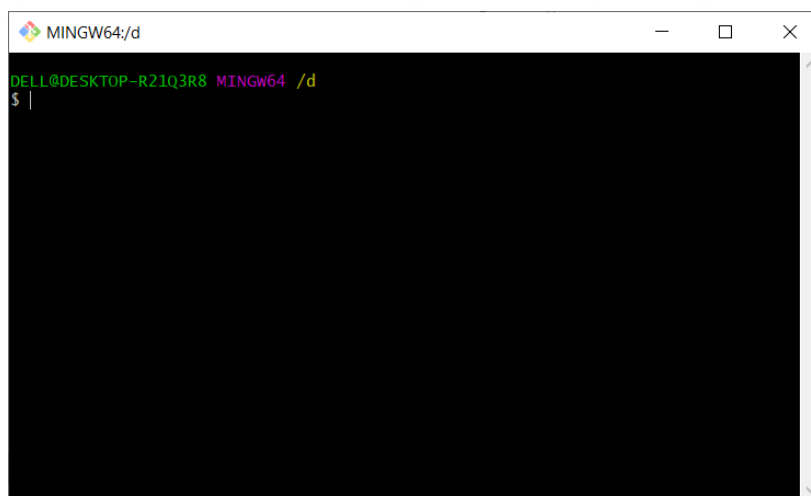




¿Qué es Git?

Es un sistema de control de versiones, es distribuido, es decir que múltiples personas pueden trabajar en equipo, es open source y también se adapta a todo tipo de proyectos desde pequeños hasta grandes, además, se pueden fusionar archivos, guarda una línea de tiempo a lo largo de todo el proyecto. Maneja una interfaz tipo Bash. GIT, es el software de control de versiones en el que se basa Github.



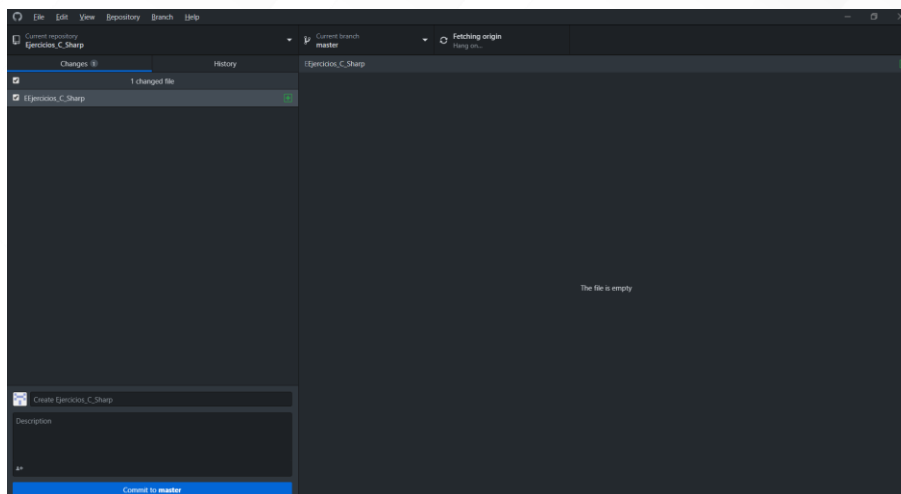
Sitio de descarga: <https://git-scm.com/downloads>

Como instalar Git: <https://www.youtube.com/watch?v=ExdLS6lZaAY>

¿Qué es Github?

A diferencia de Git, Github es un sitio web y un servicio en la nube que ayuda a los desarrolladores a almacenar y administrar su código, al igual que llevar un registro y control de cualquier cambio sobre este código. En otras palabras, es una plataforma de desarrollo colaborativo, o también llamada la red social de los desarrolladores donde se alojan los repositorios, el código se almacena de forma pública pero se puede hacer privado con una cuenta de pago.

La interfaz de GitHub es bastante fácil de usar para el desarrollador novato que quiera aprovechar las ventajas del Git. Sin GitHub, usar un Git generalmente requiere de un poco más de conocimientos de tecnología y uso de una línea de comando (Bash).



Sitio de descarga: <https://desktop.github.com/>
Como instalar Github: <https://www.youtube.com/watch?v=tn6tloweTUs>

¿Qué Es una Versión de Control?

Una Versión de Control ayuda a los desarrolladores a llevar un registro y administrar cualquier cambio en el código del proyecto de software. A medida que crece este proyecto, la versión de control se vuelve esencial.

Con la *bifurcación*, un desarrollador duplica parte del código fuente (llamado repositorio). Este desarrollador, luego puede, de forma segura, hacer cambios a esa parte del código, sin afectar al resto del proyecto.

Luego, una vez que el desarrollador logre que su parte del código funcione de forma apropiada, esta persona podría *fusionar* este código al código fuente principal para hacerlo oficial. Todos estos cambios luego son registrados y pueden ser revertidos si es necesario.

