Introducción

Un sistema de versionamiento es aquel que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo es posible recuperar estados anteriores de los mismos. Esto es, es posible obtener versiones específicas más adelante. Poder manejar un sistema de versionamiento aplicado a los archivos fuente de los proyectos que se realicen permitirán realizar un control de las versiones de esos proyectos que resulta imprescindible al momento de trabajar de manera colaborativa y cooperativa.

Sistema de control de versiones locales

Imagine la siguiente situación: ud crea un proyecto y a medida que avanza en el mismo va realizando backups de los pequeños avances. Estas copias de respaldo las organiza en un directorio, usando como estrategia probablemente el día y horario en que fue grabada la copia. Además, probablemente necesite un archivo que permita detallar que avance contiene cada copia de respaldo.
Como puede suponer esta estrategia (muy común, lamentablemente) está expuesta o es susceptible a los efectos de los errores humanos que pueda cometer el desarrollador. Es muy probable que este desarrollador olvide en qué directorio almacenó una copia de respaldo específica, así como es muy probable que no lleve un registro de las novedades de cada copia de respaldo. También es probable que el desarrollador sin desearlo sobreescriba una versión anterior o elimine una copia por error.
Para evitar esto, se desarrollaron los denominados Sistemas de Control de Versionamiento (VSC). La primera generación de estos VSC se denominó de “tipo Local”; por lo cual consistían simplemente en una base de datos en la cual es posible llevar el registro de los cambios realizados. Esto se puede observar en la figura 1.

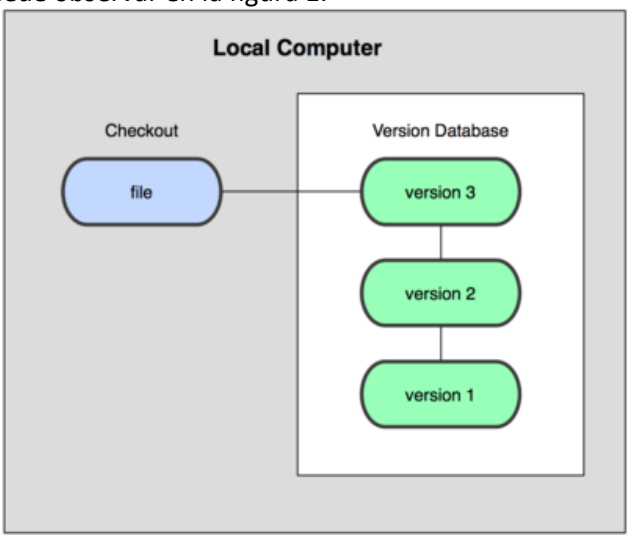


Figura 1. Esquema de funcionamiento de un VSC Local

Sistema de control de versiones centralizados

Si nos centramos exclusivamente en el desarrollo de proyectos software, nos toparemos con la necesidad natural de poder desarrollar esos proyectos de manera colaborativa, y de alguna manera usar un VSC. Por este motivo, se desarrollaron los sistemas de control de versiones centralizados (Centralized Version Control Systems o CVCSs en inglés). En estos sistemas se mantiene un único servidor que contiene todos los archivos versionados del proyecto, desde el cual los clientes descargan los archivos, tal como se puede observar en la figura 2. Durante muchos años, éste ha sido el estándar para el control de versiones.

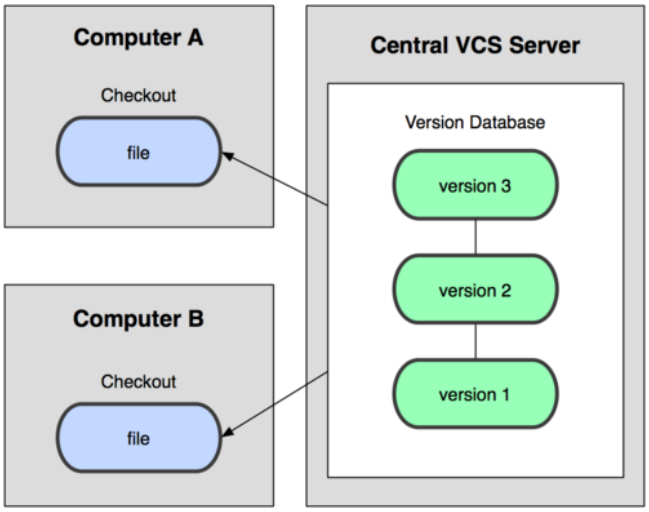


Figura 2. Esquema de un CVSC

Esta configuración ofrece muchas ventajas, especialmente frente a los VCSs locales. Una de ellas, y quizás la más importante; es que es posible establecer estrategias para la distribución de las tareas y la visualización del rendimiento de cada cliente de acuerdo con los aportes realizados en el proyecto. Además, los administradores tienen control detallado de qué puede hacer cada uno; y es mucho más fácil administrar un CVCS que tener que lidiar con bases de datos locales en cada cliente.
Sin embargo, esta configuración también tiene serias desventajas. La más obvia es el punto único de fallo que representa el servidor centralizado. Si ese servidor se cae nadie puede colaborar o guardar cambios versionados de los aportes sobre los que está trabajando. Si el disco duro en el que se encuentra la base de datos central se corrompe, y no se han llevado copias de seguridad adecuadamente, se perderá absolutamente todo, con excepción, claro está, de las versiones que han quedado en las máquinas locales. Dado que los VCSs locales sufren de este mismo problema, resulta claro que es una desventaja asociada a la amenaza que significa mantener toda la historia del proyecto en un único lugar.

Sistema de control de versiones distribuidos

Los sistemas de control de versiones distribuidos (o DVCS) fueron concebidos con el objetivo de subsanar las desventajas de los sistemas anteriores. Para lograrlo, estos sistemas determinan que, al momento de que un cliente descargue la última versión del proyecto, esta replicará completamente todo el repositorio. El término repositorio hace referencia al espacio usado para mantener un conjunto de archivos que se puede modificar, recuperar o eliminar. De esta manera cada cliente se convierte además en un servidor. Así, si un servidor se corrompe, es posible usar cualesquiera de los repositorios de los clientes como copia para restaurar en el servidor. Así, cada vez que se descarga una versión, en realidad se hace una copia de seguridad completa de todos los datos. Esto lo podemos observar en la figura 3. Este esquema de funcionamiento permite establecer un conjunto de estrategias de trabajo colaborativo que resultan muy interesantes, aunque requiere un conocimiento mayor de la forma en la cual se debe definir la forma de trabajo y la resolución de posibles conflictos que se puedan generar. Es decir, se puede establecer varios tipos de flujos de trabajo que no son posibles en sistemas centralizados.

Que es GIT

Entre el año 1991 y el año 2002, la mayor parte del mantenimiento del núcleo de Linux se realizó mediante parches y archivos. Como era de esperarse, el intenso trabajo que la comunidad realizó sobre este proyecto requirió el uso de un DVCS. Así, en 2002, el proyecto del núcleo de

Linux se alojó en un DVCS propietario denominado BitKeeper. En 2005, la relación entre la comunidad que desarrollaba el núcleo de Linux y la compañía que desarrollaba BitKeeper llegó a su fin. Así, Linus Torvalds, el creador de Linux desarrolló su propia herramienta de versionamiento basada en algunas de las experiencias adquiridas durante el uso de BitKeeper.

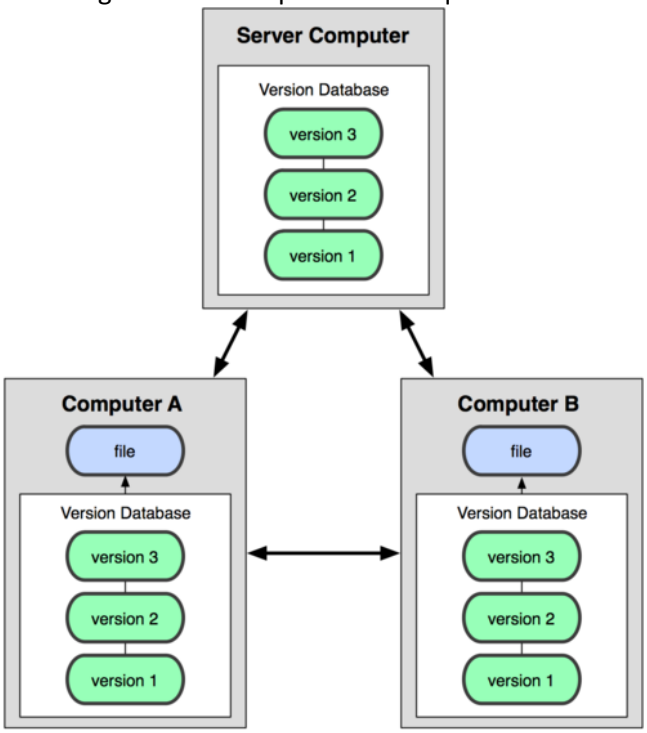


Figura 3. Esquema de un DVCS

Algunos de los objetivos del nuevo sistema fueron los siguientes:

- Velocidad
- Diseño sencillo
- Fuerte apoyo al desarrollo no lineal (miles de ramas paralelas)
- Completamente distribuido
- Capaz de manejar grandes proyectos, tales como el núcleo de Linux de manera eficiente (velocidad y tamaño de los datos)

Este sistema de versionamiento se denominó Git, y desde su nacimiento en 2005, ha evolucionado y madurado con las premisas anteriores. Por este motivo, la curva de aprendizaje resulta muy liviana, es fácil de usar, es tremendamente rápido, muy eficiente con grandes proyectos, y tiene un increíble sistema de ramas (branching) para desarrollo no lineal.

Los fundamentos de GIT

A pesar de que las bases del funcionamiento de los VCS son similares, hay que empezar indicando que Git propone algunas diferencias a tener en cuenta. Git almacena y modela la información de forma muy diferente a otros sistemas tales como Subversion, a pesar de que su interfaz sea bastante similar; por lo tanto, comprender esas diferencias ayudará a evitar confusiones. Mientras que la mayoría de los demás sistemas almacenan la información como una lista de cambios en los archivos, esto es, modelan la información que almacenan como un conjunto de archivos y las modificaciones hechas sobre cada uno de ellos a lo largo del tiempo (figura 4); Git modela sus datos más como un conjunto de instantáneas de un mini sistema de archivos. Cada vez que se confirma un cambio, o se guarda el estado actual de un proyecto en Git, básicamente se genera una foto del aspecto actual de todos los archivos del proyecto en

ese momento, y el sistema guarda una referencia a esa instantánea. Para ser eficiente, si los archivos no se han modificado, Git no almacena el archivo de nuevo, sino que genera un enlace al archivo anterior idéntico que ya tiene almacenado. Este esquema de almacenamiento es modelado en la Figura 5.

