**TODOS LOS APUNTES DE INTRODUCCIÓ A LA INFORMÁTICA:**

16.03.2021

**Introducción a la informática- Clase 1**

**GLOSARIO**

|  |  |
| --- | --- |
| **Hardware** | |
| Entrada | Son aquellos componentes que permiten el ingreso de información, en general desde alguna fuente externa o por parte del usuario. Proveen el medio fundamental para transferir hacia la computadora (al procesador) información desde alguna fuente, sea local o remota. También permiten cumplir la tarea esencial de leer y cargar en memoria el sistema operativo y los programas informáticos, los que a su vez ponen operativa la computadora y hacen posible realizar las más diversas tareas.  Entre los periféricos de entrada se puede mencionar: teclado, mouse, escáner, micrófono, cámara web, joystick, lectoras de CD, DVD o BluRay, entre otros. |
| Salida | Son aquellos que permiten dar salida a la información resultante de las operaciones realizadas por la CPU. Los más comunes de este grupo son los monitores, las impresoras, las consolas. y los altavoces. |
| Internos | El hardware interno es el conjunto de componentes físicos que forman parte del dispositivo principal, siendo inseparable de este.  En otras palabras, cada parte del hardware interno es una pieza fundamental de cara al funcionamiento correcto del dispositivo. Ya que, si faltara alguna de las partes de este, podría bien no ejecutar alguna tarea e incluso directamente no funcionar.  Ejemplos de hardware interno: Placa base, CPU, RAM, GPU, HDD, SDD. |
| **Servidores** | |
| Web | Almacena documentos HTML, imágenes, archivos de texto, escrituras, y demás material web compuesto por datos (conocidos colectivamente como contenido) y distribuye este contenido a clientes que lo piden en la red. |
| Bases de datos | Provee servicios de base de datos a otros programas u otras computadoras, como es definido por el modelo cliente-servidor. También puede hacer referencia a aquellas computadoras (servidores) dedicadas a ejecutar esos programas, prestando el servicio. |
| Archivos | Es el que almacena varios tipos de archivos y los distribuye a otros clientes en la red. |
| **Software** | |
| Software de sistemas | **Software de sistema**  Desvincula al usuario y al programador de los detalles del sistema informático en particular que se use, transparentando el procesamiento referido a las características internas de: memoria, discos, puertos y dispositivos de comunicaciones, impresoras, pantallas, teclados, etc. El software de sistema le procura al usuario y programador adecuadas interfaces de alto nivel, controladores, herramientas y utilidades de apoyo que permiten el mantenimiento del sistema global. Incluye entre otros:  **•**Sistemas operativos  **•** Controladores de dispositivos  **•** Herramientas de diagnóstico  **•** Herramientas de corrección y optimización  **•** Servidores  **•** Utilidades |
| Programación | **Software de programación**  Es el conjunto de herramientas que permite al programador desarrollar programas de informática, usando diferentes alternativas y lenguajes de programación, de una manera práctica. Incluyen en forma básica:  **•** Editores de texto  **•** Compiladores  **•** Intérpretes  **•** Enlazadores  **•** Depuradores  **•** Entornos de desarrollo integrados (IDE) |
| Aplicación | **Software de aplicación**  Es aquel que permite a los usuarios llevar a cabo una o varias tareas específicas, en cualquier campo de actividad susceptible de ser automatizado o asistido, con especial énfasis en los negocios. Incluye entre muchos otros:   * Aplicaciones para Control de sistemas y automatización industrial * Aplicaciones ofimáticas * Software educativo * Software empresarial * Bases de datos * Telecomunicaciones (por ejemplo, Internet y toda su estructura lógica) * Videojuegos * Software de diseño asistido (CAD) |
| **Interfaces de Usuarios (UI)** | |
| **Interfaz de linea de comandos (CLI)** | Interfaces alfanuméricas (intérpretes de comandos) que solo presentan texto. |
| **Interfaz grafica de usuario (GUI)** | Permiten comunicarse con la computadora de forma rápida e intuitiva representando gráficamente los elementos de control y medida. |
| **Interfaz nativa de usuario (NUI)** | Pueden ser táctiles, representando gráficamente un "panel de control" en una pantalla sensible al tacto que permite interactuar con el dedo de forma similar a si se accionara un control físico; pueden funcionar mediante reconocimiento del habla, como, por ejemplo, Siri; o mediante movimientos corporales, como es el caso de Kinect. |

**Introducción a la historia de la informática:**

* **Año 1122 a.C.** (China) 🡪 el ábaco.
* **Año 1645 d.C.** (Francia) 🡪 **Blaise Pascal** 🡪 Pascalina (crea la 1° calculadora mecánica).
* **Año 1801 d.C.** (Francia) 🡪 **Joseph Jacquard** 🡪 crea las Tarjetas perforadas, una manera de guardar información para crear tejidos complejos.
* **Año 1833-1842** (Gran Bretaña) 🡪 **Charles Babbage** 🡪 en base a los anteriores inventos, **intenta** crear la primer computadora (pero no lo logra) 🡪 En consecuencia, en el mismo periodo y país, **Ada Lovelace** crea el **1° algoritmo**.
* **Año 1936** (Reino Unido) 🡪 **Alan Turing** 🡪 la computadora logra ejecutar programas almacenados con la Máquina de Turing (para descifrar mensajes alemanes).
* **Año 1943** (EEUU) 🡪 **ENIAC**, operada por “las chicas del refrigerador” (consideradas junto a Ada Lovelace las primeras programadoras de la historia)
* **Año 1945** (Hungría) 🡪 **Von Neumann**: arquitectura de una computadora;

Todo esto da lugar a las primeras generaciones de computadoras:

* **40’s** = primera generación.
* **60’s** = Transitor-chip.
* **70’s** = microchip.
* **80’s** = Windows, Apple, Linux.
* Para **IBM**, desde el 2011, hemos ingresado a la “**Era Cognitiva**”.
* Lo que se desea *a futuro* con la IA es un **agente flexible**.

18.03.2021

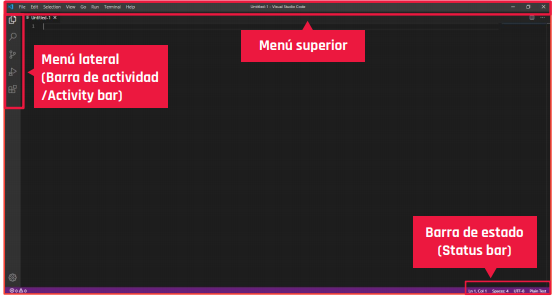
**Introducción a la informática- Clase 2.**

**VSCode**

**¿Qué es?**

Visual Studio Code o VSCode es un IDE (entorno de desarrollo integrado) desarrollado por Microsoft. Cuando hablamos de un **IDE** nos referimos a un conjunto de herramientas diseñadas para facilitarnos la creación y el desarrollo de nuestros programas o aplicaciones.

**Entorno**:



**Menú superior:**



Nos permite acceder a todas las funcionalidades de VS Code: crear nuevos archivos, guardarlos, edición de nuestro contenido, cambiar vistas, abrir terminales y mucho más.

**Menú lateral (Barra de actividad/Activity bar)**

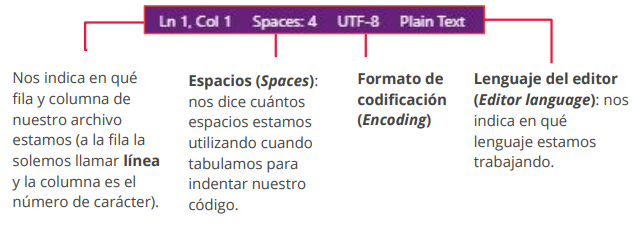
Nos permite acceder rápidamente a las funcionalidades más utilizadas de VS Code. Repasémoslas en orden de arriba hacia abajo:

* **Explorador (Explorer)**: nos da acceso a visualizar nuestra estructura de carpetas y archivos, y a los archivos que estamos editando (open editors).
* **Buscar (Search)**: nos permite buscar texto dentro de nuestros archivos, también tiene la función de buscar y reemplazar.
* **Control de fuente - versiones (Source control)**: nos va a permitir comparar nuestra versión local con la versión en la nube.
* **Ejecutar (Run)**: nos permite correr y debuggear código.
* **Extensiones (Extensions)**: nos muestra las extensiones (funcionalidades agregadas) que tenemos instaladas y nos permite buscar e instalar nuevas.

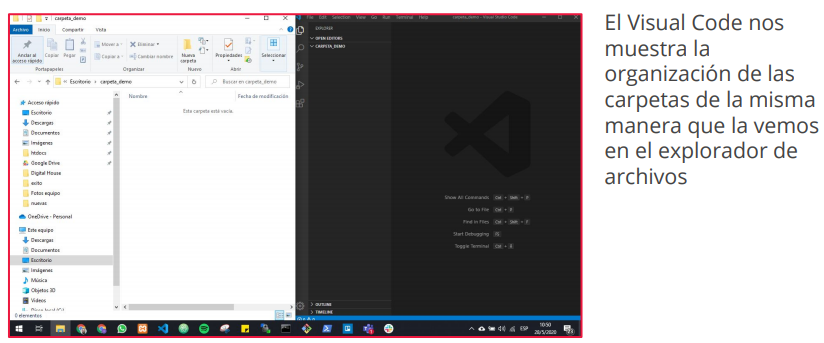


**Barra de estado (Status bar):**

Nos da información del archivo que estamos editando.



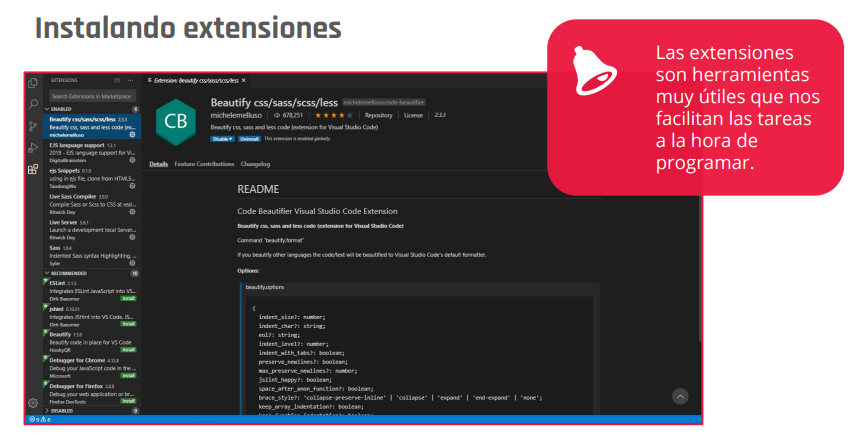
**El explorador y nuestros archivos:**



**Archivos guardados (o no)**

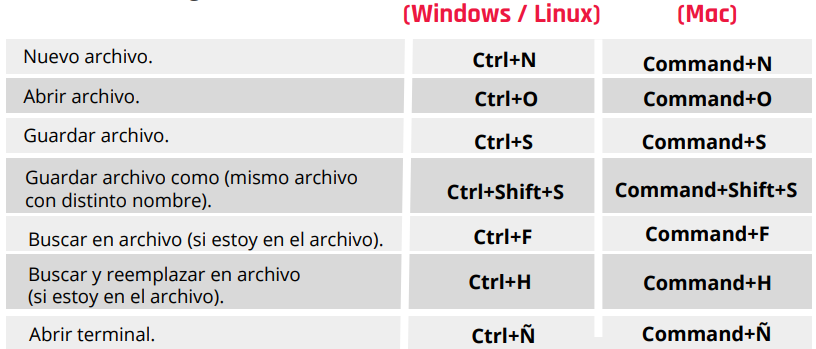
Cada vez que modifiquemos un archivo, VS Code nos indicará que hay cambios sin guardar agregando un punto junto al nombre del archivo. Es importante que todos los cambios estén guardados para poder verlos en acción. Es recomendable recordar el atajo para tener siempre guardados nuestros cambios.





**Atajos/Shortcuts:**

Nos podemos ayudar muchísimo con algunos atajos que nos provee VS Code, estos son algunos ejemplos:



**¿Qué es la terminal y por qué es importante?**

Es un **programa** que está presente en **todos** los **sistemas operativos**. Usando la terminal podemos darle **órdenes** al sistema.

La terminal o “Interfaz de línea de comandos” (CLI) nos permite darle INSTRUCCIONES a la computadora a través **DE LA EJECUCIÓN de COMANDOS**; esos comandos son órdenes que el usuario proporciona a un sistema informático mediante texto.

Todo eso que hacemos usando el mouse lo podemos lograr a través de comandos en la terminal, como, por ejemplo, **crear** una **carpeta** o un **archivo**, mover un archivo de lugar y mucho más.

**Terminal en Windows.**

Para abrir la terminal en Windows, podemos presionar las teclas Win+R y, en la ventana emergente, escribir el comando "cmd.exe". También podemos hacer uso del buscador y escribir la palabra "cmd" o "PowerShell".

**Terminal en Mac.**

Para abrir la terminal en Mac, podemos ir al Finder y pulsar en Archivo > Nueva ventana del Finder, en el menú lateral izquierdo, hacer clic en Aplicaciones y buscar en el listado la terminal.

**Terminal en Linux**.

Para abrir la terminal en Linux, podemos presionar las teclas Ctrl + Alt + T o simplemente ingresar la palabra “terminal” en el buscador que ofrece el sistema operativo.

**¿Cuáles son las ventajas de la terminal?**

* **Velocidad**. 🡪 Los comandos son más rápidos de ejecutar que en una interfaz gráfica (GUI)
* **Compatibilidad**.
* **Sin distracciones**.
* **Productividad**.
* Los comandos son más rápidos de ejecutar que una interfaz gráfica.
* Toda acción que queramos ejecutar desde una aplicación primero tendríamos que navegar en inicio, aplicaciones, buscar la aplicación, ubicar la acción en la interfaz gráfica y ejecutarla. Todos estos pasos llevan menos de un segundo utilizando un COMANDO.
* En la terminal, con solo teclear el comando “**Tree**”, podríamos visualizar todo el contenido completo en una sola interfaz.

**¿Los comandos se actualizan?**

A medida que un programa evoluciona, aparecen nuevas funcionalidades 🡪 quizás, cambie la interfaz y la manera de interactuar gráficamente…

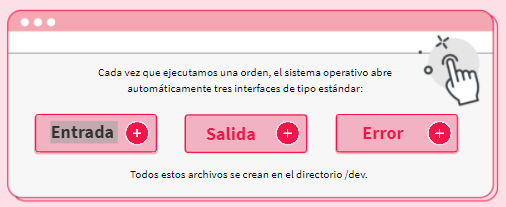
SIN EMBARGO, **los comandos SIEMPRE van a funcionar de la misma manera**. Esa es la ventaja de **COMPATIBILIDAD**, que nos permite trabajar con cualquier versión.

La línea de comando es una interfaz de usuario más limpias y **sin distracciones**, aunque la misma nos permita personalizarla.

Gracias a todas estas ventajas de la terminal, podemos ser más **productivos** en nuestro trabajo diario.

**Terminal ¿Qué podemos hacer con la terminal?**

Somos libres de usar el sistema de ventanas de Linux, pero creemos que utilizar la terminal nos será muy provechoso.



* **Entrada**: La entrada estándar (stdin) se refiere al archivo por el que una orden recibe su entrada (por defecto, es el teclado)
* **Salida**: La salida estándar (stdout) se refiere al archivo por el que una orden presenta sus resultados (por defecto, es la pantalla o más concretamente la ventana en la que se está ejecutando el intérprete de órdenes).
* **Error**: El error estándar (stderr) se refiere al archivo por el que una orden presenta los mensajes que va generando cuando ocurre un error (por defecto, también es la pantalla).

