# El comando git status

El comando git status nos permite ver el estado de nuestro **espacio de trabajo** y del **repositorio**: los nuevos archivos que hemos creado, modificado o eliminado, los cambios que se van a incluir en el siguiente commit, etc.

Cuando no hay nuevos cambios en el espacio de trabajo vas a ver algo similar a lo siguiente al ejecutar git status:

$ git status

On branch master

nothing to comit, working tree clean

Cuando hay cambios, el comando git status puede mostrar hasta 3 secciones:

1. Cambios que se van a incluir en el siguiente commit (Changes to be commited). A esto también se le conoce como el **index**.
2. Cambios que no están en el índice (Changes not staged for commit). Estos cambios no serán incluidos en el siguiente commit.
3. Nuevos archivos (Untracked files).

**Agregando cambios al índice**

Para agregar los cambios de uno o varios archivos al índice utiliza el comando git add seguido de la lista de archivos o de punto (.) para agregar todos los cambios pendientes. Por ejemplo, el siguiente comando agregaría al **índice** los cambios en index.html

git add index.html

Para agregar todos los cambios pendientes utiliza el siguiente comando:

git add .

**Recuerda:** Sólo los archivos que estén en el **índice** serán incluidos en el siguiente commit.

**Removiendo cambios del índice**

Para remover archivos del **índice** utiliza el comando git reset HEAD seguido de uno o más archivos. Por ejemplo, asumiendo que tienes un archivo index en el índice y lo quieres remover, utiliza el siguiente comando:

git reset HEAD index.html

**Explorando los cambios**

Para ver los cambios que aún no están en el **índice** utiliza el siguiente comando:

git diff

Para ver los cambios que están en el **índice** utiliza el siguiente comando:

git diff --staged

**Descartando cambios**

Si quieres descartar los cambios de un archivo completamente, es decir, volver a como estaba en el último commit, utiliza el comando git checkout -- seguido del nombre del archivo:

git checkout -- index.html

Para que este comando funcione, el archivo se debe encontrar fuera del **índice**. Si el archivo está en el **índice** debes removerlo como vimos previamente antes ejecutar este comando.

**Ver un commit específico**

Para ver un commit específico utiliza git show seguido del identificador del commit (puedes utilizar los primeros 7 caracteres). Este comando, además de la información del commit, muestra la lista de cambios.

$ git show fc3b561

commit fc3b5612f0e23158263ad1a47d91d6c8f84f1a1f

Author: Pedro Perez <pedro@gmail.com>

Date: Sun Jan 19 14:43:21 2020 -0500

Crea el landing principal

diff --git a/index.html b/index.html

index 1196288..fe8ec0c 100644

--- a/index.html

+++ b/index.html

@@ -1,2 +1,2 @@

Esta línea ya estaba

+Esta línea se agregó

-Esta línea se eliminó

Cada cambio empieza con la línea diff --git. En este ejemplo sólo tenemos un cambio (el archivo index.html). En las últimas líneas vemos qué cambio: una línea se agregó (la que empieza con +) y otra se eliminó (la que empieza con -). Generalmente en la consola aparecen las nuevas líneas en verde y las que se eliminaron en rojo.

**Recursos adicionales**

* [Guías de Make it Real - Conceptos y comandos escenciales de Git](https://guias.makeitreal.camp/git/conceptos-comandos-esenciales)

# Trabajando con ramas

Las ramas (branches) nos permiten desviarnos de la línea principal de desarrollo.

Por defecto git crea una rama por defecto llamada master.

**En qué rama estamos?**

Para ver en qué rama nos encontramos utiliza el comando git status. La primera línea nos dice en qué rama estamos:

$ git status

On branch master

...

**Listar ramas**

Para listar todas las ramas utiliza el comando git branch. Por ejemplo:

$ git branch

\* master

rama-1

rama-2

...

La rama que tiene el asterísco (\*) es en la que nos encontramos actualmente.

