



Gobierno del
CHACO

Ministerio
de la Producción y el Desarrollo
Económico Sostenible



INFORMATARIO



GIT Y GITHUB

La consola | Qué es Git | Qué es GitHub | Uso de la consola en VSCode | Tipos de terminales | Instalación de Git y GitHub | Repositorios | Operaciones básicas | Comandos y práctica | Despliegue | Documentación.

4



Índice

Introducción	2
Interfaz básica de la consola	3
Ejercicios prácticos	6
¿Qué es la consola en VSCode?	9
Git y GitHub: Instalación, creación de cuenta, repositorio	11
Repositorios en Github	12
Creación de un repositorio en Github.....	14
Inicialización de un Repositorio Local y Conexión con Github ...	15
Material de apoyo, instalación de Git	18
Creación de Cuenta en Github.....	23
Iniciar con tu cuenta de Github:	27
Comandos Git: Operaciones Básicas y Visualización del Área de Staging	32
Material de Apoyo: Operaciones Básicas con Git	33
Otros Comandos de Git y Ejemplos Prácticos	39
Despliegue en GitHub Pages	43

Introducción

El dominio de la consola, en conjunto con Git y Github, no solo aumentará tu eficiencia, sino que también te proporcionará oportunidades clave en el ámbito del desarrollo profesional..

Git y GitHub

Es importante saber que **estas herramientas son pilares fundamentales** en el desarrollo de software. No sólo agilizan la colaboración, sino que también brindan **un historial claro de todos los cambios en el código**. Sumamente importante en un proyecto, permitiéndonos lograr trazabilidad eficiente en dicho proyecto.



Es un sistema que permite guardar muchas versiones diferentes de uno o varios archivos. Software alojado localmente en el sistema. Herramienta de línea de comandos.



Es una plataforma basada en la web, que permite alojar distribuciones de archivos de Git. Permite que los usuarios los mantengan abiertos a colaboraciones.

¿Que es la consola en el desarrollo de software?

Es una herramienta clave por su capacidad para automatizar tareas, gestionar proyectos, y facilitar el control de versiones. Una consola es una interfaz que funciona por comandos, es decir, órdenes escritas en texto.

¿Por qué se utiliza directamente desde VSCode?

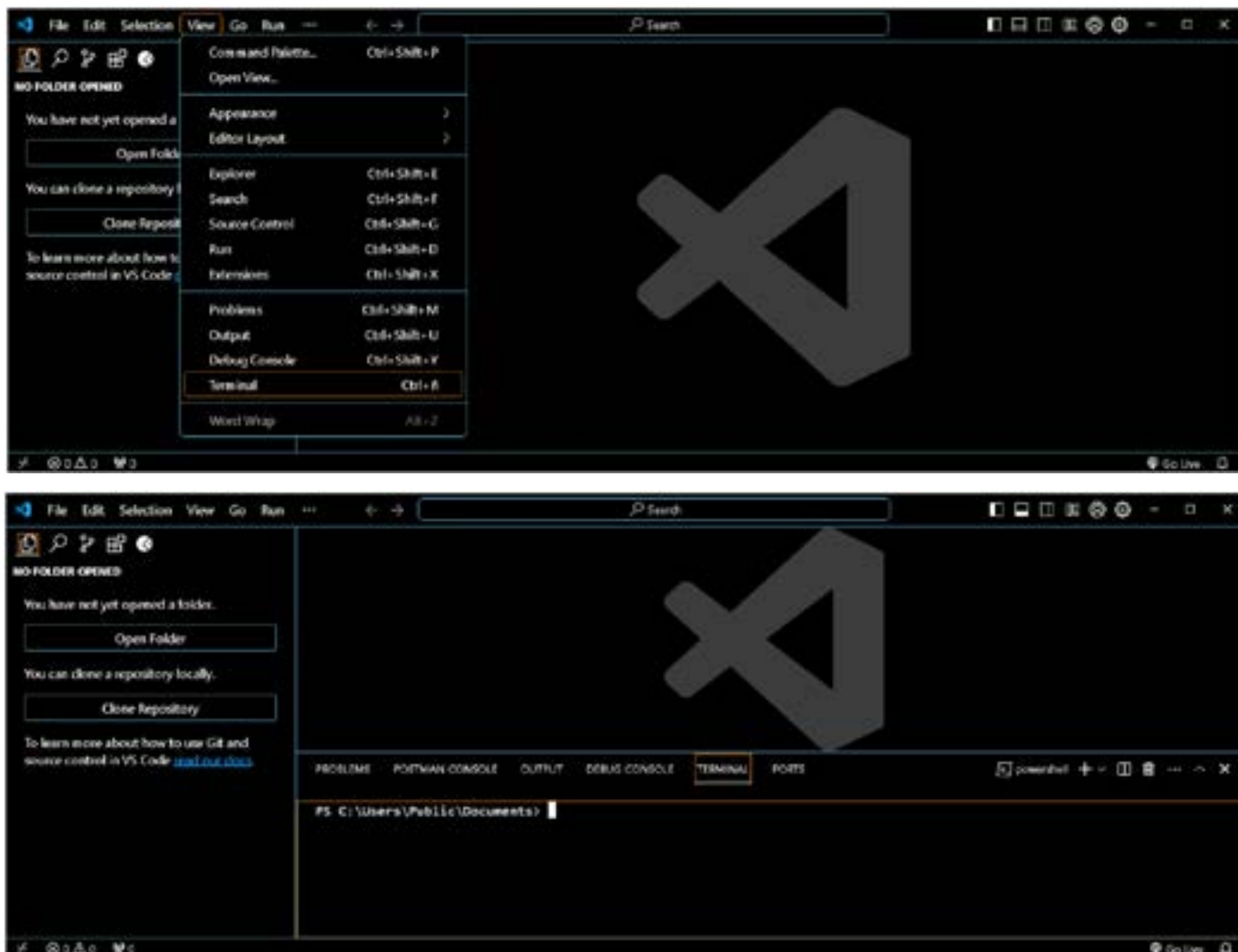
Se la suele utilizar directamente desde VSCode porque agiliza el flujo de trabajo, ofrece integración nativa, eficiencia, y coherencia en el entorno de desarrollo. Es decir, mejora nuestra productividad y facilita la transición a la automatización.

Cómo abrir la consola en VSCode

El procedimiento para abrir la consola se resume y demuestra de este modo:

ACCESO RÁPIDO

Ctrl + J o Ver → Terminal → Nueva Terminal.

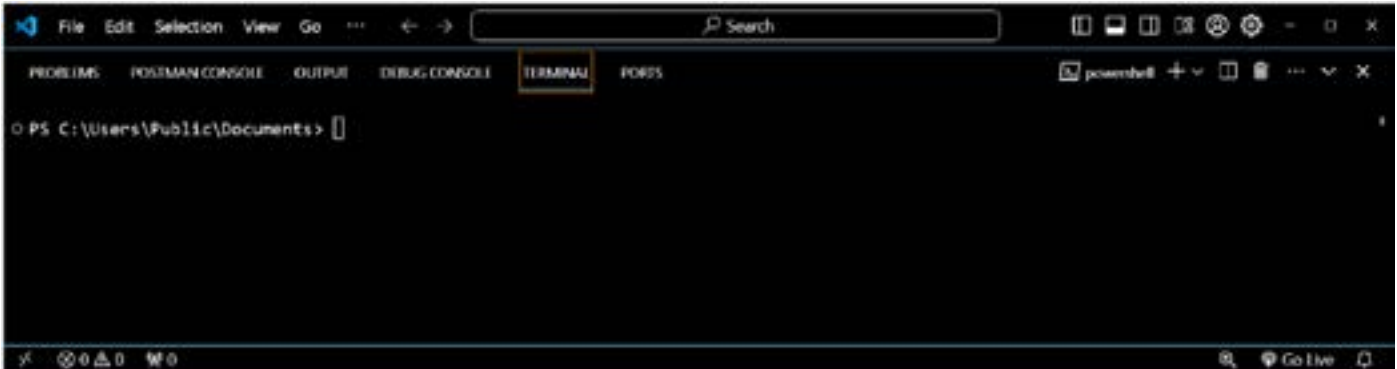


Interfaz básica de la consola

Los elementos esenciales de la interfaz de la consola en VSCode serían los siguientes:

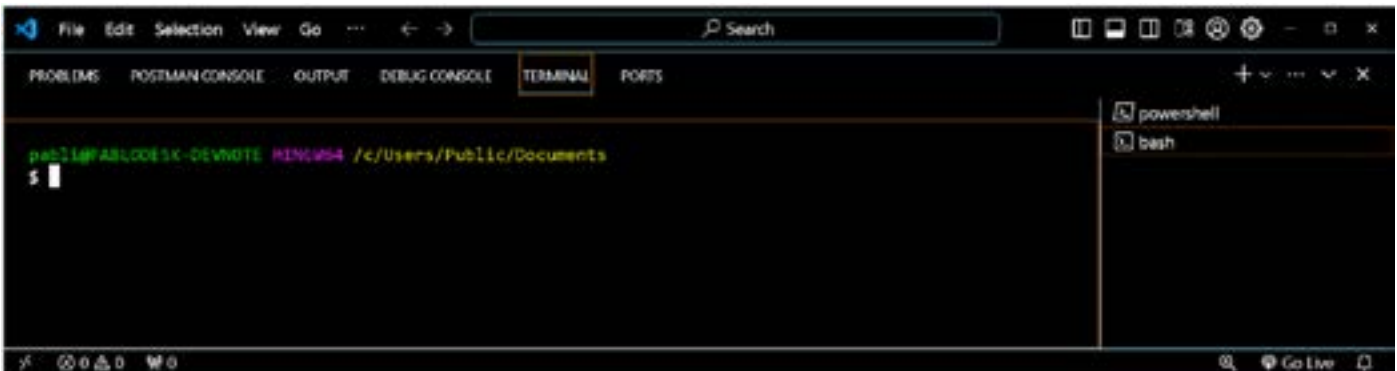
Prompt de comandos: El "prompt de comandos" se refiere a la línea de texto que aparece en la consola y espera la entrada del usuario, es decir, es la sección donde puedes escribir y ejecutar comandos, y donde se puede ob-

► **Símbolo o texto que indica que está esperando dicho comando:** En muchos sistemas, por lo general, el prompt puede ser un símbolo de dólar (\$), una flecha (>), o el nombre del usuario seguido de un símbolo (nombre_de_usuario\$), dependiendo del sistema operativo y la configuración.



```

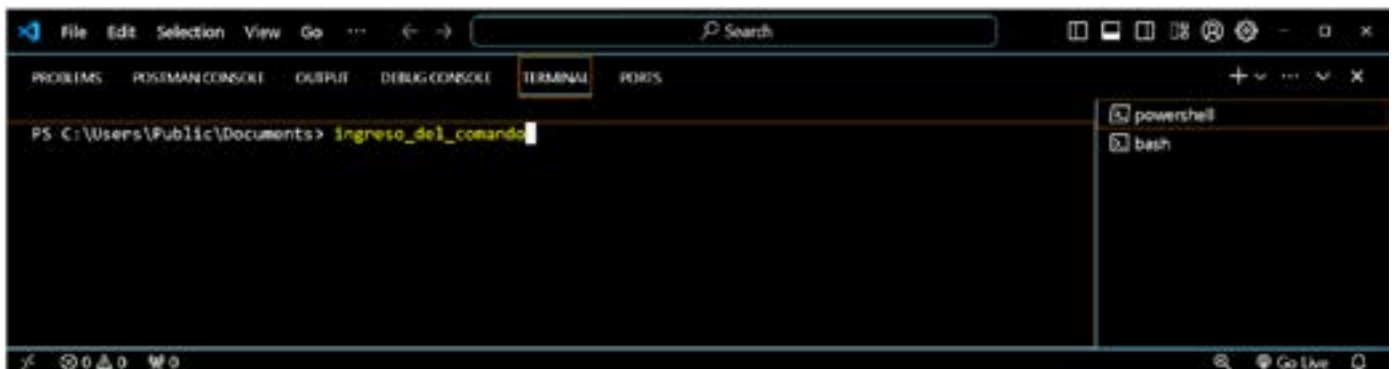
File Edit Selection View Go ... Search
PROBLEMS POSTMAN CONSOLE OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Public\Documents>
  
```



```

File Edit Selection View Go ... Search
PROBLEMS POSTMAN CONSOLE OUTPUT DEBUG CONSOLE TERMINAL PORTS
pabli@FASLCOESK-DEVNOTE MINGW64 /c/Users/Public/Documents
$
  
```

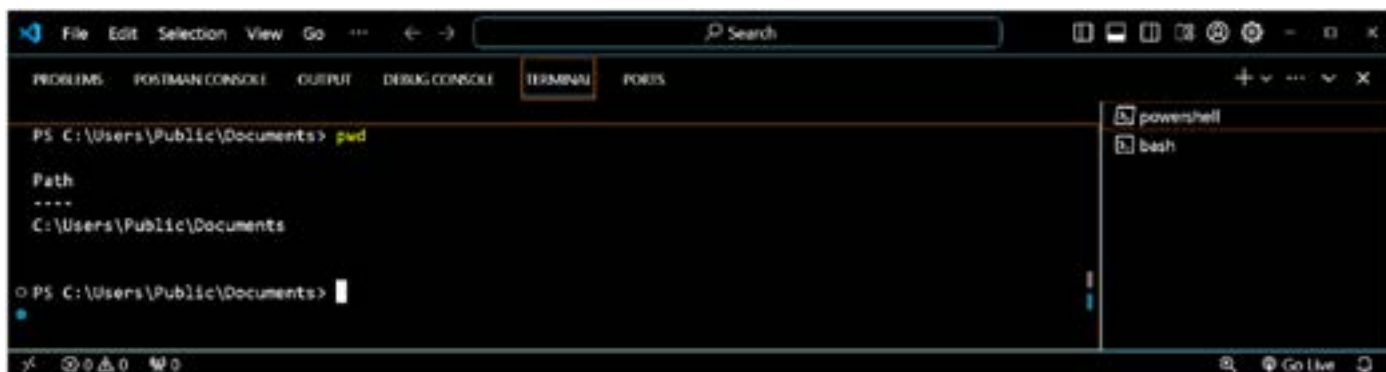
► **Interpretación del prompt:** Después del prompt es donde se ingresan los comandos. Algunas consolas también muestran información adicional, como la ruta del directorio actual.



```

File Edit Selection View Go ... Search
PROBLEMS POSTMAN CONSOLE OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Public\Documents> ingreso_del_comando
  
```

Ruta actual: Cómo se muestra la ubicación actual en la consola usando **pwd** o **cd**.



Atajos y funciones:

Ctrl + J o Ver -> Terminal -> Nueva Terminal:

Este atajo permite abrir y cerrar la terminal en VSCode. Puede ser útil para aquellos que no estén familiarizados con la ubicación de la consola en la interfaz.

Atajos de Teclado en la Consola de VSCode:

Ctrl + C: Interrumpe la ejecución actual del comando.

Ctrl + L: Limpia la pantalla de la consola.

Comandos de Navegación en la Consola:

- **pwd (solo cd en cmd):** Muestra la ruta completa del directorio actual.
- **ls (o dir en Windows):** Lista los archivos y carpetas en el directorio actual.
- **cd:** Cambia el directorio.
- **clear (o cls en Windows):** Limpia la pantalla

de la consola.

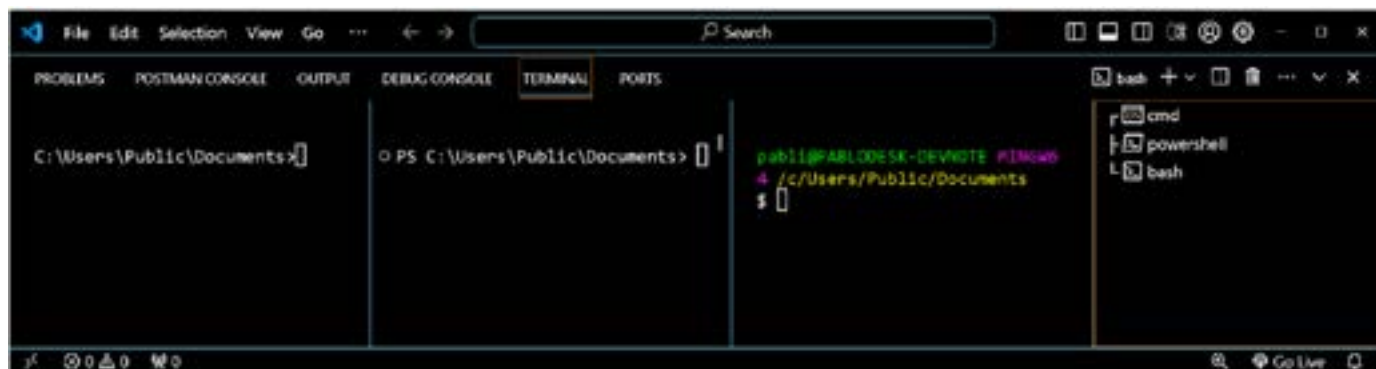
- **mkdir:** Crear carpeta.
- **echo:** Crear archivo.

