Clase 6

BA MULTIPLICA 2.0
jóvenes x jóvenes

Git y GitHub

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

BA Joven

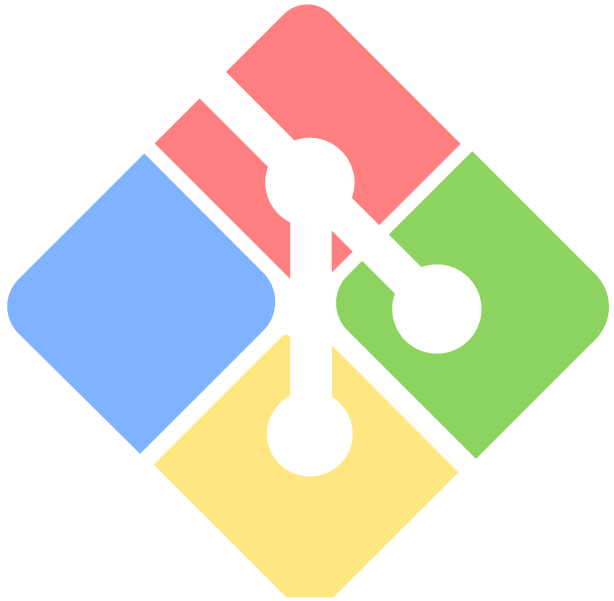
Contenidos de la Clase

- Introducción e interfaz
- Consola
- Comandos principales
- Trabajando en local y primeros commits
- GitHub: pull y push
- Git colaborativo
- Deploy

GIT

Es un sistema de **control de versiones** (VCS):

- ❑ Código
- ❑ Datos
- ❑ Texto



Ventajas:

- ✓ Rápido
- ✓ Diseño sencillo
- ✓ Desarrollo no lineal - en ramas paralelas
- ✓ Capaz de manejar grandes proyectos eficientemente

GIT - Configuración

- Primero vamos a descargarlo aquí dependiendo del sistema operativo que tengamos.

- Ahora establezcamos un nombre de usuario.

  Tenemos dos opciones:

  1. Establecer un nombre para TODOS los repositorios del dispositivo.
  2. Establecer un nombre para un repositorio único.

Pero... ¿qué es un repositorio?

Repositorios

“Un repositorio se utiliza a menudo para organizar un solo proyecto. Los repositorios pueden contener carpetas y archivos, imágenes, videos, hojas de cálculo y conjuntos de datos; todo lo que necesita tu proyecto.”

- ❑ Se encuentra online.
- ❑ Es el proyecto con todos los archivos “originales”.
- ❑ Puede ser público o privado.
- ❑ Contienen un archivo “readme”
- ❑ Puedo crear versiones diferentes a partir del repositorio principal.

GIT - Configuración

- Ahora sí... establezcamos un nombre de usuario.

  Tenemos dos opciones:

  1. Establecerlo para TODOS los repositorios del dispositivo.
  2. Establecer un nombre para un repositorio único.

Pero... ¿cuál elijo?

Documentación oficial en español: https://git-scm.com/book/es/v2
Documentación no oficial, pero más amigable:
https://www.atlassian.com/es/git/tutorials/what-is-version-control

¡Depende de cuántos proyectos esté llevando a cabo a la vez!

GIT - Configuración

Establecimiento del nombre de usuario de Git para *todos* los repositorios del equipo

1. Abra Git Bash.
2. Establece un nombre de usuario en Git:

```
$ git config --global user.name "Mona Lisa"
```

3. Confirma que has establecido correctamente el nombre de usuario en Git:

```
$ git config --global user.name
> Mona Lisa
```

GIT - Configuración

Configurar tu nombre de usuario de Git para un repositorio único

1. Abra Git Bash.
2. Cambia el directorio de trabajo actual al repositorio local donde deseas configurar el nombre que está asociado con tus confirmaciones de Git.
3. Establece un nombre de usuario en Git:

```
$ git config user.name "Mona Lisa"
```

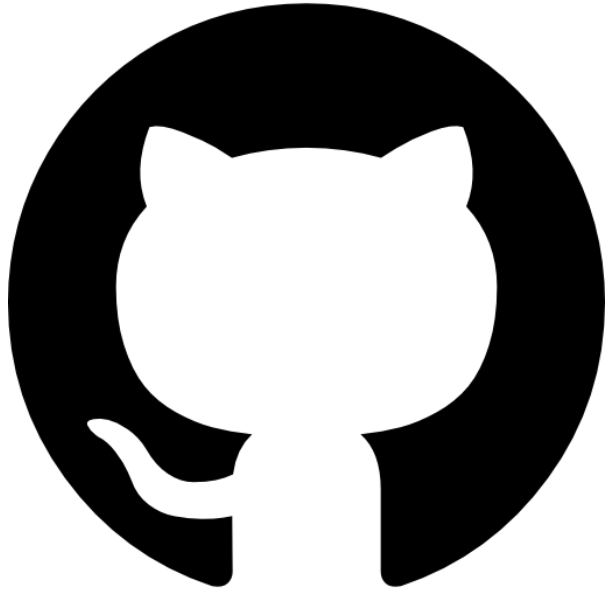
4. Confirma que has establecido correctamente el nombre de usuario en Git:

```
$ git config user.name
> Mona Lisa
```

GitHub

Es una plataforma de **hospedaje de código** para el control de versiones y la colaboración.