Documentación de Github: <https://docs.github.com/es/github>
Documentación Git: <https://git-scm.com/book/es/v2>

Creación de una cuenta

Lo primero que necesitas es una cuenta de usuario gratuita. Simplemente visita <https://github.com>, elige un nombre de usuario que no esté ya en uso, proporciona un correo y una contraseña, y pulsa el botón verde grande "Sign up for GitHub".



Formulario de registro de GitHub. Incluye campos para: Pick a username, Your email, y Create a password. Debajo del campo de contraseña, se indica: Use at least one lowercase letter, one numeral, and seven characters. En la parte inferior hay un botón verde que dice 'Sign up for GitHub'.

Lo siguiente que verás es la página de precios para planes mejores, pero lo puedes ignorar por el momento. GitHub te enviará un correo para verificar la dirección que les has dado. Confirmar la dirección ahora, es bastante importante (como veremos después).

Para ampliar esta información: <https://n9.cl/nqu9>

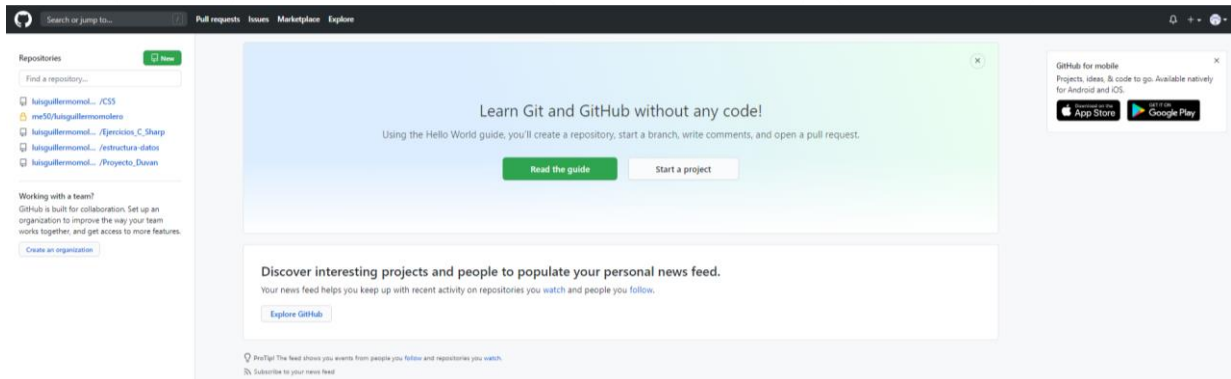
Crear un repositorio

Un **repositorio** se usa generalmente para organizar un solo proyecto. Los repositorios pueden contener carpetas y archivos, imágenes, videos, hojas de cálculo y conjuntos de datos, cualquier cosa que su proyecto necesite. Recomendamos incluir un archivo *README* o un archivo con información sobre su proyecto. GitHub facilita agregar uno al mismo tiempo que crea su nuevo repositorio. *También ofrece otras opciones comunes, como un archivo de licencia.*



Paso 1. Crear un nuevo repositorio

1. En la esquina superior derecha, junto a tu avatar o ícono de identidad, haz clic en (+) y luego seleccione **Nuevo repositorio**.



2. Nombra tu repositorio `estructura-datos`
3. Escribe una breve descripción.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Owner * luisguillermomolero / Repository name * estructura-datos ✓

Great repository names are short and memorable. Need inspiration? How about `literate-funicular`?

Description (optional) Repositorio ejercicios materia Estructura de Datos

☒ Public
Anyone on the internet can see this repository. You choose who can commit.

☐ Private
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☐ Initialize this repository with a README
This will let you immediately clone the repository to your computer.

Add .gitignore: None + Add a license: None + ⓘ

Create repository

4. Haga clic en **Crear repositorio**.



5. **Guarda** estos valores debido a que luego lo necesitaras para subir tus aplicaciones desde el *Bash* de **Git** a este **Github**

luisguillermomolero / estructura-datos

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH `https://github.com/luisguillermomolero/estructura-datos.git`

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# estructura-datos" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/luisguillermomolero/estructura-datos.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/luisguillermomolero/estructura-datos.git
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