Funciones Específicas de VSCode:

- **Auto Completado:** La consola de VSCode suele tener funciones de autocompletado, lo que facilita la escritura de nombres de archivos y directorios. **Tabulación.**
- **Historial de Comandos:** Flechas arriba/abajo para navegar por el **historial de comandos.**

Otras Funciones Avanzadas (opcional):

- **División de Pantalla:** Algunas consolas permiten dividir la pantalla para ejecutar múltiples comandos simultáneamente.
- **Integración con Git:** Si estás trabajando en un repositorio de Git, la consola puede mostrar información adicional sobre el estado del repositorio.



Hay una compatibilidad en la ejecución de los comandos en diferentes consolas, que puede ser el resultado de una combinación de factores, como:

- El sistema operativo (ya sean basados en Unix, Linux o Windows),
- Configuraciones específicas.
- Scripts.
- Extensiones o complementos instalados.

Es decir, todo esto puede influir en qué comandos son aceptados y ejecutados por una consola en particular.

Ejemplos de navegación entre directorios:

Ejercicios prácticos de navegación utilizando comandos básicos.

► **cd:** Cambiar directorio.

► **ls (o dir en Windows):** Listar contenido del directorio.

► Ejemplos de cómo navegar hacia arriba (**cd ..**) y hacia abajo (**cd nombre_de_directorio**).

► Cómo navegar entre carpetas usando rutas relativas y absolutas.

EJERCICIO PRÁCTICO 1

CAMBIAR DE DIRECTORIO

Comando: **cd**
cd "Nueva carpeta"

```

PS C:\Users\Public\Documents> cd "Nueva carpeta"
PS C:\Users\Public\Documents\Nueva carpeta> dir

Directorio: C:\Users\Public\Documents\Nueva carpeta

Mode                LastWriteTime         Length Name
----                -
d-----          7/3/2024    09:15             Nueva carpeta
-a-----          7/3/2024    09:15             @ Nuevo Documento de Microsoft Word.docx
-a-----          7/3/2024    09:15             @ Nuevo Documento de texto.txt

PS C:\Users\Public\Documents\Nueva carpeta> cd ..
PS C:\Users\Public\Documents>

```

► Este comando cambiará el directorio actual a "Nueva carpeta". Asegúrate de adaptar el nombre del directorio según tu estructura de carpetas.

EJERCICIO PRÁCTICO 2

LISTAR CONTENIDO DEL DIRECTORIO

Comando: **ls (o dir en Windows)**
ls
dir

```

PS C:\Users\Public\Documents\Nueva carpeta> ls

Directorio: C:\Users\Public\Documents\Nueva carpeta

```

EJERCICIO PRÁCTICO 2

LISTAR CONTENIDO DEL DIRECTORIO

The first screenshot shows the PowerShell terminal with the command `ls` entered. The output displays the directory contents of `C:\Users\Public\Documents\Nueva carpeta`, listing a subdirectory and two text documents.

```
PS C:\Users\Public\Documents\Nueva carpeta> ls

Directorio: C:\Users\Public\Documents\Nueva carpeta

Mode                LastWriteTime         Length Name
----                -
d-----          7/3/2024   09:15             Nueva carpeta
-a-----          7/3/2024   09:15      0 Nuevo Documento de Microsoft Word.docx
-a-----          7/3/2024   09:15      0 Nuevo Documento de Texto.txt

PS C:\Users\Public\Documents\Nueva carpeta>
```

The second screenshot shows the same terminal with the command `dir` entered. The output is identical to the first screenshot, showing the directory listing.

```
PS C:\Users\Public\Documents\Nueva carpeta> dir

Directorio: C:\Users\Public\Documents\Nueva carpeta

Mode                LastWriteTime         Length Name
----                -
d-----          7/3/2024   09:15             Nueva carpeta
-a-----          7/3/2024   09:15      0 Nuevo Documento de Microsoft Word.docx
-a-----          7/3/2024   09:15      0 Nuevo Documento de Texto.txt

PS C:\Users\Public\Documents\Nueva carpeta>
```

► Este comando mostrará una lista de archivos y carpetas en el directorio actual.

EJERCICIO PRÁCTICO 3

NAVEGAR HACIA ARRIBA

Comando: `cd ..`
`cd ..`

The screenshot shows the PowerShell terminal with the command `cd ..` entered. The prompt changes from `PS C:\Users\Public\Documents\Nueva carpeta>` to `PS C:\Users\Public\Documents>`, indicating the current directory has moved up one level.

```
PS C:\Users\Public\Documents\Nueva carpeta> cd ..
PS C:\Users\Public\Documents>
```

► Este comando moverá el directorio actual hacia arriba en la jerarquía de carpetas.

EJERCICIO PRÁCTICO 4 NAVEGAR HACIA ABAJO

Comando: `cd nombre_de_directorio`
`cd Documents`




```
File Edit Selection View Go Run ... Search
PROBLEMS PORTMAN CONSOLE OUTPUT DEBBUG CONSOLE TERMINAL PORTS
PS C:\Users\Public> cd .\Documents\
PS C:\Users\Public\Documents>
```

► Este comando cambiará al directorio llamado "Documents". Asegúrate de sustituir "nombre_de_directorio" con el nombre real del directorio.

EJERCICIO PRÁCTICO 5 NAVEGAR CON RUTAS RELATIVAS

`cd Documents/"Nueva carpeta"`



```
File Edit Selection View Go Run ... Search
PROBLEMS PORTMAN CONSOLE OUTPUT DEBBUG CONSOLE TERMINAL PORTS
PS C:\Users\Public> cd .\Documents\
PS C:\Users\Public\Documents>
```

► Este comando cambiará al directorio "Proyectos" que está dentro de "Documents". Utilizando rutas relativas, la consola interpreta que "Proyectos" es un subdirectorio de "Documents".

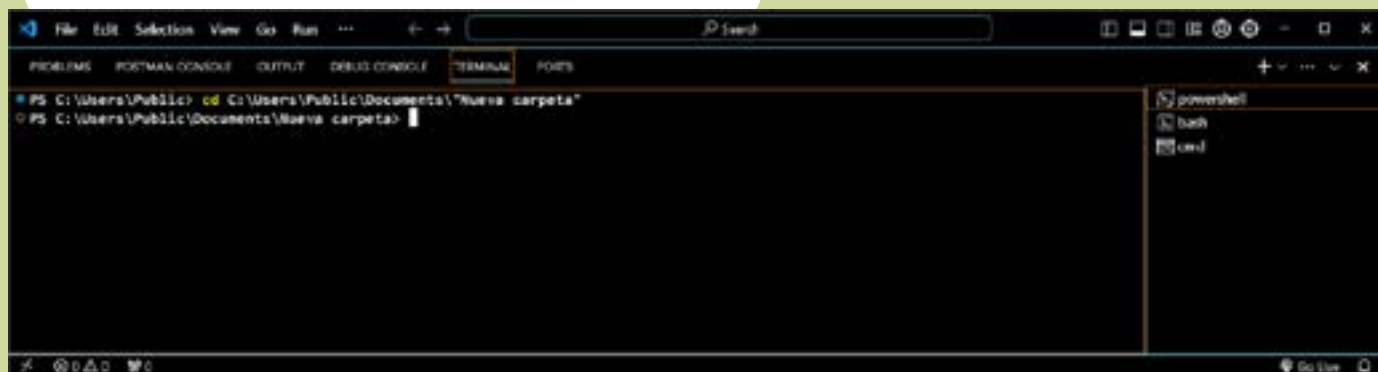
EJERCICIO PRÁCTICO 6 NAVEGAR CON RUTAS ABSOLUTAS

`cd C:/Users/Public/Documents/"Nueva carpeta"`

EJERCICIO PRÁCTICO 6

NAVEGAR CON RUTAS ABSOLUTAS

```
cd C:/Users/Public/Documents/"Nueva carpeta"
```



► Este comando cambiará directamente al directorio "Nueva carpeta" utilizando una ruta absoluta. Asegúrate de cambiar "public" con tu nombre de usuario.

Es fundamental comprender estos conceptos para gestionar eficientemente la estructura de carpetas de un proyecto.

¿Qué es la consola en VSCode?

La consola en Visual Studio Code es una interfaz de línea de comandos integrada directamente en el entorno de desarrollo.

Permite a los desarrolladores ejecutar comandos y scripts directamente desde la interfaz de usuario de VSCode, eliminando la necesidad de cambiar a una ventana de terminal externa. Puede ser utilizada para una variedad de tareas, desde la ejecución de comandos básicos hasta la gestión de paquetes, ejecución de scripts y más.

Ésta interactúa con una interfaz y permite enviar comandos directamente al sistema operativo. Proporciona una manera eficiente de realizar tareas sin salir del entorno de desarrollo.

¿Qué es Git y qué es Github?

Git es un sistema de control de versiones distribuido que permite el seguimiento de cambios en el código fuente durante el desarrollo de software.

Facilita el trabajo colaborativo, el seguimiento de versiones y la gestión eficiente de proyectos. Cada desarrollador tiene una copia completa del historial de cambios, lo que permite la colaboración en paralelo sin conflictos.

Git realiza un seguimiento de los cambios me-

diante la creación de "commits", instantáneas del código en un momento específico. Permite la creación de ramas para trabajar en nuevas características sin afectar una rama principal (main o master). Las operaciones como "pull" y "push" facilitan la colaboración, y las fusiones (merges) consolidan los cambios realizados por diferentes colaboradores.

GitHub es una plataforma web que utiliza Git para facilitar la colaboración en proyectos. Proporciona funcionalidades como control de versiones, seguimiento de problemas y colaboración entre desarrolladores, proporcionando alojamiento de repositorios (espacio de almacenamiento).

En GitHub varios desarrolladores pueden colaborar en un repositorio, realizar cambios y fusionarlos de manera eficiente, además permite realizar un seguimiento de problemas, y las pull requests facilitan la revisión e incorporación de cambios.

Además, GitHub brinda la posibilidad de alojar páginas web directamente desde el repositorio, brindando una solución sencilla para la publicación de documentación, sitios personales, etc.



Sistema de control de versiones que lleva registro de cambios y avances en el proyecto.

Cual una máquina de tiempo, podés ir al pasado de tu código o volver al presente.

Ayuda a que varias personas trabajen en un mismo proyecto y puedan modificarlo sin afectar a los demás.



Sistema de control de versiones que lleva registro de cambios y avances en el proyecto.

En GitHub los programadores pueden subir sus proyectos Git locales, ver código de otros Devs y colaborar.

También podés hacer actualizaciones, programar desde el navegador y detectar vulnerabilidades.

Importancia de Control de Versiones y Colaboración en el Desarrollo de Software:



El control de versiones, respaldado por herramientas como Git y plataformas como GitHub, es esencial en el desarrollo de software por varias razones:

- ▶ Permite un seguimiento preciso de los cambios en el código.
- ▶ Facilita la colaboración entre desarrolladores.
- ▶ Reduce conflictos en el trabajo en equipo.
- ▶ Proporciona un historial completo del proyecto.

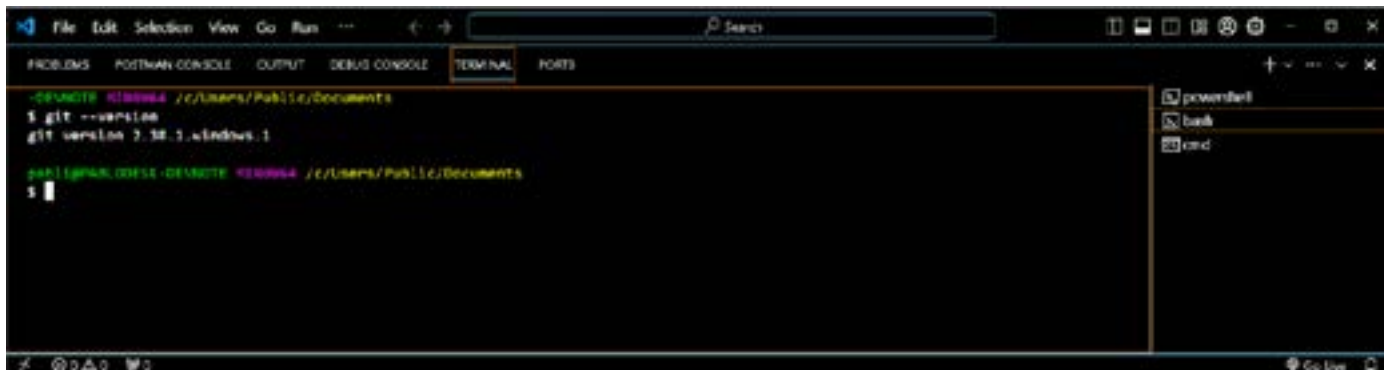
- ▶ La colaboración a través de plataformas como GitHub:
 - Mejora la transparencia.
 - Fomenta la revisión de código.
 - Agiliza el proceso de integración de nuevas funcionalidades al proyecto principal.
- ▶ Historial de cambios.
 - Permite revertir cambios.
 - Entender la evolución del código.
 - Solucionar problemas.
- ▶ Colaboración Eficiente.
 - Múltiples desarrolladores pueden trabajar simultáneamente.
 - Fusionar sus contribuciones sin conflictos.
- ▶ Revisión de Código.
 - Facilita la revisión y aprobación de cambios a través de pull requests.
- ▶ Integración Continua.
 - Permite la integración continua (CI).
 - Mejora la calidad del software.
 - Permite la entrega rápida de nuevas funcionalidades.

Git y GitHub: Instalación, creación de cuenta, repositorio

Instalación de Git y Creación de Cuenta en Github:

1. PASOS PARA INSTALAR GIT

- a). Descarga de la última versión desde el sitio oficial de Git.
Descarga GIT: [Git - Downloads](#)
- b). Iniciar el instalador y seguir los pasos de configuración.
- c). Verificación de la instalación mediante el comando `git --version`.



```

-DESKTOP: C:\Users\Public\Documents
$ git --version
git version 2.38.1.windows.1

-DESKTOP: C:\Users\Public\Documents
$
  
```

2. PASOS PARA CREAR UNA CUENTA EN GITHUB

- Acceder al sitio web de Github.
Web GITHUB: <https://github.com/>
- Completar la información requerida para la creación de la cuenta.
- Verificación del correo electrónico asociado a la cuenta.

3. CONFIGURACIÓN INICIAL DEL PERFIL

- Breve introducción a la configuración global.
 - * `git config --global user.name "Tu Nombre"`
 - * `git config --global user.email "tu@email.com"`
 - * Verificar nombre: `git config --global --get user.name`
 - * Verificar email: `git config --global --get user.email`

Repositorios en Github

Un repositorio es un espacio centralizado que almacena y organiza archivos relacionados con un proyecto de desarrollo de software.

Un repositorio funciona como un contenedor que guarda no solo el código, sino también la historia completa de cambios realizados en el código.

- Proporciona una estructura organizada.
- Facilita el seguimiento de cambios y la gestión de versiones (trazabilidad).
- Permite la colaboración entre desarrolladores, ya que todos trabajan en una copia del mismo repositorio.

Funcionamiento básico de los repositorios

INICIALIZACIÓN (INIT): Crea un nuevo repositorio de Git, estableciendo la estructura básica y el historial para rastrear cambios.

Comando: `git init` .

CONEXIÓN CON EL REPOSITORIO REMOTO (REMOTE ADD): Conecta el repositorio local con un repositorio remoto para poder sincronizar cambios.

Comando: `git remote add origin <URL-del-repositorio>`

CLONACIÓN (CLONE): Desarrolladores pueden clonar (copiar) el repositorio existente a sus máquinas locales para trabajar en el código.

Comando: `git clone <URL-del-repositorio>`

COMMITTS (COMMIT): Cada cambio realizado en el código se registra como un "commit", con detalles como el autor, fecha y un mensaje descriptivo.

Comando: `git commit -m "mensaje descriptivo"`

RAMAS (BRANCHES): Se pueden crear ramas para trabajar en nuevas características sin afectar la rama principal (main o master).

Comando: `git branch <nombre-de-la-rama>` y `git checkout <nombre-de-la-rama>` (o `git checkout -b <nombre-de-la-rama>` para crear y cambiar a la rama simultáneamente).

FUSIONES (MERGE): Cuando una funcionalidad está lista, se fusiona de vuelta a la rama principal para la integración.

Comando: `git merge <nombre-de-la-rama>`

SOLICITUDES DE FUSIÓN (PULL REQUEST): Permite proponer cambios, iniciar discusiones y realizar revisiones antes de fusionar el código.

Ejemplo: En GitHub, abres una pull request desde la interfaz web, comparando tu rama con la rama principal.

SINCRONIZACIÓN (PUSH & PULL & FETCH):

PULL: Obtiene y fusiona los cambios del repositorio remoto al local.

