Actividad 4 - Utilizando sistemas de control de versiones

Brayan Steven Bonilla Castellanos

Juan Carlos Monsalve Gómez

Corporación Universitaria Iberoamericana

Ingeniería de Software

Mantenimiento de Software

**CONTENIDO**

[1 Enlace Git 3](#_Toc120427356)

[2 Descripción del contexto 4](#_Toc120427357)

[3 Flujo de trabajo inicial 5](#_Toc120427358)

[4 Historias de usuario 9](#_Toc120427359)

[4.1 Historia de Usuario 1: API servicios desarrollados 9](#_Toc120427360)

[4.2 Historia de usuario 2: Servicio REST tipo POST 12](#_Toc120427361)

[4.3 Historia de usuario 3: vista para registro de horas 17](#_Toc120427362)

[4.4 Historia de usuario 4: servicio REST tipo GET 20](#_Toc120427363)

[4.5 Historia de usuario 5: vista consulta horas 25](#_Toc120427364)

[5 Documentación de comandos GIT usados 29](#_Toc120427365)

[5.1 git clone 29](#_Toc120427366)

[5.2 git status 29](#_Toc120427367)

[5.3 git add . 29](#_Toc120427368)

[5.4 git commit -m 30](#_Toc120427369)

[5.5 git pull 30](#_Toc120427370)

[5.6 git push 30](#_Toc120427371)

[5.7 git checkout 31](#_Toc120427372)

[5.8 git merge 31](#_Toc120427373)

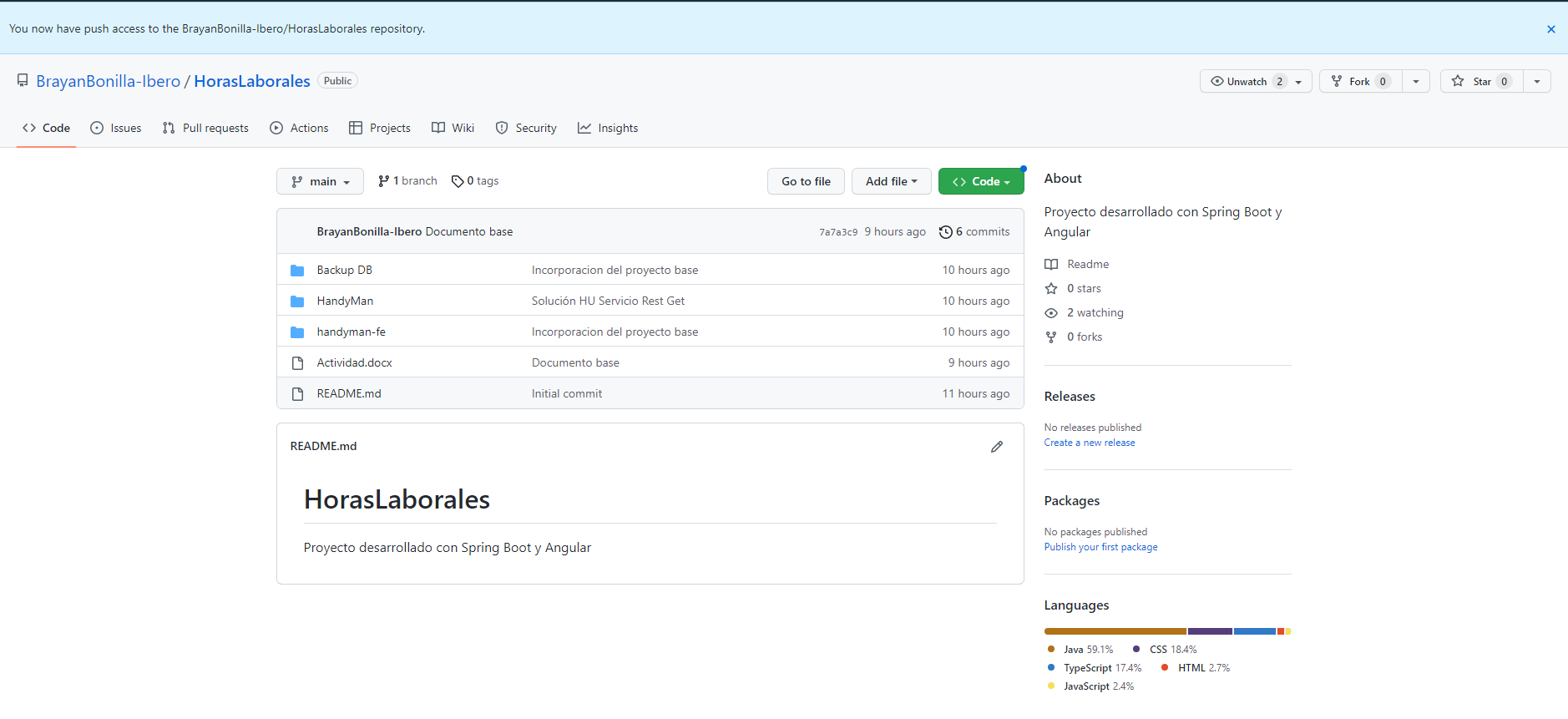
[5.9 git branch 31](#_Toc120427374)

[5.10 git log 31](#_Toc120427375)

[6 Referencias 32](#_Toc120427376)

# Enlace Git

[**https://github.com/BrayanBonilla-Ibero/HorasLaborales**](https://github.com/BrayanBonilla-Ibero/HorasLaborales)



# Descripción del contexto

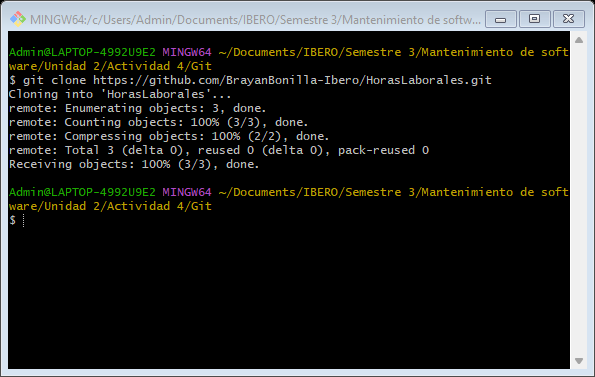
Se requiere implementar un sistema de información que permita el registro de horas laboradas por cada trabajador (Horas diurnas, horas nocturnas, horas dominicales, horas diurnas extras, horas nocturnas extras, horas dominicales extras).

El sistema es elaborado por un equipo de trabajo compuesto por dos desarrolladores, se utiliza el software GIT como sistema de control de versiones y todo el código fuente de la aplicación aparece almacenado en GitHub.

