

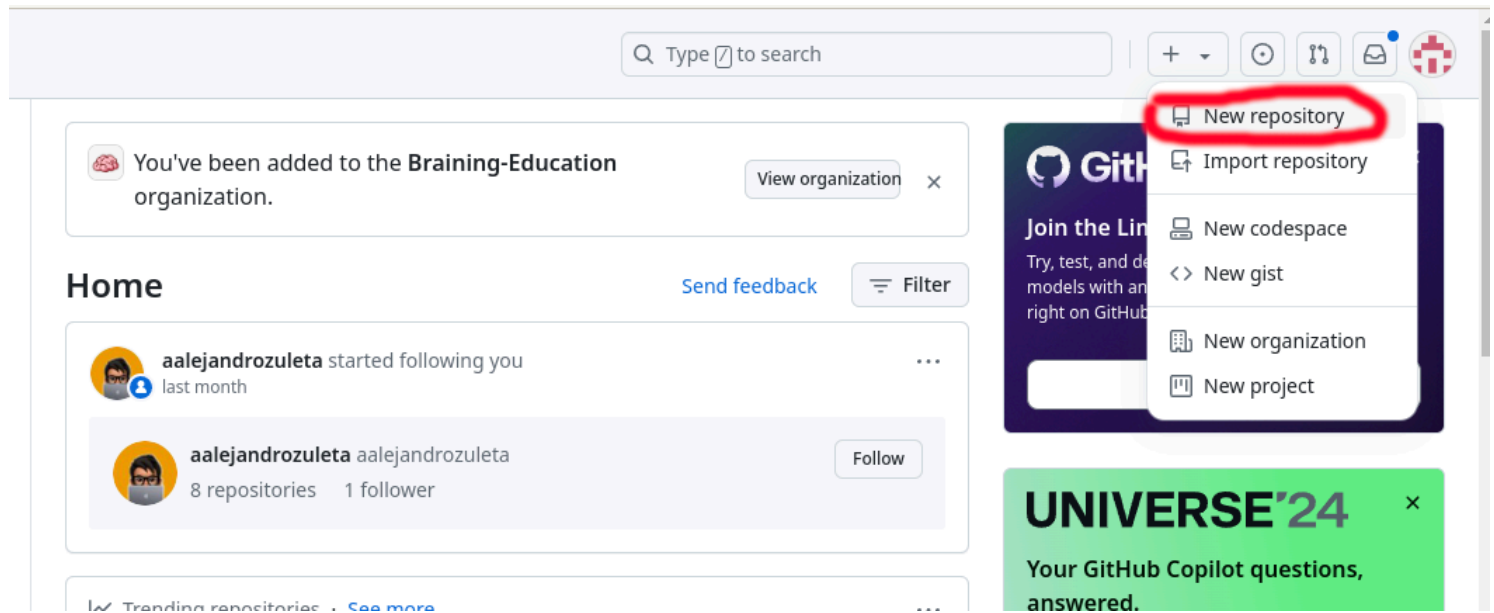
## PRÁCTICA: GIT/GITHUB Y JAVASCRIPT

### REQUISITOS:

1. CUENTA DE <https://github.com/>
2. Git instalado en su equipo. Se puede descargar desde: <https://git-scm.com/downloads>
3. Visual Studio Code con un proyecto Javascript
4. Resumen de comandos de Git

1. Iniciar sesión en su cuenta de Github

2. Crear un nuevo repositorio




3. Asignamos el nombre al repositorio, lo crearemos privado para que sólo nosotros y nuestros colaboradores tengan acceso a él

## Create a new repository

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

Required fields are marked with an asterisk (\*).

Owner \*

 gangaritah ▾

Repository name \*

primer-repo

✓ primer-repo is available.

Great repository names are short and memorable. Need inspiration? How about [curly-potato](#) ?

Description (optional)

☐

**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.

Initialize this repository with:

☐

**Add a README file**

This is where you can write a long description for your project. [Learn more about READMEs](#).