**Crear una rama**

Para crear una rama utiliza el comando git branch seguido del nombre que le quieras dar a la rama. Por ejemplo, para crear una nueva rama llamada mi-rama ejecutaríamos el siguiente comando:

git branch mi-rama

Este comando crea la rama pero no nos ubica sobre ella. Para cambiar de rama debes ejecutar el siguiente comando:

git checkout mi-rama

Sin embargo, la mayoría de veces, cuando uno quiere crear una rama también se quiere ubicar sobre ella, así que existe un atajo:

git checkout -b mi-rama

La diferencia con el comando anterior es la opción -b.

**Cambiar el nombre de una rama**

Para cambiar el nombre de la rama debes estar ubicado sobre ella y ejecutar el comando git branch -m seguido del nuevo nombre.

Por ejemplo, asumiendo que estamos sobre la rama mi-rama, y que queremos cambiarle el nombre a otra-rama, ejecutaríamos el siguiente comando:

git branch -m otra-rama

**Integrar una rama**

Para integrar los commits de una rama (p.e. mi-rama) a otra (p.e. master) primero debemos ubicarnos sobre la rama principal (p.e. master) y ejecutar el comando git merge seguido del nombre de la rama:

git checkout master

git merge mi-rama

Si la rama master tiene commits que no están en mi-rama se va a crear un commit adicional llamado **merge commit**.

**Eliminar una rama**

Para eliminar una rama que ya ha sido integrada en otra utilizamos el comando git branch -d seguido del nombre de la rama que quieres eliminar.

Por ejemplo, asumiendo que la rama mi-rama ya fue integrada a master ejecutaríamos el siguiente comando:

git branch -d mi-rama

Si la rama no ha sido integrada aparecería un error. Sin embargo, si quisieramos eliminar una rama que aún no ha sido integrada podríamos cambiar la opción -d por -D:

git branch -D mi-rama

**Recursos adicionales**

Para configurar el editor (VSCode o Atom) sigue las instrucciones que se encuentran en el siguiente enlace:

* <https://guias.makeitreal.camp/git/instalacion-configuracion#configurando-el-editor-de-texto>

# Estado de archivos y .gitignore

**Nota:** En el video empezamos haciendo un resumen del estado de los archivos, pero desde el minuto 3:48 hablamos del archivo .gitignore.

En git es posible ignorar archivos o carpetas que no queremos incluir en el sistema de control de versiones, ya sea porque tienen información sensible o son autogeneradas por alguna herramienta: logs, archivos temporales, etc.

El archivo donde se define la lista de archivos y carpetas a ignorar se llama .gitignore.

Los archivos y carpetas definidos en .gitignore no van a aparecer como archivos nuevos o modificados al ejecutar git status.

Si aún no existe el archivo .gitignore podemos crearlo con nuestro editor de texto favorito e incluirlo en el siguiente commit.

En el archivo .gitignore escribimos los nombres de los archivo o carpetas que queremos ignorar (uno por línea). También podemos tener comentarios (utilizando numeral #). Por ejemplo:

# esto es un comentario

development.log

build

En este caso estamos ignorando un archivo llamado development.log (que esté en cualquier carpeta) y la carpeta build (incluyendo todos los archivos que estén dentro de la carpeta).

Si queremos ignorar un archivo o carpeta que esté en la raíz del proyecto (no en una subcarpeta), utilizaríamos un / antes del archivo o la carpeta. Por ejemplo, para ignorar el archivo development.log que se encuentra en la raíz utilizaríamos la siguiente línea:

/development.log

También puedes utilizar patrones en el archivo .gitignore:

# ignorar todos los archivos que terminen en .log

\*.log

# excepto production.log

!production.log

# ignorar los archivos terminados en .txt dentro de la carpeta doc (pero no sus subdirectorios)

doc/\*.txt

# ignorar todos los archivos terminados en .pdf dentro de la carpeta doc y sus subdirectorios

doc/\*\*/\*.pdf

En [este enlace](https://github.com/github/gitignore) vas a encontrar algunas plantillas del archivo .gitignore que Github ha creado para algunos lenguajes de programación y frameworks populares. Puede ser un buen punto de partida para tu proyecto o también una buena forma de ver ejemplos de este archivo.