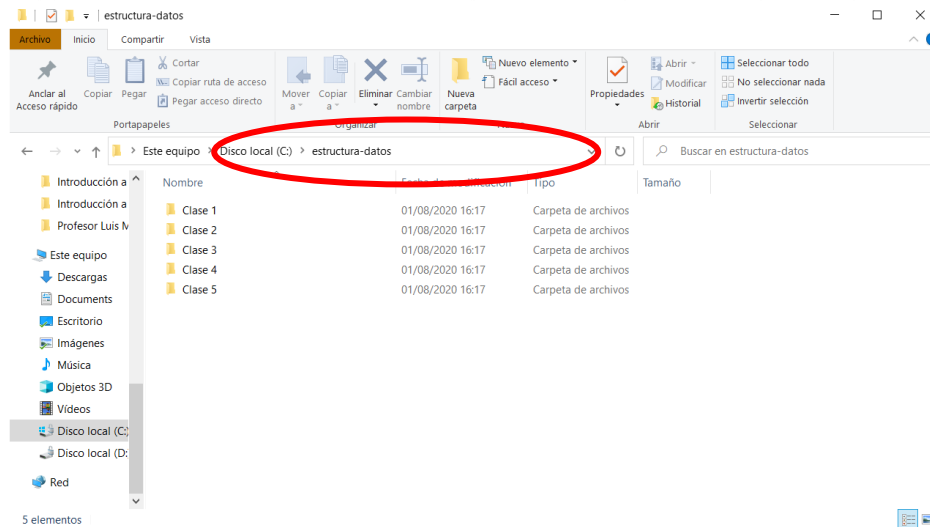
[Import code](#)



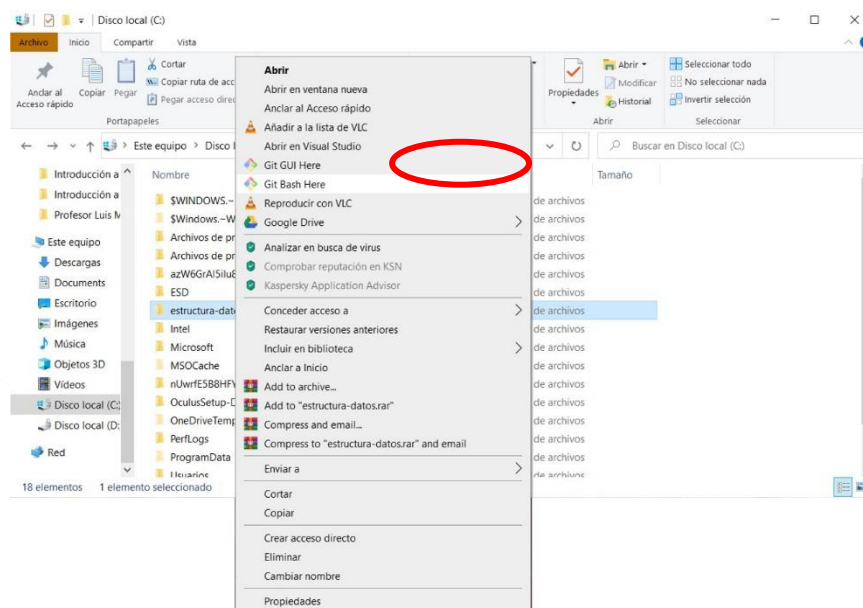
Paso 2. Subir nuestros proyectos a Github

1.- Crear una carpeta en el disco duro C:\ (ruta: C : \estructura-datos) para guardar todos los ejercicios resueltos de la materia estructura de datos.

2.- Crear una carpeta dentro de C : \estructura-datos para cada una de las clases (de hacerse de otro modo no serán evaluados los ejercicios entregados)



3.- Una vez creada las carpetas de cada clase y luego de haber instalado **Git** y **Github**, hacer click derecho sobre la carpeta “estructura-datos” y luego sobre la opción “**Git Bash Here**”.





Se abrirá la siguiente consola Bash

```
MINGW64:/c/estructura-datos
DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos
$
```

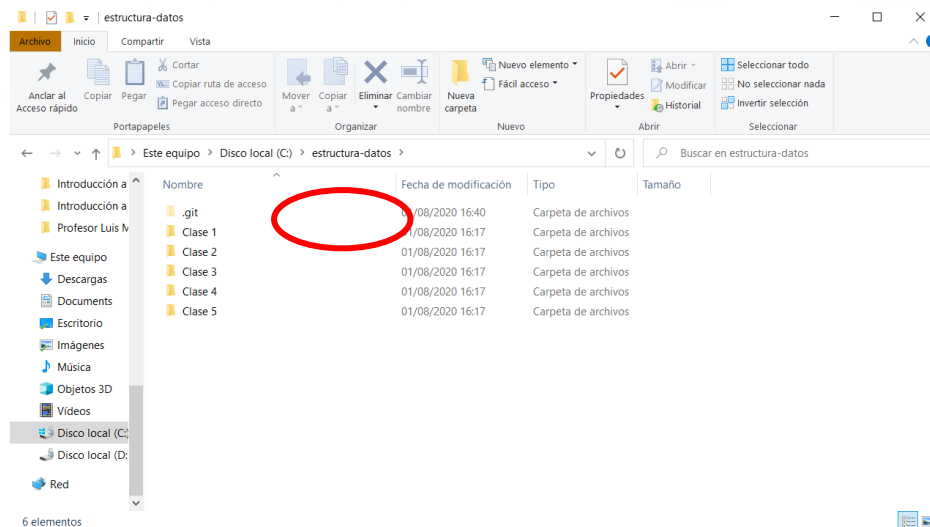
4.- Escribir la siguiente línea de comando para iniciar el **Git** en esa carpeta. **(ESTO SOLO SE HACE UNA VEZ)**