Permite que varias personas trabajen en un mismo proyecto a la vez.



Podemos...

- ✓ Crear ramas adicionales a partir de la rama principal
- ✓ Publicar código abierto
- ✓ Tener una copia actualizable de nuestro proyecto online

GitHub – Cuenta

1. Entramos a la página oficial de GitHub
2. Clickeamos “SignUp”
3. Ponemos los datos que nos piden (importante guardarlos!)
4. Confirmamos el correo

¡Listo!

GIT + GitHub – Enlace

Configurar tu dirección de correo electrónico para cada repositorio en tu computadora

1. Abra Git Bash.
2. Configurar una dirección de correo electrónico en Git. Puede usar la dirección de correo electrónico `noreply` proporcionada por GitHub o cualquier dirección de correo electrónico.

```
$ git config --global user.email "email@example.com"
```

3. Confirma que has establecido correctamente la dirección de correo electrónico en Git:

```
$ git config --global user.email
email@example.com
```

4. Agregue la dirección de correo electrónico a su cuenta en GitHub, de modo que las confirmaciones se le atribuyan y aparezcan en el gráfico de contribuciones. Para más información, vea "Adición de una dirección de correo electrónico a la cuenta de GitHub".

Crear Repositorios (remotos)

The home for all developers — including you.

Welcome to your personal dashboard, where you can find an introduction to how GitHub works, tools to help you build software, and help merging your first lines of code.

Start writing code

Start a new repository

A repository contains all of your project's files, revision history, and collaborator discussion.

RGDec / name your new repository...