# Github

Para colaborar con otros desarrolladores es fundamental que aprendas a trabajar con **repositorios remotos**.

Un **repositorio remoto** nos permite sincronizar nuestro repositorio con otro (u otros) en una ubicación diferente (generalmente en un servidor o un servicio en Internet, aunque podría ser también otra carpeta del mismo computador).

En esta guía vamos a trabajar con un servicio llamado [Github](https://github.com/" \t "_blank), que nos va a permitir crear repositorios centrales (públicos o privados).

Aunque en **git** no es necesario un repositorio central (por eso se dice que es distribuído), en la práctica es muy común y útil tenerlo. Primero, para mantener una copia de nuestro repositorio y, segundo, para facilitar la colaboración con otras personas.

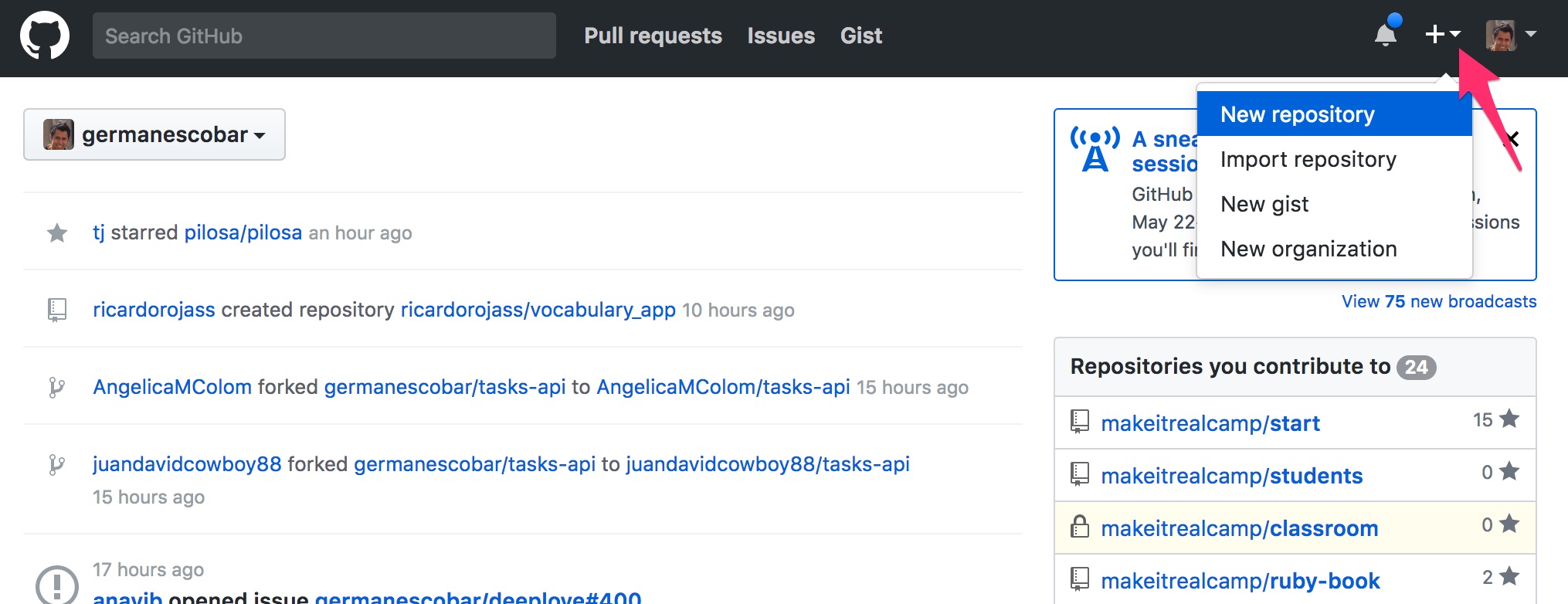
Existen otros servicios similares a [Github](https://github.com/" \t "_blank) como [Bitbucket](https://bitbucket.org/" \t "_blank) y [Gitlab](https://gitlab.com/" \t "_blank), entre otros. Incluso, si tienes un servidor (propio o alquilado), puedes configurar un servidor de **git** y alojar allí tus **repositorios remotos**. Esto esta fuera del alcance de estas guías pero puedes consultar [este recurso](https://git-scm.com/book/en/v2/Git-on-the-Server-The-Protocols" \t "_blank) para más información.

