, y procesos de deduplicación, entendiéndose por esto último a una forma específica de compresión donde los datos superfluos son eliminados.

**Tipos de Respaldos** Existen distintos tipos de respaldo, entre los que destacan:

- Respaldo Completo: Se copian todos los archivos que pueda haber en una trayectoria.
- Respaldo Diferencial: Es una copia intermedia entre la completa y la incremental. Es decir, copiara tanto los archivos que se han creado nuevos como los que se han modificado.
- Respaldo Incremental: Solo copiara los archivos que hayan sido modificados tras haber hecho una copia de seguridad previa de tipo completo o diferencial. Para ello compara las fechas de modificación de los archivos de la fuente y los de la copia previa y si hay diferencias el Software tomara la decisión de copiar solo aquellos que se hayan modificado. Lo bueno de esta copia es que no es tan pesada como la completa y permite actualizar solo lo que te interesa.
- Respaldo Espejo: Este modo de respaldo es similar al respaldo completo, con la salvedad de que los archivos no se pueden comprimir para tratar de garantizar su restauración, por lo que además de ser menos segura también ocupa más espacio de almacenamiento.

**Software de Copias de Seguridad** Existe una gran gama de programas en el mercado para realizar copias de seguridad, es importante definir previamente los requerimientos específicos de respaldo y restauración para determinar el Software adecuado a nuestras necesidades. Existe una gran

cantidad de programas adaptados a cada necesidad, por ejemplo para respaldar toda la máquina podemos usar:

- Clonezilla
- Mondo Rescue
- Partimage
- Ping
- Redo Backup

Para hacer respaldos desde el ambiente gráfico:

- Bacula
- Amanda
- Backupninja
- Areca Backup
- BackupPC
- Keep
- UrBackup
- Back in Time
- Timeshift
- Simple Backup Solution
- KBackup
- Kup Backup System
- Grsync

Para hacer respaldos desde la línea de comandos:

- tar (véase 9.2)

- scp (véase 9.3)
- rsync (véase 9.3)
- rsnapshot
- bup
- restic
- rdiff-backup
- duplicity
- rclone

Para el adecuado respaldo de ficheros con datos de carácter personal es común que las copias de seguridad de dichos datos se almacenen cifrados y en una ubicación diferente al lugar de origen.

La copia de seguridad es el mejor método de protección de datos de importancia, pero siempre existe la posibilidad de que la copia de datos no haya funcionado correctamente y en caso de necesidad de restauración de los datos no podamos realizarlo ya que la información de la copia de seguridad puede encontrarse corrupta por diversos motivos:

- el medio en el que se realizaba la copia se encuentra dañado.
- los automatismos de copia no se han ejecutado correctamente.
- y otros muchos motivos que pueden causar que nuestras copias de seguridad sean incorrectas, y por lo tanto inútiles.

Para evitar este problema es muy importante que nos cercioremos de que hacemos las copias correctamente y comprobemos que somos capaces de restaurar la copia de seguridad a su ubicación original, comprobando así que la copia sea correcta y que somos capaces de restaurarla y conocemos el método de restauración, ya que en caso de necesidad crítica los nervios afloran y nos pueden echar por tierra nuestra labor de copia al realizar algún paso erróneo a la hora de restaurar los datos.

Siempre que podamos debemos cifrar los respaldos, esto los mantiene más seguros. Si alguien llegara a tener acceso a datos de respaldo sin el debido cifrado, este se podría restaurar y con ello podría utilizar toda la información del mismo.

**rdiff-backup** es capaz de realizar una copia casi exacta del directorio origen (sin cifrar), preservando toda la información de los recursos: permisos, usuarios y grupos, fecha de modificación, enlaces simbólicos, ficheros fifos, etc., manteniéndola independientemente de la plataforma y sistema de ficheros donde se almacene, gracias a que toda esa información se guarda en ficheros independientes de metadatos. Además seremos capaces de recuperar una instantánea exacta de un determinado día, restaurando el estado que en ese momento tenía nuestro directorio (fichero eliminados, modificados, etc.).

Otra característica importante es la eficiencia del espacio usado. Los respaldos incrementales que utilizan herramientas como *backup-manager*, realizan una copia completa de los recursos modificados, en cambio, *rdiff-backup* sólo almacena las fracciones de datos que realmente han cambiado.

El respaldo de recursos remotos es otro de sus puntos fuertes y que considero importante destacar. La transferencia de recursos remotos esta optimizada; sólo se transfiere por la red la información que realmente ha cambiado, haciendo un uso eficiente del ancho de banda. El único requisito es que en la máquina remota también se encuentre instalado *rdiff-backup*. Para su instalación hacemos:

```
# apt install rdiff-backup
```

Realizar copias de seguridad con *rdiff-backup* es realmente sencillo, es como si estuviéramos usando el comando *cp* de Linux. Únicamente tendremos que indicar el directorio origen (del que se quiere realizar el respaldo) y el directorio destino (donde se almacenará el respaldo).

```
$ rdiff-backup dir_ori dir_dest
```

Podemos encontrarnos varios escenarios posibles dependiendo de donde se encuentren situados los directorios origen y destino.

- directorio origen local y destino local
- directorio origen remoto y destino local
- directorio origen local y destino remoto

Independientemente de cada caso su uso es idéntico, sólo cambia la forma de acceder a los directorios remotos.

Por ejemplo, el comando que deberíamos lanzar para realizar un respaldo de nuestro directorio local /home/autentia al directorio destino /mnt/backup situado en la misma máquina sería:

```
$ rdiff-backup /home/autentia /mnt/backup
```

Si alguno de los dos directorios estuviese en una máquina remota deberíamos hacer preceder al directorio del usuario y el nombre de la máquina. Por ejemplo, vamos a imaginar que queremos realizar el respaldo del directorio /etc de la maquina mac1 (utilizando el usuario backup para acceder a la máquina) al directorio /mnt/backup/mac1 de nuestra máquina local. El comando que deberíamos ejecutar sería:

```
$ rdiff-backup backup@mac1::/etc /mnt/backup/mac1
```

Obviamente, para que este comando funcione en modo Script, deberemos tener configurada correctamente nuestra máquina para acceder al otro equipo sin necesidad de contraseña.

Para restaura el directorio home de la copia realizada el 10 de Enero de 2021 en /tmp/home.

```
$ rdiff-backup -r 2021-01-10 /mnt/backup/home /tmp/home
```

Otro ejemplo, restauramos la copia actual situada en la máquina remota mac1 en el directorio /temp.

```
$ rdiff-backup -r now backup@mac1::/mnt/backup /temp
```

podremos obtener el listado con los cambios incrementales de un fichero:

```
$ rdiff-backup -l /mnt/backup/file
```

Con -list-changed-since podremos saber cuántos ficheros han cambiado desde una marca de tiempo en concreto. Por ejemplo con el siguiente comando estamos viendo que ficheros han cambiado en los últimos 10 días.

```
$ rdiff-backup -list-changed-since 10D /mnt/backup/etc
```

-list-at-time lista todos los ficheros que estuvieron presentes en un determinado momento. Esto incluye tanto fichero eliminados como aquellos que no han sido modificados.

```
$ rdiff-backup -list-at-time 5D /mnt/backup/home
```

Y por último podremos comparar los cambios producidos entre el resplado y un directorio en concreto. rdiff-backup proporciona dos opciones -compare y -compare-at-time; este último para poder comparar con una marca de tiempo determinada. Por ejemplo:

```
$ rdiff-backup -compare /home /mnt/backup/home
$ rdiff-backup -compare-at-time 2W /home /mnt/backup/home
```

**duplicity** es una herramienta avanzada de copias de seguridad cifradas, disponible para la línea de comandos. Está escrita en *Python*, usando herramientas como *librsync* y *GnuPG*.

Los archivos obtenidos se encuentran en formato *tar* y, si lo creemos oportuno, pueden estar o no cifrados. Además, permite realizar copias incrementales que nos permite ahorrar espacio. Para su instalación hacemos:

```
# apt install duplicity
```

Para hacer una copia de seguridad de los datos que queramos, solo hay que utilizar el nombre de la herramienta, seguido de la ruta de los datos que queremos copiar y, a continuación, del destino donde queremos almacenarla. Es decir, algo como esto:

```
$ duplicity ~/documentos file:///backup
```

la recuperación usando esa carpeta como destino. Para lograrlo, escribimos algo como esto:

```
$ duplicity file:///backup recupera
```

Como cabía esperar, la herramienta nos pide la frase de paso para cifrar antes de iniciar la recuperación.

A continuación, vamos a repetir la copia anterior, pero evitando que se copie el directorio borradores y todo su contenido. Lo logramos añadiendo la opción -exclude, seguida de la ruta a excluir

```
$ duplicity ~/documentos file:///backup –exclude ~/documentos/borradores
```

Podemos usar duplicity en un Scrip, por ejemplo:

```
export PASSPHRASE='ClaveCifrado'
export FTP_PASSWORD='ClaveServidor'
## Respalda localmente
duplicity /home/antonio/trabajo file:/home/antonio/Respaldos
## Verifica
duplicity verify file:/home/antonio/Respaldos /home/antonio/trabajo
## Respalda en el servidor
duplicity /home/antonio/trabajo scp://antonio@\
192.168.13.230//home/antonio/Respaldos
unset PASSPHRASE
unset FTP_PASSWORD
```

**rclone** es una aplicación libre para sistemas de tipo Linux, Unix y Microsoft Windows, se trata de una herramienta en línea de comando para sincronizar archivos y directorios desde la computadora con los proveedores más importantes de alojamiento de contenidos en la nube[179]. También permite efectuar copias dentro de nuestro propio sistema de archivos. Está escrito en lenguaje de programación *Go* y se basa en la utilidad *rsync*. Para instalarlo hacemos:

```
# apt install rclone
```

Para crear, editar o eliminar un dispositivo remoto, es necesario utilizar la siguiente orden

```
$ rclone config
```

Se entra a una sesión interactiva, en la que configuraremos a *rclone* con los datos de nuestra cuenta en la nube, especificando el nombre del recurso mediante el cual podremos identificar nuestro servicio de la nube (por ejemplo antonio_google) en nuestra máquina.