Public
Anyone on the internet can see this repository

Private
You choose who can see and commit to this repository

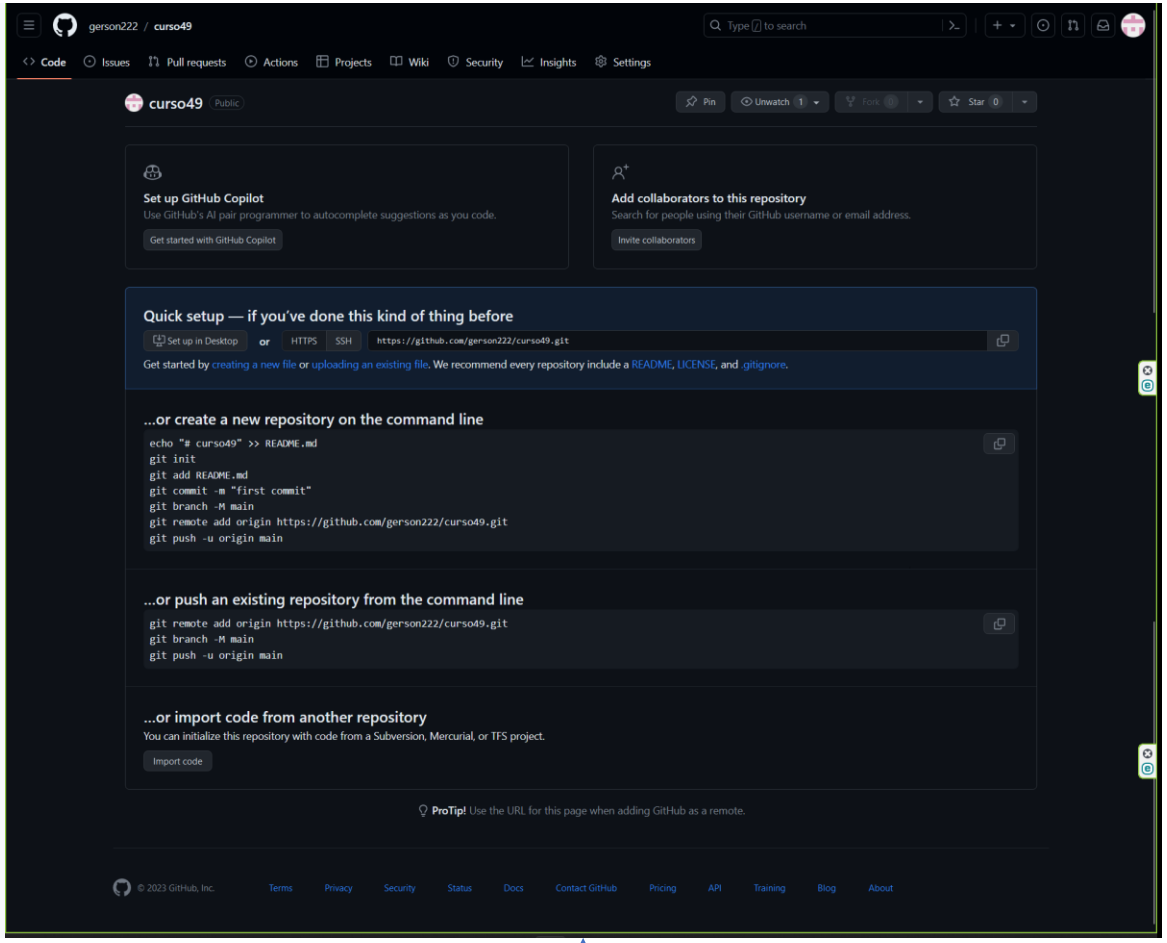
Create a new repository

Introduce yourself with a profile README

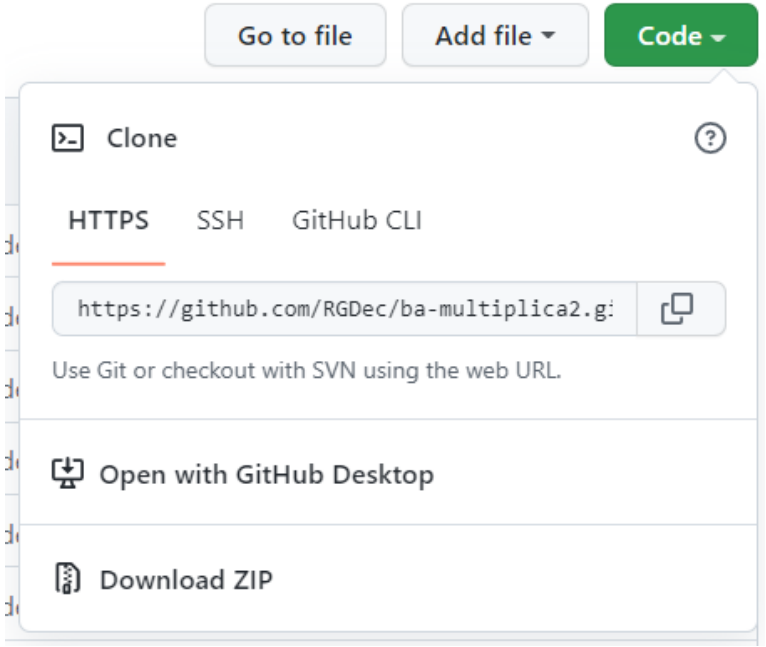
Share information about yourself by creating a profile README, which appears at the top of your profile page.

RGDec / README.md — Create

```
1 - 👋 Hi, I'm @RGDec
2 - 👀 I'm interested in ...
3 - 🌱 I'm currently learning ...
4 - 💞️ I'm looking to collaborate on ...
5 - 📫 How to reach me ...
6
```



Vincular Repositorios




```
PS C:\Users\rochu\Documents\DSK-Estudio\BA Multiplica 2.0\bamulti> git pull origin main
From https://github.com/RGDec/bamulti
 * branch            main       -> FETCH_HEAD
```

Vincular Repositorios

1) Sin repositorio local creado:
Si en la terminal de mi editor de código NUNCA ejecute el comando “git init”, github me da todas los comandos necesarios para crear dicho repositorio local, un readme, realiza el primer commit,

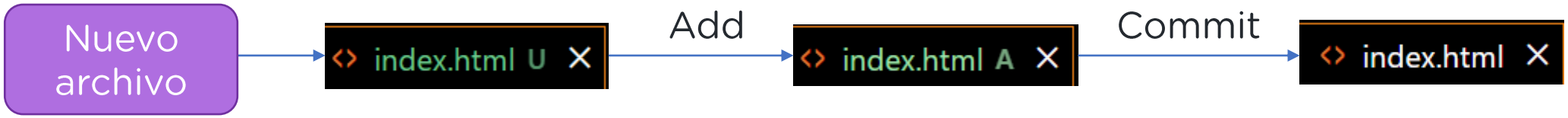
SOLO ES NECESARIO PEGAR TODAS LAS LINEAS JUNTAS EN LA TERMINAL Y TOCAR ENTER

2) Con repositorio local creado:
Si en la terminal de mi editor de código YA EJECUTE el comando “git init”

SOLO ES NECESARIO PEGAR TODAS LAS LINEAS JUNTAS EN LA TERMINAL Y TOCAR ENTER

Actualizar Repositorios

```
PS C:\Users\rochu\Documents\DSK-Estudio\BA Multiplica 2.0\bamulti> git add .
PS C:\Users\rochu\Documents\DSK-Estudio\BA Multiplica 2.0\bamulti> git commit -m "subiendo archivo"
[master 7570947] subiendo archivo
 1 file changed, 12 insertions(+)
 create mode 100644 index.html
```



Luego necesitaré subirlo a GitHub...