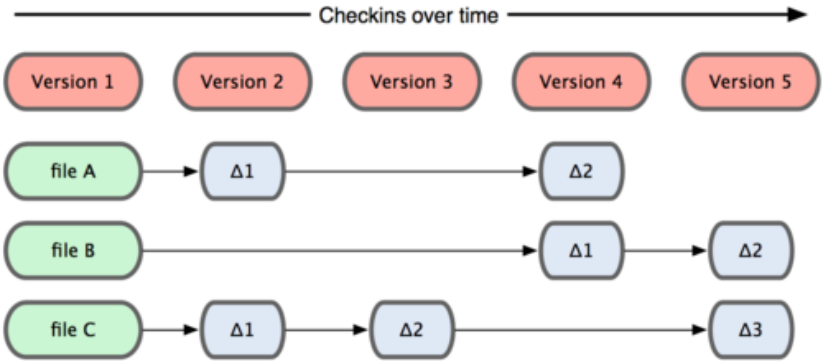


Figura 4. Modelo de almacenamiento de las versiones de otros DVCS

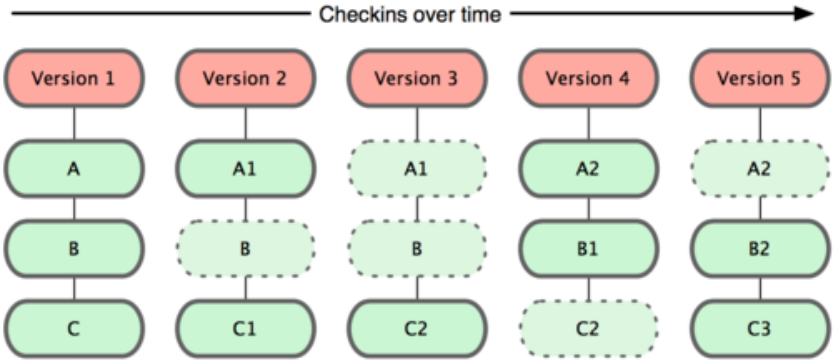


Figura 5. Modelo de almacenamiento de las versiones en Git

Esta gran diferencia significa que Git reconsidera casi todos los aspectos relacionados con el control de versiones que por tradición muchos de los demás sistemas copiaron desde las primeras generaciones de VSC. Este modelo convierte a Git en una especie de mini sistema de archivos con algunas herramientas tremendamente potentes construidas específicamente para poder adicionalmente realizar las tareas de un VCS. Los beneficios más notorios de este esquema de funcionamiento se reflejan en la manera de gestionar las ramas (branching), aspecto que se considerará más adelante.

Consecuencias de este modelo

1. Casi cualquier operación es local: La mayoría de las operaciones en Git sólo necesitan archivos y recursos locales para operar (por lo general no se necesita información de ningún otro ordenador de tu red). De esta manera la mayoría de las operaciones tienen no generar sobrecarga y por lo tanto no hay retardo en uso de la red. Como toda la historia del proyecto se encuentra en el proyecto descargado al repositorio local, la mayoría de las operaciones parecen ejecutarse prácticamente de manera inmediata. Por ejemplo, para navegar por la historia del proyecto, Git no necesita acceder al servidor para obtener la historia y mostrarla, simplemente la lee directamente desde la base de datos local. Esto significa que observarás toda la historia del proyecto casi al instante. Si quieres ver los cambios introducidos entre la versión actual de un archivo y ese archivo hace un mes, Git puede buscar ese archivo y hacer un cálculo de diferencias de manera local, en lugar de tener que pedirle a un servidor remoto que lo haga, u

obtener una versión antigua del archivo del servidor remoto y hacerlo de manera local. En definitiva, podrá realizar los cambios de manera totalmente desconectada del servidor, llegando únicamente a conectarse al mismo únicamente luego de confirmar los cambios, o cuando desee traer a su repositorio local los cambios realizados por otros clientes.

2. Git establece fuertes mecanismos de integridad: Todo en Git es verificado mediante una suma de comprobación antes de ser almacenado, y es identificado a partir de ese momento mediante dicha suma (checksum en inglés). Esto significa que es imposible cambiar los contenidos de cualquier archivo o directorio sin que Git lo sepa. Esta funcionalidad está integrada en Git al más bajo nivel y es parte integral de su filosofía. No es posible perder información durante su transmisión o sufrir corrupción de archivos sin que Git sea capaz de detectarlo.
3. Git generalmente sólo añade información: Cuando realizas acciones en Git, casi todas ellas sólo añaden información a la base de datos de Git. Es muy difícil conseguir que el sistema haga algo que no se pueda deshacer, o que de algún modo borre información. Como en cualquier VCS, puedes perder o estropear cambios que no has confirmado todavía; pero después de confirmar una instantánea en Git, es muy difícil de perder, especialmente si envías (push) tu base de datos a otro repositorio con regularidad. Podemos experimentar sin peligro de fastidiar gravemente las cosas.

Los tres estados

Esto es lo más importante a recordar acerca de Git con el objetivo de que el proceso de aprendizaje prosiga sin problemas. Git posee tres estados principales en los que se pueden encontrar tus archivos: confirmado (committed), modificado (modified), y preparado (staged). Confirmado significa que los cambios se han confirmado y los datos están almacenados de manera segura en tu base de datos local. Modificado significa que has modificado el archivo, pero todavía no lo has confirmado a tu base de datos. Preparado significa que has marcado un archivo modificado en su versión actual para que sea considera en la próxima confirmación.

El funcionamiento de estos estados y sus efectos están indicados en la figura 6:

1. El directorio de Git (git directory o repositorio git) es aquel donde Git almacena los metadatos y la base de datos de objetos de los proyectos. Es la parte más importante de Git, y es lo que se copia cuando se clona un repositorio desde otro ordenador.
2. El directorio de trabajo (working directory) es una copia de una versión del proyecto. Estos archivos se obtienen de la base de datos comprimida ubicada dentro del directorio de Git, y se colocan en disco para poder usarlo o modificarlo.
3. El área de preparación (staging área) es un archivo sencillo, generalmente contenido en el directorio de Git para el proyecto, que almacena información acerca de los cambios que se confirmar cuando se realice el próximo commit. A veces se denomina índice, pero se está convirtiendo en estándar referirse a él como el área de preparación.

El flujo de trabajo básico en Git es algo así:

1. El desarrollador modifica una serie de archivos en el directorio de trabajo.
2. El desarrollador prepara los archivos, añadiendo instantáneas de ellos a tu área de preparación
3. Confirmas los cambios, lo que toma los archivos tal y como están en el área de preparación, y almacena esa instantánea de manera permanente en tu directorio de Git.

Si una versión concreta de un archivo está en el directorio de Git, se considera confirmada (committed). Si ha sufrido cambios desde que se obtuvo del repositorio, pero ha sido añadida al

área de preparación, está preparada (staged). Y si ha sufrido cambios desde que se obtuvo del repositorio, pero no se ha preparado, está modificada (modified).

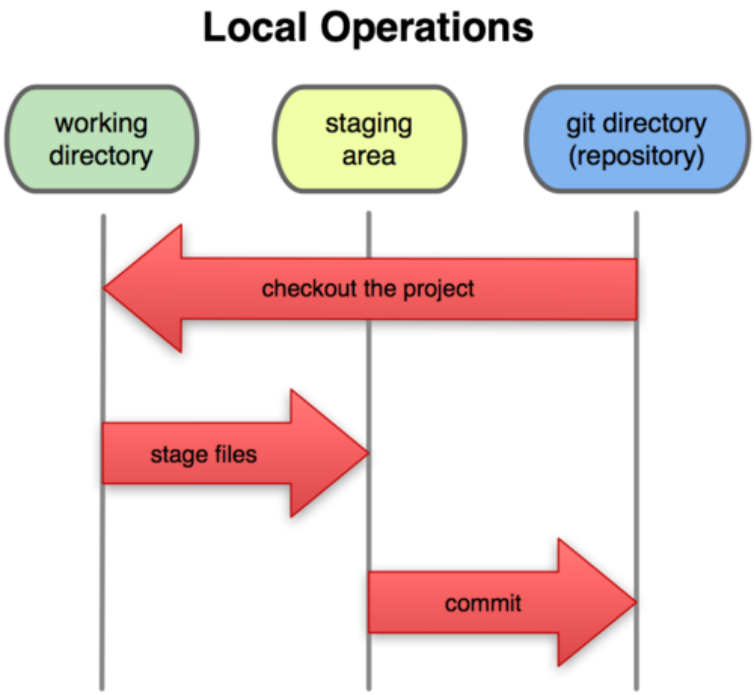


Figura 6. Directorio de trabajo, área de preparación y directorio Git

Que es GitHub

Muchas personas pueden pensar que Git y GitHub son lo mismo, ya que la mayoría de las ocasiones están estrechamente relacionadas, pero a la hora de la verdad, son cosas totalmente distintas.

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. A diferencia del proyecto Git, éste se trata de un servicio comercial, ya que, aunque tiene una parte pública gratuita, cuenta con la desventaja de que todo el código que subamos estará disponible para cualquier persona. Si queremos decidir quién puede tener acceso a nuestro repositorio, entonces sería necesario pasarse a la modalidad de pago.

Por lo demás los proyectos que alojamos utilizan el sistema de versionamiento propuesto por Git.

Entre las principales características que nos ofrece GitHub podemos destacar:

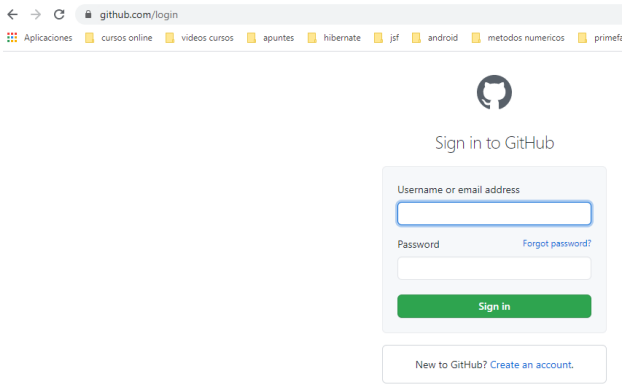
- Una Wiki que opera con Git para el mantenimiento de las distintas versiones de las páginas.
- Sistema de seguimiento de problemas. Se trata de un sistema muy parecido al tradicional ticket, donde cualquier miembro del equipo o persona (si nuestro repositorio es público) puede abrir una consulta o sugerencia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

Relación entre GitHub y STS

El objetivo será crear proyectos en el repositorio de Github. Estos proyectos serán nuestras aplicaciones Spring boot. De esta manera se podrá trabajar de manera cooperativa y colaborativa en los proyectos. STS provee las funcionalidades para realizar los comandos git de manera gráfica.

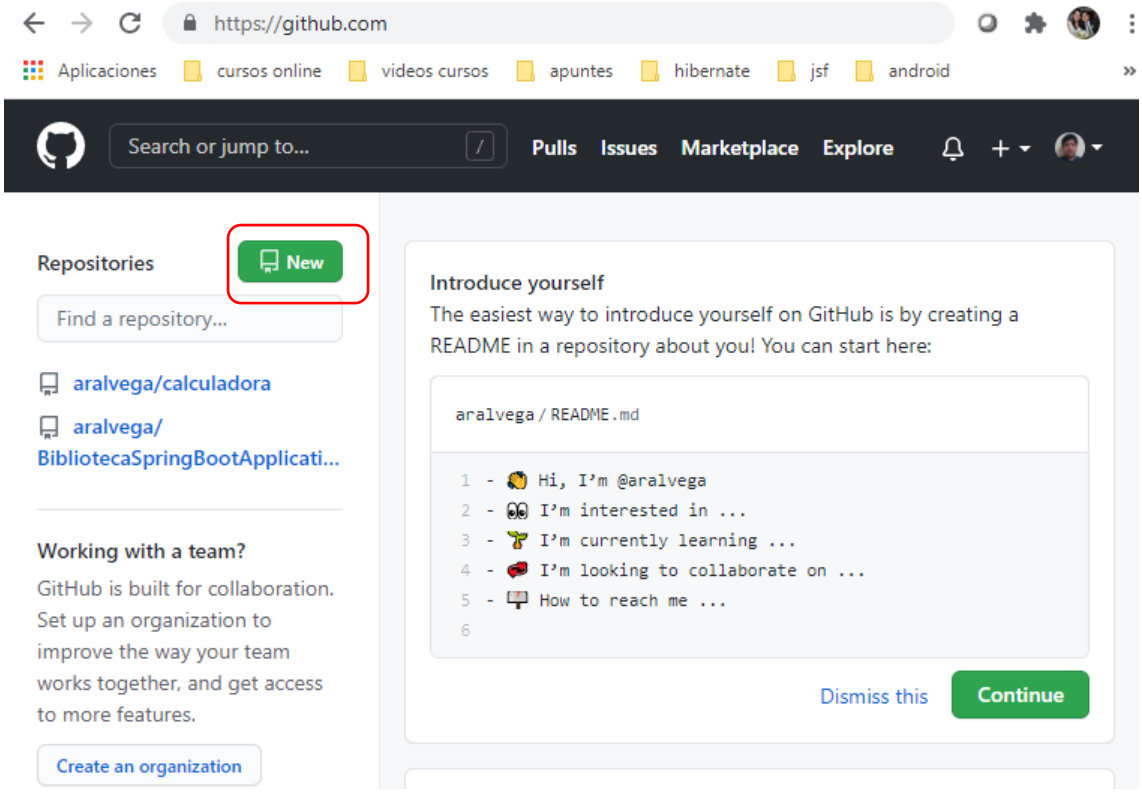
La cuenta en github

Para poder utilizar el repositorio y el VSC de Git que ofrece Github es necesario previamente crear una cuenta. En la página www.github.com/login podrá crear una cuenta. Los pasos para la creación de esta cuenta son similar a la que se utilizan para crear cuentas de correo.