Por ejemplo, podemos conocer el monto de espacio disponible en el servidor, usando:

```
$ rclone about antonio_google:
```

[179] Algunos de los servicios en la nube más importantes soportados por Rclone son: Amazon Drive, Amazon S3, Box, Dropbox, Google Cloud Storage, Google Drive, Hubic, Mega, Microsoft OneDrive, Nextcloud, OpenDrive, Oracle Cloud Storage, ownClowd, pCloud, QingCloud Object Storage, Webdav, Yandex Disk.

listamos el contenido de carpetas y contenedores del servidor en la nube:

```
$ rclone lsd antonio_google:
```

podemos crear un directorio para hacer la sincronización:

```
$ rclone mkdir antonio_google:Respaldos
```

y realizamos la sincronización del contenido de nuestro equipo con el de la nube, usamos:

```
$ rclone sync /home/antonio/trabajo/ antonio_google:Respaldos
-P
```

si necesitamos sincronizar el contenido de la nube con nuestro equipo, usamos:

```
$ rclone sync antonio_google:Respaldos /home/antonio/trabajo/
-P
```

Podemos sincronizar el contenido de una unidad o directorio, por ejemplo en el directorio *nube*, usando:

```
$ rclone mount antonio_google:Respaldos ./nube
```

este permanecerá montado hasta que usemos Ctrl-C para terminar el comando rclone, en caso de ser necesario desmontar el punto de montaje manualmente, usamos:

```
$ fusermount -u ./nube
```

Algunas otras opciones de trabajo son:

- rclone config - crear, editar o eliminar un dispositivo remoto.
- rclone copy - Copiar archivos del origen al destino.
- rclone sync - Sincronizar archivos, modificando el destino únicamente.
- rclone move - Copiar archivos del origen al destino.

- rclone delete - Borrar archivos.
- rclone purge - Borrar todo el contenido de la carpeta.
- rclone mkdir - Crear una carpeta si no existe.
- rclone rmdir - Borrar una carpeta.
- rclone rmdirs - Borrar carpetas vacías.
- rclone check - Comprueba si los archivos en el origen y destino coinciden.
- rclone ls - Lista todos los archivos.
- rclone lsd - Lista todas las carpetas y contenedores.
- rclone size - Devuelve el tamaño y el número de objetos.
- rclone version - Muestra el número de versión.
- rclone authorize - Autorización remota.
- rclone cat - Concatena archivos y los envía a la salida estándar.
- rclone copyto - Copiar archivos del origen al destino, exceptuando los ya copiados.
- rclone listremotes - Lista todos los dispositivos remotos.
- rclone moveto - Mueve archivos o carpetas desde el origen al destino.
- rclone about - Devuelve información sobre el dispositivo remoto.

## 9.5 Incrementando la Seguridad a Nivel Usuario

La mejor opción, es elegir una distribución de GNU/Linux que nos permita mantener el sistema actualizado, instalar sólo los paquetes que necesitamos y que estos provengan de una fuente confiable, además de cifrar las particiones del sistema operativo y de datos del usuario.

**Borrado de Archivos o Particiones de Forma Segura** La herramienta *shred* ("ha-cer trizas" en español) -provista por el paquete *coreutils* instalado por omisión-, permite sobrescribir un archivo o dispositivo para ocultar y eliminar todo su contenido, sin dejar rastro alguno. A fin de que sea imposible de recuperar para herramientas de Hardware que hacen análisis magnético de sectores[180], *shred* sobrescribe repetidamente el contenido con valores aleatorios, haciendo un uso intensivo de disco[181]. A su vez permite eliminar el archivo al finalizar.

Por ejemplo, se procede a sobreescribir y borrar el archivo con *shred*:

```
$ shred -n 5 -u -v -z miArchivo.dat
```

las opciones utilizadas son las siguientes:

-n 5: 5 iteraciones de sobrescritura.

-u: borrar el archivo al finalizar.

-v: muestre el progreso.

-f: fuerza el cambio de permisos para permitir la escritura de ser necesario.

-z: rellenar con ceros al final para ocultar el propio Shredding.

Se observa que en cada pasada se alterna entre rellenado con ceros, con datos aleatorios y con unos (ffffff). Esto se hace para maximizar la efectividad del borrado o Shredding, de forma que todos los Bits resulten modificados al menos una vez. Finalmente se renombra y se borra. Esto ocurre para hacer el Shredding del propio nombre del archivo en la entrada de directorio.

Lo podemos usar en múltiples archivos a la vez, mediante:

[180] Es importante aclarar que esta herramienta se basa en la suposición importante, de que el sistema de archivos sobrescribe los datos en el lugar. Generalmente es así, pero muchos sistemas de archivos modernos no lo respetan. Por ejemplo los sistemas de archivos con Journal o estructurados en Log (por ejemplo *ext4*); sistemas de archivos que escriben datos redundantes como los esquemas *RAID*; sistemas de archivos que soportan Snapshots (*LVM* o *ZFS*); sistemas de archivos que hacen un uso intensivo de Cache en locaciones temporales (*NFS*); sistemas de archivos comprimidos; etc.

En tales casos *shred* no es efectivo, o no se garantiza que lo sea.

Para el caso de los sistemas de archivos *ext3* y *ext4* esta advertencia aplica sólo si el Journal está habilitado tanto para los datos como metadatos (modo data=journal). En los modos data=ordered (por defecto) y data=writeback, *shred* funciona correctamente.

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

```
$ shred -u -v *.txt
```

o en una partición del disco (/dev/sda2/), mediante:

```
# shred -vfz /dev/sda2
```

También es posible usar el paquete *wipe*, para instalarlo usamos

```
# apt install wipe
```

y lo usamos mediante:

```
$ wipe archivo
$ wipe -r -q directorio/*
# wipe /dev/sda2
```

o el paquete *secure-delete*, para instalarlo usamos:

```
# apt install secure-delete
```

este paquete consta de cuatro herramientas, a saber:

- srm - utilizado para borrar archivos o directorios que se encuentren en disco

  ```
  $ srm archivo
  $ srm -r directorio/*
  ```

- sdmem - utilizado para limpiar restos de información en la memoria (RAM) de la máquina

  ```
  # sdmem -f -v
  ```

- sfill - utilizado para limpiar los restos de información del espacio vacío del disco

  ```
  # sfill -v /home/usuario
  ```

- sswap - utilizado para limpiar los restos de información de la partición Swap, para conocerla usamos:

  ```
  $ swapon
  ```

  supongamos que la partición Swap es: /dev/sda2, entonces:

  ```
  # swapoff /dev/sda2
  # sswap /dev/sd2
  # swapon /dev/sda2
  ```

**Cifrado de Discos y Particiones** Al momento de instalar el sistema operativo una buena práctica de seguridad es solicitar que use particiones cifradas para nuestros datos y el sistema operativo. Esto permite mantener a nuestros datos lejos de miradas indiscretas. Además cada disco externo o unidad Flash se recomienda cifrar, esto es posible al formatear el dispositivo -casi todos los escritorios de GNU/Linux tienen integrada una opción para ello-, así en caso de perdida o robo del dispositivo nuestros datos permanecen ilegibles para cualquiera.

En caso que nuestro escritorio no tenga instalada una opción gráfica para formatear y cifrar dispositivos, podemos usar GNOME Disks (también conocida como *gnome-disk-utility*, o *GNOME Disks*) es una aplicación gráfica de *udisks* incluido en el paquete *gnome-disk-utility*. Se puede utilizar GNOME Disk para la gestión de particiones (cifrar), motorización de tipo *SMART*, evaluación comparativa y Software *RAID*.

Para instalar usamos:

```
# apt install gnome-disk-utility
```

y para usar:

```
# gnome-disks
```

esto nos permite entre otras cosas al formatear una unidad y cifrarla.

**Cifrar Discos y Particiones con cryptsetup** es uno de los Softwares de cifrado de disco más usado. Este se encarga de cifrar/descifrar los datos escritos en un dispositivo de almacenamiento. El cifrado a nivel de disco garantiza que los archivos siempre se almacenen en el disco de forma cifrada. Los archivos solo están disponibles para poder ser leídos mientras el sistema está en ejecución y desbloqueado por uno de los usuarios con acceso a la clave de descifrado. Una persona no autorizada que intente acceder al contenido del disco, solo encontrará datos confusos, en lugar de los archivos reales.

Todos los métodos de cifrado de disco funcionan de tal manera que, aunque el disco realmente contiene datos cifrados, el sistema operativo lo ve como los datos legibles normales, siempre que el contenedor cifrado (es decir, la parte del disco que contiene los datos cifrados) ha sido desbloqueado y montado en alguna partición del sistema.

Instalar Paquete usamos:

```
# apt install cryptsetup
```

Es bueno conocer el rendimiento del programa *cryptsetup* para los diferentes tipos de cifrado soportado, para ello usamos:

```
# cryptsetup benchmark
```

algunos de ellos son:

> PBKDF2-sha1, PBKDF2-sha256, PBKDF2-sha512, PBKDF2-ripemd160, PBKDF2-whirlpool, argon2i, argon2id, aes-cbc 128, serpent-cbc 128, twofish-cbc 128, aes-cbc 256, serpent-cbc 256, twofish-cbc 256, aes-xts 256, serpent-xts 256, twofish-xts 256, aes-xts 512, serpent-xts 512, twofish-xts 512

esto nos permitirá elegir el algoritmo óptimo para nuestro equipo, que nos de el mejor rendimiento en el cifrado y mayor seguridad posible.

El primer paso es escoger la unidad de disco que vamos a cifrar, podemos usar el comando *lsblk* para obtener una lista de dispositivos de bloque montados en el equipo de cómputo, mediante:

```
$ lsblk
```

supongamos que trabajaremos con: */dev/sdb1*.

