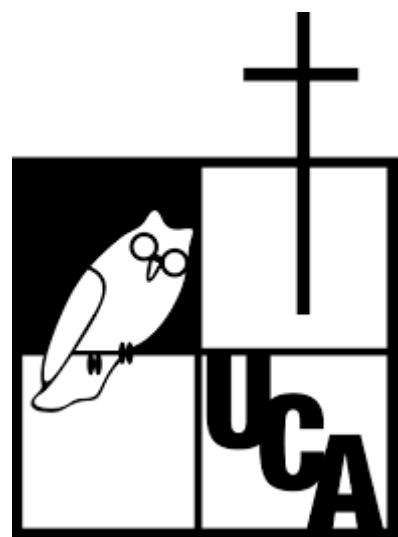


Universidad Centroamericana “José Simeón Cañas”

Facultad de ingeniería y arquitectura

Programación de Estructuras Dinámicas



Guía de trabajo GitHub

Objetivo

Familiarizar al estudiante con el manejo correcto de los comandos básicos de git y el uso fundamental de ramas para la implementación de trabajo en repositorios remotos.

Resumen de la guía

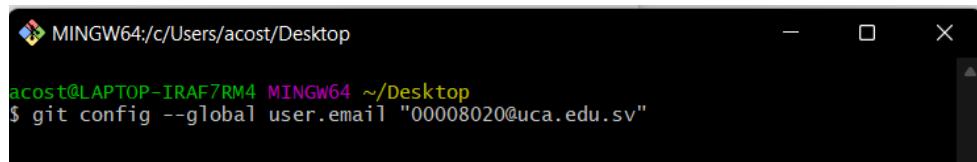
1. Crear un nuevo repositorio en github utilizando la interfaz gráfica.
2. Clonar el repositorio y ubicarnos en el directorio creado usando la terminal de git.
3. Abrir el directorio creado en visual studio code.
4. Inicializar el repositorio con un archivo *readme.md*
5. Crear dos ramas, una llamada *BubbleSort* y la otra *InsertionSort*.
6. Movernos entre las ramas anteriormente creadas y publicarlas en github.
7. Trabajando en la rama *BubbleSort*. Agregar un archivo llamado *bubbleSort.cpp*.
8. Subir el archivo creado a github (siempre trabajando en la rama *BubbleSort*).
9. Trabajando en la rama *InsertionSort*. Agregar un archivo llamado *insertionSort.cpp* y un archivo de texto *resultados.txt*.
10. Publicar los archivos anteriormente creados en github (siempre trabajando en la rama *InsertionSort*).
11. Agregar un archivo *confidencial.txt* y un archivo *.gitignore*.
12. Utilizando el archivo *.gitignore* ignorar el archivo *confidencial.txt* para que este no sea publicado al repositorio de github.
13. Publicar los cambios realizados y verificar que *confidencial.txt* no haya sido publicado solamente el archivo *.gitignore*.
14. Revisar en el repositorio de github que ambas ramas contienen archivos diferentes.
15. Hacer merge de las dos ramas (*BubbleSort* y *InsertionSort*) con la rama principal o (*main*) Utilizando la interfaz gráfica de github.
16. Moverse a la rama *main* utilizando la terminal. Notarán que los archivos no existen en dicha rama.
17. Realizar un *pull* de la rama *main* para actualizar los archivos locales con los cambios que contiene el repositorio de github.

Configurar datos de usuario

En este paso se actualizarán los datos que se mostrarán en los push referentes al sistema utilizado.

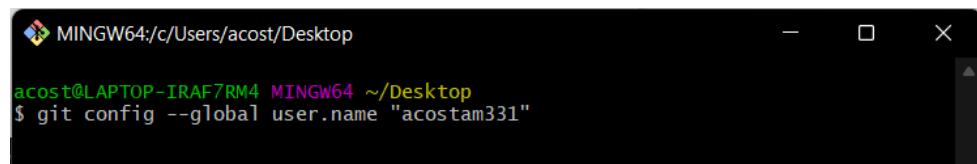
Haciendo uso de los comandos:

```
git config --global user.email "[correo]"
```



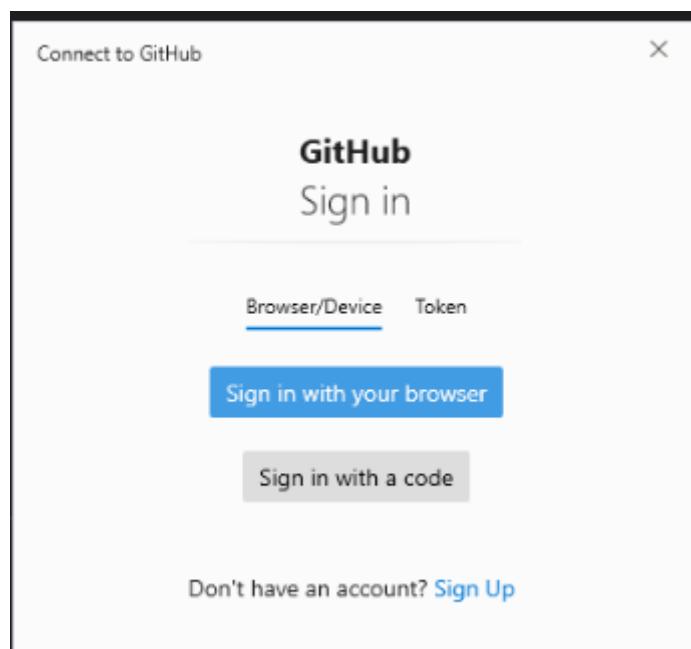
A screenshot of a terminal window titled "MINGW64:c/Users/acost/Desktop". The command \$ git config --global user.email "00008020@uca.edu.sv" is entered and executed successfully.

```
git config --global user.name "[nombre]"
```



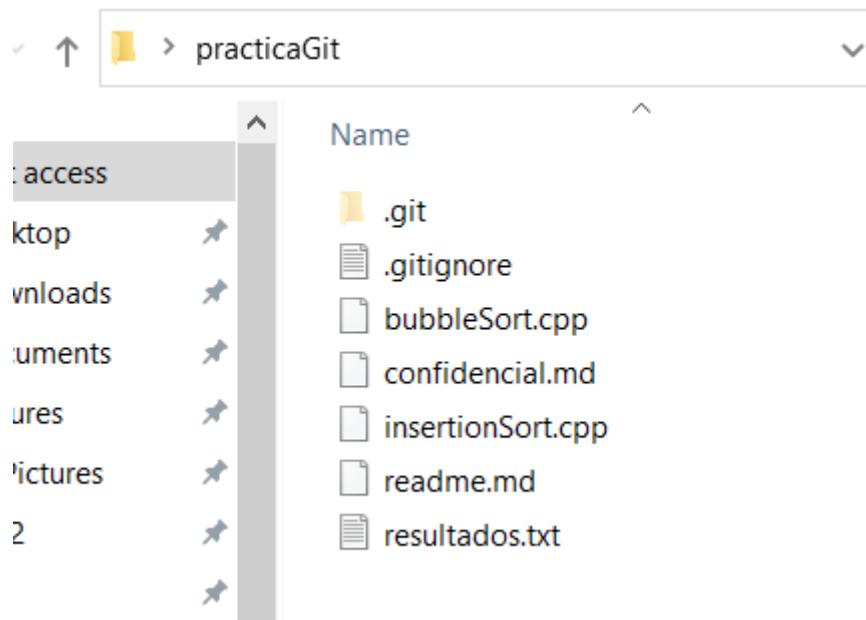
A screenshot of a terminal window titled "MINGW64:c/Users/acost/Desktop". The command \$ git config --global user.name "acostam331" is entered and executed successfully.

Nota: En el caso que se desee clonar o acceder a un repositorio protegido git les solicitará que inicien sesión con tus credenciales institucionales, posiblemente no tendrán que realizarlo al resolver esta guía pero tiene que ser tomado en cuenta.