**Navegación e interacciones con ficheros:**



* **De tipo informativo**:

|  |
| --- |
| * + **pwd** / Mostrar el nombre de la carpeta en el que uno se encuentra situado (print working directory). |
| * + **Cd** / Cambiar la carpeta de trabajo: Con este comando nos podemos mover entre diferentes directorios. Si queremos ir a un directorio en particular. |
| * + **Ls** / Listar el contenido de directorios (list): Este comando lista los ficheros y carpetas. |

* **Crear o eliminar archivos**:

|  |
| --- |
| * + **gedit** / Crear y editar archivos de texto. |
| * + **rm** / Borrar archivos. |

* **Crear o eliminar carpetas**:

|  |
| --- |
| * + **mkdir** / Crear un carpeta (make directory). |
| * + **rmdir** / Borrar un carpeta (remove directory) |

* **Copiar archivos y directorios**:

|  |
| --- |
| * + **cp** / Copiar un archivo o carpeta en el directorio especificado (copy) |
| * + **cp -r** / Copiar carpetas |
| * + **Lmv** / Mover un archivo o carpeta a un archivo o carpeta (move) |

**Comandos**

**¿Qué son los comandos y cuáles existen?**

Los comandos son instrucciones codificadas para ser interpretadas por un sistema operativo. Las computadoras nos proveen comandos que podemos ejecutar en la terminal para enviarle estas instrucciones de forma clara y precisa.

Un **comando** es una **orden** o **instrucción** capaz de ser **interpretada** y **ejecutada** por la computadora.

Para utilizar los comandos, debemos saber: **CÓMO se escribe**, y **CÓMO indicar una ruta de un directorio o un archivo**.

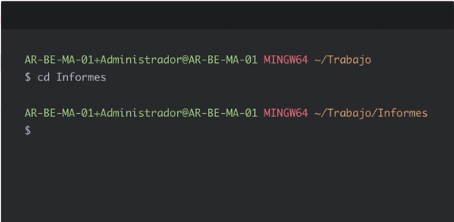
**¿Qué es la sintaxis de un comando?**

Todo comando tiene su propia **sintaxis**, es decir su **manera** de ser escrita para que la computadora pueda **entender** y **ejecutar** lo que el usuario le especifica por medio del comando.

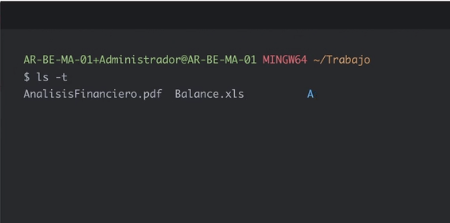
**¿Cómo se escribe un comando? ¿Cómo indicar una ruta?**

Vamos a usar el ejemplo de una persona que debe buscar un archivo para imprimir y tiene muy poco tiempo, por lo que necesita un atajo (y bueno, una característica de la terminal es su velocidad)

1. **PRIMERO**, hay que abrir la terminal. En Windows, presionamos la tecla Windows+R y escribimos cmd. En Linux, presionamos Ctrl+Alt+T al mismo tiempo. En Mac, presionamos cmd+espacio y escribimos terminal.
2. Al abrirse la terminal, esta se posiciona sobre una ruta por defecto (similar a cuando abrimos el explorador de archivos y nos lleva a un directorio).
3. Para proseguir, tenemos que saber en DÓNDE estamos posicionados para movernos entre los directorios y acceder al archivo que queremos copiar.
4. Podemos usar el comando **pwd** (print working directory = imprimir directorio de trabajo), y presionamos enter.
5. Como resultado, nos va a proporcionar la ruta en donde estamos posicionados.
6. Ahora que sabemos nuestra ubicación, recordamos que nuestro archivo se encuentra en ESA ruta, pero en el directorio llamado “Informes”. Entonces, para poder trasladarlo a esa ubicación, necesitamos usar el comando **CD** (cambio de directorio): e indicarle a qué carpeta o directorio queremos ir. Esta información adicional la llamamos “**Argumento**”, y se escribe DESPUÉS DEL COMANDO de esta manera:



1. Ahora que ingresamos en la carpeta, nos encontramos con otro problema: NO RECORDAMOS EL NOMBRE. En este caso, tenemos que intentar recordar qué fue lo último que modificamos y vamos a usar el comando **ls (list)** que al ejecutarlo nos muestra TODOS los archivos y carpetas del directorio actual.
2. ¿Cómo sabemos cuál fue el último de los archivos y carpetas que fue modificado? Luego de escribir el comando ls debemos pasarle una indicación, la cual llamaremos **OPCIÓN** o **FLAG**, para que los ordene por fecha de modificación. Esto se hace mediante un guión (-) en el medio, sin dejar espacio (ls-t). ahora tendremos la lista de archivos ordenados, y sabremos que el archivo que buscábamos es el primero:



1. Ahora solo queda copiarlo en nuestro pendrive, en donde el comando cp (copiar archivos) nos va a ayudar.

Los comandos pueden tener **ARGUMENTOS** y **OPCIONES**.

* En el caso del comando cd, le enviamos por argumento el directorio al que queremos ir, y por el comando ls le enviamos por opción que queríamos el listado de archivos ordenado por fecha de modificación. Gracias a esta sintaxis, los comandos tienen una gran flexibilidad (cd directorio / ls-opción), permitiendo que los configuremos y les podamos enviar los datos que necesitan a través de los parámetros.



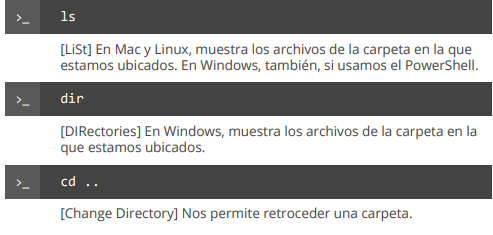
En cuanto a CÓMO indicar una ruta, directorio o archivo:

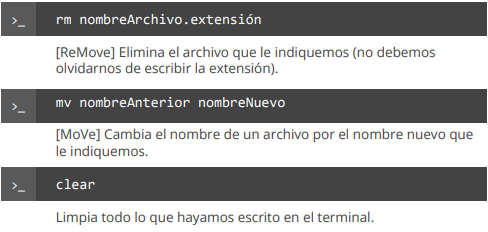
* Podemos usar la **ruta absoluta**, con comando pwd (observamos toda la ruta completa hasta el directorio en donde nos encontramos): /c/Users/Administrador/trabajo
* Podemos también usar la ruta relativa, damos por sentado que estamos haciendo la indicación desde donde estamos parados; esto lo vemos reflejado cuando ejecutamos el comando **cd informes** 🡪 /c/Users/Administrador/trabajo (relativa 🡪 informes)

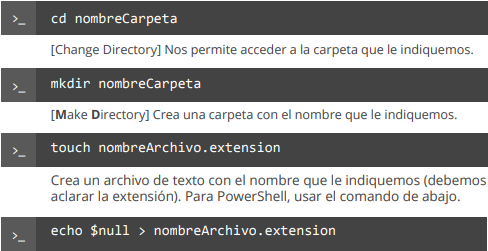
**¿Cómo distinguimos una ruta relativa de una absoluta?**

Para las direcciones ABSOLUTAS, vamos a usar la barra (/) al comienzo de la ruta (ej.: /= raíz). Mientras tanto, en las direcciones RELATIVAS no vamos a usar la barra (ej.: = raíz).

**COMANDOS BÁSICOS**:







Con el comando **mv**, podemos mover archivos/directorios, pero también re-nombrarlos. Hay particularidades:

* Por ejemplo, $mv archivo1 archivo2, cambia el nombre del primero por el segundo.
* $mv archivo1 /carpeta1/archivo1 🡪 mueve un archivo de un origen a un destino.
* $mv –f archivo1 /carpeta/archivo1 🡪 reemplaza el archivo de destino sin preguntar al usuario
* $mv –i archivo1 /carpeta 🡪 pregunta por cada archivo a sobre-escribir antes de hacerlo.
* $mv directorio1 directorio2 🡪 mueve al directorio1 al directorio2; a diferencia de cp, no necesita opciones especiales para mover directorios.

Para mover múltiples archivos, hacemos lo siguiente:

* Se especifican los archivos que queremos mover:
  + $mv file1 file2 dir1 🡪 mueve file1 y file2 al directorio dir1.
* Puede usarse un pattern-matching. Si tenemos muchos archivos .img, .txt, o lo que sea, y los queremos mover a un directorio, escribimos el comando así:
  + $mv \*.txt documentos 🡪 acá movimos todos los archivos .txt al directorio documentos. Si vamos a la carpeta documentos con el comando cd ($cd documentos), y luego ejecutamos el comando ls, deberíamos ver todos los archivos .txt allí.

Comando **rm** se usa para eliminar archivos y directorios.

Modificadores comunes que acepta rm:

* -r Procesa subdirectorios de forma [recursiva](https://es.wikipedia.org/wiki/Recursi%C3%B3n).
* -i Pide confirmación para cada borrado.
* -f Forzado, ignora archivos no existentes y elimina cualquier aviso de confirmación.
* -v Muestra el nombre de cada fichero antes de borrarlo
* -d Borra los directorios vacíos.

**23.03.2021**

**Introducción a Git**

¿Cómo se trabaja en forma colaborativa en un ambiente profesional? ¿Nos enviamos mails con código? ¿Mensajes de WhatsApp? ¿Usamos Google Drive? Mmm... ¿suena raro, no?  
  
Definitivamente no usamos nada de todo esto, pero utilizamos una herramienta que tiene muchos de los beneficios que nos dan las que mencionamos anteriormente: trabajar en grupo de forma colaborativa, comunicar cambios, **manejar distintas versiones**. Como dijimos al principio del módulo, vamos a conocer una herramienta que nos va a servir durante toda la cursada.

En el mundo de desarrollo de aplicaciones, tenemos dos necesidades:

* **Backup** de nuestros archivos continuamente actualizados.
* La posibilidad de **compartir** nuestro trabajo.

Para esto, existe **git**. **Git** permite hacer un completo backup de tus archivos y compartirlo con tus compañeros.

Pero, ¿qué es git? Es un software de control de versiones, pensado para la eficiencia y confiabilidad del mantenimiento de las actualizaciones sobre el código fuente. Su propósito es llevar registro de los cambios de los archivos y coordinar el trabajo que varias personas realizan sobre los mismos.

Con git, vas a tener un historial completo de las versiones de tus archivos sin la necesidad de crear varias copias del mismo, pudiendo tener todos los cambios registrados e INCLUSO volver a la versión inicial cuando lo desees.

Adicionalmente, con git se pueden compartir los archivos de manera fácil y sencilla, pudiendo hacer un seguimiento de los mismos para tener conocimiento sobre qué integrante del equipo realizó las modificaciones más recientes del archivo.

**CREANDO MI PRIMER REPOSITORIO ON-LINE**

Supongamos que tenés una carpeta con varios archivos .js, y que querés hacer un backup de los mismos con **git.** LO PRIMERO que tenés que hacer es, desde la terminal, posicionarte en esa carpeta y escribir **$ git init.** A partir de ese momento, deberás recibir desde la consola el mensaje de que se ha inicializado un repositorio vacío.

Lo que generó ese comando es un repositorio local en la carpeta en la que tenés los archivos. Inicialmente, se ha generado un repositorio VACÍO, porque no se le ha indicado qué archivos quieren guardarse.

Un **repositorio**, sea local o remoto (remoto= está en Github), es un almacén de archivos. Los archivos se irán guardando en pequeños paquetes llamados “commits”; estos paquetes permitirán ir haciendo un SEGUIMIENTO de todos los cambios que se van realizando, ya que todos tienen una timestamp (o fecha de creación) y un autor. Los commits, son entonces, nuestro historial de cambios que se fueron haciendo en el proyecto.

Por todo esto, es importante decirle al repositorio “quién sos vos”, porque es esta información la que le permite a git hacer un completo seguimiento de los cambios realizados 🡪 para agregar tu identidad al repositorio, **hay que escribir DOS COMANDOS DE GIT**:

1. El primero permite agregar tu nombre de usuario, o sea, el nombre de usuario con el que te registraste a github.com. Lo que hay que hacer, entonces, es acceder a la terminal, pararse en la carpeta correcta y escribir: **$ git config user.name “mi-usuario”** (o sea, tu nombre de usuario entre comillas después de la palabra name). Hecho esto, pulsas enter. A diferencia del comando git init, este comando no mostrará ningún texto adicional en la terminal. Si querés corroborar que tu usuario se agregó correctamente al repositorio, bastará con escribir el comando anterior **$ git config user.name** SIN EL NOMBRE DE USUARIO, y al presionar enter, debería verse en la terminal el texto que agregaste con anterioridad.
2. El repositorio también necesita saber cuál es el correo electrónico con el que te registraste en github. Para esto, se escribe el comando **$ git config user.email** “[miCorreo@email.com](mailto:miCorreo@email.com)”. Como con el comando anterior, para verificar que todo salió bien, se ingresará el comando **$ git config user.email**.

Si estas usando tu propia computadora y no querés configurar esto cada vez, a los comandos anteriores les podés agregar - - global (ej: git config - - global user.name). De esta manera, se indica que cualquier repositorio existente en tu computadora tendrá tu firma de tu nombre y tu email.

**Palabras clave**

A modo de resumen, repasemos las palabras claves de este video:

* **Repositorio local**: es el que tiene todos los archivos (que hayas guardado en él) en la computadora.
* **Commits:** son los paquetes que nos van a permitir ir haciendo un seguimiento de los cambios que vamos realizando, dado que cada uno de ellos tiene una timestamp, o fecha de creación, y un autor. Los commits van a ser nuestro historial de cambios que se fueron haciendo en el proyecto.

**Estos son los comandos que vimos en el video**:

* git init *// crea el repositorio*
* git config user.name "nombre de usuario" *// agrega nuestra identidad*
* git config user.email "nombreUsuario@email.com" *// agrega nuestro email*
* git remote add origin https:*//github.com/DH/RepoRemoto // apunta al repositorio remoto*

Repositorio de git en la carpeta: Certified Tech Developer > Introducción a la informática > Clase 3

usuario: abegri

email: [abelendagrisel.a@gmail.com](mailto:abelendagrisel.a@gmail.com)

**Agregando archivos al repositorio**

Para que Git pueda llevar un control de las modificaciones realizadas en un archivo tenemos que indicarle qué archivos queremos que mire. Pero... ¿Cómo se hace eso?

Para saberlo, miremos el siguiente video...

**¿Cómo agregar los archivos que yo elija al repositorio local?**

Los commits son la BASE del sistema de git, pues son los que nos permiten hacer un completo seguimiento y control sobre las versiones de nuestros archivos.

Para agregar esos archivos, hay que escribir un comando desde la terminal (SÓLO VA A FUNCIONAR SI ESTÁS EN UNA CARPETA QUE TIENE UN REPOSITORIO INICIALIZADO).

El comando se llama **git add**, al cual se le indicarán los archivos que queremos agregarle.

Otro comando que puede tenerse a mano es **git status**, el cual te dirá siempre el estatus de tus archivos y el estado de los mismos respecto del repositorio.

Al ingresar estos comandos, git podrá avisarnos también que hay archivos *SIN SEGUIMIENTO*; estos no son archivos necesariamente nuevos, también pueden ser archivos que ya estaban en el repositorio, pero que sufrió cambios.

Suponiendo que tengo un archivo llamado “funciones.js”, y sólo quiero agregar ese al repositorio, escribiré: **git add funciones.js**. Si todo sale bien, la terminal NO DEBERÍA ARROJAR NINGÚN MENSAJE. Si, ahora, ponemos **git status**, saltará un mensaje que nos dirá cómo están las cosas en el repositorio; para esta instancia, deberás ver un mensaje que dice que el archivo “funciones.js” ha sido agregado, pero que los otros archivos de la carpeta no están con seguimiento.