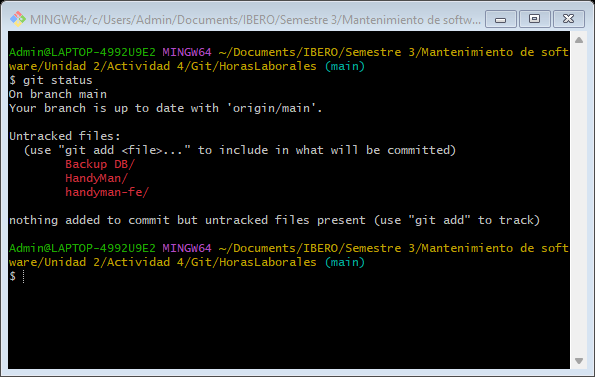
La aplicación se desarrolla en el Backend con Java con Spring Boot y en el FrontEnd con Angular v14.

# Flujo de trabajo inicial

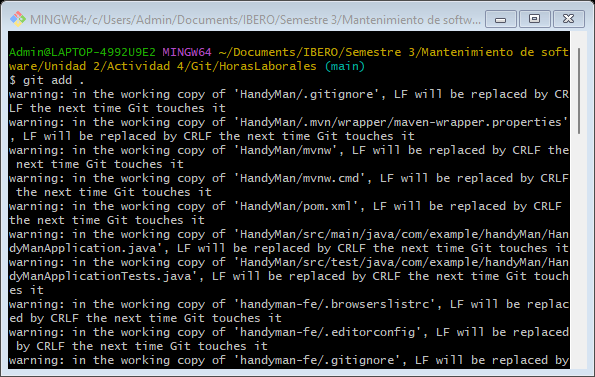
Ingresamos el comando **git clone**, para clonar el repositorio remoto al equipo local.



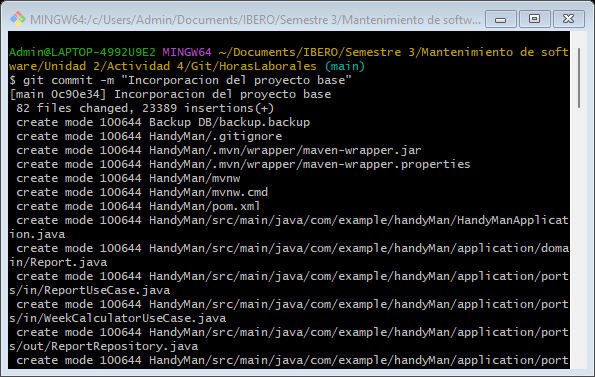
Una vez clonado el repositorio, incorporamos el proyecto base y verificamos los cambios realizados con el comando **git status**



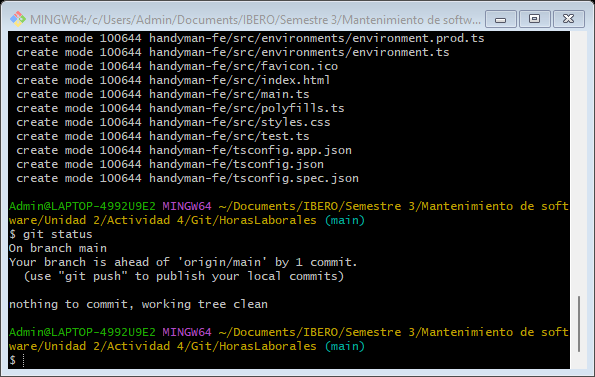
Procedemos a agregar los cambios realizados a un stage con el comando **git add .**



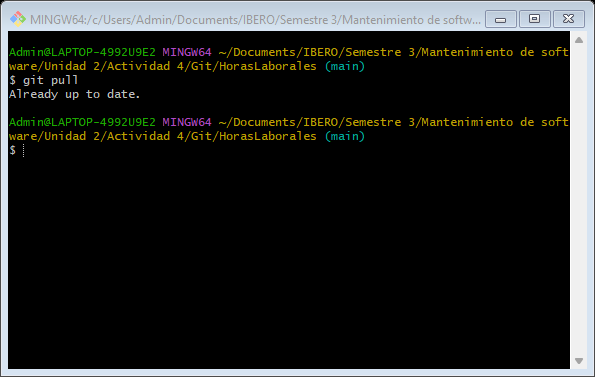
Una vez confirmado los cambios procedemos a realizar un commit, con el comando **git commit –m**



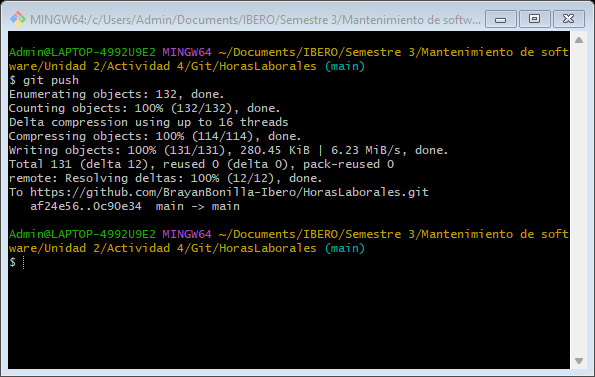
Verificamos que el commit se registrara con éxito, con el comando git status y debe registrar 1 commit pendiente por subir



Verificamos que no tengamos cambios pendientes por obtener con el comando **git pull**



Procedemos a subir los cambios al repositorio remoto en la rama principal, con el comando **git push**

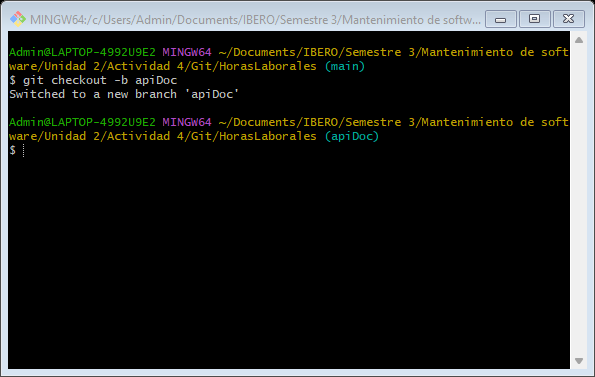


# Historias de usuario

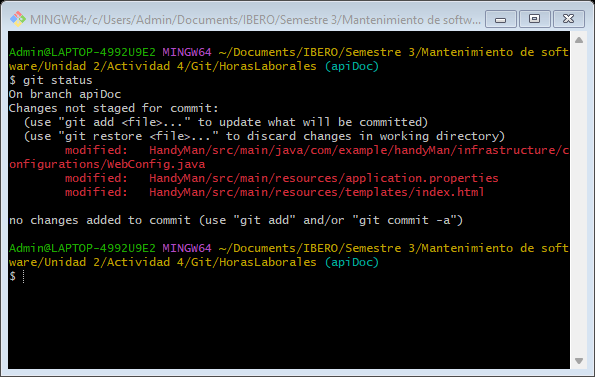
## Historia de Usuario 1: API servicios desarrollados