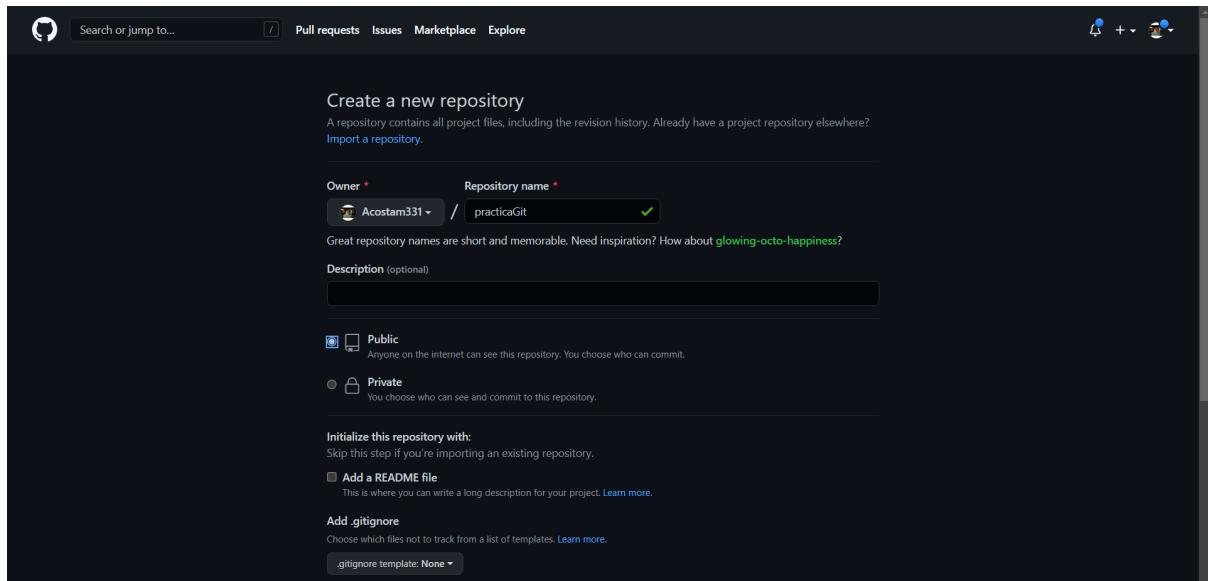
Nota aclaratoria antes de empezar con la práctica

Cuando se esté trabajando en la carpeta de nuestro proyecto (carpeta que es creada al clonar un repositorio de git), los archivos con los que trabajamos **NO SE DEBEN DE INCLUIR DENTRO DE LA CARPETA .GIT**, ellos deben estar al mismo nivel del directorio base, como se muestra en la siguiente imagen.

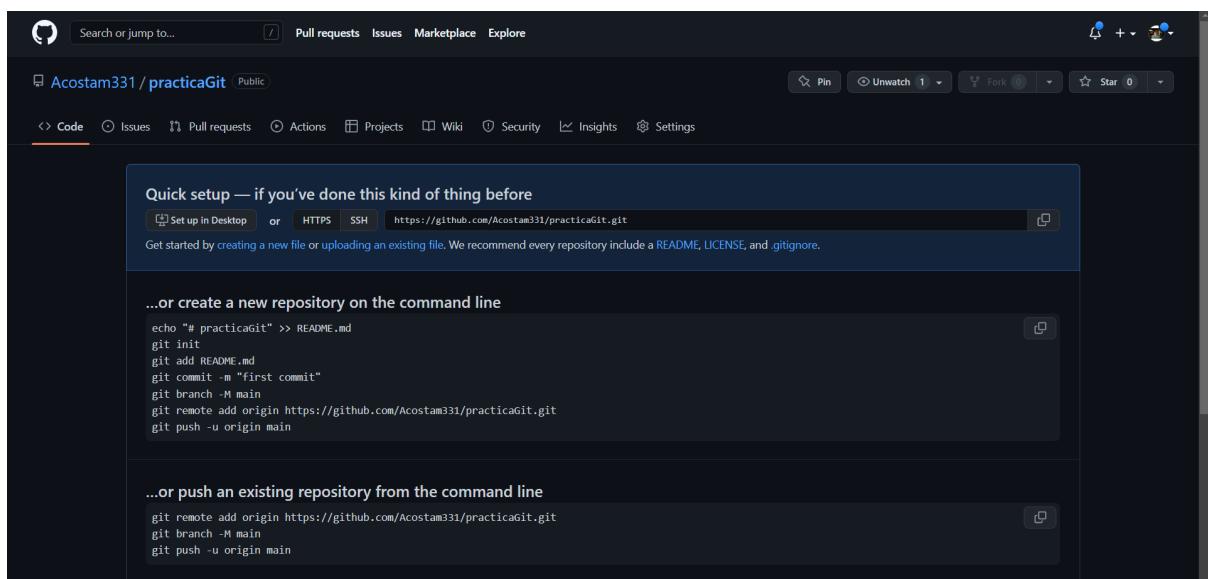


1.Crear un nuevo repositorio en github utilizando la interfaz gráfica.

Para crear un repositorio desde github debemos de seleccionar la opción *nuevo repositorio*. Posteriormente tendremos que seleccionar un nombre para nuestro repositorio en este caso *practicaGit*, y las demás opciones permanecerán por defecto.

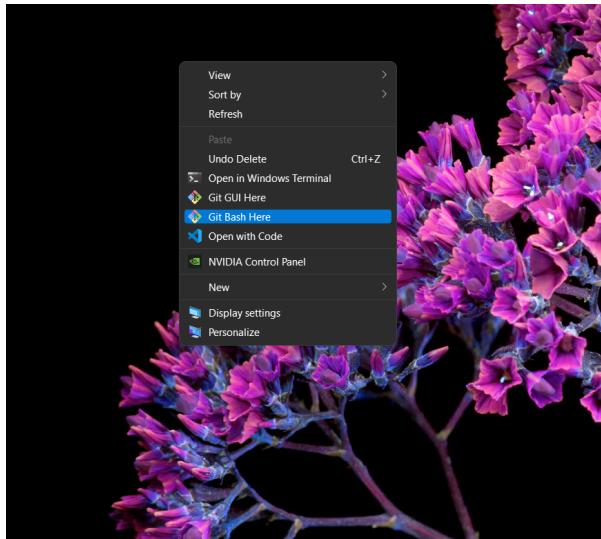


Cuando el repositorio haya sido creado aparecerá de la siguiente forma ya que es un repositorio vacío y debe ser inicializado.



2.Clonar el repositorio y ubicarnos en el directorio creado usando la terminal de git.

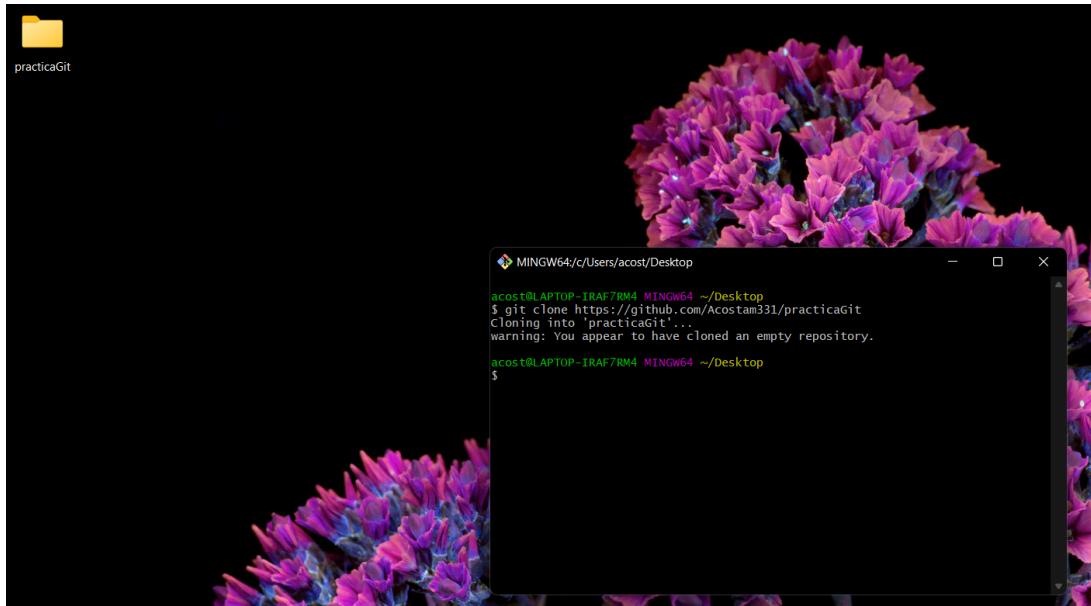
Para clonar el repositorio primero debemos definir el directorio en donde queremos que se cree la carpeta de trabajo con nuestro repositorio, en este caso utilizaremos el escritorio. en el fichero seleccionado con click derecho seleccionaremos la opción *git bash here* este comando abrirá una terminal de git que apunta al fichero actual (escritorio).



