**UNIVERSIDAD DON BOSCO**

**FACULTAD DE INGENIERÍA**

**CICLO ACADÉMICO 02 -2020**



ASIGNATURA:

**Lenguaje interpretado al cliente**

**Investigación 1**

DOCENTES:

**ING. Delmy Majano**

GRUPO:

**01T**

PRESENTADO POR:

|  |  |
| --- | --- |
| **APELLIDO, NOMBRE** | **CARNÉ** |
| Ángel Quezada Karina Lisbeth | AQ161844 |
| Candelario Chávez Alejandra Yamileth | CC161062 |
| Luis Henry Henriquez Cruz | HC180393 |
| Santos Martínez, Edwin Alexander | SM180297 |
| Sergio Isidro Rosas Carrillo | RC131146 |

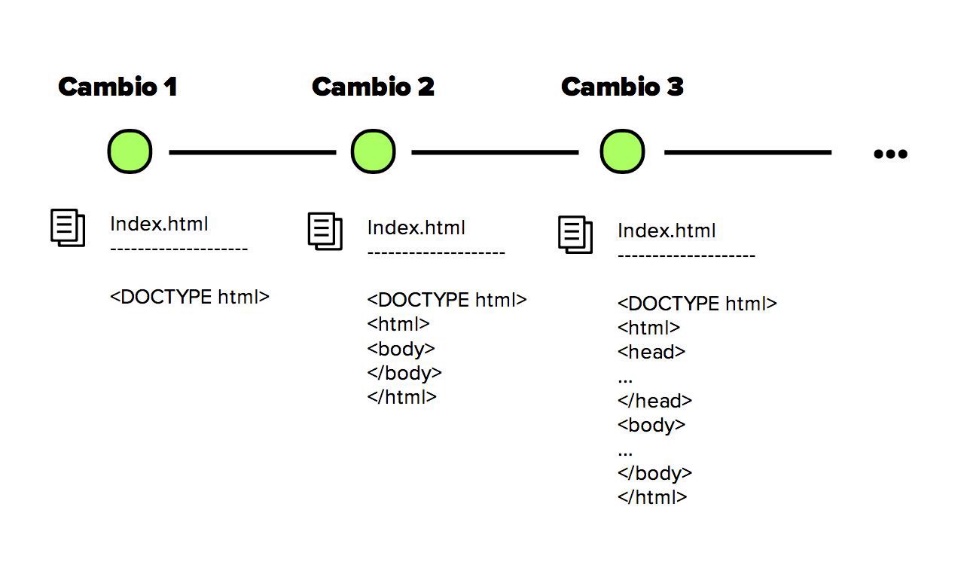
SOYAPANGO, 31 DE AGOSTO, 2020.

**Introducción**

El presente trabajo nos muestra lo que son los sistemas de control de versiones y las ventajas que presentan en el desarrollo de proyectos de software. Presentando una infografía en la que se demuestre el uso de los comandos más comunes de Git/GitHub, así como la instalación de un cliente para el sistema operativo que usted posea (Windows, Linux o MAC). Además de investigar cómo realizar la publicación de un sitio web utilizando GitHub como servidor. Realizando un ejemplo de página web utilizando la etiqueta canvas y un framework de Bootstrap.

**Sistemas de control de versiones**

El control de versiones es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo de tal manera que sea posible recuperar versiones especificas más adelante. Habitualmente se utiliza en entornos de desarrollo de software, pero puede resultar de gran utilidad para cualquier persona que necesite un control robusto sobre la tarea que está realizando.



Los sistemas de control de versiones han ido evolucionando a lo largo del tiempo y podemos clasificarlos en tres tipos: Sistemas de Control de Versiones Locales, Centralizados y Distribuidos.

**Sistemas de Control de versiones Locales**

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 de control de versiones Centralizado.**

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.

Estos sistemas centralizados abordaron la problemática de muchos usuarios trabajando en un mismo archivo al mismo tiempo, impidiendo así que los estos 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 funciono en todos aquellos proyectos que tenían o necesitaban pocas actualizaciones, pero resultó complicado para los proyectos con muchos participantes, los cuales realizaban actualizaciones diariamente.

**Sistemas de control de versiones 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 en 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. Ya que cada uno de los usuarios cuenta con una copia 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.

**¿Por qué necesito un software de control de versiones?**

VCS es una valiosa herramienta con numerosos beneficios para un flujo de trabajo de equipos de software de colaboración. Cualquier proyecto de software que tiene más de un desarrollador manteniendo archivos de código fuente debe, sin duda, usar un VCS. Además, los proyectos mantenidos por una sola persona se beneficiarán enormemente del uso de un VCS. Se puede decir que no hay una razón válida para privarse del uso de un VCS en cualquier proyecto moderno de desarrollo de software.