Crear partición cifrada

```
# cryptsetup -v -y -c aes-xts-plain -s 512 luksFormat /dev/sdb1
# cryptsetup luksOpen /dev/sdb1 Respaldos
# ls -l /dev/mapper/Respaldos
# cryptsetup -v status Respaldos
```

Formatear partición

```
# mkfs.ext4 /dev/mapper/Respaldos
# mkdir -p /opt/Respaldos_crypt
# mount /dev/mapper/Respaldos /opt/Respaldos_crypt
# df -Th | grep crypt
# cd /opt/Respaldos_crypt
```

Montar y activar sistema de archivos

```
# cryptsetup luksOpen /dev/sdc1 Respaldos
# mkdir -p /opt/Respaldos_crypt
# mount /dev/mapper/Respaldos /opt/Respaldos_crypt
# df -Th | grep crypt
# cd /opt/Respaldos_crypt
```

Desmontar y desactivar el sistema de archivos

```
# umount /opt/Respaldos_crypt
# cryptsetup luksClose Respaldos
```

El uso del dispositivo cifrado en la gran mayoría de las distribuciones de GNU/Linux que tengan instalado el paquete *cryptsetup* reconocerán el dispositivo al momento de ser introducido en el equipo de cómputo y solicitarán la clave de acceso. A partir de ese momento será posible usar el dispositivo como cualquier otro -sin cifrado- y todas las tareas de montaje/desmontaje serán transparentes para el usuario.

Podemos usar el comando fsck en sistemas basados en LUKS, mediante:

```
# umount /opt/Respaldos_crypt
# fsck -vy /dev/mapper/Respaldos
# mount /dev/mapper/Respaldos /opt/Respaldos_crypt
```

Para cambiar la contraseña para la partición cifrada, usamos:

```
# cryptsetup luksDump /dev/sdb1
# cryptsetup luksAddKey /dev/sdb1
```

se pueden configurar un máximo de 8 contraseñas para cada dispositivo
Remover o eliminar la contraseña:

```
# cryptsetup luksRemoveKey /dev/sdb1
```

tengamos en cuenta que debemos ingresar la contraseña guardada.

**Cifrado Basado en Archivos con gocryptfs** Podemos tratar de mantener nuestros datos fuera de miradas indiscretas usando gocryptfs en las máquina a las que tengamos acceso (incluso del usuario administrador *root)*[182]. Además, podemos respaldarlos y transportarlos manteniendo estos siempre cifrados y que sea casi transparente para nosotros el uso de dicho programa.

El programa *gocryptfs* utiliza cifrado basado en archivos que se implementa como un sistema de archivos *FUSE* que se puede montar. Cada archivo en *gocryptfs* se almacena como un archivo cifrado correspondiente en el disco. Los archivos cifrados se pueden almacenar en cualquier carpeta del disco, unidad *USB* o incluso dentro de una carpeta en la nube como *Google Drive* o *Dropbox*. Una ventaja del cifrado basado en archivos en comparación con el cifrado de disco es que los archivos cifrados de pueden sincronizar de manera eficiente utilizando herramientas como rsync. Además, el tamaño del sistema de archivos cifrado es dinámico y solo está limitado por el espacio disponible en disco.

Para instalarlo usamos:

```
# apt install gocryptfs
```

después, es necesario crear los directorios para guardar los datos cifrados (por ejemplo en el directorio ~/.cifrados) y los descifrados (por ejemplo en el directorio ~/descifrados), mediante:

```
$ mkdir -p ~/.cifrados ~/descifrados
```

Para inicializar la carpeta usamos:

```
$ gocryptfs -init ~/.cifrados
```

nos pedirá la clave de acceso y su confirmación. Se pueden crear tantas carpetas independientes con *gocyptfs* como se requieran y el cualquier parte del sistema de archivos al que tengamos acceso.

Ahora ya podemos montar el directorio mediante:

```
$ gocryptfs ~/.cifrados ~/descifrados
```

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

el sistema nos pedirá nuestra clave de acceso y de ser correcta nos permitirá hacer el montaje. De esta forma, la carpeta virtual *~/cifrados* mostrará nuestros datos descifrados, solo visibles para el usuario que monta la carpeta (no para root) y la carpeta *~/.cifrados*, contendrá nuestros datos de forma cifrada[183], esta carpeta puede ser copiada, respaldada y restaurada aún estando montada, manteniendo el anonimato de nuestros archivos.

Una vez que ya no sea requerida la carpeta, la desmontamos usando:

```
$ fusermount -u ~/descifrados
```

Es posible usar *gocryptfs* en modo inverso, primero debemos inicializar la carpeta usando:

```
$ gocryptfs -reverse -init ~/.descifrados
```

y usarla mediante:

```
$ gocryptfs -reverse ~/.descifrados ~/.cifrados
```

**Cifrar Archivos con GnuPG** es una herramienta en línea de comandos de seguridad en comunicaciones electrónicas en donde se utiliza criptografía de clave pública para que los usuarios puedan comunicarse de un modo seguro. En un sistema de claves públicas cada usuario posee un par de claves, compuesto por una clave privada y una clave pública. Cada usuario debe mantener su clave privada secreta; no debe ser revelada nunca. La clave pública se puede entregar a cualquier persona con la que el usuario desee comunicarse. GnuPG implementa un esquema algo más sofisticado en el que un usuario tiene un par de claves primario, y ninguno o más de un par de claves adicionales subordinadas. Los pares de claves primarios y subordinados se encuentran agrupados para facilitar la gestión de claves, y el grupo puede ser considerado como un sólo par de claves.

Dentro de las funciones de GnuPG se incluyen generar un par de claves, intercambiar y comprobar la autenticidad de claves, cifrar y descifrar documentos, etc. Para instalar el paquete GnuPG, usamos:

```
# apt install gnupg
```

[183] El nombre de los archivos tienen un límite, por ello hay que tenerlo en cuenta si se usan nombre largos en el sistema de archivos (el límite aproximado es 255 caracteres), pero no importa el número de archivos o carpetas almacenadas internamente.

La opción de la línea de comandos *–gen-key* se usa para generar un nuevo par de claves primario, mediante:

$ gpg –gen-key

también podemos pedir –full-generate-key, mediante:

$ gpg –full-generate-key

GnuPG es capaz de crear varios tipos diferentes de pares de claves, pero debe existir una clave primaria capaz de generar firmas. Al momento de ejecutar el programa, este pide[184] nombre, correo del usuario y contraseña[185], para después generar la clave.

Para poder comunicarse con otros, el usuario debe intercambiar las claves públicas. Para obtener una lista de las claves en el fichero («anillo») de claves públicas, se puede usar la opción de la línea de comandos *–list-keys*, mediante:

$ gpg –list-keys

también podemos usar:

$ gpg –list-secret-keys
$ gpg –list-public-keys

Para poder enviar una clave pública a un interlocutor, antes hay que exportarla. Para ello se usará la opción de la línea de comandos *–export*. Es necesario un argumento adicional para poder identificar la clave pública que se va a exportar, por ejemplo:

$ gpg –output antonio.gpg –export ant@na.mx

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

[185] Cuanto más larga sea la contraseña, más segura será contra ataques de «fuerza bruta». No hay límite para la longitud de una contraseña, y esta debe ser escogida con sumo cuidado. Desde un punto de vista de seguridad, la contraseña que desbloquea la clave privada es uno de los puntos más débiles en GnuPG (y en otros sistemas de cifrado de clave pública), ya que es la única protección que tiene el usuario si alguien se apodera de su clave privada. Para una contraseña lo ideal es que no se usen palabras de un diccionario, y que se mezclen mayúsculas y minúsculas, dígitos, y otros caracteres. Una buena contraseña es crucial para el uso seguro de GnuPG.

La clave se exporta en formato binario, y esto puede no ser conveniente cuando se envía la clave por correo electrónico o se publica en una página Web. Por tanto, GnuPG ofrece una opción de la línea de comandos *–armor* que fuerza que la salida de la orden sea generada en formato *armadura-ASCII*, parecido a los documentos codificados con *UUEncode*. Por regla general, cualquier salida de una orden de GnuPG, v.g. claves, documentos cifrados y firmas, pueden ir en formato *armadura-ASCII* añadiendo a la orden la opción *–armor*, por ejemplo:

$ gpg –armor –output antonio.gpg –export ant@na.mx

Cada clave pública y privada tiene un papel específico en el cifrado y descifrado de documentos. Se puede pensar en una clave pública como en una caja fuerte de seguridad. Cuando un remitente cifra un documento usando una clave pública, ese documento se pone en la caja fuerte, la caja se cierra, y el bloqueo de la combinación de esta se gira varias veces. La parte correspondiente a la clave privada, esto es, el destinatario, es la combinación que puede volver a abrir la caja y retirar el documento. Dicho de otro modo, sólo la persona que posee la clave privada puede recuperar un documento cifrado usando la clave pública asociada al cifrado.

Con este modelo mental se ha mostrado el procedimiento de cifrar y descifrar documentos de un modo muy simple. Si el usuario quisiera cifrar un mensaje para Javier, lo haría usando la clave pública de Javier, y lo descifraría con su propia clave privada. Si Javier quisiera enviar un mensaje al usuario, lo haría con la clave pública del usuario, y este lo descifraría con su propia clave privada.

**Cifrar un Archivo** para cifrar un archivo se usa la opción *–encrypt*. El usuario debe tener las claves públicas de los pretendidos destinatarios. El programa espera recibir como entrada el nombre del archivo que se desea cifrar o, si este se omite, una entrada estándar. El resultado cifrado se coloca en la salida estándar o donde se haya especificado mediante la opción *–output*. El archivo se comprime como medida adicional de seguridad, aparte de cifrarlo, por ejemplo:

$ gpg –output doc.gpg –encrypt –recipient ant@na.mx doc

La opción *–recipient* se usa una vez para cada destinatario, y lleva un argumento extra que específica la clave pública con la que sería cifrado el

archivo. El archivo cifrado sólo puede ser descifrado por alguien con una clave privada que complemente una de las claves públicas de los destinatarios. El usuario, en este caso el remitente, no podría descifrar un archivo cifrado por sí mismo a menos que haya incluido su propia clave pública en la lista de destinatarios.