Git init

```
MINGW64:/c/estructura-datos
DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos
$ git init
Initialized empty Git repository in C:/estructura-datos/.git/
DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos (master)
$
```

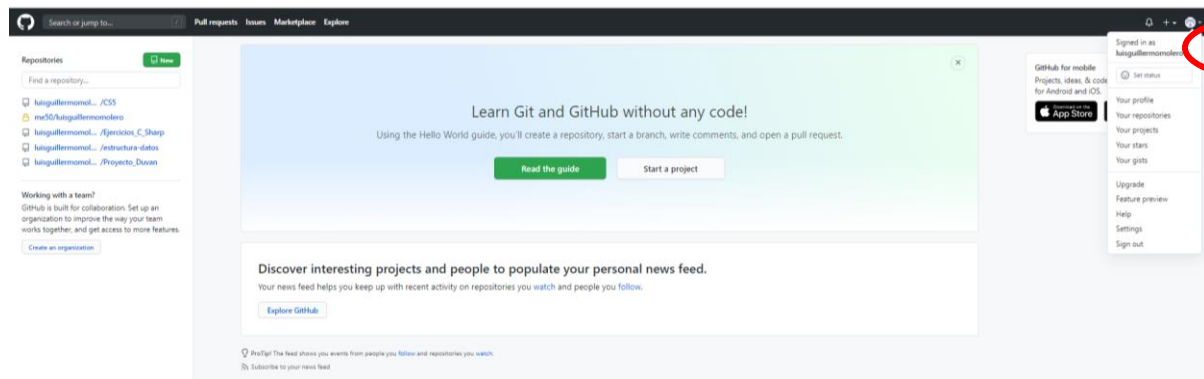


A continuación, todo lo que agregue dentro de esa carpeta estará dentro del **Git** y se podrá sincronizar en la nube. Asimismo, dentro de esa carpeta se creará una carpeta llamada **.git** que esta oculta.



5.- Escribir la siguiente línea de comandos para establecer la configuración de nombre de usuario / correo electrónico específica del repositorio: **(ESTO SOLO SE HACE UNA VEZ)**. Debes usar el mismo correo que utilizaste al crear tu cuenta en Github.

```
git config --global user.name "tu nombre de usuario en Github"
```

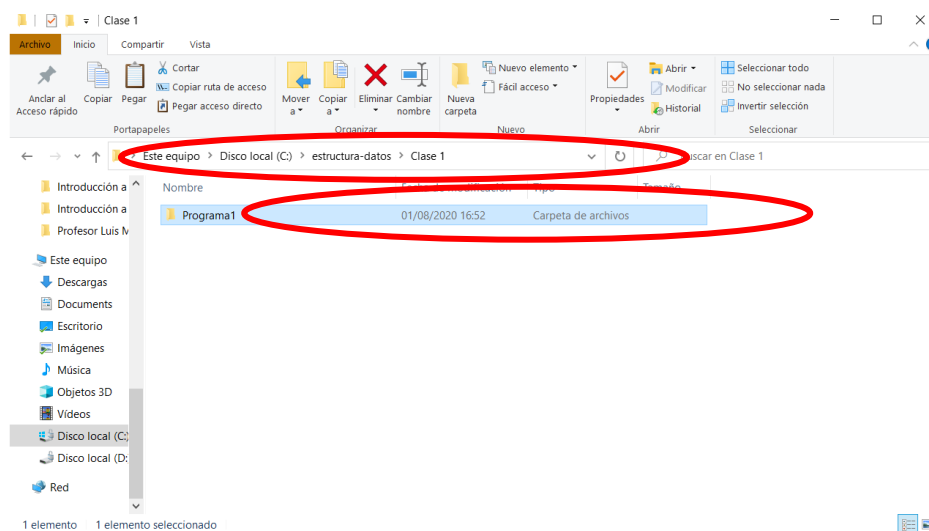


```
git config --global user.email "tu correo en Github"
```




```
MINGW64/c/estructura-datos
DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos
$ git init
Initialized empty Git repository in C:/estructura-datos/.git/
DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos (master)
$ git config --global user.name "luisguillermomolero"
DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos (master)
$
DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos (master)
$ git config --global user.email "luisguillermomolero@gmail.com"
DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos (master)
$
```

Supongamos que en esta primera entrega ud. va a subir el primer programa realizado como tarea, como se muestra en la siguiente imagen:



6.- Para subir ese primer ejercicio resuelto, debes repetir solo el paso 3 y cuando aparezca el Bash de **GIT** escribir la siguiente línea de comando:

(ESTO LO DEBES HACER CADA VEZ QUE MODIFIQUES ALGÚN EJERCICIO O HAGAS UN EJERCICIO NUEVO)

```
git add .
```



```
MINGW64:/c/estructura-datos
DELL@DESKTOP-K21Q3R8 MINGW64 /c/estructura-datos (master)
$ git add .
DELL@DESKTOP-K21Q3R8 MINGW64 /c/estructura-datos (master)
$
```

git add : Lleva el control de los archivos que se agregan luego de escribir ese comando y **git add** . sirve para agregar todos los archivos modificados después del primer envío (commit).



