

UNIVERSIDAD SAN PABLO DE GUATEMALA

Facultad de Ciencias Empresariales

Escuela de Ingeniería

Ingeniería en Ciencias y Sistemas de la Computación



REPOSITORIOS DE CÓDIGO FUENTE

Trabajo presentado en el curso de Programación II
Impartido por Lic. Mynor Alfonso López De León.

Yellsmy Lilibeth Toj García

2000579

Guatemala, 16 de septiembre de 2021.

REPOSITORIOS DE CÓDIGO FUENTE

es una instalación de archivo de archivos, una instalación de alojamiento web para el resguardo del código fuente de software, donde desarrolladores, diseñadores y programadores almacenan documentación, páginas web y toda clase de metadatos.

Aun cuando se habla indistintamente de servicios de alojamiento de repositorios de códigos fuente y de sistemas de control de versiones no son en realidad una misma cosa: Los sistemas de control de versiones son las utilidades de líneas de comando de bajo nivel empleadas para gestionar los cambios del ciclo de vida del desarrollo de software. Y los repositorios de código fuente son las aplicaciones web de terceros que encapsulan y mejoran a estos sistemas de control de versiones. Hecha esta salvedad, se trata de funcionalidades diferentes pero que operan de la mano, de ahí la confusión, y que no funcionan la una sin la otra.

El objetivo fundamental de estas herramientas es almacenar conjuntos de archivos de datos y llevar el historial de los cambios, las revisiones y las versiones que han ocurrido en ellos, el conjunto de objetos de confirmación y el histórico de referencias relacionadas (cabezas).

Los repositorios pueden ser públicos y privados y la escogencia de uno u otro dependerá del tamaño del equipo de trabajo y del grado de privacidad que requiera determinada organización en un momento dado en función de sus objetivos empresariales.

GitHub

En un nivel más alto, GitHub es un sitio web y un servicio en la nube que ayuda a los desarrolladores a almacenar y administrar su código, al igual que llevar un registro y control de cualquier cambio sobre este código.

La versión de control permite al desarrollador trabajar de forma segura a través de una bifurcación y una fusión.

Con la bifurcación, un desarrollador duplica parte del código fuente (llamado repositorio). Este desarrollador, luego puede, de forma segura, hacer cambios a esa parte del código, sin afectar al resto del proyecto.

Luego, una vez que el desarrollador logre que su parte del código funcione de forma apropiada, esta persona podría fusionar este código al código fuente principal para hacerlo oficial.

Todos estos cambios luego son registrados y pueden ser revertidos si es necesario.

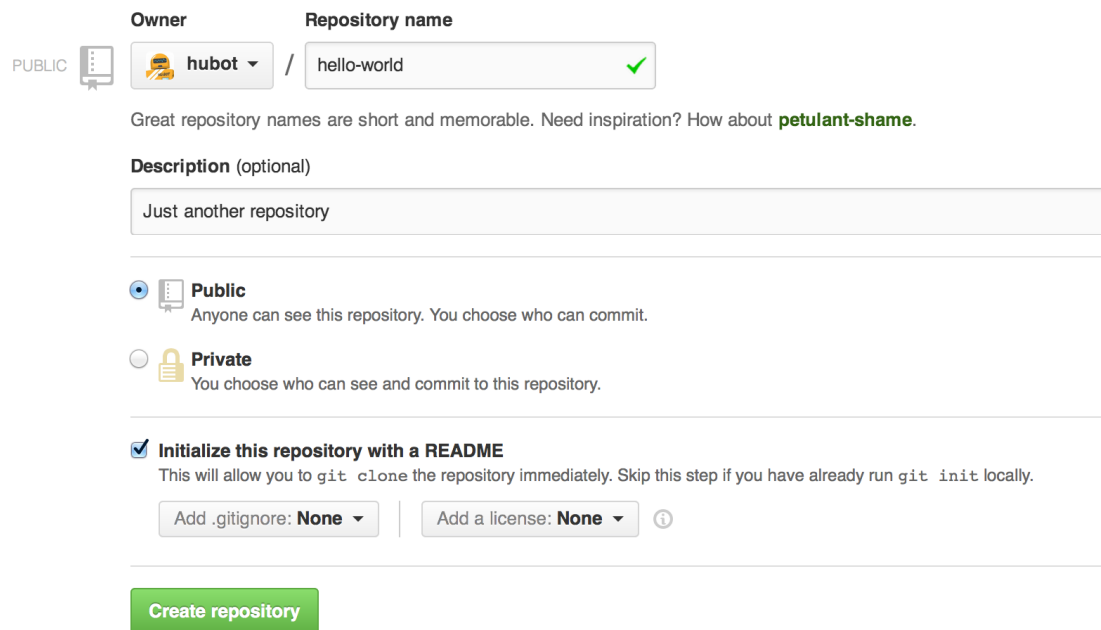
Cómo crear:

Paso 1. Crea un repositorio

Por lo general, se usa un repositorio para organizar un solo proyecto. Los repositorios pueden contener carpetas y archivos, imágenes, videos, hojas de cálculo y conjuntos de datos, todo lo que necesite su proyecto. Recomendamos incluir un archivo *README* o un archivo con información sobre su proyecto. GitHub facilita agregar uno al mismo tiempo que crea su nuevo repositorio. *También ofrece otras opciones comunes como un archivo de licencia.*

Para crear un nuevo repositorio:

1. En la esquina superior derecha, junto a su avatar o icono de identificación, haga clic en + y luego seleccione Nuevo repositorio .
2. Nombra tu repositorio `hello-world`, o como desees nombrarlo.
3. Escribe una breve descripción.
4. Seleccione Inicializar este repositorio con un archivo README .



The screenshot shows the GitHub 'Create new repository' form. At the top, there are two dropdown menus: 'Owner' (set to 'hubot') and 'Repository name' (set to 'hello-world' with a green checkmark). To the left of the 'Owner' dropdown is a 'PUBLIC' label with a lock icon. Below these is a text box for 'Description (optional)' containing the text 'Just another repository'. Underneath the description box are two radio button options: 'Public' (selected) and 'Private'. The 'Public' option is accompanied by the text 'Anyone can see this repository. You choose who can commit.' The 'Private' option is accompanied by the text 'You choose who can see and commit to this repository.' Below these options is a checked checkbox labeled 'Initialize this repository with a README', with a note: 'This will allow you to `git clone` the repository immediately. Skip this step if you have already run `git init` locally.' At the bottom of this section are two dropdown menus: 'Add .gitignore: None' and 'Add a license: None'. At the very bottom is a green button labeled 'Create repository'.

Haz clic en Crear repositorio.

Paso 2. Crea una sucursal

La ramificación es la forma de trabajar en diferentes versiones de un repositorio a la vez.

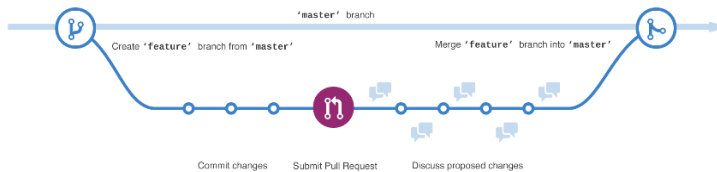
De forma predeterminada, su repositorio tiene una rama nombrada `main` que se considera la rama definitiva. Usamos ramas para experimentar y hacer ediciones antes de comprometerlas `main`.

Cuando crea una rama fuera de la `main` rama, está haciendo una copia, o instantánea, de `main` lo que estaba en ese momento. Si alguien más hizo cambios

en la `main` rama mientras trabajaba en su rama, podría incorporar esas actualizaciones.

Este diagrama muestra:

- La `main` rama
- Una nueva rama llamada `feature` (porque estamos haciendo 'trabajo de características' en esta rama)
- El viaje que `feature` lleva antes de que se fusione `main`



¿Alguna vez ha guardado diferentes versiones de un archivo? Algo como:

- `story.txt`
- `story-joe-edit.txt`
- `story-joe-edit-reviewed.txt`

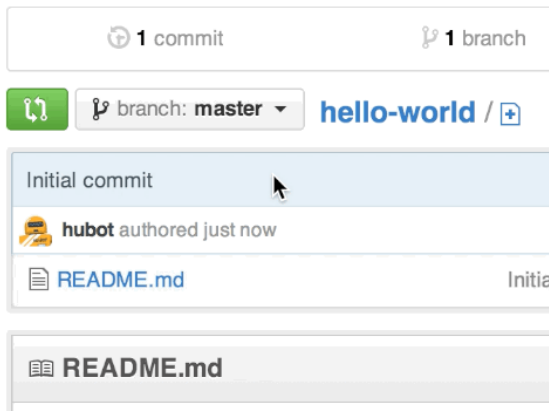
Las sucursales logran objetivos similares en los repositorios de GitHub.