**Descifrar un Archivo** para descifrar un archivo se usa la opción –*decrypt*. Para ello es necesario poseer la clave privada para la que el archivo ha sido cifrado. De igual modo que en el proceso de cifrado, el archivo a descifrar es la entrada, y el resultado descifrado la salida, por ejemplo:

```
$ gpg –output doc –decrypt doc.gpg
```

**Cifrar y Descifrar Usando Cifrado Simétrico** también es posible cifrar archivos sin usar criptografía de clave pública. En su lugar, se puede usar sólo una clave de cifrado simétrico para cifrar el archivo. La clave que se usa para el cifrado simétrico deriva de la contraseña dada en el momento de cifrar el documento, y por razones de seguridad, no debe ser la misma contraseña que se esta usando para proteger la clave privada. El cifrado simétrico es útil para asegurar archivos cuando no sea necesario dar la contraseña a otros. Un archivo puede ser cifrado con una clave simétrica usando la opción –*symmetric*, por ejemplo:

```
$ gpg –symmetric doc
```

o

```
$ gpg -c doc
```

y podemos descifrar usando:

```
$ gpg doc.gpg
```

**Otras Opciones para Cifrar y Descifrar**

**ccrypt** es otra herramienta para cifrar y descifrar archivos basada en el cifrado de bloque Rijndael, Se cree que *ccrypt* (basada en el cifrado de bloque *Rijndael*) proporciona una seguridad fuerte. Tengamos en cuenta que *ccrypt* elimina el archivo original (cifra el archivo en su lugar), no cambia significativamente el tamaño del archivo y no altera la fecha/hora del archivo para reflejar la hora en que se realizó el cifrado.

```
$ ccrypt -e archivo
```

para descifrar usamos:

```
$ ccrypt -d archivo.cpt
```

**mcrypt** el comando permite cifrar y descifrar archivos, cuando cifra deja el archivo original intacto y cambia los permisos del archivo para que el archivo cifrado solo proporcione permisos de acceso de lectura y escritura al propietario del archivo. Ofrece muchas opciones con respecto a los algoritmos de cifrado y también ofrece opciones para comprimir los archivos antes del cifrado (opciones -z comprime GZip y -p comprime Bz2). Puede funcionar con varios archivos, pero los cifra por separado, ejemplo:

```
$ mcrypt archivo
```

para descifrar usamos:

```
$ mcrypt -d archivo.nc
```

para enumerar los algoritmos de cifrado disponibles, usamos:

$ *mycrypt –list*,

El comando *mcrypt* parece usar *Rijndael-128* como su algoritmo de cifrado predeterminado.

**age** este comando permite cifrar usando una clave pública o frase, no viene instalado por omisión en el sistema, pero lo podemos instalar usando:

```
# apt install age
```

Para generar una clave pública en el archivo llave.txt usamos:

```
$ age-keygen -o llave.txt
```

Para cifrar un archivo *archivo.tar* con clave pública usamos:

```
$ tar cvz archivos | age -r llave.txt > archivo.tar.gz.age
```

para descifrarlo usamos:

```
$ age –decrypt -i llave.txt > archivo.tar.gz
```

que nos retornará a nuestro archivo original.
Para cifrar un archivo *archivo.tar* con frase usamos:

```
$ age –passphrase –output archivo.tar.gz.age archivo.tar.gz
```

para descifrarlo usamos:

```
$ age –decrypt –output archivo.tar.gz archivo.tar.gz.age
```

que nos retornará a nuestro archivo original.

**Generar Contraseñas** Para generar contraseñas disponemos de múltiples opciones, algunas de ellas son:

- Podemos usar GnuPG, para instalarlo usamos:

  ```
  # apt install gnupg
  ```

  para generar una contraseña de 32 caracteres, usamos:

  ```
  $ gpg –gen-random –armor 1 32
  ```

- Podemos usar OpenSSL, para instalarlo usamos:

```
# apt install openssl
```

para generar una contraseña de 32 caracteres mediante:

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

- Podemos usar APG, para instalarlo usamos:

```
# apt install apg
```

para generar una contraseña de mínimo 31 y máximo 32 caracteres y genere 4 claves:

```
$ apg -n 4 -m 31 -x 32 -a 1
```

- Podemos usar PWGEN, para instalarlo usamos:

```
# apt install pwgen
```

para generar una contraseña de 32 caracteres:

```
$ pwgen -ys 32 1
```

- Podemos usar Makepasswd, para instalarlo usamos:

```
# apt install makepasswd
```

para generar una contraseña de 32 caracteres:

```
$ makepasswd –char 32
```

**Cifrado Avanzado** En los procesadores modernos es común encontar el motor Intel/AMD Advanced Encryption Standard (AES) o New Instructions (AES-NI) que permite el cifrado y descifrado por Hardware de alta velocidad para el cifrado de disco completo, OpenSSL, ssh, VPN, Linux / Unix / OSX y más. ¿Cómo verifico la compatibilidad con Intel/AMD AES-NI en mi sistema basado en Linux, incluido OpenSSL?

El conjunto de instrucciones del estándar de cifrado avanzado y las nuevas instrucciones del estándar de cifrado avanzado permiten que Intel/AMD específicas y otras CPU realicen un cifrado y descifrado de hardware extremadamente rápido.

Tengamos en cuenta que la compatibilidad con AES-NI se habilita automáticamente si el procesador detectado lo soporta. Para obtener una lista de procesadores que admiten el motor AES-NI, consulte el sitio y la documentación del procesador. El AES-NI es una extensión de la arquitectura del conjunto de instrucciones x86 para microprocesadores de Intel y AMD. Aumenta la velocidad de las aplicaciones que realizan el cifrado y el descifrado mediante AES.

Uno puede descubrir que el procesador tiene el conjunto de instrucciones AES / AES-NI usando el comando:

```
$ lscpu
```

o

```
$ grep -o aes / proc / cpuinfo
```

o

```
$ grep -m1 -o aes / proc / cpuinfo
```

o

```
# cpuid | grep -i aes | sort | uniq
```

En cualquiera de los casos si aparece la bandera *aes* el conjunto de instrucciones AES / AES-NI está presente y activo.

También podemos usar el siguiente comando para verificar la compatibilidad con AES-NI en nuestro procesador:

```
$ sort -u /proc/crypto | grep module
```

Ahora que hemos verificado la compatibilidad, es hora de probarla, para ello usamos:

```
$ openssl engine
```

ejemplo de una salida que soporta AES:

```
(padlock) VIA PadLock (no-RNG, no-ACE)
(dynamic) Dynamic engine loading support
```

ejemplo de una salida que soporta AES-NI:

(aesni) Intel AES-NI engine
(dynamic) Dynamic engine loading support

y podemos probar su desempeño entre un equipo que soporte el cifrado:

$ dd if=/dev/zero count=1000 bs=1M | ssh -l usuario -c aes128-cbc maquina "cat >/dev/null"

1000+0 records in

1000+0 records out

1048576000 bytes (1.0 GB) copied, 10.6691 s, 98.3 MB/s

y uno que no la soporte:

$ dd if=/dev/zero count=1000 bs=1M | ssh -l usuario -c aes128-cbc maquina "cat >/dev/null"

1000+0 records in

1000+0 records out

1048576000 bytes (1.0 GB) copied, 31.6675 s, 33.1 MB/s

**¿Cómo puedo comparar el rendimiento de OpenSSL?** podemos correr los comandos:

$ openssl speed

o

$ openssl speed aes-128-cbc

Para la última versión de OpenSSL, podemos probar los dos comandos, el segundo comando debería tener números más altos que el primero gracias a EntropyZer0:

$ openssl speed aes-256-cbc
$ openssl speed -evp aes-256-cbc

## 9.6 AntiVurus

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

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

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

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

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

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

```
# apt install lha unrar
```

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

```
# freshclam
```

y podemos revisar la ruta que deseemos:

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

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

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

y podemos efectuar la prueba, mediante:

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

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

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

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

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

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

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

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

También se puede probar el demonio *clamdscan*:

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

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

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

## 9.7 Programando en Bash

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

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

Sin embargo, estos lenguajes interpretados también tienen sus desventajas:

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

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

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

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

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

La mayoría de los *Shell Scripts*[186] (guiones de intérprete de órdenes) Bourne pueden ejecutarse por Bash sin ningún cambio, con la excepción de aquellos Scripts del intérprete de órdenes o consola Bourne que hacen referencia a variables especiales de Bourne o que utilizan una orden interna de Bourne. La sintaxis de órdenes de Bash incluye ideas tomadas desde los intérpretes Korn Shell (ksh) y C Shell (csh), como la edición de la línea de órdenes, el historial de órdenes, la pila de directorios, las variables *$RANDOM* y *$PPID*, y la sintaxis de substitución de órdenes *POSIX*: $(...). Cuando se utiliza como un intérprete de órdenes interactivo, Bash proporciona autocompletado de nombres de programas, nombres de archivos, nombres de variables, etc., cuando el usuario pulsa la tecla *TAB*.

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

```
orden >& archivo
```

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

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

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

[186] Para generar un archivo de BASH o Script, usemos cualquier editor de texto, por ejemplo *nano miScript*. En el, la primera línea se acostumbra poner el Shebang que se utiliza para indicar al sistema qué intérprete de comandos se debe llamar:

#!/bin/bash

ya creado el Script, es necesario hacerlo ejecutable, para ello usamos:

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

o

```
$ chmod 755 miScript
```

y lo ejecutamos en la línea de comandos mediante:

```
$ ./miScript
```

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

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

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

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

entonces podemos ejecutarlo usando:

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

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

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

y podemos usar las siguientes condiciones:

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

por ejemplo:

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

o en una sola línea:

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

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

```
#!/bin/bash
if [ $1 -lt 13 ]; then
    echo "Eres un niño"
```

```
elif [ $1 -lt 18 ]; then
     echo "Eres un adolecente"
elif [ $1 -lt 65 ]; then
     echo "Eres un adulto"
else
     echo "Eres un adulto mayor"
fi
```

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

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

otro ejemplo:

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

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

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

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

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

ejemplo:

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

O usando una lista o rango, ejemplo

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

o

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

otro ejemplo:

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

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

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

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

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

o un ciclo infinito mediante:

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

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

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

ejemplo

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

o un ciclo infinito mediante:

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

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

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

ejemplo:

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

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

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

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

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

Ejemplo de funciones con parámetros

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

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

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

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

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

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

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

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

- Una gran limitación del intérprete Bourne es que no puede realizar cálculos con enteros sin lanzar un proceso externo. En cambio, un proceso Bash puede realizar cálculos con enteros[187] utilizando la orden ((...)) y la sintaxis de variables $[...] de la siguiente manera:

  asigna el valor entero 55 a la variable VAR.

