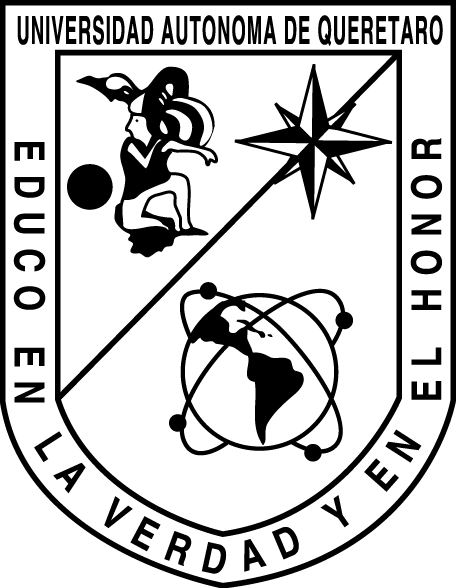
****

UNIVERSIDAD AUTÓNOMA DE QUERÉTARO

FACULTAD DE INGENIERÍA

PROGRAMA DE FORMACIÓN DUAL



PRÁCTICA #1 CONTROL DE VERSIONES

INTEGRANTES:

Espinosa Bernal Giovanni

Fuentes Flores Lorena

Martínez Olvera Judith

Ugalde Romero Dulce Carolina

Vite González Cynthia

Santiago de Querétaro, Qro., 29 de Noviembre de 2019.

Revisión:

Fecha de revisión:

ÍNDICE

[**I.** **INTRODUCCIÓN** 2](#_Toc27334032)

[**II.** **OBJETIVO** 2](#_Toc27334033)

[**Objetivo general** 2](#_Toc27334034)

[**III.** **MARCO TEÓRICO** 2](#_Toc27334035)

[**IV.** **MATERIALES Y EQUIPO** 5](#_Toc27334036)

[**V.** **METODOLOGÍA** 5](#_Toc27334037)

[**VI.** **RESULTADOS** 7](#_Toc27334038)

1. **INTRODUCCIÓN**

Durante esta práctica se hace una introducción al uso de sistemas de control de versiones: ¿para qué es?, ¿cómo usarlo? y la importancia que tiene en la industria.

1. **OBJETIVO**

**Objetivo general**

* Introducir al aprendiz en el uso de control de versiones con GIT
* Realizar una colaboración en equipo sobre la plataforma GitHub

1. **MARCO TEÓRICO**

**¿Qué es un control de versiones?**

Un control de versiones es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante.

Aunque un sistema de control de versiones puede realizarse de forma manual, es muy aconsejable disponer de herramientas que faciliten esta gestión dando lugar a los llamados **sistemas de control de versiones** o **VCS** (del inglés *Version Control System*). Estos sistemas facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas.

El control de versiones se realiza principalmente en la industria informática para controlar las distintas versiones del código fuente dando lugar a los **sistemas de control de código fuente** o SCM (siglas del inglés *Source Code Management*). Sin embargo, los mismos conceptos son aplicables a otros ámbitos como documentos, imágenes, sitios web, etc.

**Clasificación de los sistemas de control de versiones**

Dentro del campo de control de versiones, nos podemos encontrar de distintos tipos.

**a) Sistemas de control de versiones locales**

Uno de los métodos más utilizados por la gente a la hora de realizar algún tipo de control de versión de sus cambios, consistía en copiar en un directorio de su equipo local el archivo que iba a ser modificado indicando la fecha de modificación, para que en caso de error se supiese cuál era la última versión guardada.

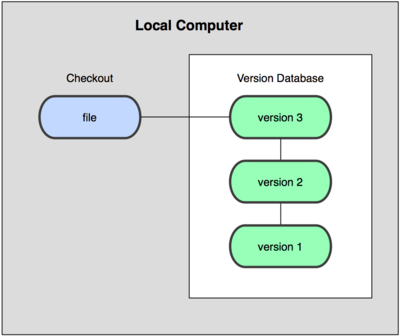


Figura 1. Diagrama de control de versiones locales

Este sistema podría valer para el desarrollo de una aplicación pequeña, pero ofrece ciertos problemas como el de no recordar dónde hemos guardado la copia o simplemente olvidarnos de hacerla.