Add .gitignore

.gitignore template: None ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files](#).

Choose a license

License: None ▾

A license tells others what they can and can't do with your code. [Learn more about licenses](#).

 You are creating a private repository in your personal account.

Create repository

#### 4. Ya tenemos nuestro primer repositorio en Github

The screenshot shows the GitHub interface for a repository named 'primer-repo'. At the top, there's a navigation bar with links for Pull requests, Actions, Projects, Security, Insights, and Settings. Below this, the repository name 'primer-repo' is displayed with a 'Private' badge. To the right, there are buttons for 'Unwatch' (1), 'Fork' (0), and 'Star' (0). The main content area has two cards: 'Set up GitHub Copilot' and 'Add collaborators to this repository'. Below these is a 'Quick setup' section with a text input field containing the repository URL 'https://github.com/gangaritah/primer-repo.git'. Underneath, it says 'Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).' Further down, there are two sections for command-line setup: '...or create a new repository on the command line' and '...or push an existing repository from the command line', each with a code block and a copy icon.

```
echo "# primer-repo" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/gangaritah/primer-repo.git
git push -u origin main
```

```
git remote add origin https://github.com/gangaritah/primer-repo.git
```

5. Ahora crearemos nuestro proyecto Javascript sobre el cual implementaremos el repositorio anterior. Debe contener un archivo `index.js` que tiene una función `name` y una carpeta `reportes` con un archivo `reporte.txt` el cual puede estar vacío para efectos del ejemplo. Esta carpeta la ignoraremos más adelante para que git no le haga seguimiento, ya que contiene reportes cuya importancia solo es en local.

The screenshot shows a code editor interface. On the left, the 'EXPLORER' sidebar shows a file tree with a folder named 'REPO' containing a subfolder 'reportes' (with a file 'reporte.txt') and a file 'index.js'. The 'index.js' file is selected. The main editor area shows the content of 'index.js', which contains a function definition:

```
1 function name(params) {
2   console.log("HOLA SENA");
3 }
4
```

En este punto tenemos la estructura inicial del proyecto, por tanto, debemos generar nuestro repositorio git a nivel local y configurarlo

6. Podemos manejar nuestro repo git desde la terminal de Visual Studio o si queremos abrimos una consola aparte como Git CMD o PowerShell en Windows, una terminal si se está en Linux. En este caso trabajaremos con la terminal por defecto de VS Code.

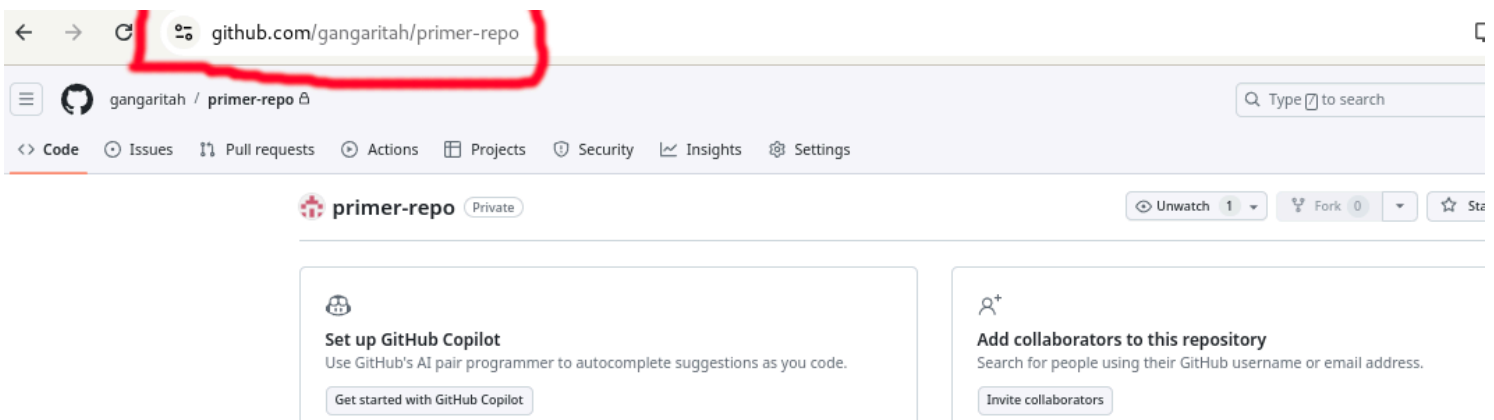
Antes de crear nuestro repositorio debemos verificar que la configuración de nuestro Git sea correcta configuramos nuestro email, name y pedimos a git que cuando iniciemos sesión en un repositorio remoto, guarde las credenciales por nosotros, esto lo hacemos con `git config --global credential.helper store`. También configuramos la manera en que trabajará el nuestro comando `git pull` (lo usaremos más tarde en otras prácticas), esto lo hacemos usando `git config --global pull.rebase false` para que el comando `git pull` haga fusiones de manera predeterminada.