7.- Una vez ejecutado el `git add` . ejecutamos la siguiente línea de comando para validar el estado actual de nuestro **Git**.

`git status`

```
MINGW64/C:/estructura-datos
C:\estructura-datos (master)
$ git add .
C:\estructura-datos (master)
$ git status
On branch master
No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   Clase 1/Programal/DesignTimeBuild/.dtb.cache.v2
    new file:   Clase 1/Programal/.vs/Programal/v16/.suo
    new file:   Clase 1/Programal/Program.cs
    new file:   Clase 1/Programal/Programal.csproj
    new file:   Clase 1/Programal/Programal.sln
    new file:   Clase 1/Programal/bin/Debug/netcoreapp3.1/Programal.deps.json
    new file:   Clase 1/Programal/bin/Debug/netcoreapp3.1/Programal.dll
    new file:   Clase 1/Programal/bin/Debug/netcoreapp3.1/Programal.exe
    new file:   Clase 1/Programal/bin/Debug/netcoreapp3.1/Programal.pdb
    new file:   Clase 1/Programal/bin/Debug/netcoreapp3.1/Programal.runtimeconfig.json
    new file:   Clase 1/Programal/bin/Debug/netcoreapp3.1/Programal.runtimeconfig.json
    new file:   Clase 1/Programal/obj/Debug/netcoreapp3.1/Programal.Assembly
    new file:   Clase 1/Programal/obj/Debug/netcoreapp3.1/Programal.Assembly
    new file:   Clase 1/Programal/obj/Debug/netcoreapp3.1/Programal.assets.cache
    new file:   Clase 1/Programal/obj/Debug/netcoreapp3.1/Programal.csproj.C
    new file:   Clase 1/Programal/obj/Debug/netcoreapp3.1/Programal.csproj.F
    new file:   Clase 1/Programal/obj/Debug/netcoreapp3.1/Programal.csprojAs
    new file:   Clase 1/Programal/obj/Debug/netcoreapp3.1/Programal.dll
    new file:   Clase 1/Programal/obj/Debug/netcoreapp3.1/Programal.exe
    new file:   Clase 1/Programal/obj/Debug/netcoreapp3.1/Programal.genruntime
    new file:   Clase 1/Programal/obj/Debug/netcoreapp3.1/Programal.pdb
    new file:   Clase 1/Programal/obj/Programal.csproj.nuget.dgspec.json
    new file:   Clase 1/Programal/obj/Programal.csproj.nuget.g.props
    new file:   Clase 1/Programal/obj/Programal.csproj.nuget.g.targets
    new file:   Clase 1/Programal/project.assets.json
    new file:   Clase 1/Programal/project.nuget.cache
```