Para hacer frente a estos problemas, los programadores desarrollaron los conocidos como VCSs locales, que consistían en una base de datos donde se llevaba un registro de los cambios realizados sobre los archivos.

**b) Sistemas de control de versiones centralizados**

El sistema comentado anteriormente nos puede valer en el caso de que estemos trabajando solos, pero en un proyecto suelen intervenir varias personas y cada una de ellas se encarga de realizar una tarea determinada. En este caso, es necesario poder contar con un sistema colaborador y es aquí donde entran en juego los sistemas de control de versiones centralizados.

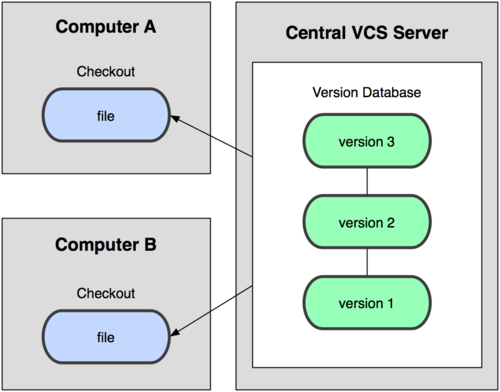


Figura 2. Diagrama de control de versiones centralizados

En estos sistemas nos encontramos un único servidor que contiene todos los archivos versionados, y los usuarios que forman parte del proyecto se los pueden descargar desde ese servidor centralizado.

Este sistema tiene un problema muy claro, y es que, al utilizar un único servidor centralizado, en caso de problema en ese servidor toda la información se podría perder.

**c) Sistemas de control de versiones distribuidas**

A diferencia del caso anterior, en estos sistemas no tenemos un único servidor que mantenga la información del proyecto, sino que cada usuario contiene una copia completa del proyecto de forma local. De esta forma, si un servidor muere, cualquiera de los repositorios de los clientes se podría utilizar para restaurar el servidor.

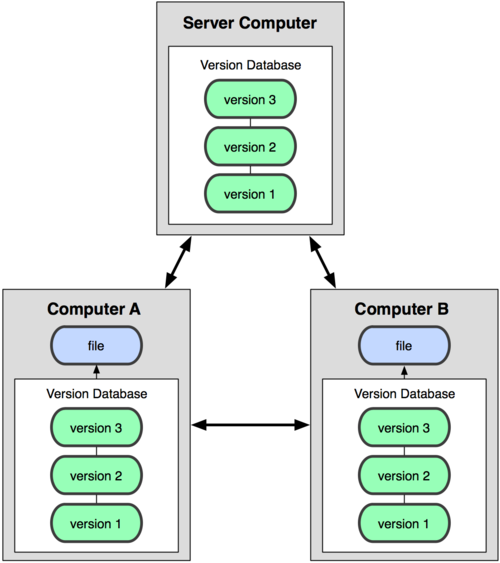


Figura 3. Control de versiones distribuidas

**¿Qué es Git?**

Git es un sistema de control de versiones distribuido, como habíamos indicado en el punto anterior, cuyo principal objetivo es ayudar en el desarrollo de cualquier tipo de aplicación manteniendo una gran cantidad de código de un gran número de programadores diferentes.

Podemos encontrar una gran diferencia del uso de Git respecto a otras herramientas similares. La primera de ella es que mientras que otros sistemas almacenan archivos originales, conservando una lista de los cambios realizados, Git guarda un snapshot del estado de cada archivo. Si el archivo no ha cambiado, no crea una nueva copia, sino que crea una referencia al archivo original.

Con todo esto, Git nos aporta:

* 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.