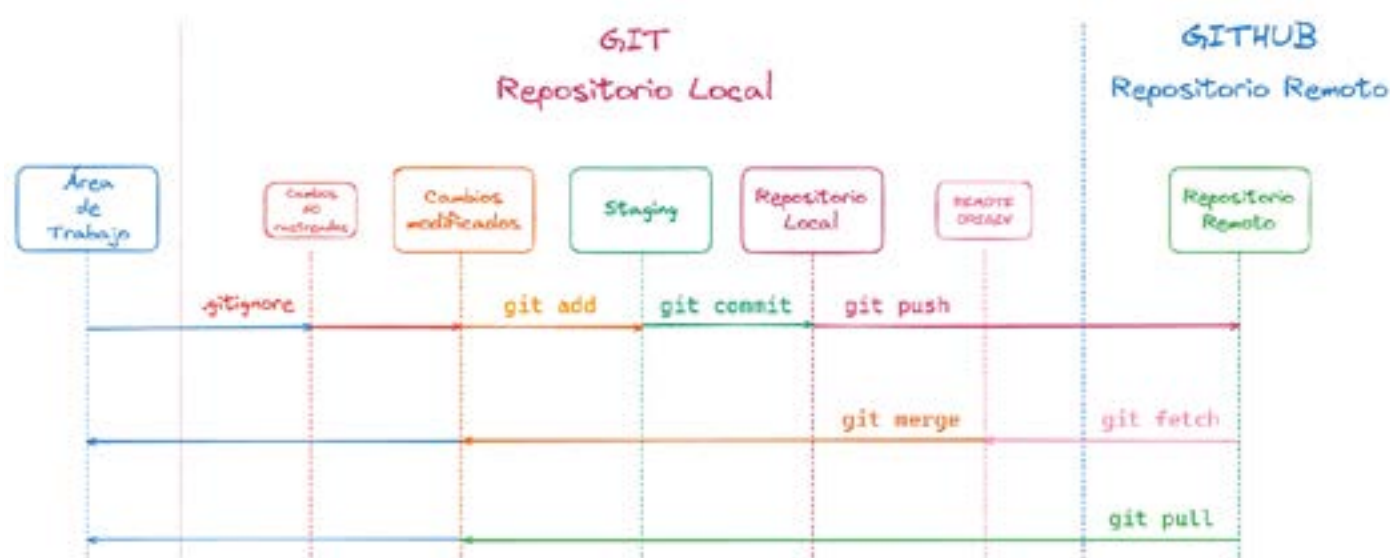
Comando: `git pull origin <nombre-de-la-rama>`

FETCH: Obtiene los cambios del repositorio remoto sin fusionarlos automáticamente en la rama actual, permitiendo revisar los cambios antes de aplicarlos.

Comando: `git fetch origin <nombre-de-la-rama>`

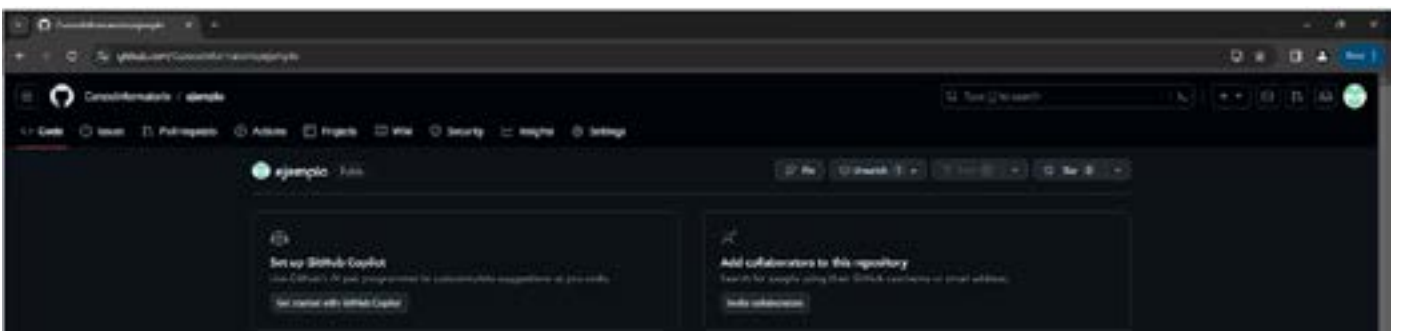
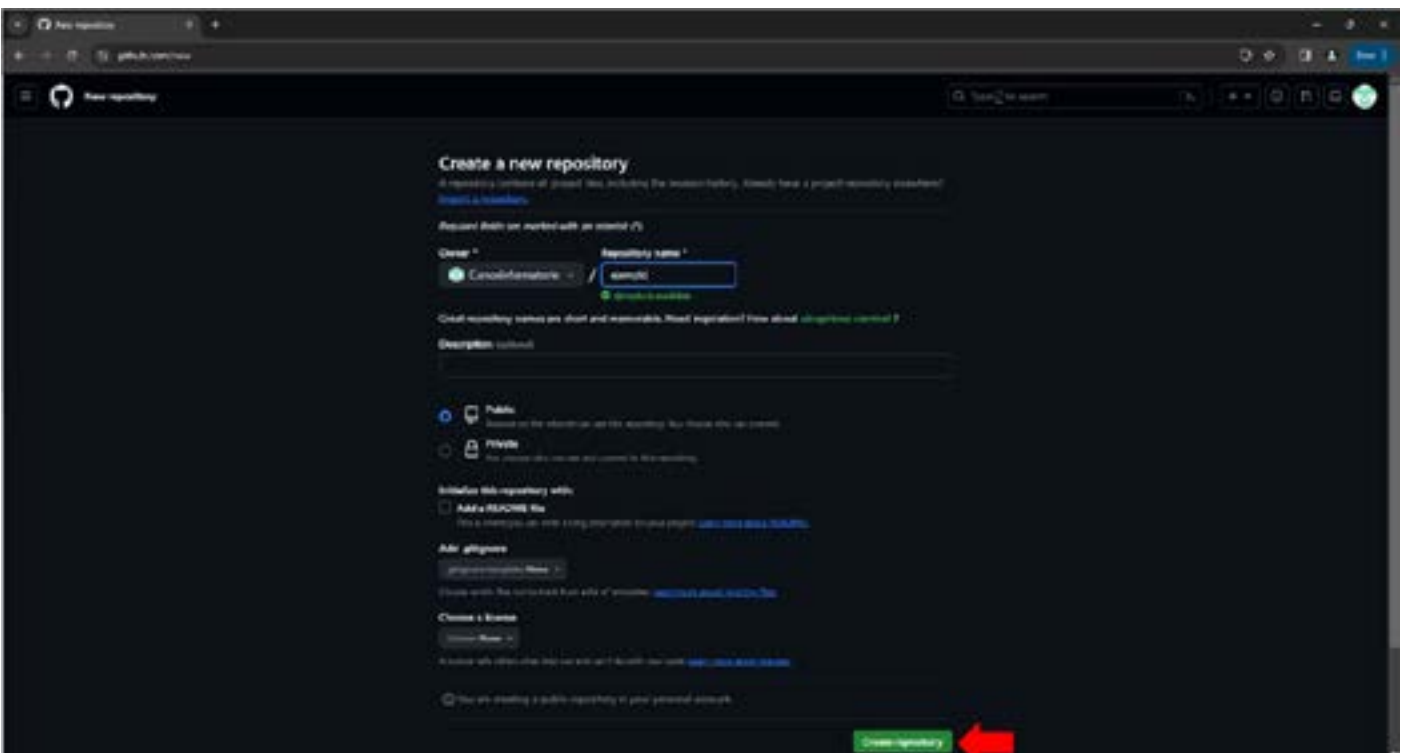
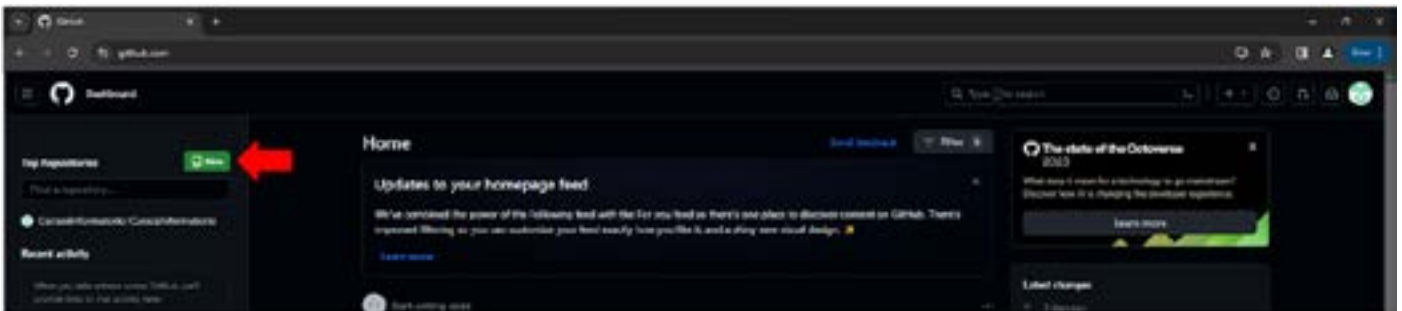
PUSH: Envía los commits locales al repositorio remoto.

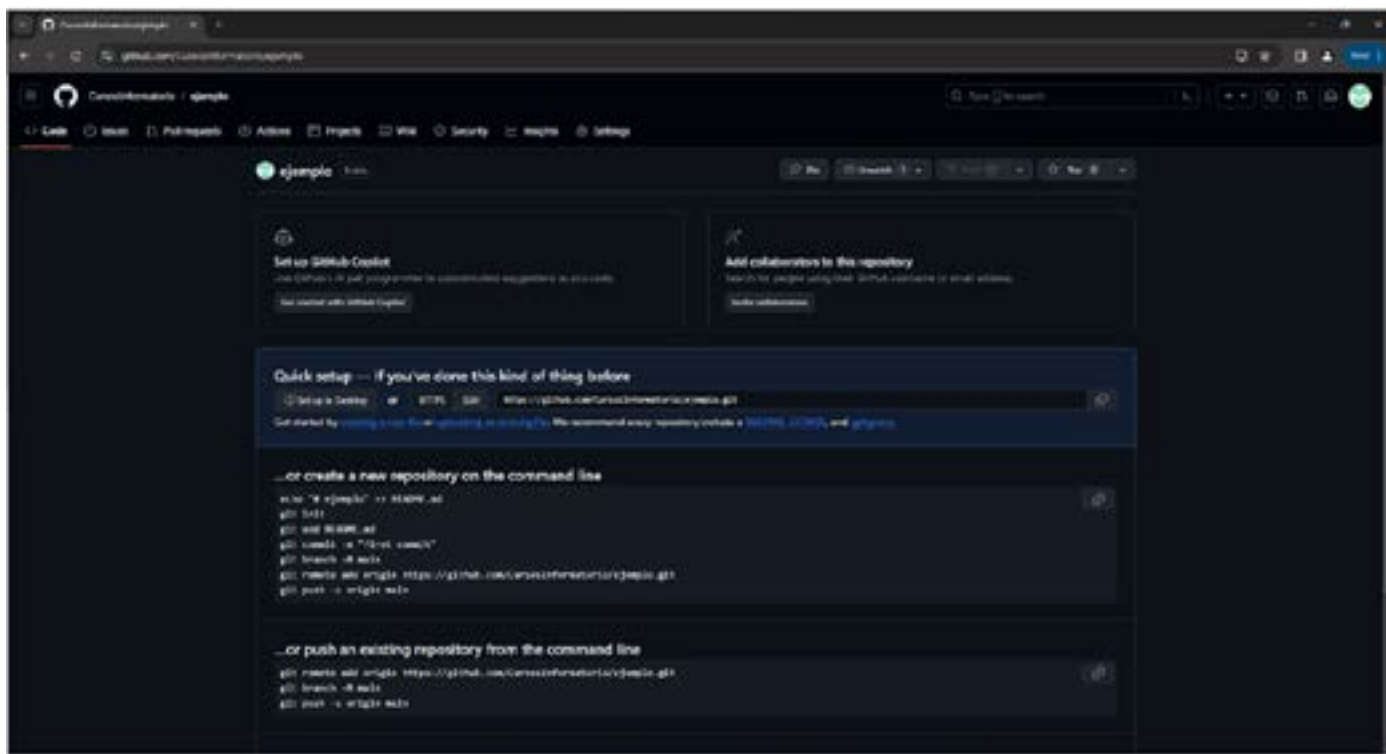
Comando: `git push origin <nombre-de-la-rama>`



Creación de un repositorio en Github:

Pasos para crear un nuevo repositorio en Github.





Configuración de opciones como descripción, público/privado y licencia.

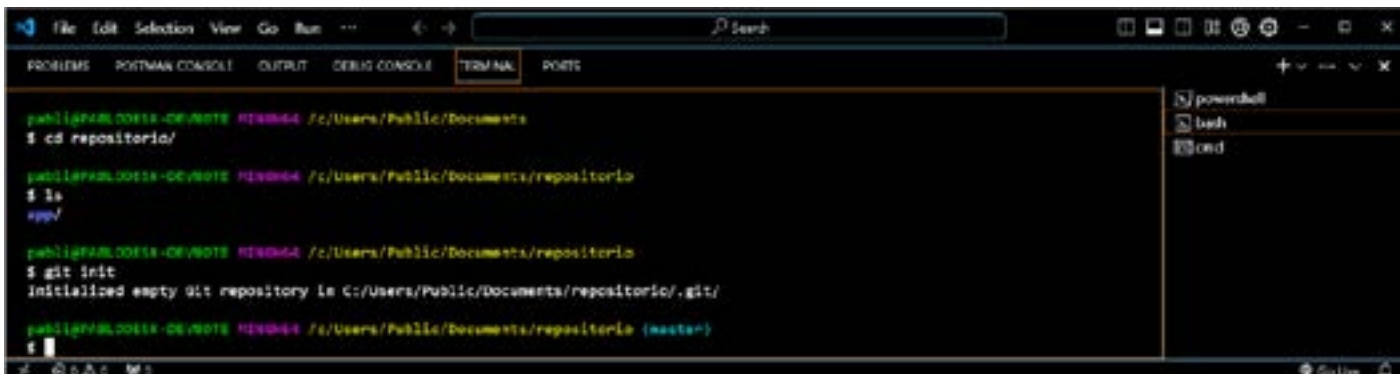
- ▶ `git init`
- ▶ `git add .`
- ▶ `git commit -m "Primer commit"`
- ▶ `git remote add origin URL_del_repositorio_en_Github`
- ▶ `git push -u origin master`

Inicialización de un Repositorio Local y Conexión con Github:

Inicialización de un repositorio local e introducción a comandos básicos:

a). Demostración del comando `git init` para inicializar un nuevo repositorio local.

Inicializa un nuevo repositorio local: **`git init`**



```

pabli@PABLODESK-DEVNOTE R210W64 /c/Users/Public/Documents
$ cd repositorio/

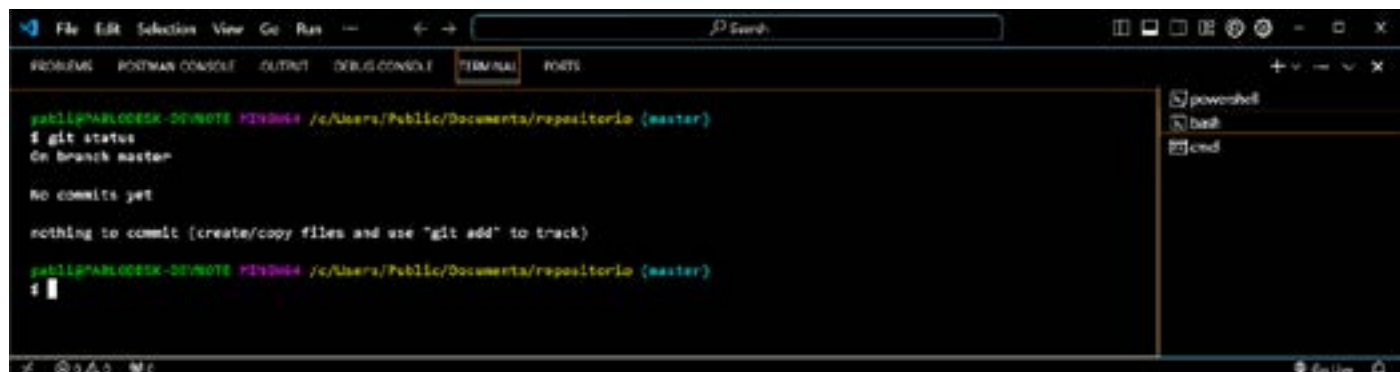
pabli@PABLODESK-DEVNOTE R210W64 /c/Users/Public/Documents/repositorio
$ ls
app/

pabli@PABLODESK-DEVNOTE R210W64 /c/Users/Public/Documents/repositorio
$ git init
Initialized empty Git repository in C:/Users/Public/Documents/repositorio/.git/

pabli@PABLODESK-DEVNOTE R210W64 /c/Users/Public/Documents/repositorio (master)
$
  
```

b). Introducción a comandos básicos: `status`, `add`, `commit`.