Ahora, ¿hay que agregar uno por uno los archivos? NO. Si queremos agregar TODOS LOS ARCHIVOS PRESENTES EN EL REPOSITORIO, ingresaremos este comando: **git add .** (va con el punto).

Si hacemos esto y luego ingresamos **git status**, se verá que todos los archivos se agregaron al repositorio y no hay ninguno sin seguimiento.

Ahora bien, **¿qué pasa cuando un archivo que ya agregaste al repositorio sufre de alguna modificación?**

Cuando ocurre esto, git asume que el archivo es “nuevo” y, por lo tanto, pasará a ser nuevamente un archivo que no tiene seguimiento. Este es el famoso **CONTROL DE VERSIONES**; para git, cada cosa nueva que sucede en un archivo, es una indicación de cambios efectuados. Lo único que hay que hacer para que git integre ese archivo de nuevo con sus cambios, es ejecutar **git add, con el nombre del archivo o con el punto si modificaste varios.**

**Estos son los comandos que vimos en el video**:

* git add . *// agrega todos los archivos*
* git status *// seguimiento del estado de los archivos*

**Confirmando archivos.**

Cuando trabajamos con archivos, estamos acostumbrados a que estos se guarden en forma automática o, a pedirle al programa que lo guarde (el famoso Ctrl + S). Confirmar las modificaciones en Git es de suma importancia ya que nos permite establecer un punto de control. ¡Veamos cómo hacerlo!

Vamos a empezar a crear commits. Los commits son confirmaciones a través de las cuales le estamos diciendo al repositorio que los archivos que fuimos agregando lo deseamos oficialmente como un pequeño paquete de adiciones y/o modificaciones que tendrán, a su vez, una marca indeleble de tiempo y estarán firmados por un autor.

Los commits generan puntos cronológicos en la línea de tiempo del proyecto que nos permiten identificar el estado del mismo hasta ese momento en específico y, a su vez, volver a versiones anteriores si se deseara.

Para poder crear un commit, tenés que haber agregado previamente los archivos modificados al repositorio y luego escribir el comando **$ git commit –m**, seguido de un mensaje entre comillas **“”**. Este mensaje puede ser cualquier cosa, pero la idea es que con ese mensaje describas brevemente qué hiciste. Ejemplo: **git commit –m “primer commit del proyecto”**; aquí se ha generado un punto en la línea de tiempo con las modificaciones que se han realizado hasta ese momento en el repositorio.

¿**Qué sucede si agregas un archivo nuevo**? Bueno, agregamos el archivo con **git add** y posteriormente volver a realizar un nuevo commit; es decir, en un commit sólo entran archivos a los que hayas hecho previamente **add**. En este caso, el mensaje podría ser $ **git commit –m “agrego el archivo usuarios.js”**.

En esta instancia, el proyecto tiene dos commits. Comienza a volverse un proceso repetitivo.

En VSCode, podemos ver los archivos con una U de “untracked”. Si vamos al Gitbash y ejecutamos el comando **git add .**, y volvemos a VSCode, veremos una “**A**” de “**added**”.

Después de eso hacemos nuestro commit: $ **git commit –m** “Creando los 3 archivos principales”. Al ejecutar git status, debería decir “nothing to commit, working directory clean”.

Más aún, el comando **git log** va a registrar un historial del cambio de nuestros proyectos.

**AHORA**, ¿**qué sucedería si ahora es un archivo que ya existe dentro del repositorio el que cambió**?

Lo agregás de nuevo con git add, y luego creas un commit si querés que ese cambio quede oficialmente en la línea del tiempo del proyecto. En VSCode, ese archivo tendrá una “m”, de “modificado”.

Estos son los comandos que utilizamos en el video para confirmar archivos:

* **git add .** *// agrega los archivos*
* **git commit -m "mensaje"** *// commitea los cambios hechos*

**CLASE EN VIVO.**

* Con el comando **rm –r\*** borras TODO.
* Para renombrar un archivo en la consola: $ mv nombre del archivo nombre nuevo. Si no existe el archivo “nombre del archivo”, se creará uno nuevo con el nombre “nombre nuevo”.
* Si queremos mover archivos (todos del mismo tipo/con la misma extensión) de una, podemos usar el símbolo \*.
* Hay que hacer un commit por cada modificación de cada archivo.
* Con el comando **git changed** te aparecen todos los cambios que hiciste desde el ultimo commit.
* Una vez que ejecutamos **git commit –m** (el –m es para agregar un mensaje), el archivo pasa a ser parte de la **RAMA PRINCIPAL**, pero antes de eso, aparece como “A” de “added”.

**25.03.2021**

**GITHUB**

Siguiendo las instrucciones del video, crearemos un repositorio en la nube. Ahora, vamos a poder descargar y modificar el repositorio desde cualquier computadora del mundo. Para poder cumplir con este objetivo vamos a utilizar [https://github.com](https://github.com/). Pero, ¿qué es Github?

**GitHub es una plataforma colaborativa que nos va a permitir llevar un control de versión sobre nuestro código.**

En el ambiente de la programación, necesitamos sistemas mucho más estables, funcionales y rápidos de lo que pueden ser DropBox o Google Drive. En este momento es que GitHub toma su protagonismo.

GitHub NO ES lo mismo que Git. GitHub es un LUGAR EN LA NUBE, en donde se pueden alojar los archivos de tus proyectos de programación, de manera gratuita, sin más trabajo que crear una cuenta.

Una vez que tenemos una cuenta, podemos comenzar a crear **Repositorios** 🡪 los repositorios son, básicamente, un lugar en donde se irán almacenando los archivos de tu proyecto y en donde podrás ir haciendo un seguimiento de los mismos.

**IMPORTANTE** = **A UN PROYECTO le corresponde UN REPOSITORIO**.

En GitHub podes tener la cantidad de repositorios que quieras. Y, los repositorios que están en GitHub son llamados “**Repositorios Remotos**”, pues están en la nube. En paralelo, cada persona de tu equipo va a tener una copia de ese repositorio en su computadora, y esos repositorios se llaman “**Repositorios locales**”.

* **Repositorio local** = copia del repositorio en la computadora.
* **Repositorio remoto** = repositorio en la nube.

Para que Git funcione bien, es necesario crear un vínculo entre local y remoto para que puedan mantenerse los backups de los archivos locales conectados con ese lugar en la nube.

<https://github.com/abegri/prueba-node.git>

En GitHub (en LA NUBE), vamos a crear una versión on-line y compartida de nuestros archivos. Cada persona de nuestro equipo, a través de los comandos de git, se descargará los cambios y subirá las actualizaciones.

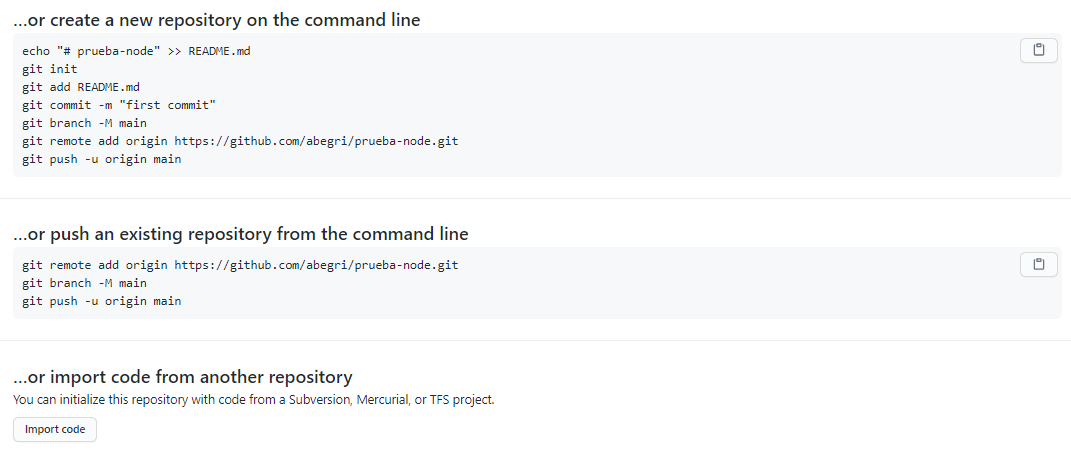
**TIP=**

* Las empresas de desarrollo se fijan bastante en los perfiles de GitHub de sus posibles contrataciones.
* Esto te va a sumar MUCHO a la hora de buscar trabajo.

**Palabras clave**

A modo de resumen, repasamos las palabras y conceptos clave del video sobre GitHub:

* GitHub es un lugar en la nube.
* **Repositorio**: es el lugar en donde se irán almacenando los archivos de nuestro proyecto y a través del cual podremos hacer seguimiento de los mismos.
* Repositorios remotos: viven en la nube, es decir, en GitHub.
* Repositorios locales: viven en nuestra computadora.



**Conectando nuestro repositorio local a GitHub**

Para que Git pueda llevar un control de las modificaciones realizadas en un archivo tenemos que indicarle qué archivos queremos que mire. Pero... ¿Cómo se hace eso?

Llegó finalmente el momento de conectar lo que estuvimos realizando localmente con GitHub. Para poder hacer esto, debemos tener:

* Una cuenta en GitHub.
* Un repositorio local que utilizaremos para poder conectarlo.

Con el objetivo de poder practicar junto al video, tenemos que poseer un repositorio local configurado:

* Debemos inicializar un repositorio. Para esto, ejecutemos git init en la carpeta que queramos conectar el repositorio.
* Luego tenemos que indicar al repositorio nuestros usuario ejecutando dos comandos:

1. git config user.name “mi usuario” (escribimos nuestro nombre de usuario).
2. git config user.email “miCorreo@email.com” (escribimos nuestra dirección de correo).

Si tenemos dudas de cómo ejecutarlos, podemos recurrir al video de la clase de git “Creando nuestro primer repositorio local”.  
  
Con estos pasos hechos, continuemos con el siguiente video.

Cuando creas un repositorio en GitHub, el mismo queda con una **URL única**, la url tiene una estructura así: <https://github.com/mi-usuario/nombre-del-repo.git>. Con esta URL en el portapapeles (o sea, copiada), se escribe el comando **git remote add origin** url de github. Si todo sale bien, la terminal no debería arrojar NINGÚN MENSAJE (yay, me salió bien); pero, para estar 100% seguros, escribí el comando **git remote –v**, y sale algo así:

origin https://github.com/abegri/prueba-node.git (fetch)

origin https://github.com/abegri/prueba-node.git (push).

Lo que hicimos con **git remote –v** fue indicarle al repositorio local con qué repositorio remoto se deberá sincronizar, así cuando quiera enviar los archivos desde la computadora a GitHub o traerlos archivos de GitHub a la computadora, los directorios ya estarán conectados entre sí.

AHORA, ya está TODO LISTO y CONECTADO ENTRE SÍ para comenzar a agregar archivos a tu repositorio local y enviarlo posteriormente al repositorio remoto en la nube.

**SUBIENDO ARCHIVOS**

* Antes de subir a GitHub, tenés que tener todo PREVIAMENTE commiteado. SÓLO SE SUBEN LOS COMMITS YA CERRADOS.
* Debemos usar el comando **git push**, se le solicita a GitHub que inserte/envíe los archivos que tenemos en nuestro repositorio local al repositorio remoto.
* Además, git push necesita adicionalmente de otro conjunto de palabras. Antes, era origin master, hoy por hoy es origin main. Quedaría así: **git push origin main**. Esto significa que querés llevar los archivos de tu repositorio local al repositorio remoto. Para git, el repositorio remoto se llama “origin”, y adicionalmente queremos que se inserten los archivos en la rama principal, que se llama “**main**”.
* Una vez que comitteamos todo, tenemos que poner **git push origin master**/main
* El proceso es git add 🡪 git commit –m”” 🡪 git push.

**RAMAS**

Las ramas dentro de un repositorio son COPIAS ALTERNATIVAS del mismo hasta el momento, es una línea paralela en la que podes ir agregando nuevas funcionalidades, SIN tener que modificar la línea original de tiempo, ni afectar qué código hay ahí.

Es como una versión dos del proyeco, en la cuál se podrán probar cosas nuevas, y, si al final te gustan, las podes fusionar con la rama principal.

Sin embargo, esto requiere un poco más de habilidad y experiencia.

**IMPORTANTE** 🡪 trabajando en equipo, puede que tengamos problemas al intentar hacer push, ya que alguien en el equipo pudo haber modificado el archivo previamente. Para esto, basta con descargar del repositorio remoto, los cambios hechos previamente.

Comando utilizados en la clase:

* git add . *// agrega los archivos*
* git commit -m "mensaje" *// comitea los cambios hechos*
* git push origin master *// envia los cambios al repositorio remoto*
* git status *// seguimiento del estado de los archivos*

**BAJANDO ARCHIVOS.**

Bajar los proyectos que tengas en GitHub a cualquier computadora.

* Comando: **git clone** // crea una copia exacta en la computadora de todos los archivos existentes en un repositorio remoto. 🡪 ESTE COMANDO SE EJECUTA UNA SOLA VEZ, LA PRIMERA VEZ QUE LOS DESCARGAS A TU COMPUTADORA Y NO ESTABAN EN ELLA.
* Git clone necesita una tercera parte, para saber de qué repositorio queremos hacer la copia. Esa tercera parte es la URL. EJEMPLO: git clone https://github.com/abegri/prueba-node.git. Formato: $ git clone <https://github.com/usuario/repositorio.git>.
* Con esto, git se va a descargar todos los archivos allí presentes y los dejará en la computadora listos para ser utilizados. COMANDOS PARA ENTRAR EN LA CARPETA:
  + Ls
  + Cd prueba-node/
  + Ls //se van a ver todos los archivos.
* Esa carpeta ya está sincronizada, y lo podemos comprobar con el comando: **git remote –v**// nos indica que estamos conectados con la url que está en pantalla.
* GIT SÓLO LO UTILIZAMOS EN LA NUBE, para seguir editando, agarramos nuestro editor de confianza.

**¿Qué sucede cuando trabajas en la máquina de tu trabajo, subís los archivos a GitHub y luego los querés descargar en tu computadora personal? ¿o cuando otra persona subió cambios y queremos descargarlos?**

Partiendo desde la concepción inicial de que en ambas computadoras YA EXISTEN esos archivos, para mantener sincronizados los archivos existentes en las computadoras se siguen estos pasos:

* Cuando queremos ACTUALIZAR LOS ARCHIVOS QUE YA ESTÁN EN TU COMPUTADORA, usas el comando: **git pull origin master**.
* Querés hacer lo mismo en otra compu, ejecutás el mismo comando.

**CLASE EN VIVO.**

Para deshacer un commit:

* **Git revert <commit>**//para revertir los cambios de un commit y reemplazarlos con el nuevo.
* **Git reset –hard <commit>** // Resetea el HEAD a un commit anterior … y descarta todos los cambios desde entonces.
* **Git reset <commit>** // preservar todos los cambios como cambios pendientes.

Cómo borrar un repositorio. No es recomendado usar el –rf para nosotros porque borra todo, ignorando los cambios.

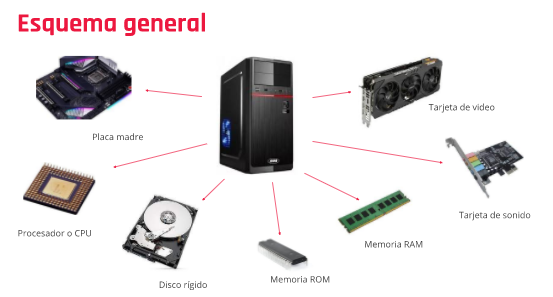
**30.03.2021**

**HARDWARE Y SOFTWARE**

**Componentes de la computadora**

**Componentes internos**

* Los componentes internos de una computadora son todos los elementos físicos inseparables de la misma. En otras palabras, si faltara alguno de ellos, podría no funcionar o hacerlo de manera incorrecta.