En GitHub, nuestros desarrolladores, escritores y diseñadores usan ramas para mantener las correcciones de errores y el trabajo de funciones separado de nuestra `main` rama (de producción). Cuando un cambio está listo, fusionan su rama en `main`.

Para crear una nueva rama

1. Vaya a su nuevo repositorio `hello-world`, o como lo nombró.
2. Haga clic en el menú desplegable en la parte superior de la lista de archivos que dice `branch: main`.
3. Escriba un nombre de rama `readme-edits`, en el cuadro de texto de nueva rama.
4. Seleccione el cuadro azul Crear rama o presione "Enter" en su teclado.

Just another repository — Edit



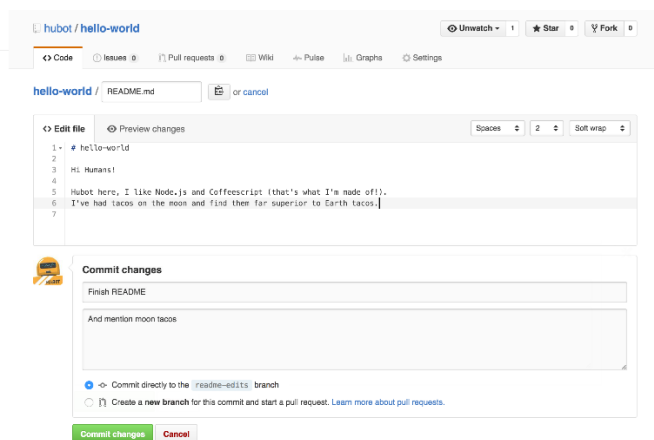
Ahora tienes dos ramas `main` y `readme-edits`. Se ven exactamente iguales, pero no lo son, a continuación, agregaremos nuestros cambios a la nueva rama.

Paso 3. Realice y confirme cambios

En GitHub, los cambios guardados se denominan *confirmaciones*. Cada confirmación tiene un *mensaje de confirmación* asociado, que es una descripción que explica por qué se realizó un cambio en particular. Los mensajes de confirmación capturan el historial de sus cambios, para que otros colaboradores puedan comprender lo que ha hecho y por qué.

Realizar y confirmar cambios

1. Haga clic en el `README.md` archivo.
2. Haga clic en el icono de lápiz en la esquina superior derecha de la vista de archivo para editar.
3. En el editor, escribe un poco sobre ti.
4. Escribe un mensaje de confirmación que describa tus cambios.
5. Haga clic en el botón Confirmar cambios.



Estos cambios se realizarán solo en el archivo README de su `readme-edits` rama, por lo que ahora esta rama contiene contenido que es diferente de `main`.

Paso 4. Abra una solicitud de extracción

Las solicitudes de extracción son el corazón de la colaboración en GitHub. Cuando abre una *solicitud de extracción*, está proponiendo sus cambios y solicitando que alguien revise y extraiga su contribución y los combine en su rama. Las solicitudes de *extracción* muestran diferencias del contenido de ambas ramas. Los cambios, sumas y restas se muestran en verde y rojo.

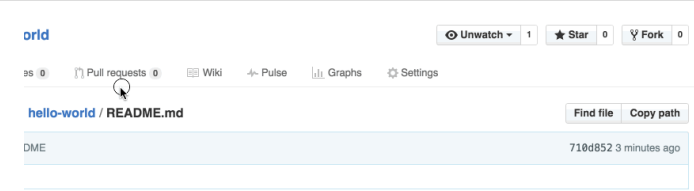

Tan pronto como realice una confirmación, puede abrir una solicitud de extracción e iniciar una discusión, incluso antes de que finalice el código.

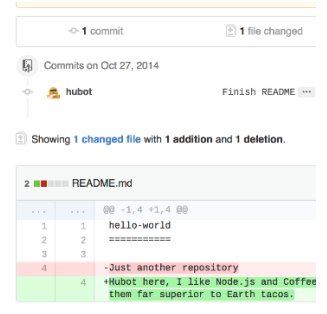
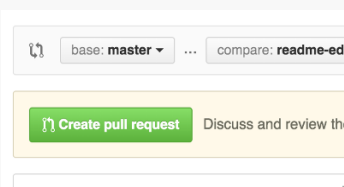
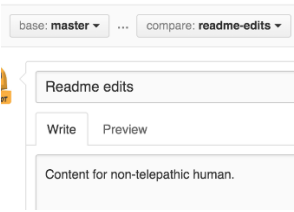
Al usar el sistema @mention de GitHub en su mensaje de solicitud de extracción, puede solicitar comentarios de personas o equipos específicos, ya sea que se encuentren en el pasillo o en 10 zonas horarias de distancia.

