



# Git Hub

Comandos básicos en Git Bash  
& Manejo de GitHub

## Contenido

Creación de llave ssh .....	3
Establecer llave en GitHub .....	4
Descargar repositorio en GitHub .....	6
Ejemplo de trabajo con Git Bash y GitHub .....	9
Acceder a rama: git <b>switch</b> .....	10
Descarga y actualiza: git pull .....	10
Crear rama: git checkout -b "feature/nombre_rama" .....	11
Estado de la rama: git status .....	12
Agregar archivos: git add .....	12
Confirmar cambios localmente: git commit -am .....	12
Mandar cambios al repositorio: git push .....	13
Otros ejemplos .....	13
Verificar ramas: git branch -a .....	13
Descargar ramas faltantes: git fetch .....	14
Realizar un pull request.....	14
Resumen para guardar cambios locales y en el repositorio .....	17
Resumen para pull request .....	17
Resumen de comandos en Git Bash.....	18
Resumen de comandos Git.....	19

## Creación de llave ssh

Si aún no está instalado Git Bash, se puede descargar conforme a las características requeridas del equipo, puede descargarse en el siguiente enlace: <https://git-scm.com/downloads>. Una vez instalado se abre Git Bash y se crea el nombre de usuario con:

```
git config --global user.name
ejemplo:

MINGW64:/c/Users/orozp
orozp@ MINGW64 ~
$ git config --global user.orozpe|
```

Después se ingresa el correo que se está iniciando sesión en GitHub, para crearlo se necesita el siguiente comando:

```
git config --global user.email
ejemplo:

MINGW64:/c/Users/orozp
orozp@ MINGW64 ~
$ git config --global user.orozpe127@gmail.com|
```

Creando el nombre y el correo para el usuario, para comprobar que los datos están ingresados, se puede verificar con:

```
git config --list
ejemplo:

orozp@ MINGW64 ~
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
```

Una vez con el nombre de usuario y el correo establecidos, se crea la llave con el siguiente comando:

```
ssh-keygen  
ejemplo:  
MINGW64:/c/Users/orozp  
orozp@MINGW64 ~  
$ ssh-keygen|
```

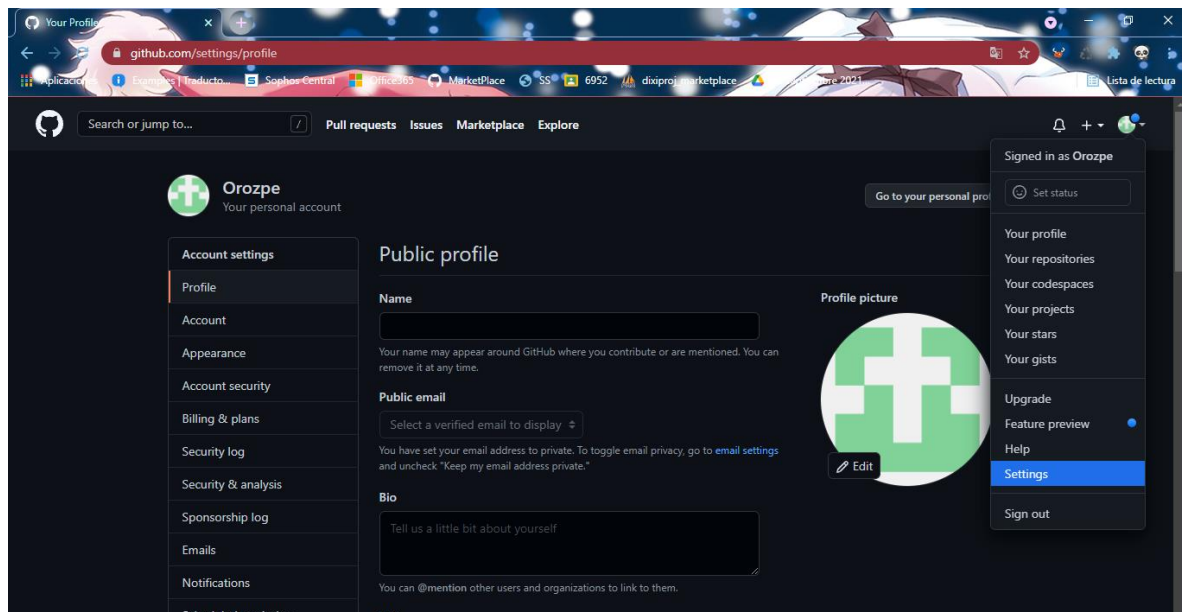
Cuando finalice el proceso, se puede conocer la llave pública creada con:

```
cat ~/.ssh/id_rsa.pub  
ejemplo:  
MINGW64:/c/Users/orozp  
orozp@MINGW64 ~  
$ cat ~/.ssh/id_rsa.pub  
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQgQDznjfg4eekqUuDIxThzdNA6GiN5RnnzreCIYPEyrHrvBPGSZ-
```

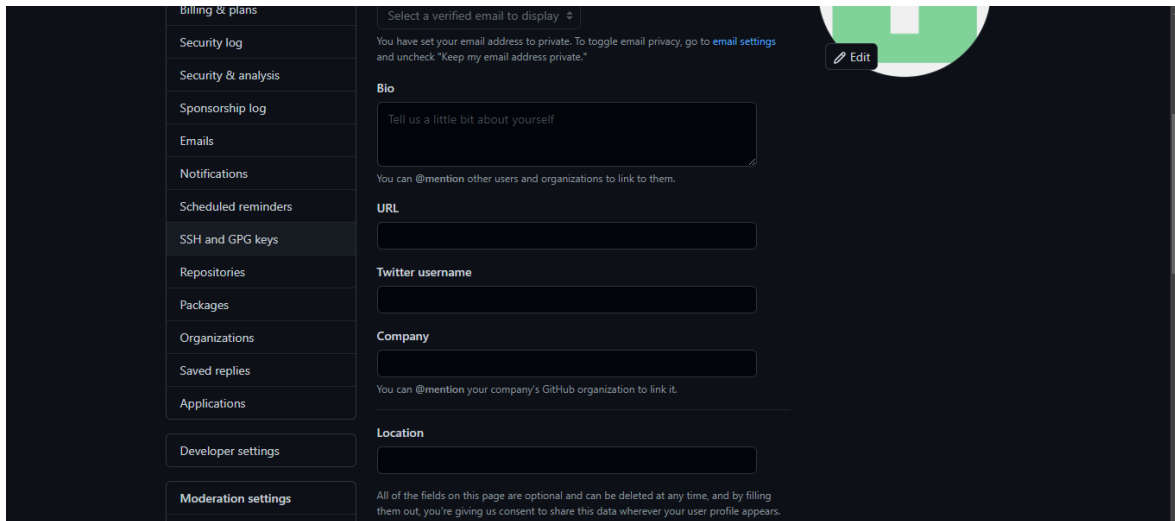
De esta forma todo lo que aparece después del comando ingresado, es la llave ssh creada.