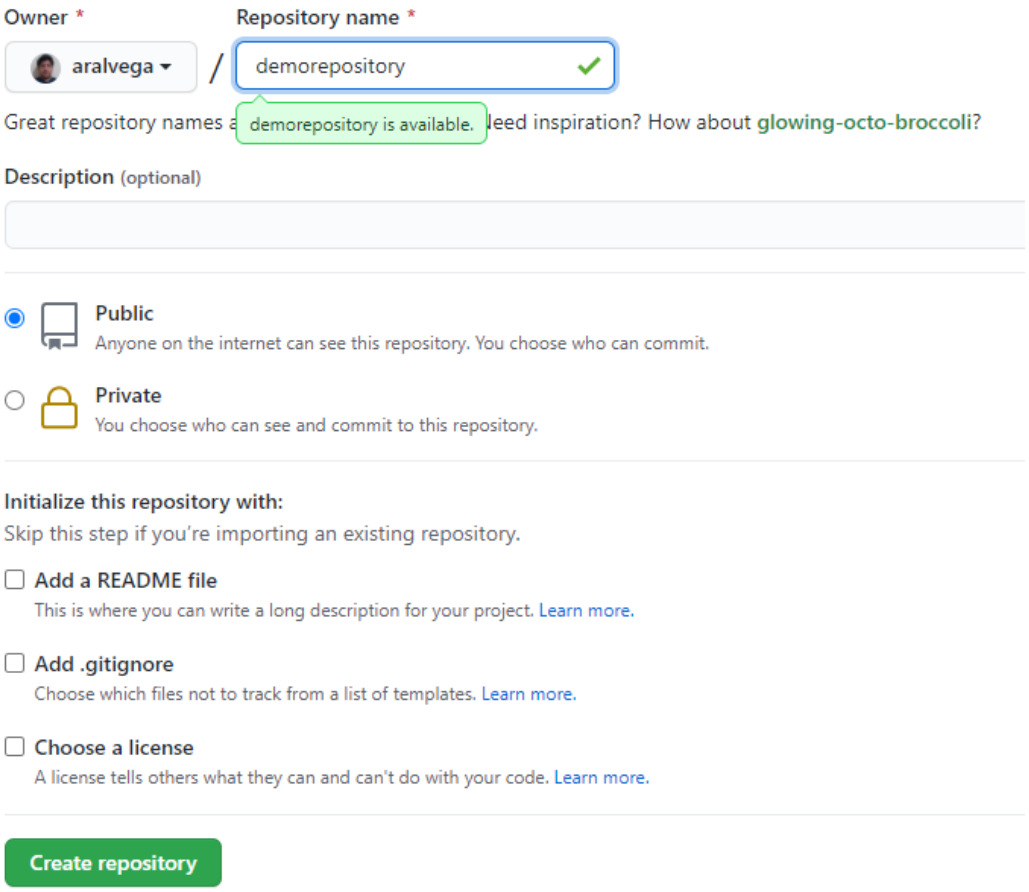
Creación de repositorios en github

Una vez que ingresamos con las credenciales dentro del sitio de github aparecerá una pantalla similar a la siguiente

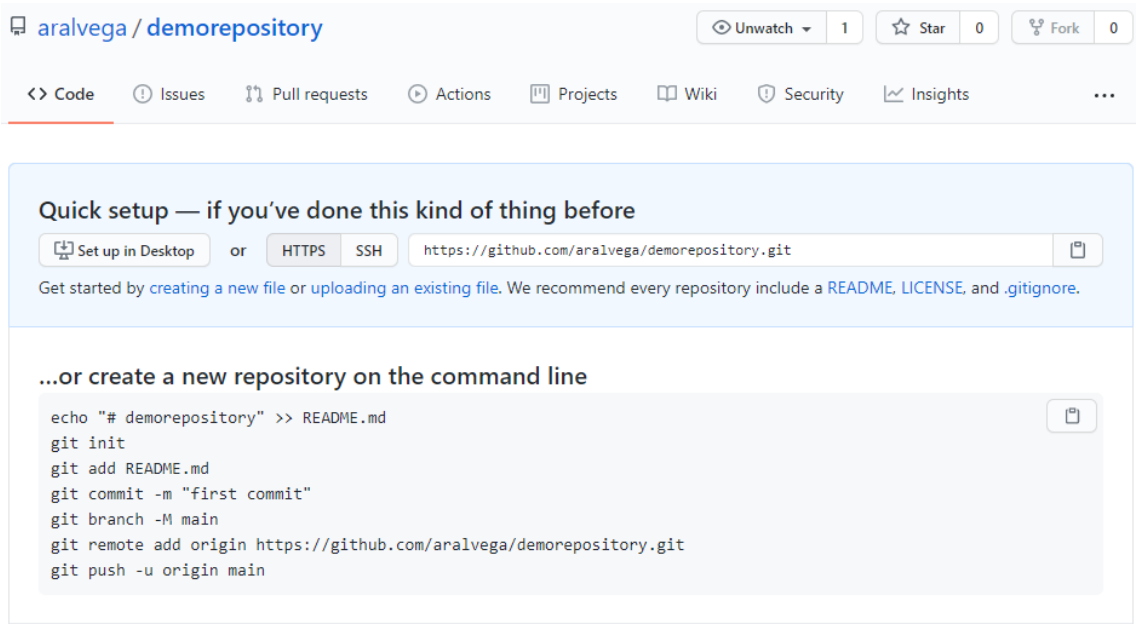


Para crear un repositorio nuevo vamos a seleccionar el botón new.

En la ventana siguiente podremos colocar el nombre del repositorio y una descripción.

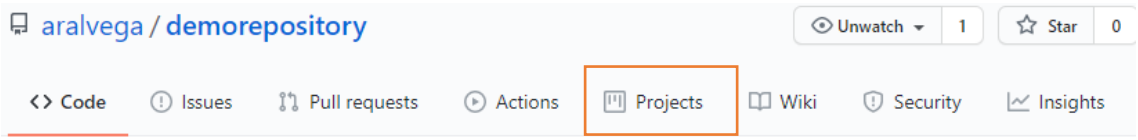


Si presionamos el botón Create repository generaremos un repositorio.

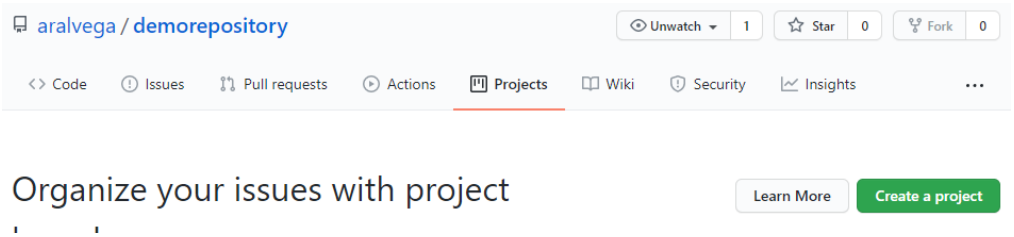


Gestionar proyectos en el repositorio creado en Github

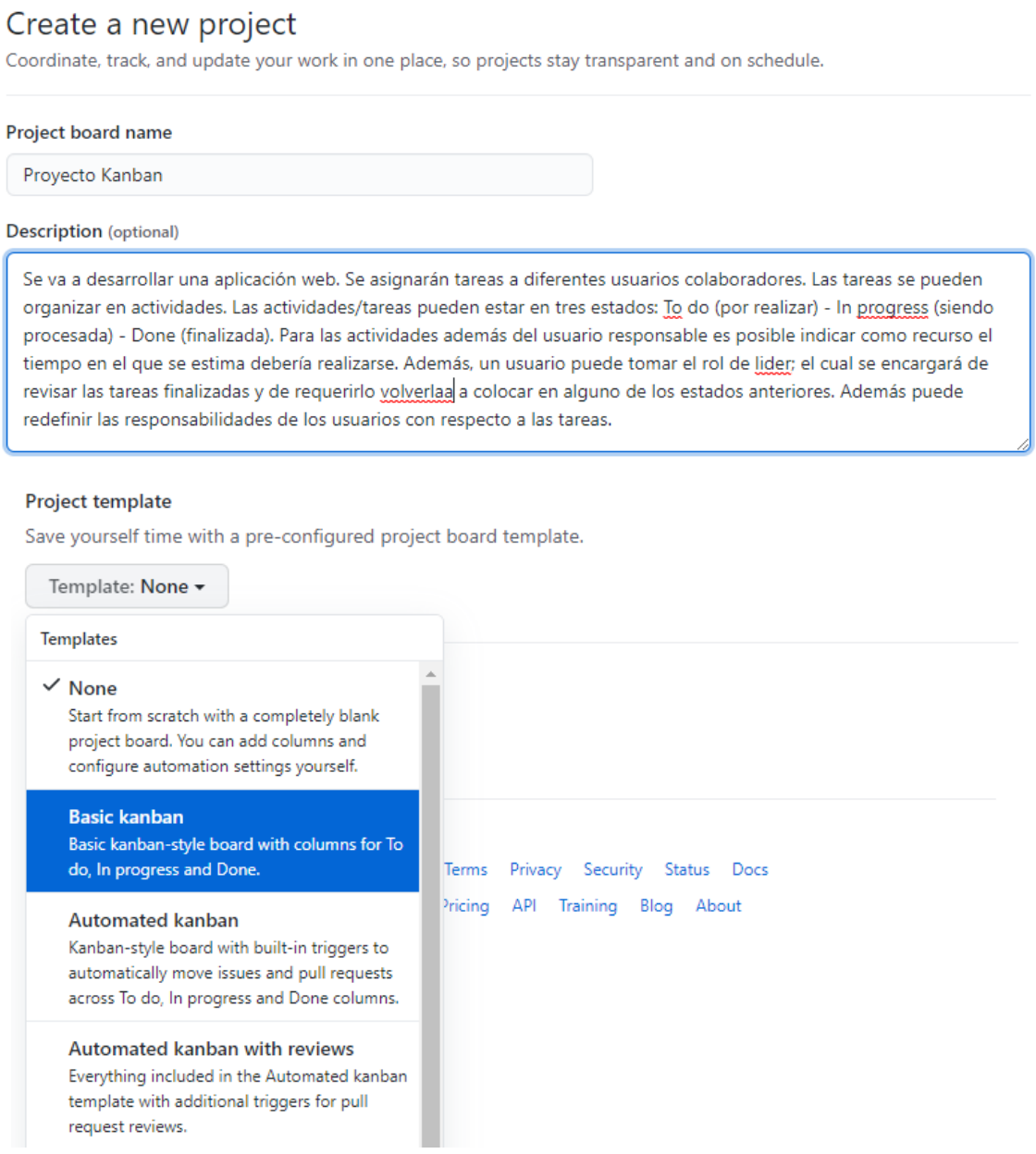
Es posible asignar un gestor de proyectos al repositorio creado. Para ello nos vamos a la solapa projects