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

VAR=55

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

((VAR = VAR + 1))

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

((++VAR))

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

((VAR++))

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

echo $[VAR * 22]

otra forma de realizar lo mismo:

echo $((VAR * 22))

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

  # apt install bc

  y lo podemos usar de esta forma:

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

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

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

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

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

podemos concatenar dos cadenas, mediante:

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

extraer una porción de la cadena:

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

extraer desde una posición hasta el final de la cadena

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

remplazar una subcadena de una cadena[188]:

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

borrar una subcadena de una cadena:

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

borrar todas las apariciones de la subcadena:

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

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

también podemos utilizar [] para especificar rangos de caracteres:

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

convertir a mayúsculas una cadena:

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

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

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

convertir a minúsculas una cadena:

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

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

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

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

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

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

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

- Manejo de ciclos con una secuencia numérica, ejemplo:

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

- Manejo de ciclos con *while*, ejemplo:

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

- Manejo de ciclos con *until*, ejemplo:

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

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

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

  Ejemplo de esto, probemos para código cero:

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

para código 127:

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

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

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

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

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

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

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

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

Para conocer las variables del medio ambiente podemos usar:

```
$ printenv
```

o

```
$ env
```

o conocer el valor de solo una, usamos:

```
$ printenv PATH
```
[189]

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

```
$ set
```

Para asignar un valor a una variable usamos:

```
$ var=valor
```

o

```
$ export var=valor
```

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

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

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

y si deseamos limpiar una variable usamos:

```
$ unset var
```

Podemos también definir variables de solo lectura, usando:

```
$ readonly var=valor
```

**Observación 4** *Un proceso Bash no puede realizar cálculos en punto flotante sin lanzar un proceso externo (como bc). Las únicas Shell Unix capaces de esto son Korn Shell (versión de 1993) y zsh (a partir de la versión 4.0).*

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

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

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

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

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

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

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

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

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

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

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

```
# apt install shc
```

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

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

y lo ejecutamos usando:

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

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

**Algunos Consejos para Programar en Bash** Un Bash Script es un archivo en el que se codifican múltiples comandos de Shell para realizar una tarea en particular, algunas sanas costumbres para escribir un Script, principalmente para mejorar la eficiencia y hacerlo más legible son:

**Comentar el código** definitivamente algo básico, pero que muchos olvidan y que es siempre de mucha utilidad, sea para uno mismo, o para otros que quieran revisar o modificar el Script, es sin duda determinante para lograr un código limpio y que logre explicar varias partes de códigos complejos. También es eficaz cuando se trabaja en grupo en un gran proyecto, ya que ayuda a comprender qué hace realmente la función o el método.

**Uso de Funciones** una función es un conjunto de comandos agrupados para realizar una tarea específica que ayuda a modular el flujo de trabajo, eliminando la repetición del código. Hace que el código sea limpio y más legible, así como fácil de mantener:

```
#!/bin/bash
function check_root() {
    echo "la función ha sido llamada";
}
```

**Uso de comillas dobles** ayudará a eliminar el englobamiento innecesario, así como la división de palabras, incluidos los espacios en blanco, cuando los valores de las variables contienen un carácter separador o un espacio en blanco.

**Terminar ejecución por error** a veces, al ejecutar un Script, puede haber algún error de ejecución. Incluso si un comando no se ejecuta, la secuencia de comandos puede continuar corriendo y afectar a los demás comandos del Script. Para evitar más errores lógicos, debemos incluir *set -o errexit* o *set -e* para finalizar el comando en caso de error:

```
#!/bin/bash
# Terminar el Script si hay error
set -o errexit
```

**Declarando Variables** según su tipo de datos y usos. Cuando la variable no es declarada, es posible que Bash no pueda ejecutar el comando relacionado. Las variables se pueden declarar global o localmente en el Script. Por ejemplo:

```
#!/bin/bash
# Declaración de variable, como notas es de formato vari-
able=valor
declare -r -i x=30
function my_variable(){
    local -r name = ${HOME}
}
```

**El uso de llaves** vincula las variables con llaves mientras usa la concatenación de variables con cadenas para evitar usos innecesarios de variables. Esto también ayuda a identificar fácilmente la variable mientras se usa en una cadena. Por ejemplo:

```
#!/bin/bash
# Termina el Script si hay error
set -o errexit
# Variable personalizada
data= "${USER}_data is being used"
```

**Sustitución del comando** al asignar la salida del comando a la variable, Bash utiliza la función de sustitución de comandos. Necesitamos usar $() en lugar de comillas inversas para asignar la salida a las variables como se recomienda:

```
#!/bin/bash
# Termina el Script si hay error
set -o errexit
#Muestra la Fecha
date_now = ${date}
```

**Nomenclatura de variables** en nuestro sistema, todas las variables de entorno se nombran con letras mayúsculas. Entonces, cuando declaramos una variable local, debemos declararla usando letras minúsculas para evitar conflictos entre el entorno y el nombre de la variable local:

```
#!/bin/bash
# Termina el Script si hay error
set -o errexit
user_var = "$HOME es tu login correcto."
```

**Declarar Variables Estáticas** si tienes datos estáticos que permanecen sin cambios durante todo el Script, puedes asignar el valor a una variable estática cuyo valor no se puede modificar. Puedes declarar la variable estática usando el comando de solo lectura:

```
#!/bin/bash
# Probando configuración nginx
readonly test_conf_path = "/etc/nginx/conf.d/test.conf"
```

**Depuración** la depuración es la parte esencial para identificar un problema. Podemos verificar el error de sintaxis del Script, por lo que para verificarlo necesitamos ejecutar el Script Bash con *-n* usando el comando Bash:

```
$ bash -n script_name
```

además, podemos habilitar y ejecutar el Script en modo de depuración usando el siguiente comando.

```
$ bash -x script_name
```

## 9.8 Algunos Ejemplos en Bash

Para generar un archivo de Bash o Script, usemos cualquier editor de texto, por ejemplo:

```
$ nano miScript
```

en el, la primera línea se acostumbra poner:

```
#!/bin/bash
```

ya creado el Script, es necesario hacerlo ejecutable, para ello usamos:

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

o

```
$ chmod 755 miScript
```

y lo ejecutamos en la línea de comandos mediante:

```
$ ./miScript
```

si necesitamos conocer la línea que se esta ejecutando de un Script podemos usar:

```
$ bash -x ./miScript
```

Estos y otros ejemplos de Bash se pueden descargar de:
Bash

**Conocer el uso de memoria de los procesos** Podemos usar el comando *top* que nos informa en tiempo real el consumo de RAM o algunas de estas opciones:

```
$ ps aux –sort -rss | head
$ top -c -b -o +%MEM | head -n 20
```

**Uso de comillas dobles y sencillas** este pequeño ejemplo nos muestra visualmente el uso de comillas sencillas y dobles:

```
$ a=7; echo $a; echo "$a"; echo '$a'; echo "'$a'"; echo '"$a"'
```

**Convertir imágenes usando xargs** para convertir imagenes .png a .jpg, usaremos el comando *ls* para obtener los nombres de archivo que pasaremos a *xargs* que nos permite llamar a *conver* mediante *bash* para hacer la conversión, mediante:

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

y para convertir imagenes de .jpg a png, usamos:

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

**Números pseudoaleatorios** se pueden generar números pseudoaleatorios entre 0 y 32767, por ejemplo generamos números entre 0 y 100 mediante:

```
$ echo $((RANDOM % 100+1))
```

**Longitud de una cadena** disponemos de varias formas de conocer la longitud de una cadena, por ejemplo:

```
#!/bin/bash
cadena="Esta es una cadena"
tam1=${#cadena}
tam2=$(expr length "$cadena")
tam3=$(echo $cadena | awk '{print length}')
tam4=$(echo -n $cadena | wc -m)
echo $tam1 $tam2 $tam3 $tam4
```

**Buscar en una cadena una subcadena** podemos buscar en una cadena una subcadena y preguntar sobre ella, mediante:

```
#!/bin/bash
url="http://www.mmc.geofisica.unam.mx/acl/Textos/"
if [[ $url =~ .*acl*. ]]
then
    echo "Se encontro la cadena acl en ${url}."
else
    echo "No se encontro."
fi
```

**Emoticonos** en la consola podemos también generar emoticonos, muestra de ello es:

```
$ printf $(printf '\\U%x\\x20' {{128512..128591},{128640..128725}})
```

**Formatear números** es posible dar formato a números mediante el uso de *numfmt*, por ejemplo:

```
$ numfmt –grouping 123456789
```

acepta sufijos como 1K, 1Ki, 1M, 1Mi, 1G, 1Gi, por ejemplo:

```
$ numfmt –from=iec-i 2Gi
$ numfmt –to=iec 1000000
```

y lo podemos usar por ejemplo en un ls:

```
$ ls -l | numfmt –header –field 5 –to=iec
```

También podemos agregar ceros a la izquierda al visualizar una cantidad numérica, por ejemplo:

```
for((i=1; i<=10; ++i)): do
    printf "%03d\n" $i
done
```

en caso de no querer mostrar directamente el resultado, podemos asignarlo a una variable usando el parámetro *-v* del comando *printf*:

```
for((i=1; i<=10; ++i)): do
    printf -v j "%03d\n" $i
    echo $j
done
```

