**GIT**

HARDWARE

----CApa de Controlar el HArdware

FIRMWARE: primera capa de software que se hace sobre el hardware. Codigo máquina. Más duro de leer, hecho para las máquinas más que para los humanos.

Controla directamente el hardware.

ENSAMBLADOR: segunda capa de abstarcción que tenemos sobre los ordenadores.

Es un lenguaje que se traduce en el lenguaje máquina, pero es más sencillo de entender para los humanos.

---- Capa de Sistema Operativo

KERNEL: núcleo del sistema operativo. Programado con lenguaje de bajo nivel. Está por encima del ensamblador y firmware, pero aún muy cercano al hardware. C o Java.

Sobre este Kernel se mnta el gestor de aplicaciones/Escritorio, la parte con la qu einteractú ael usuario.

El Kernel hace todas las tareas duras (gestionar baería, recursos, inputs/outputs, etc).

SO y aplicaciones: Se ejecutan las aplicaciones como Word, navegador Web, etc.

Todo está hecho como una cebolla.

No te preocupas de lo que hay más abajo. Programas con las reglas que te dan de abajo pero sin tener en cuenta cómo se escribiría allí, pero luego se va traduciendo hasta llegar al lenguaje máquina.

----

Terminal: programa que emula lo que se veía antes de las interfaces gráficas, que permite trabajar con el kernel mediante comandos. Una conexión directa sobre el SO.

1. GIT

Git es un programa que como la terminal trabaja con línea de comandos.

Cada sistema operativo trae una terminal (WIndows trae 2: Powershell y cmd)

¿Por qué necesitamos GIT?

Imaginad que el viernes terminas tu aplicación, lo publicamos y la gente empieza a entrar.

hoy lunes a primera hora ns llegan reportes de que peta por X motivos.

¿Cómo recordar cómo estaba el viernes?

* CONTROL DE VERSIONES: Git va guardando casi automáticamente los cambios que vayamos haciendo.
  + No es sólo backup, sino comprobación
* COORDINACIÓN ENTRE EQUIPOS: Integra los cambios y permite que todos los miembros del equipo vean los cambios que se van haciendo.
  + Cuando se va a guardar lso cambios, comprueba que esos cambios son compatibles antes de fusionarlo, y si no puede lo rechaza.
  + Útil también cuando un equipo programa sobre un mismo archivo en común.--
* COMPARTIR CÓDIGO: una vez esté terminado el proyecto, si nos pidieran comprobar nuestro código (entrevistas de trabajo, etc), tendríamos que comprimir las carpetas del proyecto, enviarlas por email (tal vez detectaría como virus), etc.
  + Gracias a GIT, el desarrollo ha evolucionado mucho más rápido, con el software libre.
  + Nos permite compartir nuestro código de manera fácil.

**QUÉ ES GIT:**

Se creó esta herramienta por los 90.

Es como un supervisor que comprueba todos los cambios que se hacen en una carpeta. Los guarda y tiene un historial de cambios.

GIT nos genera estos “Logs” y nos permite comprobar estos cambios.

Entre nosotros podemos mandarnos estos log, nuestro compañero puede ver el historial con lso cambios que ha realizado el otro miembro.

GIT tiene carencias, ya que se creó cuando no existía internet.

Se tiene que mandar el log por email, etc. No está integrado del todo en GIT.

En el 2000 se decidió subir GIT a la nube, en un Hub (servidor en la nube)==> GITHUB

GITHUB: nueva plataforma, “GIT” en la nube.

Nosotros trabajaremos en local en GIT, pero los cambios, el historial de cambios, código, etc, estará almacenado en la nube.

una vez estemos en la carpeta que queremos supervisar, con “git init” se inicia git y te crea una carpeta oculta “.git”.

GITHUB: una vez hechos todos los commits en local, llega el momento de tenerlo en la nube mediante GITHUB.

git remote add origin https://github.com/EricFC-84/My\_first\_repo.git

“origin” es el nombre de la conexión entre nuestro local y el remoto.

git PUSH: siguiente comando, que lo qu ehace es enviar nuestro repo al github.

git push -u origin master

Qué cosas son necesarios en una repo

1. LICENCIA: todo lo que desarrollas tiene que tener una licencia, que estipula qué puede hacer la gente con tu código, cómo usarlo y qué responsabilidades tienes tú.
   1. se puede hacer una uno mismo (con ayuda de un abogado) o coger alguna ya existente, por ejemplo en <https://choosealicense.com/licenses/>
      1. La más común es la MIT.
         1. Se le da uso total al usuario de que haga lo que quiera con el código, pero sin garantías ni responsabilidades.
      2. GNU v3.0: la licencia de software libre: hay una obligación legal que si alguien va a usar nuestro proyecto para otros motivos, están obligados a usar la misma licencia (de manera que no podrá hacer uso comercial del código ya que será libre).
   2. se copia el contenido de la licencia en un archivo LICENSE.md y ya tenemos la licencia en nuestro repo. Se tiene qu ehacer un git add, commit y push
2. archivo .gitignore
3. Archivo README.md (lenguaje mark down). ([link de info](https://medium.com/@meakaakka/a-beginners-guide-to-writing-a-kickass-readme-7ac01da88ab3))
   1. Descripción
   2. Funcionalidades (features)
   3. Cómo funciona
   4. How to use
   5. Next (siguientes pasos a hacer)
   6. Licencias. Como un disclaimer legal con las tecnologías etc.