**Se requiere la construcción de un api que contenga la documentación de los servicios desarrollados.**

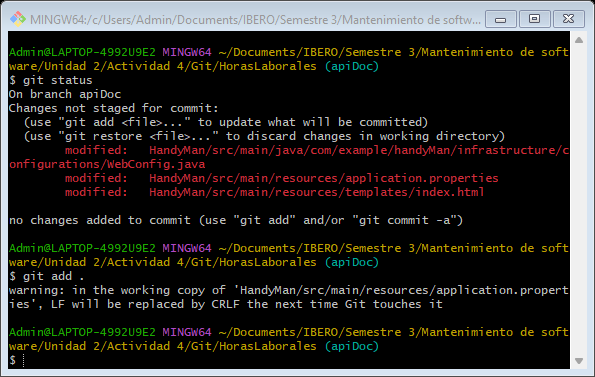
Para llevar a cabo esta tarea vamos a crear una rama específica para poder trabajar esta HU, con el comando **git checkout -b apiDoc**



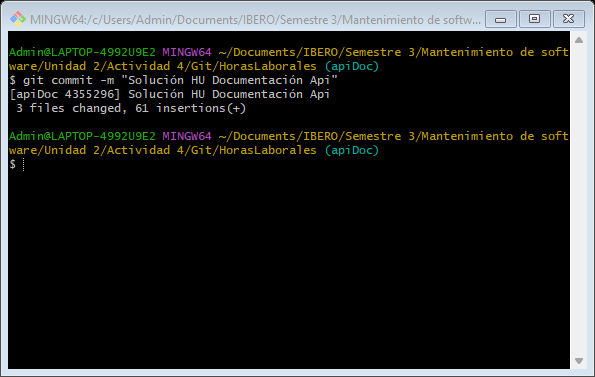
Una vez se trabaja la HU en el código fuente, procedemos a verificar los cambios, con el comando **git status**



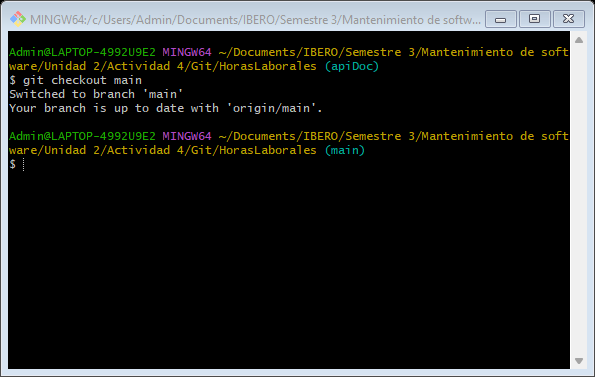
Agregamos los cambios al stage, con el comando **git add .**



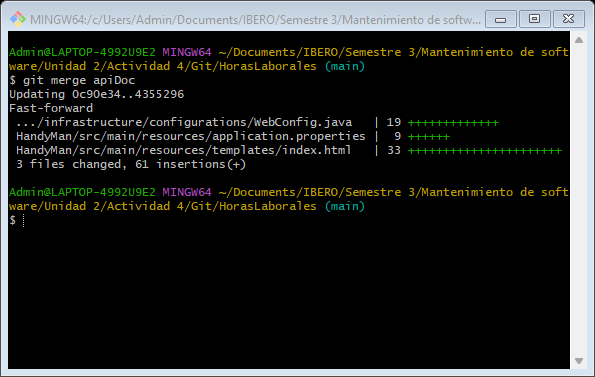
Realizamos el commit con los cambios realizados con el siguiente comando



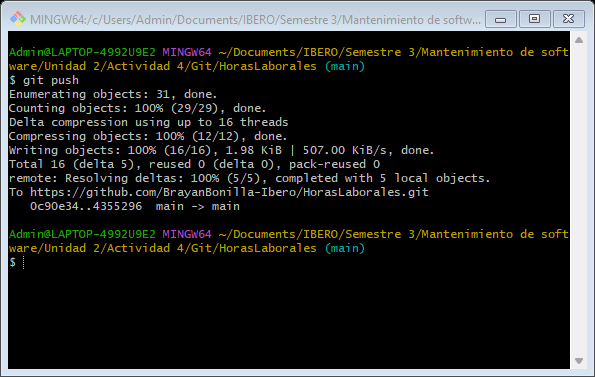
Una vez realizado el commit, procedemos a pasar a la rama principal con el siguiente comando



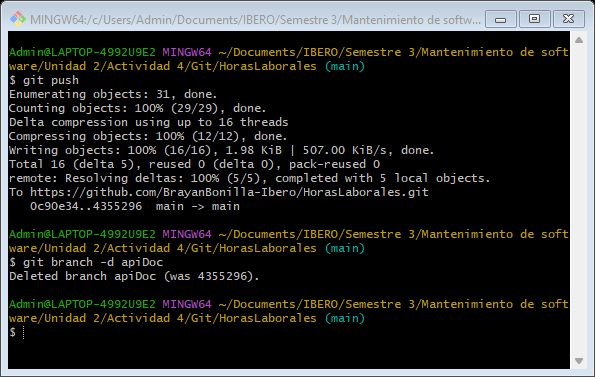
Una vez verificada la solución a la historia de usuario, procedemos a realizar el merge con la rama principal, con el siguiente comando



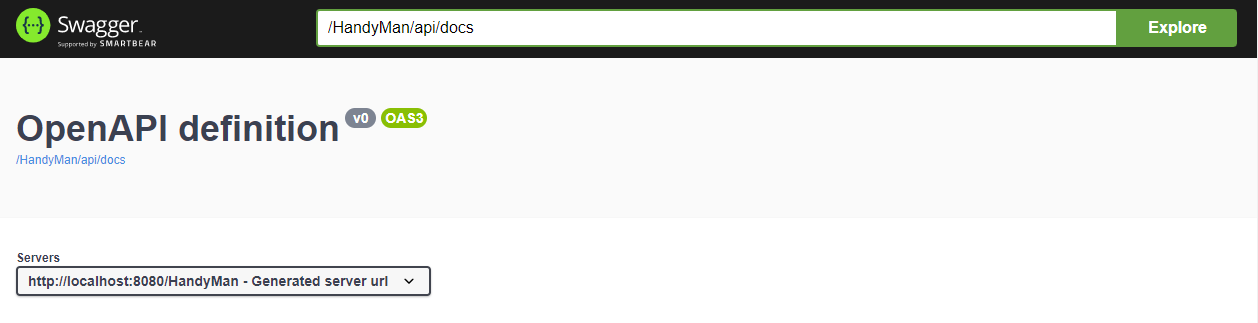
Subimos los cambios incorporados a la rama principal al repositorio remoto, con el siguiente comando



Como ya nos vamos a utilizar más la rama creada para trabajar la HU, procedemos a borrarla con el siguiente comando



