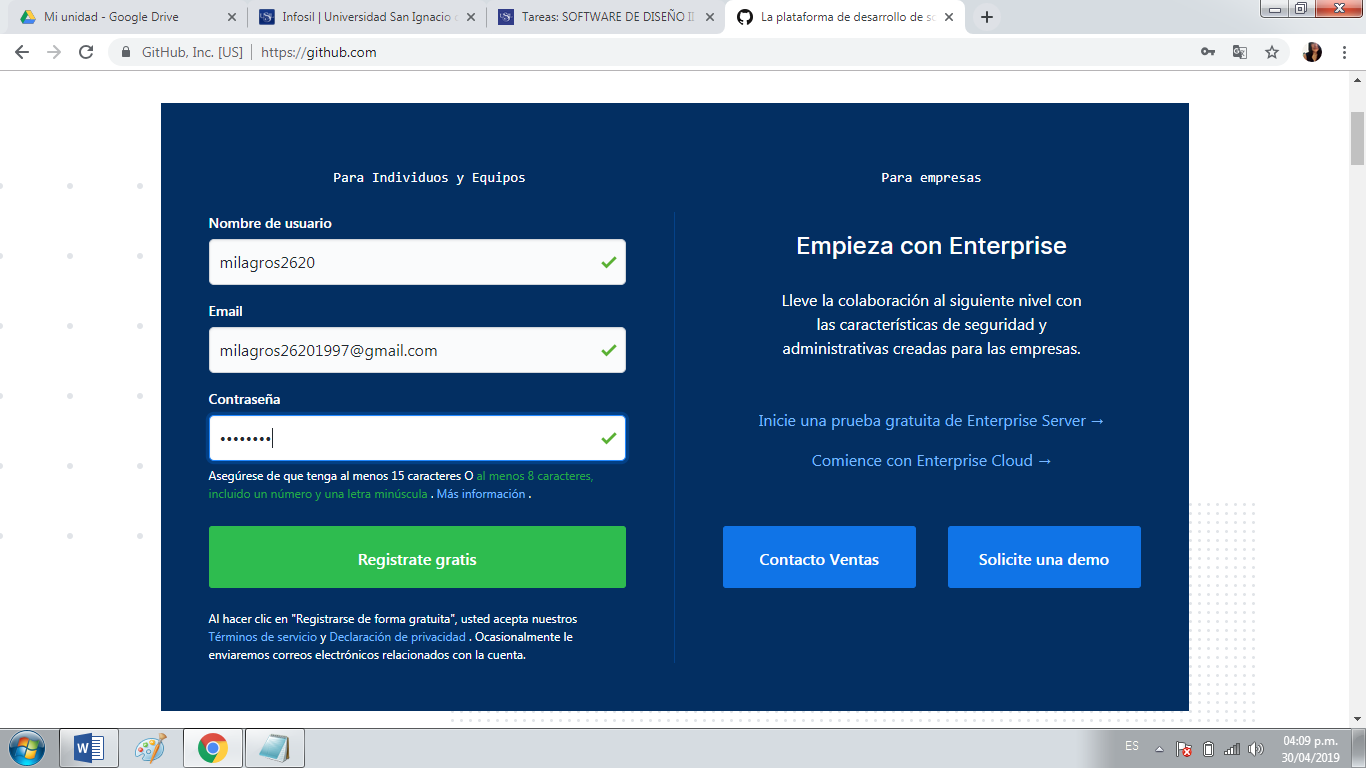