**Realizar lista de potencias de 3** podemos usar el Bash para visualizar las primeras 19 potencias de 3, usando:

```
$ printf "%s\n" $[3**{1..19}] | nl
```

o

```
$ printf "%s\n" 3^{1..19} | bc | nl
```

o

```
$ echo -n 3^{1..19} | xargs -d' ' -I@ sh -c 'printf "%6s =
%48s\n" @ $(echo "@" | bc)'
```

o

```
$ for i in {1..19}; do printf "%0.2d %s\n" $i $[3**$i]; done
```

**Manipular ruta de un archivo** dada la ruta de un archivo:

```
archivo="/usr/share/icons/64x64/application-wireshark-doc.png"
```

para obtener sólo la ruta usamos:

```
echo "${archivo%/*}"
```

para obtener sólo el nombre de archivo usamos:

```
echo "${archivo##*/}"
```

para eliminar la extensión del nombre del archivo usamos:

```
nombre="${archivo##*/}"
echo ${nombre%.*}
```

para extraer la extensión del nombre del archivo usamos:

```
echo ${achivo##*.}
```

También podemos usar al comando basename, por ejemplo:

```
$ basename /etc/passwd
$ basename /etc/sysctl.conf .conf
```

de esta forma podemos renombar todos los archivos con extensión .jpeg a jpg, mediante:

```
for file in *.jpeg; do
    mv – "$file" "$(basename $file .jpeg).jpg"
done
```

en caso de necesitar la trayectoria pero no el nombre de archivo, podemos usar:

```
$ dirname /var/spool/mail/antonio.txt??
```

y podemos combinar a *basename* y *dirname*, por ejemplo:

```
Trayectoria="/home/antonio/data/fichero.txt"
tray=$(dirname "$Trayectoria")
nomb=$(basename "$Trayectoria")
echo $tray
echo $nomb
```

**Manipular nombre de archivos** Cambiar los espacios por subrayado:

```
$ for FILE in * ; do NEWFILE=‘echo $FILE | sed ’s/ /_/g’‘
; mv "$FILE" $NEWFILE ; done
```

Añadir un sufijo .txt a cada archivo:

```
$ for FILE in * ; do mv $FILE $FILE.txt ; done
```

Quitar el sufijo .txt a cada archivo:

```
$ for FILE in *.txt ; do NEWFILE=‘echo $FILE | sed ’s/.txt$//’‘
; mv $FILE $NEWFILE ; done
```

Convertir el sufijo .TXT a .txt

```
$ for FILE in *.TXT ; do NEWFILE=‘echo $FILE | sed
’s/.TXT$/.txt/’‘ ; mv $FILE $NEWFILE ; done
```

Añadir un prefijo maths_ a cada archivo:

```
$ for FILE in * ; do mv $FILE maths_$FILE ; done
```

Quitar el prefijo maths_ a cada archivo_

```
$ for FILE in * ; do NEWFILE=‘echo $FILE | sed ’s/^maths_//’‘
; mv $FILE $NEWFILE ; done
```

Convertir nombre de mayúsculas a minúsculas a cada archivo:

```
$ for FILE in * ; do mv $FILE ‘echo $FILE | tr ’[A-Z]’ ’[a-z]’‘
; done
```

Convertir nombre de minúsculas a mayúsculas a cada archivo:

```
$ for FILE in * ; do mv $FILE ‘echo $FILE | tr ’[a-z]’ ’[A-Z]’‘
; done
```

Quiero que los espacios cambien a un guión, salvo cuando haya dos o más espacios consecutivos, en el que necesito un solo subrayado. Mientras estamos en ello, si un espacio está al lado del punto de un prefijo, vamos a quitarlo. También quiero hacer los sufijos minúsculas.

```
$ for FILE in * ; do NEWFILE=‘echo $FILE | sed -e ’s/.TXT$/.txt/’
-e ’s/[ ]*[ ]/_/g’ -e ’s/_[.]/./g’‘ ; mv "$FILE" $NEWFILE ; done
```

Dar a todos los archivos el mismo nombre, pero con un número diferente:

```
$ NUM=0 ; for FILE in * ; do NUM=‘expr $NUM + 1‘ ; mv
$FILE Archivo\($NUM\) ; done
```

Mantener el nombre original y separar el número secuencial con un subrayado:

```
$ NUM=0 ; for FILE in * ; do NUM=‘expr $NUM + 1‘ ; mv
$FILE ${FILE}_$NUM ; done
```

**Crear y remover archivos** podemos usar al comando *seq* para crear y borrar un grupo de archivos:

```
$ touch $(seq -f "Archivo%g" 4)
$ rm $(seq -f "Archivo%g" 4)
```

otra forma es:

```
$ for i in web{0..10} db{0..2} otro_{a,b}{1..4}; do touch $i;
done
$ for i in web{0..10} db{0..2} otro_{a,b}{1..4}; do rm $i;
done
```

**Manejo de parámetros** Si generamos un Script con nombre *test.sh*, entonces podemos controlar por ejemplo dos parámetros (*–alpha* o *-a* y *–config* o *-c*) mediante:

```
#!/bin/bash
while [ True ]; do
if [ "$1" = "–alpha" -o "$1" = "-a" ]; then
    ALPHA=1
    shift 1
```

```
elif [ "$1" = "–config" -o "$1" = "-c" ]; then
    CONFIG=$2
    shift 2
else
    break
fi
done
echo $ALPHA
echo $CONFIG
ARG=( "${@}" )
for i in ${ARG[@]}; do
    echo $i
done
```

y lo podemos usar mediante:

```
$ ./test.sh –config my.conf foo bar
$ ./test.sh -a –config my.conf baz
```

**Cree un árbol de carpetas** usando el comando *mkdir* podemos crear un árbol de carpetas, con carpetas A - Z en el primer nivel, luego 0 a 100 en cada uno de ellos. Y todas las demás letras que comienzan en a en las carpetas pares y todas las demás que comienzan con b en impares

```
$ mkdir -p tree/{A..Z}/{0..100..2}/{a..z..2} tree/{A..Z}/{1..100..2}/{b..z..2}
```

**Busca archivos ASCII y extrae direcciones IP de ellos** podemos buscar en la trayectoria actual a todos los archivos ASCII que contienen direcciones IP validas y pedir que nos muestre estas, usando:

```
$ find . -type f -exec grep -Iq . {} \; -exec grep -oE "(25[0-5]|2[0-4]\
[0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25\
[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9]\
[0-9]?)" {} /dev/null \;
```

**Crear directorios a partir del contenido de un archivo** si tenemos un archivo con distintos nombres de directorios que necesitamos crear, por ejemplo:

servidores

comando/Linux

comando/Bash

editores/vin

editores/nano

y están en el archivo *directorios.txt*, podemos usar para crearlos:

```
$ xargs -I{} mkdir -p "{}" < directorios.txt
```

**Crear archivos temporales** cuando se trabaja en Bash se requiere construir archivos, muchos de ellos temporales, para ellos usamos:

```
$ tmp_file=$(mktemp)
```

si queremos ver el archivo creado podemos usar:

```
$ tmp_file=$(mktemp) ; echo $tmp_file
```

y para borrarlo podemos usar:

```
$ rm ${temp_file}
```

Otras opciones son:

```
$ tmp_file=/tmp/foobar.$$
$ tmp_file=/tmp/foobar.$RANDOM
$ tmp_file=‘date ’+%Y-%m-%d-%H-%M’‘-‘uuidgen -t | head
-c 5‘
```

**Partir un archivo de texto en múltiples archivos** el comando *split* nos permite partir un archivo de texto en múltiples archivos de un tamaño indicado, por ejemplo en 50 líneas:

```
$ split –lines=50 archivo.txt
```

**Formateador de texto simple** el comando *fmt* nos permite formatear la entrada que se le de, básicamente rellena y junta líneas de texto mientras que mantiene líneas en blanco y sangrías, por ejemplo reformateamos el texto del archivo para ajustarlo a una columna de 50 caracteres de ancho:

```
$ fmt -w 50 archivo.txt
```

**Crear directorios temporales** cuando se trabaja en Bash se requiere construir directorios donde contener los archivos de nuestro programa, muchos de ellos temporales, para ellos usamos:

```
$ tmp_dir=$(mktemp -d)
```

si queremos ver el directorio creado podemos usar:

```
$ tmp_dir=$(mktemp -d) ; echo $tmp_dir
```

y para borrarlo podemos usar:

```
$ rmdir ${temp_dir}
```

o

```
$ rm -R ${temp_dir}
```

Otras opciones son:

```
$ tmp_dir=$(mktemp -d -t ci-XXXXXXXXXX)
$ tmp_dir=$(mktemp -d -t ci-$(date +%Y-%m-%d-%H-%M-
%S)-XXXXXXXXXX)
$ tmp_dir=$(mktemp -d -t ci-XXXXXXXXXX –tmpdir=/home/antonio/tmp)
```

**Leer un nombre de archivo dentro del Script** se solicita al usuario un nombre de archivo mediante el comando *read*, para por ejemplo, contar el número de líneas del archivo:

```
#!/bin/bash
echo -n "Introduzca un nombre de archivo: "
read archivo
nlineas=$(wc -l < $archivo)
echo "Tiene $nlineas lineas el archivo $archivo"
```

**Correr el Scrypt solo por el usuario root** hay veces que debemos tener certeza de que solo el usuario root deba correr el contenido del Script, para ello podemos usar:

```
#!/bin/bash
if [[ "$(whoami)" != root ]]; then
    echo "Solo el usuario root puede correr este script."
    exit 1
fi
echo "Haciendo algo..."
exit 0
```

**Pasar argumentos al Script** podemos pasar argumentos al Script en la línea de comandos al momento de su ejecución, si llamamos a este archivo como *mi-Script*, con este contenido:

```
#!/bin/bash
echo "Nombre del Script: $0"
echo "Numero total de argumentos: $#"
echo "Valor de todos los argumentos: $@"
nlineas=$(wc -l < $1)
echo "Tiene $nlineas lineas el archivo $1"
```

entonces al ejecutar usando:

