

REPOSITORIOS Y CONTROL DE VERSIONES CON GIT

TABLA DE CONTENIDO

¿QUÉ ENCONTRARÁN?	2
¿QUÉ ES UN REPOSITORIO?	2
¿Y QUE ES UN SISTEMA DE CONTROL DE VERSIONES?	2
GIT	3
Conceptos Básicos.	3
DESCARGA E INSTALACIÓN DE GIT.....	4
GITHUB.....	6
CREAR REPOSITORIO EN GITHUB.....	6
INSTALAR CLIENTE GITHUB	9
GESTIONAR REPOSITORIO DE DOCUMENTACIÓN.....	11
CREAR NUEVO REPOSITORIO	13
VINCULAR ECLIPSE CON GITHUB.....	15
CLONAR PROYECTO EN ECLIPSE	19
SUBIR CAMBIOS DESDE ECLIPSE A REPOSITORIO REMOTO.....	23
SUBIR CAMBIOS AL REPOSITORIO – USUARIO COLABORADOR.....	26
ASIGNAR PERMISOS A USUARIOS.....	29
RESOLVER CONFLICTOS	34
CONSIDERACIONES FINALES.	37

¿QUÉ ENCONTRARÁN?

En este documento encontrarán una guía rápida de cómo trabajar con un sistema de control de versiones como GIT, la guía inicia con algunos conceptos básicos para posteriormente pasar por las bases desde la instalación de las herramientas necesarias hasta el trabajo con el IDE Eclipse Neon.

En esta guía no se trabajará directamente desde la consola sino que se hará uso de un cliente para la gestión de repositorios.

Se recomienda leer muy bien los enunciados y apoyarse de la documentación oficial.

¿QUÉ ES UN REPOSITORIO?

En un lenguaje natural podríamos referirnos a un repositorio como el lugar donde almacenaremos y gestionaremos información, puede ser simplemente un sistema de archivos en un disco duro, una Base de Datos o por ejemplo podríamos decir que una Biblioteca es un gran repositorio, porque allí se almacenan libros y artículos de interés, la parte de gestión es cuando la biblioteca es administrada y sus libros actualizados.

En nuestra área los repositorios serán esos lugares dentro de nuestro equipo o en la nube donde como se mencionó anteriormente almacenaremos información (Documentos, Código fuente) y dicha información será gestionada por miembros de nuestro equipo de trabajo, para esto haremos uso de herramientas que nos faciliten el control de versiones.

¿Y QUE ES UN SISTEMA 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 esta manera se puede recuperar versiones específicas más adelante.

Sabemos que una versión representa los diferentes estados o actualizaciones de algo, si en un grupo de trabajo creamos un documento tenemos la versión inicial, pero si después de un tiempo ese documento es modificado por alguno de los integrantes del equipo, pero se requiere tener un histórico del documento original, pues ya podemos estar hablando de una segunda versión y así sucesivamente.

Ahora, un Sistema de Control de Versiones permite la gestión de los cambios que se realicen sobre determinado elemento (en nuestro ejemplo el documento), así podremos administrar quien realizó que cambio, cuando lo realizó, quién modificó, y tener el documento actualizado sin que se pierdan las versiones anteriores.

Existen muchas herramientas como Subversión, Mercurial, GIT entre otras que nos brindarán el entorno necesario para construir nuestros repositorios.

En el siguiente enlace se profundiza un poco más sobre este tema: <https://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones>

GIT

Tomando la definición de Wikipedia “Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora y coordinar el trabajo que varias personas realizan sobre archivos compartidos”

Conceptos Básicos.

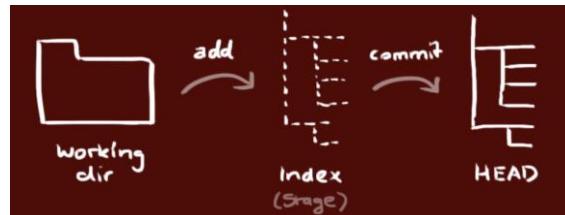
Cuando trabajamos con GIT a nivel general tendremos un esquema de tres “Arboles” que representan el estado o flujo de trabajo de nuestra información, estos son el **directorio de trabajo**, el **index** y el **HEAD**

Directorio de trabajo: Este es el espacio inicial donde se encuentran los archivos que queremos gestionar en el repositorio.

Index: representa un lugar o zona intermedia donde se van a alojar los archivos antes de ser cargados en el repositorio local en su totalidad.

HEAD: Este es el lugar o zona donde finalmente serán alojados los archivos del repositorio local antes de pasar al repositorio remoto.

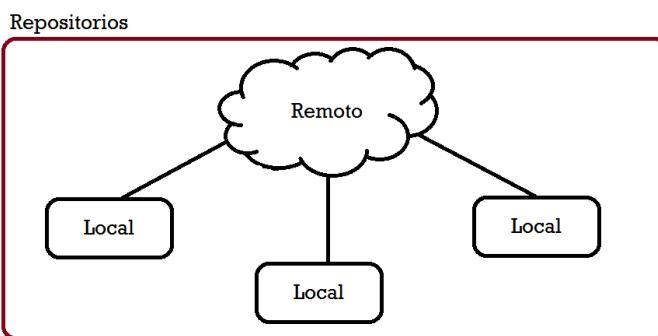
Como vemos en la gráfica se pueden evidenciar los bloques o zonas por los que pasa la información, es importante tener claro este flujo de trabajo en el que desde nuestro ambiente local donde está la información



Originalmente debemos agregar (add) los archivos al index para luego mediante un commit ser enviados al head y desde allí tenerlos listos para poderlos almacenar en un repositorio remoto.

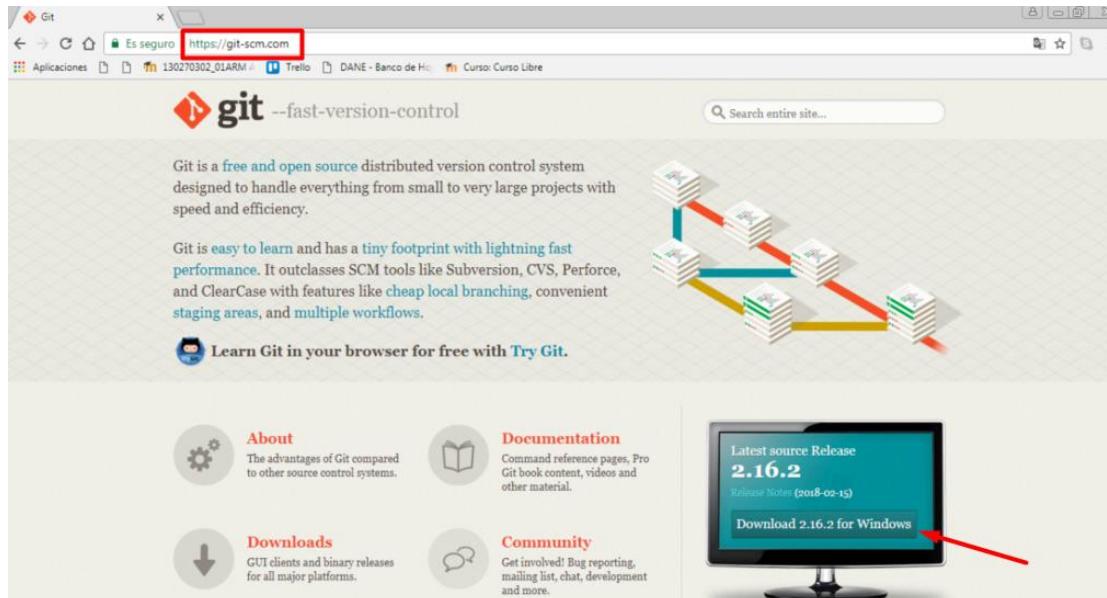
Basado en lo anterior cuando usamos una herramienta de control de versiones como GIT internamente vamos a tener un repositorio local donde trabajamos directamente con la información y un repositorio remoto donde finalmente va a estar alojada dicha información, de esa manera podemos trabajar varias personas en un mismo repositorio donde cada uno tendrá su propio ambiente local (trabajando con el mecanismo explicado anteriormente) para al final apuntar hacia un único ambiente remoto.

Así cada miembro del equipo tendrá inicialmente la versión original del sistema o información pudiendo realizar modificaciones en su ambiente local que luego serán sincronizadas para el resto del equipo de trabajo al subirlas (commit and push) al ambiente remoto compartiendo los cambios a todos los usuarios.

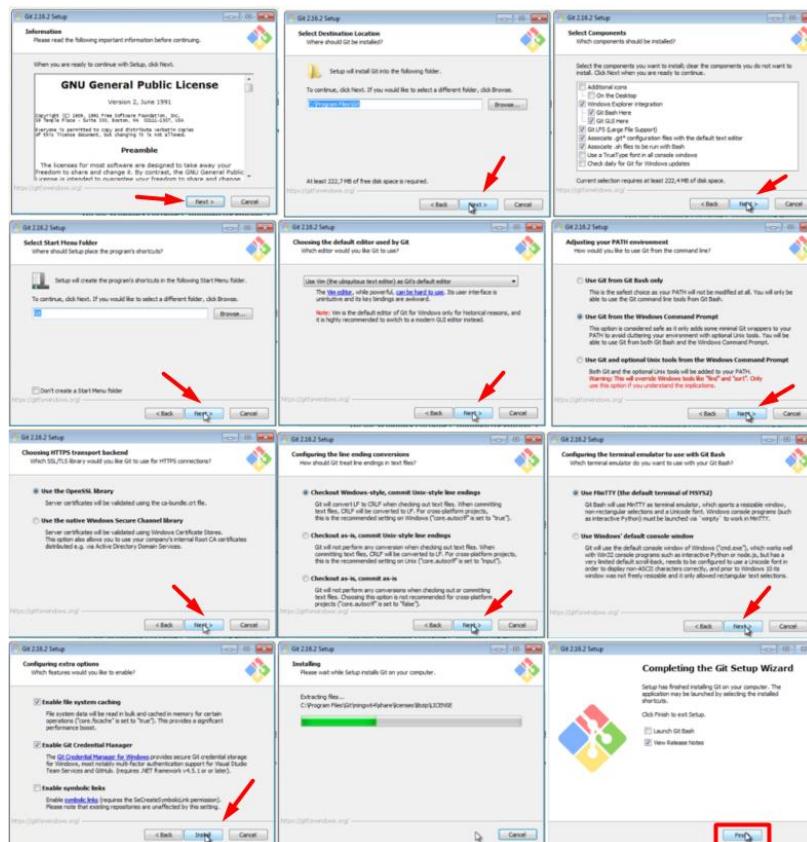


DESCARGA E INSTALACIÓN DE GIT

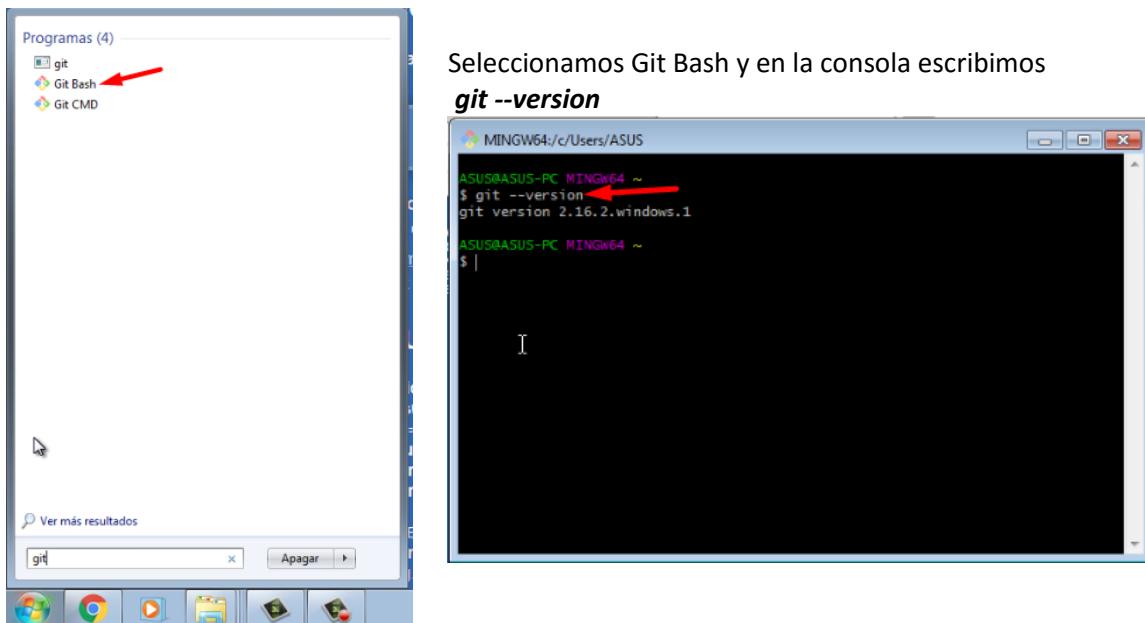
Inicialmente desde la página oficial <https://git-scm.com/>



Al descargar el instalador damos siguiente siguiente y listo.



Después de esto procedemos a verificar la instalación buscando en inicio la consola de git



Como se mencionó al inicio de esta guía, vamos a realizar la gestión de repositorios mediante un cliente gráfico que nos permitirá trabajar con GIT de forma más sencilla e intuitiva, sin embargo como buena práctica se recomienda explorar el proceso por consola con el fin de conocer los diferentes comandos y acciones que realiza por debajo la herramienta, para esto una referencia puede ser <http://rogerdudler.github.io/git-guide/index.es.html>

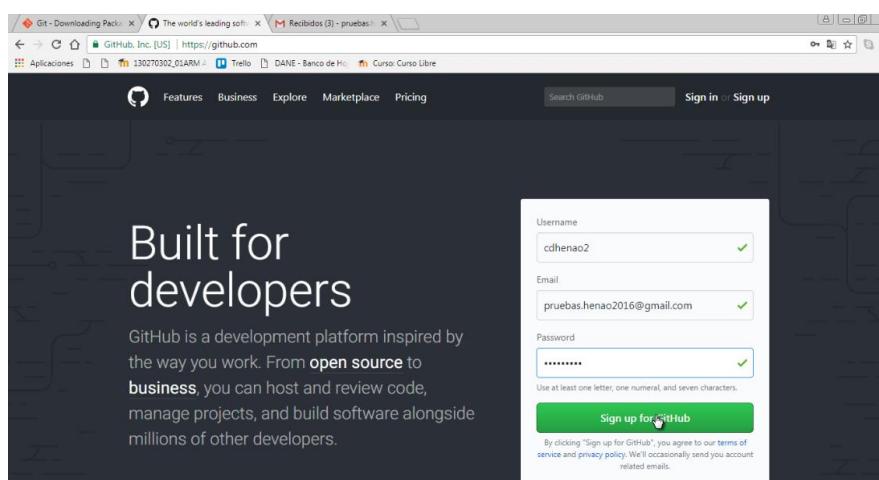
GITHUB

GitHub es un espacio gratuito para alojar proyectos en la nube utilizando el sistema de control de versiones **Git**, el código se almacena de forma pública, aunque también se puede hacer de forma privada mediante opciones de pago.

Podemos pensar en **GitHub** como una red social para desarrolladores donde las publicaciones y contenido que se comparte son principalmente con un enfoque técnico

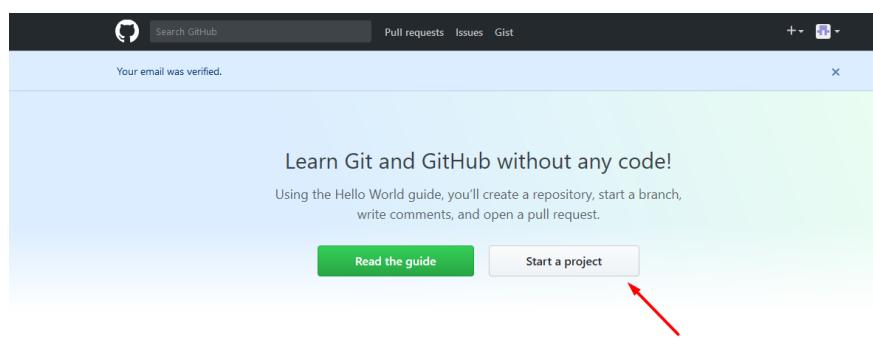
CREAR REPOSITORIO EN GITHUB.