* **Placa madre o motherboard**: es la placa principal de cualquier computadora, al que todos los demás dispositivos se conectan, tanto de manera **directa** (como los circuitos eléctricos), como **indirecta** (a través de puertos USB u otro tipo de conectores).
* **Procesador**: también llamado **unidad central de procesamiento o CPU**, es el “**cerebro**” de la computadora, su función es interpretar y ejecutar las instrucciones a través de operaciones básicas: aritméticas y lógicas. Es quien “dirige” las operaciones que realiza la computadora.
* **Memoria RAM y ROM**:
  + La memoria **RAM** es el componente que almacena información de manera temporal. Tiene la particularidad de que el contenido de la misma se elimina cada vez que se apaga la computadora.
  + La memoria **ROM**, por otra parte, almacena información de **manera permanente**. Guarda todo lo relacionado a la configuración inicial para el arranque de la máquina y funcionamiento básico.
* **Placa de video y sonido**: son componentes internos que se conectan a la placa madre.
  + La **placa de video** es la encargada de mostrar imágenes en el monitor.
  + La **placa de sonido** permite que la computadora reproduzca sonidos a través de auriculares o parlantes. También permite recibir sonidos a través de micrófonos.
* **Dispositivos de almacenamiento secundario**: almacena datos de manera permanente. Es información que la computadora no necesita de manera inmediata para su funcionamiento. Puede almacenar archivos de todo tipo como documentos, imágenes, videos, audios, etc. El dispositivo de almacenamiento secundario interno es el **disco rígido, o disco duro**.

**Componentes externos**

* Los **componentes externos** de una computadora son todos aquellos dispositivos que utiliza la computadora, pero que no son imprescindibles para su funcionamiento.
* **Dispositivos periféricos**: son aquellos que se conectan a la CPU para añadir funciones u operaciones a la computadora, pero no son parte esencial de la misma.
* Pueden ser:
  + **De entrada**: introducen datos a la computadora. Son aquellos dispositivos que sirven únicamente para introducir datos, es decir, para captar nueva información o comunicar al usuario con la computadora. Son los que comunican a la computadora con el mundo exterior de distintas maneras.



* + **De salida**: extraen datos de la computadora. Son aquellos periféricos que permiten extraer información proveniente de la computadora. Es la única manera en la que la computadora puede emitir información y comunicarse con el usuario.



* + **Mixtos**: cumplen ambas funciones. Son aquellos dispositivos electrónicos que permiten el ingreso y el egreso de información de la computadora. Llevan a cabo las tareas de los dispositivos de entrada y de salida a la vez.



* + **De almacenamiento**: permiten almacenamiento permanente. Se conectan de manera externa.
  + **De comunicación**: permiten la conexión entre computadoras.

**The Big Picture**

¿**Qué características tiene que tener un dispositivo para considerarlo una computadora**?

* Una computadora es esencialmente una máquina que recibe datos, los procesa, muestra los resultados y estos pueden ser almacenados, transmitirse o imprimirse.
* Desde las computadoras antiguas que ocupaban habitaciones enteras hasta nuestros smartphones siguen realizando las mismas operaciones básicas.

¿Cómo es que hacer “clic” en un video hace que este se pause?

* Al presionar el botón, la información sobre el clic viaja a través de impulsos eléctricos, los cuales van a la computadora a través de medios llamados “**buses de datos**”. Estos pulsos eléctricos son interpretados como “0” y “1”, los cuales se los conoce como bits. Así, la información es procesada y transformada. La información que enviamos a través del mouse es recibida por el “cerebro” de la computadora = el CPU. La **CPU** realiza miles de millones de operaciones por segundo; sin embargo, una particularidad de ella es que, para realizar CUALQUIER TAREA, NECESITA INSTRUCCIONES.
* Cuando una petición, como nuestro clic, llega al CPU, esta debe buscar en algún sitio las instrucciones necesarias para saber qué debe hacer a continuación. Estas instrucciones están guardadas en la memoria principal de la computadora. Debe buscar información sobre:
  + El botón sobre el que hicimos clic.
  + El programa que contiene ese botón.
  + Lo que sucede luego de hacer clic con el botón.
  + Encontrar la instrucción para pausar el video.
* Luego de eso, enviará la información al dispositivo de salida (el monitor, por ejemplo), otra vez en ceros y unos para que nos muestre nuestro video detenido.

**¿Quién piensa?**

**El procesador**

Es la parte más importante de la PC. Lee instrucciones almacenadas en la memoria RAM.

Buscar, leer y ejecutar son LA BASE DEL FUNCIONAMIENTO DE NUESTROS DISPOSITIVOS ELECTRÓNICOS.

¿*A qué velocidad se ejecutan estos procesos*? 🡪 la frecuencia, los núcleos, los subprocesos o hilos y la memoria caché son las características de la CPU que determinarán la velocidad en la que se van a ejecutar todos estos pasos.

La frecuencia es la velocidad en la cual el CPU trabaja, cada “tic” del reloj está medido en **CICLOS POR SEGUNDO**, y se expresan en **Hertz** (Hz); en una CPU moderna podemos ejecutar miles de millones de ciclos por segundo, y esta velocidad ya se mide en **gigahertz**.

Los núcleos nos permiten tener a más de un procesador para que ejecute tareas a la vez, y para ello, deben coordinarse.

Los subprocesos o hilos son las cosas que el CPU puede hacer al mismo tiempo.

Por último, la memoria caché nos permite almacenar temporalmente un conjunto de instrucciones que están en la RAM, en una memoria interna del procesador, de modo que no hay que ir a buscar una por una las instrucciones, pudiendo ser más rápido.

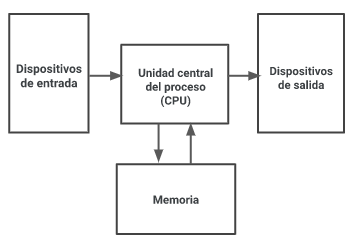
**Analizando el cerebro.**

**¿Quién piensa?**

El **microprocesador** o simplemente **procesador (CPU)** es el cerebro del sistema, justamente procesa todo lo que ocurre en la PC y ejecuta todas las acciones que existen.

**¿Cómo funciona?: Arquitectura de Von Neumann.**

El **procesador** es un dispositivo que **necesita comunicarse con** muchos otros elementos. Lee datos e instrucciones **de la memoria RAM**, requiere información desde **periféricos de entrada** y se comunica con **periféricos de salida** para mostrar los resultados.



**CPU- Características.**

* **Zócalo/ Socket de un CPU**

El **zócalo** de CPU se usa para fijar y conectar el procesador, sin soldarlo, lo cual permite quitar y poner diferentes modelos y familias sin tener que cambiar de placa base.

El **zócalo** realiza todas las comunicaciones con el exterior. Dependiendo del número de conexiones se podrán tener más elementos integrados y ofrecer un mayor ancho de banda hacia los otros componentes del sistema.

**Nota**: los celulares y laptops, como norma general, no utilizan socket.

* **Núcleos/ Cores de un CPU**

Los núcleos son como un **subprocesador en sí mismo**.

Los procesadores de un solo núcleo (single core) no pueden realizar más que una tarea al mismo tiempo.

Una CPU con **dos núcleos** (dual core) sí que podría realizar dos tareas al mismo tiempo, uno de cuatro (quad core), pues cuatro, y así de forma correlativa con tantos núcleos como incorpore.

* **Hilos/ Threads de un CPU**

Los hilos son **el flujo de control de programa**. Ayudan de forma directa a la manera en la que un procesador administra sus tareas.

Su **función** es que los “tiempos de espera” entre procesos se aprovechen mejor.

**Los hilos pueden hacer creer al usuario que sí se puede hacer más de una cosa al mismo tiempo dividiendo la tarea en porciones**, de modo que se alternan porciones de tareas para que parezca que se ejecutan al mismo tiempo.

* **Memoria caché de un CPU**

La **memoria caché** es la memoria más rápida del sistema y se utiliza para acelerar el acceso a memoria de nuestro procesador. **Almacena** los datos e instrucciones más utilizados por el sistema para evitar perder tiempo y tener que acceder a ellos en la RAM.

* **Frecuencia de un CPU**

El procesador está gobernado por un reloj que sincroniza sus componentes y limita las operaciones que es capaz de realizar en una determinada cantidad de tiempo.

La frecuencia es el número de cambios que se hace en un determinado segundo.

**CPU- Rendimiento**

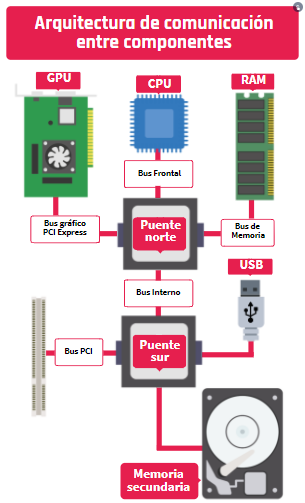
* **La frecuencia no lo es todo**

Hay parámetros más importantes para medir el rendimiento de un procesador en funcionamiento. Aquel al que se le suele dar una mayor importancia es el **IPC** o **I**nstrucciones **P**or **C**iclo.

El **IPC** mide el número de instrucciones que un procesador es capaz de realizar en cada ciclo de su reloj interno.

Mientras que una CPU con una velocidad de reloj más rápida puede completar más ciclos en un segundo, una CPU con un **IPC** más alto, pero con frecuencia más baja, puede ser capaz de completar más tareas en un segundo.

“**Los microprocesadores se están metiendo en todo. En un futuro cercano no habrá ningún accesorio –salvo una escoba, tal vez- que no tenga un procesador dentro**” – Arthur C. Clarke.



* **GPU**: Una **unidad de procesamiento gráfico** es un coprocesador dedicado al procesamiento de gráficos para aligerar la carga de trabajo del procesador central en aplicaciones, como los videojuegos o aplicaciones 3D interactivas.
* **CPU**: La **unidad central de procesamiento** es el hardware dentro de un ordenador u otros dispositivos programables, su trabajo es interpretar las instrucciones de un programa informático.
* **RAM**: En la memoria de acceso aleatorio se cargan todas las instrucciones que ejecuta la CPU y otras unidades del computador, además de contener los datos que manipulan los distintos programas.
* **Puente Norte**: Es el chip que controla las funciones de acceso desde y hasta el CPU, PCI-Express, memoria RAM, vídeo integrado (dependiendo de la placa) y el puente sur.
* **Puente Sur**: Es un chip que se encarga de coordinar los diferentes dispositivos de entrada y salida y algunas otras funcionalidades de baja velocidad. **No está conectado a la CPU** y **se comunica con ella indirectamente** **a través del puente norte**.
* **USB**: El **bus universal** en serie es utilizado como estándar para conexión de periféricos. Se puede conectar con el teclado, el mouse, la memoria USB, el joystick, el escáner, la cámara digital, el celular, el reproductor multimedia, la impresora, el módem, la grabadora de DVD externa, el disco duro externo, entre otros.
* **Memoria Secundario**: Es un tipo de almacenamiento masivo y permanente con mayor capacidad para almacenar datos e información que la memoria primaria (RAM) que es volátil, aunque la memoria secundaria **es de menor velocidad**.

06.04.2021

**Introducción a las memorias**

La memoria de la computadora está dividida en dos partes:

* **Memoria principal**: se prioriza la velocidad sobre el almacenamiento.
* **Memoria secundaria**: lo más importante es la capacidad de almacenamiento.

SIEMPRE que el microprocesador desee ejecutar una operación, PRIMERO debe cargarla dentro de la memoria principal, más precisamente, en la memoria RAM (Random Access memory/Memoria de acceso aleatorio). Todos los datos que se cargan en la RAM están alojados de manera temporal hasta que el procesador los haya ejecutado.

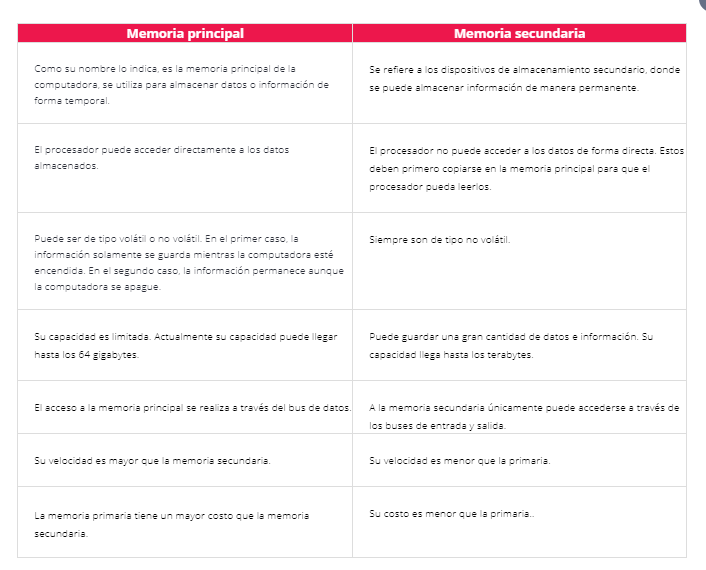
La energía eléctrica juega un papel importante en la memoria principal, ya que al ser una memoria volátil, si pierde energía, todo lo que no haya pasado a memoria secundaria se perderá.

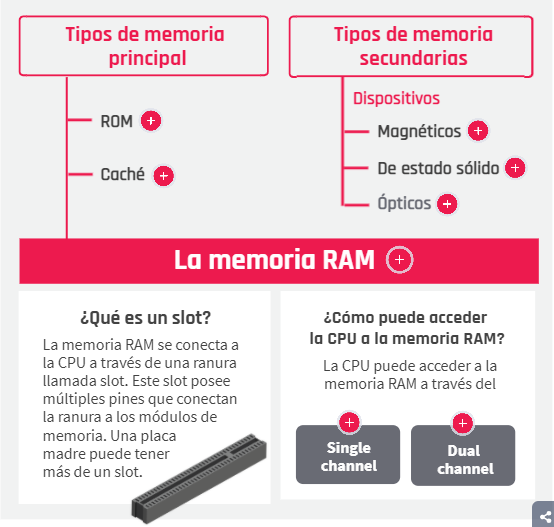
Dentro de la memoria principal existe otro tipo de memoria llamada **Caché**, cuya ventaja es que es LA MÁS VELOZ, pero tiene muy poca capacidad de almacenamiento.

Está también la **Memoria secundaria** cuya principal función es almacenar la información de manera no volátil, es decir: cuando no hay energía, la información sigue existiendo Y NO SE PIERDE. 🡪 **Es el conjunto de dispositivos que complementan el sistema de memoria**.

Para almacenar información, usamos 3 grandes grupos de energía:

* La magnética 🡪 HDD (Hard drive disk) o disco duro.
* La óptica 🡪 Discos: CD, DVD y Blu-ray
* Estado sólido. 🡪 entre las unidades de estado sólido están los Pendrive: pendrive y SSD (solid state disk).