## Establecer llave en GitHub

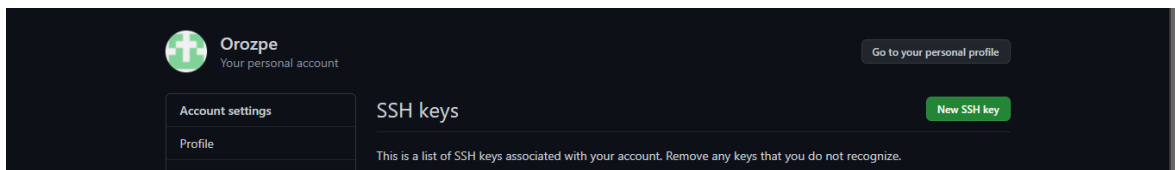
Para tener acceso a los repositorios, y los proyectos en los que se colaboraran de forma remota, lo primero, es establecer la llave creada y agregarla en la cuenta desde GitHub. Para ello al ingresar con la cuenta correctamente, en la sección de perfil, se dará clic en "Settings" o "Ajustes".



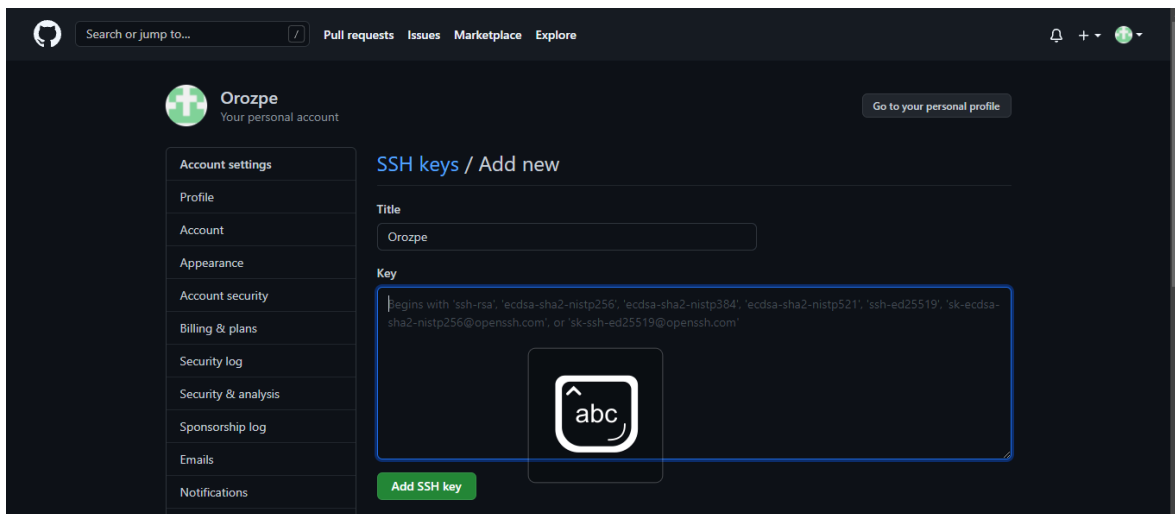
Posteriormente en la parte izquierda se selecciona "SSH and GPG keys"



Se dará clic en "New SSH Key":



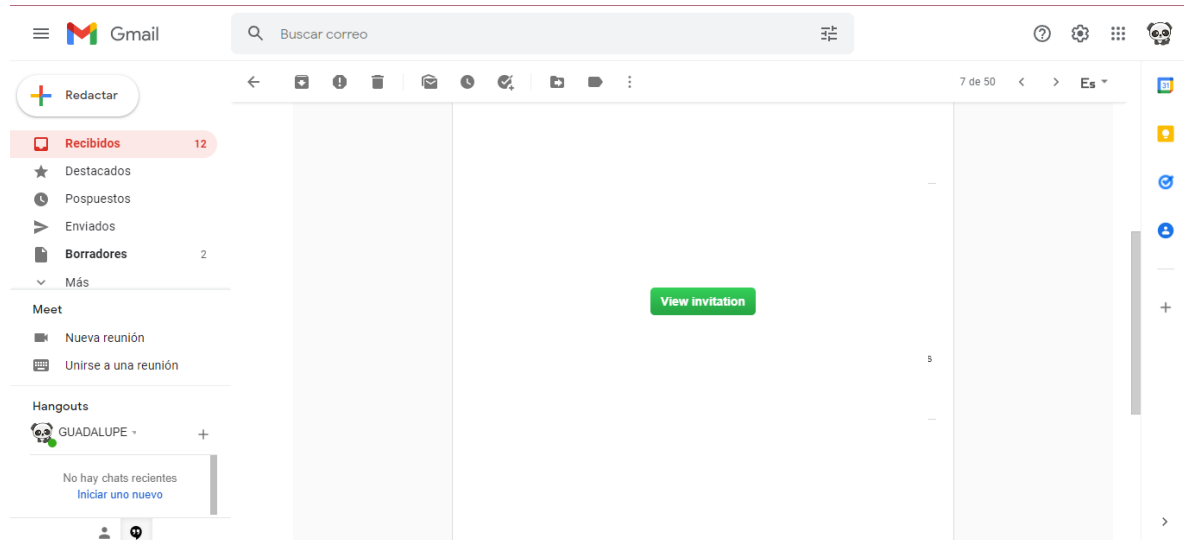
En este apartado se escribe un Título en el que se identifique la llave y en el siguiente apartado se colocará toda la llave ssh creada en Git Bash. Al finalizar estos dos pasos, se selecciona en "Add SSH Key".



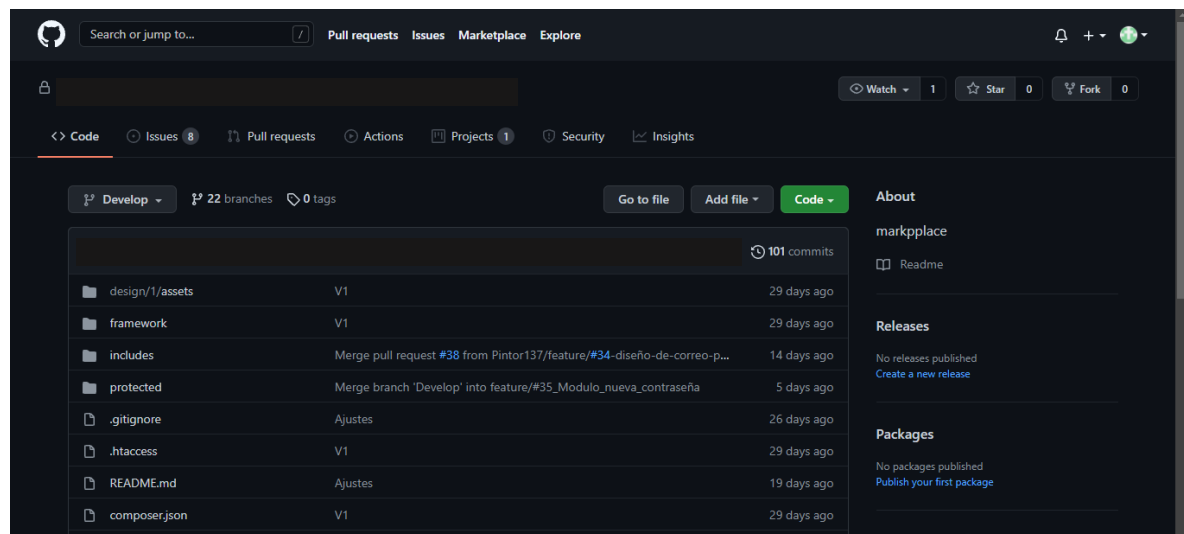
Al concluir se mostrará la llave aceptada y registrada en GitHub.