Para crear un repositorio Inicialmente debemos tener una cuenta en el sitio oficial de GitHub
<https://github.com/>



Después del registro podemos confirmar en nuestro correo electrónico el enlace para iniciar el proceso, cuando inicia damos continuar en las configuraciones donde podemos definir cuáles son nuestros gustos o intereses, damos siguiente y al final confirmamos esa configuración inicial.

Si deseamos podemos crear un repositorio directamente ingresando al botón Start a project



Al hacerlo diligenciamos el formulario para la creación de nuestro repositorio.

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner cdhenao2 / Repository name Documentos

Great repository names are short and memorable. Need inspiration? How about crispy-system.

Description (optional)
Este es un repositorio de pruebas de documentación.

Public Anyone can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: None | ⓘ

Create repository

Se debe diligenciar el nombre del repositorio, descripción, lo definimos como público y de forma opcional podemos establecer el tipo de licencia de uso.

Al final se presenta la página con la dirección de nuestro repositorio, se debe tener en cuenta que en este momento lo que tenemos es el espacio para subir nuestros proyectos.

This repository Search Pull requests Issues Marketplace Explore + ⌂

cdhenao2 / Documentos Watch 0 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Quick setup — if you've done this kind of thing before
Set up in Desktop or HTTPS SSH https://github.com/cdhenao2/Documentos.git ⓘ
We recommend every repository include a README, LICENSE, and .gitignore.

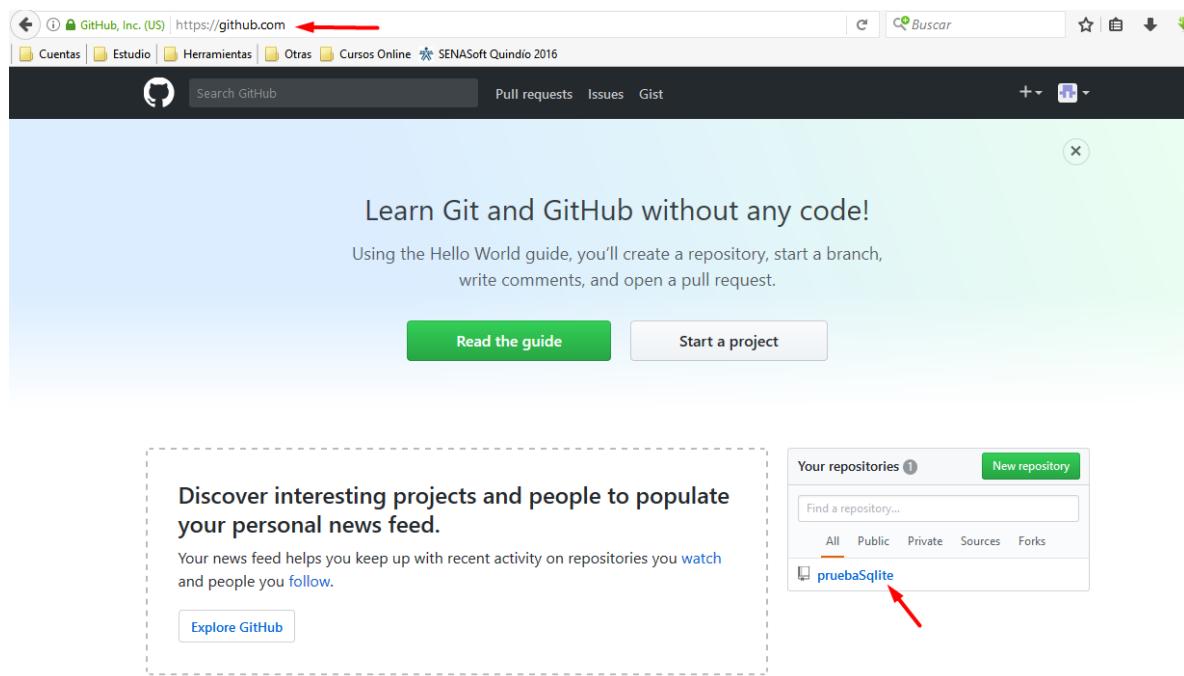
...or create a new repository on the command line

```
echo "# Documentos" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/cdhenao2/Documentos.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/cdhenao2/Documentos.git
```

Al ingresar nuevamente a la página oficial podremos ver nuestro repositorio creado.

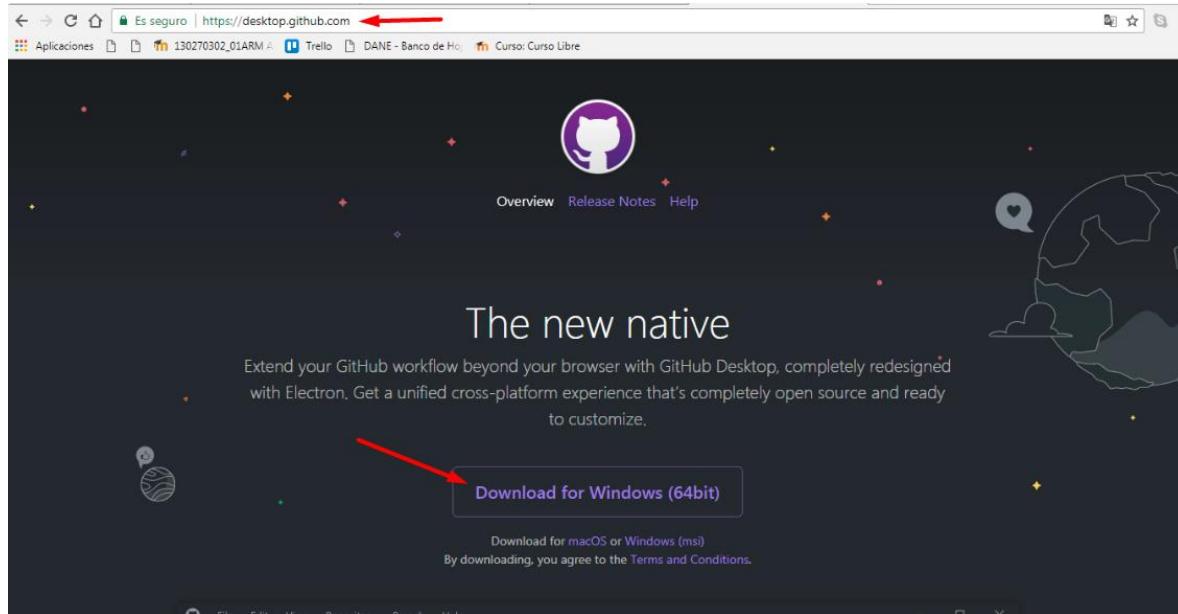


The screenshot shows the GitHub homepage. At the top, the URL https://github.com is visible with a red arrow pointing to it. Below the header, there's a navigation bar with links like 'Cuentas', 'Estudio', 'Herramientas', 'Otras', 'Cursos Online', and 'SENAsoft Quindío 2016'. The main content area features a green banner with the text 'Learn Git and GitHub without any code!' and a sub-instruction: 'Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request.' Two buttons are present: a green 'Read the guide' button and a white 'Start a project' button. On the right side, there's a sidebar titled 'Your repositories' with a 'New repository' button. Below it is a search bar labeled 'Find a repository...' and a filter section with tabs for 'All', 'Public', 'Private', 'Sources', and 'Forks'. A red arrow points to the repository 'pruebaSqlite' listed in the results. The entire screenshot is framed by a dashed border.

INSTALAR CLIENTE GITHUB

Después de haber creado nuestra cuenta, podemos descargar el cliente **GitHub** para tenerlo en instalado en nuestro pc, en este caso descargamos la versión para Windows desde la página oficial.

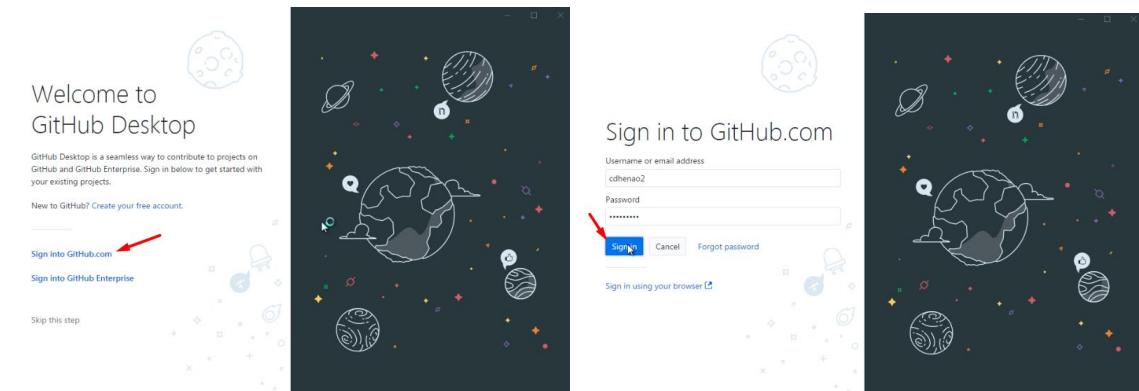
<https://desktop.github.com/>



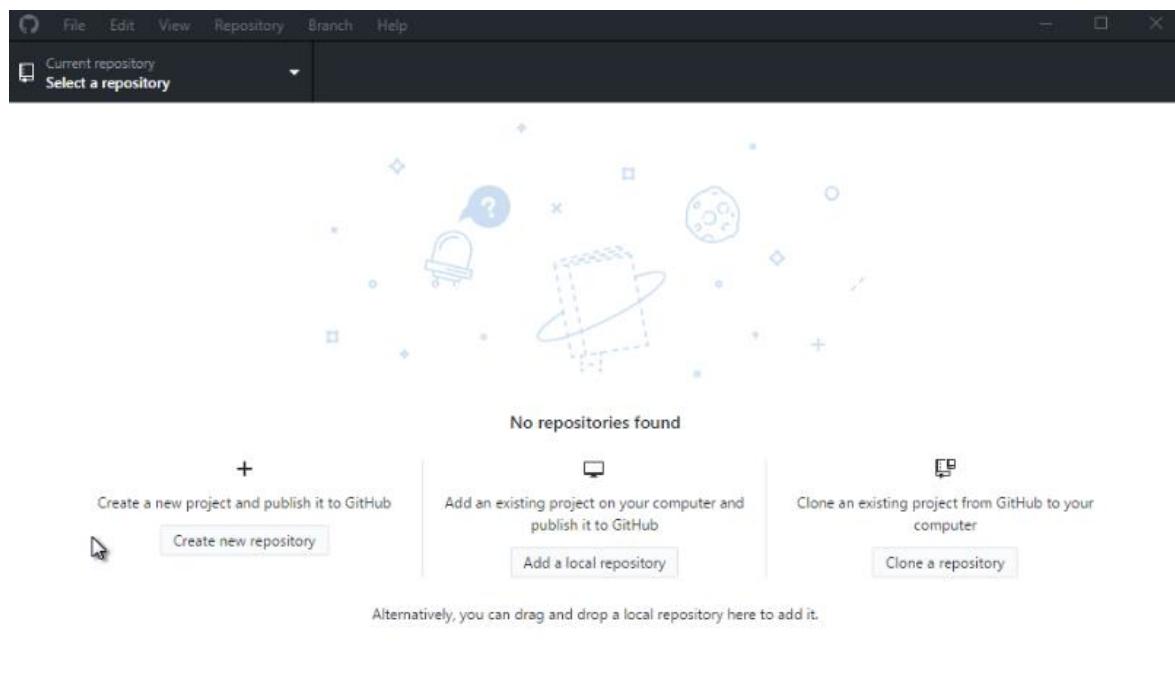
Al hacerlo se descarga el instalador, posteriormente lo ejecutamos y seguimos el paso a paso.

(El proceso de instalación no presenta mayor grado de dificultad ni demora, sin embargo en caso de producirse algún fallo se deberá intentar nuevamente.)

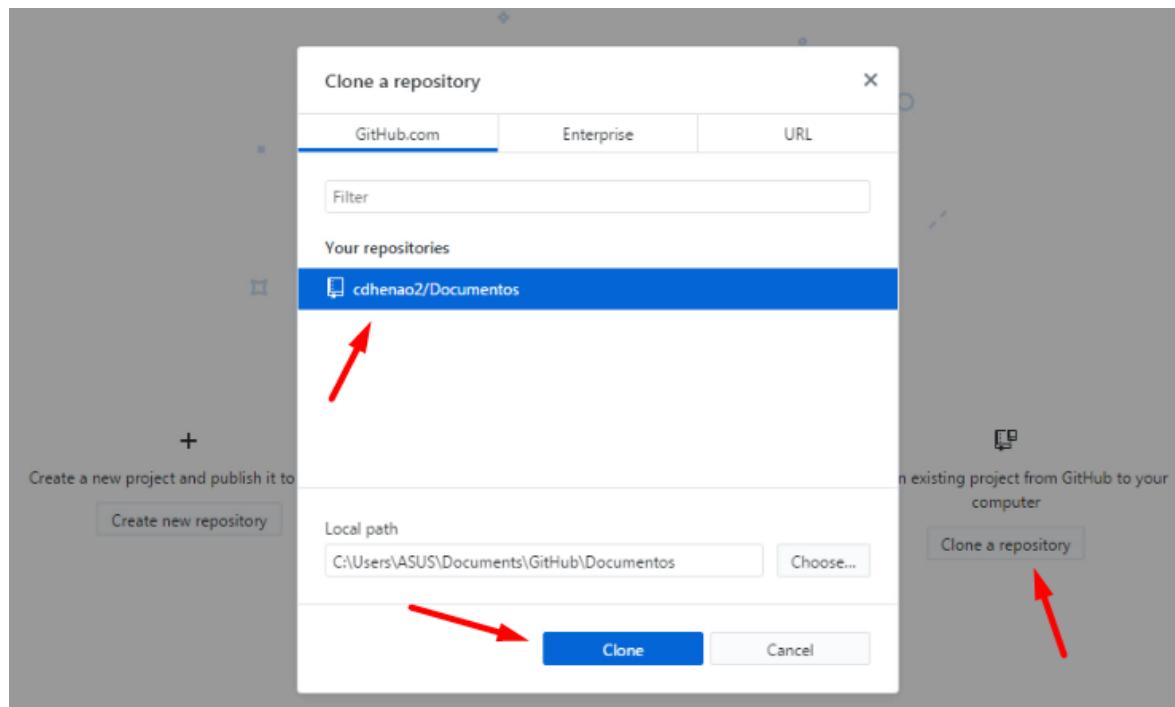
Inicialmente se presenta una ventana donde podemos realizar la vinculación con nuestra cuenta GitHub o crear una nueva.



Luego de vincular nuestra cuenta se presenta la ventana principal del sistema, desde aquí podremos realizar 3 acciones: Crear un nuevo repositorio, agregar un repositorio existente o clonar un repositorio.



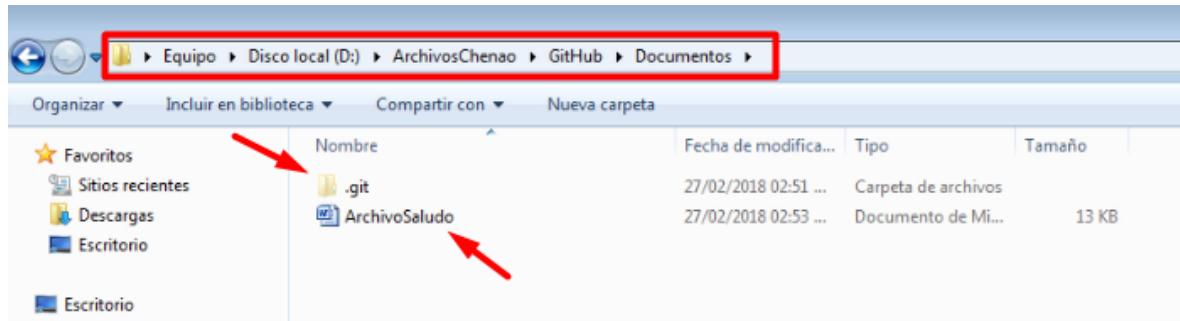
En nuestro caso vamos a clonar el repositorio creado en GitHub con anterioridad



En este proceso podemos definir cuál va a ser la ruta donde queremos que se almacenen nuestro repositorio y desde allí gestionar la información que queremos cargar a GitHub.