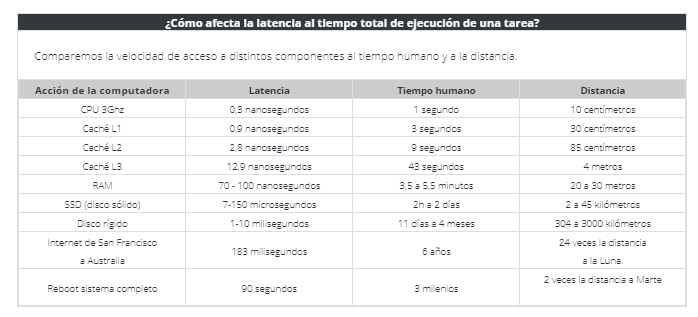




* **ROM**: Es el acrónimo de read only memory o memoria de solo lectura. Como el nombre lo sugiere, solo puede ser leída, no escrita. Guarda las instrucciones necesarias para que la computadora pueda iniciarse.
* **Caché**: La memoria caché se sitúa entre la CPU y la memoria RAM. La CPU copia en ella los datos más relevantes que va a utilizar de la memoria RAM para acceder a ellos más rápidamente.
* **Dispositivos magnéticos**: Es un dispositivo de almacenamiento que emplea un sistema de grabación magnética para almacenar información. Está formado por uno o más discos que giran a velocidad constante. De este tipo son los discos rígidos o disquetes.
* **Dispositivos de estado Sólido**: Es un dispositivo de almacenamiento que no posee partes móviles y que permiten la escritura y lectura en múltiples posiciones en la misma operación mediante pulsos eléctricos. Tipos: discos de estado sólido y memorias.
* **Dispositivos ópticos**: Los datos almacenados en una unidad óptica, pueden ser guardados o leídos a través de un láser. Son dispositivos ópticos los CD y DVD.
* **¿Cómo puede acceder la CPU a la memoria RAM**?
  + **Single channel**: Para el acceso a la información en la RAM se utiliza una única señal a un ancho de banda y frecuencia determinada.
  + **Dual channel**: Permite el acceso simultáneo a dos módulos de memoria. Para ello, todos los módulos de memoria deben tener la misma capacidad, velocidad, frecuencia, latencia y fabricante.



* **Velocidad**: Las computadoras electrónicas digitales no tenían sistema operativo. Los programas, por lo regular, manejaban un bit a la vez, en columnas de switchs mecánicos. Los programas de lenguaje máquina manejaban tarjetas perforadas.
* **Capacidad**: Es la cantidad de datos que se pueden almacenar en una RAM. La capacidad se mide en gigabytes (GB).
* **Latencia**: Es la cantidad de ciclos de reloj que transcurren entre una petición y su respuesta.
* **Voltaje**: El voltaje hace referencia a la energía consumida por el módulo de RAM.
* **Dual channel: ¿Cómo se mide la velocidad y la capacidad en las memorias?**
  + Las velocidades se suman **>** Si la velocidad de cada módulo es de 1600 Mhz, la velocidad total será de 3200 Mhz.
  + La capacidad se suma **>** Si cada módulo tiene una capacidad de 8 GB, la capacidad total será de 16 GB.
* **¿Cómo afecta la latencia al tiempo total de ejecución de una tarea?**
  + Comparemos la velocidad de acceso a distintos componentes al tiempo humano y a la distancia.



**Unidades de Medida**

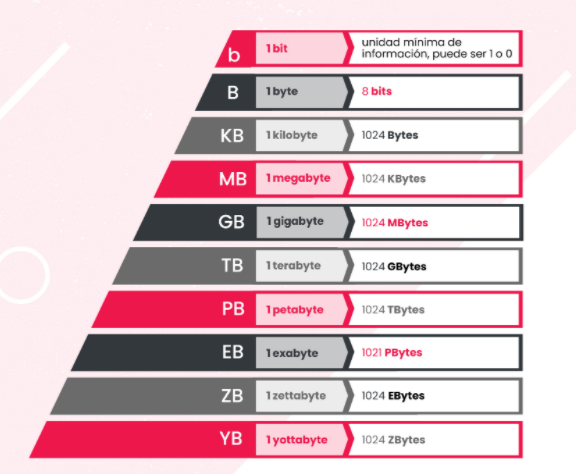
¿La información que almacenamos en la computadora ocupa espacio?

¡Por supuesto! ¡**Todo es medible**!

Desde la cantidad de lluvia que cae hasta la temperatura que sentimos puede medirse. Así también, todo dispositivo de almacenamiento o incluso la memoria principal de la computadora tiene cierto tamaño.

Una fotografía o un vídeo, físicamente, se almacenan en algún lugar, por ejemplo, en la tarjeta de memoria SD de nuestros teléfonos, tabletas o en el disco duro de la computadora.

Si analizamos el caso de la imagen, podemos ver que entre sus propiedades hay un número interesante que aparece allí:



Podemos ver que todos los píxeles y toda la información de la imagen, ocupan 16,8 kilobytes. Hay muchos archivos, particularmente los de vídeo, que ocupan muchos gigabytes de espacio. Un vídeo de buena definición, grabado desde un celular, de aproximadamente una hora, ocupa aproximadamente 2 gigabytes con alta definición. Siempre deberemos tener en cuenta el tamaño que ocupan nuestros archivos porque las capacidades de almacenamiento son limitadas.

**MEMORIA PRINCIPAL**

**¿Qué es la RAM?**

**RAM** es el acrónimo de **random access memory** (memoria de acceso aleatorio). La información almacenada en este tipo de memoria se pierde cuando se desconecta la alimentación del PC o del portátil. Se conoce generalmente como memoria principal o memoria temporal o volátil del sistema informático. Es el lugar donde se almacenan temporalmente tanto los datos como los programas que la CPU está procesando, o va a procesar, en un determinado momento.

**(VIDEO)**

La memoria RAM es la memoria de acceso aleatorio y forma parte de la memoria principal. Es un CIRCUITO INTEGRADO que almacena los datos, programas o información mientras la usamos, y cuando dejamos de usarla, pasa a una MEMORIA SECUNDARIA, liberando espacio ocupado (Cuando no usamos la memoria RAM, usamos la memoria secundaria).

Al ser la RAM aleatoria, puede saber en dónde están los datos e ir DIRECTAMENTE hacia ellos, **pudiendo así acceder a cualquier byte de memoria sin acceder a los bytes precedentes**.

Ahora, **¿por qué el proceso con la memoria RAM es tan veloz?**

A través de los buses envían datos en binario, los cuales se transmiten con una cierta frecuencia.

**¿Cada cuánto tenemos un mensaje? ¿Cada cuánto nuestra RAM tendría que revisar si tiene un mensaje en los buses?**

El que marca el ritmo es un componente electrónico llamado “Reloj” que le dice a la RAM cada CUÁNTO TIEMPO se envían los datos.

**SDRAM**= Memoria de acceso aleatorio dinámica y sincronizada.

Cuando constantemente **le solicitamos** a la memoria RAM **el mismo tipo de datos** e **instrucciones**, la información se almacena en una **MEMORIA INTERMEDIA** conocida como **Memoria caché**. La información en esta memoria, se guarda en niveles de la caché= L1, L2, L3, L4. Cada nivel es más grande que el anterior y pueden o no guardar la misma información que el nivel anterior; esto quiere decir que, cuando el procesador necesita información, comenzará buscando en las memorias más cercanas y rápidas que tenga y, si no lo encuentra, recién en ese momento buscará en la memoria RAM.

* Cuando el CPU necesita un dato, busca en su memoria caché y si no lo encuentra, se lo pide a la memoria RAM.

¿**Por qué si la memoria caché es tan rápida no hacemos una caché más grande**?

Esto se debe, principalmente, a que las memorias caché son demasiado costosas para fabricar, y entre más veloces sean, más costosas son de producir.

También, dentro del procesador, la información se carga en unas celdas diminutas, y la unión de estas forman un **REGISTRO**, el cual es el **primer** y **más pequeño paso** en el eslabón de las memorias y la información. La velocidad del registro es DEMASIADO ALTA, pero su capacidad de almacenamiento es reducida.

* La **comunicación** entre las memorias y demás componentes de la computadora **condiciona el rendimiento**.

¿**Qué sucede cuando una de las memorias o componentes frena el rendimiento de la computadora**?

Se produce un evento conocido como “**cuello de botella**”.

**Registros, caché y memorias RAM**

**Registros**

(es muy veloz pero no tiene mucha capacidad)Un registro es una memoria de muy alta velocidad, que se utiliza en los procesadores para acceder a información importante de manera rápida. La CPU tiene 5 registros internos.

1. PC: Program Counter.
2. IR: Instructins registrer.
3. MAR: Memory adress register.
4. MDR: Memory data register.
5. Accumularor

**Caché de la CPU**

Es un apoyo importante para el procesador que se divide en un total de tres niveles generales al que podemos sumar un cuarto que no resulta nada común.

La diferenciación entre memoria caché **L1**, **L2** y **L3** obedecen a un orden de jerarquía establecido por cercanía al procesador, velocidad y capacidad. El L1 es el nivel con menor capacidad pero mayor velocidad, y es el que está más cerca del CPU; cada nivel puede almacenar o no la misma información que se encuentra en el nivel anterior.

**Tipos de RAM**

**FPM (fast page mode) RAM**

El modo de página rápida es un tipo de memoria RAM que espera durante todo el proceso de localización de un bit de datos por columna y fila; y luego lee el bit antes de comenzar con el siguiente. La velocidad máxima de transferencia es de unos 176 Mbps.

**SDR (singre data rate) RAM**

La memoria RAM SDR es una forma completa de memoria de acceso dinámico síncrono. Tiene tiempos de acceso de entre 25 y 10 nanosegundos (ns) y están en módulos DIMM (módulos de memoria dual en línea) de 168 contactos.

**R (rambus) DRAM**

La memoria dinámica de acceso aleatorio rambus es una forma completa de RDRAM. Este tipo de chips de RAM funciona en paralelo, lo que le permite alcanzar una velocidad de datos de 800 MHz o 1.600 Mbps. Genera mucho más calor al funcionar a tan altas velocidades.

**V (video) RAM**

Es la memoria RAM optimizada para adaptadores de vídeo. Tiene dos puertos para que los datos de vídeo puedan escribirse al mismo tiempo que el adaptador de vídeo lee regularmente la memoria para refrescar la pantalla actual del monitor.

**EDO (extended data output) RAM**

Sus siglas significan en castellano salida de datos extendida. No espera a que finalice el procesamiento del primer bit para continuar con el siguiente. En cuanto se localiza la dirección del primer bit, la EDO RAM comienza a buscar el siguiente.

**DDR RAM**

Lanzada en el año 2000, aunque no empezó a usarse hasta casi 2002. Operaba a 2.5V y 2.6V y su densidad máxima era de 128Mb (por lo que no había módulos con más de 1GB) con una velocidad de 266 MT/s (100-200 MHz).

**DDR2 RAM**

Lanzada hacia 2004, funcionaba a un voltaje de 1.8 voltios, un 28% menos que DDR. Se dobló su densidad máxima hasta los 256 Mb (2GB por módulo). Lógicamente la velocidad máxima también se multiplicó, llegando a 533 MHz.

**DDR3 RAM**

El lanzamiento de esta memoria se produjo en 2007 y supuso toda una revolución porque aquí se implementaron los perfiles XMP. Para empezar los módulos de memoria operaban a 1.5V y 1.65V, con velocidades base de 1066 MHz pero que llegaron mucho más allá, y la densidad llegó hasta a 8GB por módulo.

**DDR4 RAM**

Lanzada en 2014. Se reduce el voltaje hasta 1.05 y 1.2V, aunque muchos módulos operan a 1.35V. La velocidad se ha visto notablemente incrementada, pero su base comenzó en los 2133 MHz. Actualmente ya hay módulos de 32GB, pero esto también se va ampliando poco a poco.

**DDR5 RAM**

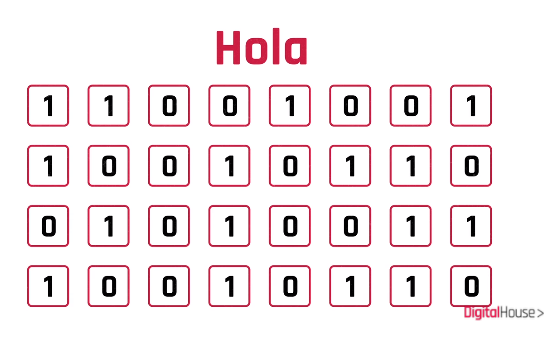
Lanzada a mediados del 2020 llega a anchos de banda de hasta 6.4 Gbps en sus modelos iniciales, **es la primera memoria DDR de doble canal en un solo chip**. Su frecuencia base es de 4800 MHz y, además, su consumo baja por la clásica reducción de voltaje, esta vez a 1.1 V. Su capacidad de almacenamiento máxima en un módulo de memoria es de 128GB.

**MEMORIA SECUNDARIA**

¿**Cómo funciona la memoria de la computadora**?

El dígito binario, o mejor conocido como **BIT** es la mínima unidad de información donde se puede guardar 1 dato; además, sólo puede aceptar dos valores: 1 y 0.

A su vez, los BIT se pueden agrupar en estructuras de 8 celdas conocidas como Byte, que constituyen una unidad direccional de memoria. Esta agrupación ayuda a interpretar lo que es el archivo en sí; por ejemplo, los siguientes octetos o bytes representan un “Hola”.



Al igual que un byte, que es una agrupación de BIT, existen estructuras más grandes, como kilobyte, megabyte, gigabyte, terabyte, etc.

Ahora, retomemos la memoria secundaria.

**MEMORIA SECUNDARIA**

Es la memoria MÁS LENTA, pero la MÁS SEGURA a la hora de almacenar información. En sus inicios, la memoria secundaria era conocida como memoria ROM (read only memory), ya que su función era la de contener información que no podía modificarse (o sea, contenía archivos SÓLO DE LECTURA).

Hoy, la memoria secundaria sigue trabajando bajo este concepto, pero con el avance de la tecnología se pudo borrar o sobre-escribir información que tenía guardada. Sin embargo, sigue siendo muy costoso en cuestión de tiempo para el procesador, en comparación con la memoria primaria.

* Para el procesador es más rápido localizar la información en la memoria principal que en la secundaria.
* Usar la memoria secundaria solamente cuando sea necesario.

Es por eso que se evita usar la memoria secundaria a menos que sea necesario.

**DENTRO** de las memorias secundarias, existen **3 tipos principales de tecnologías** que nos permiten guardar información a largo plazo:

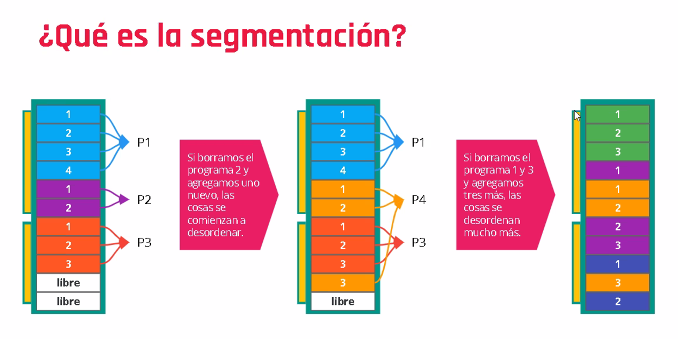
* **Magnética** = en los **almacenamientos magnéticos**, los datos se guardan según un **patrón magnético**, en un **disco giratorio** el cual está recubierto a su vez pro una **membrana magnética**. Esto hace que sean las más baratas de construir, aunque son las más lentas debido a sus limitaciones físicas.
* **Óptica** = en el caso de los almacenamientos ópticos, los **BIT** se codifican como **puntos de luz** y **puntos sin luz**, elevando de esta forma la velocidad de lectura, aunque sean limitados por su capacidad de almacenamiento.
* **De tipo sólido =** inspiradas en las memorias RAM, las tecnologías de tipo almacenamiento sólido son memorias que trabajan a través de **transistores que atrapan o elimina cargas eléctricas** dentro de su estructura, son las **MÁS VELOCES** pero las **MÁS COSTOSAS** de fabricar.

Los dispositivos de memoria, al ser físicos, tienden a desgastarse, por lo cual siempre es importante mantener la información más importante respaldada y segura ante cualquier eventualidad.