Muestra el estado actual de los archivos en el repositorio: **`git status`**



```

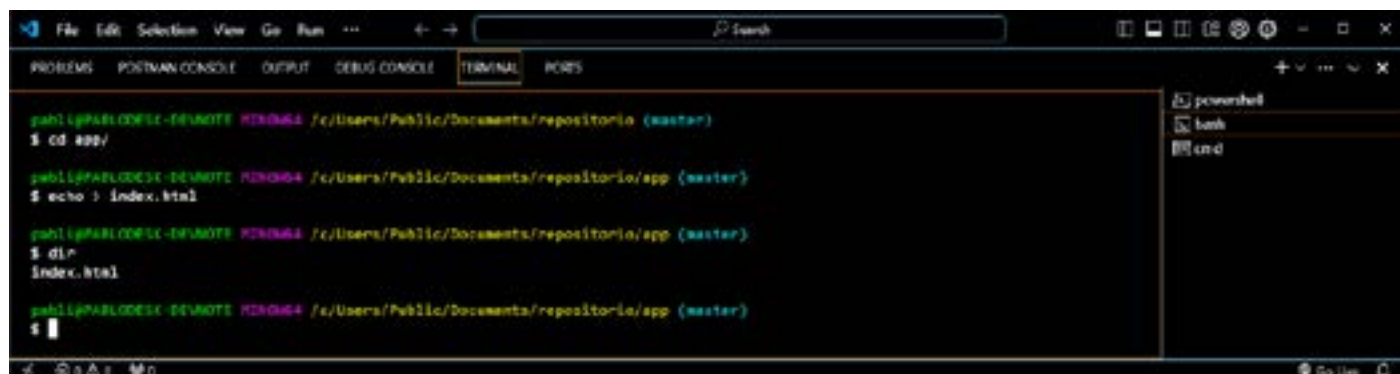
pabli@PABLODESK-DEVNOTE R210W64 /c/Users/Public/Documents/repositorio (master)
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)

pabli@PABLODESK-DEVNOTE R210W64 /c/Users/Public/Documents/repositorio (master)
$
  
```

Desplázate por los directorios y crea archivos:
`cd` y `echo > index.html`



```

pabli@PABLODESK-DEVNOTE R210W64 /c/Users/Public/Documents/repositorio (master)
$ cd app/

pabli@PABLODESK-DEVNOTE R210W64 /c/Users/Public/Documents/repositorio/app (master)
$ echo > index.html

pabli@PABLODESK-DEVNOTE R210W64 /c/Users/Public/Documents/repositorio/app (master)
$ dir
index.html

pabli@PABLODESK-DEVNOTE R210W64 /c/Users/Public/Documents/repositorio/app (master)
$
  
```

Muestra nuevamente el estado actual de los archivos en el repositorio: **`git status`**

```

pabli@PABLODEIX-DEVNOTE RINGW4 /c/Users/Public/Documents/repositorio (master)
$ git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
pabli@PABLODEIX-DEVNOTE RINGW4 /c/Users/Public/Documents/repositorio (master)
$
  
```

Agrega todos los archivos al área de preparación: **git add** .

```

pabli@PABLODEIX-DEVNOTE RINGW4 /c/Users/Public/Documents/repositorio (master)
$ git add .
warning: in the working copy of 'app/index.html', LF will be replaced by CRLF the next time Git touches it

pabli@PABLODEIX-DEVNOTE RINGW4 /c/Users/Public/Documents/repositorio (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   app/index.html

pabli@PABLODEIX-DEVNOTE RINGW4 /c/Users/Public/Documents/repositorio (master)
$
  
```

Crea un commit con un mensaje descriptivo:
git commit -m "Primer commit"

```

pabli@PABLODEIX-DEVNOTE RINGW4 /c/Users/Public/Documents/repositorio (master)
$ git commit -m "Primer commit"
[master (root-commit) 5babb7a] Primer commit
1 file changed, 1 insertion(+)
 create mode 100644 app/index.html

pabli@PABLODEIX-DEVNOTE RINGW4 /c/Users/Public/Documents/repositorio (master)
$
  
```

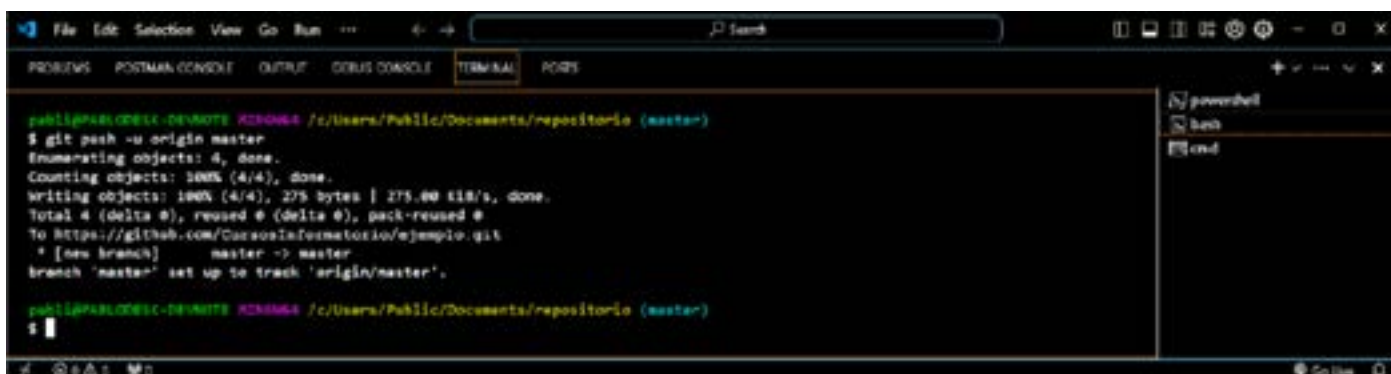
Conexión del repositorio local con Github:

Conexión del repositorio local con Github:

- a). Creación de un primer commit local.
- b). Demostración de cómo conectar el repositorio local con el repositorio en Github.
- c). Uso de comandos como **git remote** y **git push** para enviar los cambios al repositorio remoto.

Conecta el repositorio remoto **git remote add origin URL_del_repositorio_en_Github**

Envía los cambios al repositorio remoto **git push -u origin master**



```

public@PARLODESC-DEVINATE: ~/Documents/repositorio (master)
$ git push -u origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Writing objects: 100% (4/4), 275 bytes | 275.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/CursosInformatario/ejemplo.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.

public@PARLODESC-DEVINATE: ~/Documents/repositorio (master)
$
  
```

- d). Confirmar los cambios aplicados en el repositorio remoto de GitHub.
- e). Verificar en el la sección de commits.

Material de apoyo

Te mostraremos paso a paso la instalación de Git y las pantallas que aparecerán en el trayecto.

Instalación de Git

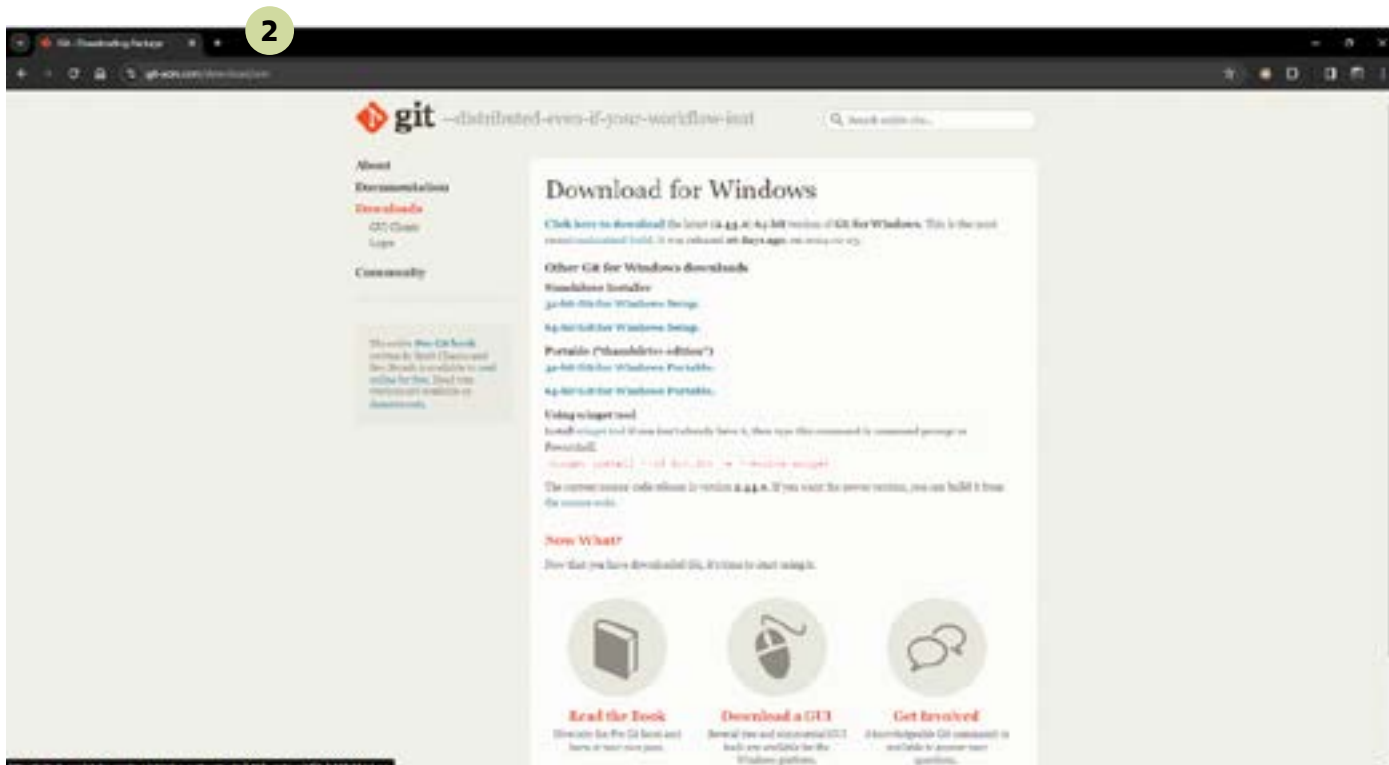
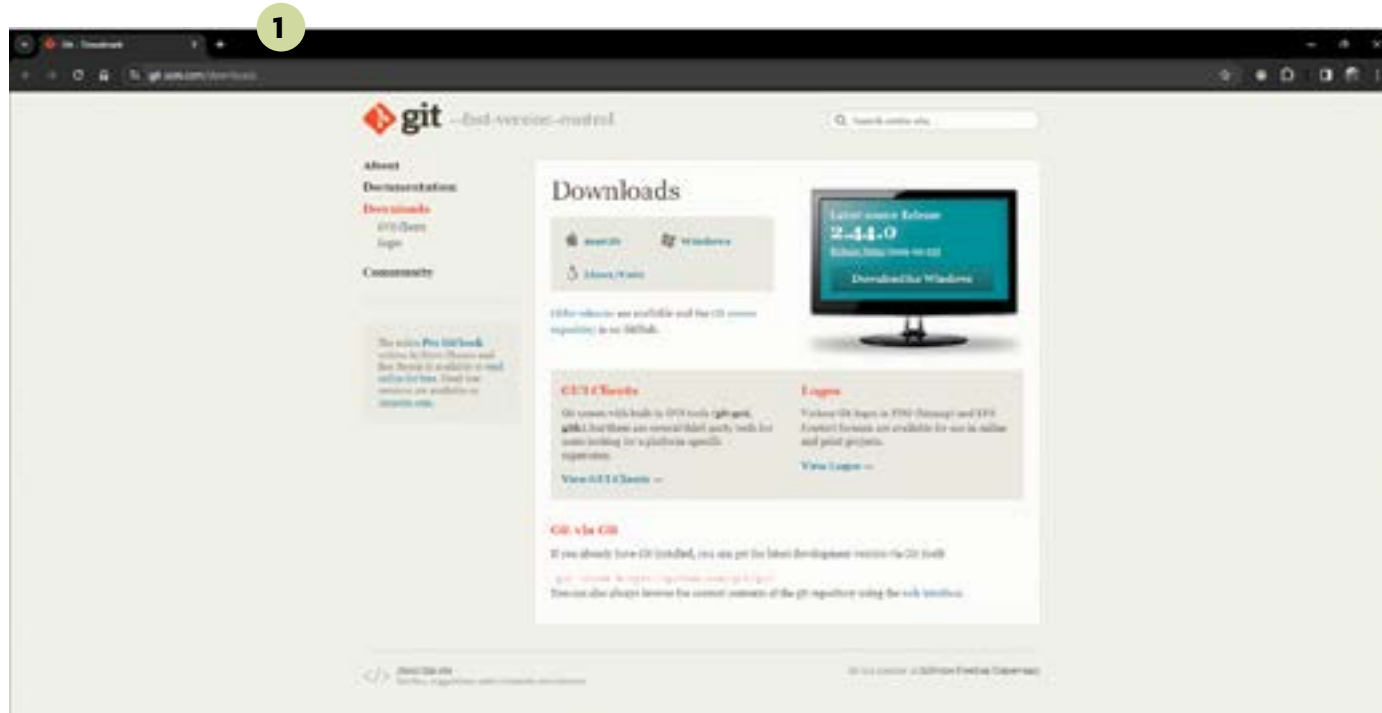
Enlaces directos a la descarga de Git para diferentes sistemas operativos.

► Descarga GIT: <https://git-scm.com/downloads>

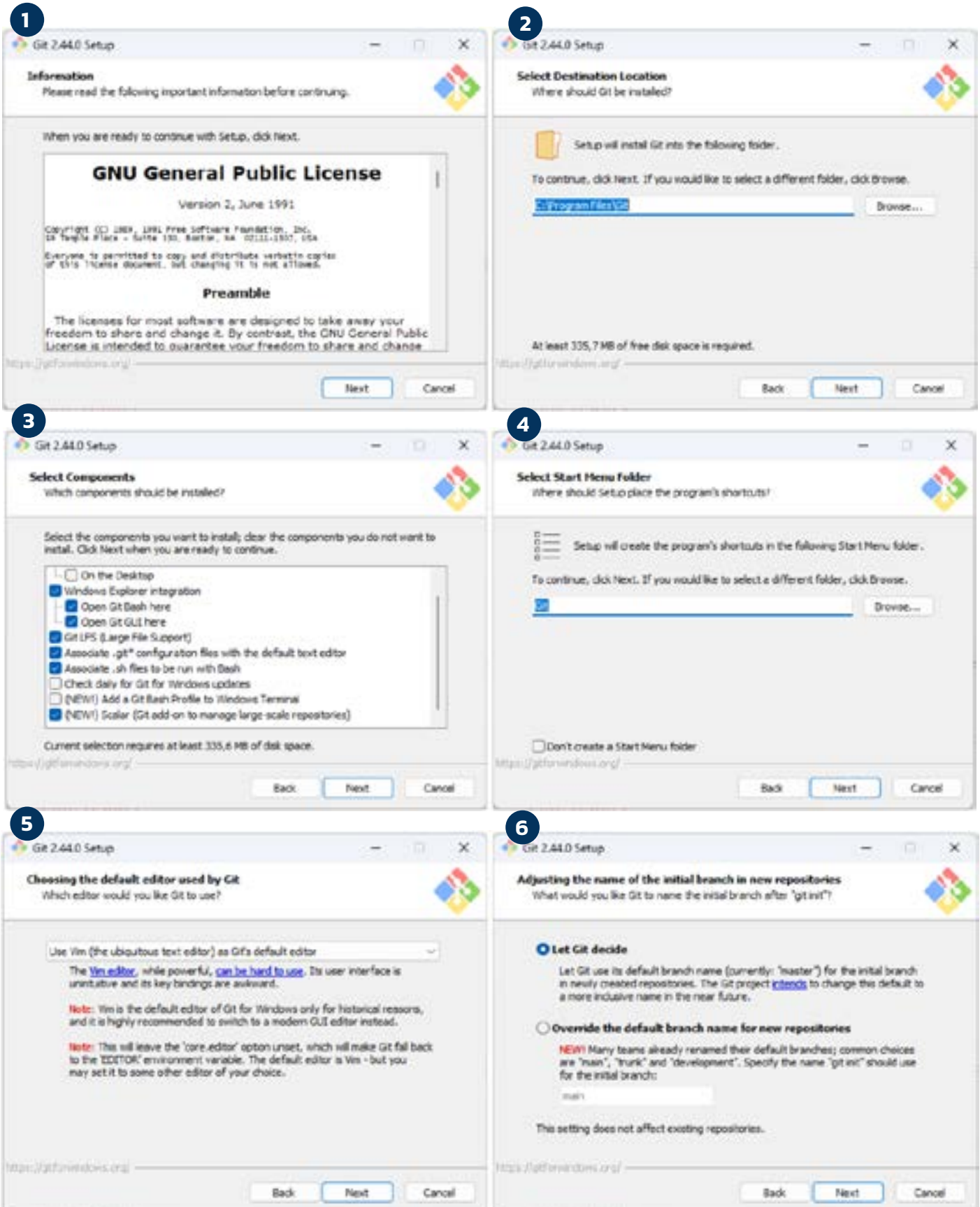
- Documentación GIT: <https://git-scm.com/doc>
- Página GITHUB: <https://github.com/>
- Documentación GITHUB: <https://docs.github.com/es>