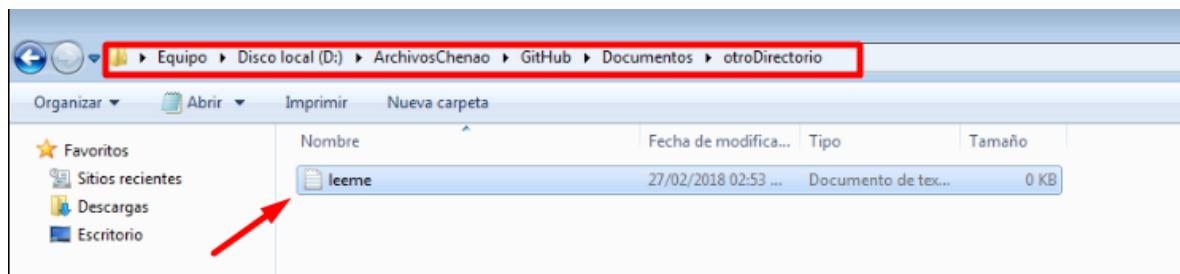
GESTIONAR REPOSITORIO DE DOCUMENTACIÓN.

Después de haber creado el repositorio remoto en GitHub y ya teniendo nuestro ambiente local mediante el cliente de escritorio ya podemos empezar a gestionar la información que queremos almacenar.

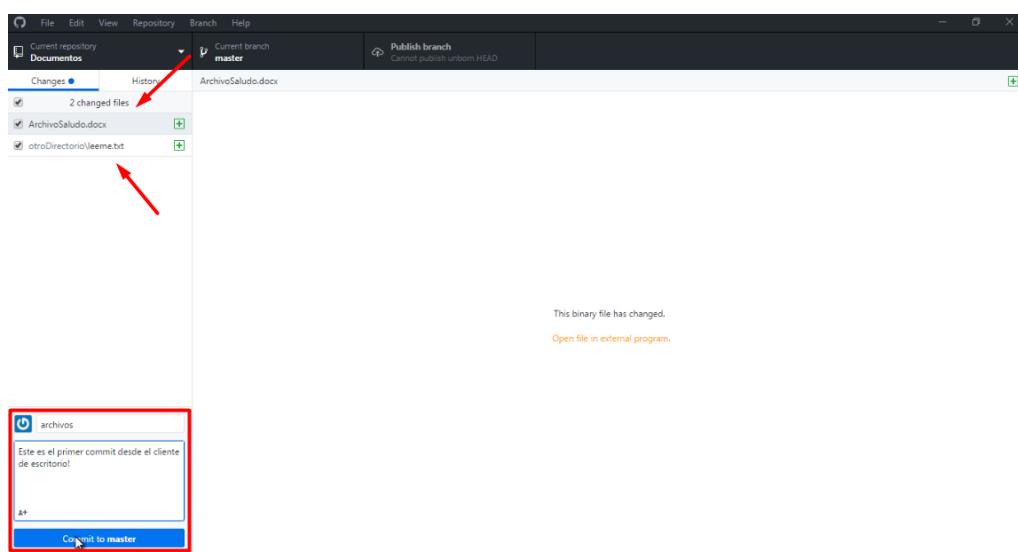


La ruta corresponde a la ruta que definimos al momento de clonar el repositorio, la carpeta .git nos indica que ese directorio está emparentado con el repositorio remoto y por ultimo tenemos un documento que vamos a gestionar.

Allí podemos agregar diferentes elementos o archivos a gestionar

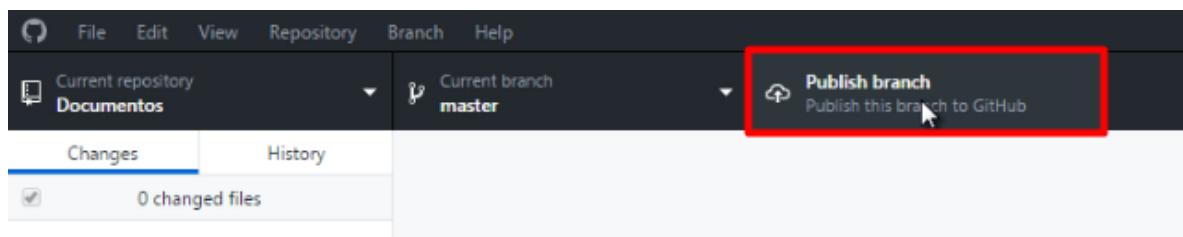


Después de esto regresamos al cliente GitHub que instalamos en nuestro equipo y veremos que ya se encuentran vinculados los archivos en la herramienta.

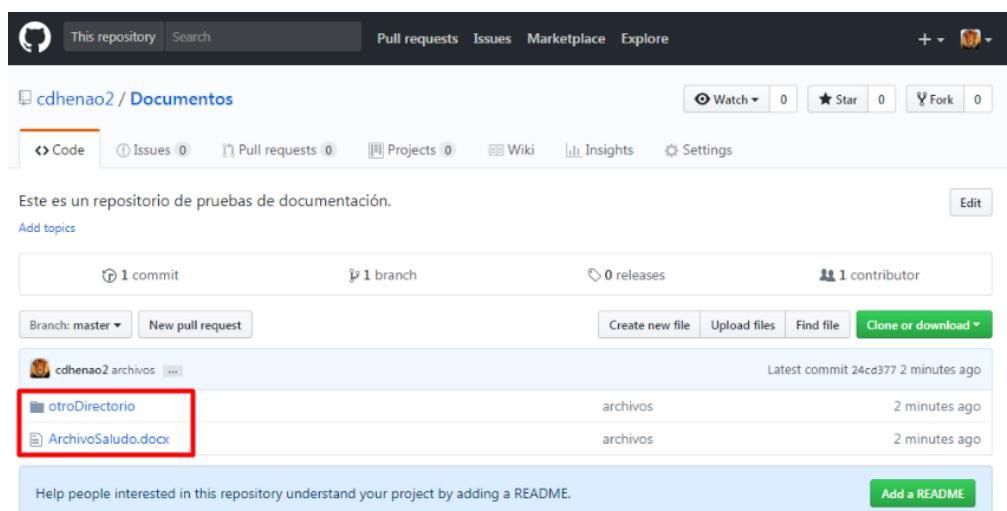


Luego de esto llenamos el recuadro inferior donde indicamos el mensaje del commit que realizaremos a nuestro repositorio local

Acto seguido presionamos la opción de **Publish branch**



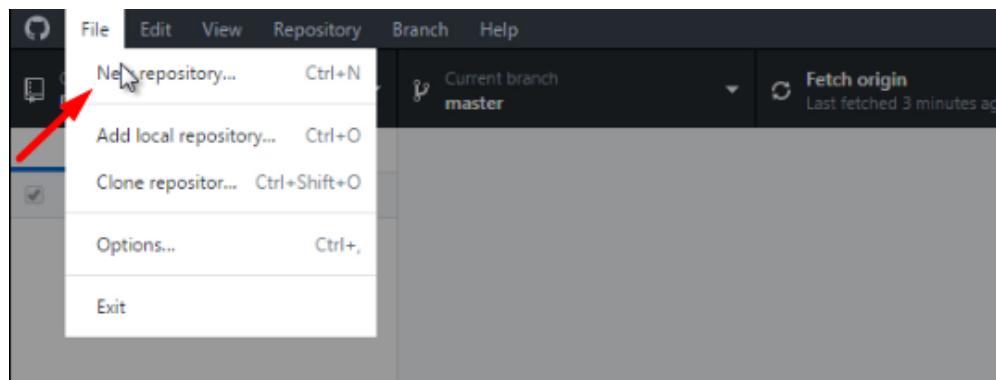
Después de esto buscamos en nuestra cuenta de GitHub el repositorio que vinculamos al cliente de escritorio y allí se verán alojados los archivos.



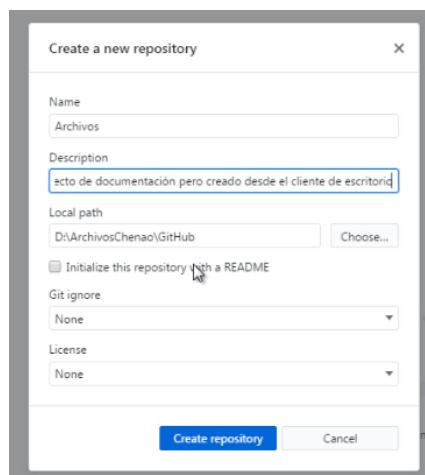
Si modificamos el contenido de los archivos automáticamente la herramienta identifica los cambios.

CREAR NUEVO REPOSITORIO

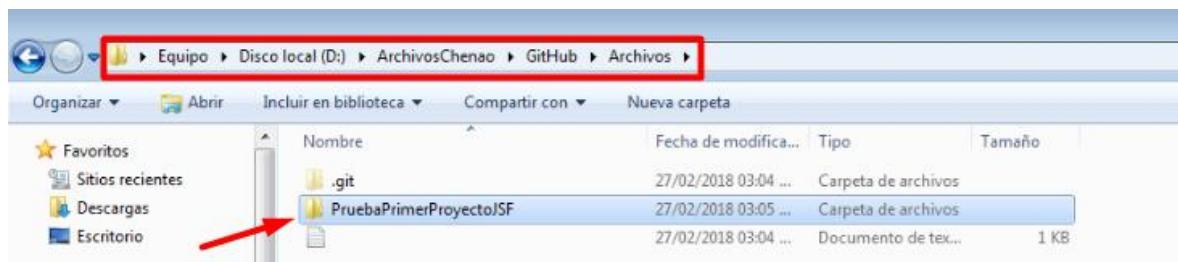
En el proceso anterior vinculamos a nuestro ambiente local un repositorio remoto que teníamos alojado en nuestra cuenta de GitHub, ahora vamos a hacer el proceso contrario, crearemos un repositorio local que será subido a nuestra cuenta GitHub, para eso vamos a file y lo creamos.



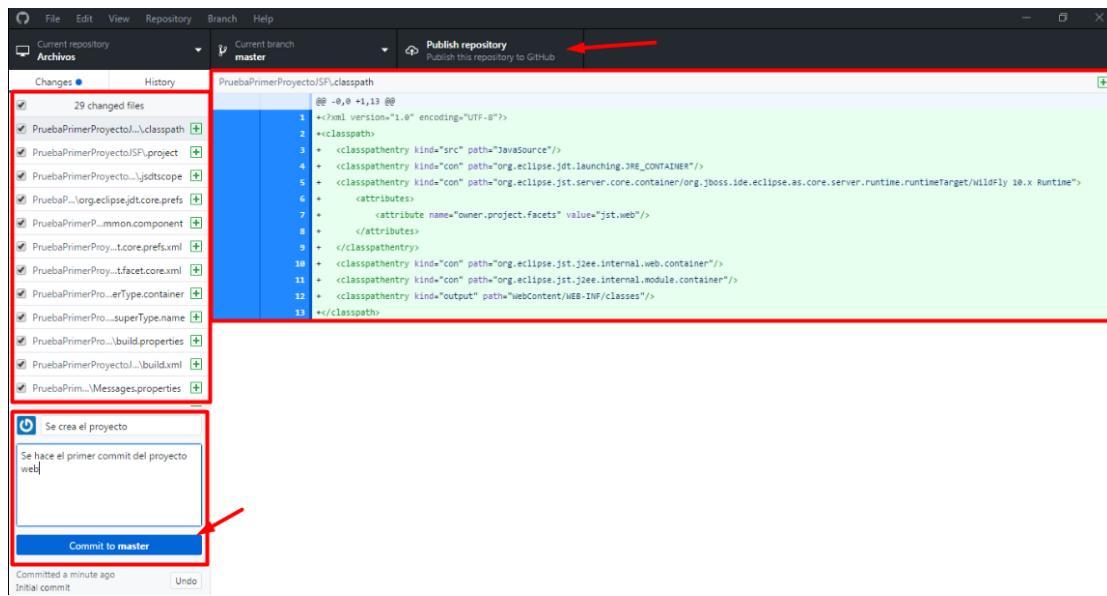
Al hacerlo se carga una ventana donde diligenciamos la información básica como nombre y descripción, así como las opciones de licencia de uso, adicionalmente podemos cambiar la ubicación por defecto, indicando la nueva ruta en donde queremos guardar el proyecto.



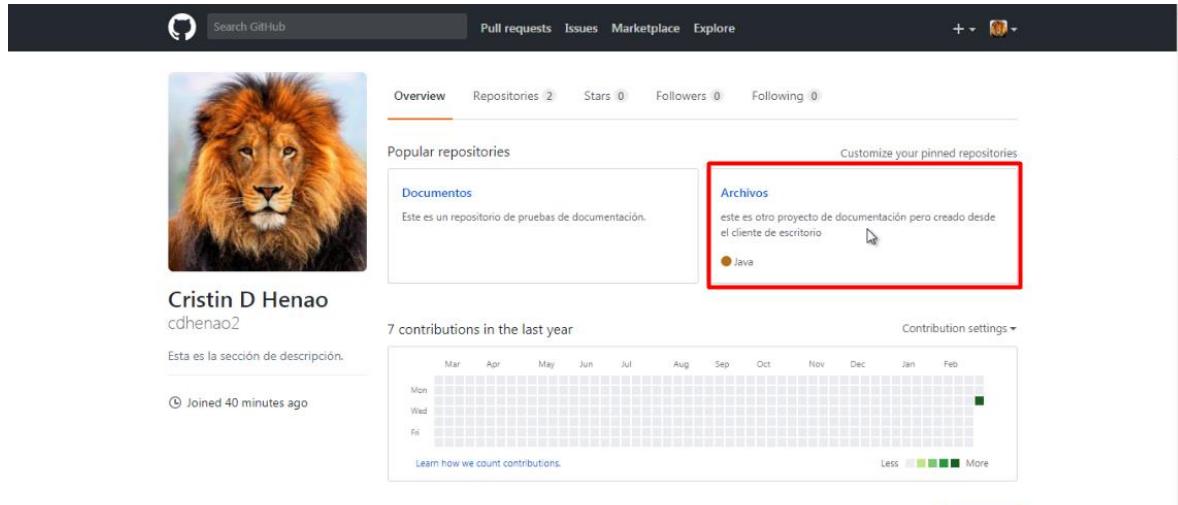
Al presionar el botón “Create repository” se genera una carpeta igual la que generamos al momento de clonar, allí podemos alojar diferentes archivos, en este caso vinculamos un proyecto web.



Cuando pasamos al cliente GitHub local veremos todos los archivos listos para subir.



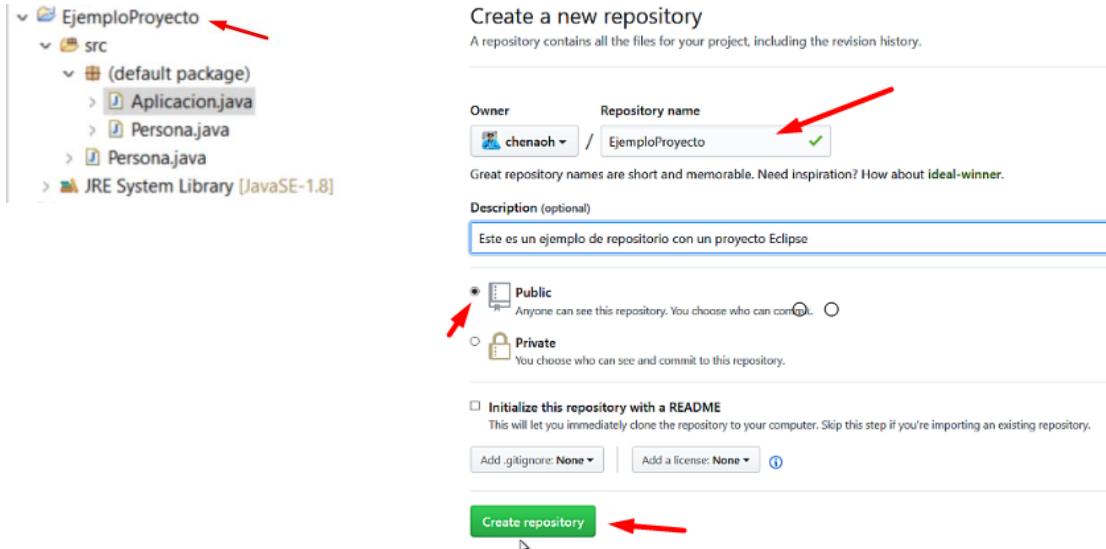
Cuando verificamos el repositorio remoto ya tendremos nuestro proyecto vinculado.