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

nos permitirá contar el número de líneas del archivo *texto.tx*t

**Agregar pausa a un Script** podemos agregar a nuestro Script que haga una pausa antes de continuar su ejecución, algunas opciones son:

```
$ read -p "Presione [Enter] para continuar ..."
$ read -t 5 -p "Esperaremos 5 segundos para continuar"
$ sleep 5 && comando
```

**Hacer cálculos de punto flotante** podemos hacer conversión de grados Celsius a Fahrenheit, mediante en programa ConvierteC2F:

```
#!/bin/bash
# bc se le indica que use dos decimales: scale=2
F=$(echo "scale=2; $1 * (9/5) + 32" | bc -l)
echo "$1 grados Celsius es igual a $F grados Fahrenheit."
```

y para usarlo:

```
$ ./ConvierteC2F 25
```

**Leer un archivo línea por línea** en muchos ejemplos de Bash es necesario leer un archivo línea por línea, aquí mostramos algunas opciones:

```
$ while read line; do echo -e "$line\n"; done < archivo.txt
$ cat emails.txt |while read line; do echo "$line"; done
```

**Leer un archivo en formato .CSV** Hay muchas aplicaciones que generan archivos de texto con valores separados por comas (*.CSV*), los cuales son fácilmente leíble en BASH, por ejemplo un archivo *a.csv* con dos columnas, puede ser leído usando:

```
#!/bin/bash
while IFS=, read -r C1 C2
do
    echo "$C1 y C2"
done
```

si lo grabamos con nombre leeCSV.sh, entonces lo podemos ejecutar mediante:

```
$ leeCSV.sh < a.csv
```

**Uso del comando column** permite mostrar la salida de algún comando separado por columnas:

```
$ mount | column -t
```

**Poner un texto al revés** el comando *rev* permite que cualquier texto que se le pase lo ponga al revés, ejemplo:

```
$ rev <<< "esta es una cadena"
```

**Encontrar los factores primos de un número** el comando factor permite encontrar los factores primos de un número, ejemplo:

```
$ factor <<< 27832878
```

**Conversión de base** convierte el número hexadecimal 2F al decimal 47, usando:

```
$ echo $((0x2F))
```

o mediante:

```
$ echo $((16#2F))
```

**Crear un archivo compactado por cada archivo indicado**

```
$ for f in *.txt ; do tar cvf "$f.tar.bz2" "$f"; done
```

**Crear un archivo tar por cada directorio hijo de nuestra actual posición:**

```
$ find . -maxdepth 1 -mindepth 1 -type d -exec tar cvf \
{}.tar {} \;
```

**Renombrar archivos** si bien existe el comando *rename* para renombrar archivos **.bak* a **.txt*, usamos:

```
$ rename .bak .txt *.bak
```

también podemos hacer un Script para renombrar todos los archivos **.bak* a **.txt*, ejemplos:

```
$ for j in *.bak; do mv – "$j" "${j%.bak}.txt"; done
$ for j in *.bak; do mv -v – "$j" "${j%.bak}.txt"; done
```

**Encontrar archivos y directorios grandes** para encontrar, digamos los diez directorios más grandes en nuestra actual trayectoria, usamos:

```
$ du -h | sort -hr | head -n 10
```

para encontrar, digamos los diez archivos más grandes en nuestra actual trayectoria, usamos:

```
$ du -ah | sort -hr | head -n 10
```

**Buscar un directorio y borrar su contenido** un opción para buscar un directorio y borrar su contenido es:

```
$ find . -type d -iname nombre -delete
```

este comando encontrará todos los directorios y tratará de borrarlos, pero mandará error si el directorio no esta vació. Para solucionarlo podemos usar:

```
$ find . -type d -name "nombre" -exec rm -rf {} +
```

otras opciones son:

```
$ find . -type d -name "nombre" -exec rm -rf \;
$ find . -type d -name "nombre" -exec rm -rf "{}" \;
$ find . -type d -name "nombre*" -print0 | xargs -0 -I {}
/bin/rm -rf "{}"
```

podemos indicarle que realice la búsqueda en hasta 4 niveles de profundidad en el presente directorio:

```
$ find . -type d -name "nombre" -depth +4 -print0 -exec rm
-rf {} +
```

Algunas veces es necesario solo borrar los directorios vacíos, una forma de hacerlo es:

```
$ find . -type d -empty -print0 | xargs -0 -I {} /bin/rm -rf
"{}"
```

**Conocer la velocidad de grabación del disco** usando el comando *dd* y generando un archivo temporal, podemos conocer la velocidad de grabación de nuestro disco, mediante:

```
$ dd if=/dev/zero of=/tmp/salida.img bs=5G count=1 oflag=dsync;
rm -rf /tmp/salida.img
```

y para medir la latencia usamos:

```
$ dd if=/dev/zero of=/tmp/salida.img bs=512 count=1000
oflag=dsync; rm -rf /tmp/salida.img
```

**Aprender algo nuevo** nos da el manual de un comando diferente instalado en nuestro sistema en cada llamada:

```
$ man $(ls /bin | shuf | head -1)
```

**Comandos usados y cuantas veces** si deseamos conocer cuales son los comandos usados y cuantas veces los hemos usado, escribimos:

```
$ history | awk '{print $2}' | sort | uniq -c | sort -nr
```

**Generando contraseñas** si bien los programas GPG y OpenSSL nos permiten generar contraseñas, podemos usar en Bash para ello, en todos los ejemplos generamos contraseñas de 32 caracteres:

```
$ date +%s | sha256sum | base64 | head -c 32 ; echo
$ < /dev/urandom tr -dc _A-Z-a-z-0-9 | head -c${1:-32};echo;
$ tr -cd '[:alnum:]' < /dev/urandom | fold -w32 | head -n1
$ strings /dev/urandom | grep -o '[[:alnum:]]' | head -n 33 | tr
-d '\n'; echo
$ < /dev/urandom tr -dc _A-Z-a-z-0-9 | head -c32
$ dd if=/dev/urandom bs=1 count=32 2>/dev/null | base64
-w 0 | rev | cut -b 2- | rev
$ </dev/urandom tr -dc '12345!@#$%qwertQWERTasdfgAS-
DFGzxcvbZXCVB' | head -c32; echo ""
$ randpw(){ < /dev/urandom tr -dc _A-Z-a-z-0-9 | head -
c${1:-32};echo;} && randpw
```

**Visualiza la IP del equipo** imprime las direcciones de IPv4 asociada a las redes del equipo:

```
$ ip address | grep inet | grep -v inet6 | awk '{ print $2 }'
```

**Lanzar procesos después de cierto tiempo** algunas veces queremos que el sistema lance eventos después de cierto tiempo, supongamos queremos que nos avise que en 5 minutos -pueden ser segundos (s), minutos (m), horas (h) o días (d)- tenemos junta, usamos:

```
$ sleep 5m && echo "ir a la junta ... ahora"
```

pero podemos lanzar una aplicación, por ejemplo que toque música para avisarnos de la junta:

```
$ sleep 5m && vlc /home/usuario/alarma.mp3
```

**Buscar archivos duplicados y cambiarlos por ligas simbólicas duras** cuando se trabaja en proyectos es común terminar con múltiples copias de algunos archivos, pero podemos encontrar todos y remplazarlos por ligas simbólicas duras, para ello usaremos el programa hardlink, lo instalamos usando:

```
# apt install hardlink
```

podemos usar hardlink con *-tnv* para que nos diga los archivos que puede procesar:

```
$ hardlink -tnv temp
```

si nos satisface, ahora hacemos el cambio de los archivos duplicados por ligas simbólicas duras:

```
$ hardlink -tv temp
```

Para conocer el número de enlaces que tiene un determinado archivo, podemos usar:

```
$ ls -l archivo
```

o

```
$ stat archivo
```

**Descargar archivos de la red** una actividad común es la descarga de archivos de red (*.iso*, *.deb*, etc.) para ello hay varios programas como: *wget, uget, curl, lftp, woof, httpie*, entre otros, su uso básico es el siguiente:

```
$ curl http://132.248.182.159/acl/hcl/HerramientasComputacionalesEnLinux.pdf
```

descargará este texto. Se pueden descargar archivos y/o directorios de los protocolos *HTTP, HTTPS, FTP, POP3, SMTP* y *Telnet*. Además es posible interactuar con APIs usando los métodos: *DELETE, GET, POST* y *PUT*

**Uso del comando awk** Visualiza solo las líneas que son de 65 caracteres o más:

```
$ cat archivo.txt | awk "length > 65"
```

Conocer todas las ligas y a quien apuntan desde nuestra actul trayectoria, usamos:

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

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

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

Nos indicará cuantos procesos tiene corriendo cada usuario:

```
$ ps -ef | awk '{print$1}' | sort | uniq -c | sort -nr
```

Nos muestra la primera y última línea de un archivo:

```
$ awk 'NR==1{print} END {print}' archivo.txt
```

o

```
$ awk 'NR==1;END {print}' archivo.txt
```

**Programa *Buscar*** para encontrar los archivos y directorios modificados en una determinada trayectoria [$1] en los últimos días [$2]:

```
#!/bin/bash
# Encuentra los archivos modificados en los ultimos $2 dias
if [ -z "$1" ]; then
    echo uso: $0 [directorio] [dias]
    exit
fi
find $1 -type f -mtime -$2 -exec ls -gGh –full-time '{}' \; | cut
-d ' ' -f 4,5,7
```

por ejemplo, para buscar desde /home los archivos modificados desde el último día, usamos:

```
$ ./Buscar /home 1
```

**Programa *Diferencia*** para encontrar la diferencia entre los archivos y directorios de dos trayectorias [$1] y [$2]:

```
#!/bin/bash
# Encuentra las diferencias entre dos directorios y subdirec-
torios
if [ -z "$1" ]; then
    echo uso: $0 [directorio1] [directorio2]
    exit
fi
diff <(cd $1 && find | sort) <(cd $2 && find | sort)
```

por ejemplo, buscar la diferencia de contenido entre /home/user/a1 y /home/user/b3, usamos:

```
$ ./Diferencia /home/user/a1 /home/user/b3
```