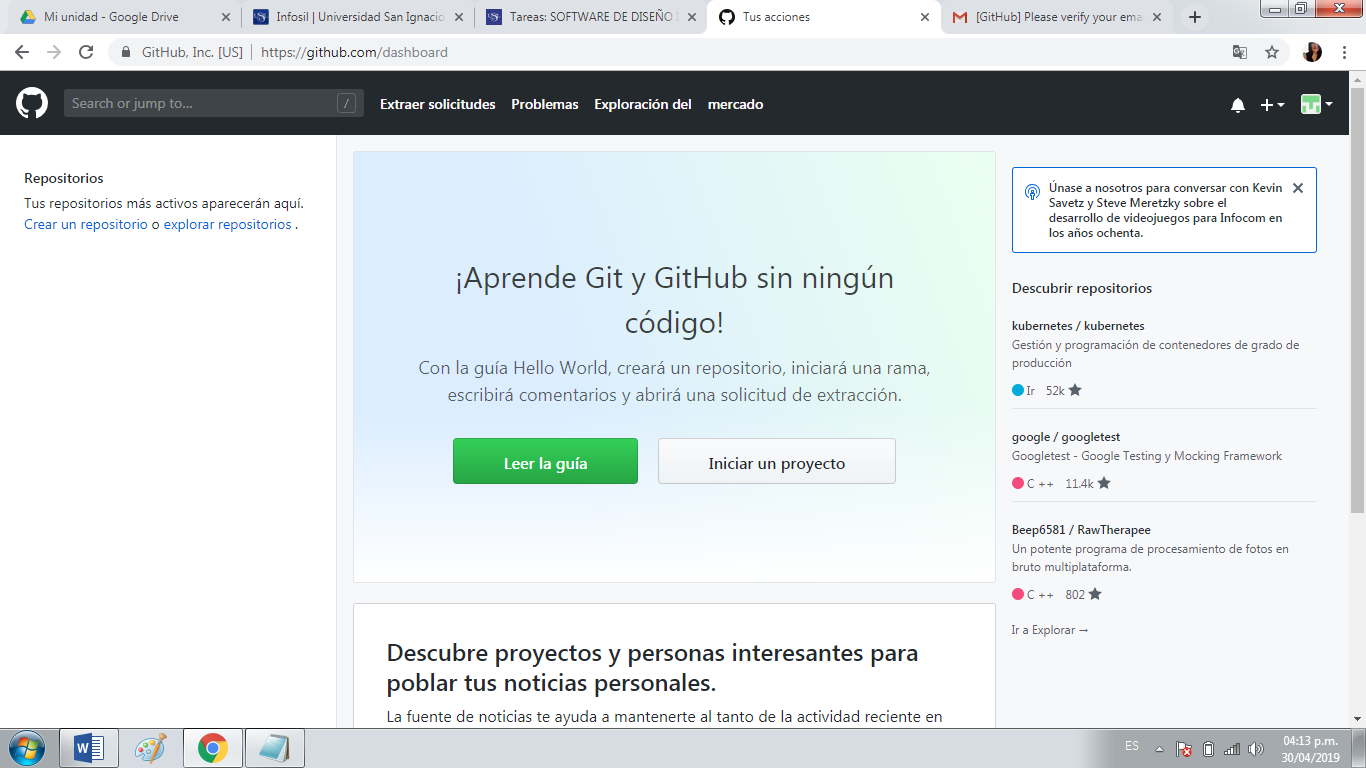
**Evidencias**