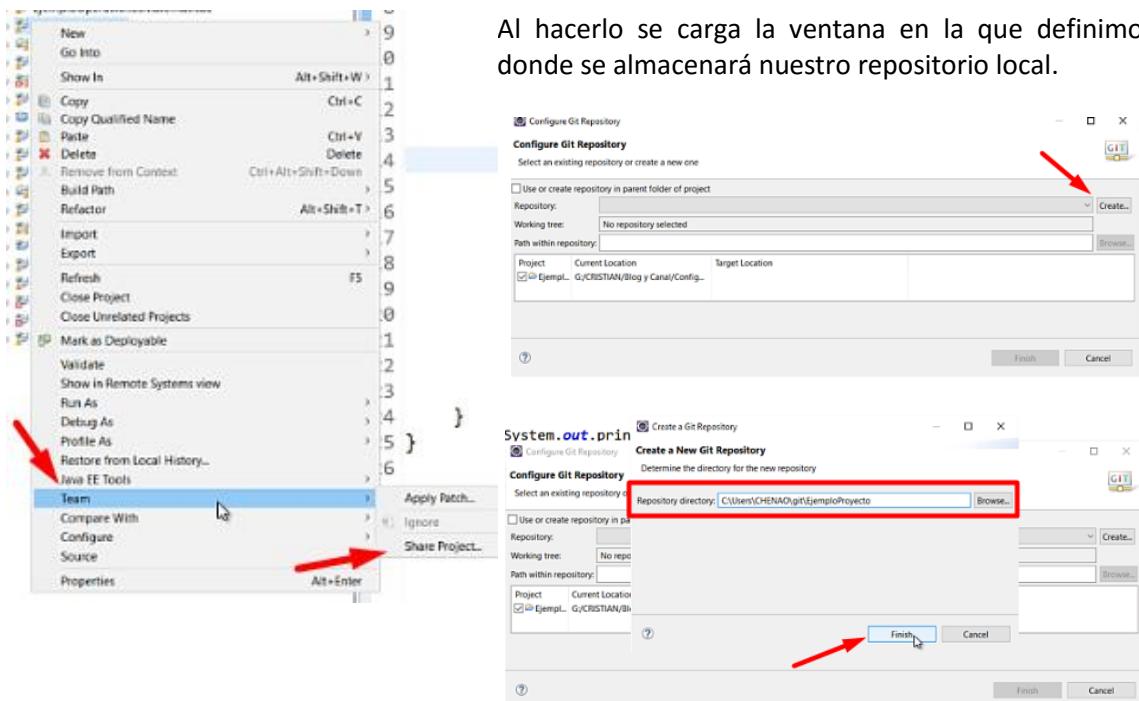
VINCULAR ECLIPSE CON GITHUB

En los procesos anteriores trabajamos con el cliente GitHub de escritorio, como vimos este nos facilita bastante el proceso a la hora de vincular los elementos al entorno remoto pero ahora vamos a trabajar directamente desde eclipse para gestionar nuestros proyectos java.

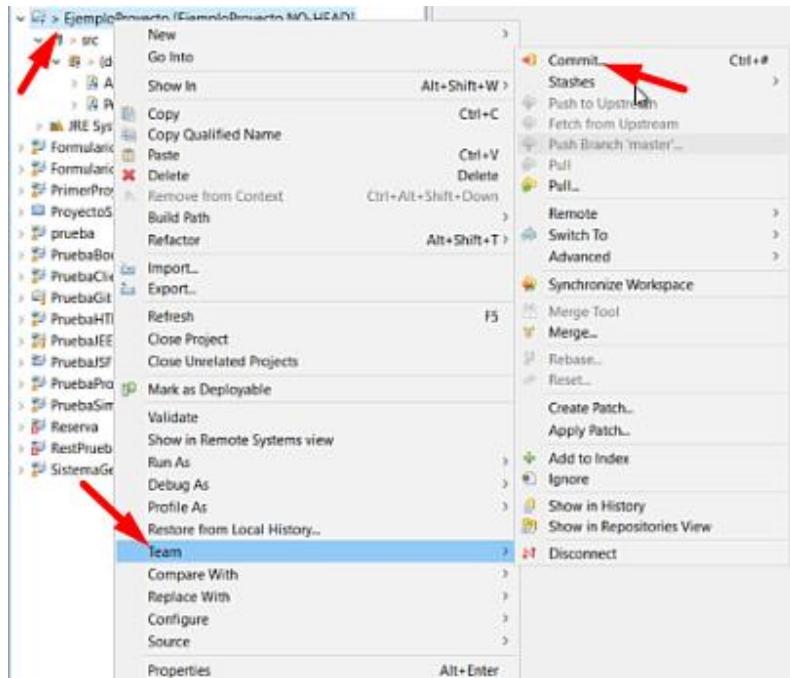
Inicialmente crearemos un nuevo repositorio, como convención podemos llamarlo con el mismo nombre del proyecto al que vamos a asociar.



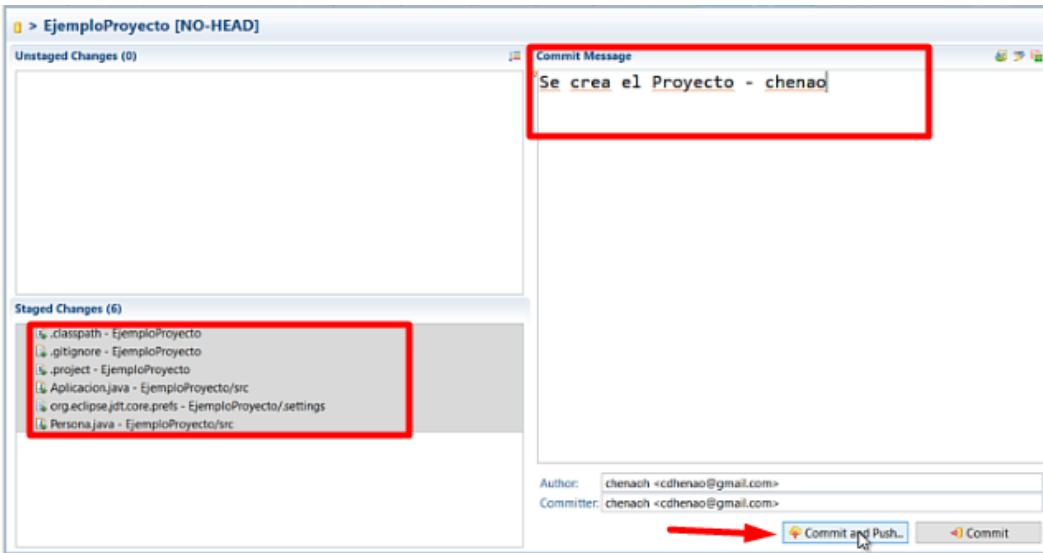
Después de haber creado el repositorio en GitHub vamos a eclipse y presionamos clic derecho en el proyecto a vincular, seleccionamos Team/Shared Project



Luego de esto nuevamente damos clic derecho al proyecto, seleccionamos Team/Commit

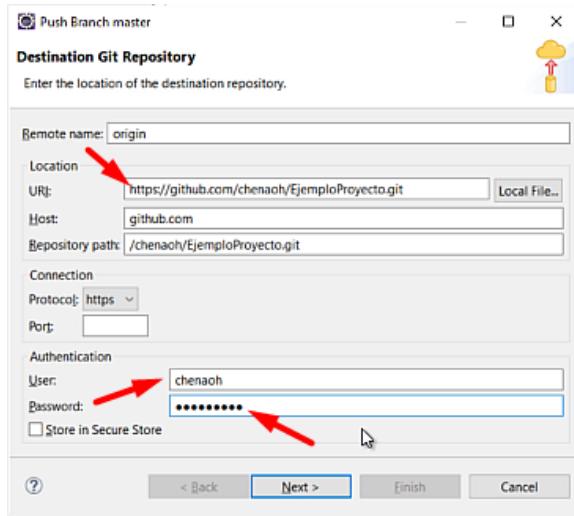


Al hacerlo se carga una nueva ventana donde seleccionamos los elementos que queremos vincular al repositorio y agregamos el mensaje del commit a enviar.



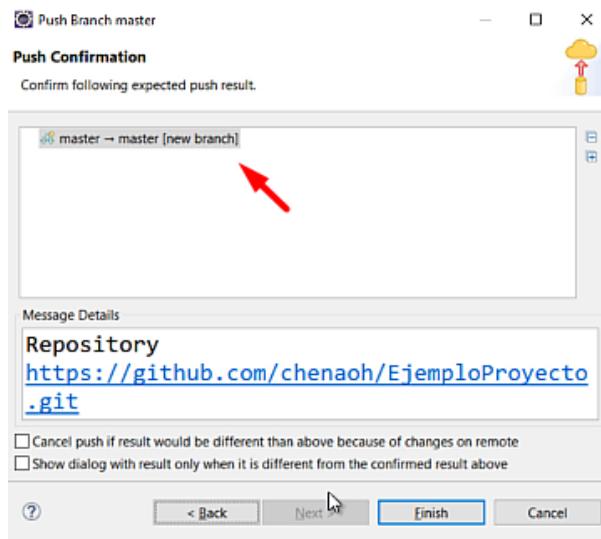
Presionamos el botón Commit and Push para vincular el proyecto y los cambios en nuestro repositorio Local y con el Push al repositorio remoto.

Al hacerlo se carga una ventana donde ingresamos la url del repositorio y las credenciales de ingreso



La url la obtenemos del repositorio remoto que creamos.

Al darle Next nos logueamos nuevamente y al final se presenta la ventana de confirmación indicando que ya ha sido cargado el proyecto en el repositorio.



Al revisar el repositorio encontramos el proyecto vinculado

Este es un ejemplo de repositorio con un proyecto Eclipse

Add topics

1 commit 1 branch 0 releases 1 contributor

Branch: master New pull request

chenaoh Se crea el Proyecto - chenao Latest commit c2ec99e a minute ago

EjemploProyecto Se crea el Proyecto - chenao a minute ago

Help people interested in this repository understand your project by adding a README. Add a README

chenaoh / EjemploProyecto

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Branch: master EjemploProyecto / EjemploProyecto / src /

chenaoh Se crea el Proyecto - chenao Latest commit c2ec99e a minute ago

..

Aplicacion.java Se crea el Proyecto - chenao a minute ago

Personajava Se crea el Proyecto - chenao a minute ago

Desde aquí ya podemos acceder a nuestro proyecto y toda la información que este contenga, si damos clic a la clase Aplicación.

Branch: master EjemploProyecto / EjemploProyecto / src / Aplicacion.java

chenaoh Se agrega cambio

contributors

26 lines (15 sloc) | 779 Bytes

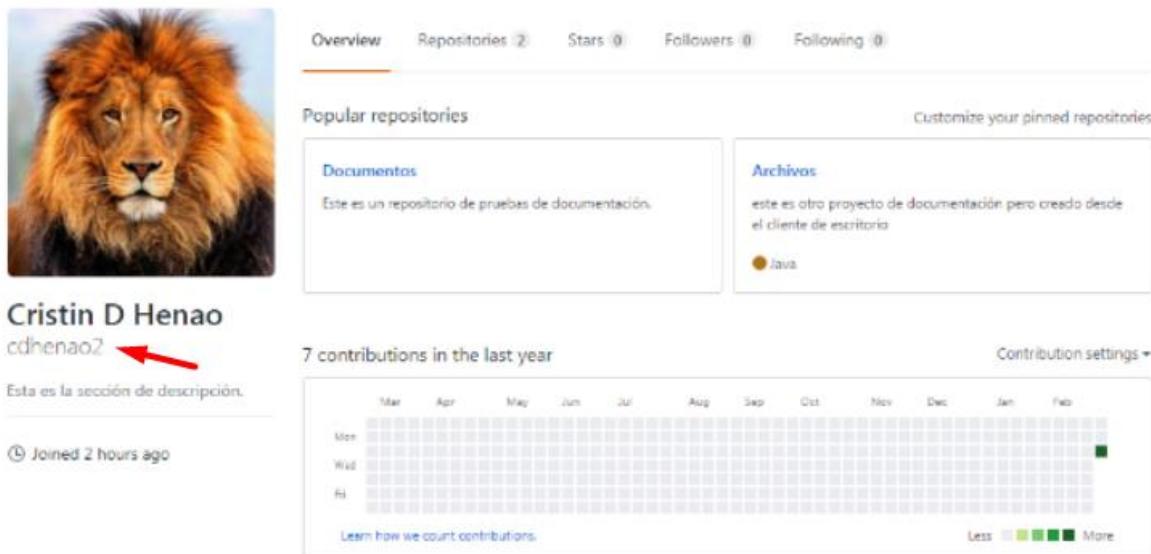
Raw Blame

```
1 import javax.swing.JOptionPane;
2
3 public class Aplicacion {
4
5     public static void main(String[] args) {
6
7         JOptionPane.showMessageDialog(null, "Se presentará la info de la persona!");
8
9         Persona miPersona=new Persona("Pepito Perez",23,"Estudiante","7584632");
10    }
```

CLONAR PROYECTO EN ECLIPSE

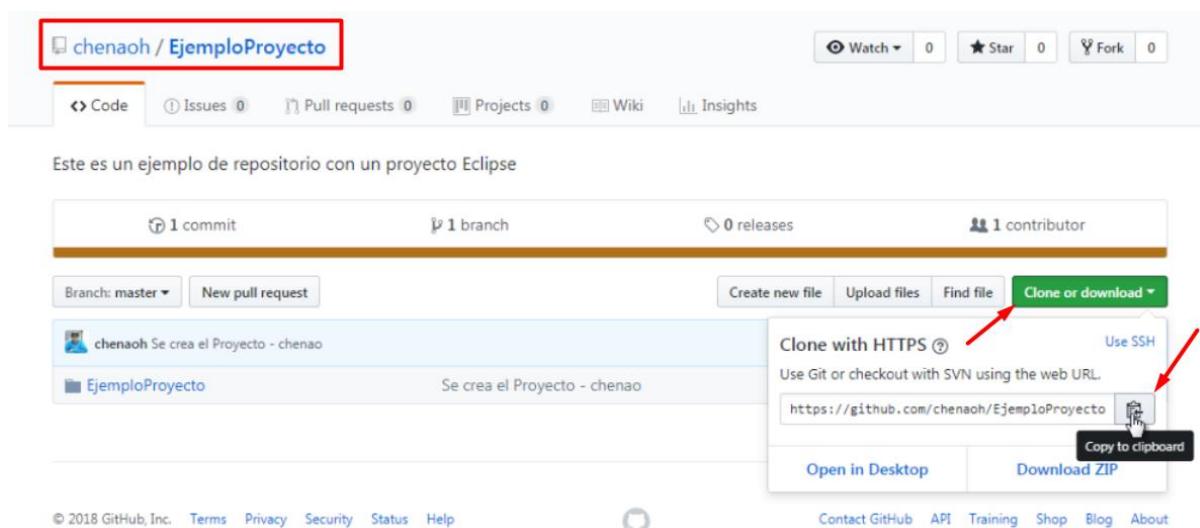
Ya vimos como subir un proyecto desde Eclipse a GitHub, ahora vamos a realizar el proceso de clonación del proyecto a otro espacio de trabajo con un usuario diferente.