* **Magnético**:
  + **Cinta magnética**: Es un tipo de medio o soporte de almacenamiento de datos que se graba en pistas sobre una banda plástica con un material magnetizado, generalmente óxido de hierro o algún cromato. El tipo de información que se puede almacenar en las cintas magnéticas es variado, puede ser vídeo, audio o datos.
  + **Diskette**: Dispositivo de almacenamiento utilizado para transportar información de una PC a otra, su capacidad podía llegar hasta 2,88 Mb. Los más utilizados eran los de 3 1/2 —llamados así debido a su apariencia física—. Destacaban los discos ZIP. Eran muy utilizados hasta la aparición de la memoria flash.
  + **Discos duros**: El disco duro está formado por uno o varios platos rígidos introducidos en una caja hermética y unidos por un eje común que gira a gran velocidad. Sobre cada uno de los patos, que normalmente tienen sus dos caras destinadas al almacenamiento, se sitúan sendos cabezales de lectura/escritura.
* **Óptico**:
  + **CD**: El disco compacto (compact disc) es un medio óptico que se usa para almacenar datos en formato digital, ya sean imágenes, vídeos, audio, documentos, como otros datos. En un principio esta tecnología fue usada para el CD audio, pero más tarde se expandió y adaptó para el almacenamiento de datos (lo que conocemos como CD-ROM), de video (conocido como VCD Y SVCD), la grabación doméstica (llamada CD-R y CD-RW). El CD puede almacenar hasta 80 minutos de audio o, lo que es igual, 700 MB de datos.
  + **DVD**: Significa "disco digital versátil". Es un disco óptico capaz de almacenar contenidos de medios. Los DVDs vienen en múltiples tipos y capacidades de almacenamiento; pueden tener uno o dos lados, una sola capa o dos capas, todas dictando la cantidad de contenidos de medios que el DVD puede almacenar. Las capacidades de almacenamiento de los DVDs van desde 1,46 GB en un DVD de un solo lado y una capa a 17,08 GB en un DVD de dos lados y dos capas. Las variaciones de DVD también consisten en DVD-R, DVD+R, DVD-RW, DVD+RW y DVD-Ram que describen la manera en la que el contenido de medios se almacena en el disco. DVD-R y DVD+R son capaces de ser escritos con datos (audio, video, entre otros) solamente una vez, mientras que DVD-RW, DVD+RW y DVD-Ram son capaces de ser escritos, borrados y reescritos múltiples veces.
  + **Blu**-**ray**: Es un formato de disco óptico, una evolución del CD y el DVD. Al igual que estos, tiene el mismo tamaño y aspecto externo, pero multiplica la capacidad del disco. En un Blu-ray de una sola capa podemos almacenar unos 25 GB de información. En un volumen como este pueden caber unos 27.000 minutos de música en formato MP3. Esto en una sola capa porque otra de las virtudes más interesantes de este formato es que puede admitir varias, multiplicando su capacidad. Así­, podemos encontrar discos Blu-ray de hasta 100 GB de capacidad.
* **Sólido**:
  + **Flash**: Es un dispositivo en forma de tarjeta, que se encuentra orientado a realizar el almacenamiento de grandes cantidades de datos en un espacio reducido, **permitiendo la lectura y escritura** de múltiples posiciones de memoria en la misma operación. Todo esto gracias a **impulsos eléctricos**.
  + **Pendrive**: Es un dispositivo portátil de almacenamiento, compuesto por una memoria flash, accesible a través de un puerto USB. Su capacidad varía según el modelo, y en la actualidad podemos encontrar en el mercado pendrives con una capacidad de hasta 256 Gb en un mínimo espacio. Es considerado la sucesión de los viejos diskettes dada su gran capacidad de almacenamiento y compatibilidad con diferentes dispositivos.
  + **SSD**: Es un dispositivo que almacena datos. Su nombre significa **disco de estado sólido**, haciendo alusión a dispositivos que **no tienen ni un solo movimiento mecánico en su interior**, al contrario que los HDD. Los SSD de hoy en día utilizan el bus SATA o el PCIe del ordenador (discos ssd M2), siendo los últimos más rápidos que los primeros dado que un SSD normal encuentra un cuello de botella en el bus SATA ya que un SSD ofrece velocidades superiores a las que ofrece el bus SATA 3.

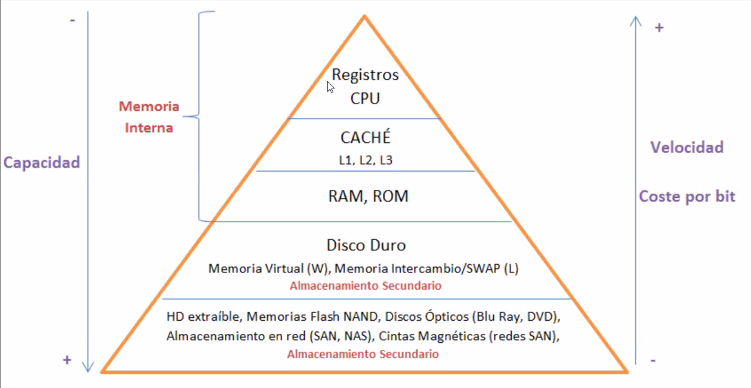
**CLASE EN VIVO**



En un principio, cuando tengo todo libre y cargo dos programas (P1, P2, P3),

La **latencia** es el tiempo que se tarda en hacer una acción y recibir la respuesta, y la **frecuencia** es la velocidad a la que trabajan las memorias.

Los **registros** 🡪 son memorias muy pequeñas que tiene el procesador (CPU) para hacer tareas, y **son las MÁS RÁPIDAS DE TODAS LAS MEMORIAS**.



La **memoria caché** está también ubicada en el CPU, las más comunes son las L1, L2 y L3. Es muy raro, pero existen las L4.

08.04.2021

**SISTEMAS OPERATIVOS**

**Funcionalidades y administración**

¿**Qué es un sistema operativo**?

“**El sistema operativo es el soporte lógico que controla el funcionamiento del equipo físico**”

“**Desde el punto de vista usuario, es un conjunto de programas y funciones que ocultan los detalles del hardware, ofreciendo al usuario una vía sencilla y flexible de acceso al mismo**”

Cuando hablamos de sistema operativo, nos referimos a un software que comprende un conjunto de programas (**software de comunicación usuario-dispositivo**). Además, **administra los recursos ofrecidos por el hardware** (**controlador de hardware de sistema**) y actúa como un intermediario entre la computadora y su usuario, ofreciéndole un ambiente amigable y sencillo de interpretar.

Un sistema operativo comienza a funcionar EN EL MOMENTO QUE ENCENDEMOS NUESTRO DISPOSITIVO, y deja de funcionar cuando lo apagamos.

Podemos encontrar a los sistemas operativos en cualquier dispositivo que use un microprocesador para funcionar, por ejemplo: celulares, computadoras, heladeras inteligentes, etc.

Por ejemplo, cuando accedemos a Instagram desde nuestros celulares, estamos usando la interfaz del sistema operativo, porque este es quien le da las órdenes al procesador para que lo ejecute. El procesador es informado de algo tan sencillo como “mové el mouse” u oprimir una tecla, todo por medio del sistema operativo.

**Los S.O. administran dispositivos, recursos y procesos** 🡪 o sea, no sólo envían las órdenes al CPU, si no que administra dispositivos de entrada y salida, la cola de procesos y los recursos del dispositivo.

**Recursos administrados por el sistema operativo:**

* Gestionar la **memoria** de acceso aleatoria (RAM) y ejecutar las aplicaciones, designando los recursos necesarios.
* Administrar la **CPU**, gracias al algoritmo de programación.
* Direccionar las **entradas y salidas** de datos (a través de ***drives***), por medio de los periféricos de entrada y salida.
* Administrar **la** **información** para el buen funcionamiento de la PC.
* Dirigir **las autorizaciones** de uso para el usuario.
* Administrar **los archivos**.

**¿Qué sucede con los servidores?**

En ellos también encontramos sistemas operativos, mayormente heredados del sistema operativo **UNIX**, como **Red Hat**, y también podemos encontrar el S.O. **Windows Server**, diseñado ESPECÍFICAMENTE para estas grandes computadoras; **Debian**.

¿En qué se diferencian los S.O. de servidores y los S.O.?

En su uso. Los S.O. de servidores son MULTI-USUARIOS, o sea, varios usuarios están conectados al mismo tiempo, trabajando sobre el mismo núcleo; en cambio, en las computadoras domésticas, tienden a ser MONOUSUARIOS.

* **Servidores**: multiusuarios
* **Domésticas**: monousuarios.

A menudo, vemos en nuestros dispositivos la solicitud de actualizar el sistema operativo, una tarea que se vuelve tediosa por la brevedad del tiempo en el que lo requiere, y hasta a veces la cantidad de tiempo que toma la actualización. Sin embargo, **recordemos que los software están diseñados para facilitar nuestra experiencia como usuarios**, y es por eso que se encuentran en constante mejoría.

**GENERACIONES DE SISTEMAS OPERATIVOS:**

* **Generación 0 (década de 1940):** Las computadoras electrónicas digitales no tenían sistema operativo. Los programas, por lo regular, manejaban un bit a la vez, en columnas de switchs mecánicos. Los programas de lenguaje máquina manejaban tarjetas perforadas.
* **Primera generación (1945-1955**) 🡪 **Tubos de vacío y tableros enchufables =** Se lograron construir máquinas calculadoras usando tubos de vacío. Estas máquinas eran enormes y ocupaban cuartos enteros con decenas de miles de tubos de vacío, pero eran mucho más lentas que incluso las computadoras personales más baratas de la actualidad. Toda la programación se realizaba en lenguaje de máquina absoluto.
* **Segunda generación (1955-1965) 🡪 Transitores y sistemas de lote:**
  + Estas máquinas se encerraban en cuartos de computadora con acondicionamiento de aire especial. Para ejecutar un programa, un programador escribía primero el programa en papel (en FORTRAN o ensamblador) y luego lo perforaba en tarjetas. Después, llevaba el grupo de tarjetas al cuarto de entrada y lo entregaba a uno de los operadores. Cuando la computadora terminaba el trabajo que estaba ejecutando en ese momento, se separaba la salida impresa y se llevaba al cuarto de salida donde el programador podía buscarla. Luego, el operador tomaba uno de los grupos de tarjeta traídos del cuarto de entrada y lo introducía en el lector. Si se requería el compilador de FORTRAN, el operador tenía que traerlo de un archivero e introducirlo en el lector.
  + Dado el alto costo del equipo, la solución que se adoptó generalmente fue el **sistema por lotes**. El principio de este modo de operación consistía en juntar una serie de trabajos en el cuarto de entrada, leerlos y grabarlos en una cinta magnética usando una computadora pequeña y (relativamente) económica.
  + Después de cerca de una hora de reunir un lote de trabajos, la cinta se rebobinaba y se llevaba al cuarto de la máquina, donde se montaba en una unidad de cinta. El operador cargaba entonces un programa especial, que leía el primer trabajo de la cinta y lo ejecutaba. La salida se escribía en una segunda cinta, en lugar de imprimirse. Cada vez que terminaba un trabajo, el sistema operativo leía automáticamente el siguiente trabajo de la cinta y comenzaba a ejecutarlo.
* **Tercera generación (1965-1970) 🡪 Circuitos integrados ( CI ) y multiprogramación=**
  + Las máquinas diferían solo en el precio y el rendimiento (memoria máxima, velocidad del procesador, número de dispositivos de E/S permitidos, entre otros). IBM trató de resolver simultáneamente ambos problemas introduciendo la System/360, puesto que todas las máquinas tenían la misma arquitectura y conjunto de instrucciones, los programas escritos para una máquina podían ejecutarse en todas las demás, al menos en teoría.
  + Los 360 y los sistemas operativos de tercera generación parecidos a él producidos por otros fabricantes de computadoras lograron satisfacer a sus clientes en un grado razonable y también popularizaron varias técnicas clave que no existían en los sistemas operativos de la segunda generación. Tal vez la más importante de ellas haya sido la **multiprogramación**.
  + El problema era el tiempo de espera, **la solución a la que se llegó fue dividir la memoria en varias secciones**, con un trabajo distinto en cada partición. Mientras un trabajo estaba esperando que terminara su E/S, otro podía estar usando la CPU. Si se podían tener en la memoria principal suficientes trabajos a la vez, la CPU podía mantenerse ocupada casi todo el tiempo. También, tenían la capacidad de leer trabajos de las tarjetas al disco tan pronto como se llevaban al cuarto de computadoras. Luego, cada vez que un trabajo terminaba su ejecución, el sistema operativo podía cargar uno nuevo del disco en la partición que había quedado vacía y ejecutarlo.
* **Cuarta generación (1980-a nuestros días) 🡪 Computadoras personales=**
  + Con la invención de los circuitos integrados a gran escala (LSI), chips que contienen miles de transistores en un cm2 de **silicio**, nació la era de la **computadora personal**.
  + Dos sistemas operativos dominaron inicialmente el campo de las computadoras personales y las estaciones de trabajo: **MS-DOS de Microsoft** y **UNIX**. MS-DOS se usaba ampliamente en la IBM PC y otras máquinas basadas en la CPU Intel 8088 y sus sucesoras. Más tarde, la Pentium y Pentium Pro. Aunque la versión inicial de MS-DOS era relativamente primitiva, versiones subsecuentes han incluido características más avanzadas, muchas de ellas tomadas de UNIX. El sucesor de Microsoft para MS-DOS, **Windows**, originalmente se ejecutaba encima de MS-DOS, pero a partir de **1995** se produjo una versión autosuficiente de **WINDOWS**.
  + El otro competidor importante es **UNIX**, que domina en las estaciones de trabajo y otras computadoras del extremo alto, como **los servidores de red**. UNIX es popular sobre todo en máquinas basadas en chips **RISC** de alto rendimiento.

**CLASIFICACIÓN DE LOS SISTEMAS OPERATIVOS**

No todos los hardware están en la capacidad de soportar o requerir el mismo S.O.; por ejemplo, una computadora con menos de 4GB de RAM, difícilmente soportará un sistema operativo de 64 bits.

Para poder clasificar los S.O., debemos buscar todos aquellos que guarden similitudes o relación entre sí:

* Gestión de usuarios
* Gestión de tareas
* Gestión de recursos
* Su estructura interna.

**TIPOS DE SISTEMA OPERATIVO:**

Varían según el hardware y la función de cada dispositivo.

**Según el usuario pueden ser**:

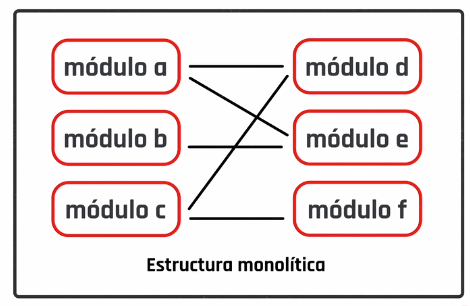
* **MULTI-USUARIO**: Sistema operativo que permite que varios usuarios ejecuten simultáneamente sus programas. Puede ser por medio de terminales conectadas a la computadora o por sesiones remotas en una red de comunicaciones. EJEMPLOS: **UNIX**, **LINUX**, **SOLARIS**, **MAC OSX**, **Windows (a partir de XP)**
* **MONOUSUARIO**: Sistema operativo que solamente permite ejecutar los programas de **un usuario a la vez**. Ejemplo: todas las versiones de Windows para computadoras domésticas. **Windows** (hasta Me), **DOS**.