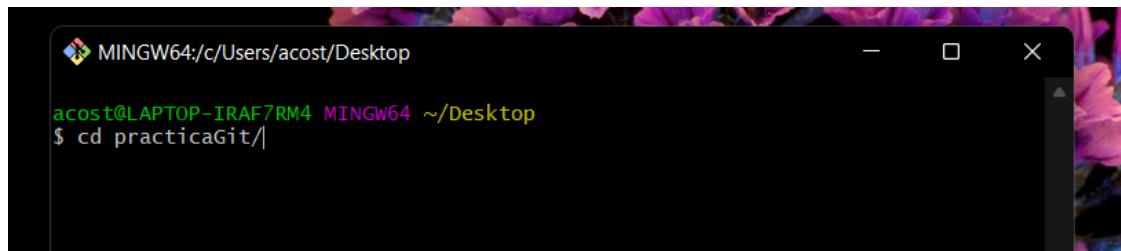
Utilizando el comando *git clone [URL]*, clonaremos nuestro repositorio.

A screenshot of a terminal window titled "MINGW64/c/Users/acost/Desktop". The window shows the command "git clone https://github.com/Acostam331/practicaGit" being typed at the prompt. The background of the terminal window is a dark image of purple flowers.

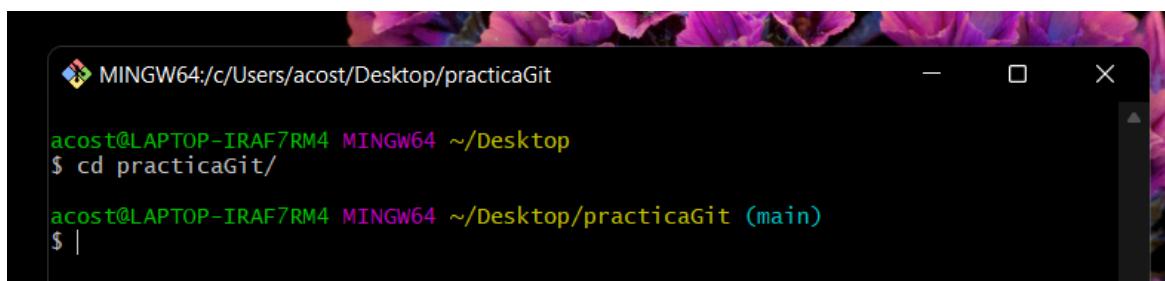
Si la operación fue exitosa podremos observar que se agregó una carpeta con el nombre del repositorio. En este caso también notaremos una advertencia en la terminal de git, pero solo nos indica que hemos clonado un repositorio vacío, más adelante lo inicializamos realizando nuestro primer commit.



Si bien ya clonamos nuestro repositorio nuestra terminal de git sigue apuntando al escritorio por lo que debemos navegar hacia la carpeta que tiene el nombre de nuestro repositorio, en esta carpeta agregaremos todo nuestro trabajo. Para movernos hacia ella utilizaremos el comando **cd [nombre de carpeta]**.

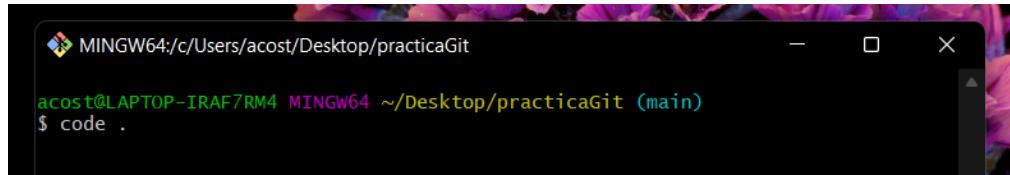


Cuando hayamos cambiado de directorio podremos observar la ruta actual en la terminal, esta se muestra de color amarillo, ademas podemos verificar que la carpeta en cuestión si corresponde a una carpeta que trabaja con git porque nos indica la rama de trabajo actual esta se muestra de color celeste en este caso es *main*.



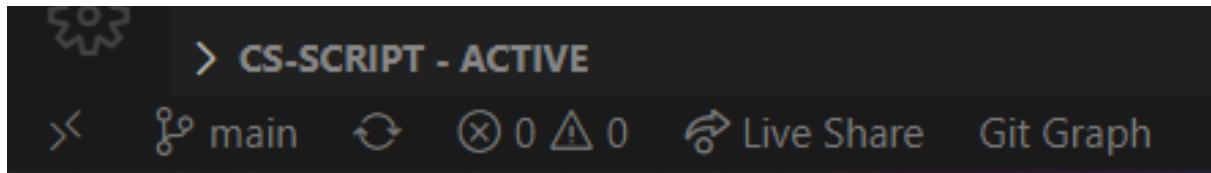
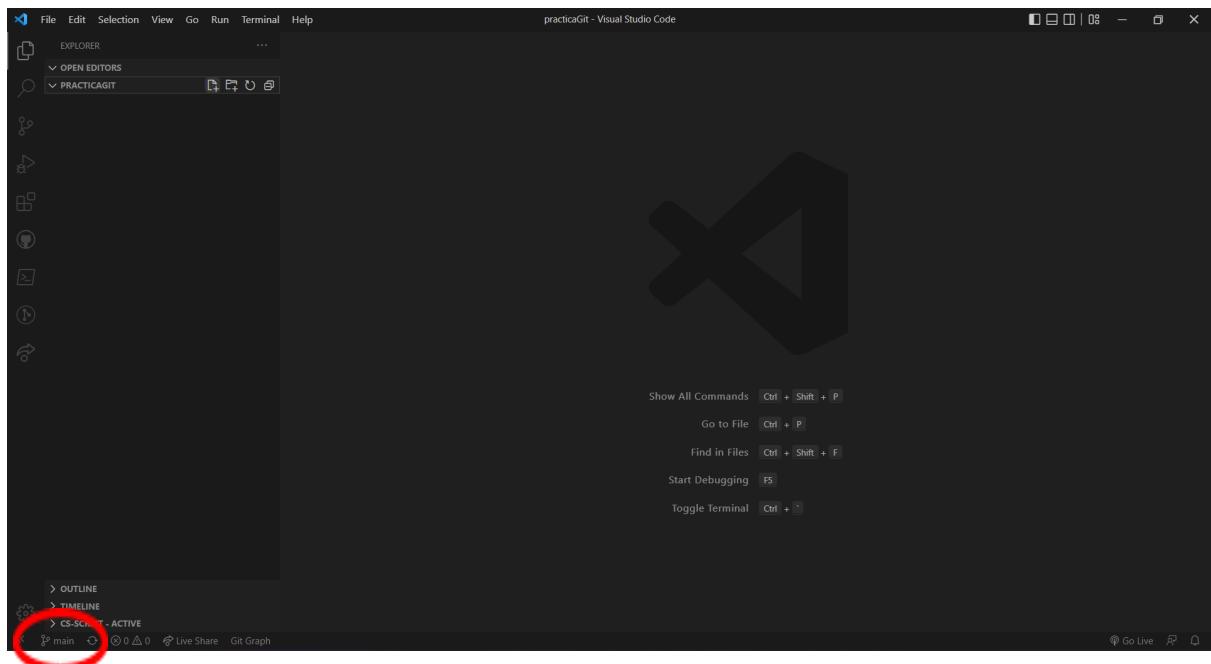
3.Abrir el directorio creado en visual studio code.

Para abrir visual studio code trabajando sobre la carpeta que deseamos (la carpeta de nuestro repositorio) desde la terminal utilizaremos el comando: code .



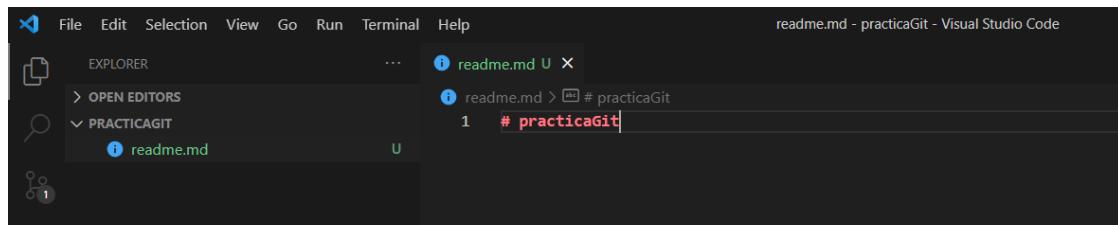
```
MINGW64:/c/Users/acost/Desktop/practicaGit
acost@LAPTOP-IRAF7RM4 MINGW64 ~/Desktop/practicaGit (main)
$ code .
```

Después de utilizar el comando anterior se abrirá visual studio code y podremos verificar que estamos trabajando en la carpeta deseada porque en el panel izquierdo aparecerá el nombre de la misma. de igual forma en la esquina inferior izquierda nos indica en qué rama estamos trabajando en este caso (main).