**¿Qué es GitHub?**

GitHub se puede definir como un servidor donde alojar los repositorios de los proyectos, añadiendo funcionalidades extra para la gestión del proyecto y del código fuente.

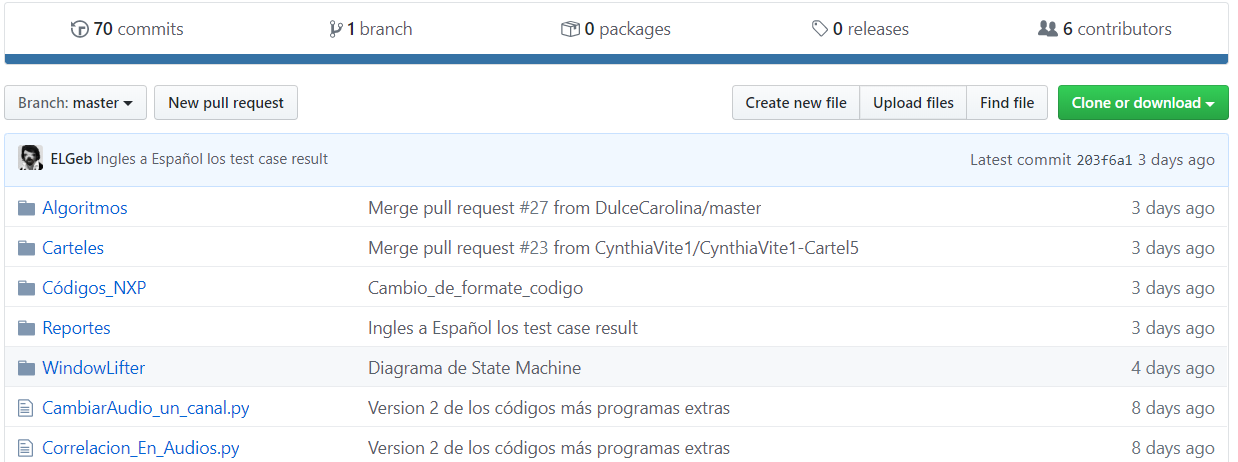


Figura 4. Ejemplo control de versiones usando GitHub

Entre las principales características que nos ofrece GitHub podemos destacar:

* Una Wiki que opera con Git para el mantenimiento de las distintas versiones de las páginas.
* Sistema de seguimiento de problemas. Se trata de un sistema muy parecido al tradicional ticket, donde cualquier miembro del equipo o persona (si nuestro repositorio es público) puede abrir una consulta o sugerencia que se quiera hacer.
* Herramienta de revisión de código. Permite añadir anotaciones en cualquier punto de un fichero.
* Visor de ramas que permite comparar los progresos realizados en las distintas ramas de nuestro repositorio.

1. **MATERIALES Y EQUIPO**
2. Software Git
3. **METODOLOGÍA**

Empezar a trabajar con Git y GitHub no es nada complicado. Lo primero que debemos hacer es realizar la instalación de Git en nuestro equipo; dependiendo del tipo de sistema operativo que utilicemos, descargaremos una versión u otra. Una vez instalado, será necesario realizar una configuración básica en la que indicaremos nuestro nombre y correo electrónico. Para ello ejecutamos la aplicación Git, con lo que abriremos el terminal desde donde ejecutaremos todas las instrucciones necesarias.