**Según la gestión de tareas puede ser**:

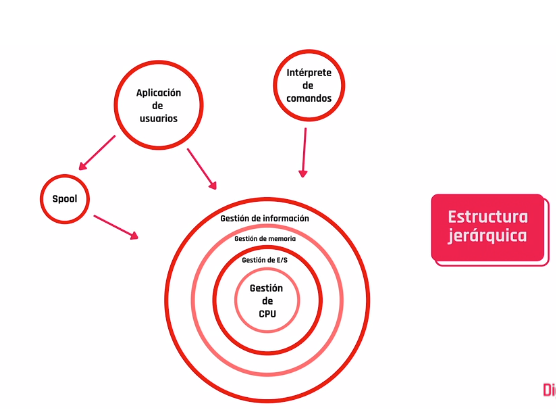
* **MULTITAREA**: sistema operativo que puede ejecutar varios procesos al mismo tiempo. Son los más comunes, y los dispositivos actuales suelen tener estos S.O. **Windows, Linux, Unix, MAC OSX.**
* **MONOTAREA**: sistema operativo que solamente **permite ejecutar un proceso a la vez**. Una tarea a la vez. Estos son los S.O. más primitivos, como **Windows Me** y **Windows Vista (DOS)**. Ejemplo: si queremos imprimir un archivo en estos S.O., no vamos a poder hacer ninguna otra tarea hasta que termine con la impresión y reciba otra instrucción.

**Según su estructura interna puede ser**:

* **MONOLÍTICA**: constituido por **UN SOLO PROGRAMA**, compuesto de **UNA serie de rutinas entrelazadas** entre sí de tal forma que pueden comunicarse entre ellas. Estos sistemas operativos suelen estar hechos a medida, por lo que son muy rápidos, pero no tienen la flexibilidad para soportar diferentes tipos de aplicaciones. Ejemplos: **VMS, Linux, Multics, Windows (hasta Me)**

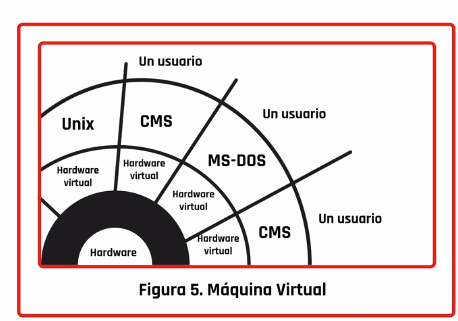


* **ESTRUCTURA JERÁRQUICA**: a medida que fueron creciendo las necesidades de los usuarios y se perfeccionaron los sistemas, se hizo necesaria una mayor organización del software del S.O., donde una parte del sistema contenía sus partes, y esto organizado en forma de niveles. Subdivide en forma de capas o anillos perfectamente definidos y con una clara interfaz con respecto al resto de los recursos. Ejemplos: **UNIX, Multics**.

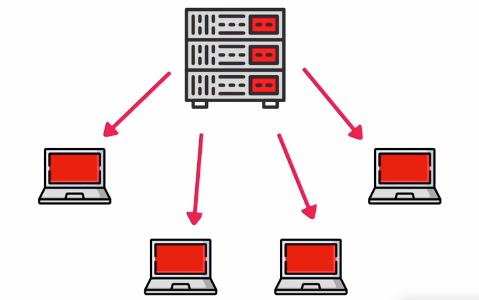


* **MÁQUINA VIRTUAL**: separan dos conceptos que suelen estar unidos en el resto de los sistemas
  + **MULTIPROGRAMACIÓN**
  + **LA MÁQUINA EXTENDIDA.**

El **objetivo** de los S.O. de máquina virtual es el de integrar distintos S.O., dando la sensación de ser varias máquinas diferentes. Ejemplos: **Microsoft Hyper-V, VMware, VirtualBox, QEMU, Kernel-Based Virtual Machine**.



* **CLIENTE-SERVIDOR**: es el más reciente/nuevo de todos. Sirve para **TODA CLASE DE APLICACIONES**, por lo tanto, **es de propósito general** y cumple con las mismas actividades que los sistemas operativos convencionales (**sirve para todas las aplicaciones**). La idea es mantener la visión que tiene UN USUARIO de una computadora personal, pero la red le permite compartir el espacio del disco o la impresora, con **el fin de ECONOMIZAR LOS RECURSOS**. El problema que tienen es que **NO RESUELVEN LOS PROBLEMAS DE COMPARTIR INFORMACIÓN**, lo que **dificulta el desarrollo en grupo**.



**Según la gestión de recursos puede ser**:

* **CENTRALIZADO**: sistema operativo solo permite utilizar los recursos de un solo ordenador.
* **DISTRIBUIDO**: sistema operativo que permite ejecutar los procesos de más de un ordenador al mismo tiempo.

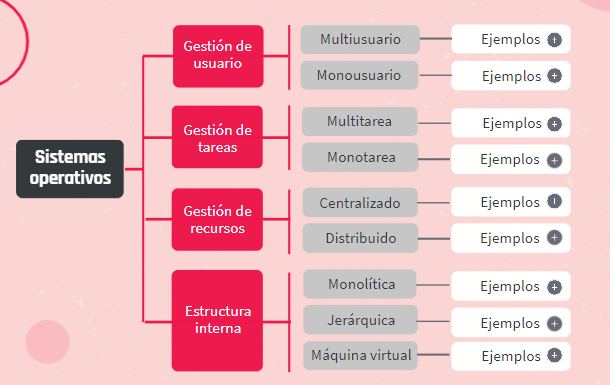
**Sistemas operativos según su tipo de licencia 🡪** En este caso, se dividen en =

* **OPEN SOURCE:** permiten usar, modificar y adaptar un S.O. a voluntad de usuario; ejemplos de estos son UBUNTU y Red Hat.
* **PROPRIETARY SOFTWARE:** son de propietarios y NO PERMITEN MODIFICACIONES, tal como **Windows** (limitaciones del propietario).

**¿Qué tipo de sistema operativo se debe elegir?**

El SO lo podemos elegir en base a nuestras necesidades.

**Ejemplo**: si queremos una computadora para trabajar desde casa, lo más recomendable sería un SO MONOUSUARIO, en vez de uno multiusuario.



**LLAMADAS AL SISTEMA**

**KERNEL = EL CEREBRO DEL SISTEMA OPERATIVO**

El kernel es una parte del sistema operativo encargado de interactuar entre las diferentes aplicaciones y sus necesidades con los recursos de los que dispone el dispositivo para ejecutarlos.

Ejemplo: cuando hacemos clic en “Guardar” un documento, es el kernel quien se encarga de de interactuar con la memoria secundaria para guardar la información.

También es el kernel el que **asigna y prioriza recursos**; decide cuándo asignar o quitar recursos de hardware a las aplicaciones que se ejecutan en el software. Como así también, **asignar prioridades según las necesidades del SO**.

¿**Cómo se llevan a cabo estas interacciones**?

**A través de las llamadas al sistema.**

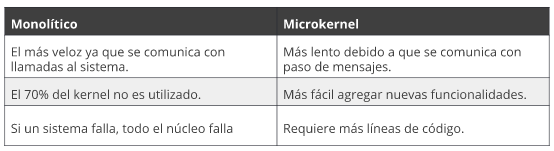
**Kernel y llamadas al sistema**

* El **kernel** es la capa fundamental de un sistema operativo, es el **encargado** de **comunicar** y **administrar** los recursos de la computadora, como la RAM o el uso del procesador.

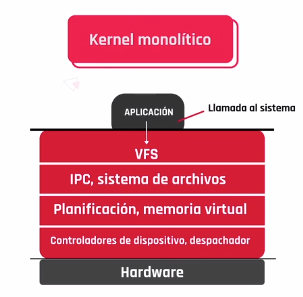
El kernel se encuentra alojado dentro del SO, y es una parte ESCENCIAL del mismo. Windows, Linux y MAC tienen sus propios kernel, aunque hay otros SO que utilizan o se basan en kernels ya creados, como es el caso de Android y el kernel de Linux

**Tipos de Kernel.**

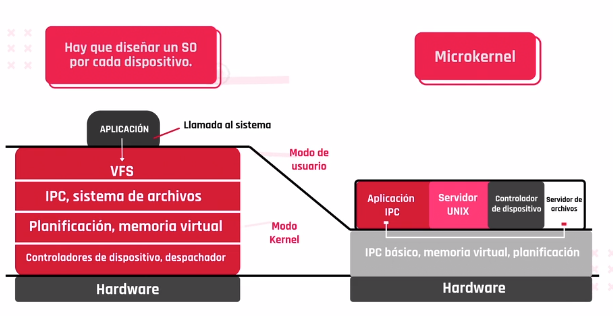
Existen diversos tipos de estructuras de un kernel, pero **dos ramas** entre las cuales se divide su enfoque que son las siguientes:



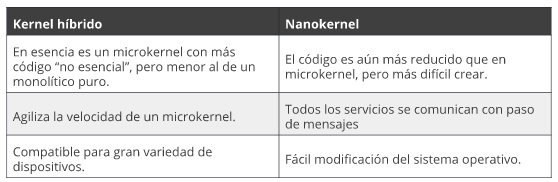
* El **monolótico** es un código de muchas líneas que están alojadas en un solo espacio de memoria, y posee todos los drivers, los servicios, y los métodos de administración de recursos.
* **DESVENTAJA** = se desperdicia MUCHO ESPACIO EN MEMORIA, porque cuando se carga el kernel completo se cargan drivers y diferentes métodos para todo tipo de dispositivos que el SO puede operar.



* La **CONTRAPROPUESTA del diseño monolítico** es el **MICROKERNEL**.
  + Sólo posee las instrucciones básicas de administración en un pequeño espacio de memoria, y deja a los dispositivos su propio manejo.
  + Se encarga de las tareas MÁS BÁSICAS DE ADMINISTRACIÓN.
  + ¡Muy bien! Siempre y cuando el hardware sea de las mismas características podrá funcionar correctamente. 🡪 o sea, el microkernel de una PC puede usarse en una notebook.
  + **DESVENTAJA**: pertenece ÚNICAMENTE a UN dispositivo, y al ser el kernel parte fundamental del SO, hay que diseñar un SO por cada dispositivo con su microkernel determinado.



A su vez, también existen variantes sobre las ramas de kernel, los cuales buscan explotar más aún sus fortalezas y mejorar sus debilidades:

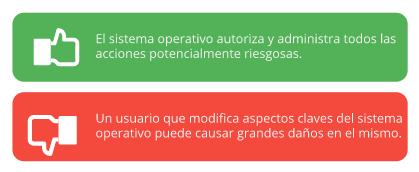


**Llamadas al sistema**

* Las **llamadas al sistema** son la manera en la cual un programa **solicita** una acción al sistema operativo con el que interactúa.
* Esta acción es el **punto de enlace** entre el modo **usuario** y el modo **privilegiado** del sistema operativo. Lo que permite a las aplicaciones **utilizar** recursos de hardware.

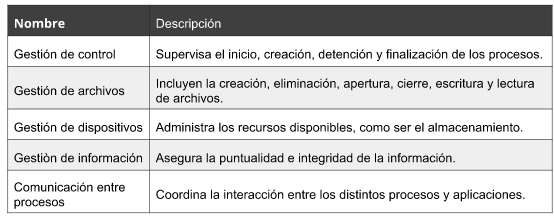
**Objetivo de las llamadas al sistema**:

Es la forma en la que los sistemas operativos diferencian las acciones que puede realizar un usuario (modo usuario) de las que no ya que algunas pueden llegar a ser muy dañinas para el sistema operativo que solo debe controlarlas el mismo (modo privilegiado) 🡪 EN EL MODO PRIVILEGIADO SE PUEDEN HACER COSAS MUY DAÑINAS



**Clasificación de llamadas al sistema**

Si bien todas trabajan como unidades de control para el sistema operativo, se establecieron cinco tipos de llamadas al sistema:



* **CUANDO UN DISPOSITIVO O PRECESO FALLA** = El Kernel detiene TODO LO QUE ESTÁ HACIENDO LA COMPUTADORA, para evitar daños en el SO.
* **TODAS LAS ACCIONES DE UNA COMPUTADORA PASAN POR UN KERNEL, y la performance de nuestro sistema operativo depende de un kernel.**

12.04.2021

**INTRODUCCIÓN A PROCESOS**

Dentro de las operaciones más básicas, pero también más complejas, de nuestras computadoras, se encuentran los procesos.

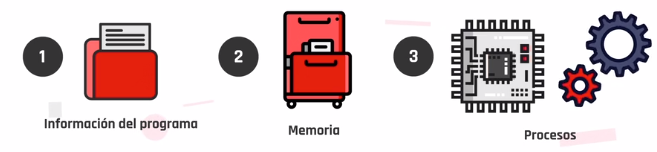
Siempre que queramos que nuestra computadora realice alguna **instrucción**, son los **procesos** los que asumen el trabajo, para que así, la **CPU**, **ejecute la tarea** por medio del **plan de acción diseñado** por el **sistema operativo**.



**¿Qué son los procesos?**

**Procesos**= Ejecución de un programa o instrucción.

* Un proceso es un **programa en ejecución** junto con el entorno asociado (registros, variables, entre otros).



Cuando la información que tiene **ESE PROGRAMA** se carga en la **MEMORIA** y se pone en **EJECUCIÓN**, se le denomina **proceso**.

**TODOS** los softwares ejecutables se organizan en procesos que quiere utilizar la CPU, y es el sistema operativo quien organiza el orden en que se van ejecutando esos procesos.

* **Los softwares ejecutables se organizan en procesos**.
* **El SO organiza el orden de ejecución de los procesos**.

A este **cambio de proceso** lo llamamos “**Cambio de contexto**”. 🡪 las acciones que el sistema operativo realiza para cambiar el proceso A por el B se denominan cambio de proceso.

Y, aunque creamos que todos los procesos se están ejecutando al mismo tiempo, lo cierto es que:

* **Los procesos se ejecutan UNO A LA VEZ**.
* **LOS PROCESOS no pueden almacenarse en la memoria principal** 🡪 esto es porque consumen espacio y, a la larga, llegarían a llenar la memoria RAM. Por esto:
* **LOS PROCESOS SON EFÍMEROS 🡪** se crean y se terminan.

**CREACIÓN DE LOS PROCESOS:**

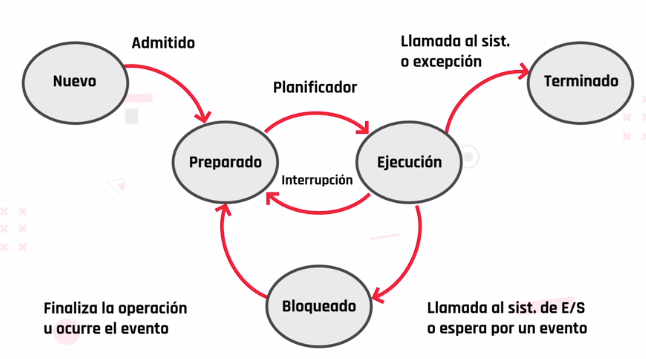
* **PROCESO DE MANERA INTERACTIVA**: con el usuario. Por ejemplo, cuando exportamos un archivo, estamos generando un proceso.
* **PROCESO DE LLAMADOS AL SO**: se crean en segundo plano, conocidos como LLAMADOS AL SISTEMA OPERATIVO (de la clase de SO). Ejemplo: cuando el software no puede acceder directamente a un recurso, le solicita al sistema operativo que lo gestione.

**TERMINACIÓN DE LOS PROCESOS:**

Para que un proceso termine, tiene que pasar por algún estado que determine su condición.