Iniciamos el repo

```
german@fedora:~/Documentos/CodeBase/Git/repo$ git init
hint: Usando 'master' como el nombre de la rama inicial. Este nombre de rama predeterminado
hint: está sujeto a cambios. Para configurar el nombre de la rama inicial para usar en todos
hint: de sus nuevos repositorios, reprimiendo esta advertencia, llama a:
hint:
hint:   git config --global init.defaultBranch <nombre>
hint:
hint: Los nombres comúnmente elegidos en lugar de 'master' son 'main', 'trunk' y
hint: 'development'. Se puede cambiar el nombre de la rama recién creada mediante este comando:
hint:
hint:   git branch -m <nombre>
Inicializado repositorio Git vacío en /home/german/Documentos/CodeBase/Git/repo/.git/
```

Agregamos el resto de la configuración inicial, usamos configuración global para que afecte todos los repos. Si queremos que la configuración sea solo para cada repo en particular sin afectar a todos, deberíamos quitar la bandera `--global`

7. Copiamos la URL de nuestro repositorio en GitHub



8. configuramos nuestro repositorio remoto, esto le dice a Git a dónde enviar nuestros cambios a nivel de nube, en este caso a un repositorio de github. Nuestro repositorio remoto tendrá el alias de origin, pero este alias puede ser cualquier palabra que deseemos implementar.

```
german@fedora:~/Documentos/CodeBase/Git/repo$ git remote add origin https://github.com/gangaritah/primer-repo
```

9. Verificamos el estado de nuestro proyecto con *git status*

```
german@fedora:~/Documentos/CodeBase/Git/repo$ git status
En la rama master

No hay commits todavía

Archivos sin seguimiento:
(usa "git add <archivo>..." para incluirlo a lo que será confirmado)
    index.js
    reportes/

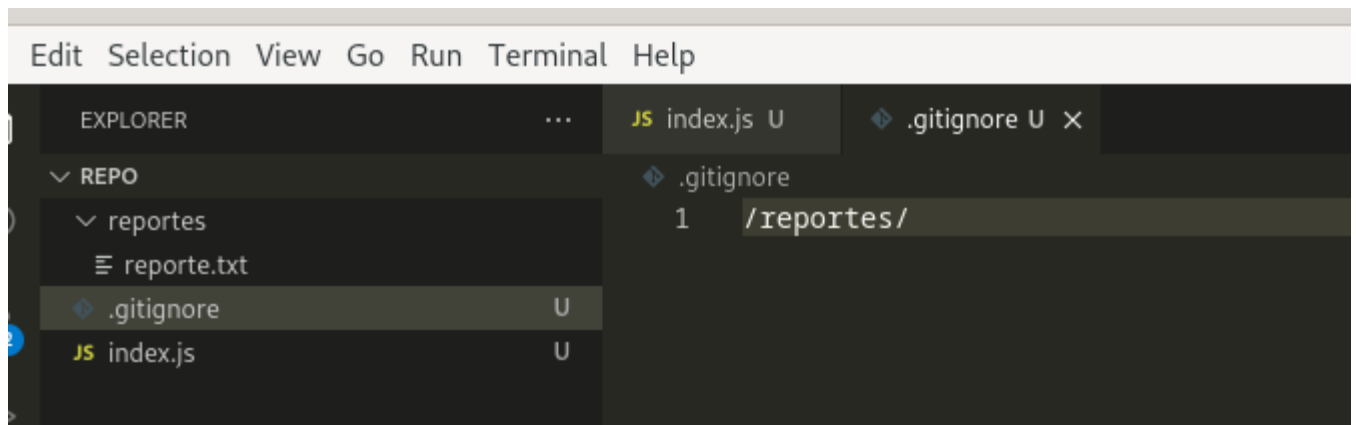
no hay nada agregado al commit pero hay archivos sin seguimiento presentes (usa "git add" para hacerles seguimiento)
```