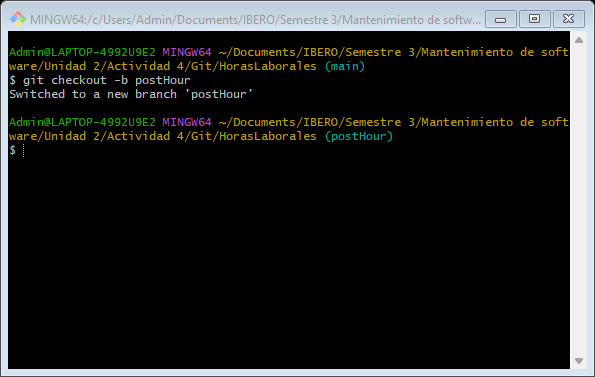
El resultado obtenido es el siguiente



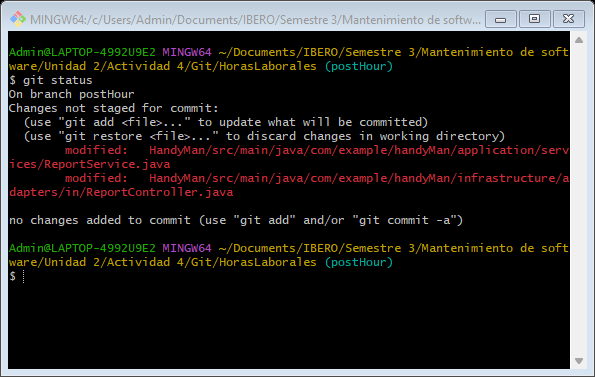
## Historia de usuario 2: Servicio REST tipo POST

**Se requiere crear un servicio Rest de tipo Post que permita guardar el registro de las horas por trabajador.**

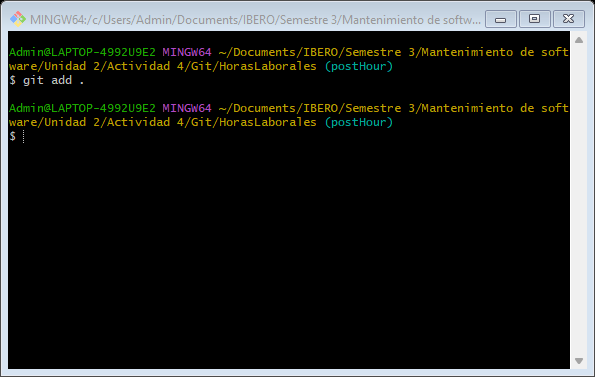
Para llevar a cabo esta tarea vamos a crear una rama específica para poder trabajar esta HU, con el comando **git checkout -b postHour**



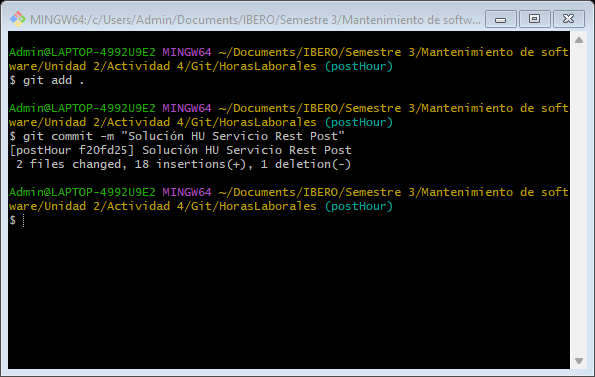
Una vez trabaja la HU en el código fuente, procedemos a verificar los cambios, con el comando git status



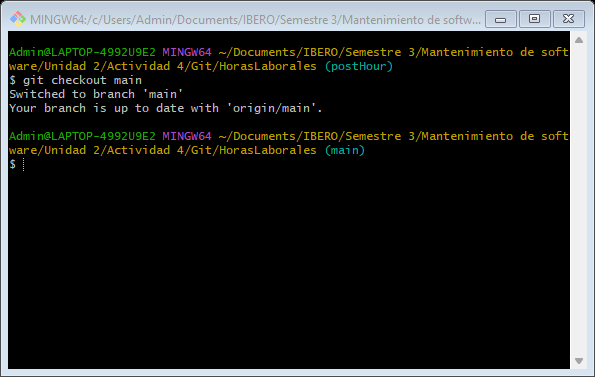
Agregamos los cambios al stage, con el comando **git add .**



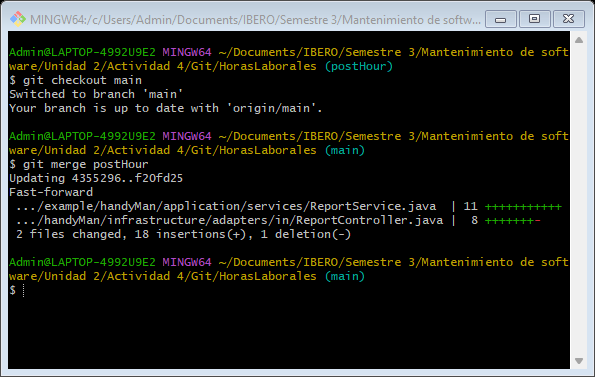
Realizamos el commit con los cambios realizados con el siguiente comando



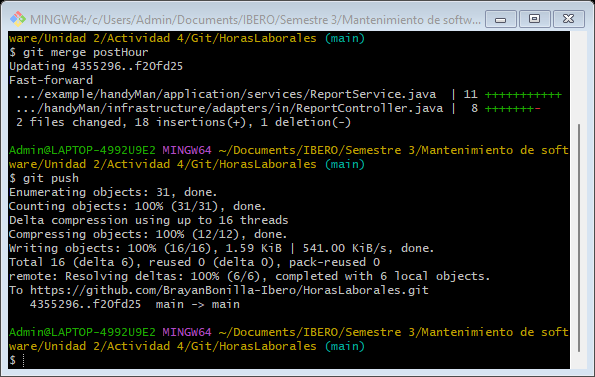
Una vez realizado el commit, procedemos a pasar a la rama principal con el siguiente comando



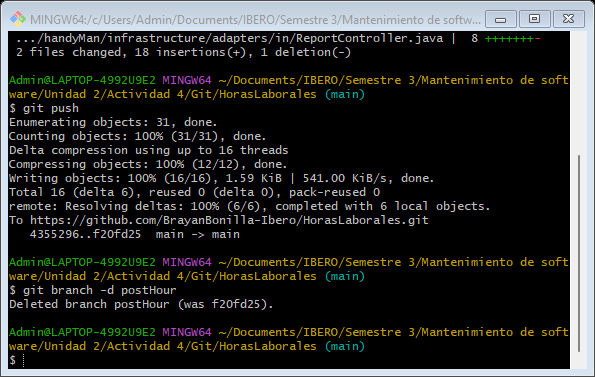
Una vez verificada la solución a la historia de usuario, procedemos a realizar el merge con la rama principal, con el siguiente comando