Y presionamos el botón Create Project

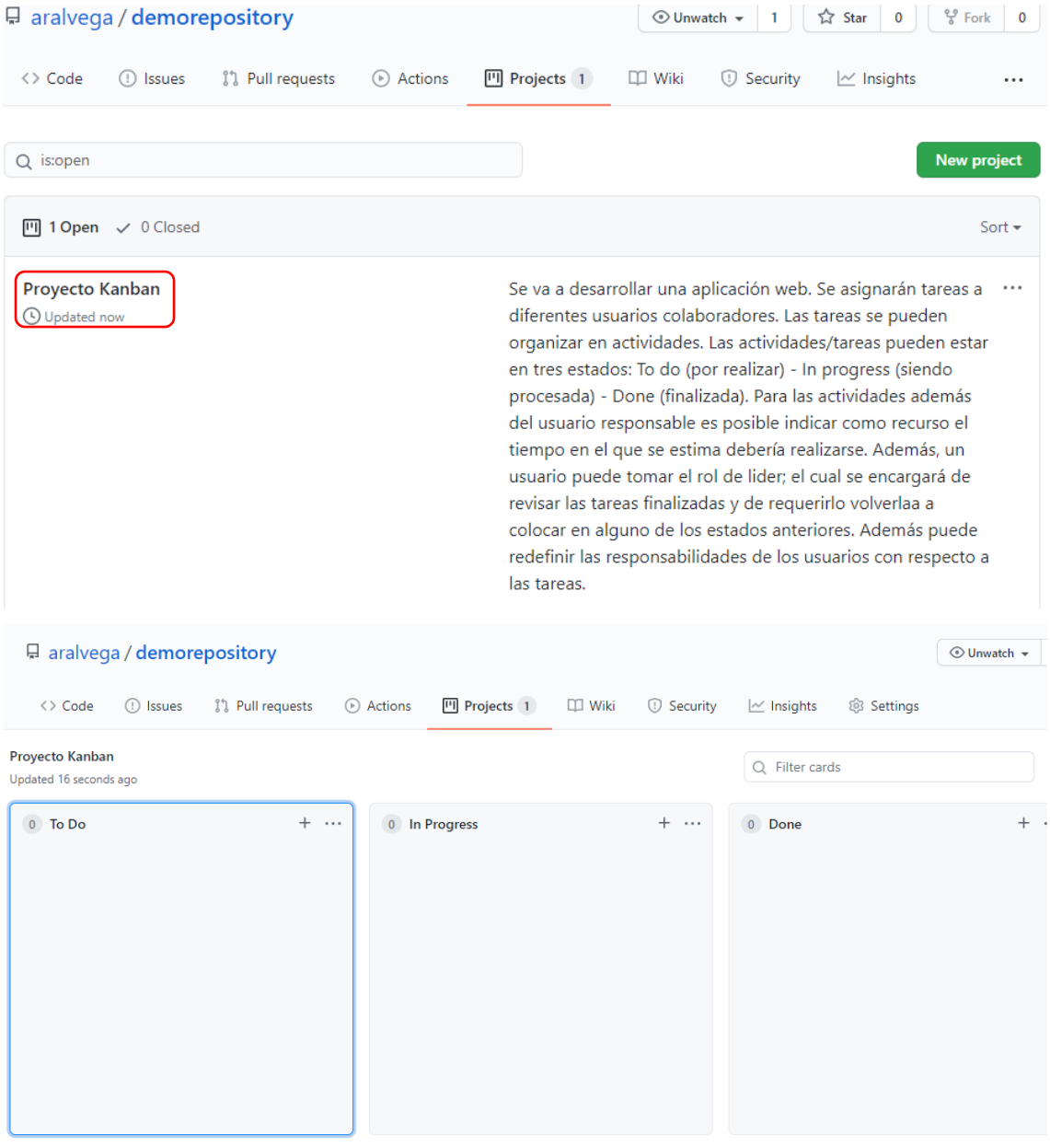


Y en la página siguiente estableceremos lo siguiente



En el cual lo más importante es que usaremos un template de un proyecto Kanban. Este gestor de proyectos se basa en la idea de creación de tareas que están en tres estados. Por hacer, en proceso y finalizados.

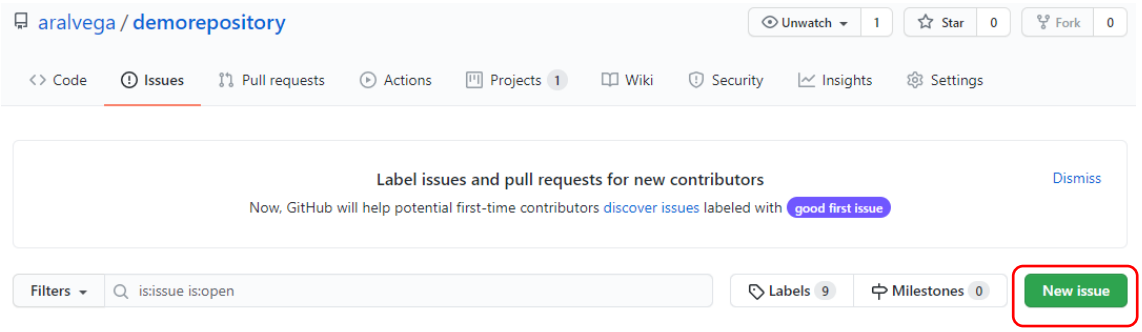
Una vez que seleccionamos todo presionamos el botón Create Project. Ahora cada vez que ingresemos a la sección projects veremos nuestro tablero kanbam



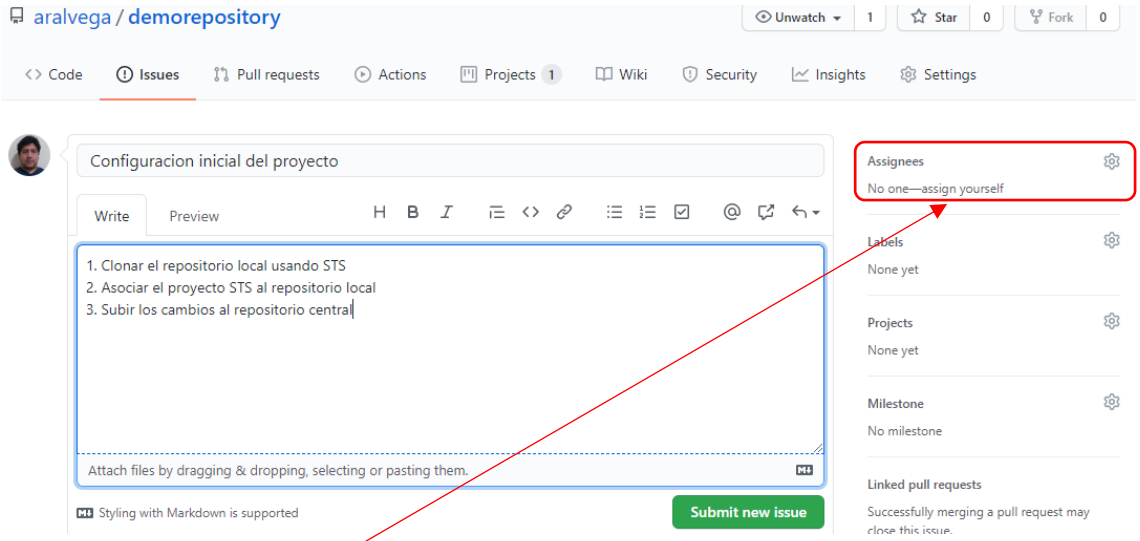
Creación de actividades y asignación de responsabilidades

Hasta este momento el proyecto pertenece a un solo usuario. La primera actividad debería ser clonar el repositorio creado a uno local que será gestionado desde STS. Así en primer lugar debemos crear esta actividad y asignarle un responsable.

Para ello utilizaremos los issues de Gibhub. Debemos ir a esta solapa y crear una nueva tarea, presionando el botón New Issue

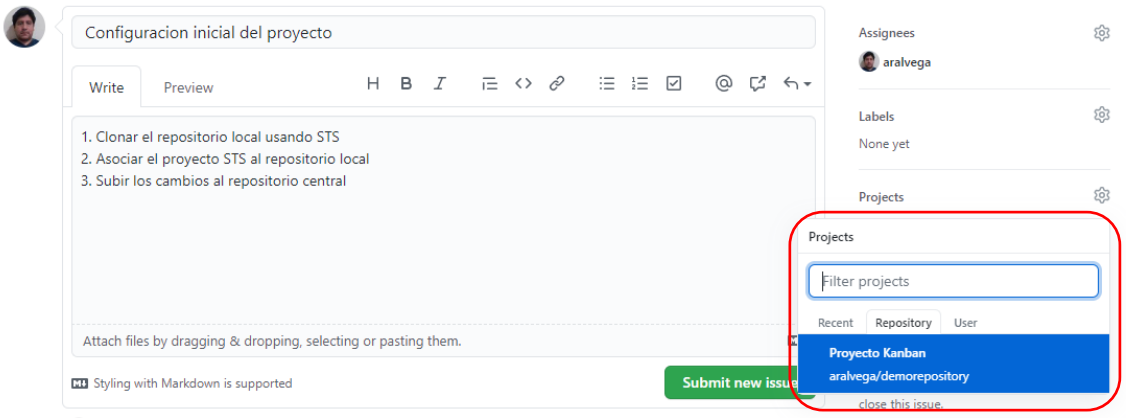


Ahora vamos a definir el issue y vamos a listar las tareas involucradas

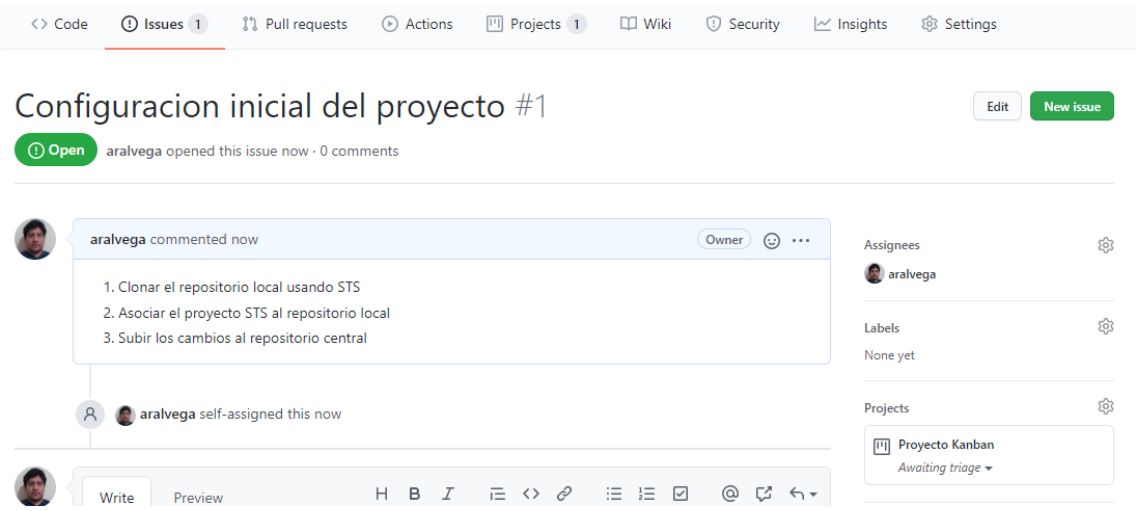


Un aspecto a configurar dentro de este issue es definir quien es el responsable de concretar este issue. Esto lo hacemos en Assigners. Como es de esperarse, se indica que no está asociado a nadie y la única opción posible es autoasignársela, dado que no existen otros colaboradores aún.