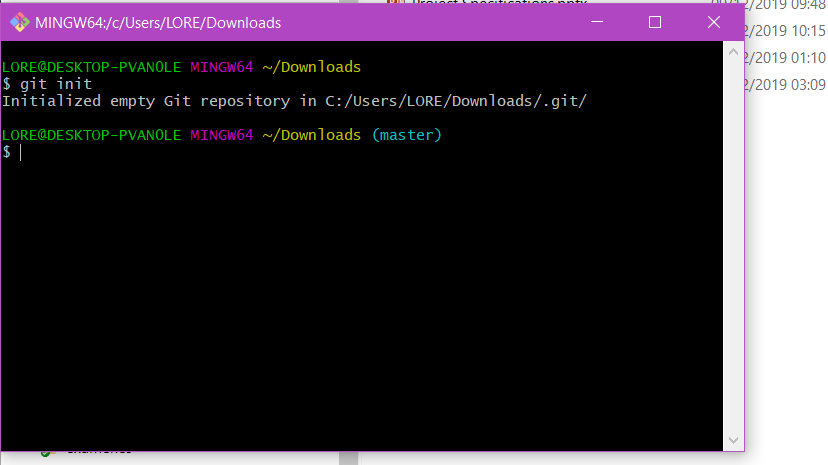


Figura 5. Inicialización de repositorio local en pantalla de Git

**1.- Descargar un proyecto por primera vez**

Para descargarnos un proyecto por primera vez, hay que utilizar el comando “git clone” seguido de la URL de github que corresponde al proyecto.

**git clone https://github.com/acens/xxxxxxx.git**

Cada proyecto tendrá una ruta distinta que asignará GitHub por defecto. Al ejecutar la instrucción anterior, lo que hemos conseguido es hacer una copia del proyecto que estuviese subido en GitHub a nuestro equipo local.

**2.- Ver los cambios realizados en el repositorio**

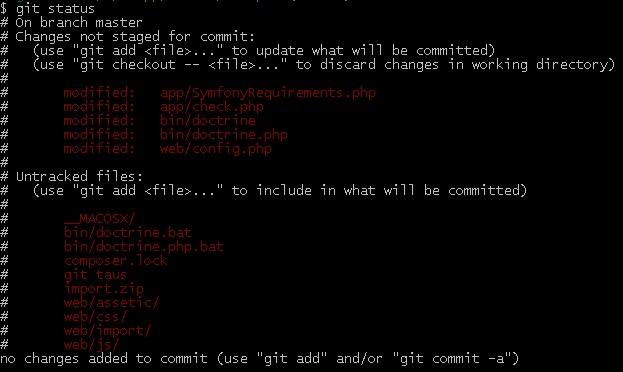
Si queremos ver los cambios que hemos ido haciendo en el repositorio, lo que se utiliza es el comando “git status”.

Figura 6. Apariencia de terminal de Git utilizando el comando "git status"

En este caso nos encontramos dos apartados bien diferenciados:

* **Changes not staged for commit.** Se corresponde con archivos ya existentes en el repositorio y que han sido modificados.
* **Untracked files.** En este caso corresponden a nuevos archivos que aún no han sido subidos al repositorio.

**3.- Añadir cambios**

Si queremos subir un archivo modificado al repositorio alojado en GitHub, lo primero que deberemos hacer es indicarlo mediante la instrucción “git add”, seguido de la ruta hasta llegar al archivo que queremos subir.

**git add ruta\_archivo\_subir**

Si ahora volvemos a ejecutar el comando “git status”, este archivo nos aparecerá en otra sección llamada “changes to be committed”.

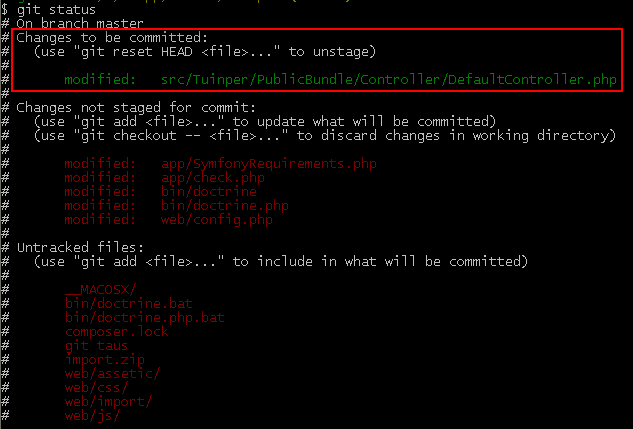


Figura 7. Commit creado con modificaciones realizadas

1. **RESULTADOS**

Implementación de control de versiones para el desarrollo de proyectos en la empresa Continental, utilizando el software Git con la plataforma GitHub. En los resultados podemos ver que el trabajo en equipo y el control de versiones es una forma muy importante de obtener un proyecto completo de manera oportuna.