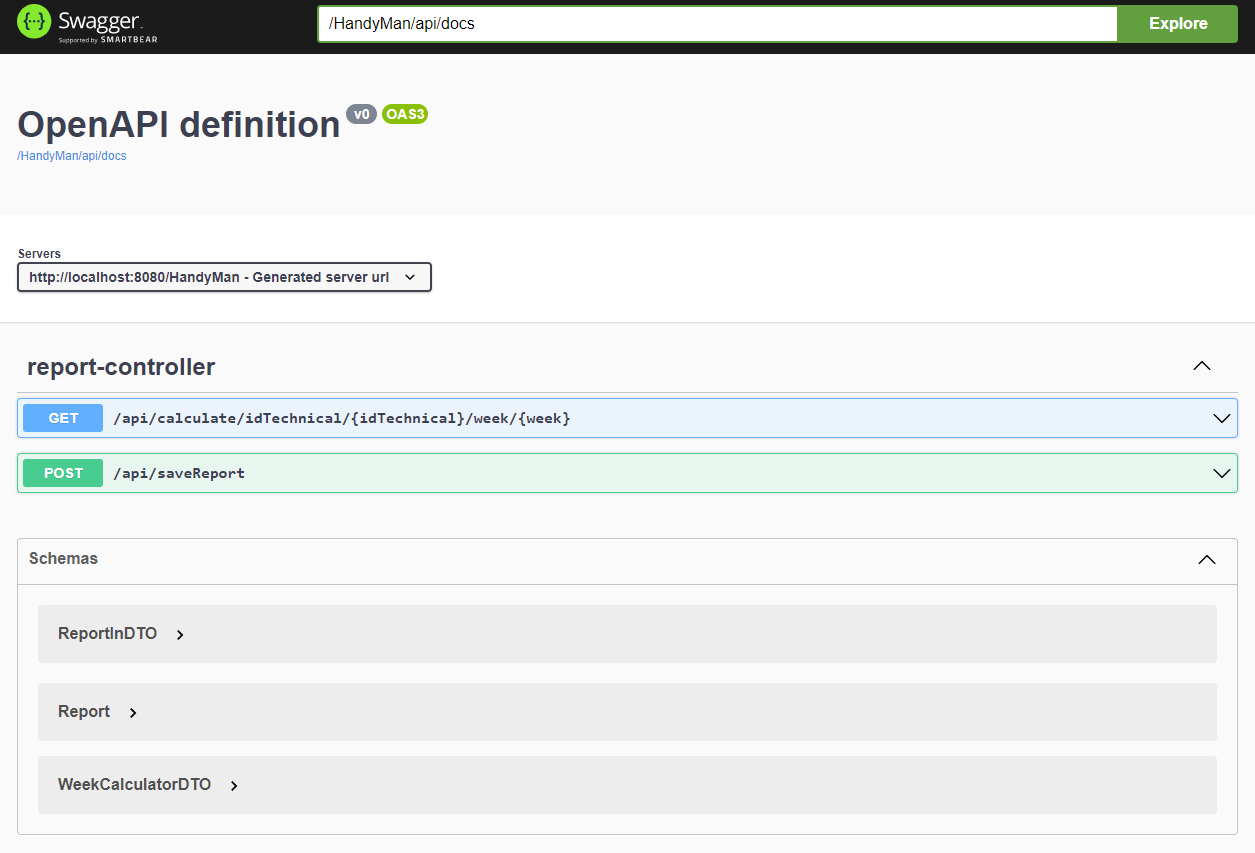
Subimos los cambios incorporados a la rama principal al repositorio remoto, con el siguiente comando



Como ya nos vamos a utilizar más la rama creada para trabajar la HU, procedemos a borrarla con el siguiente comando



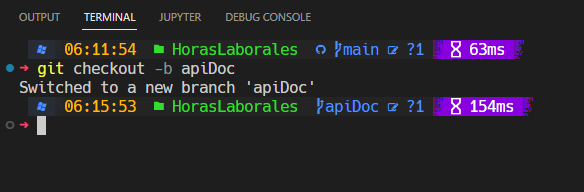
El resultado obtenido es el siguiente



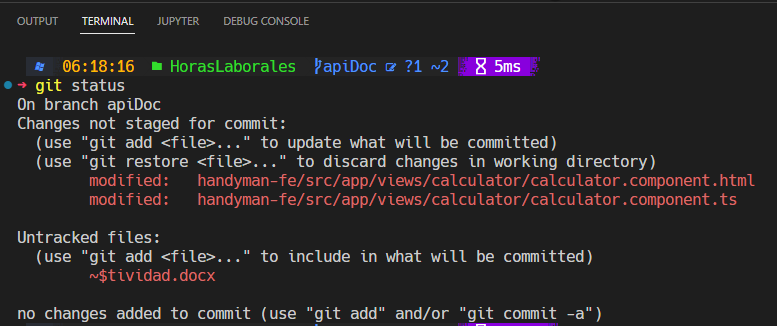
## Historia de usuario 3: vista para registro de horas

**Se requiere diseñar la vista al usuario que le permita registrar sus horas laboradas.**

Para llevar a cabo esta tarea vamos a crear una rama específica para poder trabajar esta hu, con el comando **git checkout -b apiDoc**



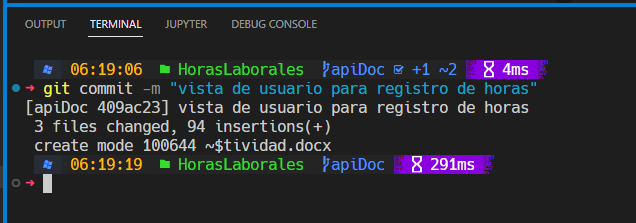
Una vez trabaja la HU en el código fuente, procedemos a verificar los cambios, con el comando **git status**



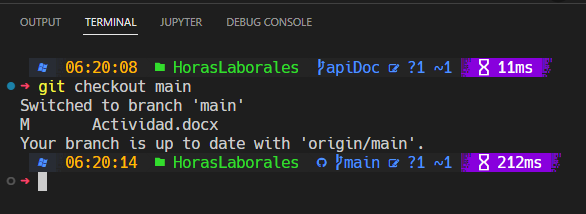
Agregamos los cambios al stage, con el comando **git add .**



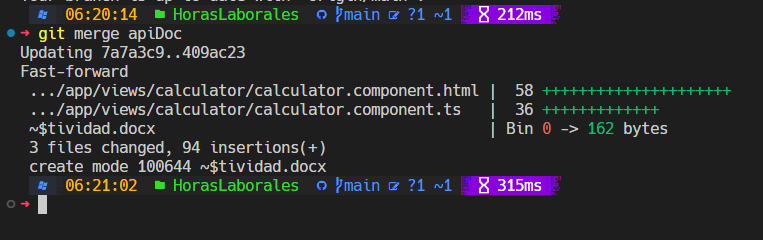
Realizamos el commit con los cambios realizados con el siguiente comando