En [Github](https://github.com/" \t "_blank) puedes crear repositorios gratis, aunque hay algunas limitaciones. Consulta la [página de precios](https://github.com/pricing" \t "_blank) para más información.

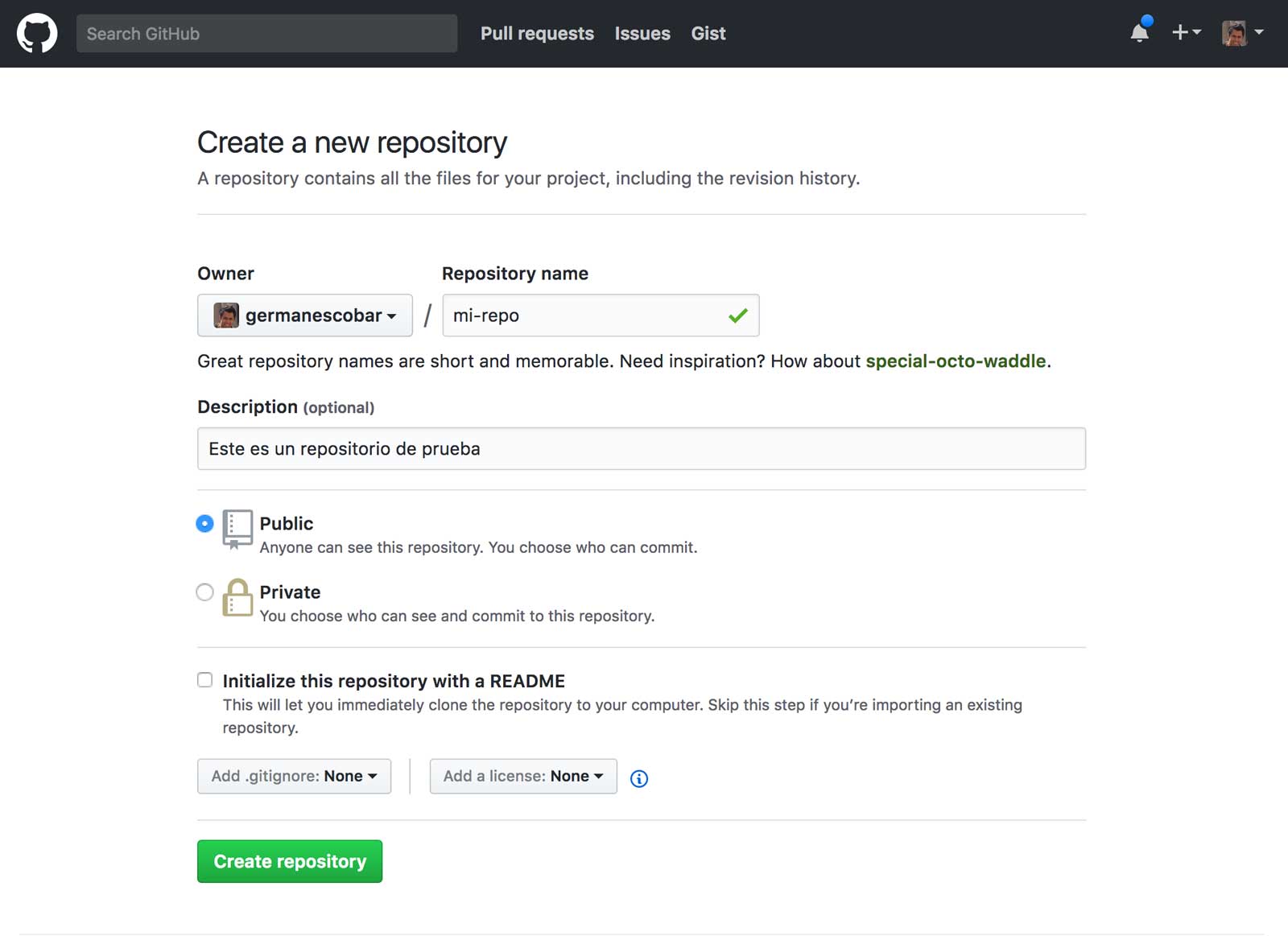
Si aún no tienes una cuenta en [Github](https://github.com/" \t "_blank) es hora de crear una antes de continuar.