## Descargar repositorio en GitHub

Para descargar el repositorio, lo primero es tener los permisos para el repositorio, el encargado debe permitir el acceso, de esta forma se recibe una invitación via correo electrónico.



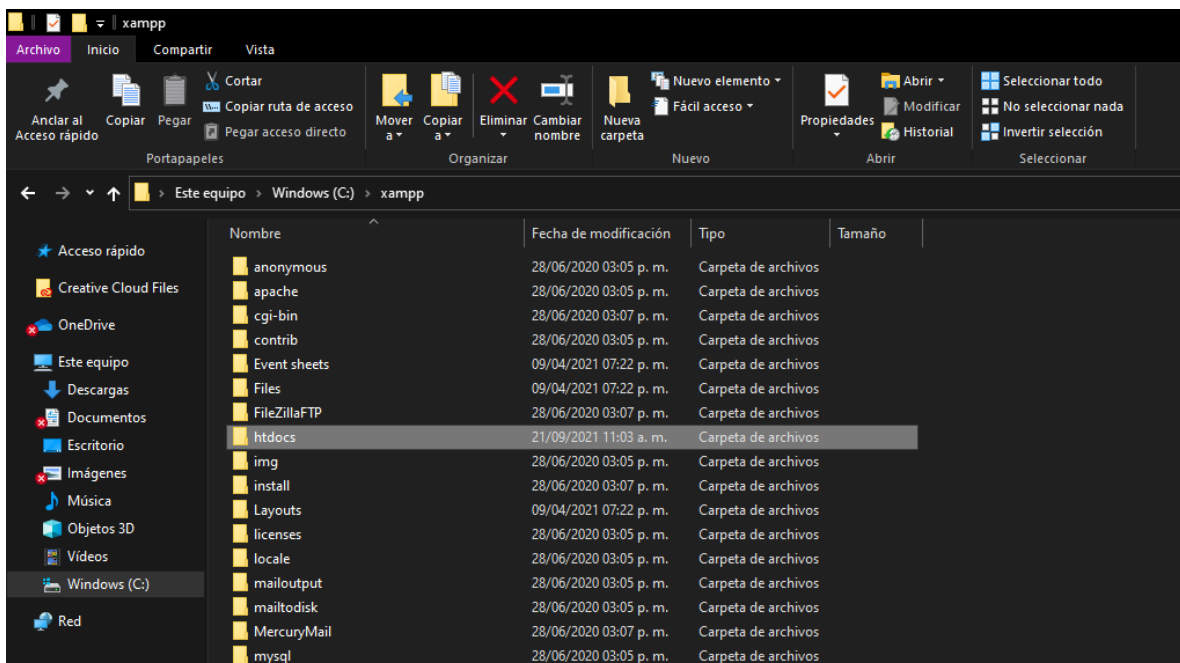
Al dar clic en "view invitation" se podrá observar el repositorio. En esta sección se descargan todos los archivos necesarios, desde el modelo, la base de datos y todos los archivos fundamentales del proyecto.



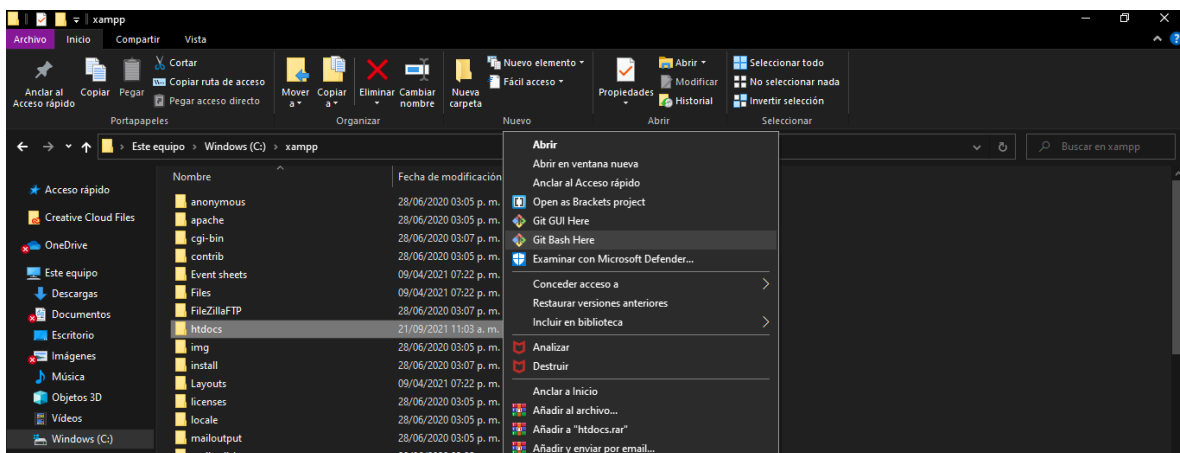
Para descargarlo a través de Git Bash, lo primero es verificar la ruta en donde se almacenarán todos los archivos, en este caso el servidor es XAMPP, en donde todos los proyectos se guardan en la carpeta "htdocs" para que funcionen localmente. Se tiene que abrir Git Bash y acceder con el comando "cd" y la ruta de la siguiente forma:

```
MINGW64/c:/xampp/htdocs
oroazp@MINGW64 ~
$ cd c:/xampp/htdocs
```

Otra forma es buscar la carpeta:



Se selecciona "htdocs", clic derecho y clic en "Git Bash here"



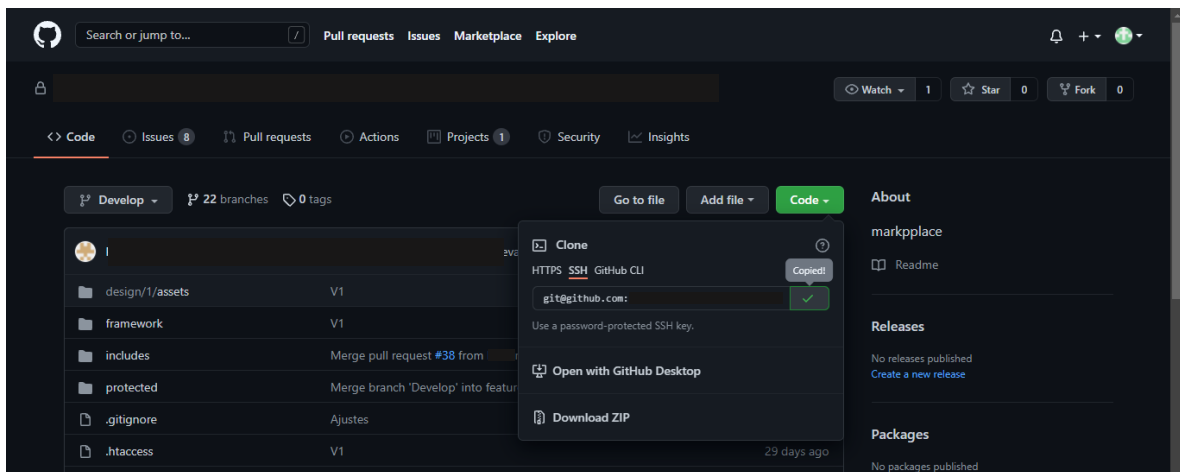


Y así es como se accede a la ruta en donde queremos que se almacenen los archivos que se clonaran o se descargarán:

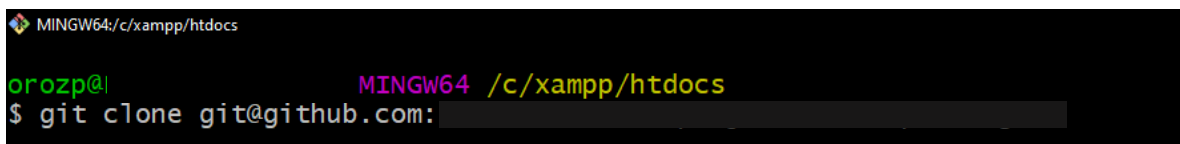


```
MINGW64/c:/xampp/htdocs
orozp@MINGW64 /c:/xampp/htdocs$
```

Una vez que se establezca la ruta de descarga en Git Bash, se vuelve a GitHub, en donde podremos copiar la ruta del repositorio desde el apartado de SSH, se selecciona o simplemente se da clic en el icono que se encuentra en la parte derecha que sirve para copiar:

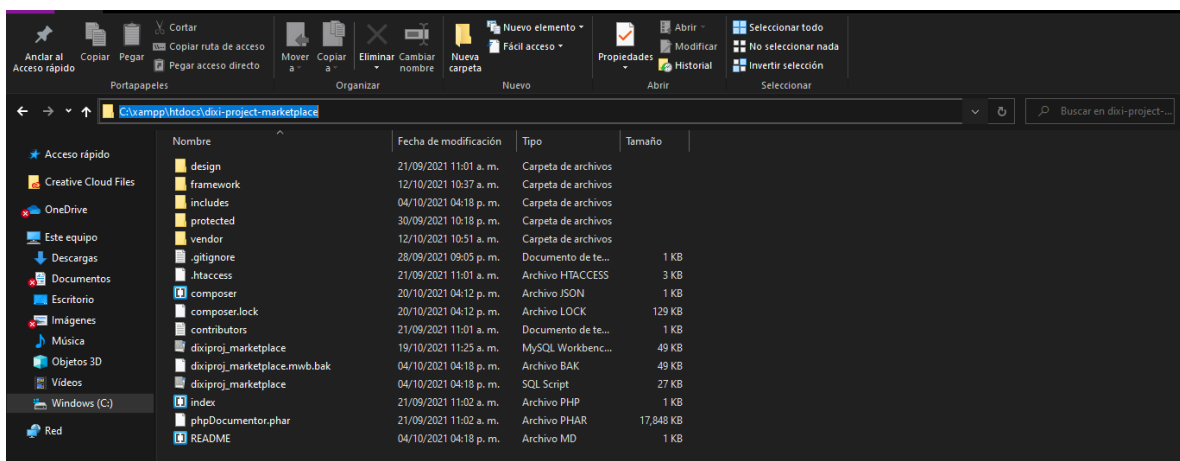


Ahora se deberá ir a Git Bash colocar "git clone" seguido del link copiado:



```
MINGW64/c:/xampp/htdocs
orozp@MINGW64 /c:/xampp/htdocs$ git clone git@github.com:
```

Y verificamos que el repositorio se descargó de forma correcta, desde el explorador de archivos de Windows:

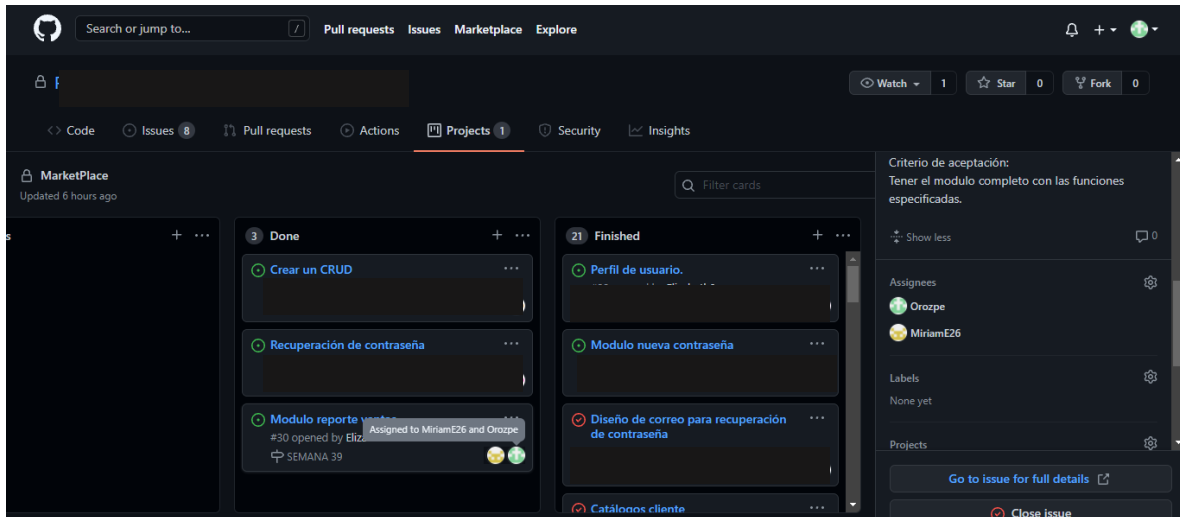




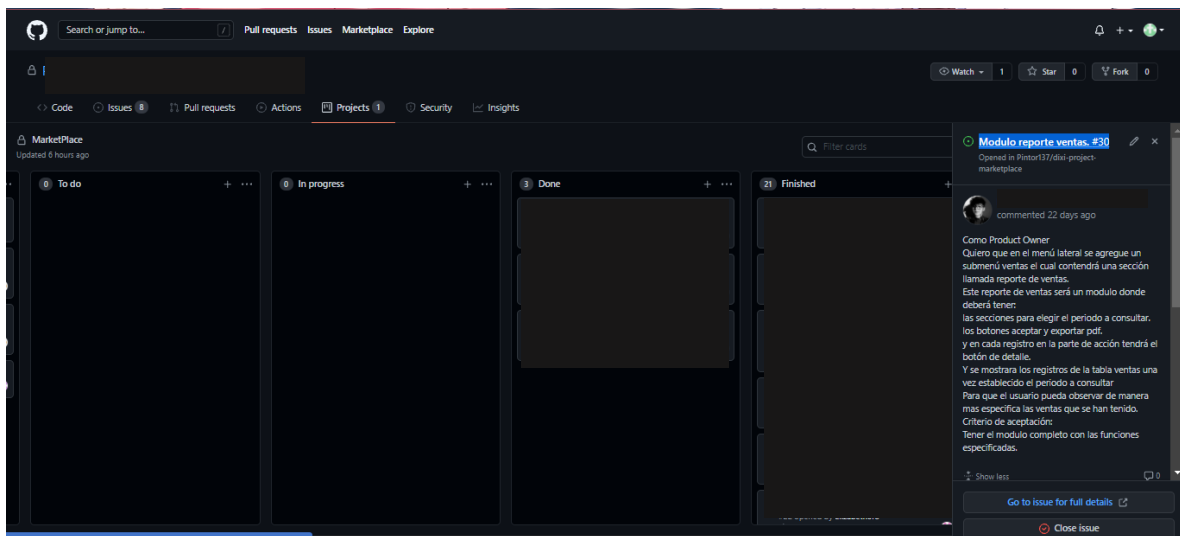
## Ejemplo de trabajo con Git Bash y GitHub

Normalmente se sigue una serie de procesos, sin embargo, estos comandos no tienen un orden en específico, en el siguiente ejemplo se muestra el proceso general en el que se ocupan los comandos y se comienza a trabajar con Git.

Lo primero que se debe verificar es que se tenga asignada una tarjeta creada por el encargado del proyecto:



Al dar clic sobre la tarjeta, se puede observar el número, el título, el nombre del usuario a quien se le asigno la tarea y la descripción de los criterios que se consideran para que sea aprobada por los encargados.



Después se debe ingresar a la ruta del proyecto, ya sea a través del explorador de archivos o con el comando "cd", como se mostró en el apartado "Descargar repositorio en GitHub"

### Acceder a rama: git switch

Al tener la ruta, se debe de asegurar que Develop que es la rama principal del proyecto, siempre este actualizada, por lo que, se accede a develop de la siguiente forma: git switch Nombre\_rama

Como se muestra a continuación, una vez ejecutada la sentencia, se posiciona en la rama correcta:

```
orozp@ /c/xampp/htdocs/dixi-project-marketplace (feature/#30_modulo_reporte_ventas)
$ git switch Develop
Switched to branch 'Develop'
Your branch is up to date with 'origin/Develop'.
```

*Nota: Nunca se deben de modificar o comenzar a trabajar desde la rama principal, al comenzar a trabajar, se debe crear una rama y ahí trabajar y modificar archivos, una vez que se aprueben, serán subidos los cambios a Develop por el encargado.*

### Descarga y actualiza: git pull

Y siempre se debe de verificar que Develop este actualizado, por lo que este comando "git pull", si hay archivos faltantes los descarga y mantiene actualizado el proyecto. Solo se ejecuta, se coloca la contraseña que se estableció cuando se registró el nombre y correo para manejar Git Bash:

```
orozp@ MINGW64 /c/xampp/htdocs/dixi-project-marketplace (Develop)
$ git pull
Enter passphrase for key '/c/Users/orozp/.ssh/id_rsa':
Updating b9f7936..83de0d5
Fast-forward
 includes/img/productos/marca/88774.jpg | Bin 0 -> 799600 bytes
 includes/img/productos/marca/descarga (1).jpg | Bin 0 -> 49027 bytes
 includes/img/productos/originales/88774.jpg | Bin 0 -> 282876 bytes
 includes/img/productos/originales/descarga (1).jpg | Bin 0 -> 13266 bytes
 includes/structure/structure_ask.str | 1 +
 includes/structure/structure_caracteristica.str | 1 +
 .../structure_caracteristica_has_product.str | 1 +
 includes/structure/structure_category.str | 1 +
 includes/structure/structure_client.str | 1 +
 .../structure/structure_client_has_red_social.str | 1 +
 includes/structure/structure_comentario.str | 1 +
 includes/structure/structure_descuento.str | 1 +
 .../structure/structure_descuento_producto.str | 1 +
 includes/structure/structure_detalle_venta.str | 1 +
 includes/structure/structure_favoritos.str | 1 +
 includes/structure/structure_generales.str | 1 +
 includes/structure/structure_generales_cliente.str | 1 +
```

## Crear rama: git checkout -b "feature/nombre\_rama"

Una vez actualizado la rama Develop, se crea la rama con el número de la tarjeta que se asignó en GitHub, seguido del nombre de la rama:

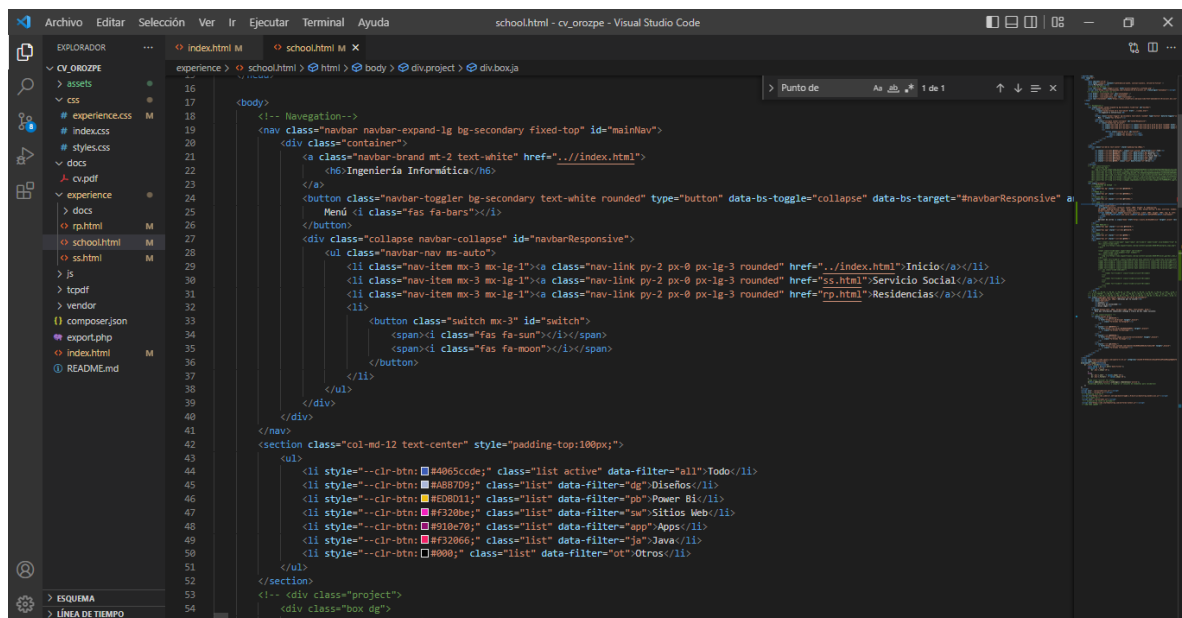
```
orozp@ MINGW64 /c:/xampp/htdocs/cv_orozpe (gh-pages)
$ git checkout -b "feature/#1Prueba_uno"
Switched to a new branch 'feature/#1Prueba_uno'

orozp@ MINGW64 /c:/xampp/htdocs/cv_orozpe (feature/#1Prueba_uno)
$ |
```

*Nota: Siempre se le asigna el nombre entre comillas comenzando con "feature/" seguido de "númeroTarjeta\_nombre\_tarjeta"*

*Es importante que no se dejen espacios en el nombre, por lo que se pueden utilizar guiones medios o guiones bajos.*

Al crearse, se puede comenzar a trabajar dentro de la rama, para poder asegurarse que se está trabajando en la rama creada, se puede visualizar en Visual Studio en el parte inferior izquierdo.