**Ejemplo de uso de rsync** ejemplo de uso de rsync en el cual el usuario podría indicar cual función ejecutar dentro del Script:

```
#!/bin/bash
sync_root(){
      spath="/srv/www/unam/https"
      echo "Running rsync..."
      rsync -ar $spath/* root@backup.unam.mx:$spath
}
case "$1" in
      *unam*) sync_root ;;
      *) echo "Error: El dominio no existe.";;
esac
```

**Intentar sincronizar con srync hasta lograrlo** al intentar sincronizar una carpeta usando *rsync*, puede que por cualquier razón no se logre completar la operación, por ello podemos usar el comando *until* para que siga intentando la sincronización por *rsync* a través de una conexión no confiable. Cuando *rsync* falla, se iniciará un reintento después de 60 segundos. Cuando se complete *rsync*, el ciclo terminará, para ello usamos:

```
$ until rsync -avy –delete-after /volume1/backup/ \
root@backup.unam.mx:.; do sleep 60; done
```

**Programa Respalda** para generar el respaldo de la trayectoria indicada [$2] con el nombre [$1] (véase 9.2):

```
#!/bin/bash
#Respalda el contenido dado $2 con el nombre $1
if [ -z "$1" ]; then
    echo uso: $0 [Archivo].tar [Archivo o Directorio]
    exit
fi
# Variables de trabajo
A=$1
```

```
B=$(date +%Y%m%d)
# Genera el archivo TAR
echo Generando el archivo TAR ...
shift 1
tar -zcvpf $A-$B.tar.gz $*
# Visualiza el nuevo contenido
/bin/ls -al –color=tty
```

por ejemplo, para respaldar el contenido de /home con el nombre respaldo, usamos:

```
$ ./Respalda respaldo /home
```

**Programa para un menú** este ejemplo muestra un menú del cual el usuario puede elegir entre las diferentes opciones, supongamos que lo llamamos menu y tiene el siguiente código:

```
#!/bin/bash
PS3='Elige tu comida favorita: '
comida=("Pizza" "Pho" "Tacos" "Salir")
select op in "${comida[@]}"; do
  case $op in
    "Pizza")
      echo "Se come en el mundo 1000 acres de $op cada dia"
      # opcional llamar a funcion o correr mas comandos
      break
      ;;
    "Pho")
      echo "$op es un tipo de sopa Vietnamita"
      # opcional llamar a funcion o correr mas comandos
      break
      ;;
    "Tacos")
      echo "Se comen al año 3 mil millones de $op cada año"
      # opcional llamar a funcion o correr mas comandos
      break
```

```
        ;;
      "Salir")
        echo "Usuario solicito salir"
        exit
        ;;
      *) echo "Invalida la opcion $REPLY";;
    esac
  done
```

y lo ejecutamos usando:

```
$ ./menu
```

**Jugar a viborita** podemos usar Bash para jugar, usando las flechas podemos mover la barra dentro de toda la terminal usando el código:

```
$ reset;clear;x=$[COLUMNS/2];y=$LINES;u=0;v=-1;while :;do
read -sr -t0.02 -n3 d; case "${d:2:1}" in A) v=-1;u=0;;B)v=1;u=0;;C)
v=0;u=1;;D) v=0;u=-1;;esac;s=$x;t=$y;x=$[x+u];y=$[y+v];printf
"\033[%s;%sH\033[46m \033[0m\033[%s;%sH\033[44m \033[0m\033[0;0H"
$y $x $t $s;sleep .01;done
```

una versión más completa es:

```
$ clear;x=$(($COLUMNS/2)); y=$LINES;unset spacesused[*];declare
-a spacesused;xdir=0;ydir=-1;while true ; do read -s -r -t0.02 -
n3 direction ; case "${direction:2:1}" in A) ydir=-1;xdir=0 ;;
B)ydir=1;xdir=0 ;; C) ydir=0;xdir=1 ;; D) ydir=0;xdir=-1 ;;
esac ; space=$(( $COLUMNS * $y + $x )) ; if [[ "${space-
sused[$space]}" == "1" || $x -gt $COLUMNS || $x -lt 0 || $y -gt
$LINES || $y -lt 0 ]]; then printf '\033[%d;%dHBOOM!\nEND
OF LINE.\n' $y $x; break ; else spacesused[$space]=1 ; fi ;
ox=$x; oy=$y ; x=$(( $x + $xdir )) ; y=$(( $y + $ydir ))
; printf "\033[%s;%sH\033[46m \033[0m\033[%s;%sH\033[44m
\033[0m\033[0;0H" $y $x $oy $ox ; sleep 0.01 ; done
```

**Generar disco USB a partir de un archivo ISO** es posible generar un USB de cualquier imagen *ISO* descargada de la red, para ello primero debemos conocer el punto de montaje después de insertar la *USB*, mediante:

```
$ lsblk
```

o

```
# fdisk -l
```

o

```
dmseg | grep -i usb
```

si suponemos que el punto de montaje es:/dev/sdb, entonces procederemos a desmontarlo mediante:

```
$ umount /dev/sdb
```

ahora podemos usar el comando *dd* para generar el *USB* del archivo *ISO* descargado, mediante:

```
# dd if=/ruta-del-iso of=/dev/sdb bs=512M; sync
```

si queremos conocer el progreso de la generación del ISO, podemos usar:

```
# dd if=/ruta-del-iso of=/dev/sdb bs=512M status=progress;
sync
```

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

Sistemas Operativos

# 10 Apéndice D: 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[190], 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.

[190] 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.

## 10.1 Software Propietario

No existe consenso sobre el término a utilizar para referirse al opuesto del Software libre. La expresión «Software propietario (Proprietary Software)» (véase [12]), 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.

## 10.2 Software Libre

La definición de Software libre (véase [14], [15], [10], [11], [9] y [13]) 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 [14]), 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[191]; 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

[191] ¿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.

## 10.3 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.

## 10.4 Tipos de Licencias

Tanto la Open Source Initiative como la Free Software Foundation mantienen en sus páginas Web (véase [14], [15], y [13]) 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.

### 10.4.1 Licencias Creative Commons

Las Licencias[192] 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:

[192] 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[193], 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.

[193] 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.

### 10.4.2 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.

## 10.5 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-.

### 10.5.1 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.

### 10.5.2 ¿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.

### 10.5.3 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.

### 10.5.4 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.

### 10.5.5 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.

## 10.6 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.

### 10.6.1 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.

### 10.6.2 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.

# 11 Bibliografía

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

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

Herramientas
http://132.248.181.216/Herramientas/

# Referencias

[1] http://es.wikipedia.org/wiki/Microsoft_Windows 353

[2] http://es.wikipedia.org/wiki/Linux 367

[3] http://es.wikipedia.org/wiki/Mac_OS 364

[4] http://es.wikipedia.org/wiki/Android 381

[5] https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n) 112, 232

[6] https://es.wikipedia.org/wiki/Python 121, 232

[7] https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n) 141, 232

[8] https://es.wikipedia.org/wiki/C%2B%2B 145, 232

[9] http://www.gnu.org/philosophy/free-sw.es.html 652

[10] http://es.wikipedia.org/wiki/Software_libre 652

[11] http://www.hispalinux.es/SoftwareLibre 652

[12] http://es.wikipedia.org/wiki/Software_propietario 650

[13] Diferentes Tipos de Licencias para el Software, http://www.gnu.org/licenses/license-list.html 113, 652, 662

[14] FSF, Free Software Fundation, http://www.fsf.org/ 652, 662

[15] GNU Operating System, http://www.gnu.org/ 652, 662

[16] W. Gropp, E. Lusk, A. Skjellem, *Using MPI, Portable Parallel Programming Whit the Message Passing Interface*. Scientific and Engineering Computation Series, 2ed, 1999. 98, 99, 105

[17] I. Foster; *Designing and Building Parallel Programs*. Addison-Wesley Inc., Argonne National Laboratory, and the NSF, 2004. 98, 99, 105

[18] Jorge L. Ortega-Arjona, *Patterns for Parallel Software Design*, Wiley series in Software Design Patterns, 2010. 98

[19] S.J. Pennycook, S.D. Hammond, S.A. Jarvis and G.R. Mudalige, *Performance Analysis of a Hybrid MPI/CUDA Implementation of the NAS-LU Benchmark*. ACM SIGMETRICS Perform. Eval. Rev. 38 (4). ISSN 0163-5999, (2011). 106

[20] XMPI, *A Run/Debug GUI for MPI*. 98

http://www.lam-mpi.org/software/xmpi/

[21] VAMPIR, *Performance Optimization*. 106

http://www.vampir.eu/

[22] El economista, http://eleconomista.com.mx/tecnociencia/2013/01/22/clausuraran-negocios-mexico-uso-ilegal-Software

[23] PCworld, http://www.pcworld.com.mx/UNAM-y-BSA-promueven-el-uso-de-software-legal/

[24] Handbook of Floating-Point Arithmetic 2010th Edition, Jean-Michel Muller, Nicolas Brisebarre, Et alii, Birkhäuser, 2010.

[25] What Every Computer Scientist Should Know About Floating-Point Arithmetic, David Goldbergm, ACM Computing Surveys, Vol 23, No 1, March 1991

[26] 754-2008 - IEEE Standard for Floating-Point Arithmetic. IEEE. 29 de agosto de 2008. ISBN 978-0-7381-5752-8. doi:10.1109/IEEESTD.2008.4610935. (NB. Superseded by IEEE Std 754-2019, a revision of IEEE 754-2008.)

[27] Stochastic Rounding and Its Probabilistic Backward Error Analysis, Michael P. Connolly, Nicholas J. Higham , Methods and Algorithms for Scientific Computing, SIAM Journal on Scientific Computing Vol. 43, Iss. 1 (2021)

[28] IEEE, https://www.ieee.org/

[29] Intel, https://www.intel.la

[30] AMD, https://www.amd.com

[31] ARM, https://www.arm.com/

[32] Cerebras, https://www.cerebras.net/

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