Para que cada miembro del equipo de trabajo tenga acceso al repositorio, cada uno debe tener cuenta en GitHub, en este caso trabajaremos con el usuario **cdhenao2**



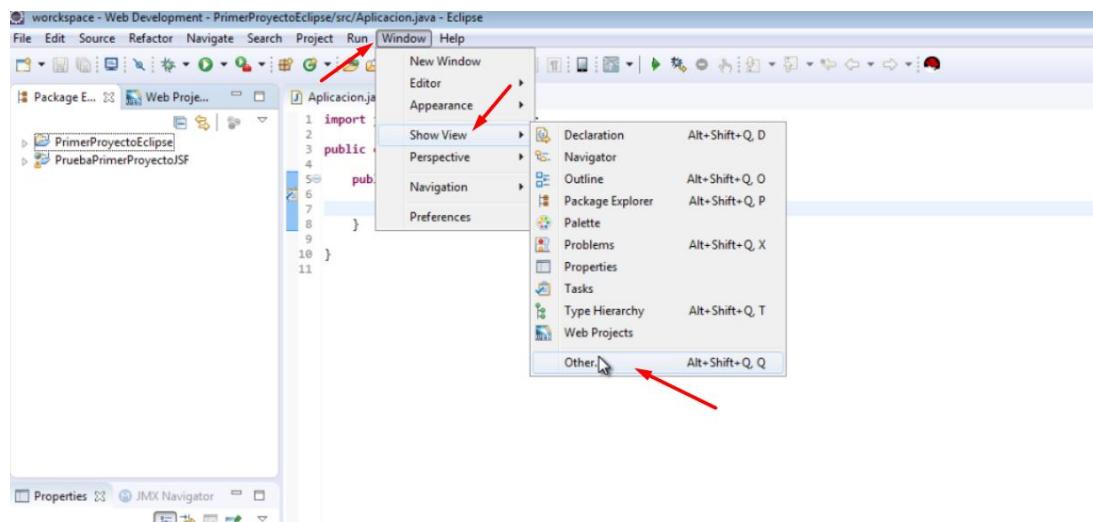
The screenshot shows the GitHub profile of 'Cristin D Henao'. At the top, there is a large profile picture of a lion. Below it, the user's name 'Cristin D Henao' and handle 'cdhenao2' are displayed. A red arrow points to the handle. The bio says 'Esta es la sección de descripción.' Below the bio, it says 'Joined 2 hours ago'. On the right, there are sections for 'Popular repositories' showing 'Documentos' and 'Archivos', and a 'Contribution settings' chart for the last year.

Posteriormente desde este usuario buscamos el repositorio que queremos clonar para copiar el enlace del mismo, en este caso ingresamos al GitHub del usuario que tiene el repositorio original, damos clic en el botón “Clone or download” desde el cual podemos descargar el proyecto como un .zip, abrirlo en nuestro cliente de escritorio o copiar el código para clonarlo en el entorno de desarrollo (Eclipse)

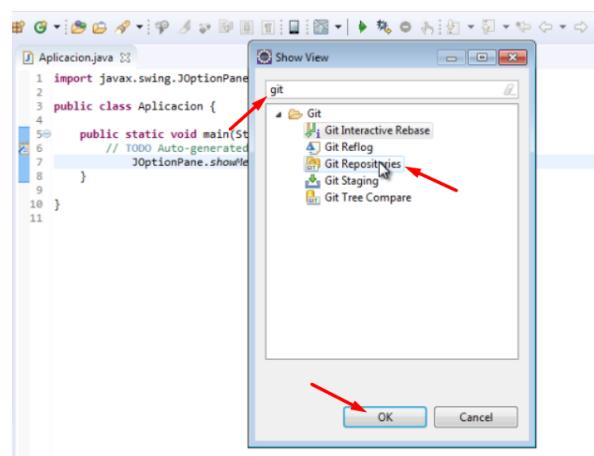


The screenshot shows the GitHub repository page for 'chenaoh / EjemploProyecto'. The repository has 1 commit, 1 branch, 0 releases, and 1 contributor. The 'Clone or download' button is highlighted with a red arrow. A dropdown menu from this button shows options like 'Clone with HTTPS', 'Use SSH', 'Copy to clipboard', 'Open in Desktop', and 'Download ZIP'. Another red arrow points to the 'Copy to clipboard' link in the dropdown.

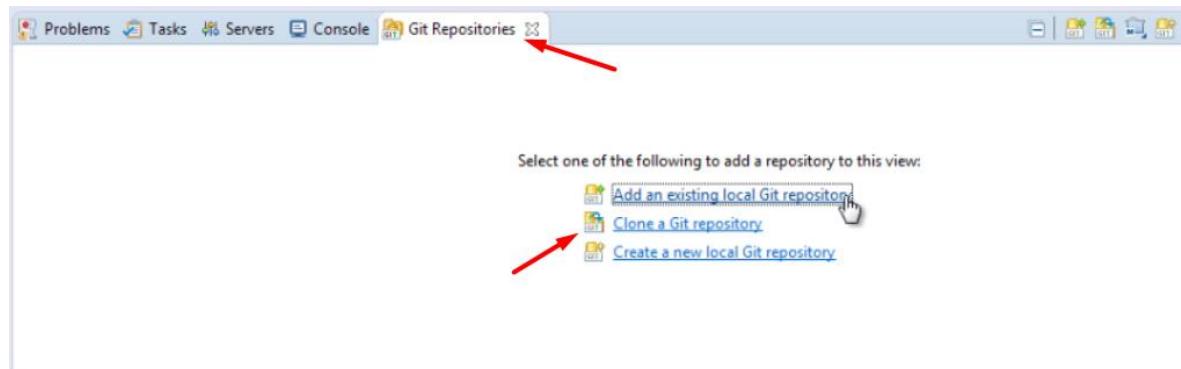
Posteriormente vamos a nuestro entorno Eclipse y buscamos la vista “Git Repositories” en Window>Show View/Other



Al hacerlo se carga una ventana en la que definimos la vista.

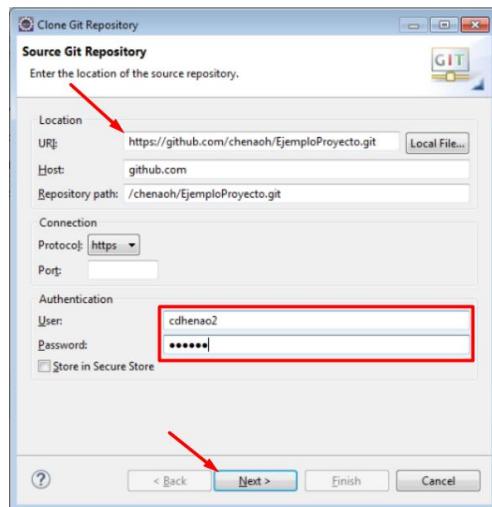


Al dar clic en la opción, nos carga el panel de Repositorios donde inicialmente nos aparecen las 3 opciones (agregar, clonar o crear) que también se presentó en el cliente de escritorio, en nuestro caso vamos a clonar el repositorio directamente desde GitHub.

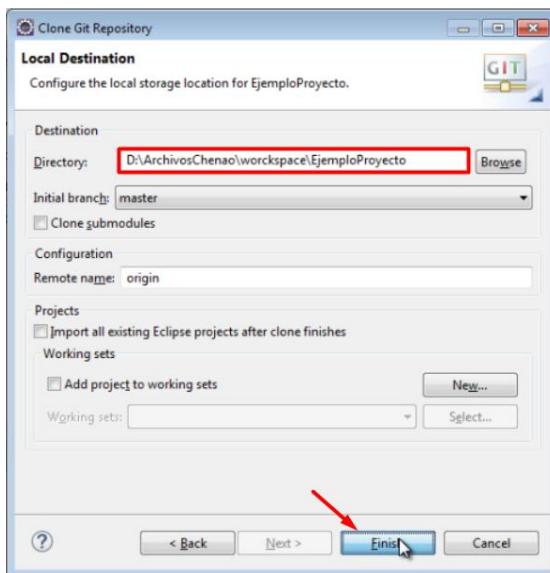


Después de esto se carga una ventana donde debemos ingresar la ruta del repositorio que vamos a clonar, si antes de realizar estos pasos copiamos la ruta desde GitHub entonces esta aparecerá previamente en el formulario, sino entonces procedemos a copiarla y pegarla en el campo correspondiente.

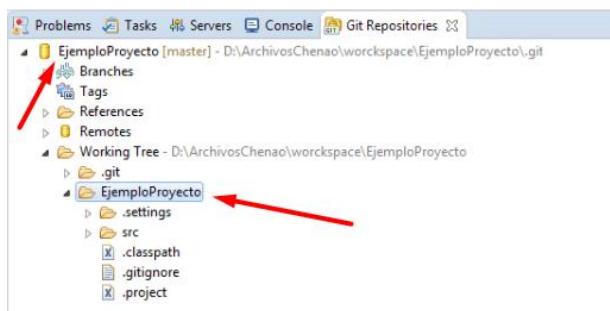
Allí al copiar la URL automáticamente se actualiza el campo host y el Path, al final ingresamos el Usuario y Password y damos next next.



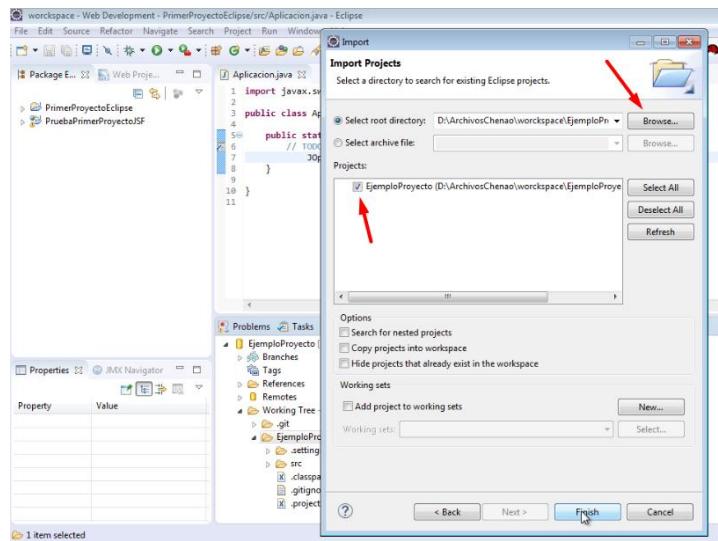
Después de un punto se presenta la siguiente ventana donde definimos el directorio donde queremos almacenar el proyecto clonado, por defecto el sistema asigna una ruta pero lo ideal es poderla cambiar a la ruta de nuestro workspace.



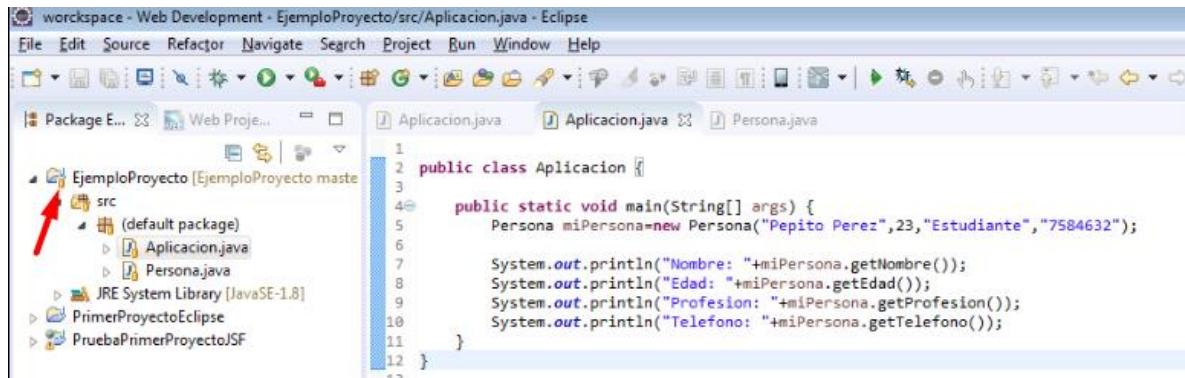
Al darle finish ya podemos verificar el repositorio donde se encuentra nuestro proyecto



Después de esto podemos importar el proyecto a nuestro entorno y verificar su funcionamiento.



Al hacerlo comprobamos que corresponde al proyecto almacenado en GitHub y que se presenta un icono indicando que está siendo gestionado en un sistema de control de versiones.

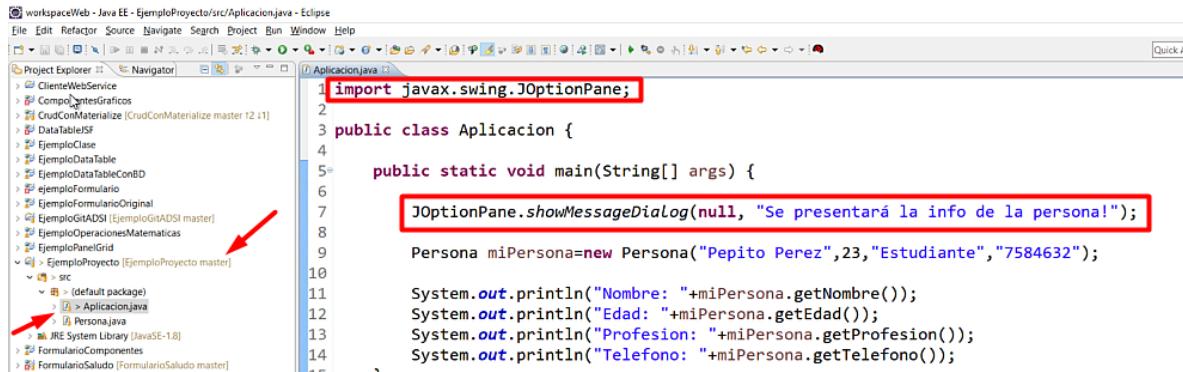


Como vimos ya tenemos el proyecto clonado, esto genera automáticamente una copia Local del repositorio en la que podemos trabajar, luego cada vez que hagamos un cambio podemos compartirlo con el equipo de trabajo al subirlo al repositorio Remoto una vez tengamos los permisos correspondientes.

SUBIR CAMBIOS DESDE ECLIPSE A REPOSITORIO REMOTO.

Ya vimos como subir un proyecto base a nuestro repositorio **GitHub**, también vimos como desde otro usuario podíamos descargar dicho proyecto, ahora vamos a ver como subir cambios al proyecto y que el otro usuario los pueda actualizar (modificaciones o archivos nuevos al proyecto).

Inicialmente realizamos la modificación que vamos a hacer a nuestro código y la guardamos, como vemos en el proyecto se marca la clase modificada con un símbolo “>” lo que nos indica que hay una modificación en nuestro archivo.



```

workspaceWeb - Java EE - EjemploProyecto/src/Aplicacion.java - Eclipse
File Edit Refactor Source Navigate Search Project Run Window Help
Project Explorer Navigator
> ClienteWebService
> CompoInterGraficos
> CrudConMaterialize [CrudConMaterialize master t2 t1]
> DataTableJSF
> EjemploClase
> EjemploDataTable
> EjemploDataTableConBD
> EjemploFormulario
> EjemploFormularioOriginal
> EjemploGitADS [EjemploGitADS master]
> EjemploOperacionesMatematicas
> EjemploPanelGrid
> EjemploProyecto [EjemploProyecto master]
  > src
    > (default package)
      > Aplicacion.java
      > Persona.java
    > JRE System Library [JavaSE-1.8]
  > FormularioComponentes
> FormularioSaludo [FormularioSaludo master]

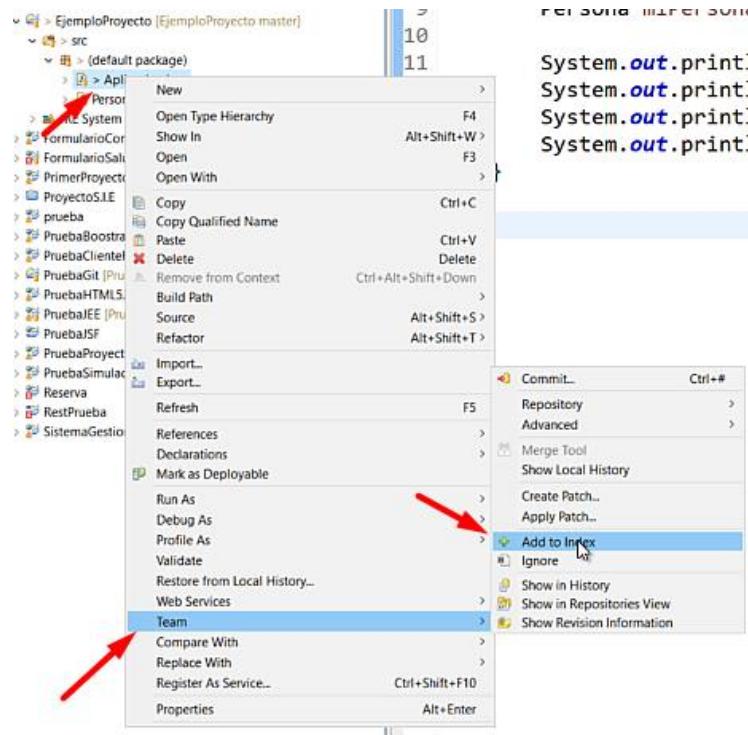
```

```

1 import javax.swing.JOptionPane;
2
3 public class Aplicacion {
4
5     public static void main(String[] args) {
6
7         JOptionPane.showMessageDialog(null, "Se presentará la info de la personal");
8
9         Persona miPersona=new Persona("Pepito Perez",23,"Estudiante","7584632");
10
11         System.out.println("Nombre: "+miPersona.getNombre());
12         System.out.println("Edad: "+miPersona.getEdad());
13         System.out.println("Profesion: "+miPersona.getProfesion());
14         System.out.println("Telefono: "+miPersona.getTelefono());

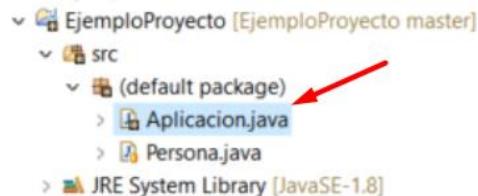
```