El comando `git status` te mostrará los diferentes estados de los archivos en tu directorio de trabajo y área de ensayo. Qué archivos están modificados y sin seguimiento y cuáles con seguimiento pero no confirmados aún. En su forma normal, también te mostrará algunos consejos básicos sobre cómo mover archivos entre estas etapas.



```
ejemplo: git commit -m "primera actualizacion"
```

```

MINGW64/c/estructura-datos
$ git commit -m "primera actualizacion"
[master 60c662] primera actualizacion
26 files changed, 308 insertions(+), 0 deletions(-)
create mode 100644 Clase 1/Programa1/DesignTimeBuild/.dtbcache.v2
create mode 100644 Clase 1/Programa1/.vs/Programa1/v16/.suo
create mode 100644 Clase 1/Programa1/.vs/Programa1/Program.cs
create mode 100644 Clase 1/Programa1/Programa1.csproj
create mode 100644 Clase 1/Programa1/Programa1.sln
create mode 100644 Clase 1/Programa1/bin/Debug/netcoreapp3.1/Programa1.deps.json
create mode 100644 Clase 1/Programa1/bin/Debug/netcoreapp3.1/Programa1.dll
create mode 100644 Clase 1/Programa1/bin/Debug/netcoreapp3.1/Programa1.exe
create mode 100644 Clase 1/Programa1/bin/Debug/netcoreapp3.1/Programa1.pdb
create mode 100644 Clase 1/Programa1/bin/Debug/netcoreapp3.1/Programa1.runtimeconfig.dev.json
create mode 100644 Clase 1/Programa1/bin/Debug/netcoreapp3.1/Programa1.runtimeconfig.json
create mode 100644 Clase 1/Programa1/obj/Debug/netcoreapp3.1/Programa1.AssemblyInfo.cs
create mode 100644 Clase 1/Programa1/obj/Debug/netcoreapp3.1/Programa1.AssemblyInfoInputs.cache
create mode 100644 Clase 1/Programa1/obj/Debug/netcoreapp3.1/Programa1.assets.cache
create mode 100644 Clase 1/Programa1/obj/Debug/netcoreapp3.1/Programa1.csproj.CoreCompileInputs.cache
create mode 100644 Clase 1/Programa1/obj/Debug/netcoreapp3.1/Programa1.csproj.FileListAbsolute.txt
create mode 100644 Clase 1/Programa1/obj/Debug/netcoreapp3.1/Programa1.csproj.AssemblyReference.cache
create mode 100644 Clase 1/Programa1/obj/Debug/netcoreapp3.1/Programa1.dll
create mode 100644 Clase 1/Programa1/obj/Debug/netcoreapp3.1/Programa1.exe
create mode 100644 Clase 1/Programa1/obj/Debug/netcoreapp3.1/Programa1.genruntimeconfig.cache
create mode 100644 Clase 1/Programa1/obj/Debug/netcoreapp3.1/Programa1.pdb
create mode 100644 Clase 1/Programa1/obj/Programa1.csproj.nuget.dgspec.json
create mode 100644 Clase 1/Programa1/obj/Programa1.csproj.nuget.g.props
create mode 100644 Clase 1/Programa1/obj/Programa1.csproj.nuget.g.targets
create mode 100644 Clase 1/Programa1/obj/project.assets.json
create mode 100644 Clase 1/Programa1/obj/project.nuget.cache

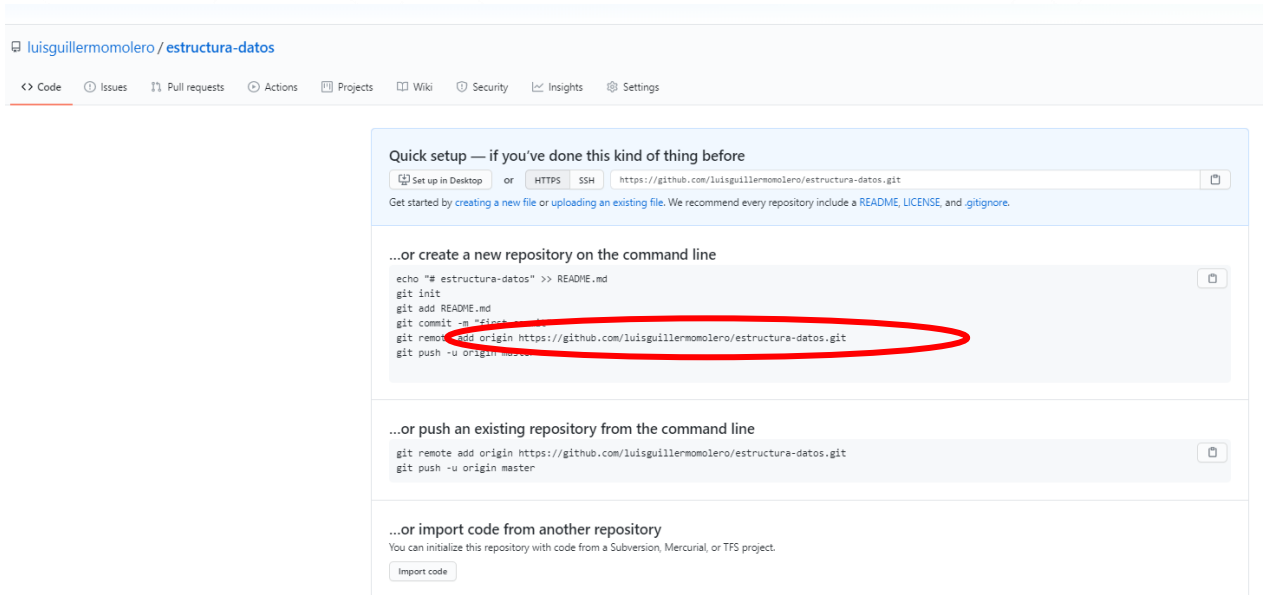
MINGW64/c/estructura-datos (master)

```

git commit : Este comando indica que esta lista alguna funcionalidad para que sea una versión del código. Este comando se repite n veces cada vez se cambia el código.