Veamos las capturas de pantallas en cada paso de la instalación:

Ingreso desde el enlace para la descarga de Git:



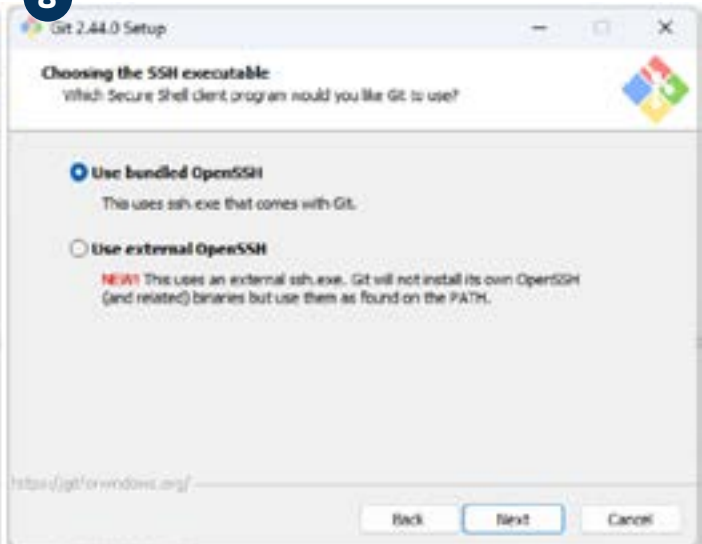
A partir de aquí la instalación y configuración del programa en nuestra PC:



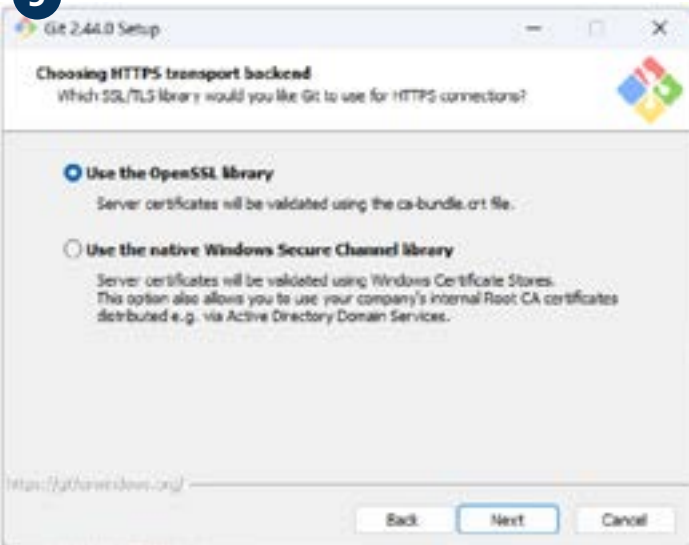
7



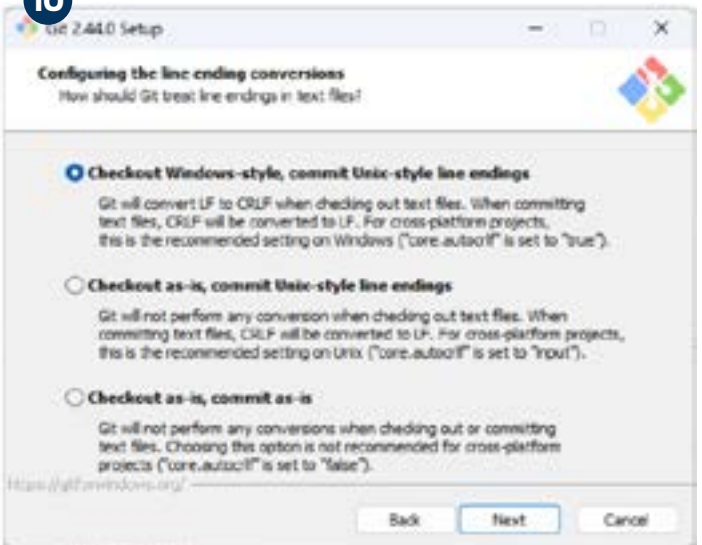
8



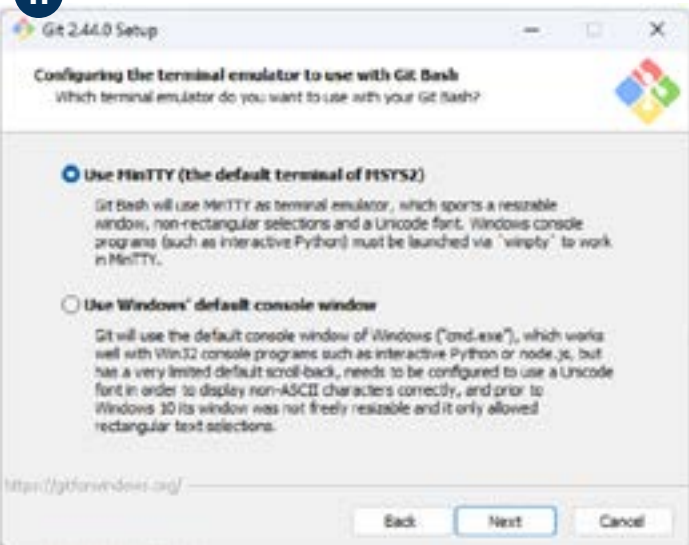
9



10



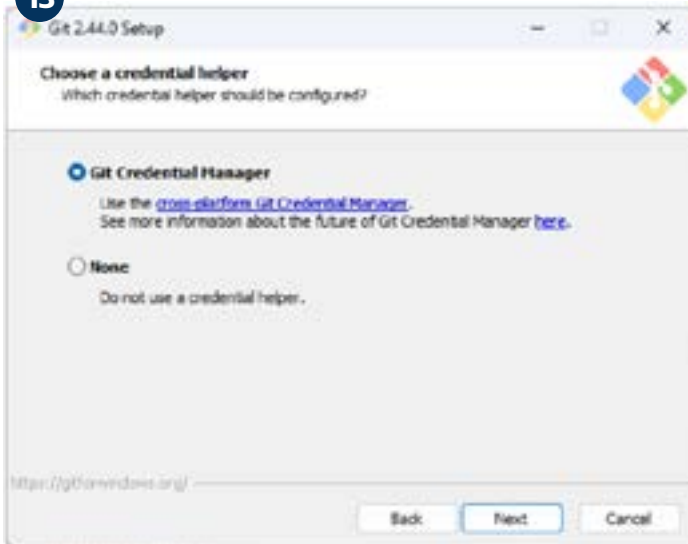
11



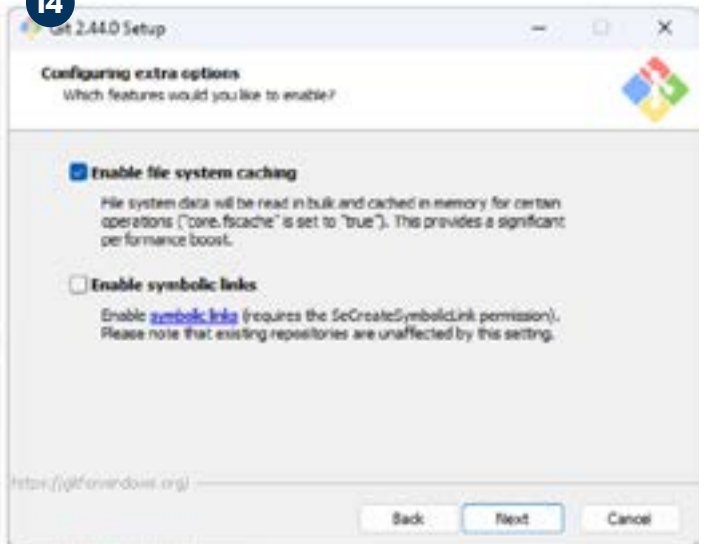
12



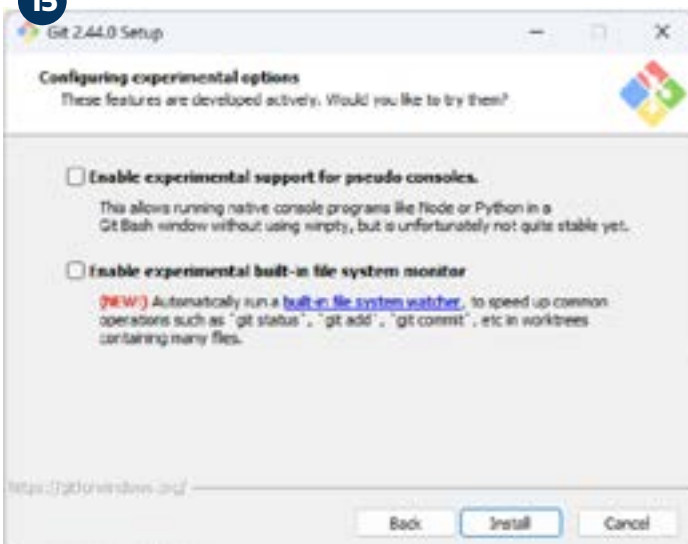
13



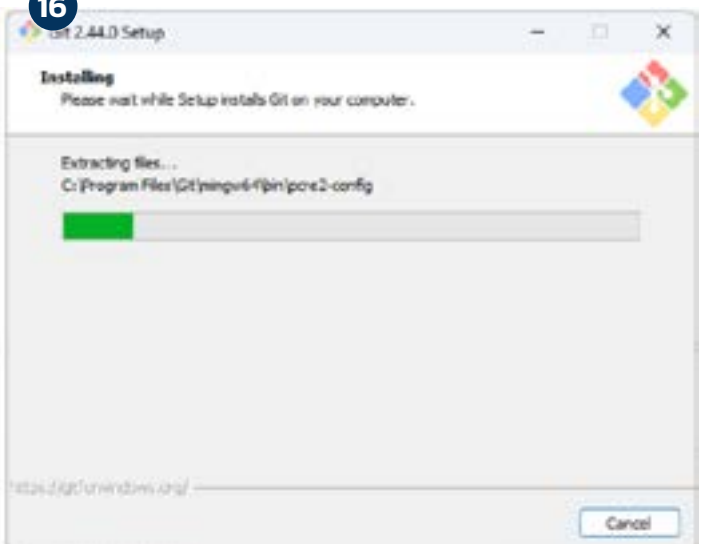
14



15



16

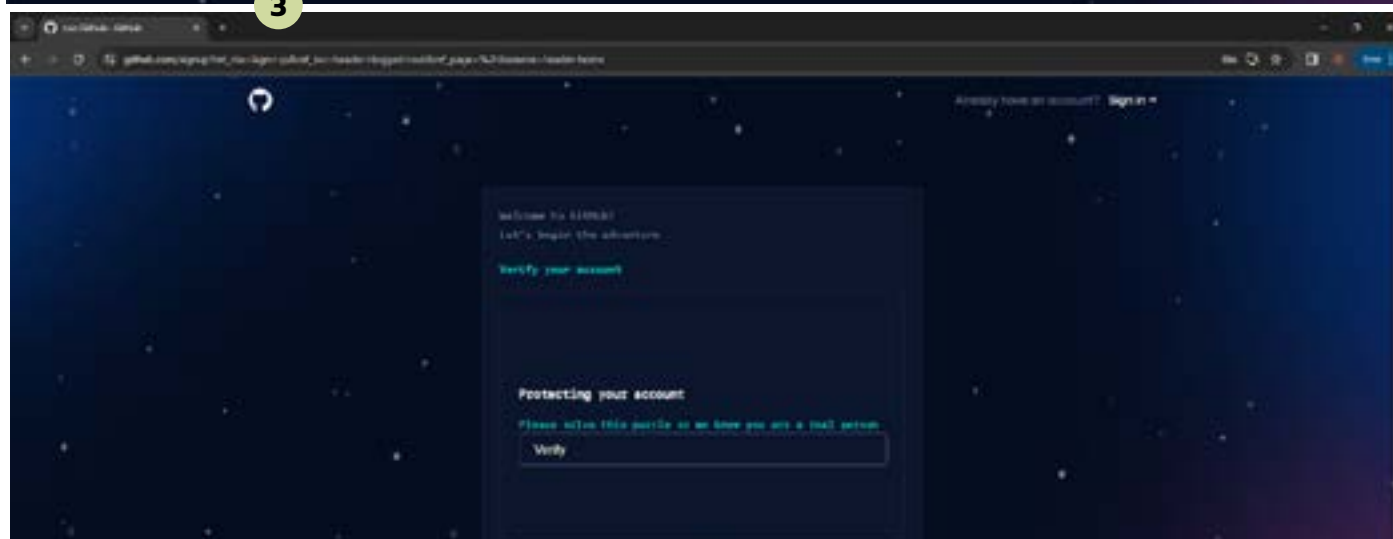
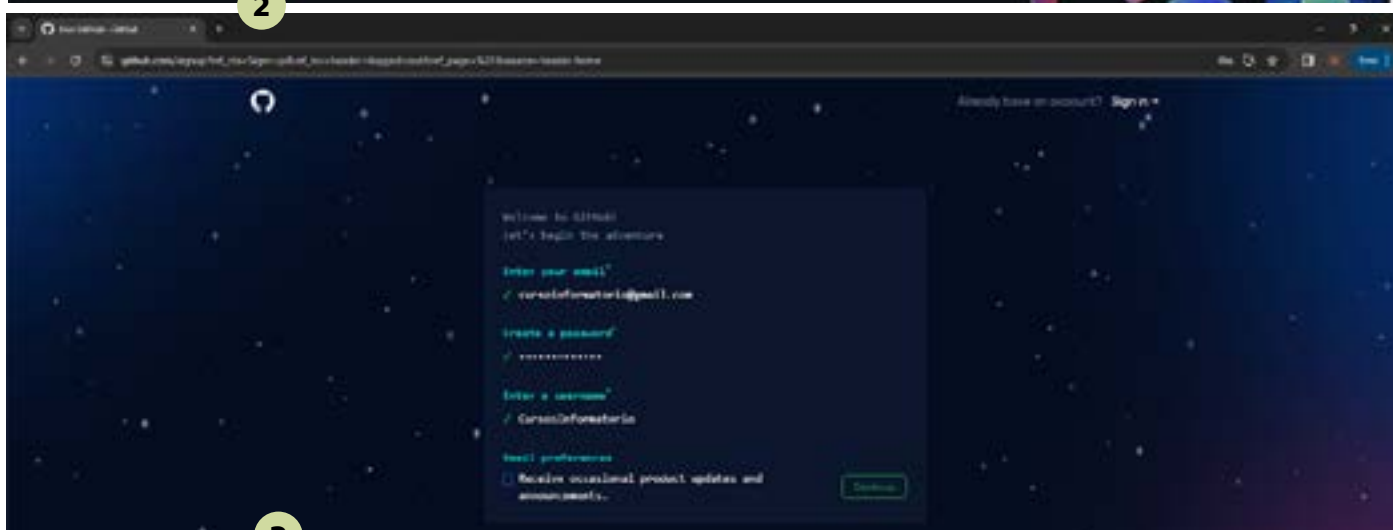
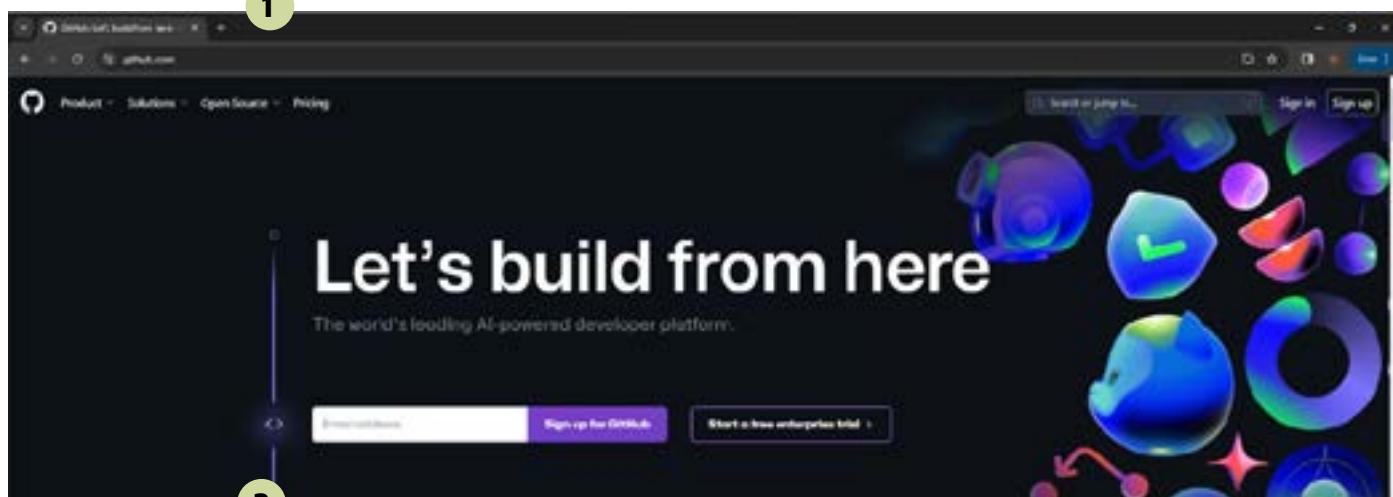