**Creando un repositorio en Github**

Para crear un repositorio en [Github](https://github.com/" \t "_blank) ingresa a tu cuenta y haz click sobre la opción "New Repository" como se muestra en la siguiente imagen:



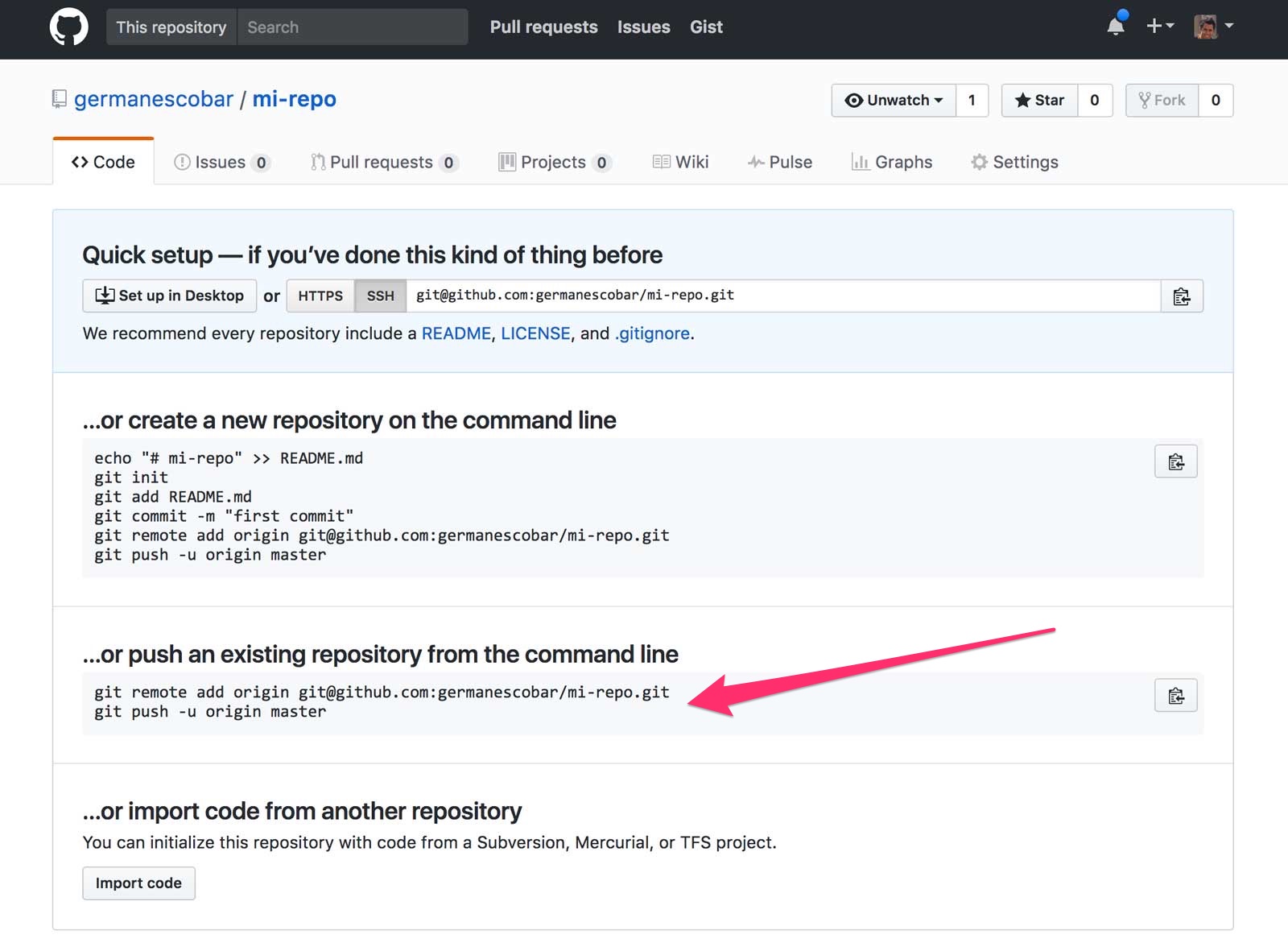
En la siguiente pantalla debes darle un nombre al repositorio, asegurarte que esté público y oprimir el botón "Create repository":