4. Inicializar el repositorio con un archivo *readme.md*

Para crear y publicar la rama **main**, es necesario hacer un commit inicial, en este agregaremos un archivo *readme.md*, el cual tendrá el nombre de nuestro repositorio. Para agregar un archivo a la carpeta podemos utilizar los atajos de VS Code o utilizando click derecho.



Una vez creado el archivo *readme.md*, se procede a crear un primer commit, esta es una operación que se hace una única vez en la rama **main** usando los comandos:

```
git add .
```

A screenshot of a terminal window titled 'MINGW64:c/Users/acost/Desktop/practicaGit'. The prompt shows 'acost@LAPTOP-IRAF7RM4 MINGW64 ~/Desktop/practicaGit (main)'. The user has typed '\$ git add .' and is awaiting the command's execution.

Para generar un primer comentario utilizaremos el comando:

```
git commit -m "First commit"
```

A screenshot of a terminal window titled 'MINGW64:c/Users/acost/Desktop/practicaGit'. The prompt shows 'acost@LAPTOP-IRAF7RM4 MINGW64 ~/Desktop/practicaGit (main)'. The user has typed '\$ git add .' on the previous line and '\$ git commit -m "First commit"' on the current line, and is awaiting the command's execution.

y finalmente el comando: `git push origin main`, con el cual se realizará el primer push a nuestro repositorio.

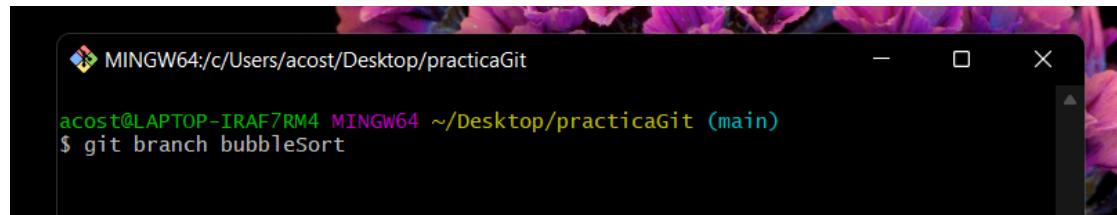
A screenshot of a terminal window titled 'MINGW64:c/Users/acost/Desktop/practicaGit'. The prompt shows 'acost@LAPTOP-IRAF7RM4 MINGW64 ~/Desktop/practicaGit (main)'. The user has typed '\$ git add .' on the previous line, '\$ git commit -m "First commit"' on the previous line, and '\$ git push origin main' on the current line. The output shows the commit message '[main (root-commit) 0475631] First commit' and '1 file changed, 1 insertion(+) create mode 100644 readme.md'. The final prompt shows 'acost@LAPTOP-IRAF7RM4 MINGW64 ~/Desktop/practicaGit (main)'.

Una vez finalizado, en nuestro repositorio se puede visualizar la rama **main** junto con el archivo **readme.md** publicado en el primer commit.

The screenshot shows a GitHub repository page for 'Acostam331 / practicaGit'. The repository is public and contains one branch ('main') and one commit ('First commit'). The commit was made by 'Acostam331' one minute ago and includes a file named 'readme.md' with the content 'practicaGit'. The repository has 0 stars, 1 watching, and 0 forks. The 'About' section indicates 'No description, website, or topics provided.' The 'Packages' section shows 'No packages published' and 'Publish your first package'. The bottom navigation bar includes links for Terms, Privacy, Security, Status, Docs, Contact GitHub, Pricing, API, Training, Blog, and About.

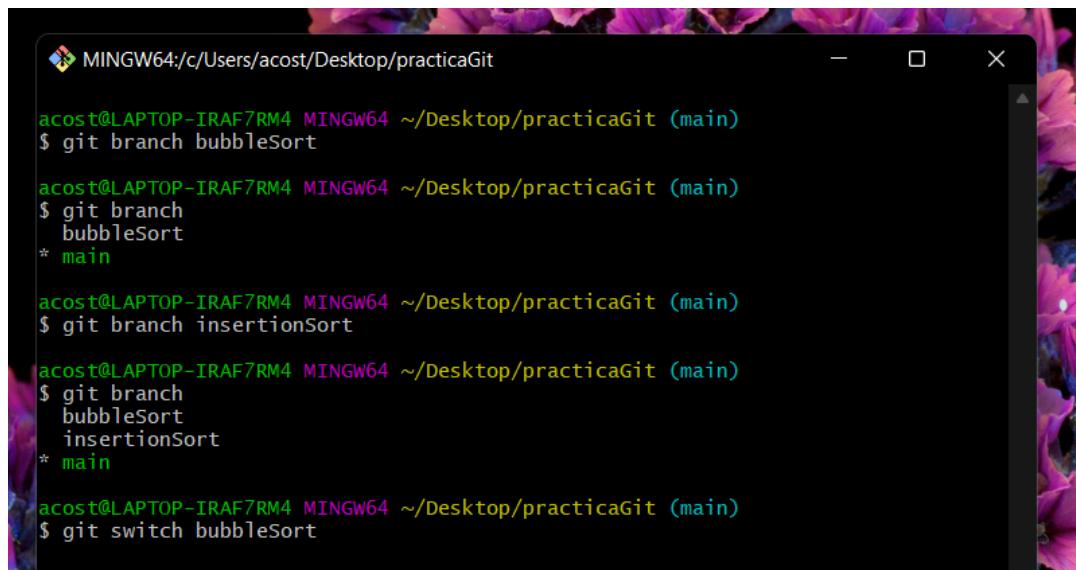
5.Crear dos ramas, una llamada **bubbleSort** y la otra **insertionSort**.

En la terminal de git, crearemos una rama llamada: **bubbleSort**.



```
MINGW64:/c/Users/acost/Desktop/practicaGit
acost@LAPTOP-IRAF7RM4 MINGW64 ~/Desktop/practicaGit (main)
$ git branch bubbleSort
```

Con el comando: `git branch`, podemos listar todas las ramas que hemos creado. Posteriormente crearemos la siguiente rama llamada: **insertionSort**. Luego nos colocaremos sobre la rama **bubbleSort** utilizando el comando:
`git switch bubbleSort`



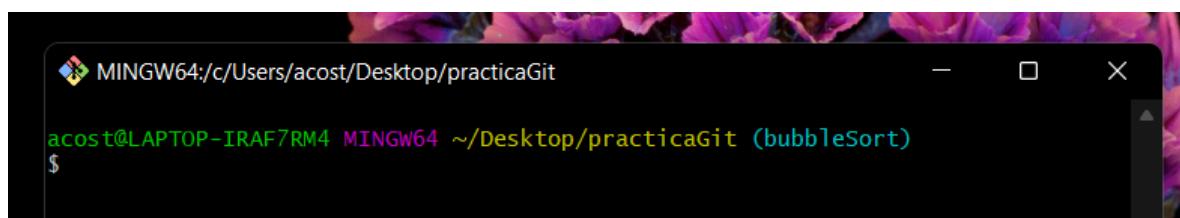
```
MINGW64:/c/Users/acost/Desktop/practicaGit
acost@LAPTOP-IRAF7RM4 MINGW64 ~/Desktop/practicaGit (main)
$ git branch bubbleSort
* main

acost@LAPTOP-IRAF7RM4 MINGW64 ~/Desktop/practicaGit (main)
$ git branch insertionSort

acost@LAPTOP-IRAF7RM4 MINGW64 ~/Desktop/practicaGit (main)
$ git branch
  bubbleSort
  insertionSort
* main

acost@LAPTOP-IRAF7RM4 MINGW64 ~/Desktop/practicaGit (main)
$ git switch bubbleSort
```

Cuando hayamos cambiado de rama notaremos en la terminal que el indicador azul que muestra en qué rama estamos trabajando tendrá como valor: (**bubbleSort**)



```
MINGW64:/c/Users/acost/Desktop/practicaGit
acost@LAPTOP-IRAF7RM4 MINGW64 ~/Desktop/practicaGit (bubbleSort)
$
```

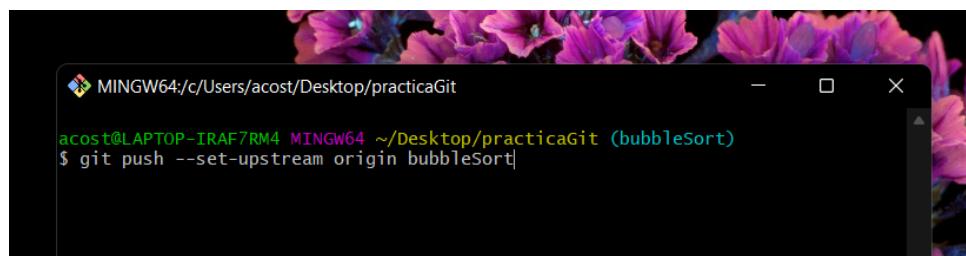
De igual forma podemos observar en VS code que la rama de trabajo cambió a **bubbleSort**, esto lo podemos verificar en la esquina inferior izquierda.