Observamos en rojo los archivos a los cuales git no les está haciendo seguimiento. De acá, nos interesa que git siga únicamente el código fuente(index.js), la carpeta reportes debemos excluirla del seguimiento de git ya que como dijimos contiene archivos los cuales son relevantes solo en la carpeta local del desarrollador, cómo hacemos esto último?. Para excluir carpetas o archivos usamos un archivo de configuración que le dice a git qué debemos excluir el cual es llamado .gitignore , acá está una lista de reglas para excluir carpetas, subcarpetas, archivos etc

REGLA	EJEMPLO(ESTO IRÍA EN EL ARCHIVO .gitignore)
Para omitir una carpeta completa y todo su contenido (subcarpetas y archivos), usa:  /nombre_carpeta/	/bin/
Para omitir todos los archivos con una determinada extensión, usa:  *.extensión	*.log # Omitir todos los archivos .log *.class # Omitir todos los archivos .class
Para omitir todas las carpetas con un nombre específico en cualquier nivel del directorio, usa:  nombre_carpeta/	target/ # Omitir todas las carpetas 'target' en cualquier nivel
Para omitir un archivo con un nombre específico en cualquier nivel del directorio, usa:  nombre_archivo	.DS_Store # Omitir todos los archivos .DS_Store (típico en macOS)
Para omitir archivos o carpetas específicos dentro de un subdirectorio particular, usa la ruta relativa:  /ruta/relativa/nombre_carpeta/ /ruta/relativa/*.extensión	/src/main/resources/*.properties # Omitir todos los archivos .properties en /src/main/resources/

Para omitir archivos temporales o de respaldo generados por editores de texto o el sistema, usa:  *~ *.swp	*~ # Omitir todos los archivos de respaldo terminados en ~ *.swp # Omitir todos los archivos temporales de Vim

Creemos nuestro .gitignore en nuestro



abrimos el archivo .gitignore con un editor de texto y agregamos la carpeta reportes para ser excluida de nuestro repositorio

guardamos nuestro archivo.

10. Verificamos el estado del proyecto con `git status` y observamos que nuestro .gitignore se encuentra sin seguimiento, también vemos que la carpeta reportes ha desaparecido para git

```
german@fedora:~/Documentos/CodeBase/Git/repo$ git status
En la rama master

No hay commits todavía

Archivos sin seguimiento:
(usa "git add <archivo>..." para incluirlo a lo que será confirmado)
.gitignore
index.js

no hay nada agregado al commit pero hay archivos sin seguimiento presentes (usa "git add" para hacerles seguimiento)
```

11. Agregamos nuestro .gitignore al staging area (área donde los archivos se encuentran listos para hacer commit y ser agregados al repositorio), junto con nuestro archivo index.js. Ejecutamos `git add .` y consultamos de nuevo nuestro proyecto

```
german@fedora:~/Documentos/CodeBase/Git/repo$ git add .
german@fedora:~/Documentos/CodeBase/Git/repo$ git status
En la rama master

No hay commits todavía

Cambios a ser confirmados:
(usa "git rm --cached <archivo>..." para sacar del área de stage)
nuevos archivos: .gitignore
nuevos archivos: index.js
```

Hemos agregado al staging área nuestros archivos .gitignore e index.js y excluido la carpeta reportes.