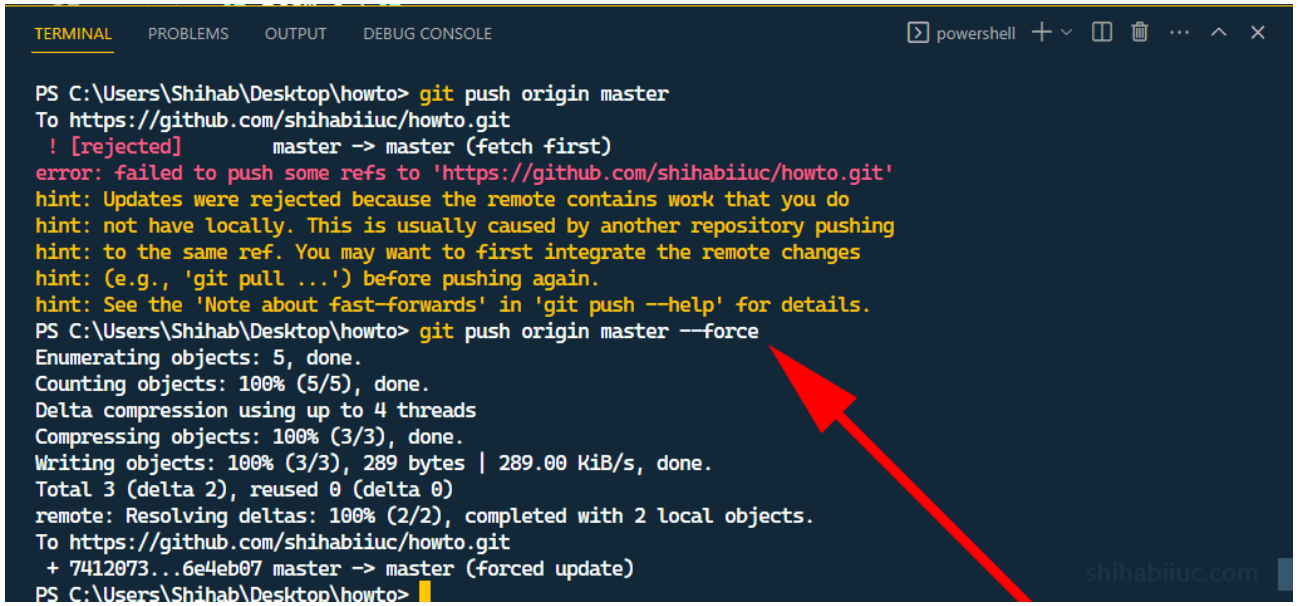
Push

git push es un comando fundamental en Git que se utiliza para enviar los cambios locales de tu repositorio Git a un repositorio remoto. Esto es esencial para colaborar en proyectos con otros desarrolladores y mantener una copia actualizada de tu código en el repositorio remoto.

```
Usuario@manolo22 MINGW64 ~/Desktop/proyecto/BAM2 (main)
$ git push origin main
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 345 bytes | 345.00 KiB/s, done.
Total 4 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/gerson222/curso49.git
   f045334..d45270e  main -> main
```

La primera vez que ejecutas git push origin main, estás configurando la relación inicial entre tu rama local y la rama remota con el mismo nombre en el repositorio remoto. Después de esta configuración inicial, en los siguientes git push, puedes usar simplemente git push sin especificar "origin main", ya que Git recordará automáticamente la configuración.

El comando git push --force es una variante que anula estos chequeos de seguridad y fuerza la actualización de la rama remota con la versión de tu rama local, sin importar si otros colaboradores han realizado cambios en la misma rama remota.
Normalmente se usa cuando no realice el comando “git pull”, el riesgo es que forzas a que el repositorio remoto sea como lo que estamos tratando de subir, ósea que si hay cosas en el remoto que en el local no, se va a borrar



Pull

El comando git pull se utiliza para actualizar tu repositorio local con los cambios más recientes del repositorio remoto. Básicamente, combina dos acciones:

“git merge” o git rebase (dependiendo de la configuración): Después de que “git fetch” haya descargado las actualizaciones del repositorio remoto, esta parte del comando combina los cambios locales con los cambios remotos. Puede hacerlo de dos formas diferentes, dependiendo de la configuración:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/ImDwivedi1/GitExample2
   f1ddc7c..0a1a475  master     -> origin/master
Updating f1ddc7c..0a1a475
Fast-forward
 design2.css | 6 ++++++
 1 file changed, 6 insertions(+)
 create mode 100644 design2.css
```

```
kb@phoenixNAP:~/project$ git fetch --all
Fetching remote
From github.com:phoenixNAP-KB/test
 * [new branch]      global      -> remote/global
 * [new branch]      main        -> remote/main
 * [new branch]      test        -> remote/test
 * [new branch]      test_alias  -> remote/test_alias
 * [new branch]      test_branch -> remote/test_branch
Fetching origin
From github.com:phoenixNAP-KB/test
 * [new branch]      global      -> origin/global
 * [new branch]      main        -> origin/main
 * [new branch]      test        -> origin/test
 * [new branch]      test_alias  -> origin/test_alias
 * [new branch]      test_branch -> origin/test_branch
```

git fetch: Esta parte del comando obtiene las actualizaciones más recientes del repositorio remoto, pero no realiza ninguna fusión en tu rama local. Descarga las referencias y objetos remotos, lo que te permite conocer los cambios que han ocurrido en el repositorio remoto sin aplicarlos automáticamente a tu rama local.

git merge: Realiza una fusión (merge) de las ramas local y remota, creando un nuevo commit de fusión si es necesario. Esta es la opción predeterminada en la mayoría de los casos.