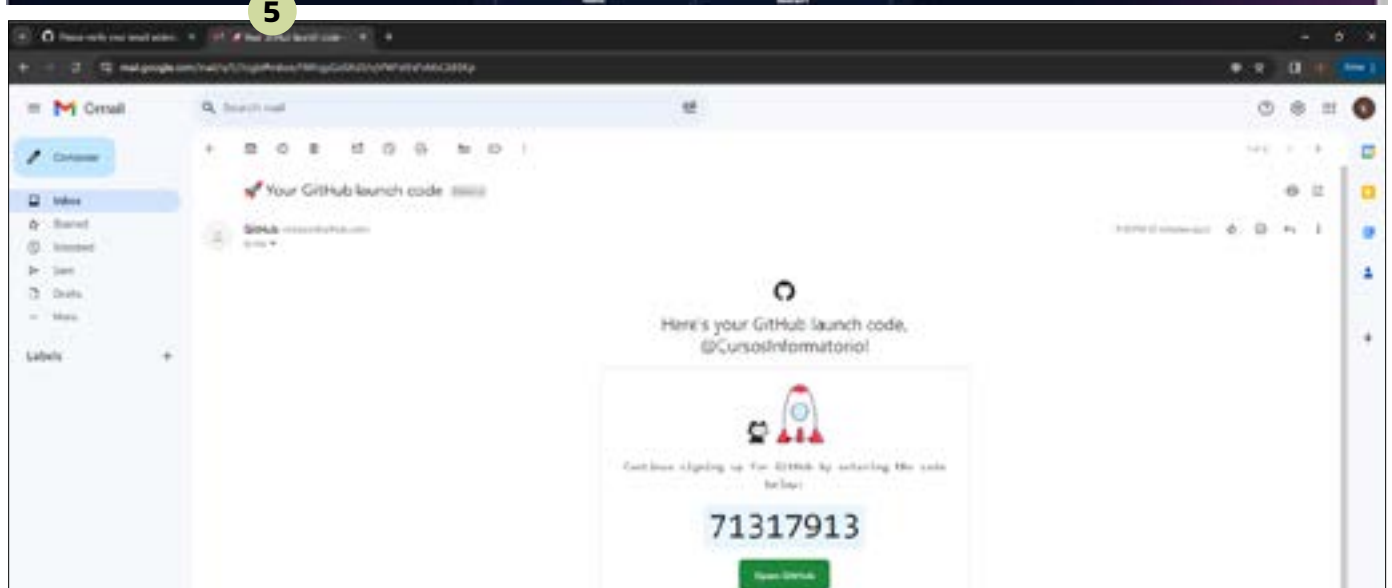
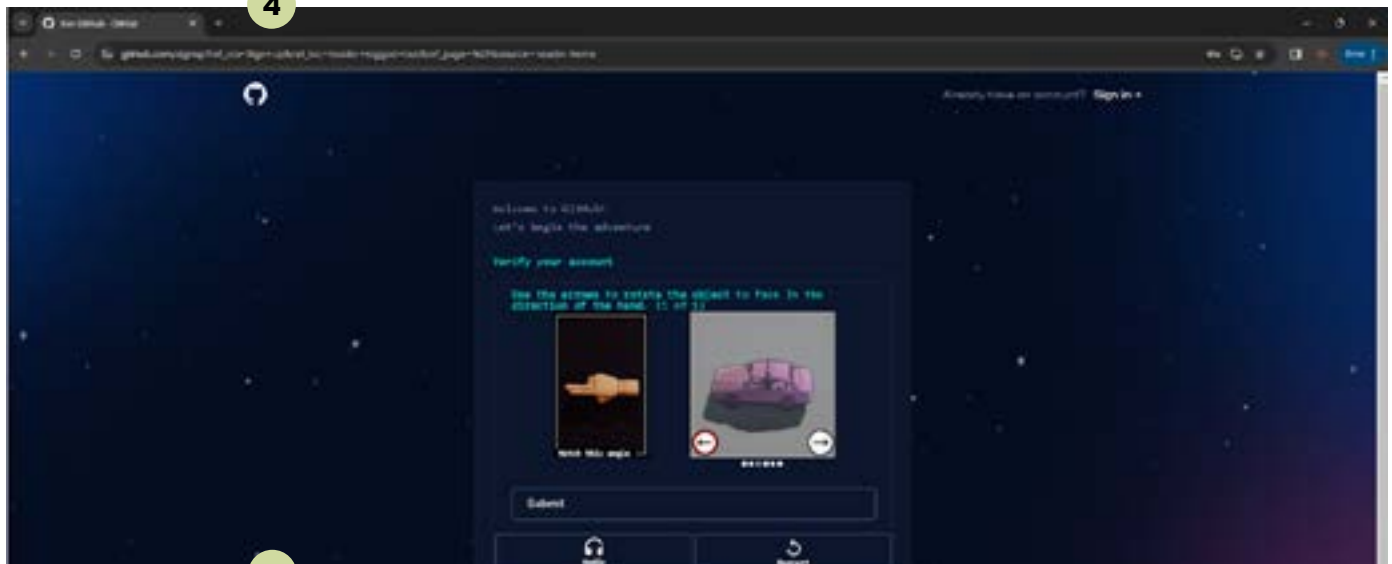
17



Creación de Cuenta en Github:

Te mostraremos a través de capturas el procedimiento paso a paso de la creación de cuenta en GitHub:







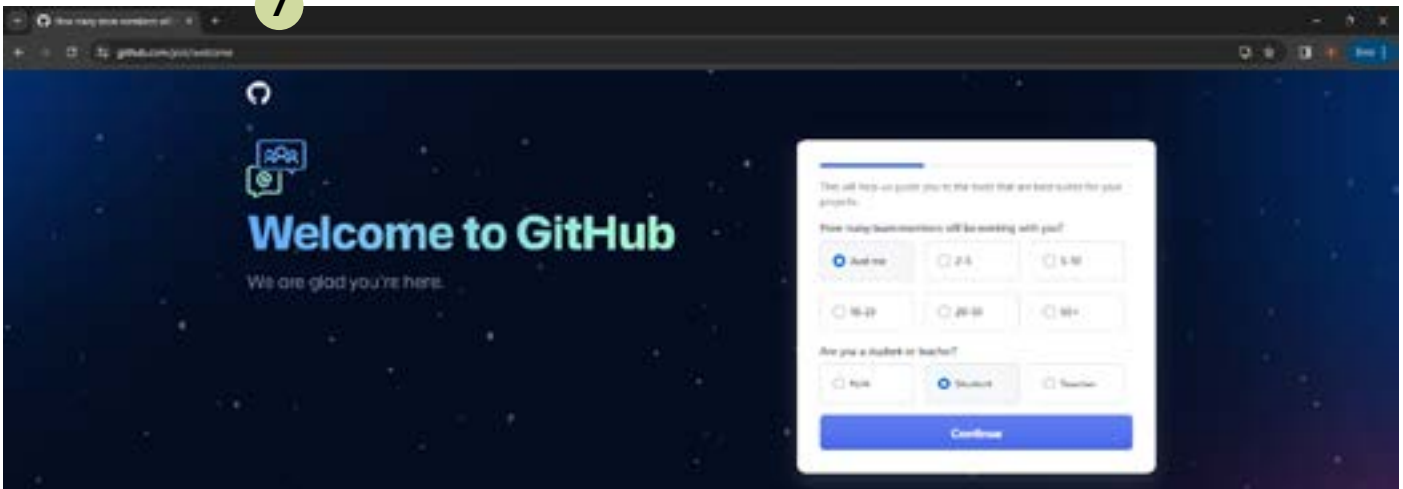
Gobierno del
CHACO

Ministerio
de la Producción y el Desarrollo
Económico Sostenible

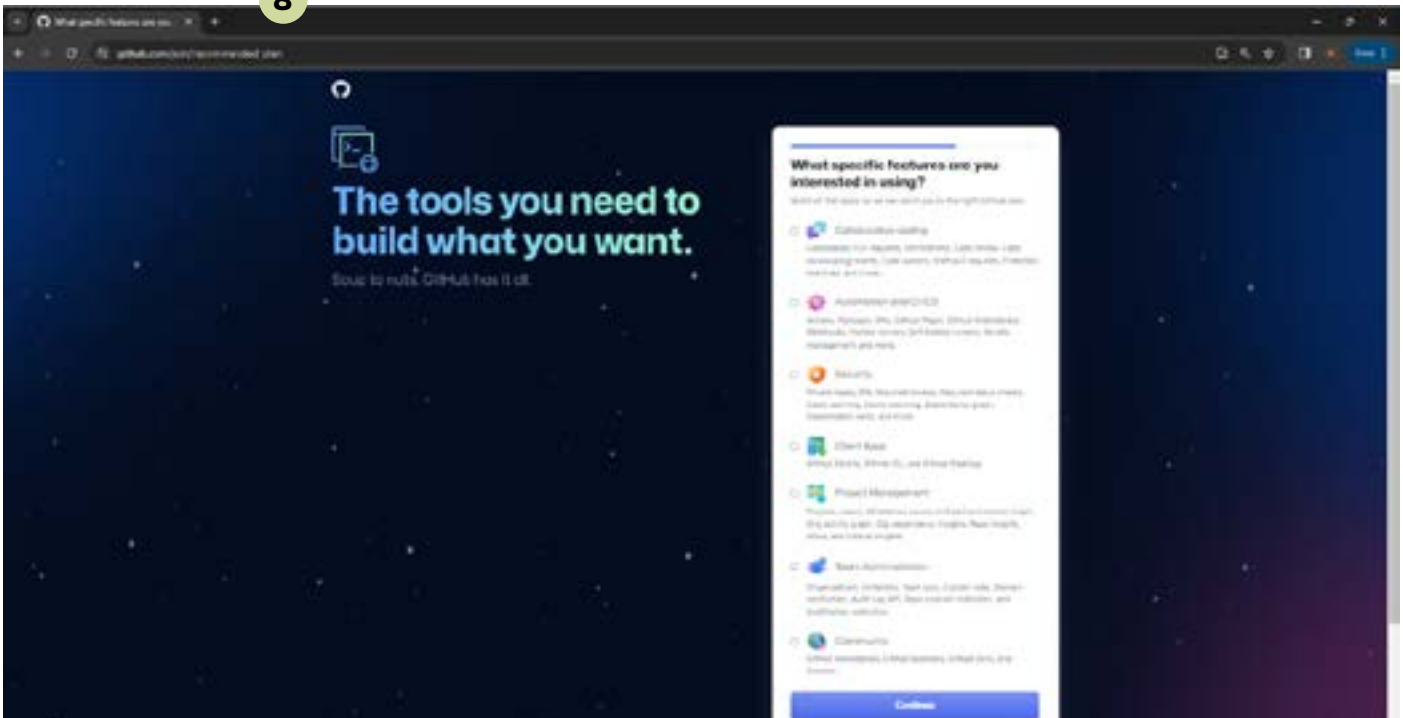


INFORMATARIO

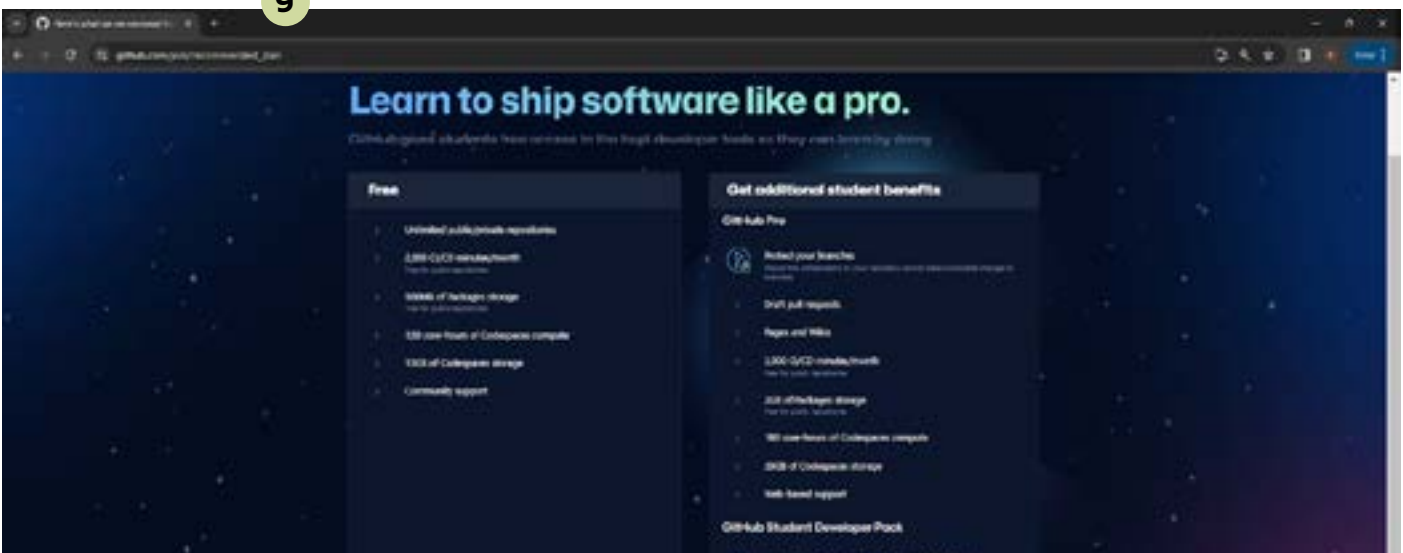
7



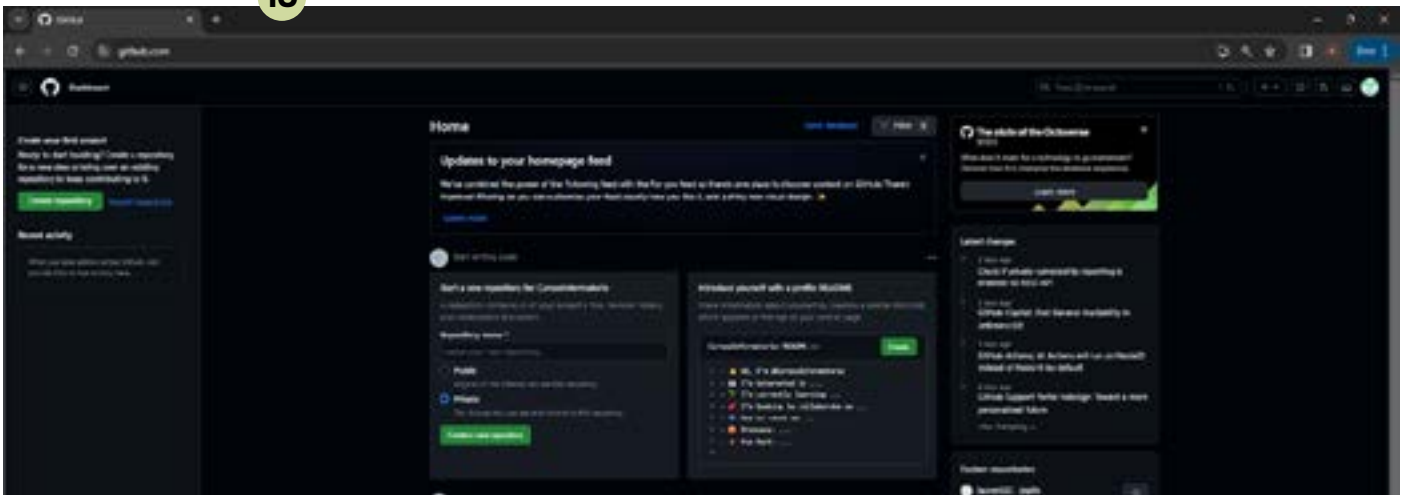
8



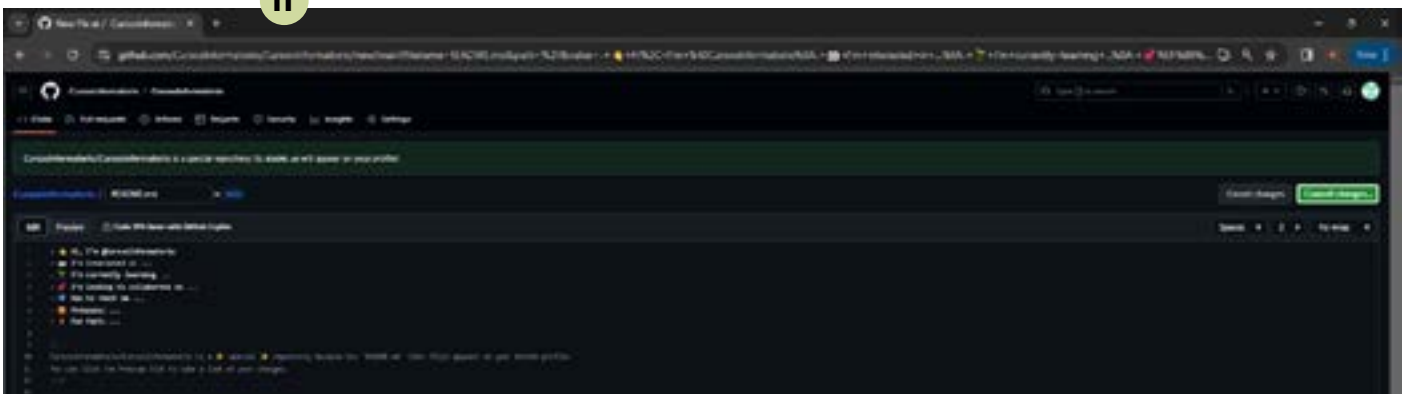
9



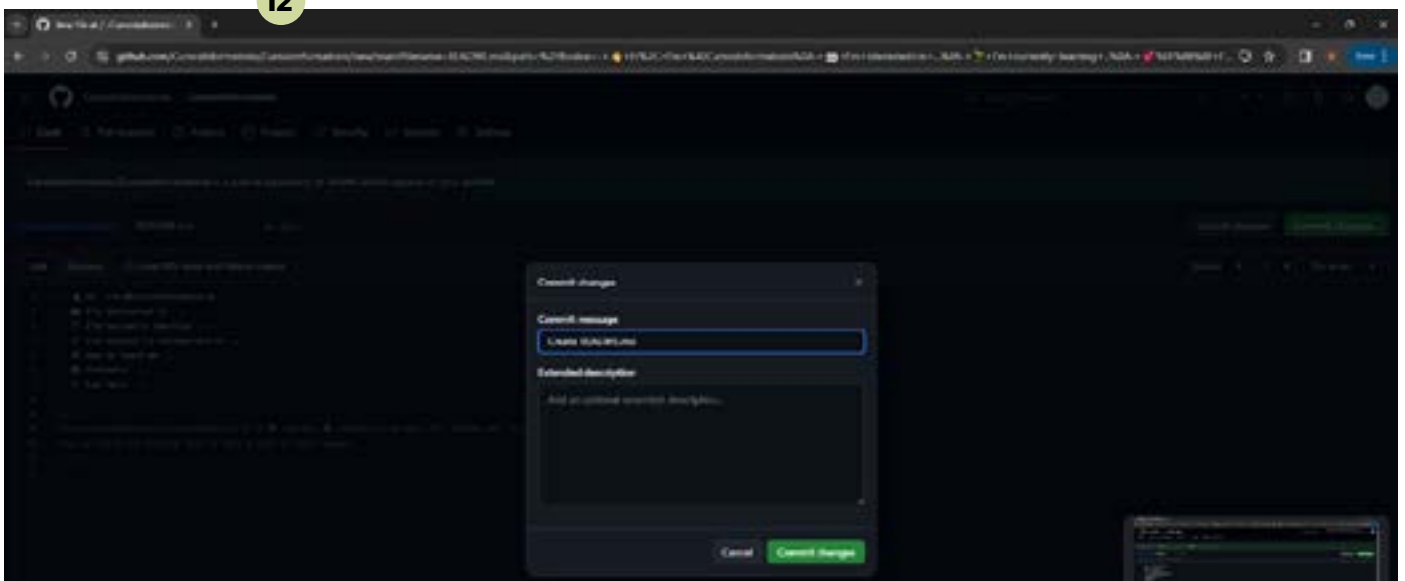
10



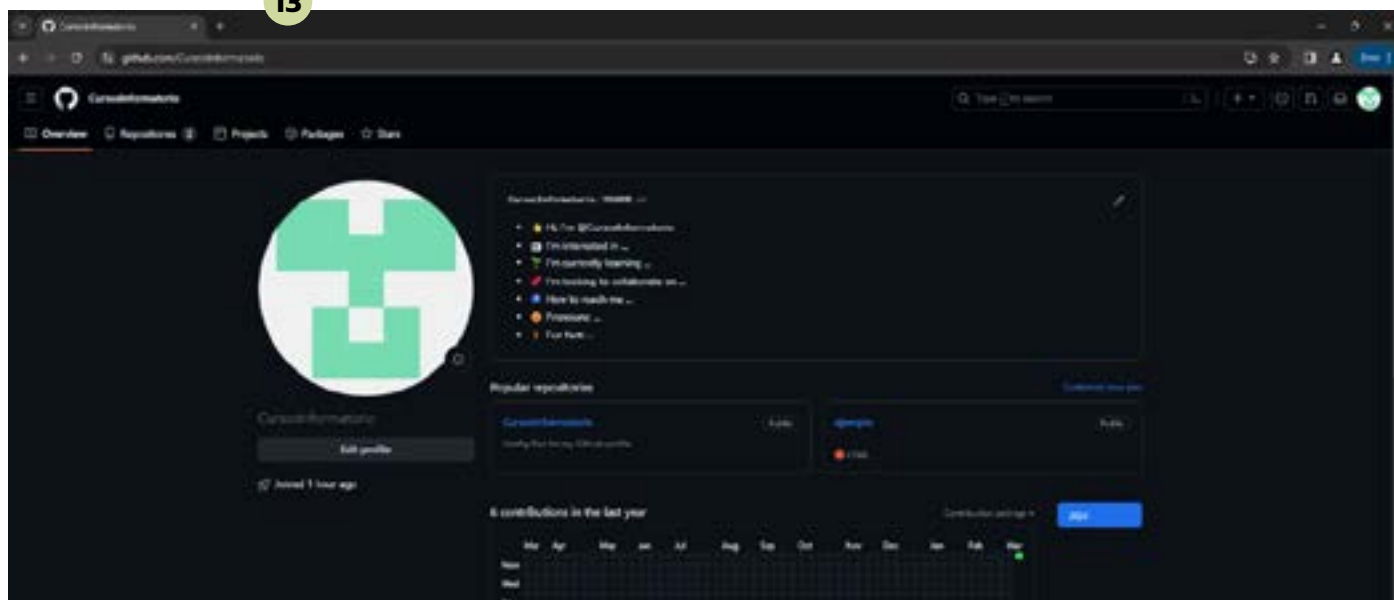
11



12



13



Iniciar con tu cuenta de Github:

<https://docs.github.com/es/get-started/onboarding/getting-started-with-your-github-account>

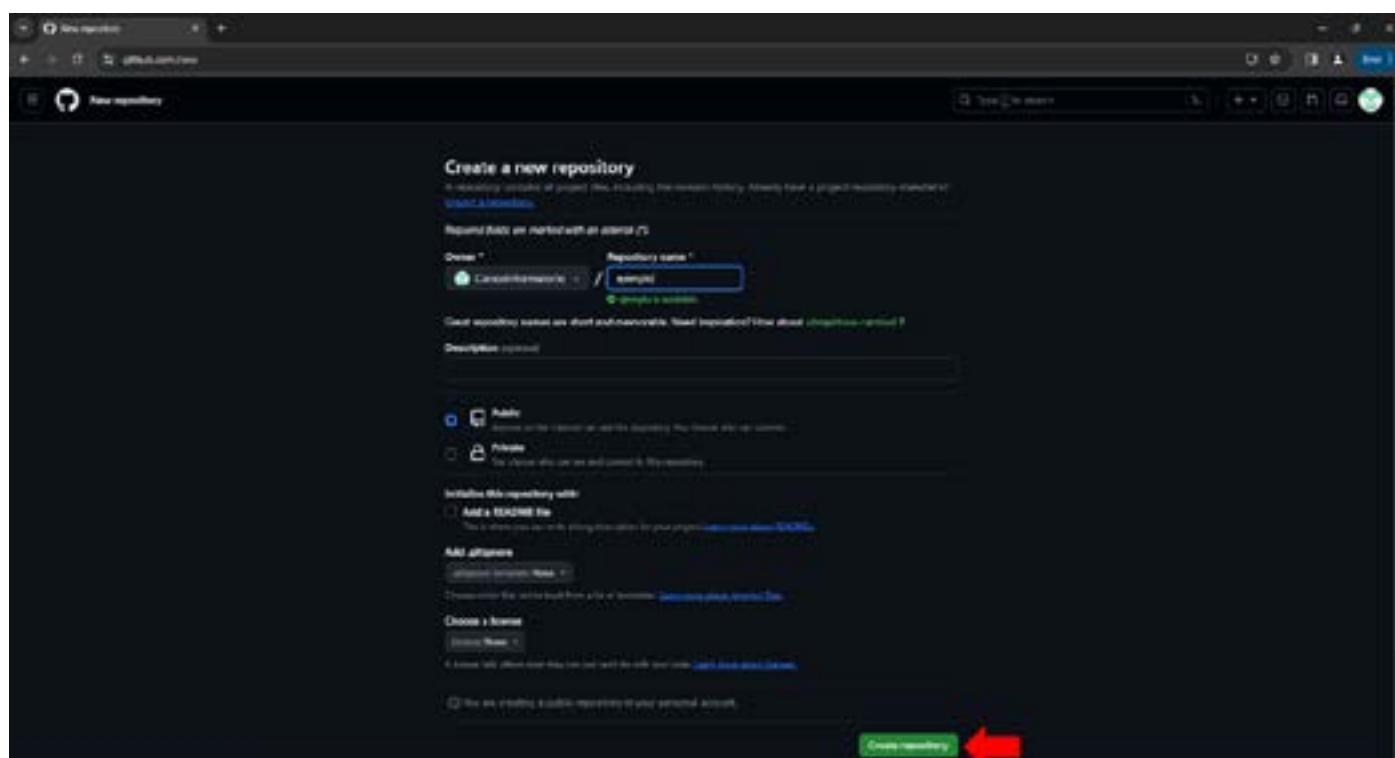
Repositorios en Github:

► Gráficos o diagramas de la estructura y funcionamiento de un repositorio.

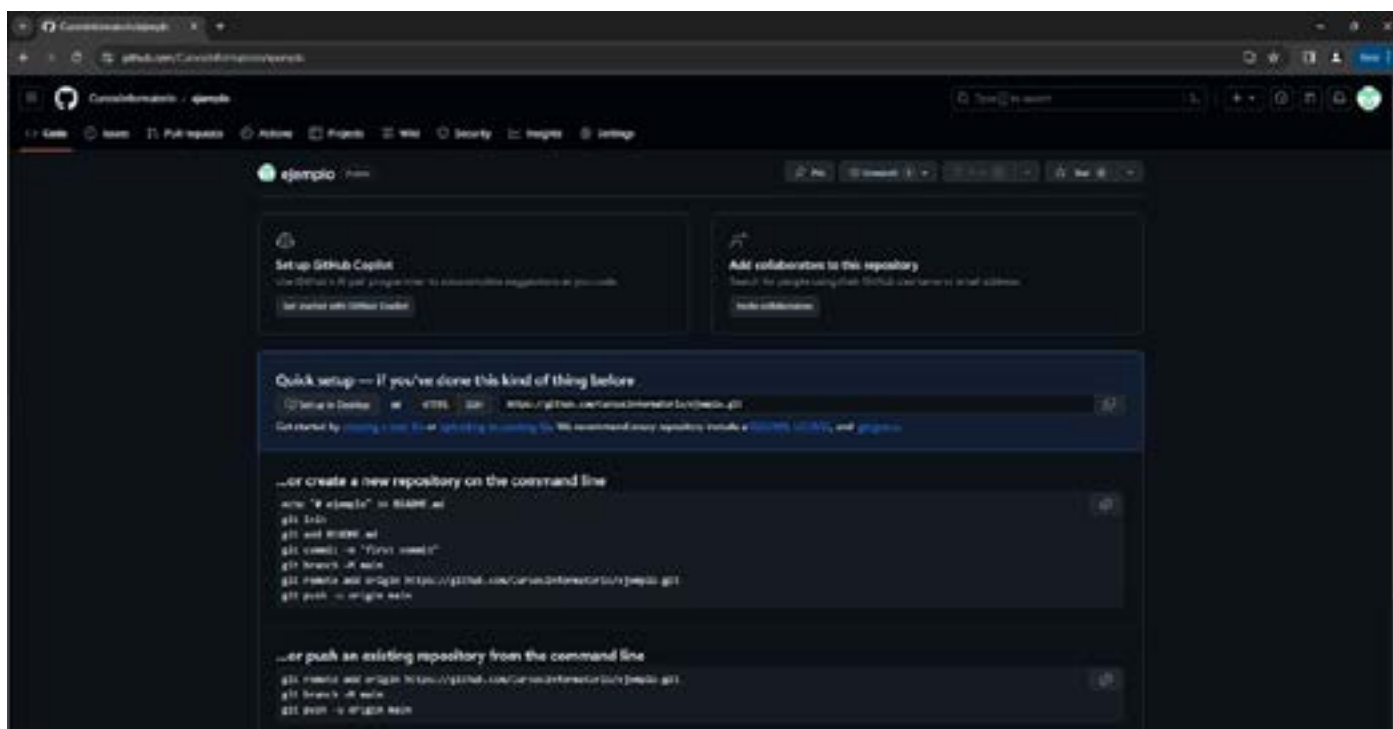


Creación de un Repositorio en Github:

► Capturas de pantalla de los pasos para crear un nuevo repositorio en Github.



Finalmente, el repositorio creado se presentaría en primera instancia como en la siguiente imagen:



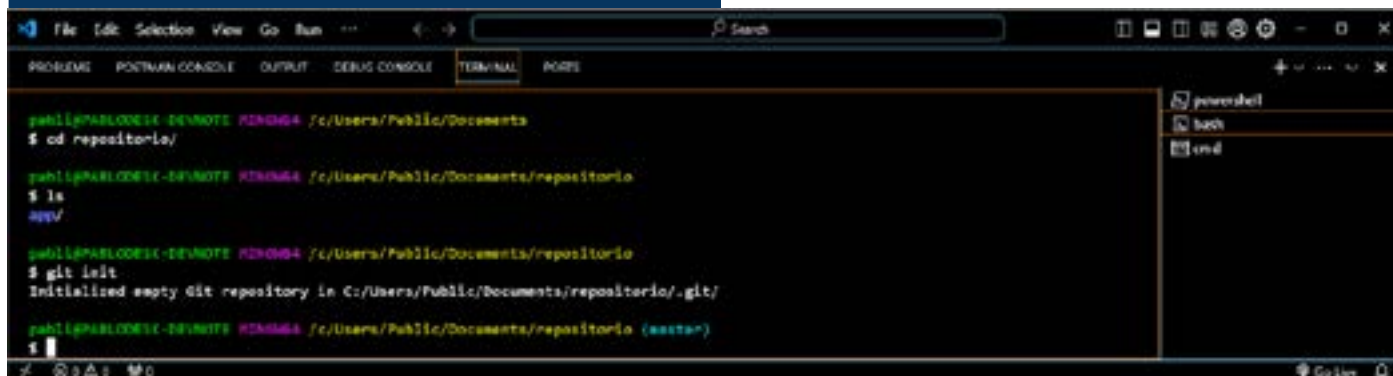
Inicio rápido para repositorios – Documentación de GitHub:

<https://docs.github.com/es/repositories/creating-and-managing-repositories/quicks-tart-for-repositories>

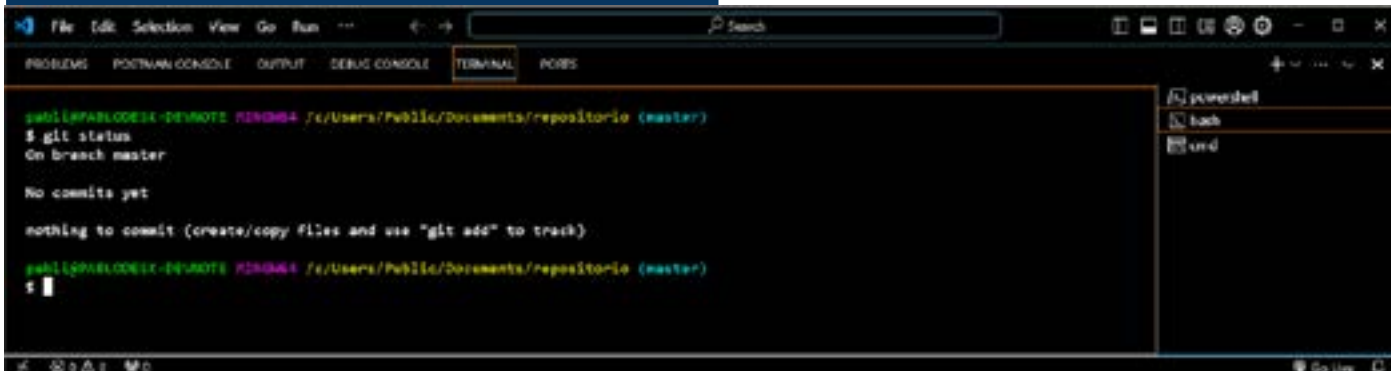
Inicialización de un Repositorio Local y Conexión con Github:

► Capturas de pantalla de comandos en la consola para inicializar, realizar commits y conectar el repositorio local con Github.