**https://github.com/milagros26201997/Mi-Nuevo-Proyecto**

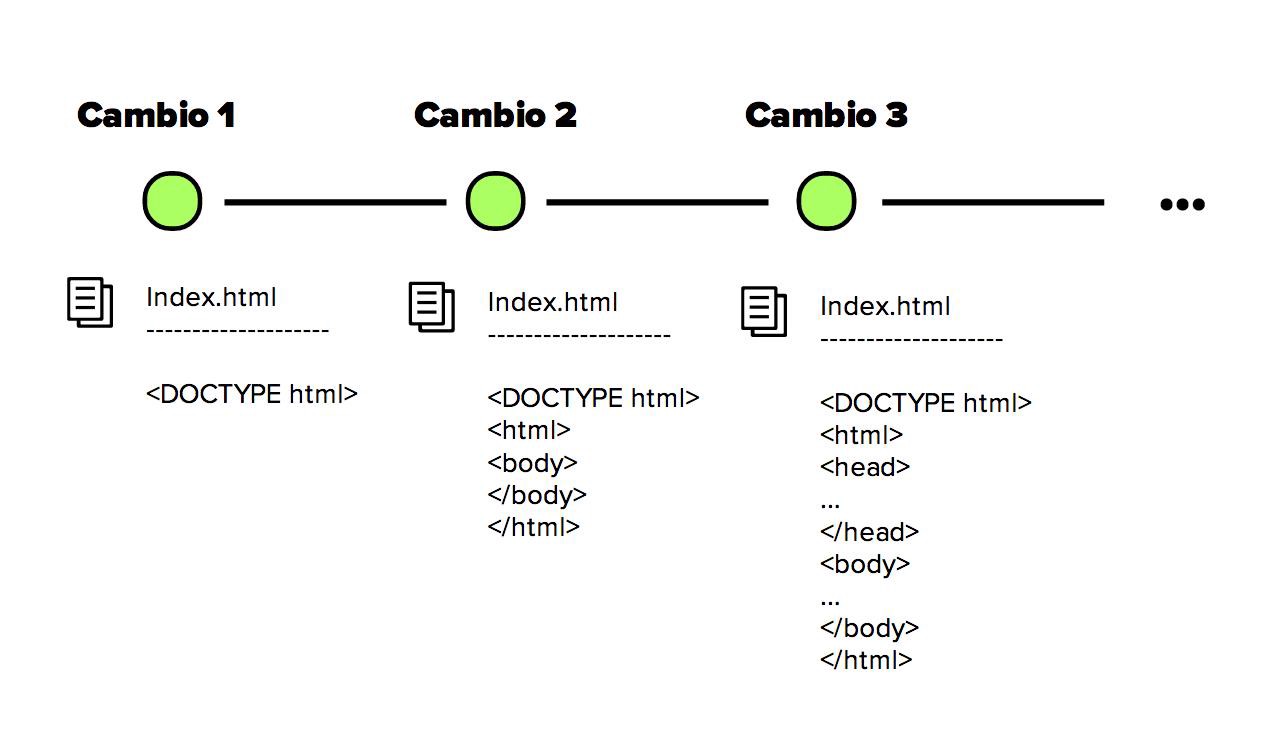


**¿Qué es un Sistema de Control de Versiones?**

Un Sistema de Control de Versiones es un software que controla y organiza los distintos cambios que se realicen sobre uno o varios documentos. Como ejemplo añadir un párrafo, cambiar el texto, etc. (Durante – Recio, s/f). Al decir control de versiones hablamos de los métodos y herramientas disponibles que tenemos para controlar todo lo referente a los cambios en el tiempo de un archivo.

Un sistema de control de versiones es una herramienta capaz de registrar todos los cambios que se realizan en uno o más proyectos, guardando a su vez versiones anteriores del proyecto, versiones a las que podemos acudir en caso de haber cometido algún tipo de error o no funciona de forma correcta.

Alcázar – Álvarez, (2014) mencionan que los sistemas de control de versiones son programas que tienen como objetivo controlar los cambios en el desarrollo de cualquier tipo de software, permitiendo conocer el estado actual de un proyecto, los cambios que se le han realizado a cualquiera de sus piezas, las personas que intervinieron en ellos, etc.

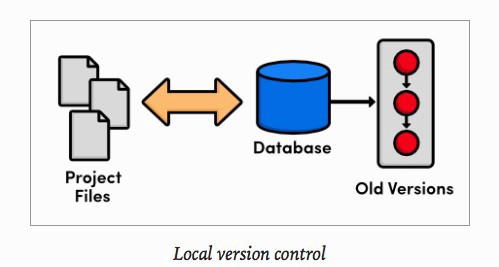


**Tipos de Sistemas de Control de Versiones**

**Sistemas Locales**

Según Developer (2017) Los sistemas de control de versiones locales en vez de mantener las versiones como archivos independientes, los almacenaban en una base de datos. Cuando era necesario revisar una versión anterior del proyecto se usaba el sistema de control de versiones en vez de acceder directamente al archivo, de esta manera en cualquier momento solo se tenía una copia del proyecto, eliminando la posibilidad de confundir o eliminar versiones.