```
Asus@Asus-PC MINGW64 /d/GeeksForGeeks-Branching and merging (master)
$ git merge next-five-even-odd
Updating a8d9e6b..3c477e1
Fast-forward
 Even-numbers.txt | 10 +++++++++-
 Odd-Number.txt   | 10 +++++++++-
 2 files changed, 18 insertions(+), 2 deletions(-)
```

Git Colaborativo (ramas)

```
Usuario@manolo22 MINGW64 ~/Desktop/proyecto/BAM2 (main)
$ git branch
* main
  prueba1
  prueba2
  prueba3
```

El comando **git Branch** se usa para que la terminal nos muestra una lista de las ramas creadas. Marcando con un * la rama en la cual nos encontramos ubicados. Además, en la terminal se puede ver (entre paréntesis) La rama en la cual estamos.

Al ejecutar el comando **git Branch "Rama"**. Lo que hacemos es crear una rama. Dicha rama, su nombre va a ser el que nosotros hayamos definido después de la palabra branch.

```
Usuario@manolo22 MINGW64 ~/Desktop/proyecto/BAM2 (main)
$ git branch rama4
```

```
Usuario@manolo22 MINGW64 ~/Desktop/proyecto/BAM2 (main)
$ git checkout rama4
Switched to branch 'rama4'

Usuario@manolo22 MINGW64 ~/Desktop/proyecto/BAM2 (rama4)
$
```

El comando **Git Checkout "rama"** lo que hace es movernos de una rama a la que nosotros hayamos escrito después de checkout. Pero, la rama tiene que haber sido creado previamente. De lo contrario, va a generar error.
Nos podemos dar cuenta que nos movimos exitosamente debido a que la siguiente línea, en la cual podemos introducir comando, ahora va a decir la rama en la cual estamos al final de la linea.

Git Colaborativo (ramas)

Si quisiera ejecutar los pasos de crear una rama y a la vez mudarme a ella en un solo paso, puedo ejecutar el comando **Git Checkout –b “rama”**. Esto lo que va a hacer es moverme una rama, la cual la va a crear instantáneamente y me lo va a decir en la siguiente línea que me estoy moviendo a una rama nueva indicando el nombre después del “-b”

```
Usuario@manolo22 MINGW64 ~/Desktop/proyecto/BAM2 (rama4)
$ git checkout -b rama5
Switched to a new branch 'rama5'

Usuario@manolo22 MINGW64 ~/Desktop/proyecto/BAM2 (rama5)
$
```

```
Usuario@manolo22 MINGW64 ~/Desktop/proyecto/BAM2 (rama5)
$ git branch -d rama4
Deleted branch rama4 (was d45270e).
```

En el caso de querer eliminar una rama, tengo que ejecutar el comando **git Branch –d “rama.** Donde yo voy a indicar. Qué rama quiero eliminar después de “-d”. En caso de que hubiera problemas y yo igual aún así, quiero eliminar la rama. En lugar de poner “-d”, pongo “-D”, esto va a forzar el borrado de la rama. Ignorando todo tipo de problema.

En caso de que yo haya hecho cambios en una rama y me parecen útiles y lo quiero trasladar a otra rama en lugar de andar, copiar y pegando a mano lo que puedo hacer es fusionar las ramas de tal forma que yo voy a indicar siempre qué rama quiero fusionar a la cual yo me encuentro presente. Usando el comando **git merge “rama”** lo que voy a hacer es fusionar la **“rama”** a la cual estoy haciendo indicación en el comando, en la rama en la cual me encuentro parado. O sea, la rama que aparece al final de la consola.
En la imagen, todo lo que está en “rama5” se manda a fusionar a “main”, ósea quien recibe la fusión es “main”.

```
Usuario@manolo22 MINGW64 ~/Desktop/proyecto/BAM2 (main)
$ git merge rama5
Updating d45270e..61c7d0b
Fast-forward
 clase5/index.html  |  2 +-
 clase5/index2.html | 11 ++++++++++
 2 files changed, 12 insertions(+), 1 deletion(-)
 create mode 100644 clase5/index2.html
```

Git Colaborativo (ramas)

```
Usuario@manolo22 MINGW64 ~/Desktop/proyecto/BAM2 (prueba3)
$ git branch --merged
  prueba2
* prueba3