Por ahora el otro aspecto fundamental que falta establecer consiste en indicar a que proyecto vamos a asociar este issue

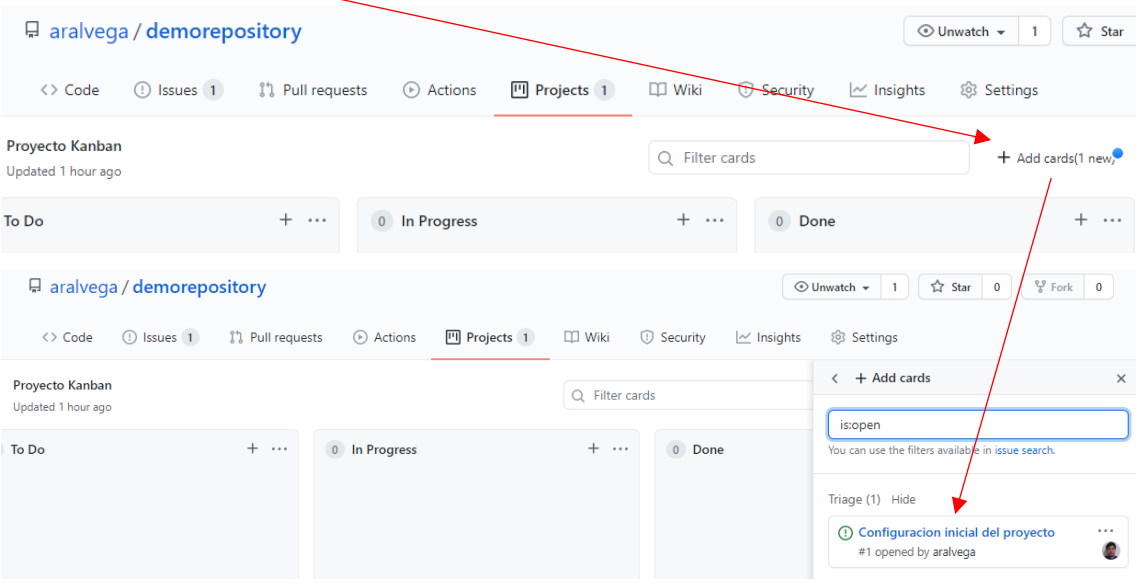


Así, al presionar el botón Submit new Issue quedará creado el mismo. De esta manera quedará establecido que un usuario a creado o abierto un nuevo issue.

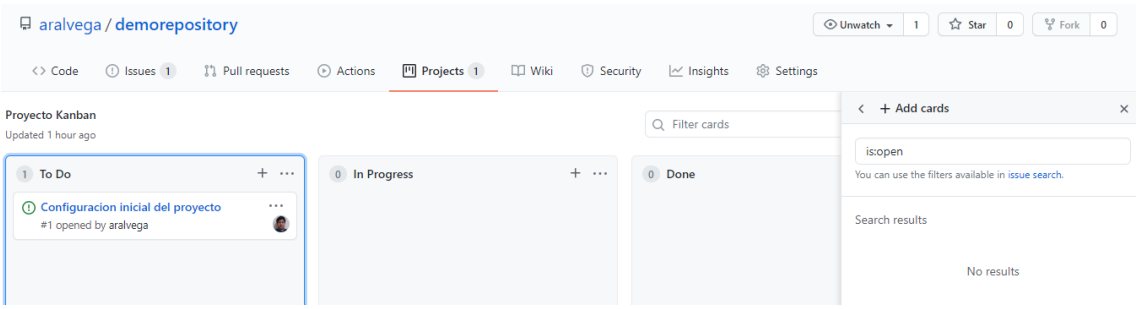


Si volvemos a la sección de proyectos y seleccionamos nuestro proyecto podremos ver que es posible agregar el issue como una tarjeta. En esta metodología en el tablero se colocan tarjetas (cards) que son las que contienen las actividades. En este caso entonces serán nuestras issues.

Así, si presionamos sobre la opción add cards veremos que ya está disponible la tarjeta asociada al issue creado

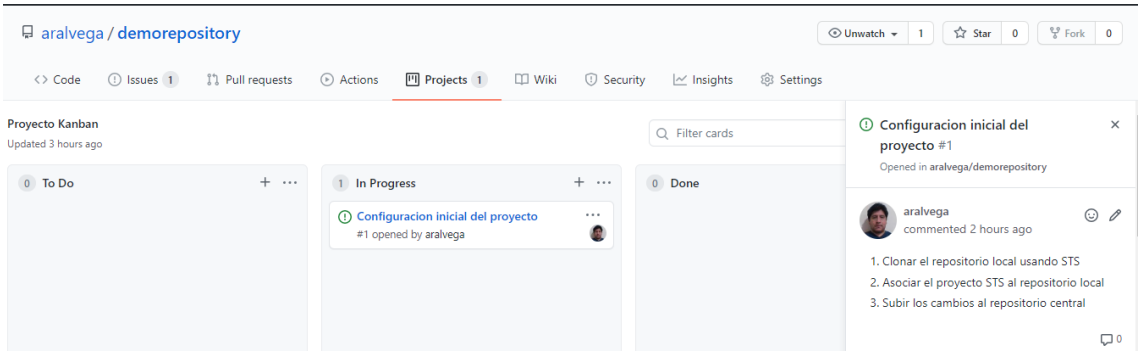


Es posible arrastrar con el mouse la tarjeta, en este caso la llevaremos a To do



Bien, técnicamente hablando aquí se colocarían todas las tarjetas por realizar. Dado que el usuario aralvega va a realizar esta actividad se la pasará a In Progress. Además, observe que el

nombre del issue es un enlace. Al presionarlo se puede observar detalle del mismo a la derecha, en la zona de los cards.



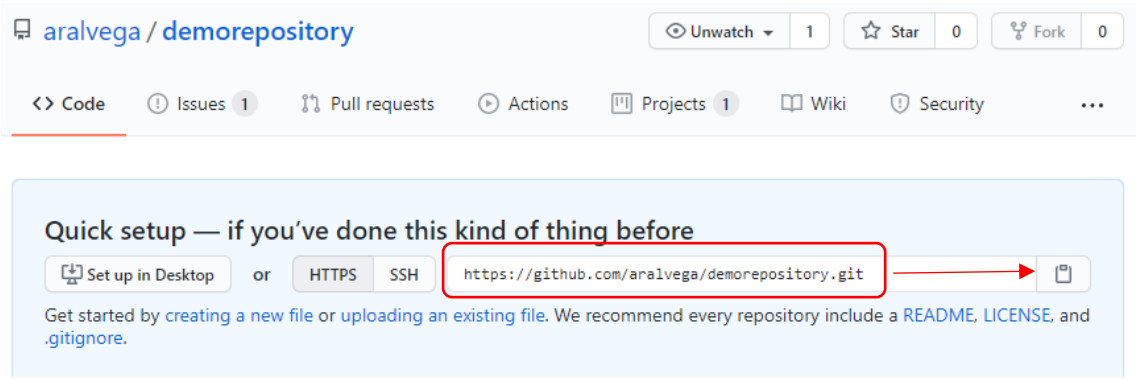
Clonación del repositorio desde STS

El proceso de copiar por primera vez, un repositorio Git a un repositorio local se conoce como clonar.

Git utiliza el comando clone para poder clonar en un repositorio local (el working directory) el repositorio central (el git repository). Técnicamente hablando, se necesita instalar en el ordenador cliente una herramienta que procese los comandos git y realice la gestión de los directorios usados. En este caso nos valdremos del módulo para gestión de Git disponible en STS. Además, se requiere usar las credenciales.

Paso 1: Obtener la URI del repositorio central

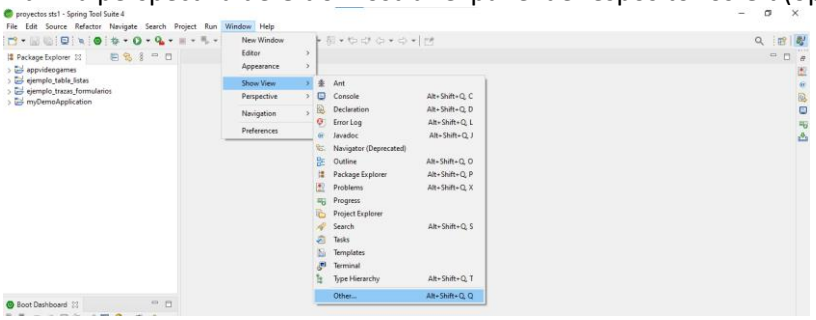
Si vamos a la solapa <code> podremos observar la url que permite identificar donde se halla alojado nuestro repositorio. Además nos da la opción de poder copiar esta información.



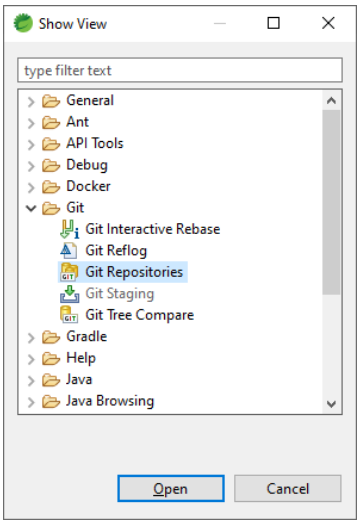
Paso 2: Crear el repositorio local

Si no hemos creado previamente un repositorio estos son los pasos a realizar:

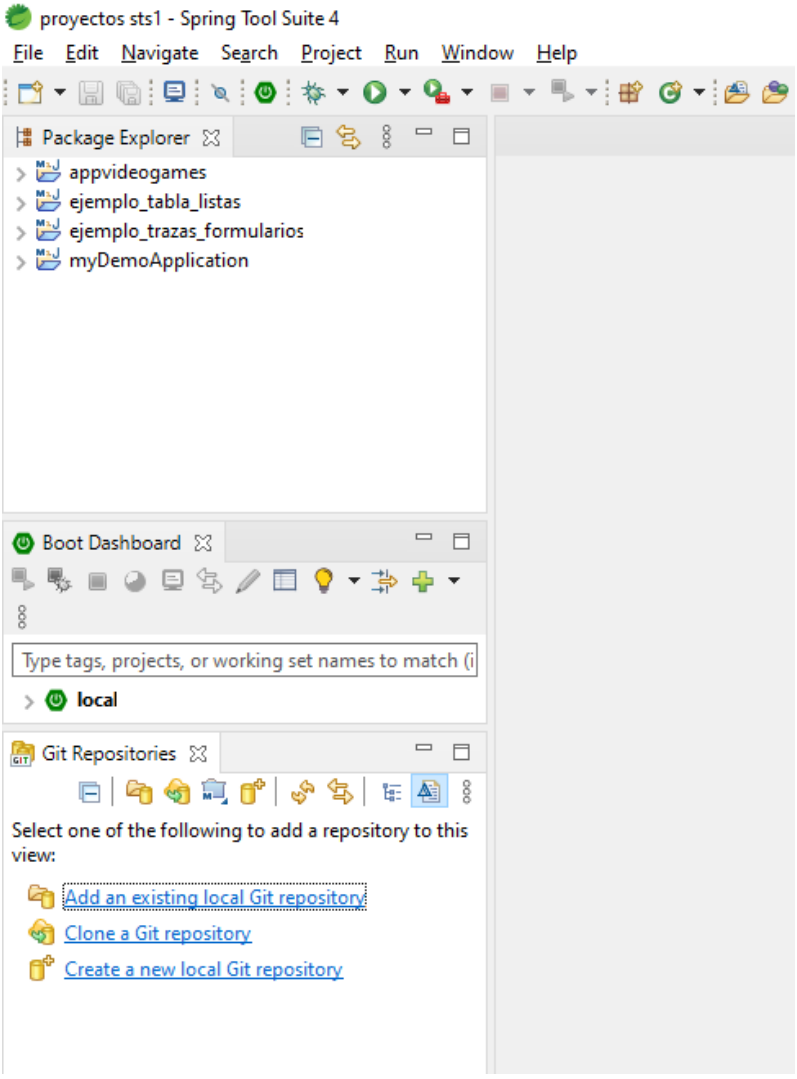
a) Abrir la perspectiva de Git o mostrar el panel de respositorios Git (opción elegida)