Has creado un repositorio en [Github](https://github.com/" \t "_blank), es hora de configurarlo en el repositorio local y subir el historial.

**Configurando el repositorio remoto**

Cuando creas un repositorio en [Github](https://github.com/" \t "_blank) te va a aparecer una pantalla como la que ves a continuación. Copia y pega en la consola las líneas que se indican con la flecha roja:



La primera línea configura el repositorio remoto en tu repositorio local. **Esta línea sólo la debes ejecutar una vez por repositorio remoto**.

En general, para configurar un repositorio remoto se utiliza el siguiente comando:

git remote add <nombre> <url>

**Nota:** aunque es posible tener múltiples repositorios remotos con diferentes nombres, de ahora en adelante vamos a asumir que sólo tienes uno llamado origin. Si se llama diferente, debes cambiar origin por el nombre correspondiente en los comandos que lo requieran. Para consultar el nombre del repositorio remoto ejecuta el comando git remote.

Para sincronizar los commits de la rama master con el **repositorio remoto** por primera vez se utiliza el siguiente comando (como se muestra en la imagen anterior):

git push -u origin master

Cuando crees nuevos commits puedes ejecutar el mismo comando sin la opción -u o puedes sólo ejecutar:

git push

Automáticamente **git** sabe a qué repositorio y rama enviar los nuevos commits (siempre y cuando hayas utilizado la opción -u la primera vez que hiciste el push).

**Recuerda que los commits no se envían automáticamente al remoto cuando los creas, debes ejecutar git push para enviarlos.**

**Clonando un repositorio**

Para descargar un repositorio a tu computador (a esto se le conoce como clonar) utiliza el comando git clone seguido de la URL del repositorio. Por ejemplo, el siguiente comando clonaría el repositorio de [Node.js](https://github.com/nodejs/node" \t "_blank):

git clone https://github.com/nodejs/node.git

Al ejecutar este comando se crea una carpeta llamada node con una copia completa del repositorio original (la carpeta oculta .git) y el espacio de trabajo (los archivos y carpetas del proyecto). También se configura un **repositorio remoto** llamado origin.

Puedes clonar cualquier repositorio público y repositorios privados a los que tengas acceso.

Al clonar un repositorio únicamente se descarga el historial de la rama master, a continuación veremos cómo descargar otras ramas.

**Descargando otras ramas**

Para descargar otras ramas del **repositorio remoto** utiliza el comando git checkout seguido del nombre de la rama. Por ejemplo, si en el repositorio remoto existe una rama llamada rama-1, ejecutaríamos los siguientes comandos para descargarla:

git fetch origin

git checkout rama-1

El primer comando actualiza la información del **repositorio remoto** (incluyendo las ramas que existen). El segundo comando descarga la rama si no existe localmente.