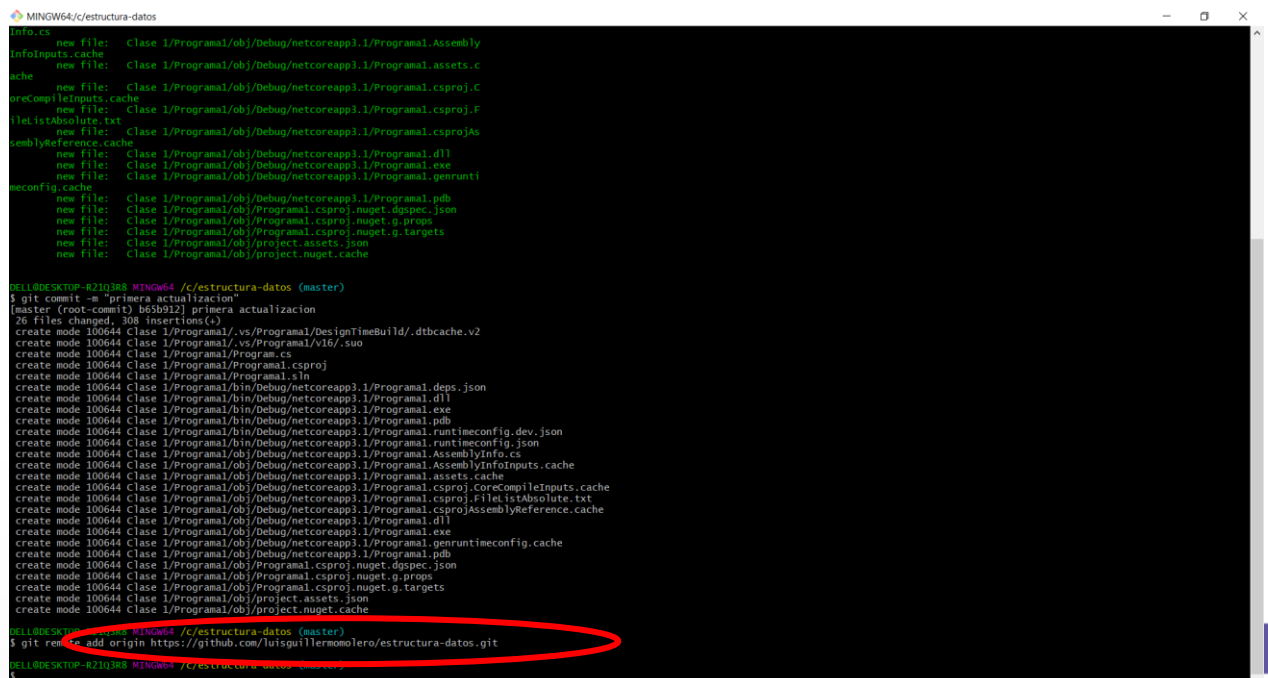


9.- Una vez ejecutado el `git commit` ubicamos los datos que guardamos en el sub-paso 5 (Guarda estos valores debido a que luego lo necesitaras para subir tus aplicaciones desde el Bash de Git a este Github) de nuestro Paso 1. (Paso 1. Crear un nuevo repositorio)



10.- Ejecutamos la siguiente línea de comando que tomamos del paso agregar un nuevo control remoto.

```
git remote add origin https://github.com/luisguillermomolero/estructura-datos.git
```





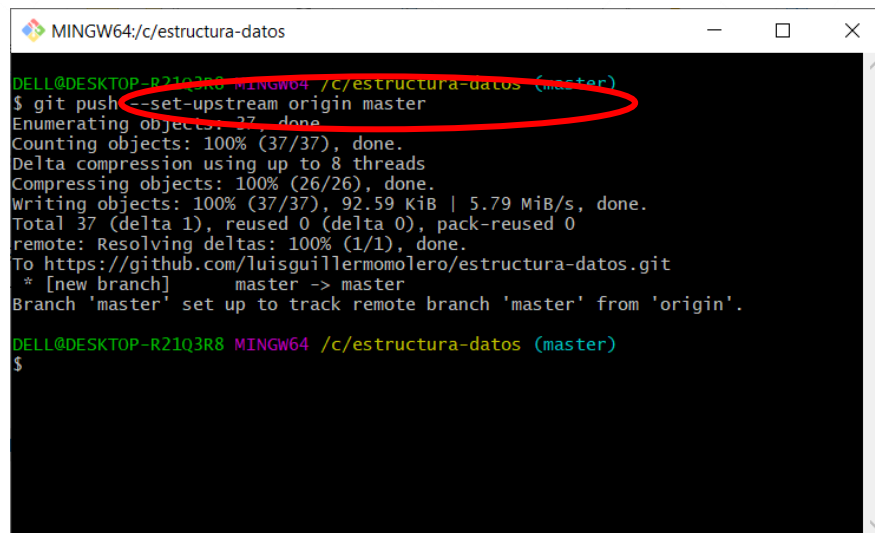
El futuro digital
es de todos

MinTIC



11.- Finalmente, luego de ejecutar la línea de comando anterior, ejecutamos un `git push` para subir tus cambios locales a tu repositorio en línea.