Hasta este momento se comienza a realizar las actividades definidas en la tarjeta o "issues".

## Estado de la rama: git status

Cuando se finalicen los cambios, antes de guardarlos localmente, es recomendable verificar todas las acciones que se realizaron en el proyecto, ya sea los archivos que se agregaron, los archivos que se modificaron o los que fueron eliminados:

```
orozp@ MINGW64 /c/xampp/htdocs/cv_orozpe (gh-pages)
$ git status
On branch gh-pages
Your branch is up to date with 'origin/gh-pages'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   css/experience.css
        modified:   experience/rp.html
        modified:   experience/school.html
        modified:   experience/ss.html
        modified:   index.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        assets/img/references/

no changes added to commit (use "git add" and/or "git commit -a")
```

## Agregar archivos: git add

Cuando se detecten que los archivos que se modificaron, que fueron agregados o los que fueron eliminados son los correctos, con el comando "git add ." donde el punto que se agrega es para que todos los archivos que se modificaron dentro del proyecto se suban. Se ejecuta de la siguiente forma:

```
orozp@ MINGW64 /c/xampp/htdocs/cv_orozpe (feature/#1Prueba_uno)
$ git add .
```

*Nota: Si solo se quisiera agregar un solo archivo que se estructura de la siguiente forma "git add nombre\_archivo.extensión" ejemplo: "git add ventasdetalle.php".*

## Confirmar cambios localmente: git commit -am

El comando "git commit" se utiliza para verificar y confirmar todas las acciones que se realizaron localmente, el "-am" se refiere a que además de confirmar agregue los archivos con la "a" se especifica que la acción solo incluya las modificaciones a los archivos con seguimiento (los que se han añadido con git add en algún punto del historial) y la "m" es para agregar un mensaje de confirmación:

```
orozp@ MINGW64 /c/xampp/htdocs/cv_orozpe (gh-pages)
$ git commit -am "Correcciones finales"
[gh-pages 82ad3c0] Correcciones finales
8 files changed, 64 insertions(+), 5 deletions(-)
create mode 100644 assets/img/references/anaconda.jpg
create mode 100644 assets/img/references/dixi.jpg
create mode 100644 assets/img/references/tescha.jpg
```

## Mandar cambios al repositorio: git push

Para subir los commits desde la rama local al repositorio remoto, se puede ejecutar "git push", al ingresar el comando pedirá la contraseña que se estableció en Git Bash y si todo es correcto, subirá los cambios al repositorio:

```
orozp@MINGW64 /c/xampp/htdocs/cv_oroze (gh-pages)
$ git push
Enter passphrase for key '/c/Users/orozp/.ssh/id_rsa':
Enumerating objects: 25, done.
Counting objects: 100% (25/25), done.
Delta compression using up to 4 threads
Compressing objects: 100% (15/15), done.
Writing objects: 100% (15/15), 30.71 KiB | 1.10 MiB/s, done.
Total 15 (delta 8), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (8/8). completed with 8 local objects.
```

## Otros ejemplos

Cuando se trabaja de forma remota, comúnmente se requiere acceder a una rama que no se ha creado en el local, para ello lo primero es verificar que se tenga en la rama en la que se está trabajando.

## Verificar ramas: git branch -a

Para verificar todas las ramas que hay en el local se ingresa el comando de la siguiente forma:

```
orozp@
$ git branch -a
  Develop
  feature/#15-Crear-crud-cursos-optimizado
  feature/#2-crear-un-crud
  feature/#29-Modulo-para-crear-producto
* feature/#30_modulo_reporte_ventas
```

La que está en verde es la rama en la que nos encontramos, los que están en blanco, son los que se tienen en el local y finalmente los que están en rojo, son los que faltan.

## Descargar ramas faltantes: git fetch

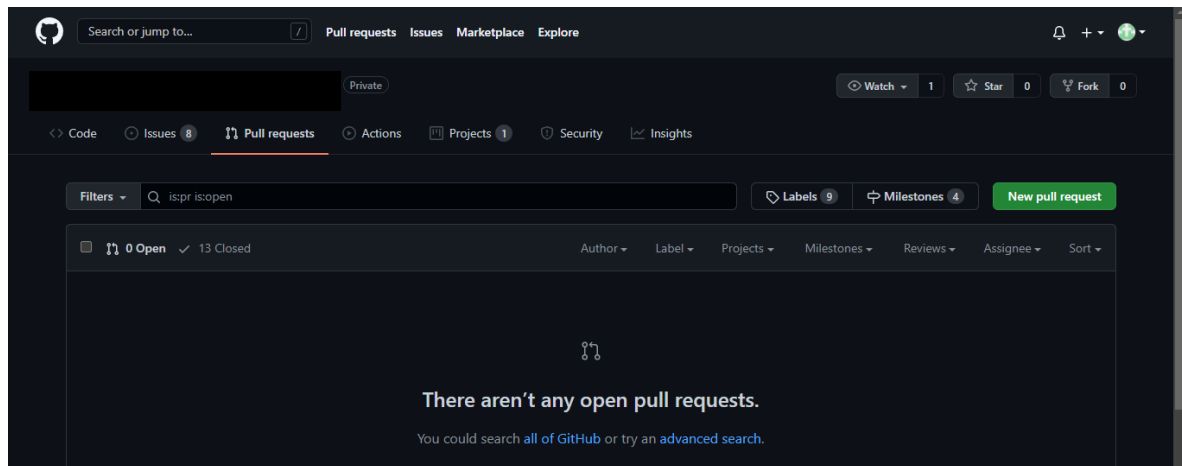
Una forma sencilla es descargar todas las ramas faltantes, para eso se utiliza "git fetch".

```
orozp@
$ git fetch
Enter passphrase for key '/c/Users/orozp/.ssh/id_rsa':
```