12. Estamos listos para hacer nuestro primer commit(nuestra primera versión de nuestro programa que será puesta en nuestro repositorio local inicialmente). Hacemos *git commit -m "Commit inicial"*

```
german@fedora:~/Documentos/CodeBase/Git/repo$ git commit -m "Commit inicial"
[master (commit-raíz) 7244e28] Commit inicial
2 files changed, 4 insertions(+)
create mode 100644 .gitignore
create mode 100644 index.js
```

Observamos el mensaje de confirmación de nuestro commit

13. Ahora miramos la rama en la que estamos trabajando con *git branch*

```
german@fedora:~/Documentos/CodeBase/Git/repo$ git branch
* master
```

observamos que en color verde aparece master, esta es la rama predeterminada en git. Si tu rama predeterminada es main(usas una versión de Git más reciente), debes pasar al punto 15

*En Git, una rama (o branch) es una versión paralela del proyecto que permite desarrollar nuevas características, corregir errores o experimentar sin afectar la rama principal del proyecto.*

14. Como estamos en la rama master, la renombraremos como main, ya que main es la rama predeterminada en github. Esto lo hacemos con *git branch -M main*, a continuación verificamos en qué rama estamos con *git branch*

```
german@fedora:~/Documentos/CodeBase/Git/repo$ git branch -M main
german@fedora:~/Documentos/CodeBase/Git/repo$ git branch
* main
```

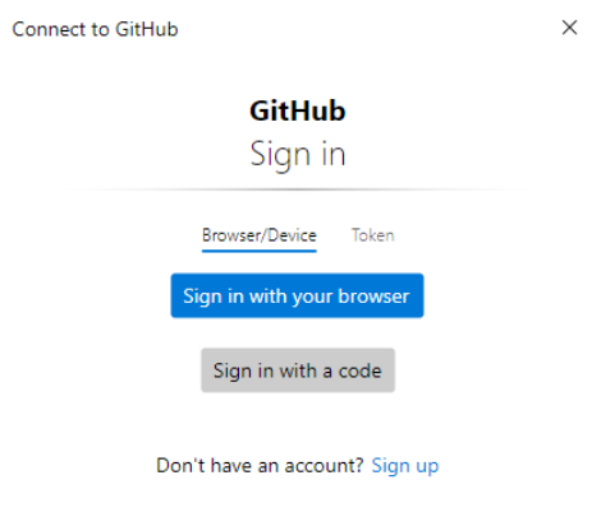
Observamos que nuestra rama actual ya es la rama main

15. Ahora estamos listos para enviar nuestros cambios al repositorio remoto en github, esto lo hacemos con *git push origin main*.

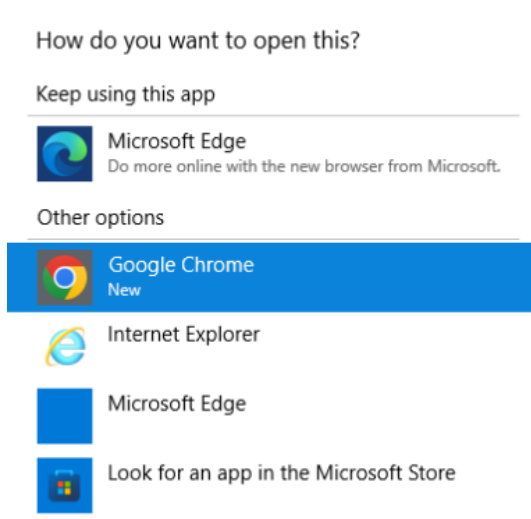
```
german@fedora:~/Documentos/CodeBase/Git/repo$ git push origin main
```

Este comando enviará nuestros cambios al repositorio cuyo alias es origin(el cual configuramos anteriormente) a la rama main. Dependiendo de tu sistema, se puede abrir una ventana donde escogeremos la manera de conectar con