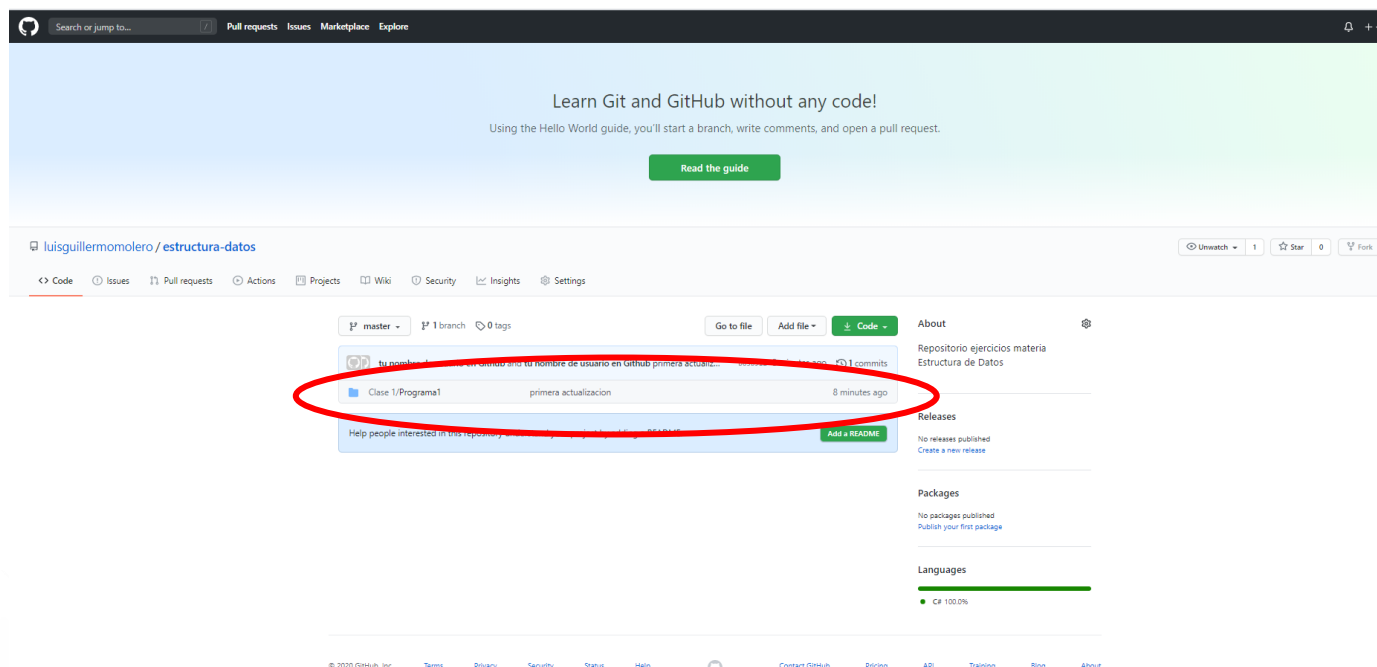
```
git push --set-upstream origin master
```



```
MINGW64/c/estructura-datos
DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos (master)
$ git push --set-upstream origin master
Enumerating objects: 37, done.
Counting objects: 100% (37/37), done.
Delta compression using up to 8 threads
Compressing objects: 100% (26/26), done.
Writing objects: 100% (37/37), 92.59 KiB | 5.79 MiB/s, done.
Total 37 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/luisguillermomolero/estructura-datos.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.

DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos (master)
$
```

Ya podemos ver nuestro primer proyecto (Programa) en la nuevo de **GitHub**





IMPORTANTE

Cada vez que se modifique un proyecto (programa) o se cree uno nuevo, se debe ejecutar del paso 2 (Paso 2. Subir nuestros proyectos a Github) los siguientes sub-pasos:

3.- Una vez creada las carpetas de cada clase y luego de haber instalado **Git** y **Github**, hacer clic derecho sobre la carpeta “*estructura-datos*” y luego sobre la opción “**Git Bash Here**”.

6.- Para subir ese primer ejercicio resuelto, debes repetir solo el paso 3 y cuando aparezca el Bash de **GIT** escribir la siguiente línea de comando:

7.- Una vez ejecutado el `git add .` ejecutamos la siguiente línea de comando para validar el estado actual de nuestro **Git**.

8.- Una vez ejecutado el comando `git status` ejecutamos la siguiente línea de comando para instanciar los cambios preparados en ese momento.

10.- Ejecutamos la siguiente línea de comando que tomamos del paso agregar un nuevo control remoto.

11.- Finalmente, luego de ejecutar la línea de comando anterior, ejecutamos un `git push` para subir tus cambios locales a tu repositorio en línea.