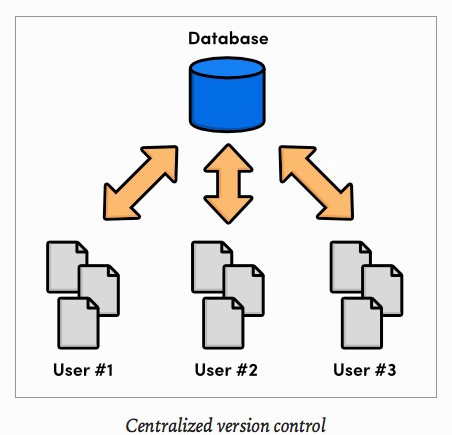
En este punto el control de versiones se llevaba a cabo en el computador de cada uno de los desarrolladores y no existía una manera eficiente de compartir el código entre ellos.



**Sistemas centralizados**

Para facilitar la colaboración de múltiples desarrolladores en un solo proyecto los sistemas de control de versiones evolucionaron: en vez de almacenar los cambios y versiones en el disco duro de los desarrolladores, estos se almacenaban en un servidor.

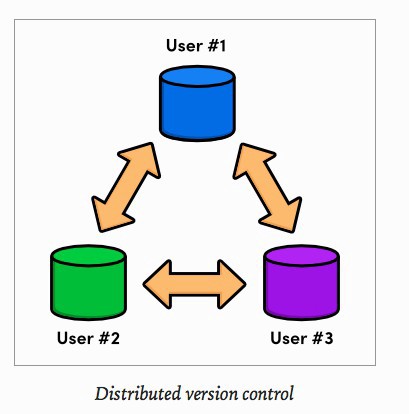
Los sistemas de control de versiones centralizados abordaron este problema impidiendo que los usuarios invalidaran el trabajo de los demás. Si dos personas editaban el mismo archivo y se presentaba un conflicto alguien debía solucionar este problema de manera manual y el desarrollo no podía continuar hasta que todos los conflictos fueran resueltos y puestos a disposición del resto del equipo.

Esta solución funcionó en proyectos que tenían relativamente pocas actualizaciones y por ende pocos conflictos pero resulto muy engorroso para proyectos con docenas de contribuyentes activos que realizaban actualizaciones a diario.

**Sistemas distribuidos**

La siguiente generación de sistemas de control de versiones se alejó de la idea de un solo repositorio centralizado y optó por darle a cada desarrollador una copia local de todo el proyecto, de esta manera se construyó una red distribuida de repositorios, en la que cada desarrollador podía trabajar de manera aislada pero teniendo un mecanismo de resolución de conflictos mucho más elegante que un su versión anterior.

Al no existir un repositorio central, cada desarrollador puede trabajar a su propio ritmo, almacenar los cambios a nivel local y mezclar los conflictos que se presenten solo cuando se requiera. Cómo cada usuario tiene una copia completa del proyecto el riesgo por una caída del servidor, un repositorio dañado o cualquier otro tipo de perdida de datos es mucho menor que en cualquiera de sus predecesores.



**Git:**

Según Dan (2016) Git es un sistema específico diseñado para controlar versiones en un entorno Linux; con el objetivo primordial de gestionar código fuente.

**Historia:**

Git fue impulsado por Linus Torvalds y el equipo de desarrollo del Kernel de Linux. Ellos estaban usando otro sistema de control de versiones de código abierto, al pasar el tiempo aquel sistema de control de versiones lo convirtieron en un software propietario. Lógicamente, no era compatible estar construyendo un sistema de código abierto, tan representativo como el núcleo de Linux, y estar pagando por usar un sistema de control de versiones propietario. Por ello, el mismo equipo de desarrollo del Kernel de Linux se tomó la tarea de construir desde cero un sistema de versionado de software, también distribuido, que aportase lo mejor de los sistemas existentes hasta el momento, y es ahí donde nace GIT. (Alcázar – Álvarez, 2014)

**Beneficios**:

* Auditoría completa del código, sabiendo en todo momento quién ha tocado algo, cuándo y qué. Control sobre cómo ha ido cambiando nuestro proyecto con el paso del tiempo.
* Volver uno o más pasos hacia atrás de forma rápida.
* Control de versiones del proyecto por medio de etiquetas.
* Seguridad, ya que todas las estructuras internas de datos irán cifradas con el algoritmo SHA1.