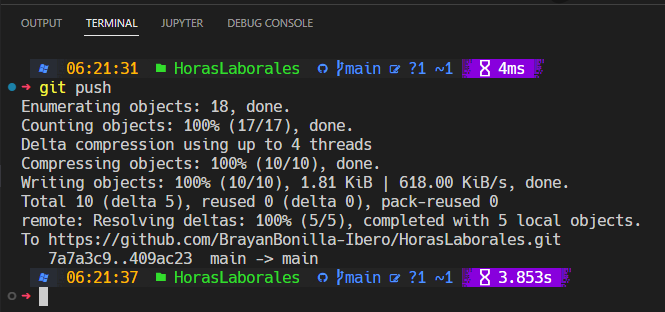
Una vez realizado el commit, procedemos a pasar a la rama principal con el siguiente comando



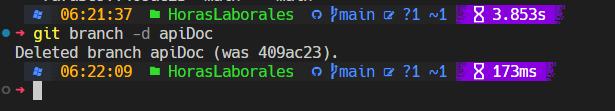
Una vez verificada la solución a la historia de usuario, procedemos a realizar el merge con la rama principal, con el siguiente comando



Subimos los cambios incorporados a la rama principal al repositorio remoto, con el siguiente comando



Como ya nos vamos a utilizar más la rama creada para trabajar la HU, procedemos a borrarla con el siguiente comando

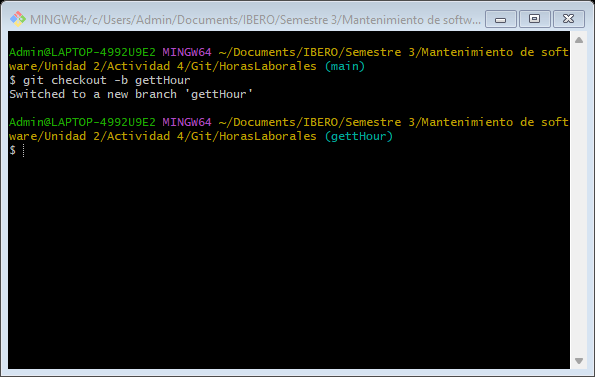


El resultado obtenido es el siguiente

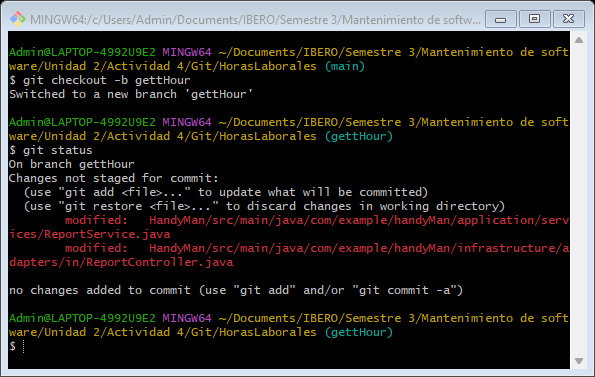
## Historia de usuario 4: servicio REST tipo GET

**Se requiere crear un servicio Rest de tipo Get que permita consultar las horas laboradas de un trabajador en una semana determinada.**

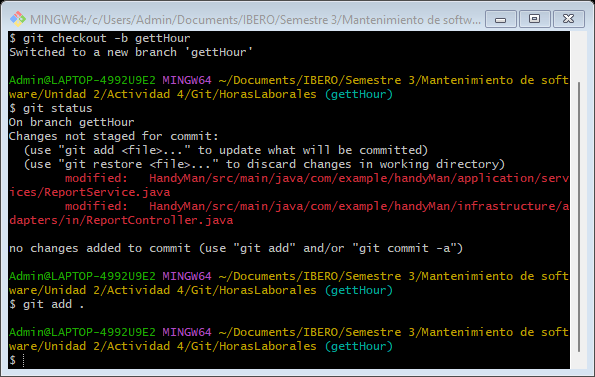
Para llevar a cabo esta tarea vamos a crear una rama específica para poder trabajar esta hu, con el comando git checkout -b getHour



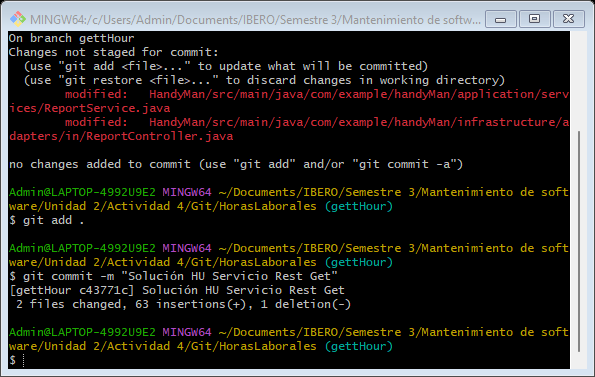
Una vez trabaja la HU en el código fuente, procedemos a verificar los cambios, con el comando git status



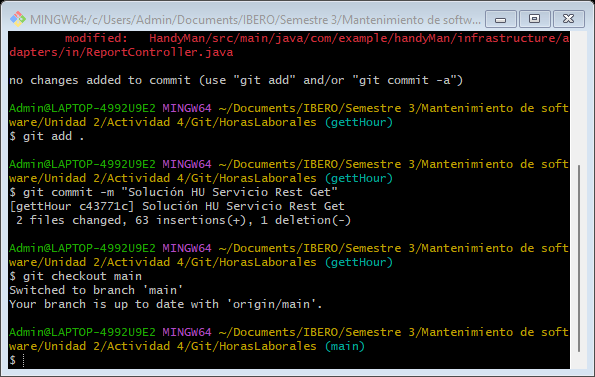
Agregamos los cambios al stage, con el comando git add .



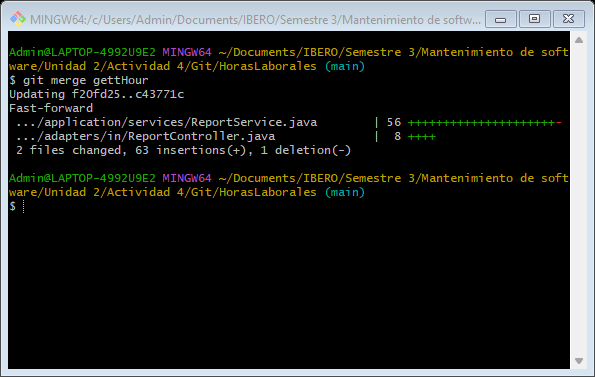
Realizamos el commit con los cambios realizados con el siguiente comando



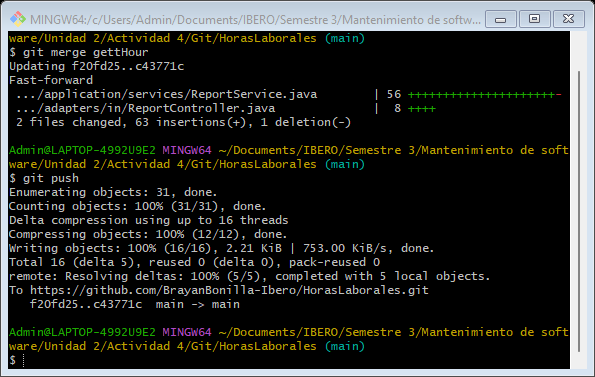
Una vez realizado el commit, procedemos a pasar a la rama principal con el siguiente comando