Estos estados son:

* **Nuevo** = cuando el proceso se crea.
* **Listo** = pasa a listo cuando el sistema operativo los carga en la memoria (“preparado”)
* **Ejecución** = cuando se comienza a ejecutar, entramos en el estado de ejecución. De aquí, hay varios destinos posibles para el proceso…
* **Bloqueado** = llamada al sistema de E/S o espera por un evento. Cambiar el contexto, que sea suspendido, cuando hacemos un llamado para leer el archivo.
* **Salida** = finalmente, cuando el proceso es ejecutado y cumple con su objetivo, pasa al estado de salida, dando lugar a que otro proceso sea ejecutado.



Existen varios **mecanismos de comunicación entre procesos o IPC** (Inter Process).

* Uno de ellos, son las **señales** 🡪 son avisos que puede enviar un proceso a otro. Luego, el SO se encarga de que el proceso que recibe la señal, tome una acción para gestionarla.
* **Memoria compartida** 🡪 recurso para que los software puedan intercambiar información. Sería un recurso compartido a disposición del software para que puedan intercambiar información.
* Cuando es un proceso que no puede resolverse instantáneamente, como cuando ocurre una llamada al sistema, se van a crear otros procesos denominados **HIJOS** 🡪 El proceso hijo realiza subtareas del proceso padre, para que el proceso padre pueda cumplir su objetivo. Los procesos padres pueden tener varios procesos hijos, pero los procesos hijos sólo pueden tener un proceso padre.

**COMUNICACIÓN DE PROCESOS**

Básicamente, existen **dos tipos de procesos** que se ejecutan de manera concurrente:

* **Procesos independientes**.
  + Tienen total y completa autonomía 🡪 **AUTÓNOMOS**. No pueden ser afectados ni afectar a otros procesos que estén siendo ejecutados en el sistema.
  + TAMPOCO SE PUEDEN COMUNICAR CON LOS OTROS PROCESOS
* **Procesos cooperativos**. 🡪 **PUEDEN AFECTAR Y SER AFECTADOS**. Cualquier proceso que comparta cualquier tipo datos o de recursos con otros procesos es considerado cooperativo. ¿**Por qué deberíamos realizar una cooperación entre procesos**?

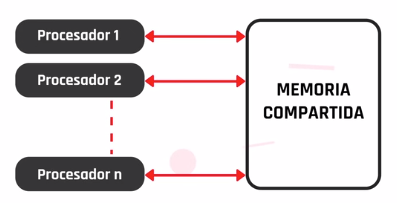
**MOTIVOS** =

* + **Información compartida**: algunos procesos carecen de información, entonces deben consultarla para poder ejecutarse. Por ello, la información debe ser compartida.
  + **CPU eficiente y veloz**: gracias a que la información es compartida, el CPU trabaja de una manera más eficiente y veloz. Esto da como resultado la **MODULARIDAD**.
    - **Modularidad:** Ejecución independiente y simultánea de varios pasos de una tarea. Cuando una tarea contiene varios pasos, el CPU puede ejecutarlos de manera independiente y simultánea.
    - **¿Puede este tipo de comunicación traernos problemas?** SÍ, siempre y cuando un proceso ejecute una tarea de forma errónea o no haya planificación.

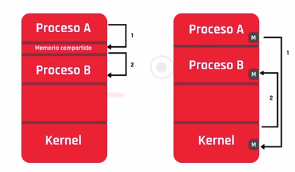
¿Alguna vez enviaste un archivo a imprimir pero se generó un error? Sucede porque el recurso se encuentra bloqueado (**RECURSO BLOQUEADO**) por otro proceso que se está ejecutando.

Para que los **procesos se comuniquen entre sí**, existen **DOS MÉTEDOS DE INTERCOMUNICACIÓN/PROCESOS COOPERATIVOS, o IPC** (Inter-Process Communication), que constan de dos modelos:

* **La memoria compartida**: en este modelo, **se establece un espacio en memoria que será compartido por los procesos**. Imaginemos que escribimos un resumen en una hoja, y le prestamos nuestro resumen para que lo fotocopie y no modifique nuestro resumen; a su vez, este compañero realiza sus propios apuntes que son fotocopiados por otros compañeros, y de esta forma se van compartiendo los recursos entre todos para que cada uno pueda completar su objetivo.
  + **En el modelo de memoria compartida se establece un espacio de memoria que será compartido por los procesos.**



* **Los pasos de mensajes**: imaginemos que nuestro resumen quedó en la fotocopiadora, y a otro compañero le resulta interesante; entonces, le pide al encargado si puede realizar una copia; ese encargado, a su vez, se contacta con nosotros y nos pide permiso, y nosotros accedemos. Aquí, el encargado de la fotocopiadora es un **INTERMEDIARIO** entre ambos procesos. En los SO, esta tarea es realizada por el **KERNEL**.
  + **Los procesos no comparten memoria**
  + **La comunicación se hace mediante operaciones explícitas de envío y recepción.**



Estos procesos poseen sus propias ventajas y desventajas:

* La **memoria compartida** es generalmente más económica que usar un multiprocesador.
* En el **paso de mensajes,** no existen los errores como exclusión mutua, y son compatibles con cualquier tipo de arquitectura de computadora.

Al elegir uno u otro, es importante considerar qué tipo de trabajo vamos a realizar y de qué características disponemos.

**SINCRONIZACIÓN DE PROCESOS**

**La sincronización de procesos permite que mientras un proceso está escribiendo un registro, otro proceso no pueda leer**.

Los procesos pueden transitar por diferentes estados, los cuales indican en qué parte de su ciclo de vida se encuentra el mismo, y, en base a este, el SO toma decisiones sobre él:

Los estados son:

* **Nuevo** = cuando el proceso se crea.
* **Listo** = pasa a listo cuando el sistema operativo los carga en la memoria (“preparado”)
* **Ejecución** = cuando se comienza a ejecutar, entramos en el estado de ejecución. De aquí, hay varios destinos posibles para el proceso…
* **Bloqueado** = llamada al sistema de E/S o espera por un evento. Cambiar el contexto, que sea suspendido, cuando hacemos un llamado para leer el archivo. En la espera de que un proceso o recurso pueda ser utilizado.
* **Salida** = finalmente, cuando el proceso es ejecutado y cumple con su objetivo, pasa al estado de salida, dando lugar a que otro proceso sea ejecutado.
* **Terminado** = ha sido ejecutado y su ciclo de vida termina.

Existe una herramienta para la **sincronización de procesos**, en base a estos estados = **LOS SEMÁFOROS**.

Mientras que un proceso se está ejecutando, y aparece una llamada de **Espera**, pasa a una lista de **Bloqueados**, y permanece ahí hasta que un proceso diferente le envía la señal de **avance**, y el proceso que permanecía Bloqueado se coloca en una fila de espera para utilizar el CPU.

Existen **2 tipos de procesos**:

* **INDEPENDIENTES**: trabajan por ellos mismos, sin requerir ayuda de nadie.
* **COOPERATIVOS**: trabajan en función y disponibilidad de otros recursos.

**Área crítica**: en una analogía de un negocio, tenemos a los empleados que trabajan independientes de los demás, pero que sólo pueden registrar su venta de uno a la vez, ya que sólo hay una caja registradora, y se forma una fila de espera. En ese caso, la caja registradora sería el CPU, los empleados serían los procesos, y los clientes las operaciones que deben realizar estos procesos.

Es sumamente **importante** llevar una **buena planificación del uso del CPU**, de lo contrario, esto lleva a que la cola de procesos colapse o tenga una **INANICIÓN**.

* **INANICIÓN**: que funcione de forma tan ineficiente porque le negaría recursos a otros procesos que necesitan ejecutarse, debido a que los recursos necesarios para el procesamiento no están disponibles.

Para combatir esto, existen distintas **técnicas de planificación**, las cuales ayudan a solucionar los grandes problemas de procesos.

¿**QUÉ ES LA PLANIFICACIÓN**?

Son las políticas y mecanismos que poseen los sistemas operativos actuales para realizar la gestión del procesador. Su objetivo es dar un buen servicio a todos los procesos que existan en un momento dado en el sistema.

* El **planificador del procesador** tiene como **misión** la asignación del mismo a los procesos que están en la cola de procesos preparados.

**CRITERIOS QUE SE DEBEN TENER EN CUENTA A LA HORA DE ELEGIR O DISEÑAR UN ALGORITMO DE PLANIFICACIÓN**:

* **RENDIMIENTO**: Es el número de trabajos o procesos realizados por unidad de tiempo, que debe ser lo mayor posible.
* **TIEMPO DE RESPUESTA**: Es la velocidad con que el ordenador da la respuesta a una petición. Depende mucho de la velocidad de los dispositivos de entrada y salida.
* **TIEMPO DE SERVICIO**: **Es el tiempo que tarda en ejecutarse un proceso**, donde se incluye el tiempo de carga del programa en memoria, el tiempo de espera en la cola de procesos separados, el tiempo de ejecución en el procesador y el tiempo consumido en operaciones de entrada/salida.
* **TIEMPO DE PROCESADOR**: Es el tiempo que un proceso está utilizando el procesador sin contar el tiempo que se encuentra bloqueado por operaciones de entrada/salida.
* **TIEMPO DE EJECUCIÓN**: **Es idéntico al tiempo de servicio menos el tiempo de espera en la cola de procesos separados**; es decir, es el tiempo teórico que necesitaría el proceso para ser ejecutado si fuera el único presente en el sistema.
* **TIEMPO DE ESPERA**: Es el tiempo que los procesos están activos, pero sin ser ejecutados, es decir, **los tiempos de espera en las distintas colas**.
* **EFICIENCIA**: Se refiere a la utilización del recurso más caro en un sistema, el procesador, que debe estar el mayor tiempo posible ocupado para lograr así un gran rendimiento.

**POLÍTICAS DE PLANIFICACIÓN**:

* **FIFO**: Fist in, First Out**.** Se asigna tiempo de ejecución al CPU **al primer proceso que lo solicite;** el procesador ejecuta cada proceso hasta que termina; por tanto, los procesos entren en cola de procesos preparados permanecerán encolados en la orden en que lleguen hasta que les toque su ejecución. 🡪 **por tiempo de ejecución**. Ej.: este tipo de planificación es similar a la fila de personas esperando para poder pagar.
* **SJF**: Shortest Job First, o “**el trabajo más corto primero**”. La **prioridad** de ejecución está dada **por quien posee el menor tiempo de ejecución**. o. Para ello, debe saber el tiempo de ejecución que necesita cada proceso, lo cual no es tarea fácil, pero es posible a través de diversos métodos como puede ser la información suministrada por el propio usuario o por el propio programa, basándose en la historia anterior. Si dos procesos solicitan el CPU al mismo tiempo, primero se procesa el proceso que requiere menor tiempo de ejecución.
* **SRTF**: “Shortest Remaining Time”, si un proceso se está ejecutando y llega un segundo de menor tiempo, **se interrumpe el primero ejecutándose el segundo**. Una vez terminado este, comienza nuevamente el primer proceso, en el sector donde fue cortado, a MENOS QUE aparezca uno más chico y se ejecute ese 🡪 **Por tiempo más corto**.
  + **Es muy eficiente;**
  + **Presenta mayor sobrecarga;**
  + **Puede ser injusta ya que un proceso corto puede echar a uno largo que esté haciendo uso del procesador y que además esté terminando;**
* **ROUND ROBIN (RR)**: **Por cantidad de tiempo**. Existe una porción de tiempo establecida (o quantum de tiempo) en el que los procesos, a medida que van llegando a la fila de espera, se ejecutan en el CPU hasta que el quantum se cumple; una vez cumplido, se interrumpe el proceso y si es que aún le falta tiempo de ejecución, vuelve a la cola, ubicándose al final hasta que es nuevamente su turno. De esta forma, se establece que todos los procesos de ejecución llevan un tiempo equitativo, y todos los procesos serán ejecutados de la misma manera.

Consiste en conceder a cada proceso de ejecución un determinado periodo de tiempo q(quantum), transcurrido el cual, si el proceso no ha terminado, se le devuelve al final de la cola de procesos preparados, concediéndose el procesador al siguiente proceso por su correspondiente quantum.

Existen **OTRAS PLANIFICACIONES** que combinan de forma híbrida las ya explicadas, o presentan algoritmos diferentes; como ser:

* **RETROALIMENTACIÓN MULTINIVEL**.
* **PLANIFICACIÓN POR COMPORTAMIENTO**.
* **COLAS MÚLTIPLES**: Cuando los procesos que van a ser ejecutados en una computadora se pueden agrupar en distintos grupos, podemos asignarlos a diferentes colas, cada una con distinta planificación, para darle a cada una de ella la que realmente necesite. Esta política divide la cola en procesos preparados en varias colas separadas, de manera que los procesos se asignan a una determinada cola según sus necesidades y tipo.

Estableciendo prioridades entre los procesos, o cambiando el tiempo de ejecución.

Estos procesos viven en constante innovación, gracias a los avances de la tecnología.

**HILOS DE EJECUCIÓN (threads)**

¿**Cómo están compuestos los procesadores**?

* Físicamente, son un conjunto de un gran número de transistores que están configurados de tal forma que realizan **operaciones binarias** con impulsos de **ENERGÍA ELÉCTRICA**.
* Estos, a su vez, contienen un núcleo; ese núcleo puede ser más de uno 🡪 a **mayor cantidad de núcleos**, **mayor es la cantidad de procesos que pueden ejecutarse en paralelo**.
* **Proceso** = conjunto de operaciones que componen a un programa, los cuales, a la hora de ejecutarse, re reparten la utilización del procesador para realizar su tarea.

**Un proceso, ¿puede dividirse?**

**¡SÍ!** 🡪 Un proceso puede dividirse en **secuencias de tareas**, también llamadas **HILOS**.

* **HILOS:** **Porciones de código** que se ejecutan en simultáneo con otros subprocesos.
* **Procesos simultáneos = + EFICIENCIA.**
* **NO FORMAN PARTE FÍSICA DEL PROCESADOR**.
* **Se tarda menos en crear un hilo nuevo en un proceso ya existente a crear un nuevo proceso**.

Podríamos decir que múltiples hilos pueden existir en un proceso, ejecutándose de forma concurrente, compartiendo recursos y memoria.

En este punto encontramos una gran diferencia con los procesos **EN SÍ**, ya que **los procesos NO COMPARTEN RECURSOS ENTRE ELLOS CUANDO SE EJECUTAN**.

Al ser tantos hilos, trabajando en conjunto, es muy importante la **SINCRONIZACIÓN**, ya que un subproceso PUEDE BLOQUEAR UN RECURSO y negarle el acceso a otro hilo.

Hasta la década de los 2000, **los procesadores eran MONOLÍTICOS** (**UN SOLO NÚCLEO**), por lo cual solo podían trabajar con un solo hilo a la vez.

Luego, aparecieron los procesadores **MULTINUCLEOS** (**VARIOS NÚCLEOS**), los cuales comenzaron con esta metodología de trabajo de varios hilos de ejecución, aumentando así la velocidad del procesamiento.

Los sistemas **MONOLÍTICOS**:

* Tienen una **capacidad de respuesta menor**.
* Su **comportamiento** es más **predecible**.
* **No presentan errores**, que sí podrían presentar los multi-hilos.
* Por esto, hay **menores bloqueos de recursos**.

Los sistemas **MULTINUCLEOS**:

* **Excelente capacidad de respuesta** a operaciones con un…
* buen **trabajo en paralelo** de sus tareas.
* La **sincronización es compleja de planificar**, y su **comportamiento** es **difícil de predecir**, ya que…
* **Puede presentar errores** pasados por alto en la etapa de prueba y desarrollo.

Los hilos, tanto en sistemas monolíticos como multi-hilos, son un tema complejo e interesante; además, ponen a prueba la capacidad de los programadores cuando trabajan con los mismos.

* Cuando un proceso utiliza un recurso lo bloquea para que ningún otro proceso lo utilice. La idea es que el CPU no les dé el mismo recurso a varios procesos a la vez para que no se produzcan conflictos.