Entonces elegimos git repositories

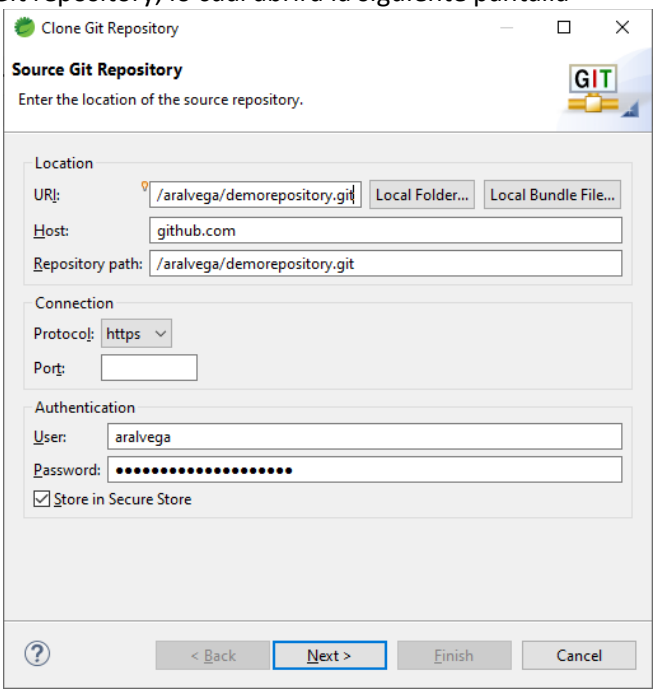


Como podrá observar se abre el panel de repositorios Git

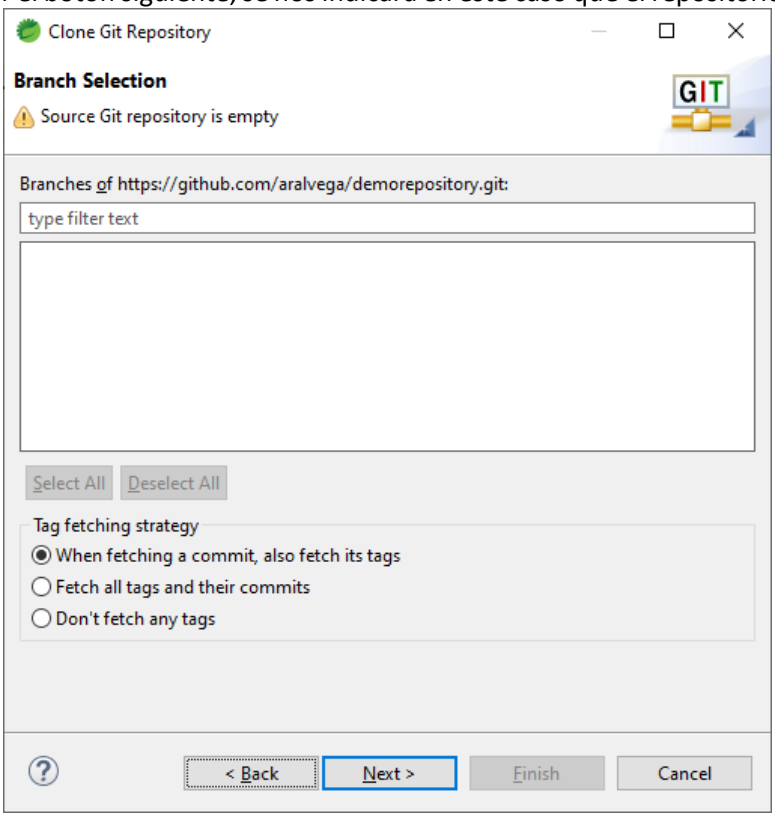


El cual en este momento no posee ningún repositorio local.

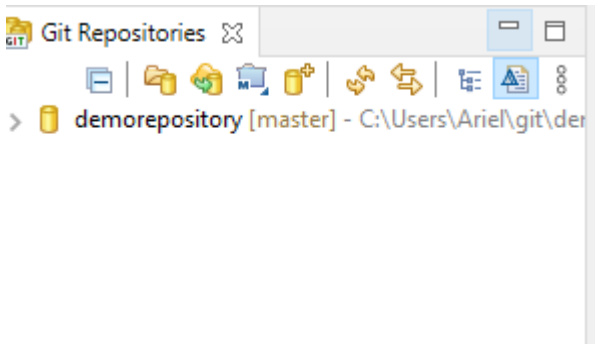
b) Para crear el repositorio local a partir del repositorio remoto, debemos seleccionar la opción Clone a Git repository, lo cual abrirá la siguiente pantalla



Puede notar que en URI hemos colocado la url del repositorio central. El host es la URL del sitio github.com. En repository path indica donde se generará el repositorio local. En la sección de authenticarion colocaremos las credenciales que poseemos de Github.
Al presionar el botón siguiente, se nos indicará en este caso que el repositorio está vació



Presionaremos el botón Next y luego el botón Finish. Esto generará el repositorio de manera local, tal como se observa en la siguiente imagen

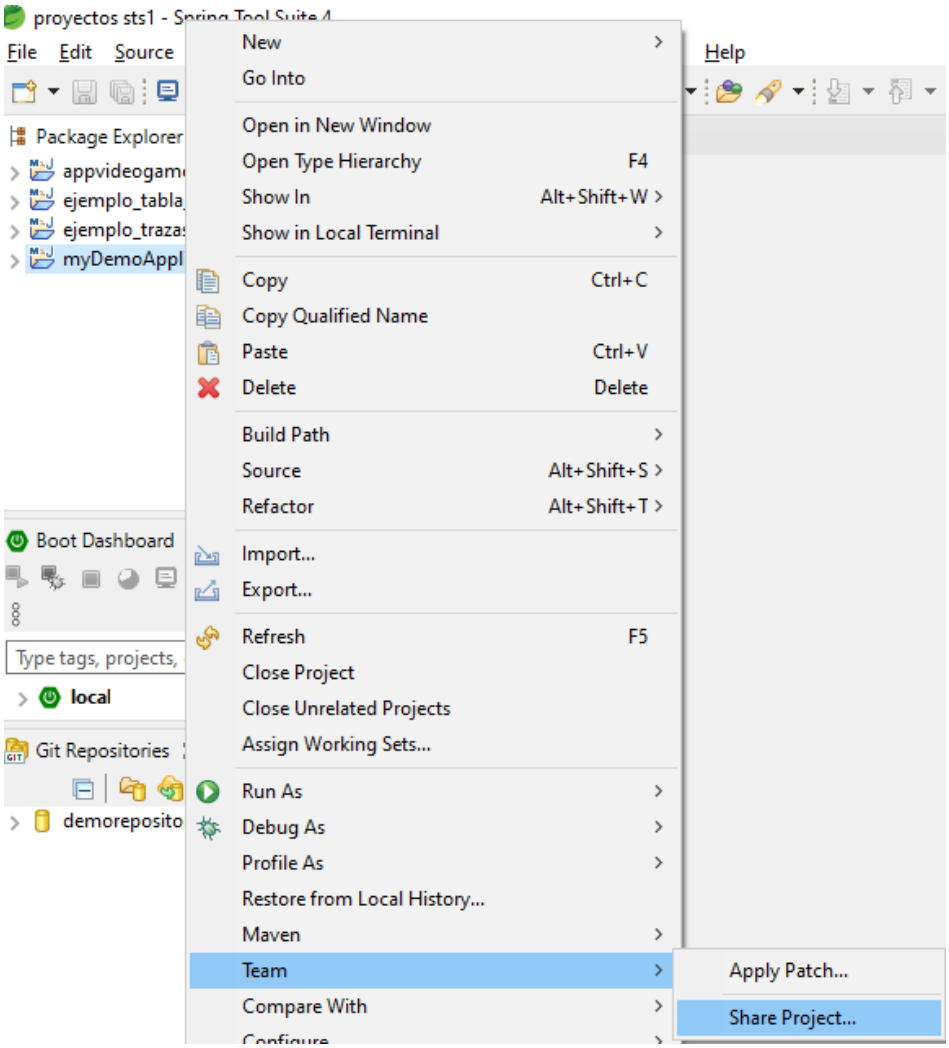


Subir los cambios realizados desde STS: El comando add, pull y commit

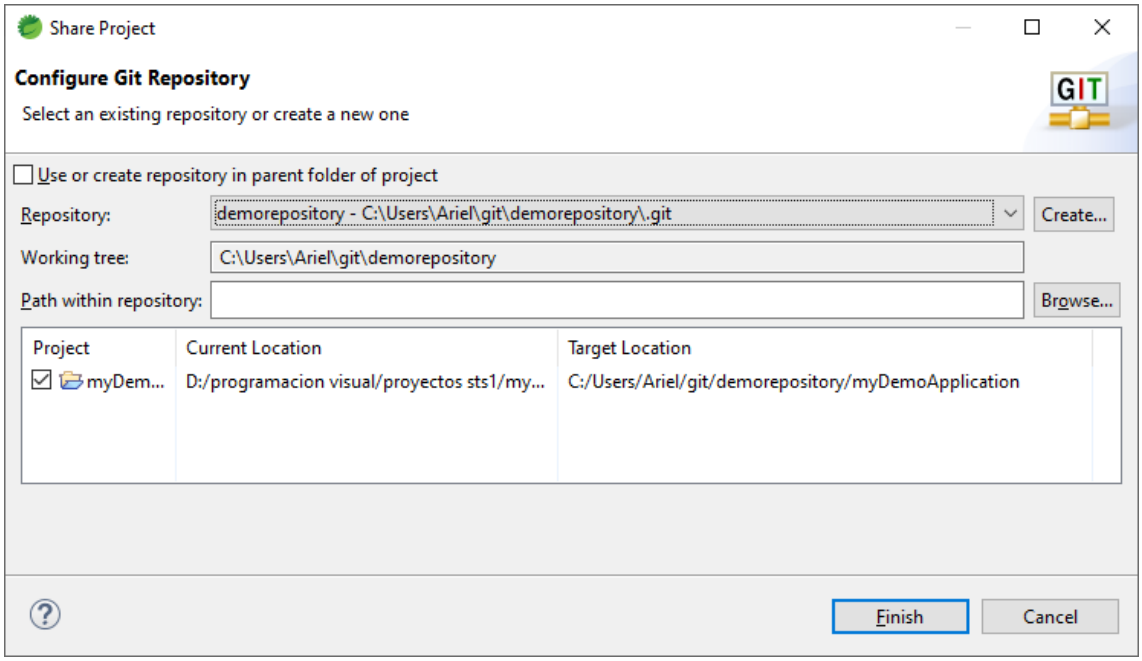
A continuación, incluiremos los pasos para subir los cambios realizados en un proyecto al repositorio. En primer lugar, nuestro repositorio local debe estar asociado a un proyecto. En este caso aún no lo hemos realizado, así que debe realizar el paso 1, caso contrario saltar al paso 2.