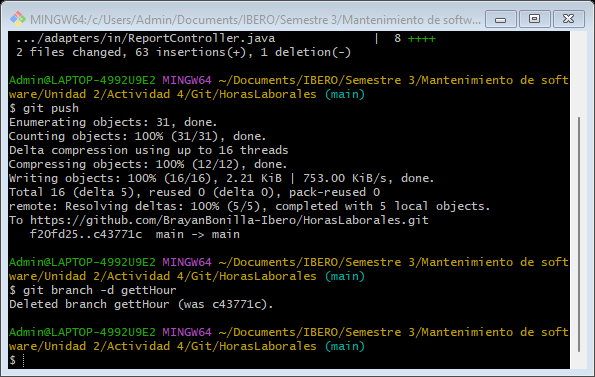
Una vez verificada la solución a la historia de usuario, procedemos a realizar el merge con la rama principal, con el siguiente comando



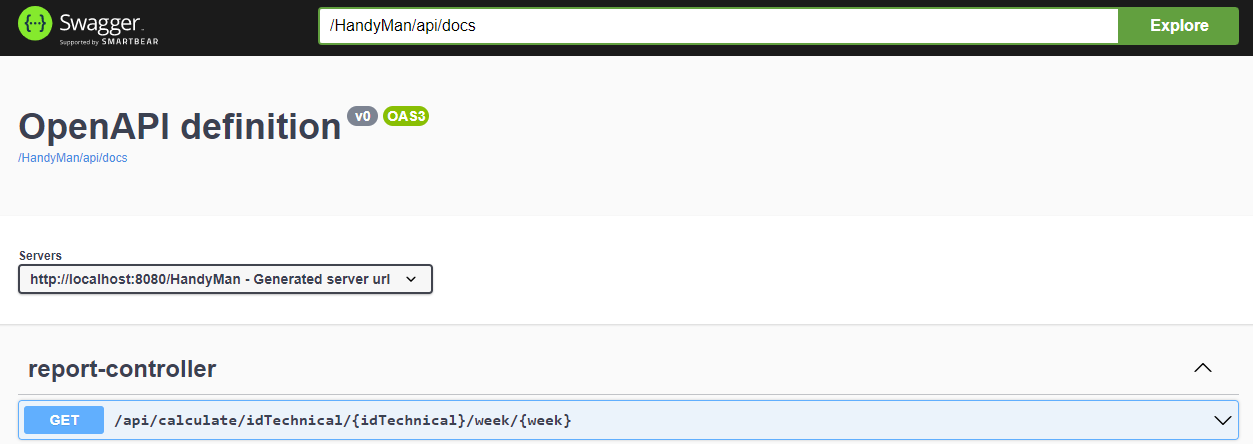
Subimos los cambios incorporados a la rama principal al repositorio remoto, con el siguiente comando



Como ya nos vamos a utilizar más la rama creada para trabajar la HU, procedemos a borrarla con el siguiente comando



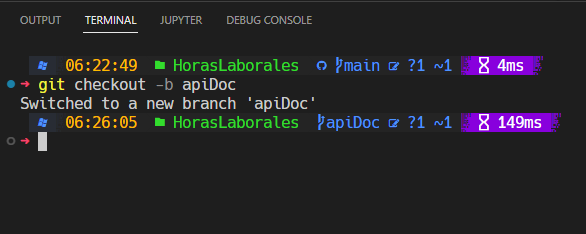
El resultado obtenido es el siguiente



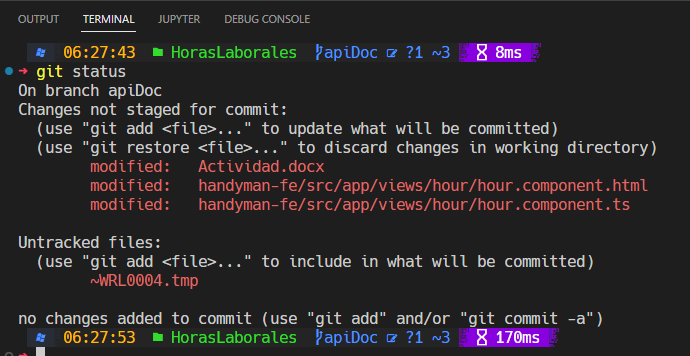
## Historia de usuario 5: vista consulta horas

**Se requiere diseñar la vista al usuario que le permita consultar las horas laboradas en una semana determinada.**

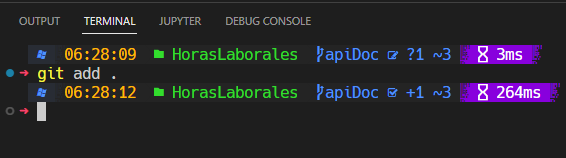
Para llevar a cabo esta tarea vamos a crear una rama específica para poder trabajar esta hu, con el comando git checkout -b apiDoc



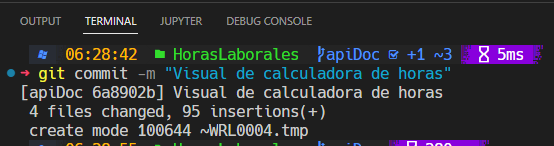
Una vez trabaja la HU en el código fuente, procedemos a verificar los cambios, con el comando git status



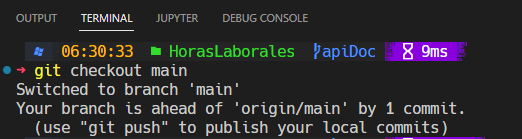
Agregamos los cambios al stage, con el comando git add .



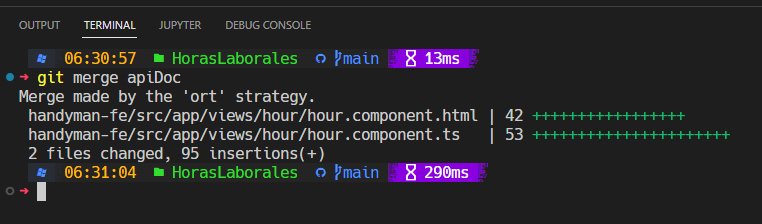
Realizamos el commit con los cambios realizados con el siguiente comando



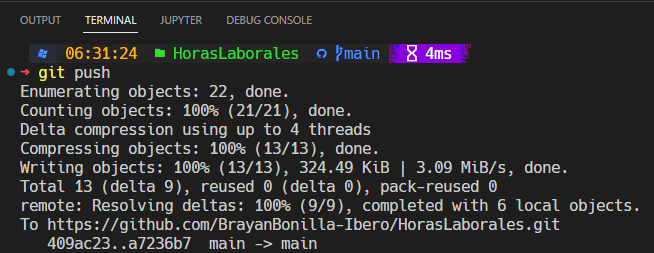
Una vez realizado el commit, procedemos a pasar a la rama principal con el siguiente comando