Incluso puede abrir solicitudes de extracción en su propio repositorio y fusionarlas usted mismo. Es una excelente manera de aprender el flujo de GitHub antes de trabajar en proyectos más grandes.

Abra una solicitud de extracción para cambios en el archivo README

Haga clic en la imagen para una versión más grande

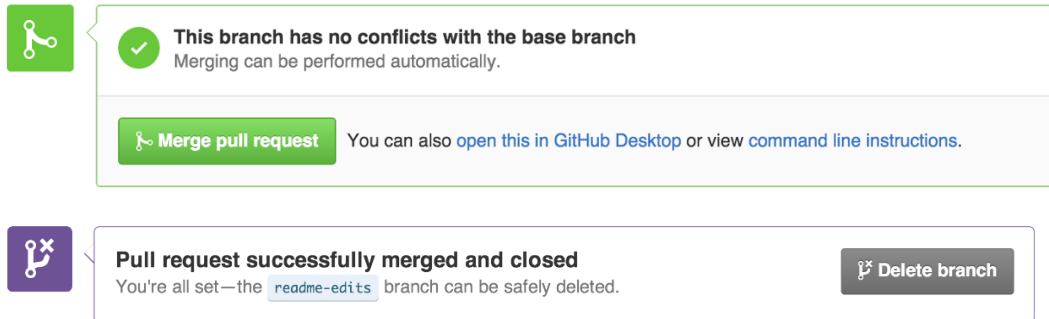
Paso	Captura de pantalla
Haga clic en el Pull Request , luego en la página Pull Request, haga clic en el botón verde New pull request .	
En el cuadro Comparaciones de ejemplo , seleccione la rama que hizo <code>readme-edits</code> , para comparar con <code>main</code> (el original).	

Paso	Captura de pantalla
Revise sus cambios en las diferencias en la página Comparar, asegúrese de que sean los que desea enviar.	
Cuando esté satisfecho de que estos son los cambios que desea enviar, haga clic en el botón verde grande Crear solicitud de extracción.	
Dale un título a tu solicitud de extracción y escribe una breve descripción de tus cambios.	

Cuando haya terminado con su mensaje, haga clic en Crear solicitud de extracción.

Paso 5. Fusiona tu solicitud de extracción

1. En este paso final, es hora de unir los cambios: fusionar su `readme-edits` rama en la `main` rama.
2. Haga clic en el botón verde Fusionar solicitud de extracción para fusionar los cambios en `main`.
3. Haga clic en Confirmar combinación.
4. Continúe y elimine la rama, ya que se han incorporado sus cambios, con el botón Eliminar rama en el cuadro morado.



The image shows two GitHub notification boxes. The top box is green and white, indicating a successful merge. It contains a green checkmark icon, the text "This branch has no conflicts with the base branch" and "Merging can be performed automatically." Below this is a green button labeled "Merge pull request" and a link to "open this in GitHub Desktop" or "view command line instructions." The bottom box is purple and white, indicating a pull request has been successfully merged and closed. It contains a purple icon of a branch being merged, the text "Pull request successfully merged and closed" and "You're all set—the `readme-edits` branch can be safely deleted." To the right of this box is a grey button labeled "Delete branch".

COMANDOS BÁSICOS

Aquí hay algunos comandos básicos de GIT que debes conocer:

git init creará un nuevo repositorio local GIT. El siguiente comando de Git creará un repositorio en el directorio actual:

```
git init
```

Como alternativa, puedes crear un repositorio dentro de un nuevo directorio especificando el nombre del proyecto:

```
git init [nombre del proyecto]
```

git clone se usa para copiar un repositorio. Si el repositorio está en un servidor remoto, usa:

```
git clone nombredeusuario@host:/path/to/repository
```

A la inversa, ejecuta el siguiente comando básico para copiar un repositorio local:

```
git clone /path/to/repository
```

git add se usa para agregar archivos al área de preparación. Por ejemplo, el siguiente comando de Git básico indexará el archivo temp.txt:

```
git add <temp.txt>
```

git commit creará una instantánea de los cambios y la guardará en el directorio git.

```
git commit -m "El mensaje que acompaña al commit va aquí"
```