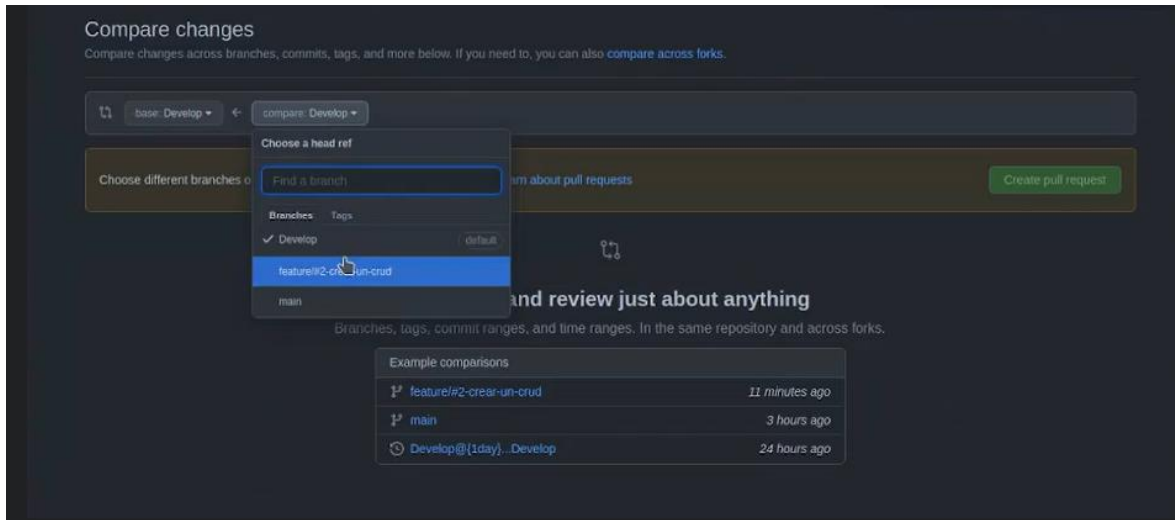
*Nota: Solo se descargarán las ramas que estén aprobadas en el repositorio.*

## Realizar un pull request

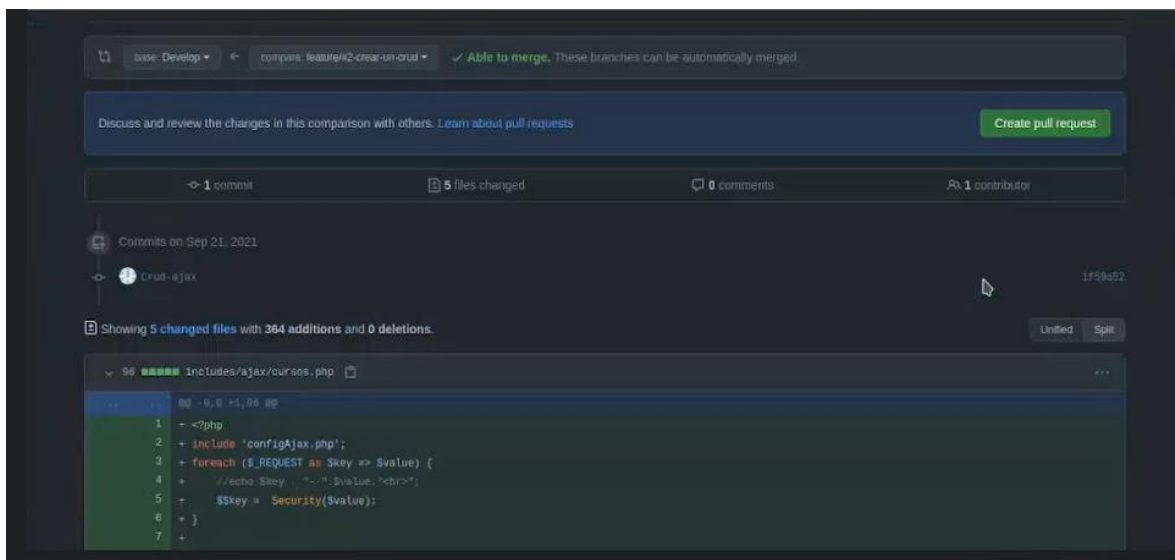
Cuando sea aprobado y autorizado todo lo que se realizó en el modulo que fue asignado, se deberá hacer un "pull request", para unir todos los cambios que se realizan de forma remota e integrarla en la rama principal del proyecto (Develop). Para realizarlo, debe encontrarse dentro del proyecto e ir a la sección de pull request y dar clic en "New pull request".



Cuando se confirma el envío de los cambios desde Git Bash con los comandos para confirmar los cambios en el repositorio, al dar clic muestra el siguiente apartado donde se puede ver la rama en la que se hará el pull request, en este caso, se compara la rama principal (Develop) con la rama en la que se trabajó, en este caso (feature/crear\_un\_crud):

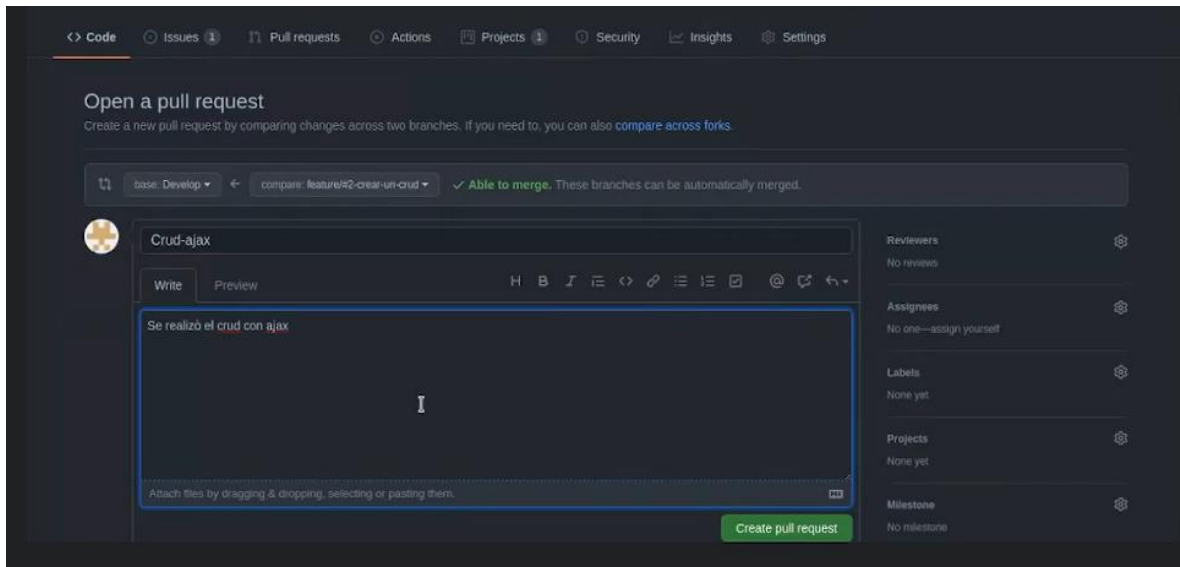


Cuando se selecciona la rama para comparar, se muestran los commits realizados en Git Bash, todo el código que se agregó, se modificó o se eliminó, una vez que se seleccione la rama, se dará clic en "create pull request":