El siguiente paso es agregar dicho archivo modificado al index para eso damos clic derecho en el archivo/Team/add to index

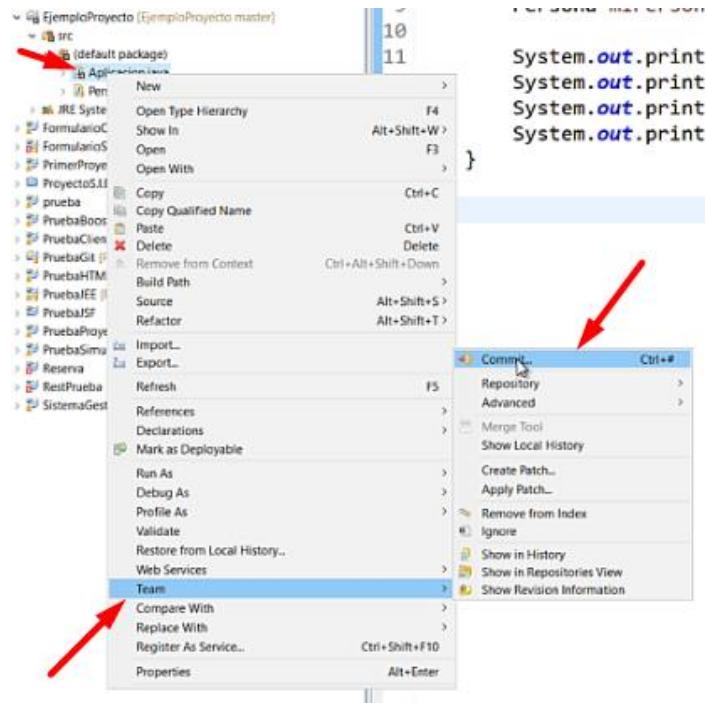


Este proceso se hace ya que como se mencionó al principio de esta guía, debemos pasar por 3 ramas base “Espacio de trabajo, index y heap”

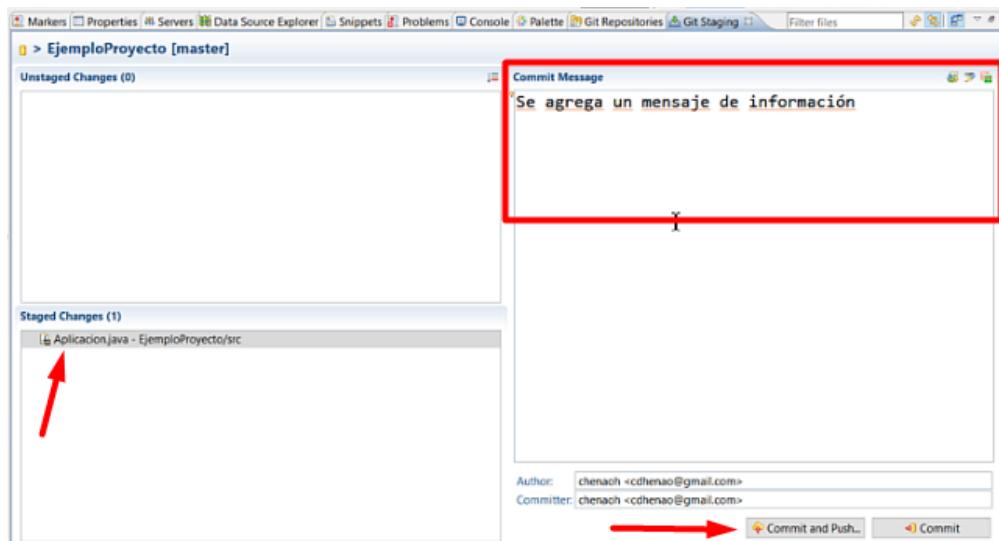
Al hacer este cambio vemos como se actualiza el archivo indicándonos que ya se encuentra vinculado.



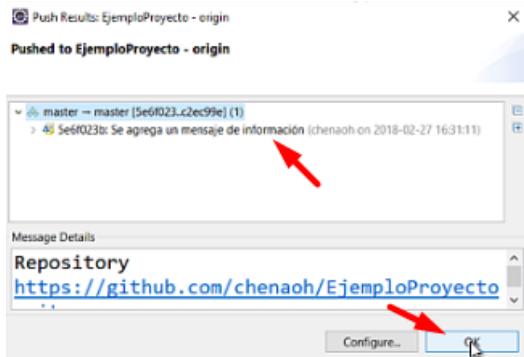
Luego de esto podemos hacer el commit a nuestro repositorio.



En la siguiente vista debemos agregar el mensaje indicando el cambio y presionar Commit and Push



El sistema nos pedirá que confirmemos el usuario y contraseña (este proceso puede realizarse en diferentes oportunidades) y posteriormente procederá a enviar la modificación confirmándonos el proceso como se muestra en la siguiente imagen.



Como vemos allí nos muestra el repositorio remoto, el cual, si consultamos ya tendrá reflejado el cambio realizado.

A screenshot of a GitHub repository page for 'EjemploProyecto / EjemploProyecto / src /'. The branch is 'master'. It shows two commits: 'chenao' (35 seconds ago) which adds a message to 'Aplicacion.java', and 'Personaje.java' (an hour ago) which creates the project. A red arrow points to the commit message for 'chenao' and another red arrow points to the file name 'Aplicacion.java'.

Y al ingresar al archivo en cuestión podremos ver la nueva modificación.

A screenshot of a GitHub file view for 'Aplicacion.java'. The code is as follows:

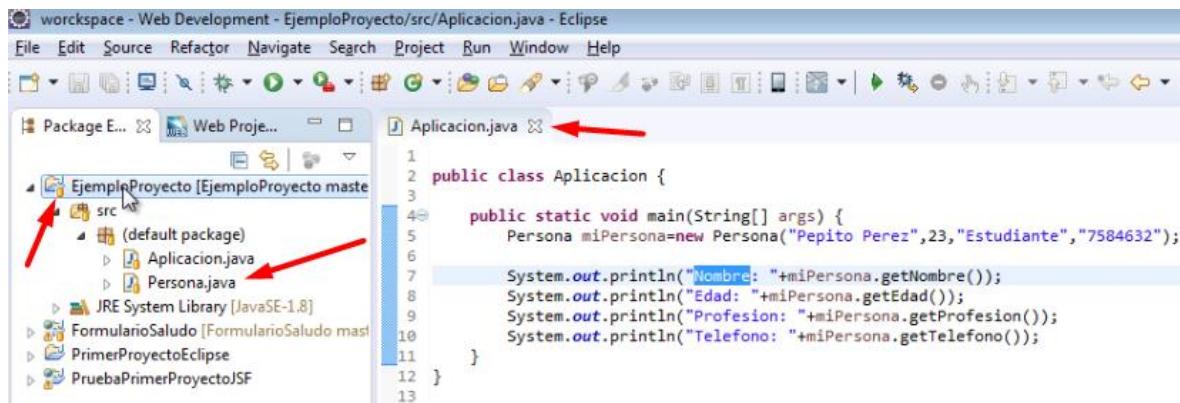
```
1 import javax.swing.JOptionPane;
2
3 public class Aplicacion {
4
5     public static void main(String[] args) {
6
7         JOptionPane.showMessageDialog(null, "Se presentará la info de la persona!");
8
9         Persona miPersona=new Persona("Pepito Perez",23,"Estudiante","7584632");
10
11         System.out.println("Nombre: "+miPersona.getNombre());
12         System.out.println("Edad: "+miPersona.getEdad());
13         System.out.println("Profesion: "+miPersona.getProfesion());
14         System.out.println("Telefono: "+miPersona.getTelefono());
15     }
16 }
```

A red arrow points to the line of code: 'JOptionPane.showMessageDialog(null, "Se presentará la info de la persona!");'.

Y de esta manera ya realizamos un cambio en el repositorio remoto que los demás usuarios pueden ver, sin embargo tengamos en cuenta que este cambio fue realizado desde el usuario creador del repositorio, veamos ahora como actualizar los cambios desde el otro usuario y que pasa si este intenta realizar un cambio también.

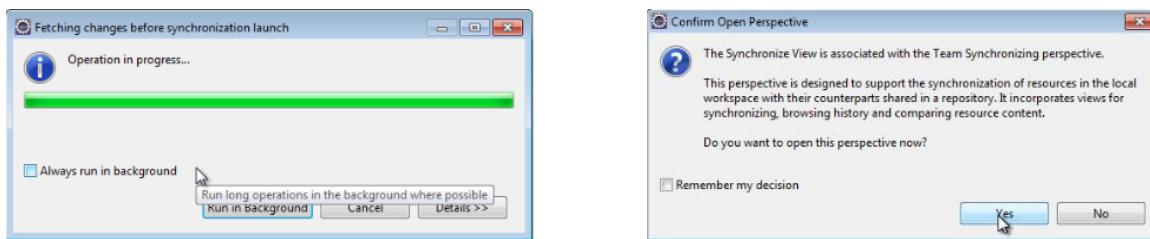
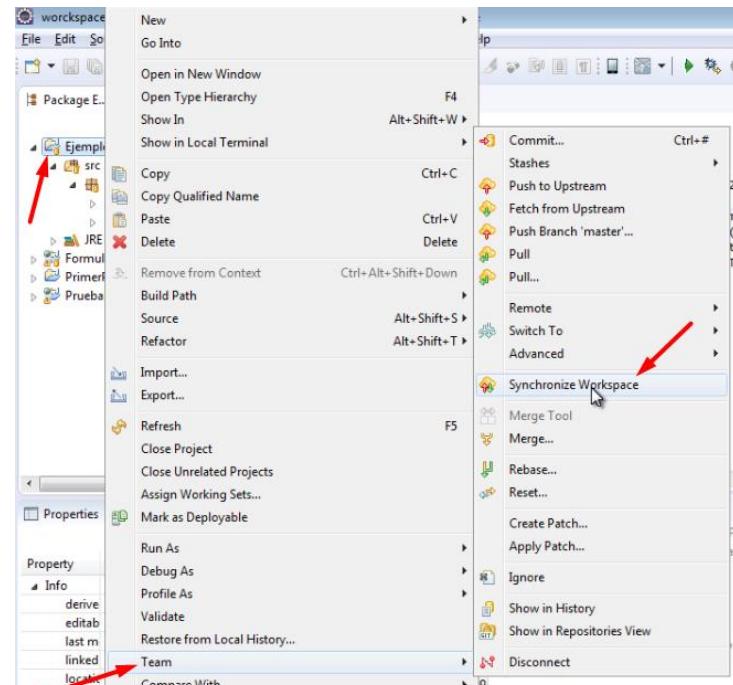
SUBIR CAMBIOS AL REPOSITORIO – USUARIO COLABORADOR.

En la sección anterior modificamos la clase Aplicacion.java, allí agregamos unas líneas de código, estas son suficientes para decirle al sistema que algo pasó, sin embargo cuando el segundo usuario ingresa al proyecto se encuentra con que los cambios no han sido reflejados.

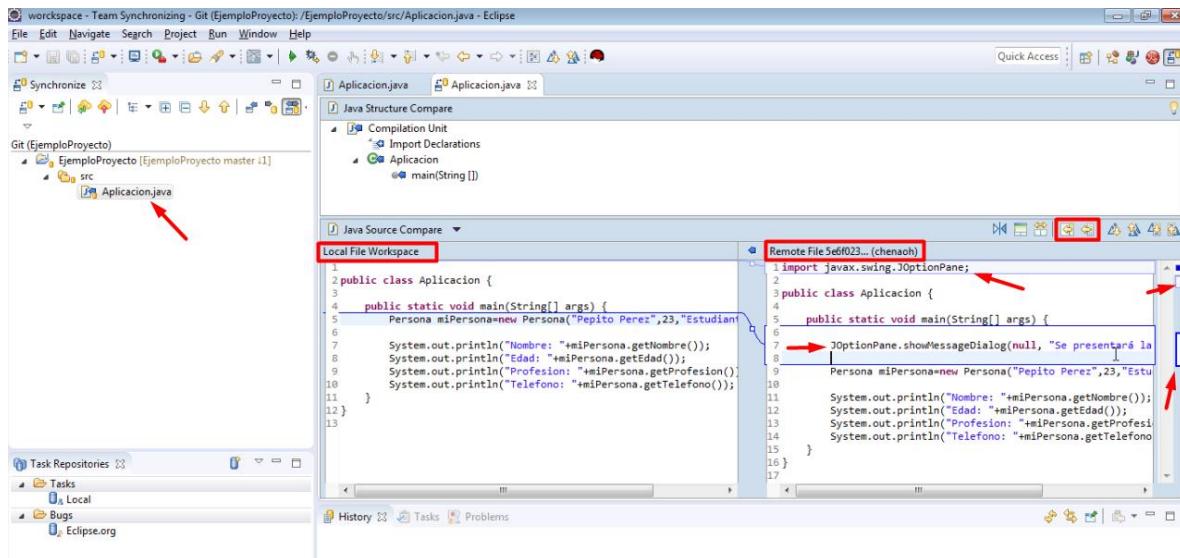


Esto se da porque para que podamos tener los cambios primero se debe revisar que cambios son para poderse actualizar.

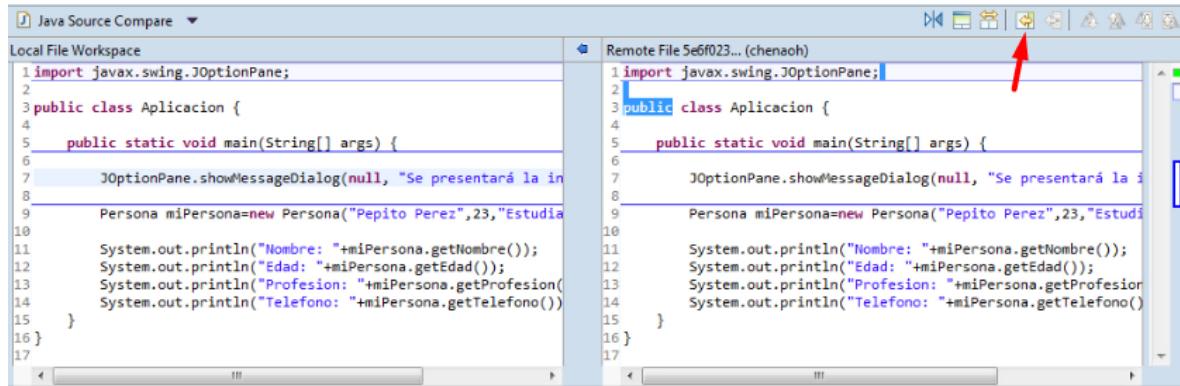
para hacer esto debemos darle clic derecho al proyecto/Team/Synchronize Workspace



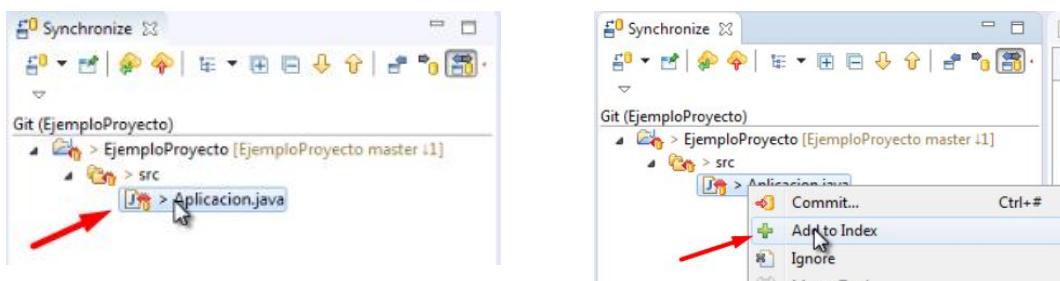
Cuando sincronizamos se presenta la perspectiva donde verificamos cual fue el archivo cambiado y podemos comparar que tenemos en nuestro ambiente remoto vs que tenemos en el ambiente remoto.