**Ventajas del uso de un sistema de versiones**

Utilizar un sistema de control de versiones tiene, entre otras, las siguientes ventajas:

* permite mantener un histórico de todo el desarrollo del proyecto
* añade trazabilidad al desarrollo de software, ya que se puede ver qué cambios se han hecho el código en cada versión.
* muestra mucha información estadística de cómo se está desarrollando el proyecto (principales autores, número de versiones, etcétera)
* facilita mucho trabajo en equipo.
* permite desarrollar varias versiones de un mismo programa a la vez.

**Características Fundamentales**

* **Resolución de conflictos**  
  Es muy probable que los miembros del equipo tengan la necesidad de realizar cambios en el mismo archivo de código fuente al mismo tiempo. Un VCS monitoriza y ayuda a poder resolver los conflictos entre varios desarrolladores.
* **Revertir y deshacer los cambios en el código fuente**  
  Al empezar a monitorizar un sistema de archivos de códigos fuente, existe la posibilidad de revertir y deshacer rápidamente a una versión estable conocida.
* **Copia de seguridad externa del código fuente**

Se debe crear una instancia remota del VCS que se puede alojar de forma externa con un tercero de confianza y con ello, se conservará una copia del código fuente.

## Softwares de controles de versiones

1. [**Git**](https://git-scm.com/)**:** es una de las mejores herramientas de control de versiones disponible en el mercado actual. Es un modelo de repositorio distribuido compatible con sistemas y protocolos existentes como HTTP, FTP, SSH y es capaz de manejar eficientemente proyectos pequeños a grandes.
2. [**CVS**](http://www.nongnu.org/cvs/)**:** es otro sistema de control de versiones muy popular. Es un modelo de repositorio cliente-servidor donde varios desarrolladores pueden trabajar en el mismo proyecto en paralelo. El cliente CVS mantendrá actualizada la copia de trabajo del archivo y requiere intervención manual sólo cuando ocurre un conflicto de edición.
3. [**Apache Subversion**](https://subversion.apache.org/)**(SVN):** abreviado como SVN, apunta a ser el sucesor más adecuado. Es un modelo de repositorio cliente-servidor donde los directorios están versionados junto con las operaciones de copia, eliminación, movimiento y cambio de nombre.
4. [**Mercurial**](https://www.mercurial-scm.org/)**:** es una herramienta distribuida de control de versiones que está escrita en Python y destinada a desarrolladores de software. Los sistemas operativos que admite son similares a Unix, Windows y macOS. Tiene un alto rendimiento y escalabilidad con capacidades avanzadas de ramificación y fusión y un desarrollo colaborativo totalmente distribuido. Además, posee una interfaz web integrada.
5. [**Monotone**](https://www.monotone.ca/)**:** está escrito en C ++ y es una herramienta para el control de versiones distribuido. El sistema operativo que admite incluye Unix, Linux, BSD, Mac OS X y Windows. Brinda un buen apoyo para la internacionalización y localización. Además, utiliza un protocolo personalizado muy eficiente y robusto llamado Netsync.

**GitHub**

GitHub es la plataforma de desarrollo colaborativo de software en donde usando el sistema de control de versiones de Git podemos gestionar mejor nuestro código además de alojarlo y compartirlo con la comunidad.

Lo que tal vez pocos sepan es que GitHub cuenta con una característica que nos permite crear o publicar una página web para nuestro proyecto, llamada GitHub Pages.

En este documento se explicará cómo podemos usar esta característica y sacarle el máximo provecho. Antes de empezar debes tener en consideración lo siguiente:

● Debes tener una cuenta de GitHub creada. Si no la tienes, puedes crearla aquí: https://github.com/join.