GitHub, ya sea por sesión del navegador o por token (ver guía autenticación por token) , en este caso si es por sesión del navegador(debes tener iniciada la sesión en GitHub en tu navegador) entonces haces click en *Sign in with your browser*



En caso de que se nos abra una ventana para escoger nuestro navegador, debemos escoger el navegador donde estamos logueados en GitHub

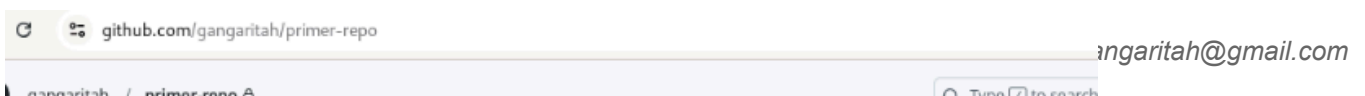


Esperamos un poco a que se haga el login y cambios han sido enviados a nuestro repositorio remoto

luego observaremos que los

```
• german@fedora:~/Documentos/CodeBase/Git/repo$ git push origin main
Enumerando objetos: 4, listo.
Contando objetos: 100% (4/4), listo.
Compresión delta usando hasta 8 hilos
Comprimiendo objetos: 100% (3/3), listo.
Escribiendo objetos: 100% (4/4), 319 bytes | 319.00 KiB/s, listo.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/gangaritah/primer-repo
* [new branch]      main -> main
```

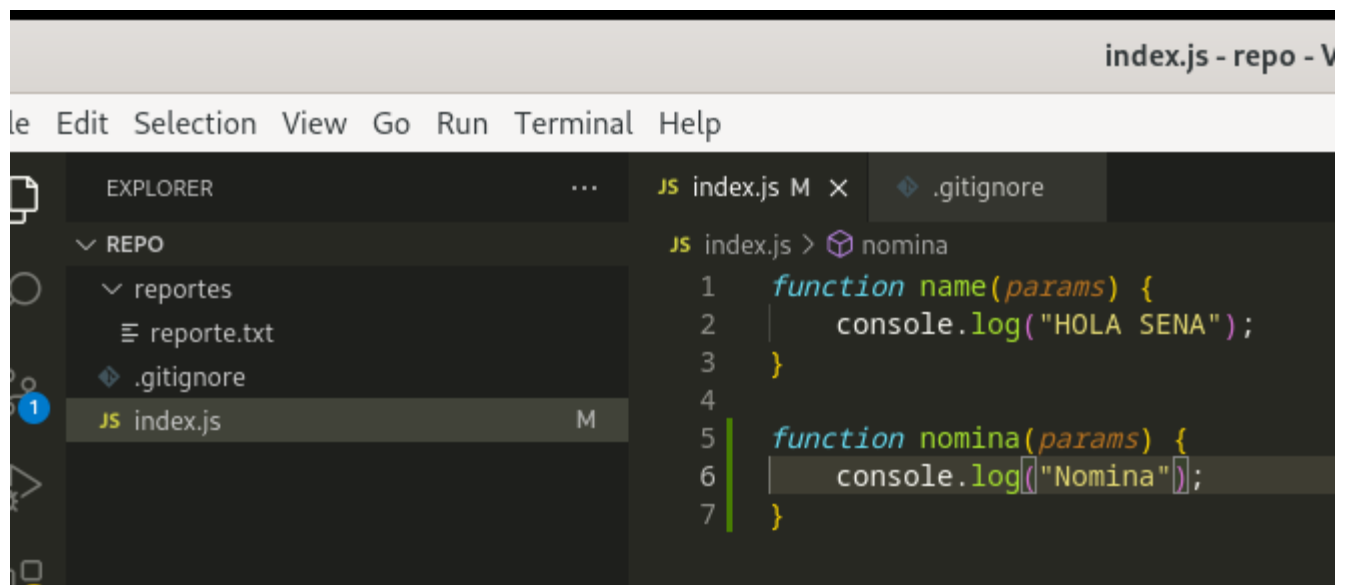
Si actualizamos el sitio de nuestro repositorio en GitHub debemos observar que el código se ha subido con éxito





Acá observamos que la rama actual es la main. El tiempo que aparece es el tiempo que ha transcurrido desde que se hizo el commit localmente.