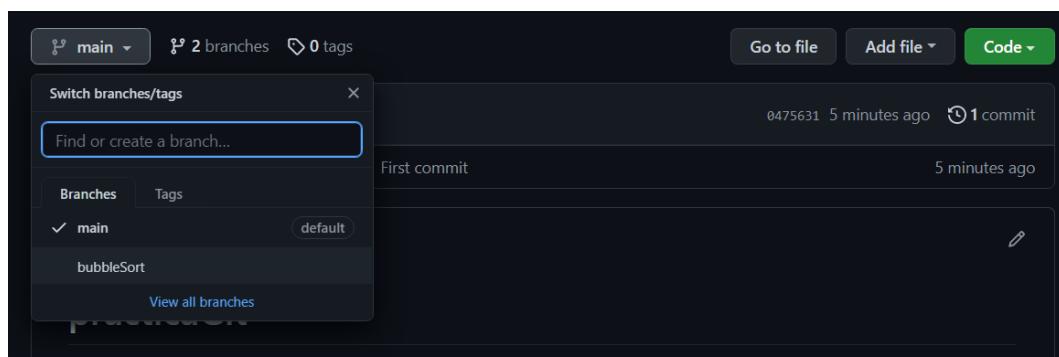
6.Movernos entre las ramas anteriormente creadas y publicarlas en github.

Una forma de publicar una rama, es haciendo uso del comando:
git push --set-upstream origin [rama]

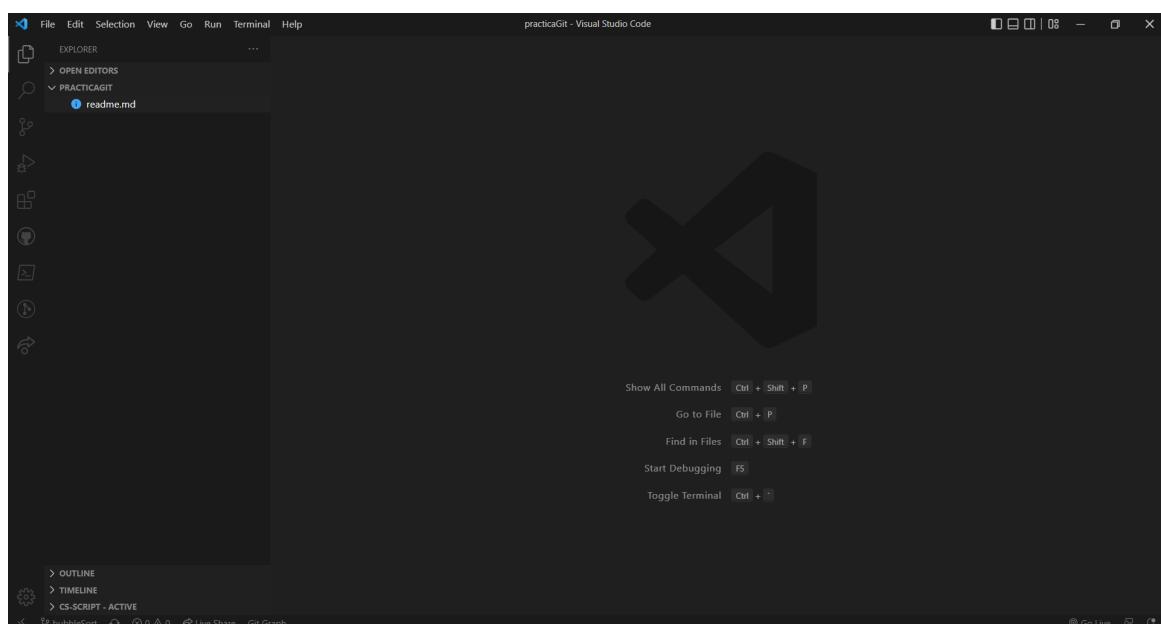
Con esto, git establece la rama en la que haces push para que pueda “rastrear” la rama en el repositorio remoto. Añadir esto significa que git sabe en qué rama remota quieres trabajar cuando haces “push” en un futuro.



Esto creará una rama dentro de nuestro repositorio de github.



Podemos observar que siempre que creamos una rama esta heredará el contenido que en ese momento tenga la rama principal.

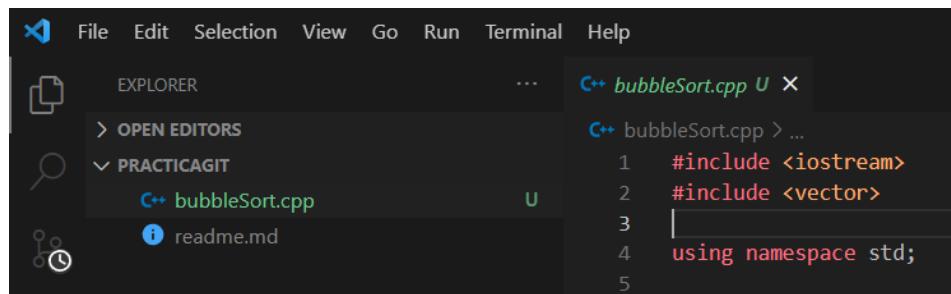


Repetir este procedimiento para la rama **insertionSort**. Es decir, primero nos movemos hacia esa rama y después la publicamos.

7.Trabajando en la rama *BubbleSort*. Agregar un archivo llamado *bubbleSort.cpp*.

Primero debemos verificar que nos encontremos trabajando en la rama ***bubbleSort***, en el caso que estemos en una rama distinta podemos ubicarnos en esta rama utilizando el comando: `git switch bubbleSort`

Agregaremos un archivo ***bubbleSort.cpp***, el archivo puede NO CONTENER NADA, ya que el objetivo de esta práctica es trabajar y familiarizarse con git.



8.Subir el archivo creado a github (siempre trabajando en la rama *BubbleSort*).

Utilizando el comando `git status`, podemos ver el estado de los ficheros de nuestro entorno de trabajo. En este caso observamos que el archivo ***bubbleSort.cpp***. Aún no ha sido agregado a git y por lo tanto de momento no puede seguir los cambios o modificaciones que se realicen en el mismo.

De igual manera este comando nos puede ayudar a identificar qué archivos han sido modificados, agregados o eliminados antes de realizar un commit.

A screenshot of a terminal window titled 'MINGW64:/c/Users/acost/Desktop/practicaGit'. The command \$ git status is run, showing:

```
acost@LAPTOP-IRAF7RM4 MINGW64 ~/Desktop/practicaGit (bubbleSort)
$ git status
On branch bubbleSort
Your branch is up to date with 'origin/bubbleSort'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    bubbleSort.cpp

nothing added to commit but untracked files present (use "git add" to track)

acost@LAPTOP-IRAF7RM4 MINGW64 ~/Desktop/practicaGit (bubbleSort)
$ |
```

The terminal window has a dark theme with a purple floral background image.

Antes de realizar un commit, recordar que se recomienda utilizar diferentes emojis de la página <https://gitmoji.dev/> para aportar significado a nuestros commits. De igual manera, sugerimos investigar sobre las buenas prácticas al momento de realizar nuestros commits.



Para agregar todos nuestros archivos a un commit utilizamos el comando:

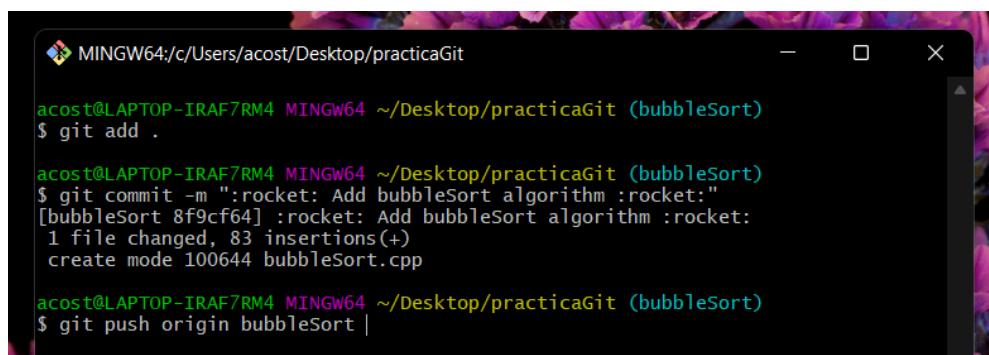
```
git add .
```

Después realizamos el commit y agregamos un comentario con el comando:

```
git commit -m "Add bubbleSort algorithm :rocket:"
```