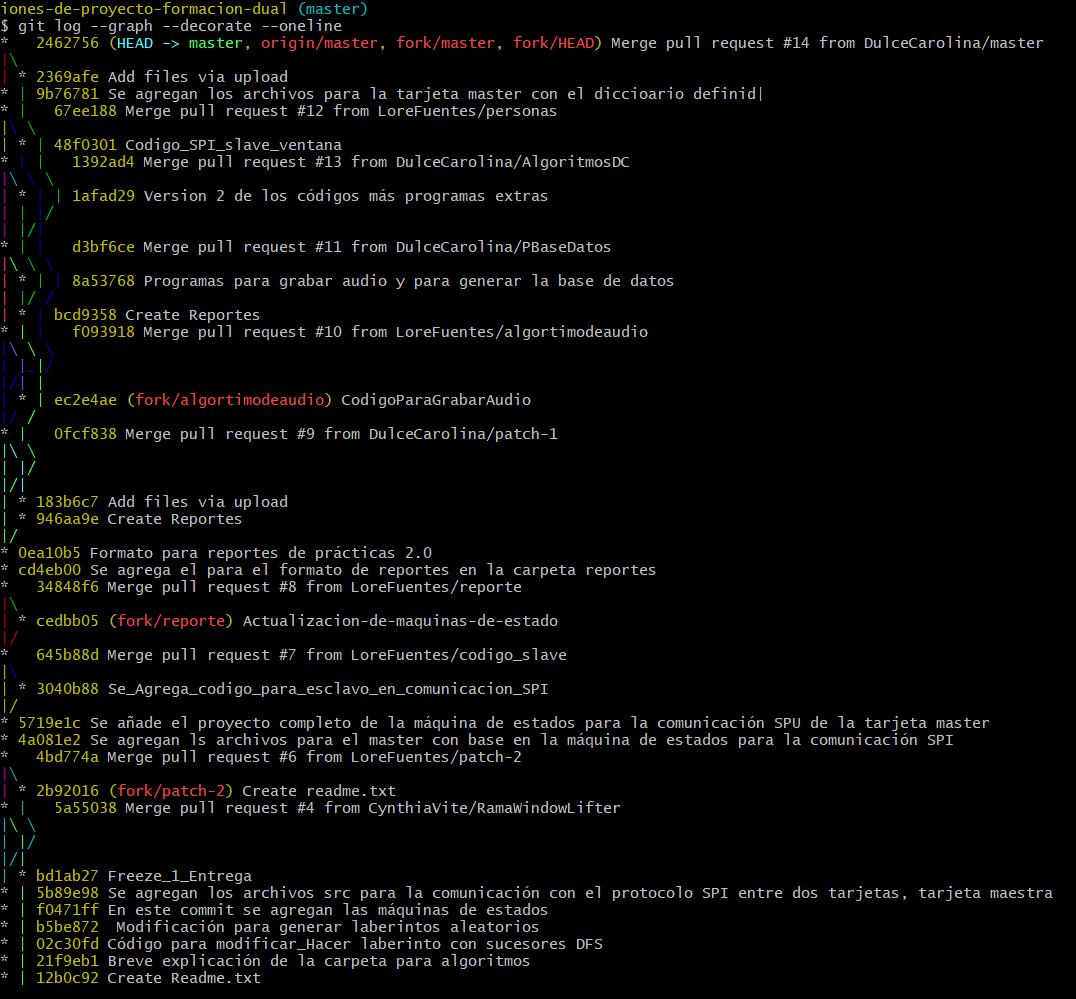
****

Figura 8. Gráfico obtenido de control de versiones de proyectos en estancia en Continental