● Debes de tener GIT instalado en tu equipo https://git-scm.com/downloads

● Tu proyecto debe contar con un archivo index.html

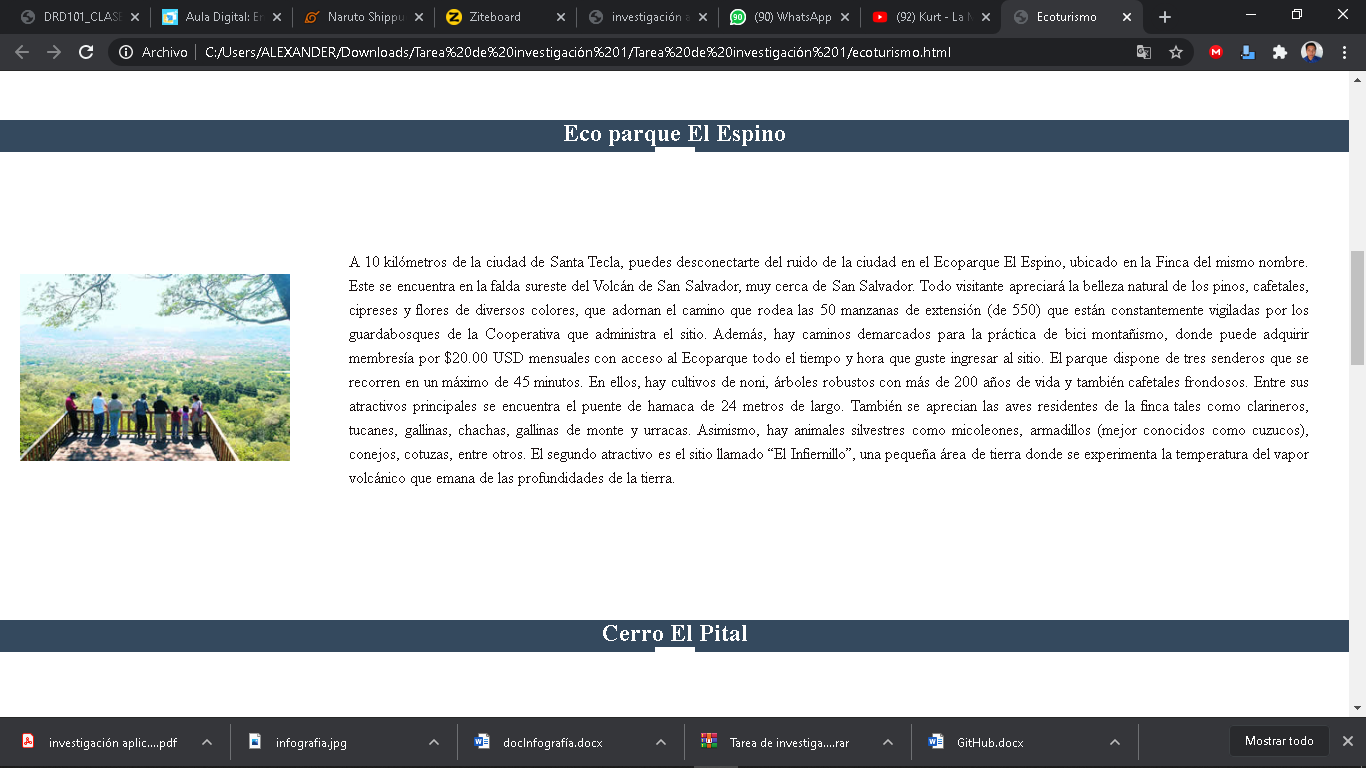
**Implementación**

* Primero vamos a github.com y creamos un nuevo repositorio para nuestro código, le pondremos un nombre a nuestro repo, se recomienda que el nombre del repositorio sea igual al nombre de proyecto en nuestro ordenador. En este caso el proyecto se llama “ejemplo”.
* Una vez creado el repositorio, copiaremos la ruta que nos mostrará, aquí es donde subiremos nuestro proyecto almacenado en el ordenador utilizando Git.
* Mediante el terminal nos ubicamos dentro de la carpeta “ejemplo”, la cual contiene los archivos de la web que subiremos.
* Inicializamos nuestro repositorio local y hacemos nuestro primer commit con los siguientes comandos y los cambios quedarán registrados, ahora podemos subir estos cambios a nuestro repositorio en GitHub, para eso escribimos:
* Nos pedirá nuestro usuario y contraseña y si el proceso se realizó correctamente debería mostrarte algo así:
* Ahora los archivos del proyecto aparecerán en nuestro repositorio en GitHub.
* En la página de nuestro repositorio vamos a dar click en el botón Branch: master se nos desplegará un cuadro y dentro de la caja de texto escribimos gh-pages, de esta manera crearemos una nueva rama.
* A continuación accederemos a la opción de settings y luego a branches.
* Cambiaremos la rama por defecto “master” por “gh-pages”, luego damos click en el botón “update” y aceptamos el mensaje que nos muestra.
* Regresamos a Options y bajamos hasta el cuadro llamado “GitHub Pages”, ahí abrimos el enlace que se nos muestra.
* Y así tenemos nuestra página publicada en GitHub.

**Pagina presentada**







**Rubrica**