Paso 1: Asociar un proyecto al repositorio

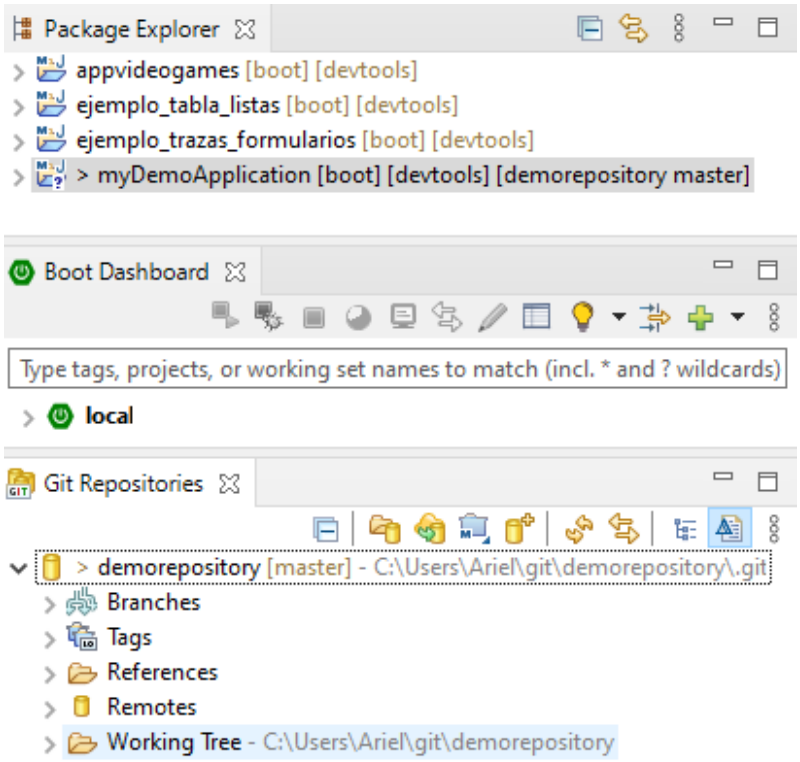
Para lograr esto, sobre el proyecto elegido presionaremos el botón derecho, elegiremos la opción Team -> My share projects



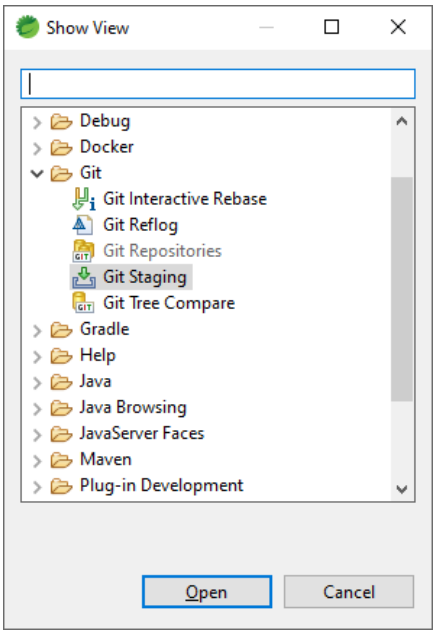
Luego de esto, se abrirá una ventana donde podremos seleccionar el repositorio disponible, eligiéndolo desde la lista desplegable del campo repository



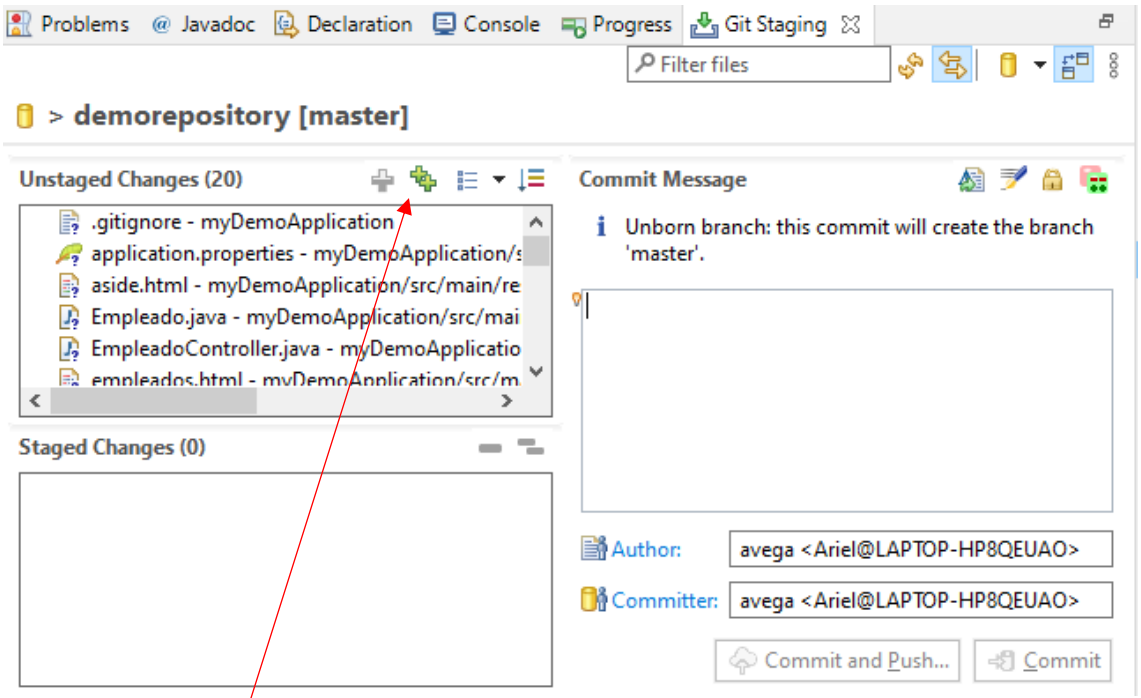
Finalmente, al presionar el botón Finish se iniciará el proceso de actualización del proyecto. Observe que ahora en el panel del proyecto aparece a cuál repositorio está asociado el proyecto. Además, aparece un signo de pregunta en el nombre del proyecto



Esto significa que hay elementos que han sido modificados. Para ver un detalle de los elementos modificados puede volver a visualizar el menú Window-> Show View -> Other -> Git -> Git Staging



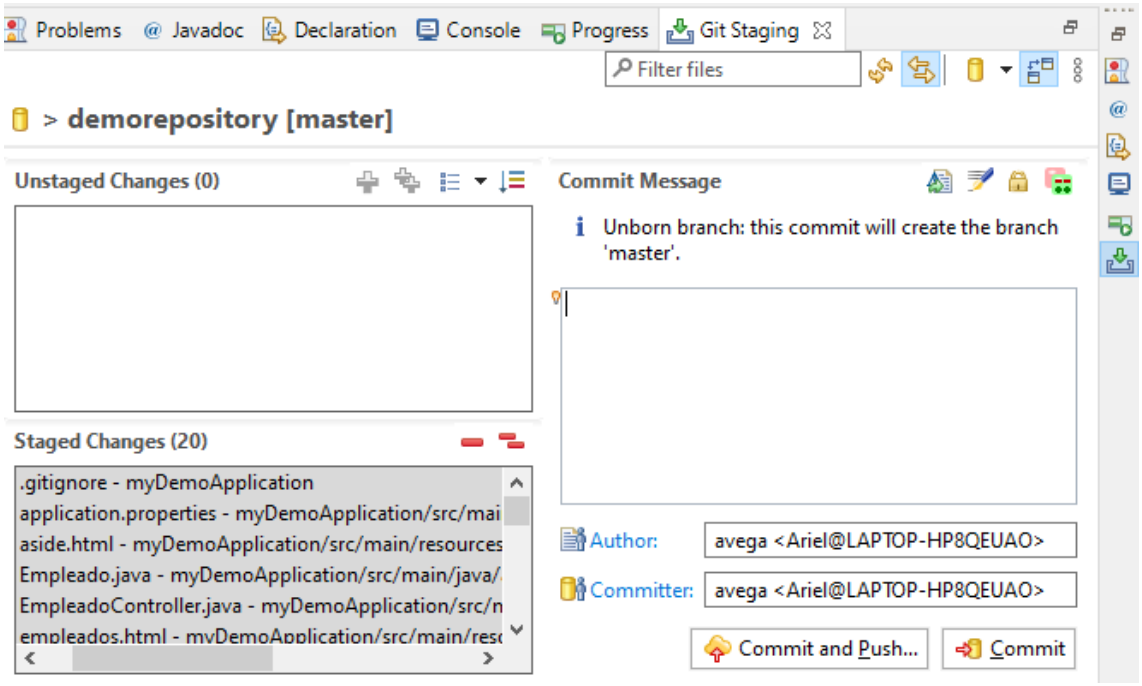
Con lo cual se abrirá un panel como el siguiente:



Donde en la ventana Unstaged Changes se visualiza la cantidad de cambios realizados (20 archivos nuevos) con respecto al repositorio original.

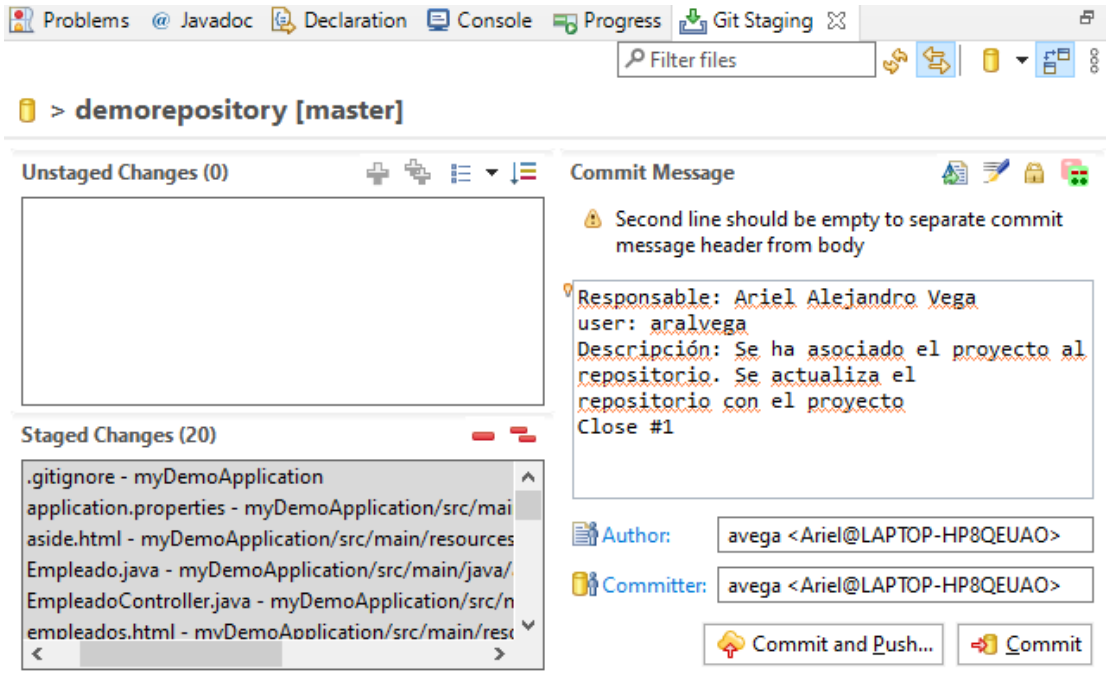
Paso 2: Agregar los cambios realizados al área de preparación (Stating área): el comando add

Una vez que tenemos archivos modificados debemos agregarlos al área de preparación. Para ello usamos el comando add de Git. Dado que estamos utilizando STS, este comando se ejecuta de manera transparente al presionar el botón con el signo de doble suma (++) en la ventana de Unstaged Changes. Esto provocará que todos los cambios pasen a la ventana de Staged Changes, tal como se observa en la siguiente imagen:



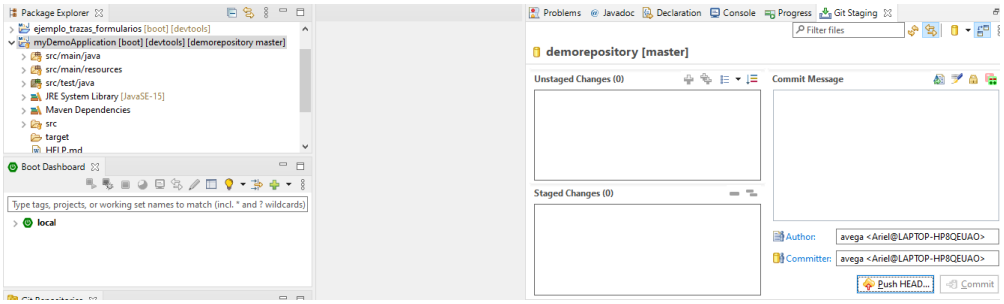
Paso 3: Confirmar los cambios realizados en el repositorio local: Comando commit

Los cambios que están en el área de preparación no han sido confirmados, por lo tanto, el repositorio local (working directory) no refleja los cambios que se generaron al asociar el proyecto con el repositorio. Para que los cambios se presenten en el repositorio local hay que ejecutar el comando commit. Adicionalmente se suele colocar un mensaje de confirmación (commit message) que permite realizar una traza de los cambios que se están realizando en el working directory. Observe lo que se ha colocado en el este mensaje



Observe que la ultima línea indica un comando interno close con el valor #1. Este valor representa al issue asociado a la tarjeta definida. Esto significa que al momento en que se actualice el servidor, esta tarjeta pasará del estado In progress al estado Done.

