Tutorial comandos Git y GitHub

Autor: Enrique Niebles, @esacon

Configuración inicial de Git Bash:

Para instalar Git Bash (consola de Git) dirigirse al siguiente link: Git for Windows, y seguir los pasos de forma predeterminada. Una vez finalizada la instalación, es necesario configurar el nombre de usuario y el correo con el cual se subirán los cambios.

Nota: Es importante tener en cuenta que el correo introducido, debe coincidir con el correo de su cuenta de GitHub.

- Configurar nombre de usuario:

```
git config --global user.name "nombre_de_usuario"
```

- Configurar correo electrónico:

```
git config --global user.email "correo@ejemplo.com"
```

- Configurar colores de la interfaz:

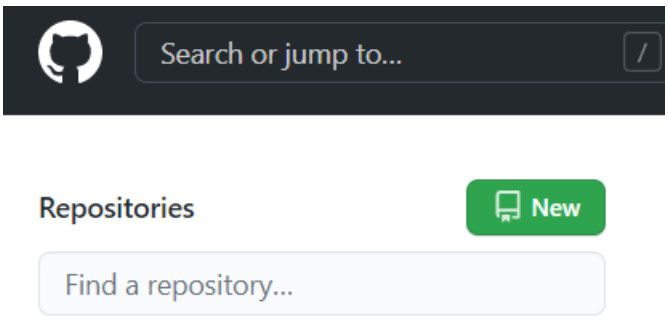
```
git config --global color.ui auto
```

Estos pasos se ejecutan una única vez, y sirven para configurar el usuario que realizará los cambios en el repositorio.

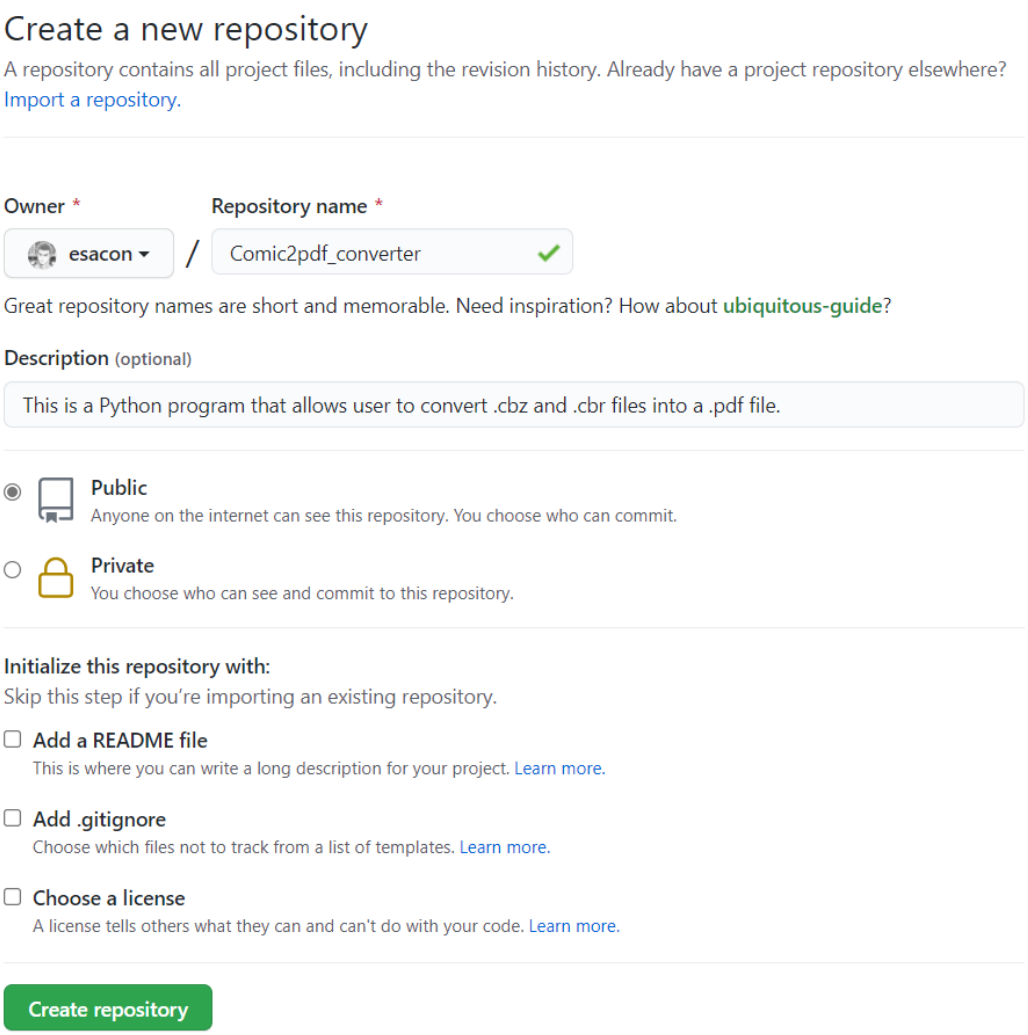
Crear un nuevo repositorio:

Para crear un nuevo repositorio, es necesario estar previamente autenticado en la página principal de GitHub. Una vez iniciada sesión, puede realizar lo siguiente:

1. En la parte superior izquierda del sitio web, escoger el botón verde que dice **"New"**, el cual permitirá crear un nuevo repositorio:



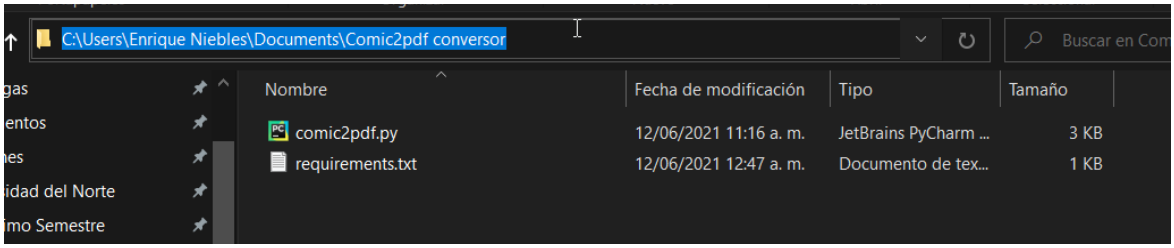
2. Una vez seleccionado el botón, llenar la información correspondiente y hacer clic en el botón **"Create repository"**:



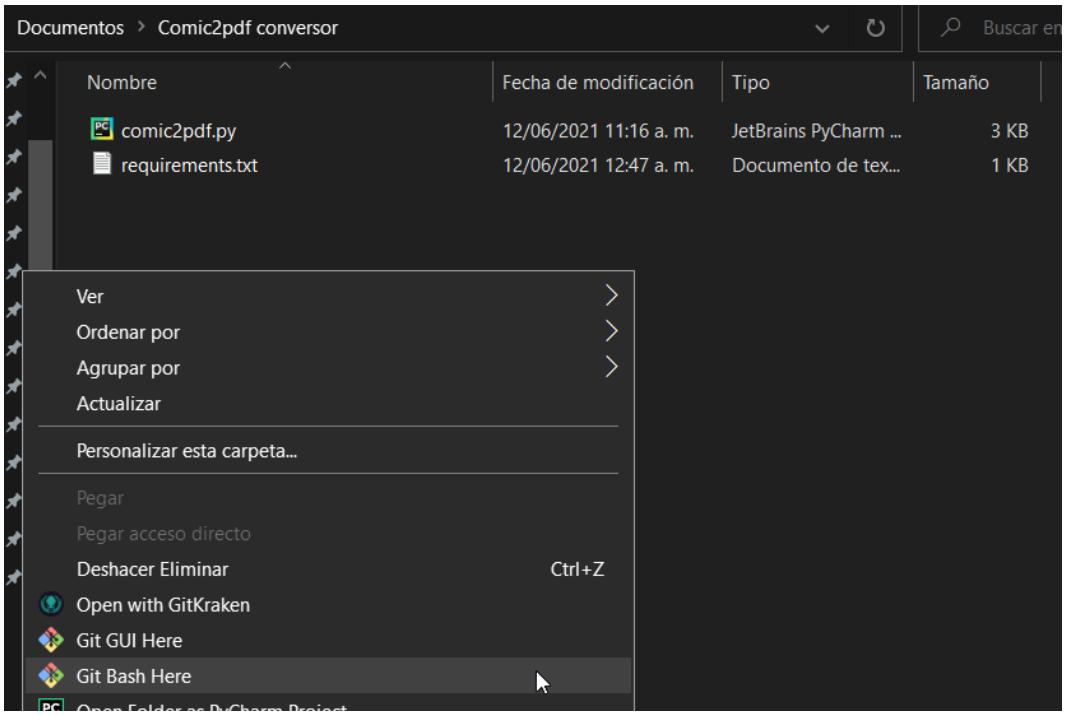
Nota: Un repositorio público es aquel que puede ser accedido por cualquier usuario en internet, sin problema alguno. Mientras que un repositorio privado permite únicamente a los colaboradores del mismo, acceder a la información que este posea.
Al momento de inicializar un repositorio es posible también, crear un archivo README.md (el cual brinda una descripción del proyecto en la venta principal del repositorio), o un archivo .girignore (permite indicarle a Git qué extensiones de archivos no debe subir al repositorio online). En caso de tener una carpeta local ya creada, es recomendable **no** escoger estas dos opciones, pues producirán un conflicto entre los commits, y requiere una solución no tan elemental.