Usuario@manolo22 MINGW64 ~/Desktop/proyecto/BAM2 (prueba3)
$ git branch --no-merged
  main
  rama5
```

Supongamos que por algún motivo yo quiero saber si hay cambios de otras ramas que no lo tienen mi rama. O. ¿Qué ramas ya fusioné a la rama en la cual me encuentro presente?. Git nos provee de comandos para poder ver esto.
Siendo que para ver esto tengo que ejecutar el comando **Git branch**, seguido de **--merged** o **--no-merged**
En ambos casos lo que me va a dar como respuesta a la terminal es una lista.

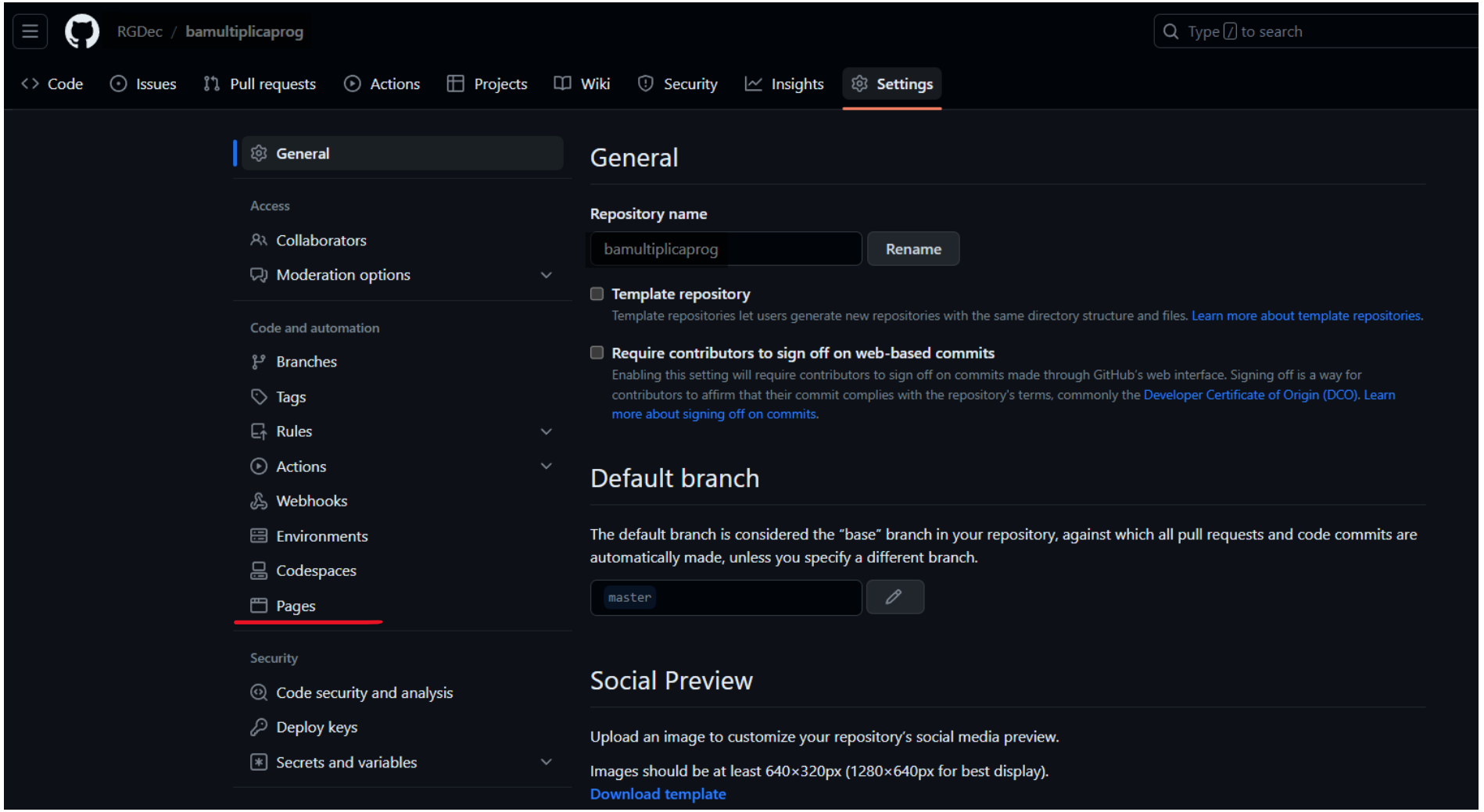