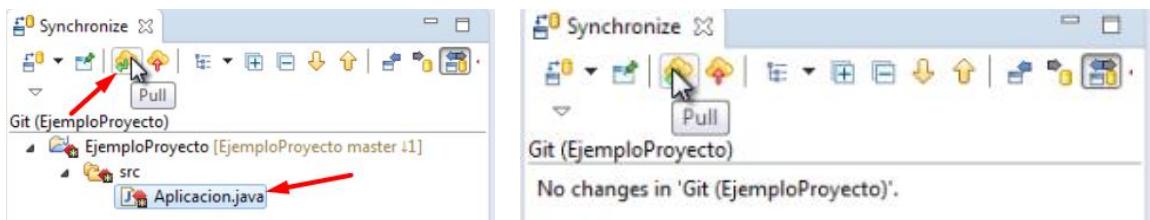
Desde aquí podemos empezar a pasar los cambios, para eso usamos los iconos de la sección superior derecha donde podemos pasar todos o elegir uno a uno lo que queremos mover.



Cuando guardemos el cambio vemos que el ícono de nuestra clase cambia indicando que posiblemente hay un conflicto, para solucionar esto debemos agregar el cambio a nuestro entorno local dando clic derecho/Add to Index.



Y luego de esto damos clic en Pull y el sistema indicará que ya no hay cambios con respecto a la versión remota.



Y al final tenemos los cambios reflejados

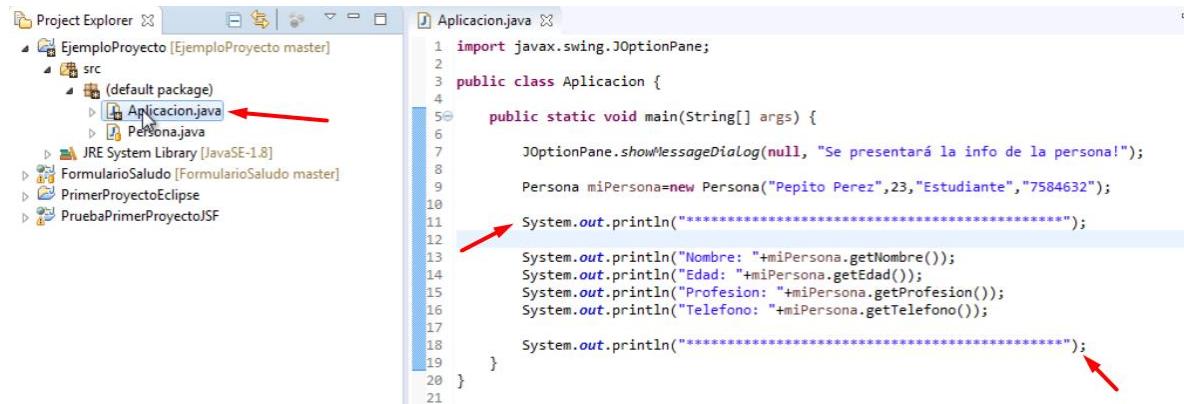
```
1 import javax.swing.JOptionPane;
2
3 public class Aplicacion {
4
5     public static void main(String[] args) {
6
7         JOptionPane.showMessageDialog(null, "Se presentará la info de la persona!");
8
9         Persona miPersona=new Persona("Pepito Perez",23,"Estudiante","7584632");
10
11        System.out.println("Nombre: "+miPersona.getNombre());
12        System.out.println("Edad: "+miPersona.getEdad());
13        System.out.println("Profesion: "+miPersona.getProfesion());
14        System.out.println("Telefono: "+miPersona.getTelefono());
```

The image shows a code editor window with Java code. Lines 1 through 14 are visible. Lines 1, 7, and 14 are highlighted with a red rectangular box. Line 7 contains the text 'JOptionPane.showMessageDialog(null, "Se presentará la info de la persona!");'. The code defines a class 'Aplicacion' with a main method that creates a new 'Persona' object and prints its details to the console.

ASIGNAR PERMISOS A USUARIOS

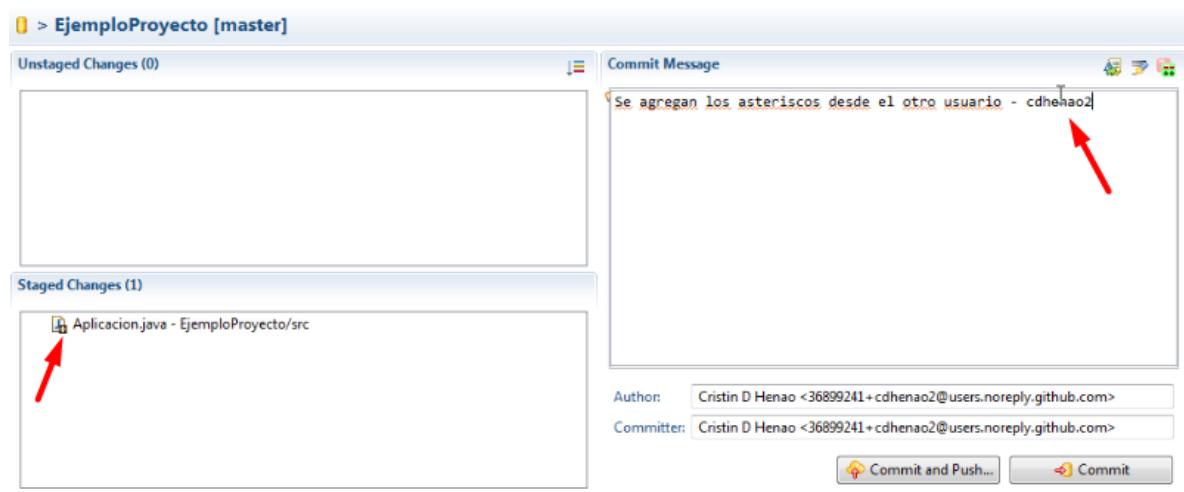
Ya realizamos commit desde el usuario inicial y bajamos los cambios desde el segundo usuario, ahora vamos a hacer lo mismo pero desde la perspectiva del segundo usuario intentando subir un cambio desde su ambiente local.

Para eso realizamos un cambio cualquiera y lo agregamos al index

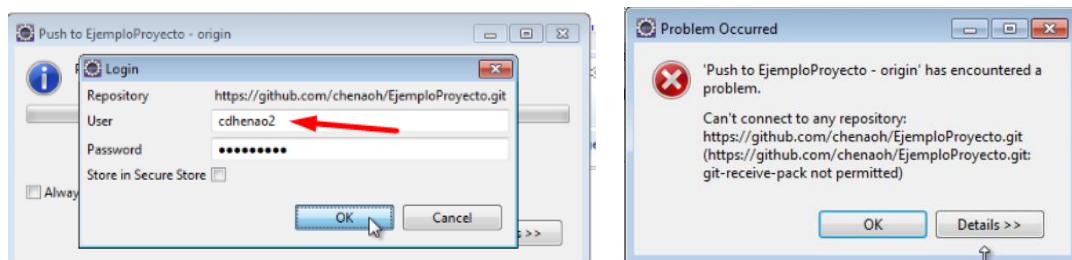


```
Aplicacion.java
1 import javax.swing.JOptionPane;
2
3 public class Aplicacion {
4     public static void main(String[] args) {
5         JOptionPane.showMessageDialog(null, "Se presentará la info de la persona!");
6         Persona miPersona=new Persona("Pepito Perez",23,"Estudiante","7584632");
7         System.out.println("*****");
8         System.out.println("Nombre: "+miPersona.getNombre());
9         System.out.println("Edad: "+miPersona.getEdad());
10        System.out.println("Profesion: "+miPersona.getProfesion());
11        System.out.println("Telefono: "+miPersona.getTelefono());
12        System.out.println("*****");
13    }
14 }
```

Luego hacemos el commit normalmente



Ingresamos las credenciales del segundo usuario y al darle ok se presenta un mensaje de error.



Esto se da porque no tenemos permisos de escritura sobre el repositorio remoto

Para solucionar este inconveniente debemos solicitar que se agreguen los permisos desde el usuario que creó el repositorio ya que este es el dueño o administrador, esto lo hace dicho usuario ingresando al repositorio desde su cuenta y dando clic en “Settings”

A screenshot of a GitHub repository page for 'chenaoh / EjemploProyecto'. The top navigation bar shows the repository name and basic stats: 1 commit, 1 branch, 0 releases, and 1 contributor. Below the stats, there are buttons for 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. The 'Settings' tab is highlighted with a red box. The main content area displays a commit history with two entries: 'Se crea el Proyecto - chenao' and 'EjemploProyecto'.

Posteriormente damos clic en la opción “Collaborators”

A screenshot of the 'Settings' page for the same repository. On the left, a sidebar menu has 'Collaborators' highlighted with a red box. The main area contains sections for 'Repository name' (set to 'EjemploProyecto') and 'Features'. A 'Collaborators' section is present, showing a message: 'This repository doesn't have any collaborators yet. Use the form below to add a collaborator.' Below this is a search bar with 'cdhenao' typed in, and a dropdown menu showing suggestions: 'cdhenao', 'cdhenao2 Cristin D Henao', and 'cdhenao3'. A red arrow points to the suggestion 'cdhenao2 Cristin D Henao'.

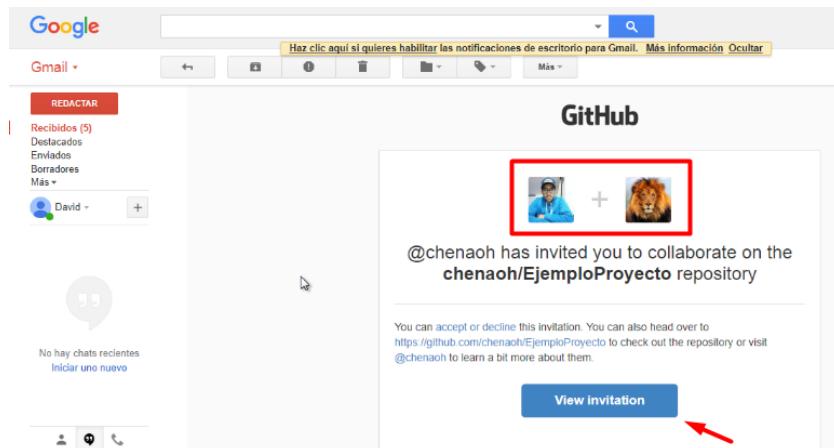
Después de esto buscamos el usuario al que queremos agregar al repositorio y presionamos “Add collaborator”

A screenshot of the 'Collaborators' search results. The search bar shows 'cdhenao'. Below it, a list of users includes 'cdhenao2 Cristin D Henao' with a small profile icon next to it. A red arrow points to this user entry.

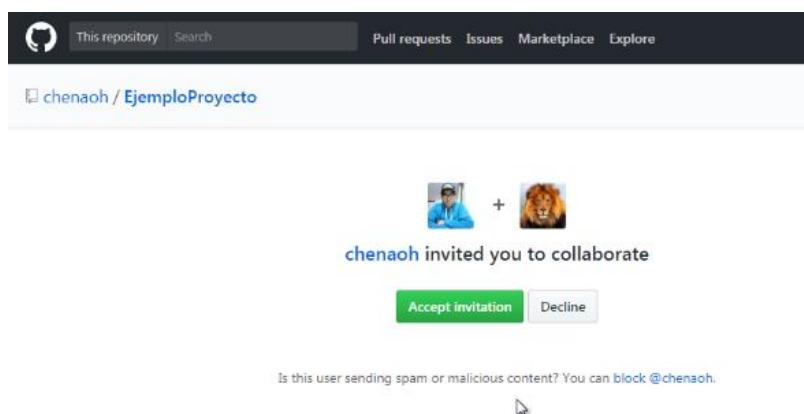
Así podremos agregar la cantidad de colaboradores que necesitemos a los que se les enviará un correo de confirmación.

A screenshot of the 'Collaborators' section after adding a collaborator. It shows a list with one item: 'Cristin D Henao' with a small profile icon, followed by the text 'Awaiting cdhenao2's response'. To the right of this item are 'Copy invite link' and 'Cancel invite' buttons. A red arrow points to the user 'Cristin D Henao'.

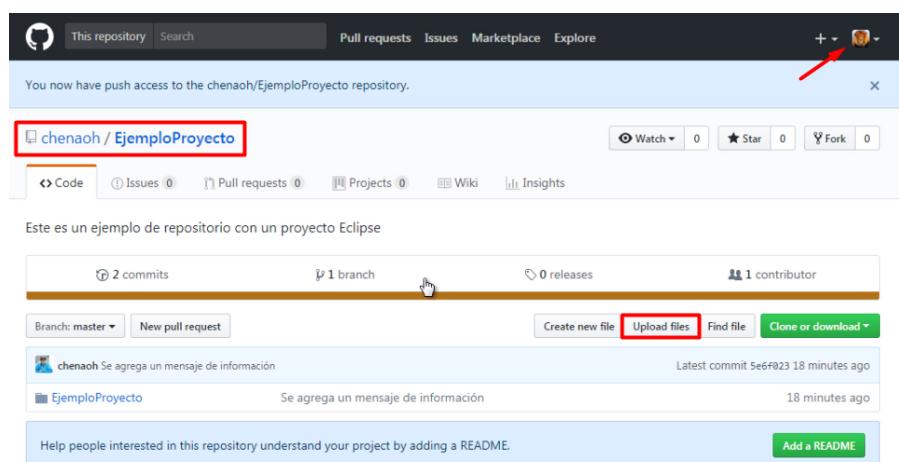
Posteriormente cada usuario al que se le ha invitado a colaborar o contribuir con el proyecto deberá aceptar la invitación desde el correo electrónico.



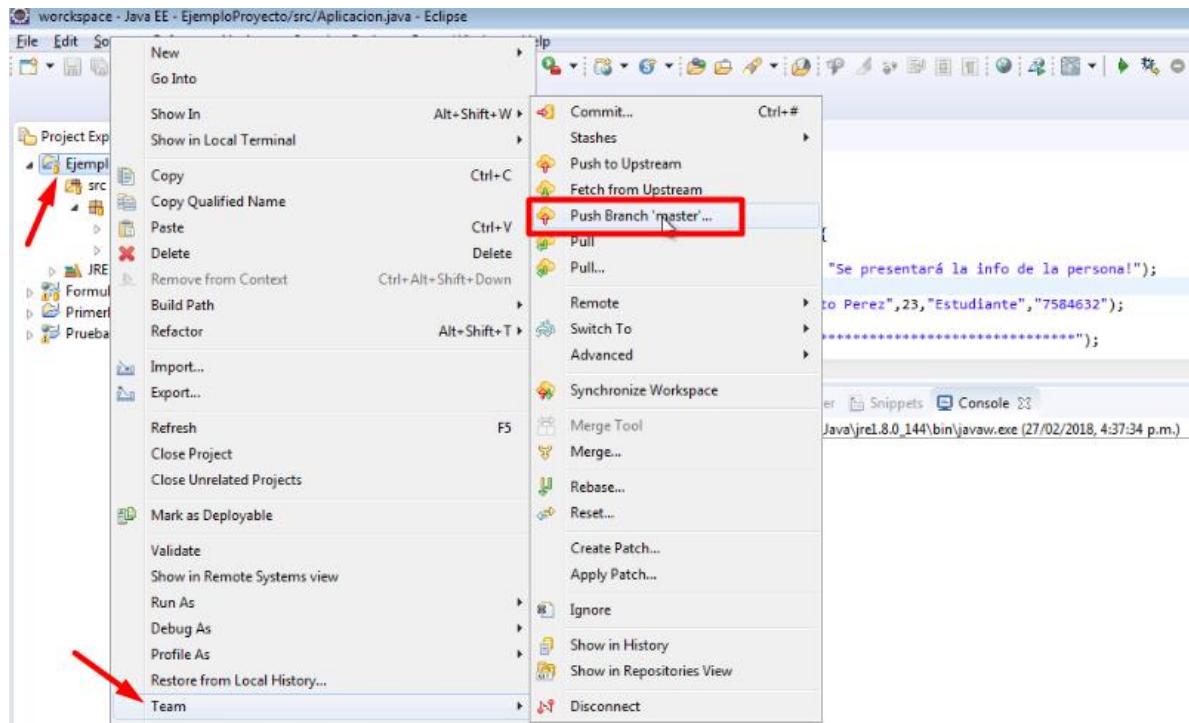
Al presionar el botón el sistema redirecciona a la cuenta de GitHub donde podremos aceptar la invitación o rechazarla.