Finalmente publicamos el commit en nuestro repositorio con el comando:

```
git push origin bubbleSort
```

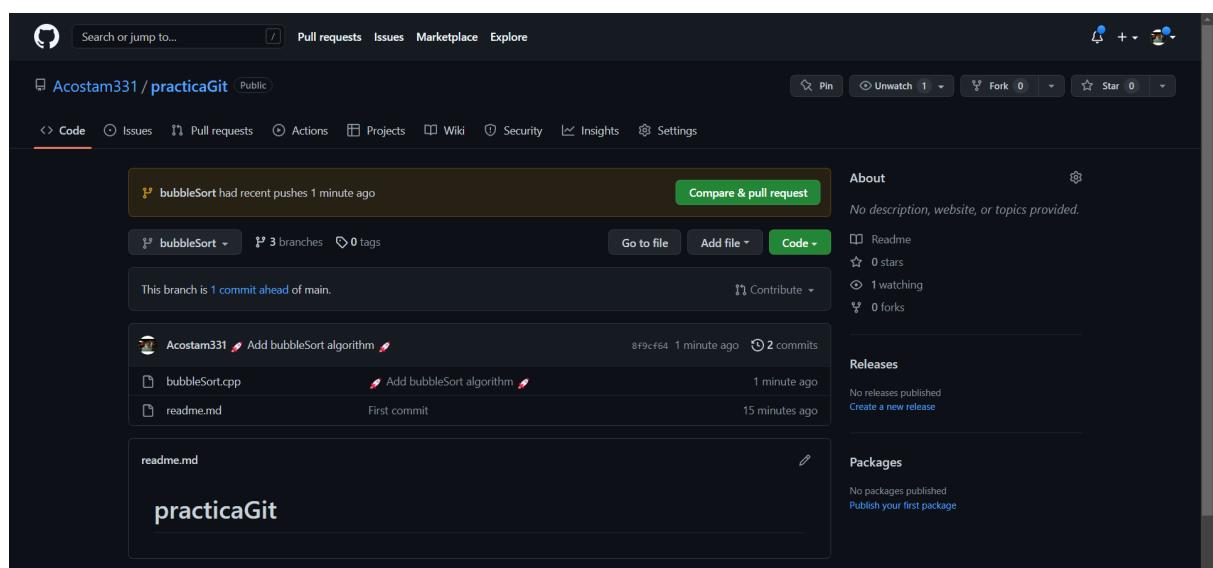


```
MINGW64:/c/Users/acost/Desktop/practicaGit
$ git add .

MINGW64:/c/Users/acost/Desktop/practicaGit (bubbleSort)
$ git commit -m ":rocket: Add bubbleSort algorithm :rocket:"
[bubbleSort 8f9cf64] :rocket: Add bubbleSort algorithm :rocket:
 1 file changed, 83 insertions(+)
 create mode 100644 bubbleSort.cpp

MINGW64:/c/Users/acost/Desktop/practicaGit (bubbleSort)
$ git push origin bubbleSort |
```

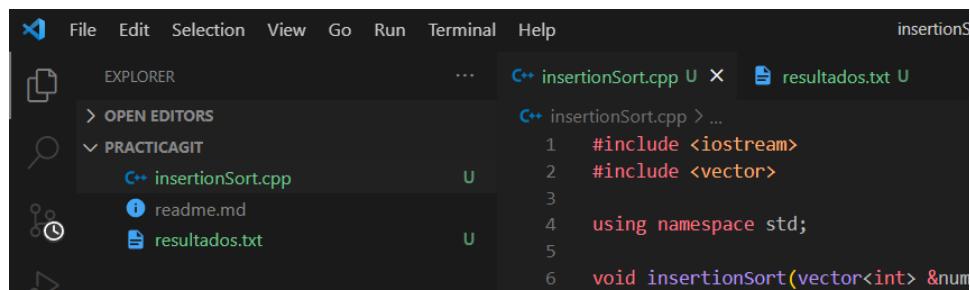
Si después verificamos nuestro repositorio de github en la rama **bubbleSort**, podremos observar nuestro commit con el archivo **bubbleSort.cpp**.



9.Trabajando en la rama *InsertionSort*. Agregar un archivo llamado *insertionSort.cpp* y un archivo de texto *resultados.txt*.

Primero debemos verificar que nos encontremos trabajando en la rama *insertionSort*, en el caso que estemos en una rama distinta podemos ubicarnos en esta rama utilizando el comando: `git switch insertionSort`

Notaran que el archivo ***bubbleSort.cpp*** ha desaparecido, esto se debe a que este archivo no forma parte de la rama en la que estamos trabajando actualmente. Despues, Agregaremos los archivos *insertionSort.cpp* y *resultados.txt*



```
#include <iostream>
#include <vector>
using namespace std;
void insertionSort(vector<int> &num)
```

10.Publicar los archivos anteriormente creados en github (siempre trabajando en la rama *InsertionSort*).

Para agregar todos nuestros archivos a un commit utilizamos el comando:

```
git add .
```

Después realizamos el commit y agregamos un comentario con el comando:

```
git commit -m "Add insertionSort algorithm :white_check_mark:"
```

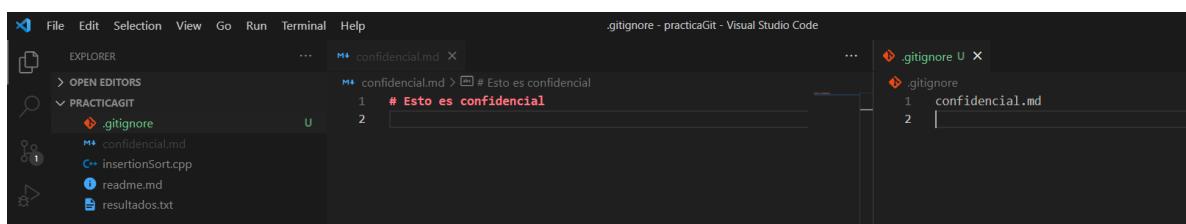
Finalmente publicamos el commit en nuestro repositorio con el comando:

```
git push origin insertionSort
```

11.Agregar un archivo *confidencial.txt* y un archivo *.gitignore*.

12.Utilizando el archivo *.gitignore* ignorar el archivo *confidencial.txt* para que este no sea publicado al repositorio de github.

Al agregar un archivo *.gitignore*, le decimos a github que ignore o no tome en cuenta los archivos o carpetas listados dentro del mismo, en este ejemplo se puede observar como el archivo *confidencial.txt* aparece oscurecido, lo cual es una indicación de que git ignorara ese archivo, carpeta o directorio al momento de publicar un commit.



```
# Esto es confidencial
```

```
confidencial.md
```

13. Publicar los cambios realizados y verificar que *confidencial.txt* no haya sido publicado solamente el archivo *.gitignore*.

Deberá realizar un commit para publicar los cambios realizados al repositorio de github. Este proceso ya ha sido explicado con anterioridad en esta guía.

Al observar la página de git referente a la rama de trabajo, observamos como el archivo *confidencial.txt* no se encuentra, confirmando la funcionalidad de *.gitignore*

The screenshot shows a GitHub repository page for a project named 'insertionSort'. At the top, there are two notifications: one from 'bubbleSort' and another from 'insertionSort'. Below the notifications, the repository summary shows 3 branches and 0 tags. There are buttons for 'Go to file', 'Add file', and 'Code'. A message indicates the branch is 2 commits ahead of 'main'. The commit history lists three commits:

- Acostam331 Add .gitignore (10 seconds ago)
- insertionSort.cpp add insertion sort algorithm (5 minutes ago)
- readme.md First commit (29 minutes ago)
- resultados.txt add insertion sort algorithm (5 minutes ago)

The commit 'Add .gitignore' is highlighted. In the bottom right corner of the commit list, there is a note: 'practicaGit'.

About
No description, website, or topics provided.

Readme
0 stars
1 watching
0 forks

Releases
No releases published
Create a new release

Packages
No packages published
Publish your first package

14. Revisar en el repositorio de github que ambas ramas contienen archivos diferentes.

Revisando la rama **bubbleSort** se puede observar como solo existe un archivo **bubbleSort.cpp**

This branch is 4 commits behind main.

- Acostam331 Add bubbleSort algorithm
- bubbleSort.cpp Add bubbleSort algorithm
- readme.md First commit

8f9cf64 yesterday 2 commits

yesterday

yesterday

practicaGit

Revisando la rama **insertionSort** se puede observar como existen dos archivos totalmente distintos, las ramas de github trabajan de forma individual, por lo que el flujo de trabajo es independiente el uno del otro, sin interferir entre sí.