Si ejecuto el comando **git branch --merged**. Me va. Lo que va a hacer es comparar mi rama con las otras ramas y me va a devolver una lista de aquellas ramas que sean exactamente igual a la mía. Osea, cuyo contenido sea igual a mi rama.

En cambio. Si ejecuto el comando **git branch --no-merged**. Lo que va a hacer es nuevamente comparar mi rama con todas las ramas y aquellas que tengan contenido que mi rama no tiene me las va a devolver en una lista

En la imagen se puede ver que al ejecutar **git branch --merged** me devuelve “prueba2“ y “prueba3” con un* lo cual me indica que me encuentro en la rama prueba3 y su contenido es exactamente igual al de la rama prueba2.
En cambio, cuando ejecute **git branch --no-merged** me devolvió la rama “main” y la “rama5” porque dichas ramas tienen contenido que mi rama “prueba3” no contiene.

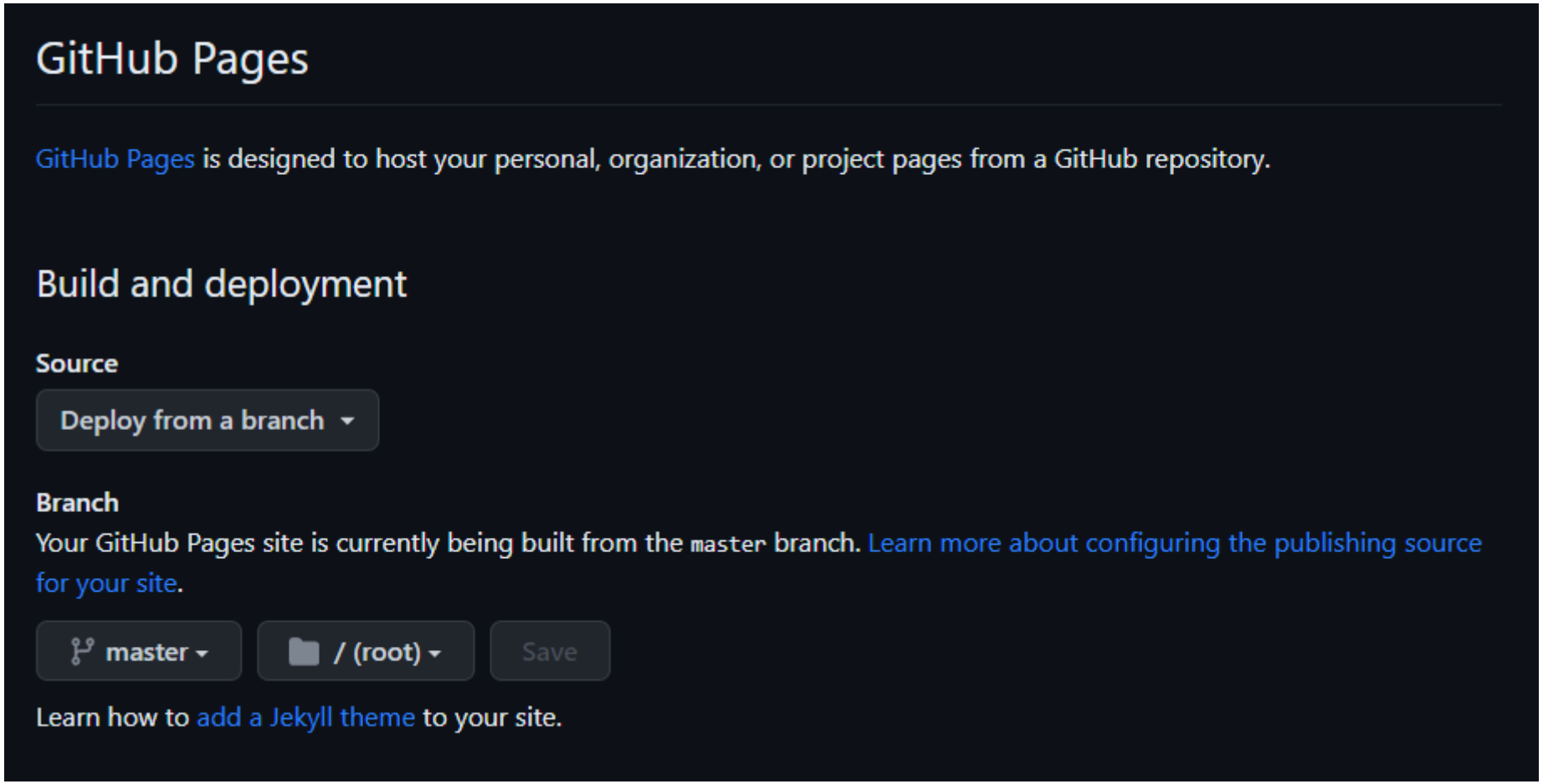
Deploy

Nos vamos a Settings → Pages



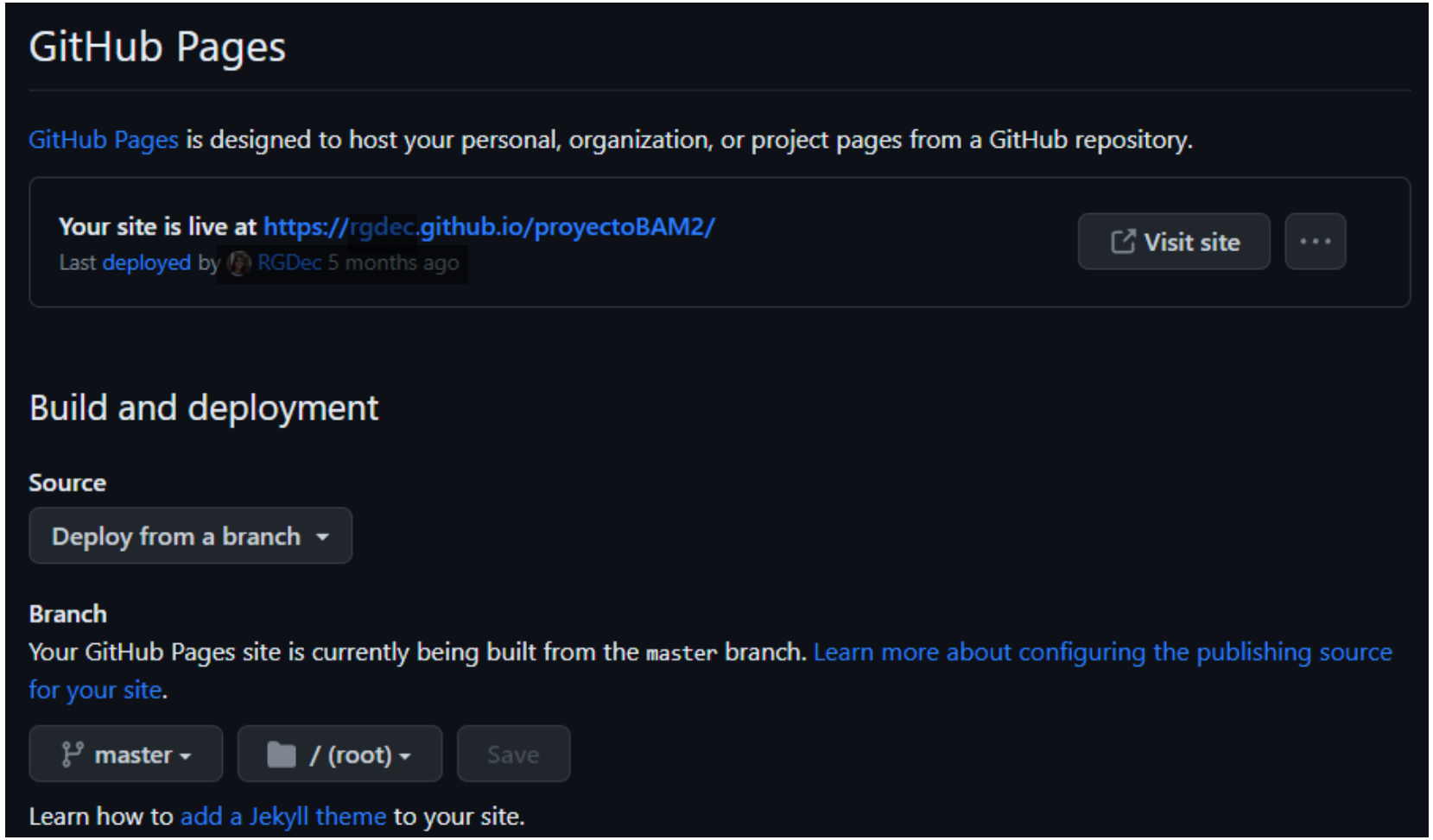
Deploy

"Deploy from a branch" y elijo mi rama principal → Save



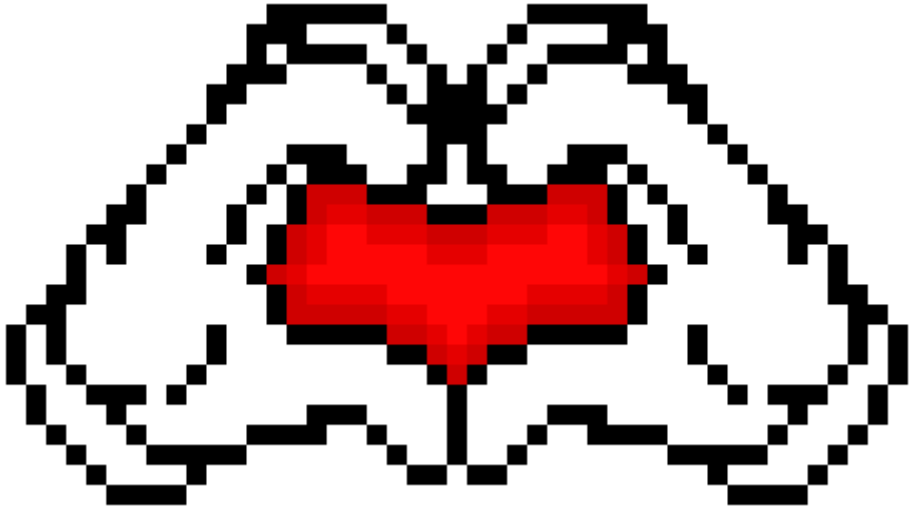
Deploy

Espero un ratito hasta que al actualizar la página me aparezca así...



¿Preguntas?

¡Nos vemos la próxima clase!



BA MULTIPLICA 2.0

jóvenes x jóvenes

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

BA Joven