3. Una vez creado el repositorio en GitHub, es necesario hacer una conexión entre el repositorio local (aquel que se encuentre en el disco duro de su computador), y el repositorio online (el que se creó en el paso anterior). Para esto es necesario ubicarse en el repositorio local y abrir la ventana de comandos de Git Bash. Los dos métodos para hacerlo son

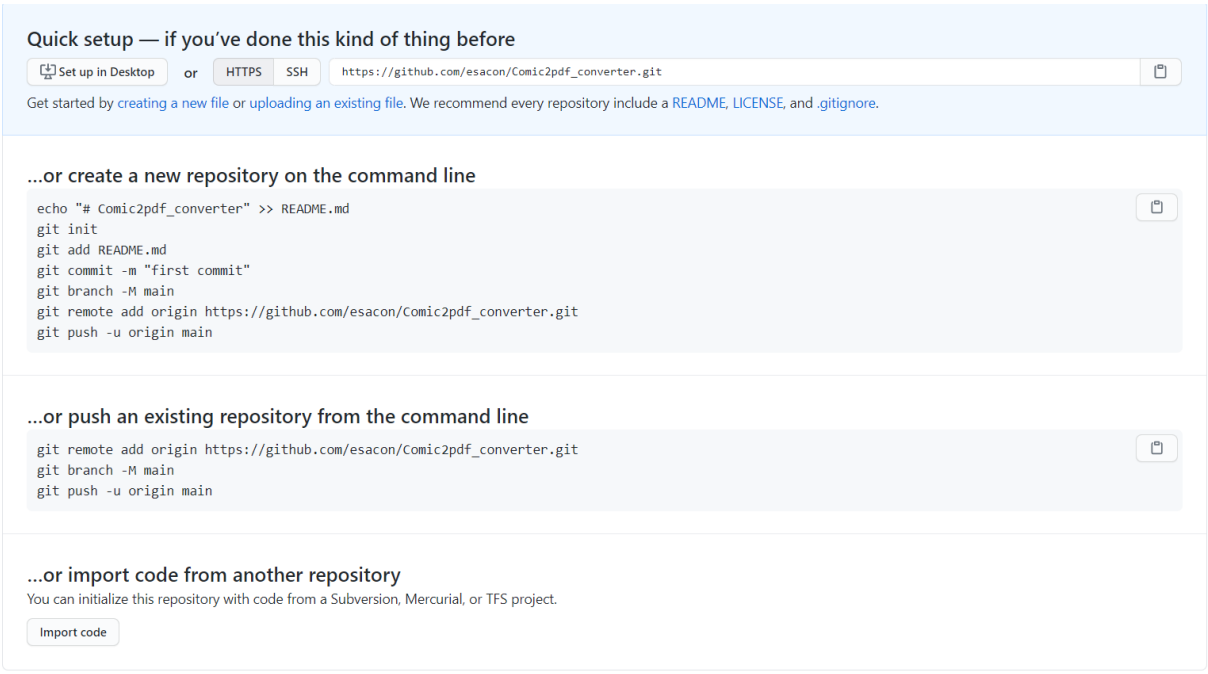
   Método 1: Ubicarse en la barra de direcciones y escribir “Git Bash.lnk”