This branch is 3 commits behind main.

- Acostam331 Add gitignore
- .gitignore Add gitignore
- insertionSort.cpp add insertion sort algorithm
- readme.md First commit
- resultados.txt add insertion sort algorithm

26a1278 yesterday 3 commits

yesterday

yesterday

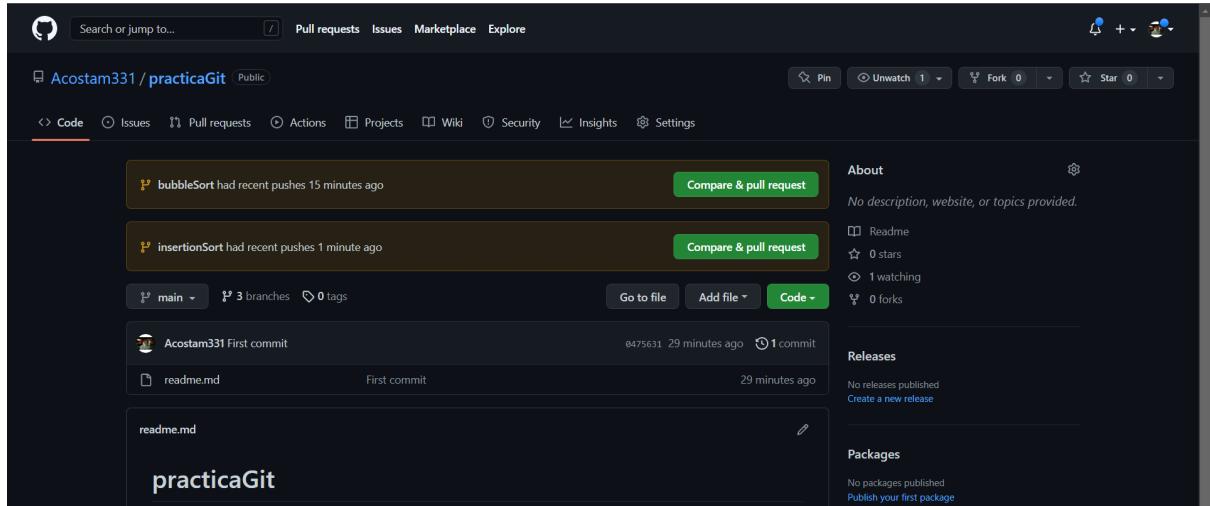
yesterday

practicaGit

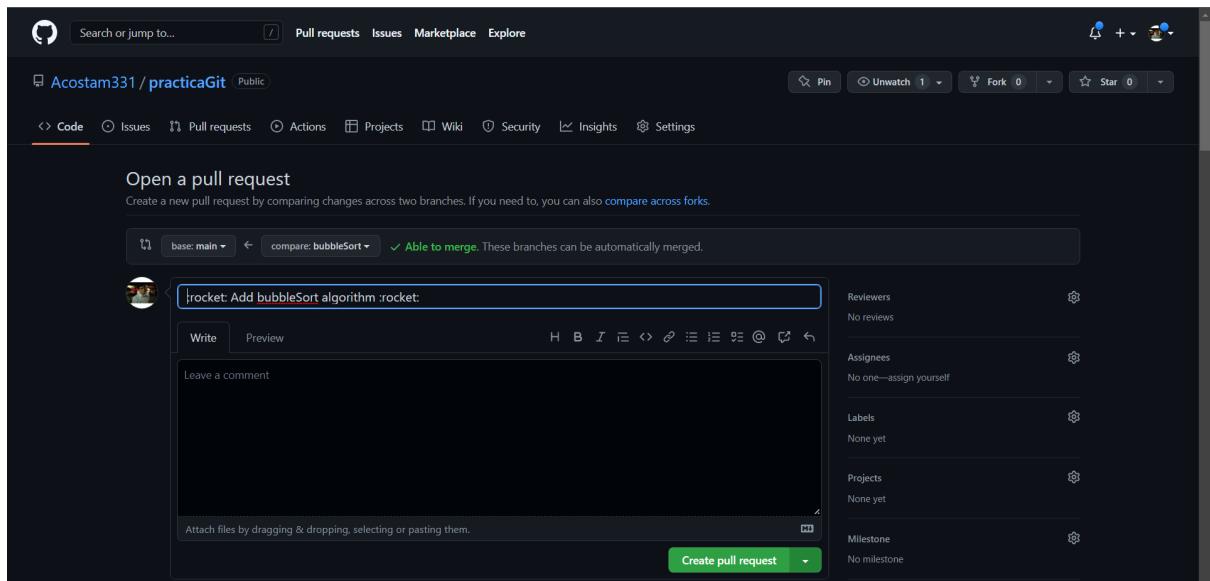
15. Hacer merge de las dos ramas (*BubbleSort* y *InsertionSort*) con la rama principal o (*main*) Utilizando la interfaz gráfica de github.

Realizar merge, significa combinar el contenido de dos ramas en una sola, en este caso agregaremos el contenido de las ramas ***bubbleSort*** y ***insertionSort*** a la rama principal o ***main***.

Como podemos observar, github nos muestra que diferentes ramas pueden ser comparadas y combinadas. Primero seleccionaremos *compare & pull request* para la rama ***bubbleSort***.

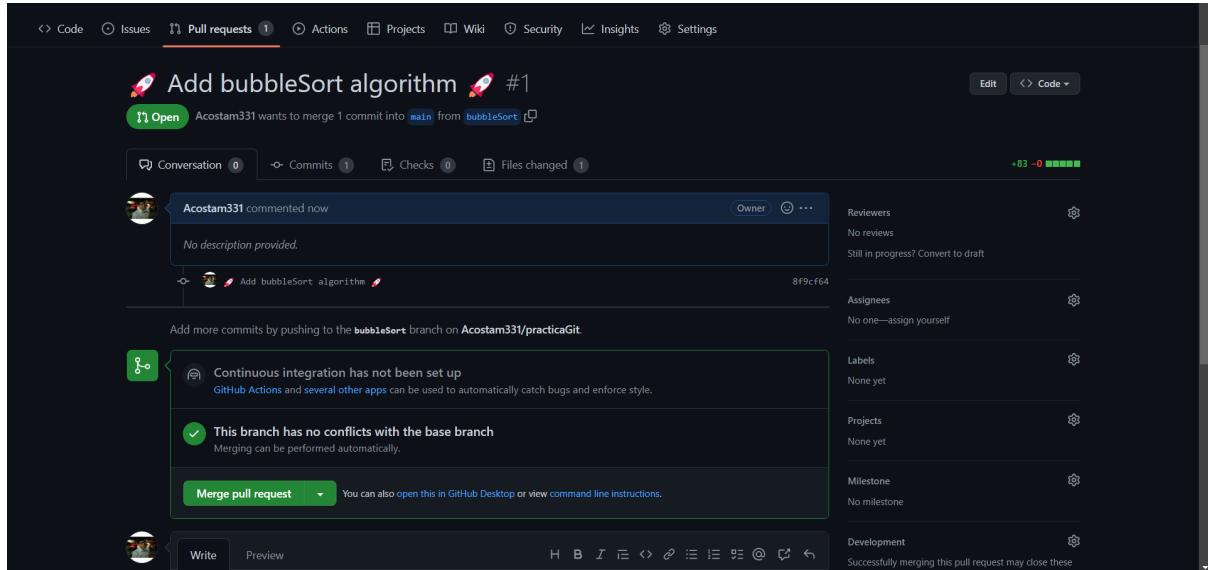


Se mostrará el siguiente formulario en donde crearemos una *pull request*, en trabajos colaborativos esta solicitud deberá ser revisada por los demás integrantes del equipo.