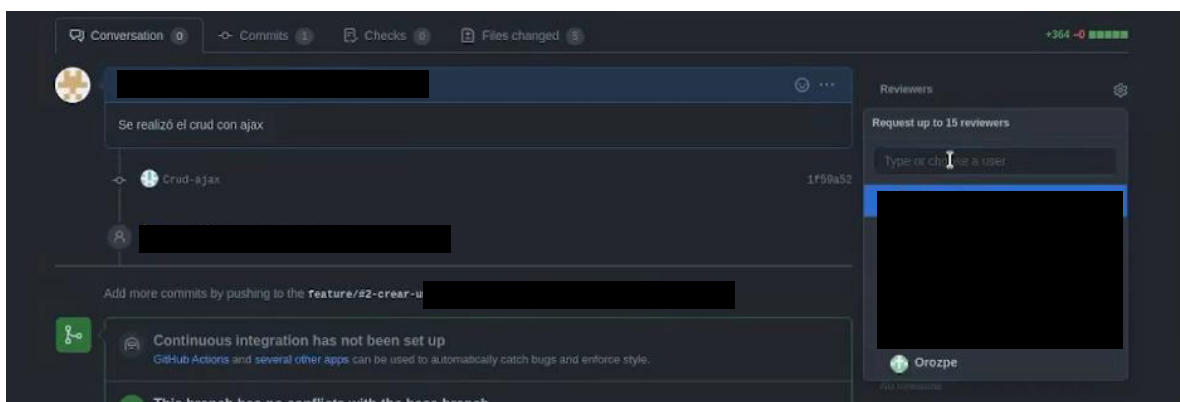




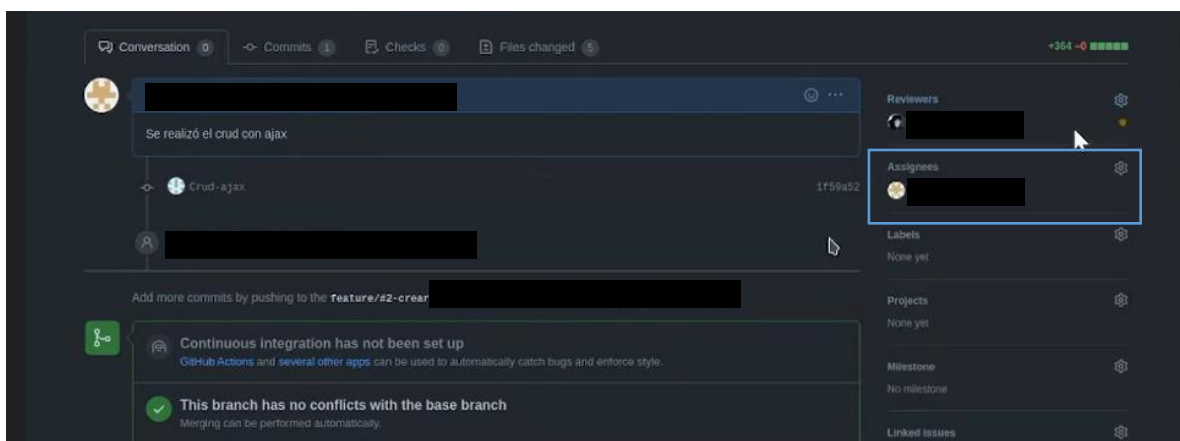
Finalmente se agrega un titulo y un comentario de lo que se realizó:



Es importante tomar en cuenta que antes de crear el pull request, se debe de asignar quien revisara el código, en este caso el encargado del proyecto. Se asigna en la parte derecha, en el apartado "reviewers"



Y también se debe de colocar a quien le fue asignado esa tarea, para ello en el apartado "Assignees" se coloca el usuario de quien realizó las modificaciones en el proyecto. En la siguiente imagen se muestra un ejemplo del usuario:



## Resumen para guardar cambios locales y en el repositorio

La siguiente secuencia de comandos es un ejemplo general, sin embargo, no hay una secuencia fija para guardar los cambios locales y para agregar los archivos en el repositorio remoto.

- git status
- git add .
- git status
- git commit -am " Descripción "
- git push

*Nota: En caso de que se muestre un error al ejecutar git push, solo se requiere copiar todo lo que aparece después de git push. Ejemplo: git push –set-upstream origin "feature/#1-Crear-crud"*

## Resumen para pull request

Los pasos finales para concluir las tareas que se asignen en GitHub son las siguientes:

1. Ir a la sección pull request
2. Clic en "New request"
3. Seleccionar la rama realizada
4. Colocar nombre y descripción de lo que se realizó
5. Colocar quien lo revisa
6. Colocar a quien se le asigne la tarea
7. Clic en "Create pull request"

*Nota: Estos pasos solo deben realizarse cuando se haya cumplido con todos los criterios de la tarea, el encargado lo haya aprobado y que finalmente acepte hacer el pull request sobre la rama en la que se trabajó.*

## Resumen de comandos en Git Bash

En la siguiente tabla se muestra la estructura de los comandos más utilizados.

Comando	Descripción
git branch -a	Verifica todas las ramas que hay en el local
git fetch	Descarga las ramas faltantes
git switch nombre_rama	Para acceder a una rama
git checkout -b "feature/#1_nombre_rama"	Si no existe una rama con ese nombre, se crea y se posiciona en la rama creada
git status	Muestra el estado de la rama en la que esta posicionado
git add .	Agrega todos los archivos que se han modificado en el equipo.
git add 'nombre_archivo'	Se agrega un archivo específico, se coloca el nombre y la extensión del archivo.
git commit -am "Pequeña descripción"	Se guardarán todos los cambios con una pequeña descripción.
git pull	Descarga todos los recursos faltantes y menciona si ya está actualizado o no.
Git push	Sube todos los cambios y commits de la rama. Esto solo se realiza hasta que lo autoricen.

Si se desea conocer más comandos o saber cómo funciona git, se pueden consultar los siguientes enlaces:

- <https://www.atlassian.com/es/git/tutorials>
- <https://www.atlassian.com/es/git/tutorials/saving-changes>
- <https://www.atlassian.com/es/git/tutorials/syncing>

## Resumen de comandos Git

Comando	Descripción
git config --global user.name	Establecer nombre en Git Bash
git config --global user.email	Establecer correo en Git Bash
git config --list	Muestra los detalles de usuario en Git Bash
ssh-keygen	Crea llave pública
cat ~/.ssh/id_rsa.pub	Muestra la llave pública creada
git clone	Se ocupa cuando se quiere clonar un repositorio
pwd	Muestra la ruta en la que está posicionada
mkdir	Crea carpetas
cd	Navega entre carpetas, se puede acceder a una carpeta o ruta específica
touch	Crea un archivo
rm	Elimina un archivo
cat	Para ver el contenido de un archivo
ls	Muestra todos los archivos en donde se encuentra ubicado
git pull	Descarga todos los recursos faltantes y menciona si ya está actualizado o no.
Git push	Sube todos los cambios y commits de la rama. Esto solo se realiza hasta que lo autoricen.