Método 2: Hacer clic derecho en una parte vacía del explorador y escoger la opción “Git Bash here”



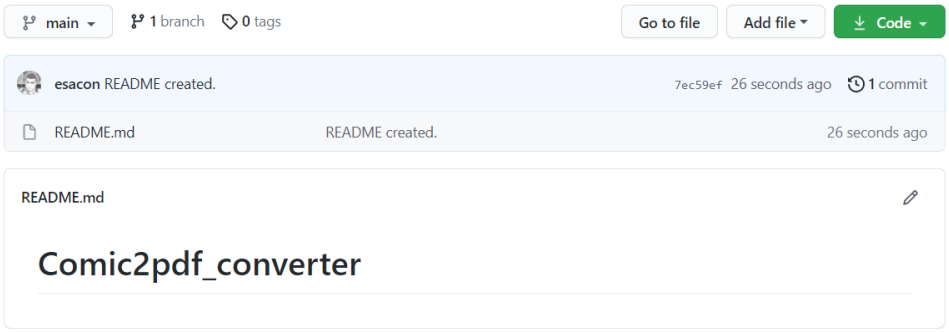
Ambas opciones abren la consola de comandos. Luego debe copiarse el siguiente set de instrucciones, el cual apareció una vez creado el repositorio en GitHub



```
echo "# Comic2pdf_converter" >> README.md
git init
git add README.md
git commit -m "README created."
git branch -M main
git remote add origin
https://github.com/esacon/Comic2pdf_converter.git
git push --set-upstream origin main
```

Nota: Cabe resaltar que la última línea de código recibió una pequeña alteración, por lo cual debe tenerse en cuenta para no tener ningún inconveniente al momento de hacer el push.

4. Una vez completado el paso anterior, el repositorio local y el remoto, se encontrarán perfectamente vinculados. Sin embargo, la información que se encontraba en el repositorio local todavía no ha sido cargada al repositorio remoto.



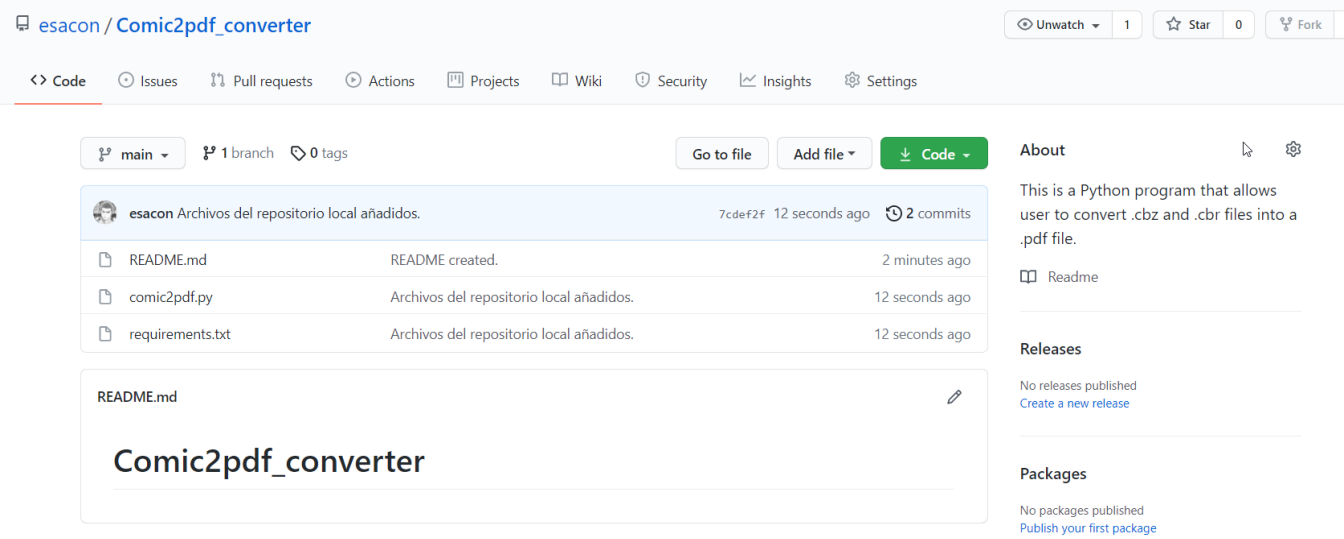
Para añadir los archivos es necesario seguir la siguiente secuencia de pasos:

- Cargar todos los datos de la carpeta. El punto (.) indica que se subirán todos los archivos de la carpeta.
- Comentar el cambio realizado. El -m indica que se introducirá un mensaje entre comillas dobles "…"
- Empujar los datos al servidor (push).

```
git add .
git commit -m "Archivos del repositorio local añadidos."
git push
```

Nota: Una vez conectado el repositorio local con el remoto, es necesario introducir los 3 comandos anteriormente mencionados, cada vez que se realice un cambio en el repositorio local. De lo contrario, los cambios no se verán reflejados en el servidor.

5. Finalmente, se habrán conectados los repositorios entre sí. Junto con los datos del mismo.



6. **Aclaración importante:**
   Al trabajar con un repositorio remoto, el cual puede ser modificado por un colaborador externo, es necesario verificar primero si este ha realizado un cambio o no. En caso afirmativo, debe primero bajarse (pull) la última versión del proyecto, para poder subir los cambios.
   Para esto existe el comando pull

```
git pull
```

Clonar un repositorio de GitHub existente:

En caso de contar con un repositorio ya existente, y querer descargarlo en local, deben seguirse los siguientes pasos:

1. Ubicar la carpeta local donde se guardará la información.
2. Abrir el Git Bash en la carpeta. Puede escoger cualquiera de los dos métodos presentados en la sección anterior.
3. Buscar y copiar el link del repositorio de GitHub que se desea descargar.
4. Clonar el repositorio.