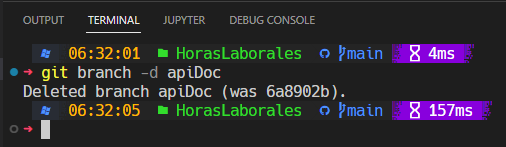
Una vez verificada la solución a la historia de usuario, procedemos a realizar el merge con la rama principal, con el siguiente comando



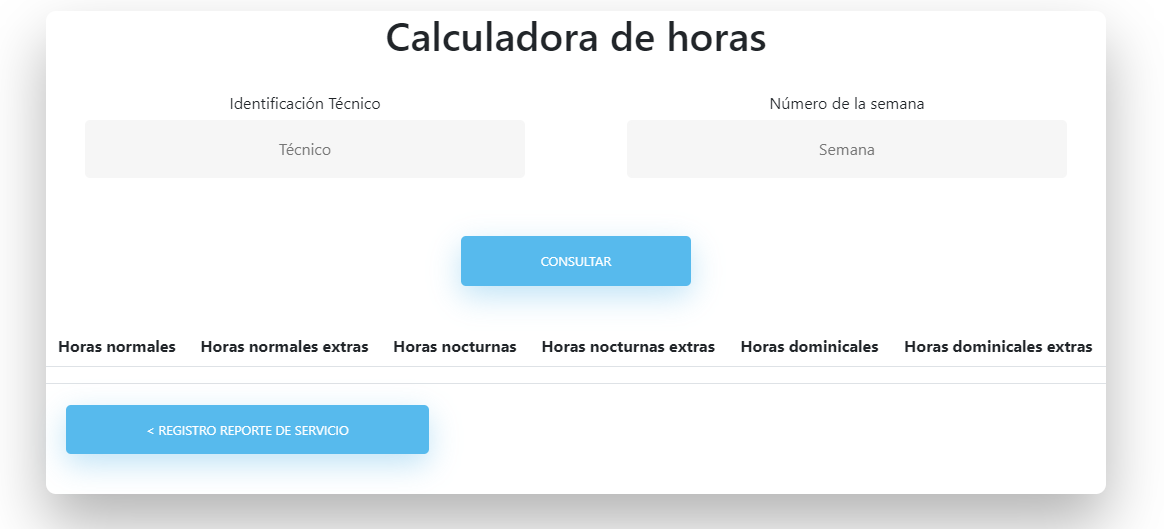
Subimos los cambios incorporados a la rama principal al repositorio remoto, con el siguiente comando



Como ya nos vamos a utilizar más la rama creada para trabajar la HU, procedemos a borrarla con el siguiente comando



El resultado obtenido es el siguiente



# Documentación de comandos GIT usados

GIT es un software de control de versiones muy potente y ampliamente utilizado en la actualidad. De acuerdo con Sánchez Alberca (n.d.) el control de versiones hace referencia a la administración de los diferentes cambios que se realizan sobre los componentes de un sistema de software.

GIT es tan solo uno de los tantos software CVS existentes en la actualidad, según Gómez Bachiller (2015) existen otras alternativas como: Subversion, SuurceSafe, ClearCase, Bazaar, Mercurial, etc.

## git clone

Este es uno de los primeros comandos que empleamos cuando empezamos a trabajar con GIT en proyectos colaborativos, pues nos permite tener una copia en local de un repositorio que se encuentra de manera remota en GitHub. Siempre que se emplea git clone se coloca, luego del comando, la URL del proyecto que siempre termina en .git, como puede verse en el código del proyecto.

## git status

Permite ver el estado actual del proyecto, mostrando los archivos que han sido modificados o agregados al stage área.

## git add .

Este comando agrega los cambios que hemos realizado en uno o varios archivos al staging área, en este lugar se encuentran los datos del proyecto y los cambios que se han realizado sobre estos. Para entender mejor este concepto es importante repasar los tres estados del trabajo en GIT.



Figura 1. Directorio de trabajo, área de almacenamiento y el directorio Git. Fuente: Documentación oficial de GIT, recuperado de: <https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Fundamentos-de-Git>

## git commit -m

El comando GIT commit permite realizar una captura de los cambios que han sido prepagrados y agregados con el comando git add, estas se consideran como “versiones seguras de un proyecto” (Atlassian, n.d.)

## git pull

El comando git pull nos permite traer toda la información desde un proyecto remoto hacia la copia del proyecto que tenemos en local, cuando estamos trabajando en equipo, por ejemplo, podemos hacer un git pull para que nuestro proyecto en local se actualice con los últimos cambios que pueda tener el proyecto en remoto.

## git push

Este comando realiza el trabajo contrario al comando anterior, pues en esta ocasión push nos permite enviar al remoto todos los cambios que tenemos efectuados en el repositorio local.

## git checkout

Este comando nos permite movernos entre ramas, si deseamos, por ejemplo, crear una nueva rama y ubicarnos en ella con un solo comando le podemos agregar el argumento –b, así lo empleamos en este proyecto.

## git merge

Nos permite fusionar ramas, esta es la forma que usa git para volver a unir un historial de trabajo que había sido bifurcado, cuando se crearon nuevas ramas anteriormente para desarrollar funcionalidades específicas, cuando estas funcionalidades han sido probadas y revisadas se pueden sumar a la rama principal a través de merge.

## git branch

El trabajo con ramas es una de las grandes posibilidades que brinda GIT, si bien este no es exclusivo de GIT, pues cualquier sistema de control de versiones actual presenta algún mecanismo para soportar el trabajo con ramas (Chacon & Straub, 2021, p. 58), su uso en GIT es bastante sencillo. Las ramas generalmente se usan para tener, en ciertos momentos del desarrollo, una copia del código del proyecto “aislado del resto del proyecto” para agregar nuevas funcionalidades, las cuales serán integradas posteriormente de nuevo a la rama principal.

## git log

Nos permite visualizar un historial de todos los commit realizados sobre un proyecto en particular. Podemos ver, entonces, cada uno de los commit (con su id único), el autor, la fecha y el mensaje que agregamos.

# Referencias

Atlassian. (n.d.). *Aprende Git con Bitbucket cloud*. https://www.atlassian.com/es/git/tutorials/saving-changes/git-commit

Chacon, S., & Straub, B. (2021). *Pro Git* (Segunda Ed). Apress. file:///C:/Users/jcmon/Downloads/progit.pdf

Gómez Bachiller, S. (2015). *Introducción a GIT y GITHUB*. https://www.uco.es/aulasoftwarelibre/wp-content/uploads/2015/11/git-cosfera-dia-1.pdf

Sánchez Alberca, A. (n.d.). *Introducción a GIT: sistema de control de versiones*. https://aprendeconalf.es/docencia/git/manual/manual-git.pdf