#### 16. Agregamos otra función a nuestro código



The screenshot shows the Visual Studio Code editor interface. The Explorer sidebar on the left displays the file structure of a repository named 'repo', including 'reportes', 'reporte.txt', '.gitignore', and 'index.js'. The 'index.js' file is selected and open in the editor. The editor window title is 'index.js - repo - V'. The menu bar includes 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', 'Terminal', and 'Help'. The code in 'index.js' shows two functions: 'name' and 'nomina'. The 'nomina' function is being edited, with the cursor at the end of the line 'console.log("Nomina");'. The function signature is 'function nomina(params) {'.

```
index.js - repo - V
File Edit Selection View Go Run Terminal Help
EXPLORER
  REPO
    reportes
      reporte.txt
      .gitignore
    JS index.js M
JS index.js M
  nomina
  1 function name(params) {
  2   | console.log("HOLA SENA");
  3   }
  4
  5 function nomina(params) {
  6   | console.log("Nomina");
  7   }
```

17. Verificamos el estado del proyecto

```
german@fedora:~/Documentos/CodeBase/Git/repo$ git status
En la rama main
Cambios no rastreados para el commit:
  (usa "git add <archivo>..." para actualizar lo que será confirmado)
  (usa "git restore <archivo>..." para descartar los cambios en el directorio de trabajo)
      modificados:      index.js

sin cambios agregados al commit (usa "git add" y/o "git commit -a")
```

Acá vemos que index.js aparece en rojo, esto nos indica que la hemos modificado pero que esas modificaciones no han sido pasadas al staging area. Lo hacemos con *git add*.

```
german@fedora:~/Documentos/CodeBase/Git/repo$ git add .
```

18. Estamos listos para agregar la nueva funcionalidad a nuestro repositorio, hacemos *git commit -m "AGREGADO funcion nomina"*

```
german@fedora:~/Documentos/CodeBase/Git/repo$ git commit -m "AGREGADO funcion nomina"
[main 3955212] AGREGADO funcion nomina
1 file changed, 4 insertions(+)
```

19. Enviamos los últimos cambios a nuestro repositorio remoto

```
german@fedora:~/Documentos/CodeBase/Git/repo$ git push origin main
Enumerando objetos: 5, listo.
Contando objetos: 100% (5/5), listo.
Compresión delta usando hasta 8 hilos
Comprimiendo objetos: 100% (3/3), listo.
Escribiendo objetos: 100% (3/3), 357 bytes | 357.00 KiB/s, listo.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/gangaritah/primer-repo
7244e28..3955212  main -> main
```

Observemos que esta vez no se nos piden credenciales, ya que éstas han sido almacenadas por git

20. Verificamos nuestro repositorio remoto actualizando el sitio web:

The screenshot shows the GitHub interface for a repository named 'primer-repo' owned by 'gangaritah'. The repository is marked as 'Private'. The main branch is 'main', with 1 branch and 0 tags. A search bar is present with the text 'Type / to search'. Below the repository header, there is a commit history table. The table has three columns: file name, commit message, and time ago. The first commit is for '.gitignore' with the message 'Commit inicial' and was made '24 minutes ago'. The second commit is for 'index.js' with the message 'AGREGADO funcion nomina' and was made '1 minute ago'. Below the commit history, there is a 'README' section which is currently empty, indicated by an open book icon.

File	Commit Message	Time Ago
.gitignore	Commit inicial	24 minutes ago
index.js	AGREGADO funcion nomina	1 minute ago

Observamos que el nuevo cambio **AGREGADO funcion nomina** ya aparece en el repositorio