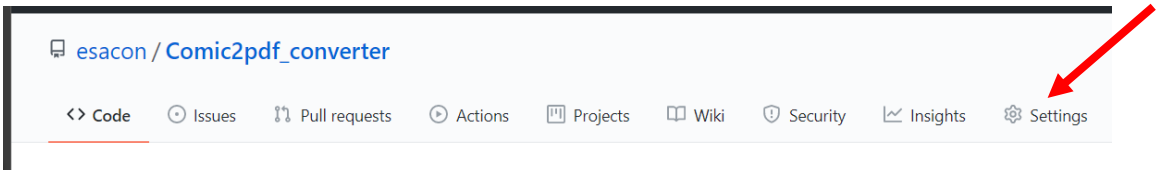
```
git clone <url_del_repositorio>
```

Ejemplo: `git clone https://github.com/esacon/Comic2pdf_converter`

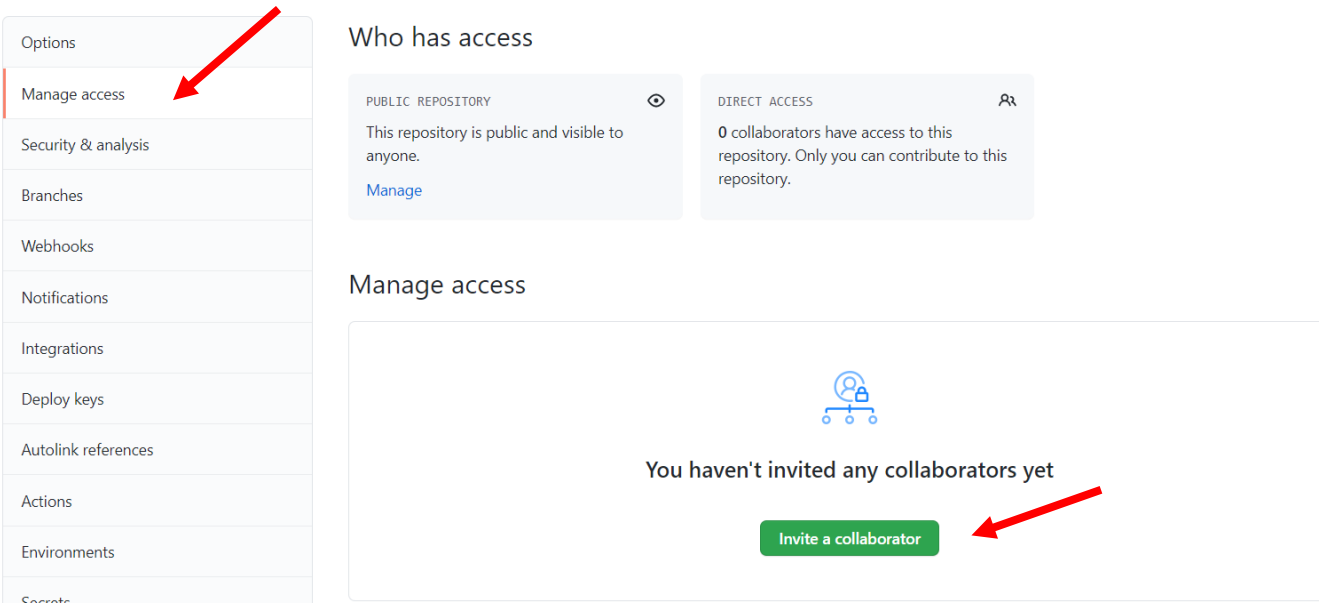
Añadir colaboradores en un repositorio privado:

Si desea añadir colabores a un repositorio privado ya creado:

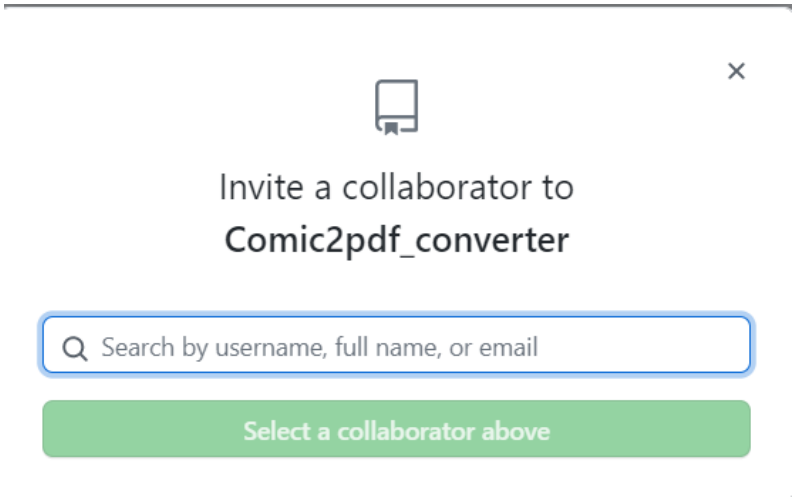
1. Ubicar en la pestaña de opciones, la opción Settings o Configuraciones:



2. Seleccionar la opción “Manage Access” o “Manejar acceso” y hacer clic en la opción **“Invite a collaborator”.**



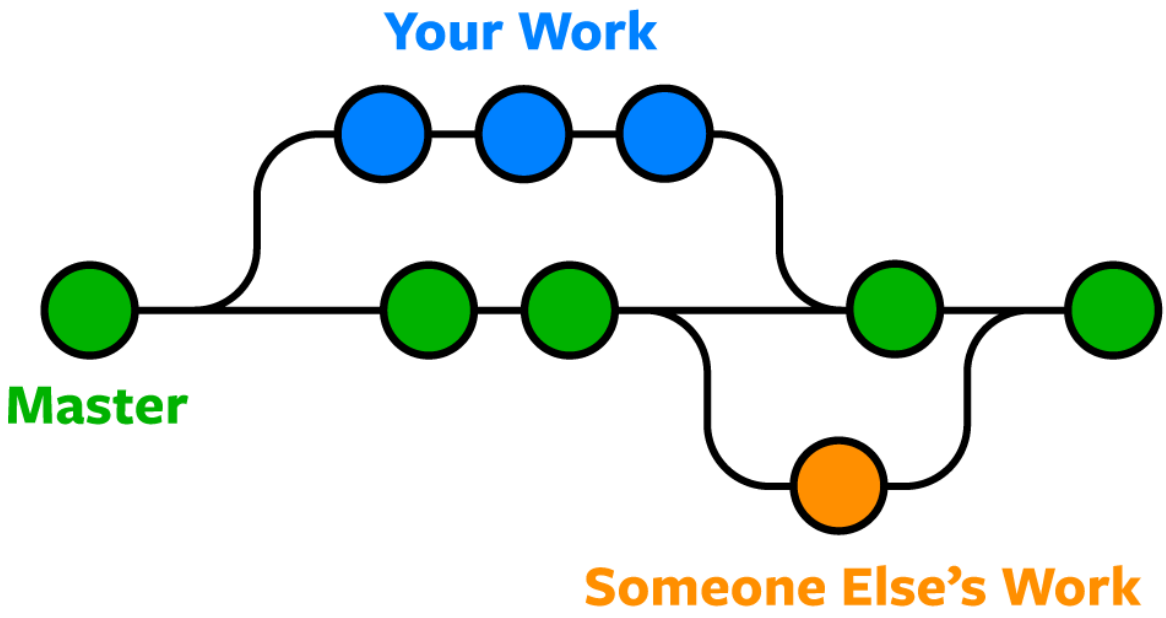
3. Buscar el usuario que se desea añadir, ya sea por su nombre de usuario en GitHub o por su correo electrónico con el cual se encuentra registrado:



4. Una vez el colaborador acepte la invitación que le llegará al correo electrónico, podrá acceder y modificar el repositorio privado.

Manejo de ramas en git:

Las ramas son una herramienta muy útil porque permiten manejar cambios en un proyecto, sin afectar la rama principal.



Para **crear** una rama, y hacer el **cambio** enseguida, se utiliza el siguiente comando:

```
git checkout -b nombre_rama
```

Una vez en la rama, se sigue trabajando de manera normal, sin preocupación alguna. Y así, cuando se complete el trabajo, se **fusionan** los cambios realizados con la rama principal. De la siguiente forma

```
git merge nombre_rama -m "Esto es un merge con mensaje"
```

Y se **suben** los cambios al repositorio remoto

```
git push -u origin nombre_rama
```

Si desea **eliminar** la rama en la cual estaba realizando los cambios

```
git branch -d nombre_rama
```

Para **cambiar** entre ramas (ya sea del master a la rama alterna, o viceversa):

```
git checkout nombre_de_la_rama
```

Para **crear** una rama, sin cambiar de inmediato a esta:

```
git branch nombre_rama
```

Enlaces de referencia con más información:

- Manejo de ramas con git: Trabajar con ramas en Git: git branch (desarrolloweb.com)
- Setup git push: How To Set Upstream Branch on Git – devconnected
- Comandos básicos de git: github-git-cheat-sheet

Comandos en detalle:

git add

Añadimos todos los archivos para el commit

```
git add .
```

Añadimos el archivo para el commit

```
git add <archivo>
```

Añadimos todos los archivos para el commit omitiendo los nuevos

```
git add --all
```

Añadimos todos los archivos con la extensión especificada

```
git add *.txt
```

Añadimos todos los archivos dentro de un directorio y de una extensión especifica

```
git add docs/*.txt
```

Añadimos todos los archivos dentro de un directorio

```
git add docs/
```

git commit

Cargar en el HEAD los cambios realizados

```
git commit -m "Texto que identifique por que se hizo el commit"
```