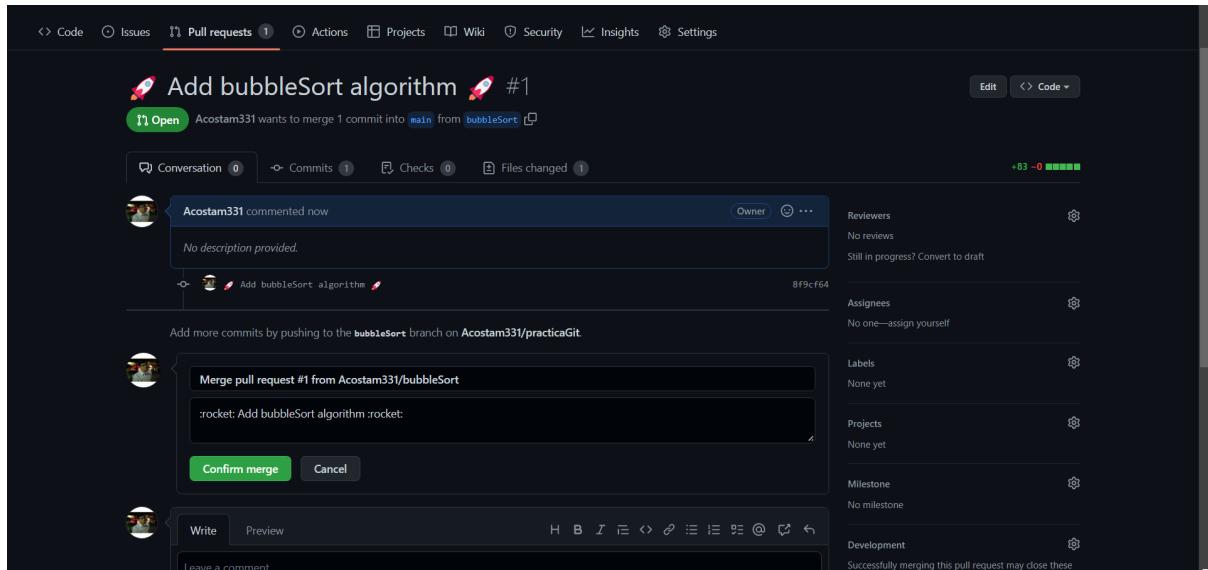


Posteriormente en la siguiente ventana podremos ver diferentes comentarios relacionados con la solicitud que realizamos. De igual manera, git nos indicará si nuestro trabajo puede ser combinado sin problema con la rama principal o si debemos solucionar algún conflicto primero.

En este caso, como la rama principal no tiene el archivo **bubbleSort.cpp** no habrá ningún conflicto que resolver.



Finalmente confirmamos el merge y podremos observar que los archivos y los commits de la rama secundaria **BubbleSort** ahora se encuentran también en la rama principal o **main**.



Estos pasos se deben de repetir con la rama ***InsertionSort***.

El resultado final debe ser la rama ***main*** con el contenido de ambas ramas secundarias.

The screenshot shows a GitHub repository page for 'Acostam331 / practicaGit'. The repository is public and has 3 branches and 0 tags. The 'Code' tab is selected. The main branch contains 6 commits from 'Acostam331' merging pull request #2 from 'Acostam331/insertionSort'. The commits are:

- .gitignore: Add gitignore (2 minutes ago)
- bubbleSort.cpp: Add bubbleSort algorithm (17 minutes ago)
- insertionSort.cpp: add insertion sort algorithm (7 minutes ago)
- readme.md: First commit (31 minutes ago)
- resultados.txt: add insertion sort algorithm (7 minutes ago)

The 'About' section indicates no description, website, or topics provided. It shows 0 stars, 1 watching, and 0 forks. The 'Releases' section shows no releases published, with a link to 'Create a new release'. The 'Packages' section shows no packages published, with a link to 'Publish your first package'.

16.Moverse a la rama *main* utilizando la terminal. Notarán que los archivos no existen en dicha rama.

Utilizando la terminal de git, nos movemos a la rama principal utilizando el comando:

```
git switch main
```

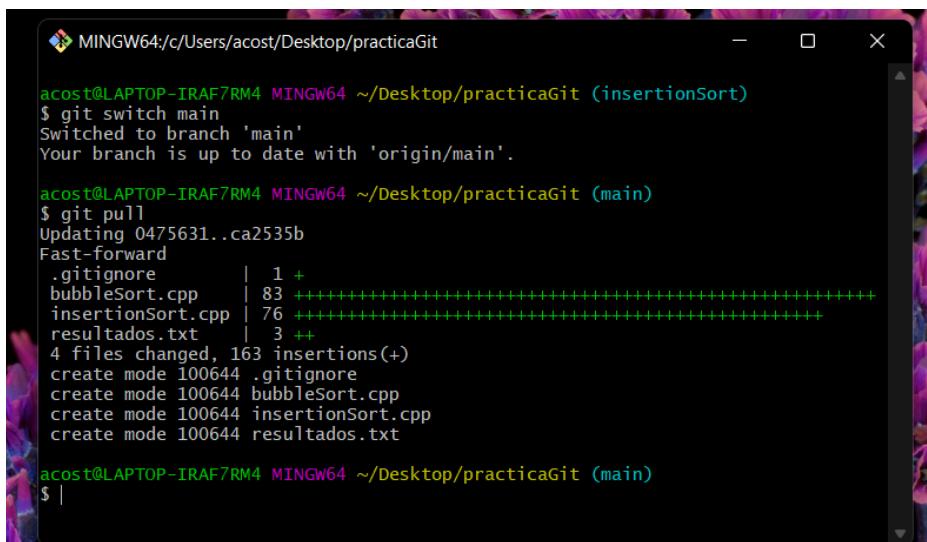
Al moverse a la rama *main* por medio de la terminal, se puede observar que no contamos con los archivos de las ramas secundarias, en este caso es necesario descargarlos desde el repositorio haciendo uso de la terminal de git.

17.Realizar un *pull* de la rama *main* para actualizar los archivos locales con los cambios que contiene el repositorio de github.

Es necesario realizar una descarga o actualización de los archivos locales (ya que queremos los archivos que contiene la rama *main* en el repositorio de github) por medio de la consola, para ello, se utilizará el comando:

```
git pull
```

El cual realizará una verificación y actualización de los archivos locales con respecto a los almacenados en el repositorio

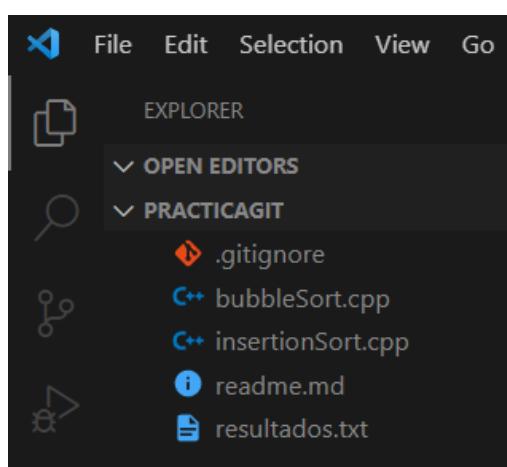


```
MINGW64:/c/Users/acost/Desktop/practicaGit
acost@LAPTOP-IRAF7RM4 MINGW64 ~/Desktop/practicaGit (insertionSort)
$ git switch main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

acost@LAPTOP-IRAF7RM4 MINGW64 ~/Desktop/practicaGit (main)
$ git pull
Updating 0475631..ca2535b
Fast-forward
 .gitignore      |  1 +
 bubbleSort.cpp  | 83 ++++++=====
 insertionSort.cpp| 76 ++++++=====
 resultados.txt  |  3 ++
 4 files changed, 163 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 bubbleSort.cpp
 create mode 100644 insertionSort.cpp
 create mode 100644 resultados.txt

acost@LAPTOP-IRAF7RM4 MINGW64 ~/Desktop/practicaGit (main)
$ |
```

Una vez realizada la actualización, se puede observar como los archivos locales se modificaron (agregando los archivos faltantes), mostrando el contenido actualizado de la rama *main* luego del merge.



Resumen de comandos

Clonar un repositorio:

```
git clone [url del repositorio]
```

Crear ramas

```
git branch [rama]
```

Cambiar de rama seleccionada

```
git switch [rama]
```

Publicar rama

```
git --set-upstream origin [rama]
```

También es posible cambiarse de ramas con el comando *checkout*, pero el comando de *switch* fue específicamente creado para este propósito, puesto que *checkout* tiene una variedad de funciones más que no eran muy amigables para el usuario que empezaba a usar git.

Para **crear una rama y cambiarse a la rama recién creada**, hacemos el uso de una bandera (en inglés “flag”, que significa que modificaremos la operación del comando a utilizar. También conocidas como “opciones”):

```
git switch -c [rama]
```

Publicar rama forma alternativa:

Cuando ya se ha creado una rama y estamos trabajando en ella, podemos publicarla realizando un commit y push (a la rama) del trabajo que hayamos agregado, git la creará de manera automática en github.

Verificar el estado de los ficheros

```
git status
```

Publicar cambios realizados

```
git add .
```

```
git commit -m "Este es el comentario del commit"
```

```
git push origin [rama]
```

Actualizar el contenido asociado a una rama, descargando los cambios que contiene el repositorio

```
git pull
```

Nota: Hay algunos comandos que utilizan corchetes con la finalidad de representar el nombre de alguna variable que necesita el comando. Esta tiene que ser determinada por el usuario, lo que significa que ***no deben escribirse los corchetes de manera explícita.***