Ten en cuenta que los cambios confirmados no llegarán al repositorio remoto.

git config puede ser usado para establecer una configuración específica de usuario, como el email, nombre de usuario y tipo de formato, etc. Por ejemplo, el siguiente comando se usa para establecer un email:


```
git config --global user.email tuemail@ejemplo.com
```

La opción `-global` le dice a GIT que vas a usar ese correo electrónico para todos los repositorios locales. Si quieres utilizar diferentes correos electrónicos para diferentes repositorios, usa el siguiente comando:

```
git config --local user.email tuemail@ejemplo.com
```

git status muestra la lista de los archivos que se han cambiado junto con los archivos que están por ser preparados o confirmados.

```
git status
```

git push se usa para enviar confirmaciones locales a la rama maestra del repositorio remoto. Aquí está la estructura básica del código:

```
git push origin <master>
```

Reemplaza `<master>` con la rama en la que quieres enviar los cambios cuando no quieras enviarlos a la rama maestra.

git checkout crea ramas y te ayuda a navegar entre ellas. Por ejemplo, el siguiente comando crea una nueva y automáticamente se cambia a ella:

```
command git checkout -b <branch-name>
```

Para cambiar de una rama a otra, sólo usa:

```
git checkout <branch-name>
```

git remote te permite ver todos los repositorios remotos. El siguiente comando listará todas las conexiones junto con sus URLs:

```
git remote -v
```

Para conectar el repositorio local a un servidor remoto, usa este comando:

```
git remote add origin <host-or-remoteURL>
```

Por otro lado, el siguiente comando borrará una conexión a un repositorio remoto especificado:

```
git remote <nombre-del-repositorio>
```

git branch se usa para listar, crear o borrar ramas. Por ejemplo, si quieres listar todas las ramas presentes en el repositorio, el comando debería verse así:

```
git branch
```

Si quieres borrar una rama, usa:

```
git branch -d <branch-name>
```

git pull fusiona todos los cambios que se han hecho en el repositorio remoto con el directorio de trabajo local.

```
git pull
```

git merge se usa para fusionar una rama con otra rama activa:

```
git merge <branch-name>
```

git diff se usa para hacer una lista de conflictos. Para poder ver conflictos con respecto al archivo base, usa:

```
git diff --base <file-name>
```

El siguiente comando se usa para ver los conflictos que hay entre ramas antes de fusionarlas:

```
git diff <source-branch> <target-branch>
```

Para ver una lista de todos los conflictos presentes usa:

```
git diff
```

git tag marca commits específicos. Los desarrolladores lo usan para marcar puntos de lanzamiento como v1.0 y v2.0.

```
git tag 1.1.0 <insert-commitID-here>
```

git log se usa para ver el historial del repositorio listando ciertos detalles de la confirmación. Al ejecutar el comando se obtiene una salida como ésta:

```
commit 15f4b6c44b3c8344caasdac9e4be13246e21sadm
```

```
Author: Alex Hunter <alexh@gmail.com>
```

```
Date: Mon Oct 1 12:56:29 2016 -0600
```

git reset sirve para resetear el index y el directorio de trabajo al último estado de confirmación.

```
git reset - -hard HEAD
```

git rm se puede usar para remover archivos del index y del directorio de trabajo.

```
git rm filename.txt
```

git stash guardará momentáneamente los cambios que no están listos para ser confirmados. De esta manera, puedes volver al proyecto más tarde.

```
git stash
```

git show se usa para mostrar información sobre cualquier objeto git.

```
git show
```

git fetch le permite al usuario buscar todos los objetos de un repositorio remoto que actualmente no se encuentran en el directorio de trabajo local.

```
git fetch origin
```

git ls-tree te permite ver un objeto de árbol junto con el nombre y modo de cada ítem, y el valor blob de SHA-1. Si quieres ver el HEAD, usa:

```
git ls-tree HEAD
```

git cat-file se usa para ver la información de tipo y tamaño de un objeto del repositorio. Usa la opción **-p** junto con el valor SHA-1 del objeto para ver la información de un objeto específico, por ejemplo:

```
git cat-file -p d670460b4b4aece5915caf5c68d12f560a9fe3e4
```

git grep le permite al usuario buscar frases y palabras específicas en los árboles de confirmación, el directorio de trabajo y en el área de preparación. Para buscar por www.hostinger.com en todos los archivos, usa:

```
git grep "www.hostinger.com"
```