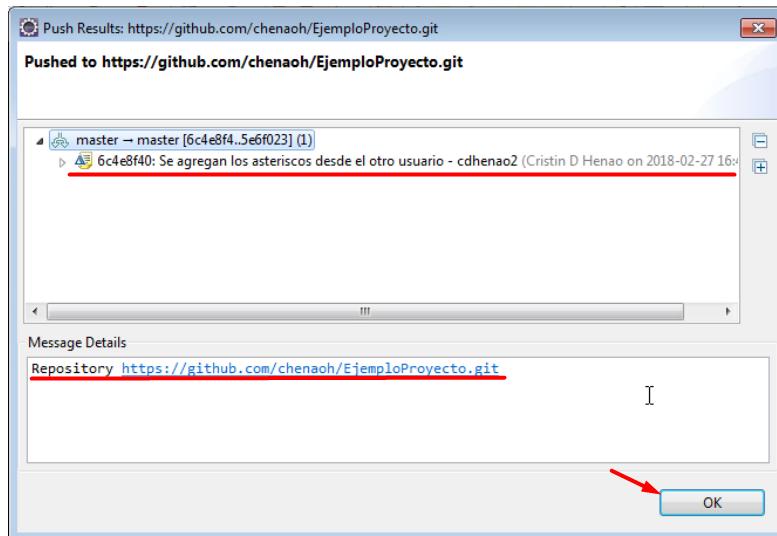
Al aceptar tendremos acceso al repositorio con permisos de modificación.



Ahora si regresamos a nuestro workspace ya podremos subir los cambios al repositorio, recordemos que previamente hicimos el commit pero salió el error cuando el sistema intentó hacer el push, por lo tanto solo hacemos el push dando clic derecho en el proyecto/Team/Push Branch 'master'...



Hacemos el mismo proceso de validar las credenciales y al final tenemos el mensaje de confirmación



Si vamos al repositorio veremos que ya se realizó el commit desde el usuario 2 con los cambios correspondientes.

Branch: master → EjemploProyecto / EjemploProyecto / src / Aplicacion.java ←

cdhenao2 Se agregan los asteriscos desde el otro usuario - cdhenao2 ← 6c4e8f4 12 minutes ago

2 contributors

21 lines (13 sloc) | 670 Bytes Raw Blame History

```
1 import javax.swing.JOptionPane;
2
3 public class Aplicacion {
4
5     public static void main(String[] args) {
6
7         JOptionPane.showMessageDialog(null, "Se presentará la info de la persona!");
8
9         Persona miPersona=new Persona("Pepito Perez",23,"Estudiante","7584632");
10
11        System.out.println("*****");
12
13        System.out.println("Nombre: "+miPersona.getNombre());
14        System.out.println("Edad: "+miPersona.getEdad());
15        System.out.println("Profesion: "+miPersona.getProfesion());
16        System.out.println("Telefono: "+miPersona.getTelefono());
17
18        System.out.println("*****");
19    }
20 }
```

Al presionar el botón “History” podremos revisar todos los commit realizados y el usuario que los envió.

History for EjemploProyecto / EjemploProyecto / src / Aplicacion.java

Commits on Feb 27, 2018

Se agregan los asteriscos desde el otro usuario - cdhenao2
cdhenao2 committed 12 minutes ago

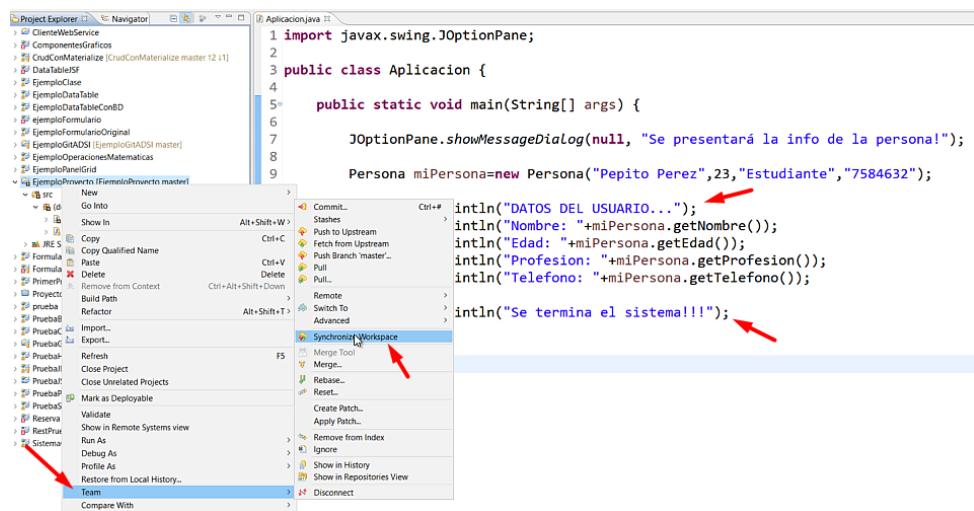
Se agrega un mensaje de información
chenaoh committed 26 minutes ago

Se crea el Proyecto - chenaoh
chenaoh committed an hour ago

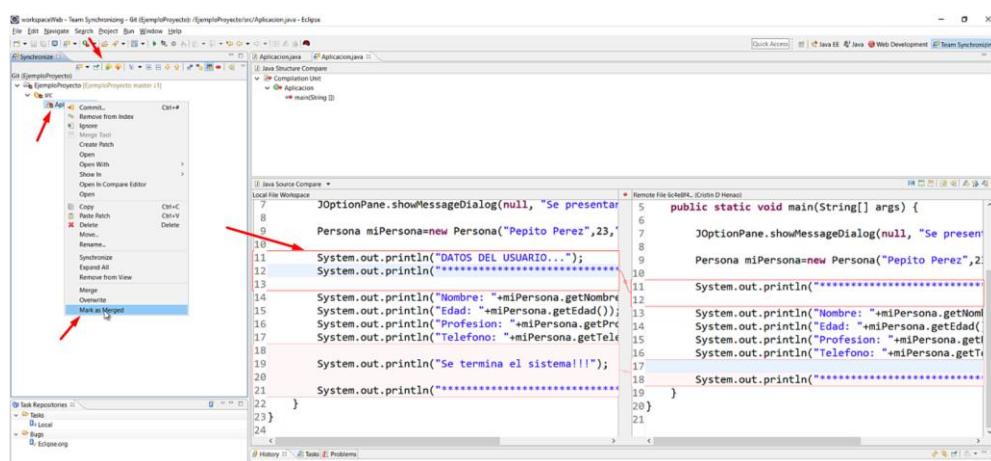
RESOLVER CONFLICTOS

Los conflictos son un proceso normal cuando trabajamos con repositorios, estos representan inconsistencias en el código o la información a procesar, se da generalmente cuando 2 o más usuarios modifican el mismo archivo en la misma línea, así cuando se intenta subir un nuevo cambio al repositorio el sistema lo identifica como un posible error o conflicto que se debe resolver de forma manual.

Para verificar este proceso vamos a realizar una modificación en el ambiente local del usuario 1, (este usuario aún no ha actualizado los cambios que subió el usuario 2 en la sección anterior.) lo agregamos al index y posteriormente damos clic en sincronizar para verificar las diferencias con respecto al repositorio remoto.

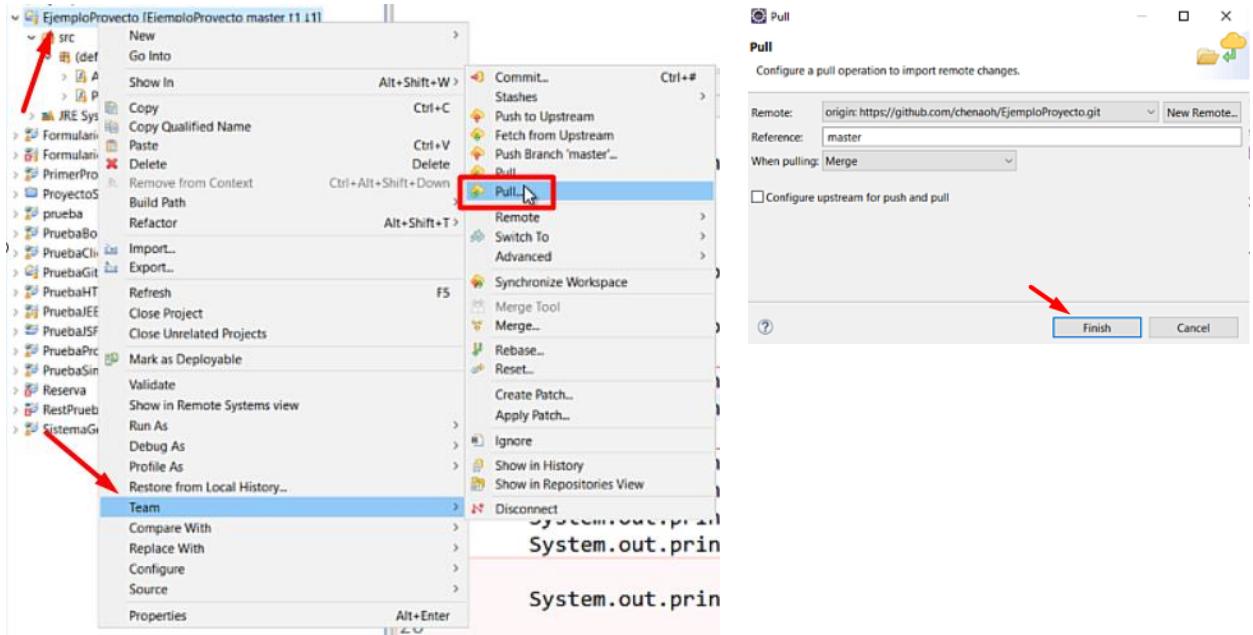


Al hacerlo nos damos cuenta que se tienen conflictos, esto se da porque las líneas 11 y 12 del ambiente local tienen diferencias con las líneas 11 y 12 del ambiente remoto.

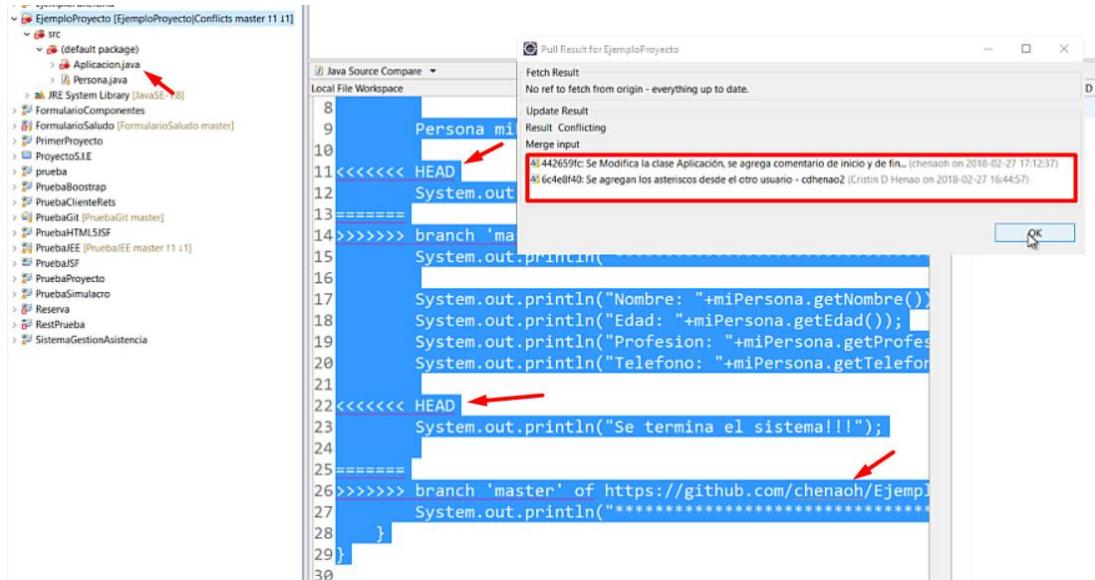


Lo que intentaremos hacer es combinar las diferencias en nuestro ambiente local y posteriormente subir el código actualizado, para esto damos clic en "Mark as Merged".

Si no se resuelve el conflicto regresamos a la perspectiva JEE y damos clic derecho al proyecto/Team/Pull... y confirmamos la acción.



Al hacerlo podemos ver que se actualiza nuestro código con la referencia a los cambios del HEAD (repositorio local) y la rama principal del repositorio remoto, así como los commit realizados.



Ahora debemos corregir los errores de forma manual, eliminando las líneas incorrectas y vinculando los cambios que queremos actualizar, realizamos el commit y listo.

The screenshot shows a Git commit dialog box. On the left, there is a code editor window displaying Java code. On the right, the commit dialog has several sections:

- Unstaged Changes (0)**: Shows two staged commits:
 - 4d0376ad: Se agregan los cambios para el bloque de inicio y final... (chenaoh on 2018-02-27 17:23:22)
 - 442b59fc: Se Modifica la clase Aplicacion, se agrega comentario de inicio y fin... (chenaoh on 2018-02-27 17:23:22)
- Commit Message**: A large text area for entering the commit message.
- Message Details**: Includes fields for **Repository** (<https://github.com/chenaoh/EjemploProyecto>) and **Author** (chenaoh <cdhenao@gmail.com>).
- OK** button at the bottom right.

Two red arrows point to the commit message area: one pointing to the commit log entries and another pointing to the repository URL field.

```
1 import javax.swing.JOptionPane;
2
3 public class Aplicacion {
4
5     Push Results: EjemploProyecto - origin
6     Pushed to EjemploProyecto - origin
7
8     master -- master [4d0376a..6c4e8f4] (2)
9     > 4d0376ad: Se agregan los cambios para el bloque de inicio y final... (chenaoh on 2018-02-27 17:23:22)
10    > 442b59fc: Se Modifica la clase Aplicacion, se agrega comentario de inicio y fin... (chenaoh on 2018-02-27 17:23:22)
11
12
13 Message Details
14 Repository
15 https://github.com/chenaoh/EjemploProyecto
16 ...
17
18 System.out.println("Telefo|
```

Como vemos este proceso puede ser un poco tedioso, por eso como buena práctica se debería actualizar cada vez el repositorio antes de empezar a realizar nuestros cambios con el fin de evitar tener muchas actualizaciones pendientes y con esto gran diferencia entre nuestro entorno local y remoto.

CONSIDERACIONES FINALES.

Vale la pena aclarar que se deben realizar diferentes prácticas con el uso de repositorios con el fin de adquirir mayor dominio y control de nuestro código o documentación al momento de trabajar en un equipo de desarrollo.

Por otro lado pudimos ver que se puede gestionar diferente material tanto de documentación como de código fuente y proyectos en general, de esa manera si estamos en un equipo de desarrollo cada persona puede tener una versión local del proyecto y realizar los cambios correspondientes, en este caso usamos Eclipse como cliente para gestionar el repositorio de código pero el proceso puede hacerse en cualquier otra herramienta que soporte esta tecnología o realizarse directamente desde el cliente de escritorio que trabajamos al principio de la guía, la vinculación de usuarios se hace de la misma forma y cada uno administrará el control de versiones mediante la herramienta directamente.

Como se mencionó ni GIT ni GITHUB son las únicas tecnologías o herramientas para gestionar repositorios y el control de versiones, existen muchas más como SubVersion, Mercurial entre otros con sus respectivos clientes, para GIT podemos encontrar GitLab, Bitbucket entre otros.

Por último es importante tener en cuenta que todas las prácticas fueron realizadas directamente a la rama principal (Master) se recomienda explorar un poco más sobre la creación de “**branches**” para garantizar la integridad de nuestros proyectos, pues estos independizan nuestras versiones pudiendo tener una copia en el branch o rama de desarrollo, otra en el branch de pruebas y finalmente dejar el branch master como rama de producción donde serán reflejados todos los cambios finales que ya hayan sido verificados.