Agregar y Cargar en el HEAD los cambios realizados

```
git commit -a -m "Texto que identifique por que se hizo el commit"
```

De haber conflictos los muestra

```
git commit -a
```

Agregar al ultimo commit, este no se muestra como un nuevo commit en los logs. Se puede especificar un nuevo mensaje

```
git commit --amend -m "Texto que identifique por que se hizo el commit"
```

git push

Subimos al repositorio

```
git push <origien> <branch>
```

Subimos un tag

```
git push --tags
```

git log
Muestra los logs de los commits

```
git log
```

Muestras los cambios en los commits

```
git log --oneline --stat
```

Muestra graficos de los commits

```
git log --oneline --graph
```

git diff
Muestra los cambios realizados a un archivo

```
git diff
git diff --staged
```

git head
Saca un archivo del commit

```
git reset HEAD <archivo>
```

Devuelve el ultimo commit que se hizo y pone los cambios en staging

```
git reset --soft HEAD^
```

Devuelve el ultimo commit y todos los cambios

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

Devuelve los 2 ultimo commit y todos los cambios

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

Rollback merge/commit

```
git log
git reset --hard <commit_sha>
```

git remote
Agregar repositorio remoto

```
git remote add origin <url>
```

Cambiar de remoto

```
git remote set-url origin <url>
```

Remover repositorio

```
git remote rm <name/origin>
```

Muestra lista repositorios

```
git remote -v
```

Muestra los branches remotos

```
git remote show origin
```

Limpiar todos los branches eliminados

```
git remote prune origin
```

git branch

Crea un branch

```
git branch <nameBranch>
```

Lista los branches

```
git branch
```

Comando -d elimina el branch y lo une al master

```
git branch -d <nameBranch>
```

Elimina sin preguntar

```
git branch -D <nameBranch>
```

git tag

Muestra una lista de todos los tags

```
git tag
```

Crea un nuevo tags

```
git tag -a <version> - m "esta es la versión x"
```

git rebase

Los rebase se usan cuando trabajamos con branches esto hace que los branches se pongan al día con el master sin afectar al mismo

Une el branch actual con el mastar, esto no se puede ver como un merge

```
git rebase
```

Cuando se produce un conflicto no das las siguientes opciones:
cuando resolvemos los conflictos --continue continua la secuencia del rebase donde se pauso

```
git rebase --continue
```

Omite el conflicto y sigue su camino

```
git rebase --skip
```

Devuelve todo al principio del rebase

```
git reabse --abort
```

Para hacer un rebase a un branch en especifico

```
git rebase <nameBranch>
```

git diff

El comando Diff se usa para visualizar las diferencias entre objetos (ramas, commits, archivos, entre otros)

Muestra los archivos que no se encuentran en el repositorio en la rama actual

```
git diff
```

Muestra las diferencias entre commits

```
git diff 1234abc 6789def    #antiguo   nuevo
```

Muestra las diferencias de archivos de la "staged area"

```
git diff –staged
```

Muestra las diferencias entre ramas

```
git diff rama1 rama2
```

Muestra las diferencias en un archivo especifico

```
git diff miarchivo.txt
```

Muestra las archivos con cambios en un directorio

```
git diff documentation
```

git fork

Descargar remote de un fork

```
git remote add upstream <url>
```

Merge con master de un fork

```
git fetch upstream
git merge upstream/master
```

Otros Comandos

Lista un estado actual del repositorio con lista de archivos modificados o agregados

```
git status
```

Quita del HEAD un archivo y le pone el estado de no trabajado

```
git checkout -- <file>
```

Crea un branch en base a uno online

```
git checkout -b newlocalbranchname origin/branch-name
```

Busca los cambios nuevos y actualiza el repositorio

```
git pull origin <nameBranch>
```

Cambiar de branch

```
git checkout <nameBranch/tagname>
```

Une el branch actual con el especificado

```
git merge <nameBranch>
```

Verifica cambios en el repositorio online con el local

```
git fetch
```

Borrar un archivo del repositorio

```
git rm <archivo>
```