gitk muestra la interfaz gráfica para un repositorio local. Simplemente ejecuta:

```
gitk
```

git instaweb te permite explorar tu repositorio local en la interfaz GitWeb. Por ejemplo:

```
git instaweb -http=webrick
```

git gc limpiará archivos innecesarios y optimizará el repositorio local.

```
git gc
```

git archive le permite al usuario crear archivos zip o tar que contengan los constituyentes de un solo árbol de repositorio. Por ejemplo:

```
git archive - -format=tar master
```

git prune elimina los objetos que no tengan ningún apuntador entrante.

```
git prune
```

git fsck realiza una comprobación de integridad del sistema de archivos git e identifica cualquier objeto corrupto

```
git fsck
```

git rebase se usa para aplicar ciertos cambios de una rama en otra. Por ejemplo:

```
git rebase master
```

OTRAS OPCIONES:

1.Bitbucket

Bitbucket es un servicio de alojamiento de repositorios de control de versiones, que fue creado en 2008 y es propiedad de Atlassian. Esta solución de gestión de repositorios de Git está escrita en Python, y construida utilizando el marco web de Django. En términos de funcionalidad, Bitbucket y GitHub funcionan de una manera muy similar. Con ambos, puedes realizar comandos básicos como:

- Creación y gestión de repositorios
- Conectarse usando la Autenticación de Dos Factores (2FA)
- Hacer peticiones de extracción
- Realizar revisiones del código
- Usar la edición en línea y el soporte de Markdown
- Realizar el seguimiento de los problemas

Las características adicionales de Bitbucket incluyen:

1. Integración directa con Jira, Bamboo, Crucible y Jenkins
2. La capacidad de importar repositorios desde Git, Codeplex, GoogleCode y SVN
3. Soporte de autenticación externa para GitHub, Google, Facebook y Twitter
4. Integración masiva con Trello
5. Un cliente para Mac y Windows (Sourcetree) y una aplicación para Android (Bitbeaker)

2.SourceForge

Datándose en 1999 fue el primer sitio web open source de este tipo. El código fuente del software de SourceForge estaba a disposición del público durante los primeros años, hasta que en 2001 no se volvieron a lanzar revisiones “oficiales” del mismo.

No obstante, con el paso de los años y a pesar de no liberar nuevas versiones de su código, sus desarrolladores no han dejado de añadir funcionalidades más allá del sencillo gestor de versiones, como son el seguimiento de errores, Pull Request, mirrors para una gestión de descarga más veloz... Proporcionando al público y sus proyectos un alojamiento web gratuito y muy eficaz.

Características:

1. El primer hosting para el control de versiones.
2. Sólo aloja proyectos open source.
3. Seguimiento de fallos.
4. Pull Request.
5. Soporte para CVS, SVN, Git y Mercurial.

3.GitLab

Otra plataforma muy similar a GitHub, escrita en Ruby y que cuenta con tres versiones que ofrecer: Community Edition (CE), Enterprise Edition (EE) y su versión de alojamiento en los servidores de la propia GitLab. Además, entidades como: NASA, CERN, Alibaba hacen uso de su software.

Sus niveles de permisos, protección de ramificaciones, controles de autenticación, en definitiva, la seguridad que ofrece tanto a usuarios como a proyectos hace que destaque sobre sus rivales de sector.

Características:

1. Una muy trabajada interfaz ofrece acceso y gestión de todas sus características desde una única pantalla (proyectos, usuarios, comentarios, estadísticas...)
2. “Snippet Support” o “soporte de fragmentos” facilita a los usuarios compartir, sugerir o añadir pequeños bloques de código sin necesidad de que el propietario comparta el proyecto al completo.
3. Protección de las ramificaciones con diferentes niveles de autorización en la realización de cambios.
4. Estrictos niveles de autenticación ofrecen un plus de seguridad en comparación con su competencia, pudiendo incluso establecer verificación en dos pasos, permisos de lectura / escritura, entre otros...
5. Permite establecer “metas” con las que el equipo de desarrollo podrá tener presente en todo momento una fecha límite, así como controlar el flujo de trabajo general.
6. La etiqueta “Work in Progress” marcará las ramas o proyectos que estén siendo modificados en tiempo real por los usuarios designados.
7. Cuenta con un eficaz sistema de notificaciones (clásicas o de tipo “push”) con el que la comunicación entre el equipo de desarrollo será mucho más fluida.