**Según Rojas,(2017) los comandos básicos son:**

|  |  |
| --- | --- |
| Descripción | Código |
| * Agrega el archivo al Stage | git add [archivo] |
| * Quita el archivo del Stage | git rm --cached [archivo] |
| * git add -A | git add -A |
| * Quita los archivos del Stage y borrarlos | git rm -f [archivo] |
| * Realiza commit al archivo y envía un mensaje para saber que se hizo. | git commit -m "[mensaje]" |
| * Muestra el historial de todos los commit que hemos realizado | git log |
| * Versión resumida del historial | git log --oneline |
| * Muestra la cantidad de commits según el número que indiques | git log -[numero] |
| * Anexa al último commit, un cambio que hayamos olvidado hacer. | git commit --amend |
| * Crea un tag (etiqueta) a el último commit realizado | git tag -a [version] -m "[mensaje]" |
| * Crear un tag a un hash especifico | git tag [version] [hash1] |
| * Muestra todos los tag | git tag -l |
| * Elimina el tag | git tag -d [version] |
| * Crea una nueva rama | git branch [nombre] |
| * Lista todas las ramas que tienes | git branch -l |
| * Borra la rama si esta está vacía. | git branch -d |
| * Borra la rama aun así esta contenga información. | git branch -D |
| * Cambia de nombre a una rama | git branch -m [NombreActual][NombreNuevo] |
| * Muestras todas las propiedades que git ha configurado | git config --list |
| * Muestra las diferencias entre el hash que has puesto con el ultimo que tenemos | git diff [hash1] |
| * Quita un cambio, pero lo mantiene en el Stage. | git reset --soft |
| * Quita un cambio, lo quita del stage. Los mantiene en el working directory | git reset --mixed |
| * Quita un cambio, los borra totalmente, pero si tienes guardado el hash1, lo puedes recuperar | git reset --hard |
| * Nos permite pasearnos entre los branchs (ramas), también crearlas. | git checkout [branch] |

**GitHub**

GitHub es una plataforma de alojamiento de código para el control de versiones y la colaboración. Te permite a ti y a otros trabajar juntos en proyectos desde cualquier lugar.

**Herramientas útiles para el trabajo en equipo:**

* Una wiki para el mantenimiento de las distintas versiones de las páginas.
* Un sistema de seguimiento de problemas que permiten a los miembros de tu equipo detallar un problema con tu software.
* Una herramienta de revisión de código, donde se pueden añadir anotaciones en cualquier punto de un fichero y debatir sobre determinados cambios.
* Un visor de ramas donde se pueden comparar los progresos realizados en las distintas ramas de nuestro repositorio.

**Referencias**

* Alcázar I. & Álvarez M. (2014) Una introducción a los sistemas de control de versiones, su importancia y sus características a lo largo de la historia. Descripción general de lo que son Git y Github. Recuperado de: <https://desarrolloweb.com/articulos/introduccion-git-github.html>
* Developer (2017) Sistemas de Control de Versiones, qué son y por qué amarlos. Recuperado de: <https://medium.com/@jointdeveloper/sistemas-de-control-de-versiones-qu%C3%A9-son-y-por-qu%C3%A9-amarlos-24b6957e716e>
* Durante R. & Recio P. (s/f). Sistemas para el Control de Versiones. Recuperado de: <https://rodin.uca.es/xmlui/bitstream/handle/10498/9785/trabajoSCV.pdf>
* Rojas H. (2017) Comandos basicos de GIT. Recuperado de: <https://platzi.com/tutoriales/1170-git-github/3557-comandos-basicos-de-git/>