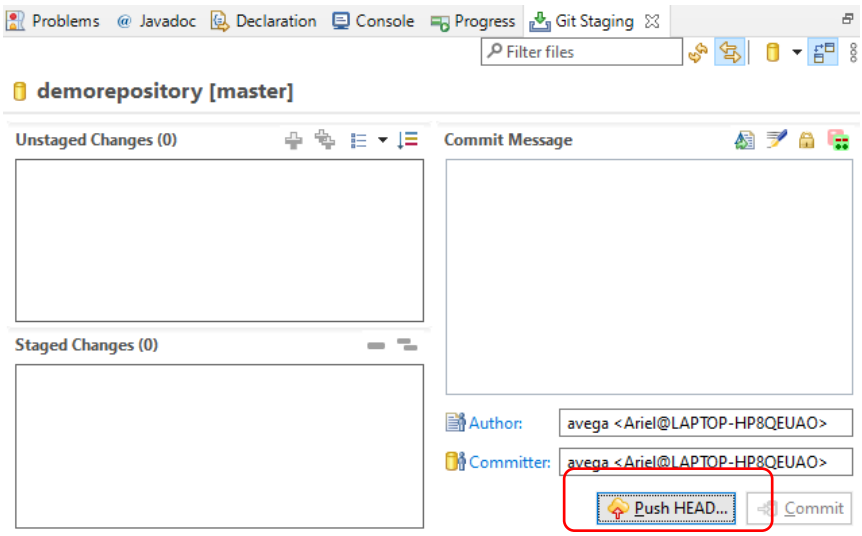
Bien como imaginar, si presiona el botón commit los cambios se aplicarán al repositorio local. Lo cual generará que los ítems desaparezcan del área de preparación. Además, ya no se observan los cambios indicados con el signo ? en explorador de proyectos, sino que se ha agregado un icono de repositorio de color amarillo al nombre del proyecto.



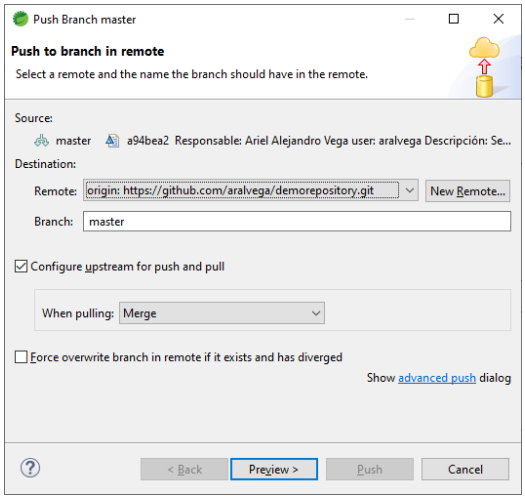
Paso 3: Subir el repositorio local al repositorio central: Comando push

Si volvemos a Github y observamos nuestro repositorio y el gestor de proyectos veremos que no se reflejan los cambios del paso anterior. Esto se debe a que no hemos ejecutado el comando push de Git. Este comando es quien efectivamente actualizará el git repository.

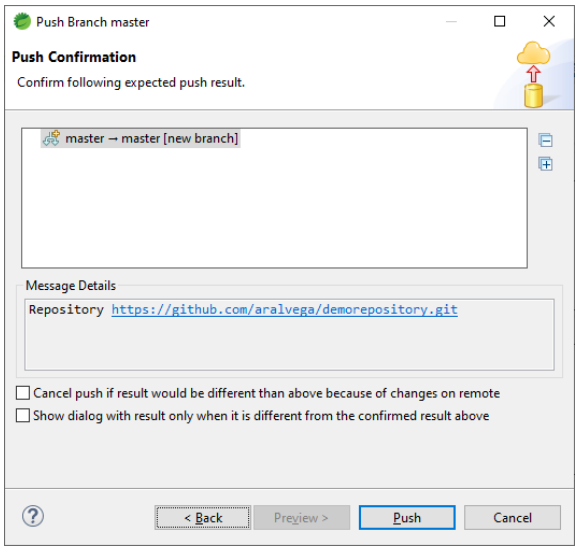
Debemos presionar el botón indicado en la siguiente imagen



Al presionar este botón se abrirá la siguiente ventana



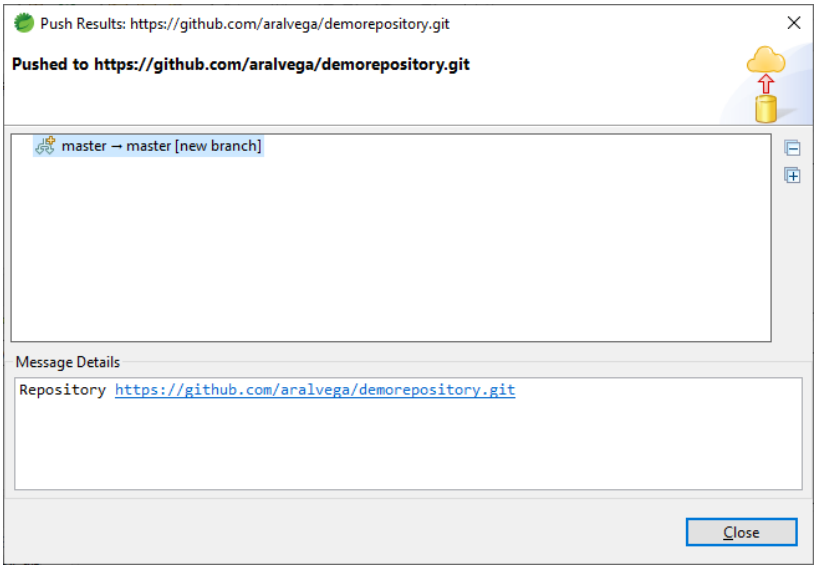
En esta ventana debemos confirmar sobre cual repositorio realizaremos la actualización y sobre cual rama, en este caso la rama Master, sobre la cual detallaremos conceptos en próximos apuntes. Por otro lado, puede observar que la estrategia elegida es Merge (algo sobre lo que entraremos en detalles más adelante). En definitiva, la ventana nos ofrece la opción de previsualización.



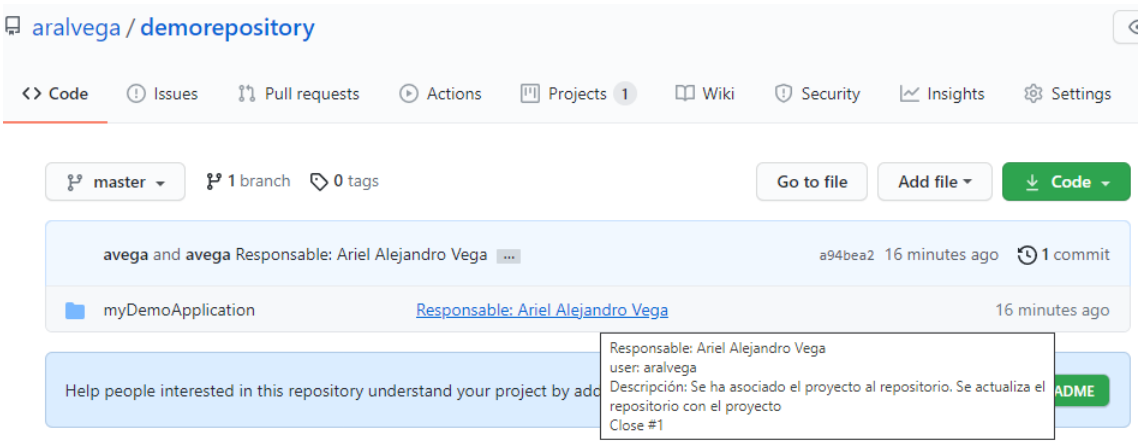
Dado que nos brinda un mensaje orientativo de que no habrá conflictos al actualizar el repsositorio central, entonce, simplemente debemos ejecutar el comando push

En la parte inferior se observará el porcentaje de avance de la operación

Y finalmente, observamos que se ha realizado los cambios



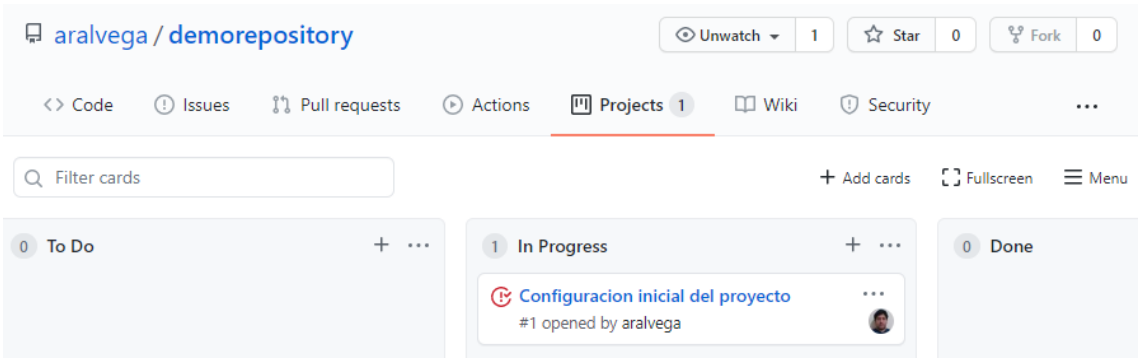
Podremos confirmar estos cambios observando dos cosas en Guthub, por un lado se puede visualizar la carpeta del proyecto en el repositorio



Hay mucha información disponible:

Por un lado, se indica quien ha realizado la tarea, quien es el responsable de ella (al posar sobre el enlace que dice responsable: Ariel Alejandro Vega). Esta información es la indicadada en el commit message. También se puede ver el historial de commit. Hasta ahora solo se posee un commit. Podríamos observar los detalles de los cambios, esto lo estudiaremos más adelante.

Por otro lado, en la sección de proyectos veremos lo siguiente



El ícono delante del issue indica que se ha cerrado la actividad. Por lo tanto está disponible para pasarlo a la columna Done.

En teoría el líder del proyecto revisará esta tarjeta y podrá confirmar que la misma ha finalizado o podrá reabrirla.

También podrá observar que en la solapa issues, se indica que hay uno cerrado y cero abiertos