**Actualizando una rama**

Si estás trabajando con otros desarrolladores, es posible que tu repositorio local se desactualice con respecto al remoto. Sin embargo, no es posible actualizar todas las ramas a la vez, debes actualizarla una por una.

Para actualizar una rama úbicate sobre ella y ejecuta el comando git pull para actualizarla. Por ejemplo, para actualizar la rama master ejecutaríamos los siguientes comandos:

git checkout master

git pull

git pull es equivalente a ejecutar los siguientes dos comandos (asumiendo que el remoto se llama origin):

git fetch origin

git merge origin/master

git pull es recomendable ejecutarlo únicamente si no se han creado nuevos commits localmente. La razón es que si hay commits diferentes en la rama remota y la local, se crearía un **merge commit**. Por esta razón, si hay nuevos commits localmente, es mejor rebasar la rama como veremos en la siguiente sección.

**Rebasando una rama local contra una rama remota**

Si se han creado commits diferentes tanto en la rama local como en la rama remota es preferible rebasar la rama local contra la rama remota.

**Nota:** para aprender sobre rebasar ramas (localmente) te recomendamos referirte a la guía [Trabajando con ramas](https://fundamentos.makeitreal.camp/curriculum/git-github/ramas.md" \t "_blank).

Por ejemplo, para rebasar una rama local rama-1 contra una remota con el mismo nombre, ejecutaríamos los siguientes comandos:

git checkout rama-1

git fetch origin

git rebase origin/rama-1

El primer comando nos ubica en rama-1. El segundo comando actualizaría la información del repositorio remoto (incluyendo nuevos commits que puedan existir en la rama remota). El tercer comando rebasa la rama contra la rama remota.

También es posible rebasar una rama local contra otra rama remota diferente. Esto es muy útil cuando queremos actualizar nuestra rama local contra master pero ya tenemos nuevos commits en nuestra rama:

git fetch origin

git rebase origin/master

**Reemplazando una rama local**

En ocasiones necesitamos que una rama local quede igual que una rama remota. Por ejemplo, si quisieramos que la rama rama-1 quedara como la remota, ejecutaríamos los siguientes comandos:

git fetch origin

git reset --hard origin/rama-1

El comando git reset resetearía la rama en la que estamos actualmente ubicados y reemplazaría el historial con el que existe en la rama remota.

**Reemplazando una rama remota**

En ocasiones necesitamos que una rama remote quede igual que la rama local (el caso inverso de la sección anterior). Por ejemplo, si queremos que la rama remota rama-1 quede como la local, ejecutaríamos el siguiente comando:

git push -f <remoto> <rama>

Este comando es muy útil cuando hemos reescrito el historial de commits en nuestra rama local y queremos actualizar la rama remota. El push normal fallaría porque los commits no coincidirían.

**Nota:** Ten mucho cuidado con esta opción -f, puede que se pierdan cambios en la rama remota. Nunca lo utilices sobre master o sobre ramas en las que otras personas estén trabajando. En la mayoría de servicios como [Github](https://github.com/" \t "_blank) es posible proteger las ramas que no queremos que se puedan reemplazar.

**Resumen de los comandos**

* git clone <url> - clona un repositorio.
* git remote add <nombre> <url> - configura un repositorio remoto.
* git remote - lista los repositorios remotos.
* git push -u <remoto> <rama> - sincroniza la rama con la remota por primera vez.
* git push - envía los nuevos commits a la rama remota.
* git push -f <remoto> <rama> - reemplaza la rama remota con el historial de la rama local.
* git pull - actualiza los commits de la rama remota.
* git checkout <rama> - descarga la rama del remoto si no existe localmente.
* git fetch <remoto> - actualiza la información del repositorio remoto.
* git rebase <remoto>/<rama> - rebasa la rama local contra la remota.