Inicializa un nuevo repositorio local: `git init`



Muestra el estado actual de los archivos en el repositorio: **git status**



```

publi@PABLODESI-DEVNOTE R214W64 /c/Users/Public/Documents/repositorio (master)
$ git status
On branch master

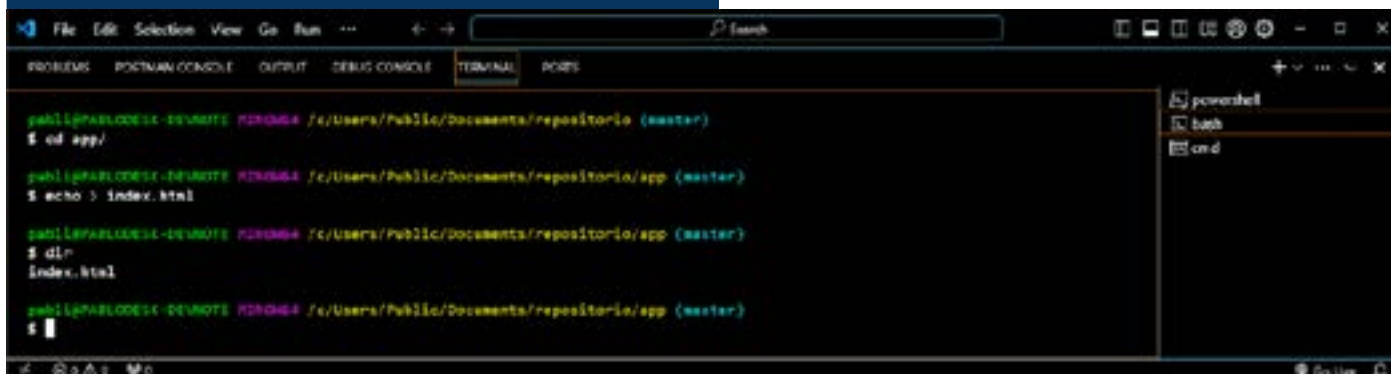
No commits yet

nothing to commit (create/copy files and use "git add" to track)

publi@PABLODESI-DEVNOTE R214W64 /c/Users/Public/Documents/repositorio (master)
$

```

Desplázate por los directorios y crea archivos: **cd** y **echo > index.html**



```

publi@PABLODESI-DEVNOTE R214W64 /c/Users/Public/Documents/repositorio (master)
$ cd app/

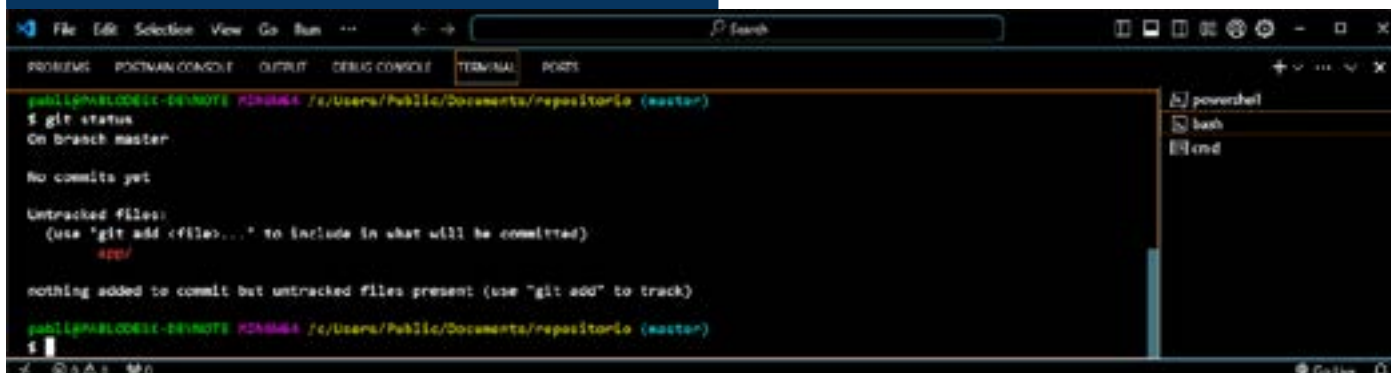
publi@PABLODESI-DEVNOTE R214W64 /c/Users/Public/Documents/repositorio/app (master)
$ echo > index.html

publi@PABLODESI-DEVNOTE R214W64 /c/Users/Public/Documents/repositorio/app (master)
$ dir
index.html

publi@PABLODESI-DEVNOTE R214W64 /c/Users/Public/Documents/repositorio/app (master)
$

```

Muestra nuevamente el estado actual de los archivos en el repositorio: **git status**



```

publi@PABLODESI-DEVNOTE R214W64 /c/Users/Public/Documents/repositorio (master)
$ git status
On branch master

No commits yet

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

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

publi@PABLODESI-DEVNOTE R214W64 /c/Users/Public/Documents/repositorio (master)
$

```


Agrega todos los archivos al área de preparación: **git add .**



```

pabli@PABLODESK-DEVNOTE R3INQM64 /c/Users/Public/Documents/repositorio (master)
$ git add .
warning: in the working copy of 'app/index.html', LF will be replaced by CRLF the next time Git touches it

pabli@PABLODESK-DEVNOTE R3INQM64 /c/Users/Public/Documents/repositorio (master)
$ git status
On branch master

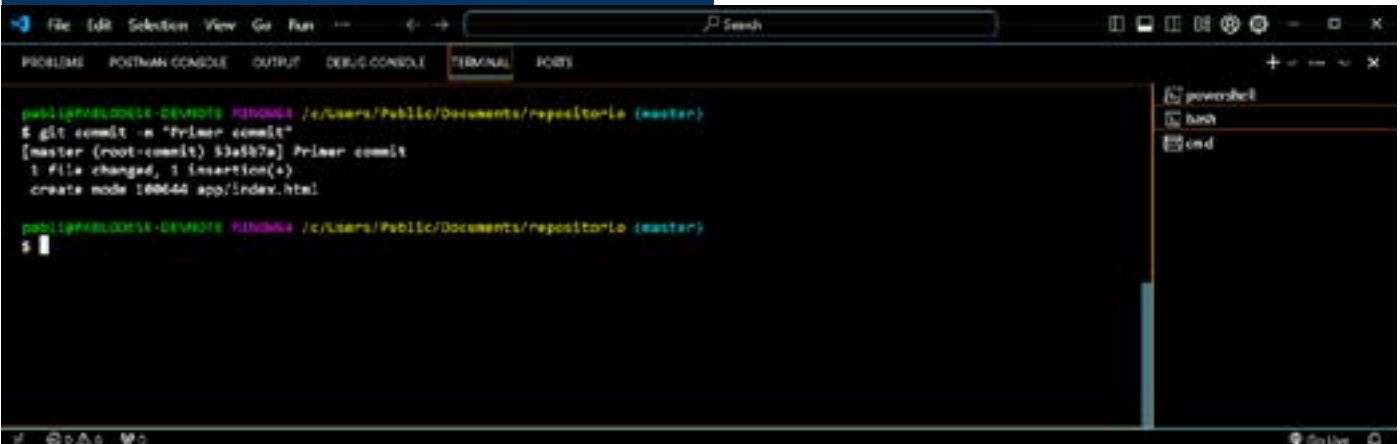
No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   app/index.html

pabli@PABLODESK-DEVNOTE R3INQM64 /c/Users/Public/Documents/repositorio (master)
$

```

Crea un commit con un mensaje descriptivo:
git commit -m "Primer commit"



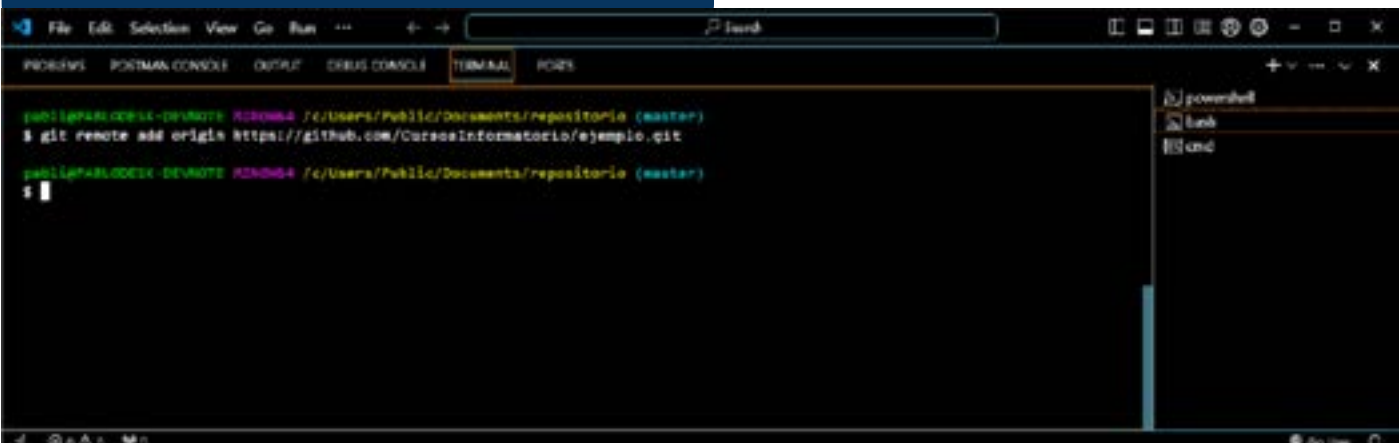
```

pabli@PABLODESK-DEVNOTE R3INQM64 /c/Users/Public/Documents/repositorio (master)
$ git commit -m "Primer commit"
[master (root-commit) 53a5b7a] Primer commit
 1 file changed, 1 insertion(+)
 create mode 100644 app/index.html

pabli@PABLODESK-DEVNOTE R3INQM64 /c/Users/Public/Documents/repositorio (master)
$

```

Conecta el repositorio remoto **git remote add origin URL_del_repositorio_en_Github**



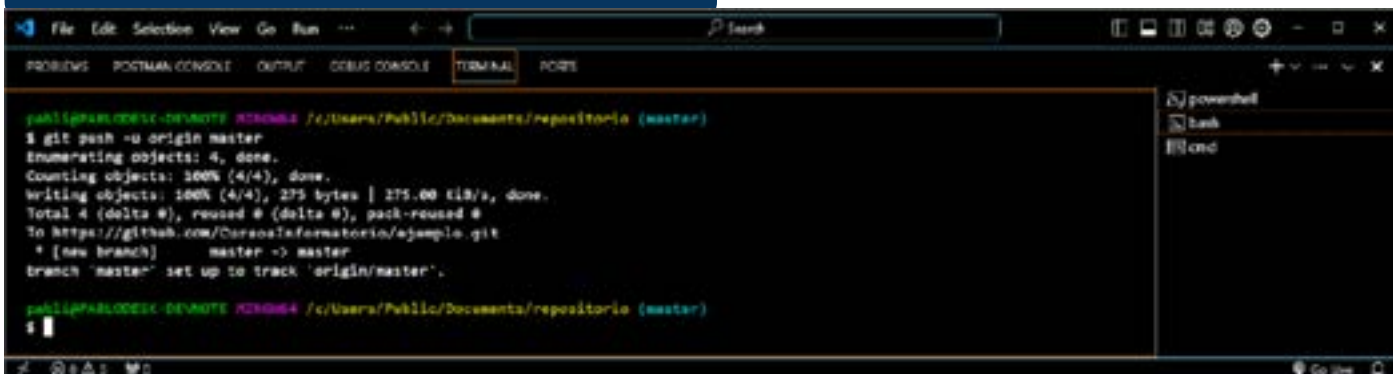
```

pabli@PABLODESK-DEVNOTE R3INQM64 /c/Users/Public/Documents/repositorio (master)
$ git remote add origin https://github.com/Cursosinformatorio/ejemplo.git

pabli@PABLODESK-DEVNOTE R3INQM64 /c/Users/Public/Documents/repositorio (master)
$

```


Envía los cambios al repositorio remoto **git push -u origin master**



```

publi@PARLODESK-DEVNOTE MINGW64 /c:/Users/Public/Documents/repositorio (master)
$ git push -u origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Writing objects: 100% (4/4), 275 bytes | 275.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/ChacoInformatorio/ajemplo.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.

publi@PARLODESK-DEVNOTE MINGW64 /c:/Users/Public/Documents/repositorio (master)
$
  
```

Comandos Git: Operaciones Básicas y Visualización del Área de Staging

Operaciones Básicas con Git:

1. DETALLES DE LOS COMANDOS

STATUS, ADD, COMMIT, Y PUSH:

- a). Muestra el estado actual del repositorio, destacando cambios no rastreados, modificados y en el área de staging: **git status**
- b). Agrega archivos específicos al área de staging, listos para ser incluidos en el próximo commit: **git add <archivo>** o **git add .** para todos los archivos.
- c). Crea un nuevo commit con los cambios en el área de staging, acompañado de un mensaje descriptivo: **git commit -m "Mensaje del commit"**
- d). Envía los commits locales al repositorio remoto en Github: **git push**

2. VISUALIZACIÓN DEL

ÁREA DE STAGING Y EXPLICACIÓN DE STAGES:

a). VISUALIZACIÓN DEL ÁREA DE STAGING:

► Mostrar cómo los cambios pasan de no rastreados a modificados y finalmente al área de staging.

► Uso de git status para visualizar estos cambios.

► Destacar la importancia del área de staging como una etapa intermedia antes de realizar un commit.

b). EXPLICACIÓN DE STAGES:

► **Área de trabajo:** Donde trabajamos los archivos y realizamos cambios en el proyecto. Incluye todos los archivos del proyecto, ya sean rastreados o no por Git.

► **Cambios no rastreados:** Archivos que existen en el directorio de trabajo pero aún no han sido identificados por Git.

► **Cambios modificados:** Archivos que han sido modificados y son conocidos por

Git pero aún no están en el área de staging.

► **Área de Staging:** Donde se preparan los cambios específicos que se incluirán en el próximo commit.

► **Repositorio local:** Contenedor donde se almacenan todos los commits y la historia del proyecto. Cada vez que realizas un commit, se crea un nuevo punto en la historia del repositorio que registra los cambios realizados en el proyecto en ese momento.

Material de Apoyo:

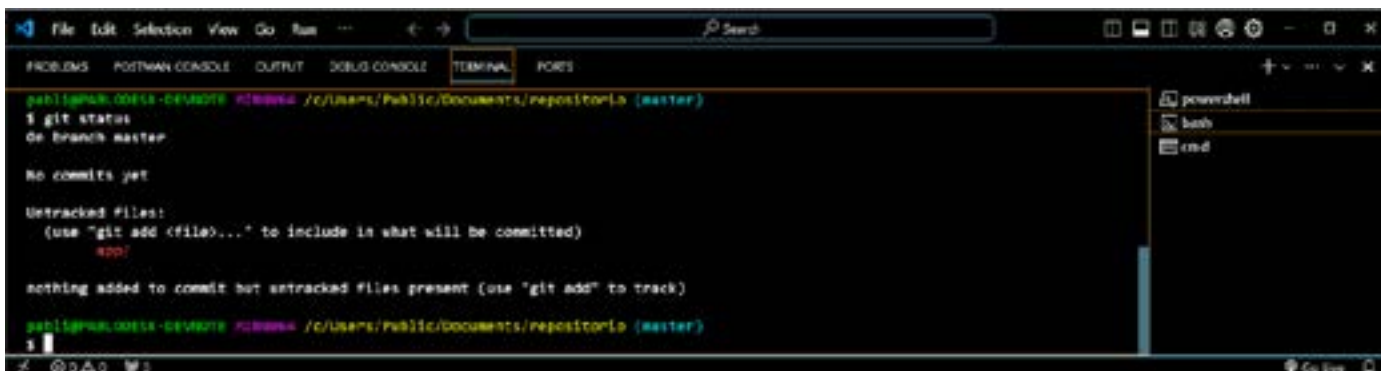
Operaciones Básicas con Git:

1. GIT STATUS:**a). ESCENARIO 1 (CAMBIOS NO RASTREADOS):**

► Descripción: Mostrar cómo aparecen los cambios no rastreados.

► Comando: **git status**

► Captura de Pantalla: (Captura que muestra archivos no rastreados).



```

publi@PS-C0615-DE-VARDE /c:/Users/Public/Documents/repositorio (master)
$ git status
On branch master

No commits yet

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

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

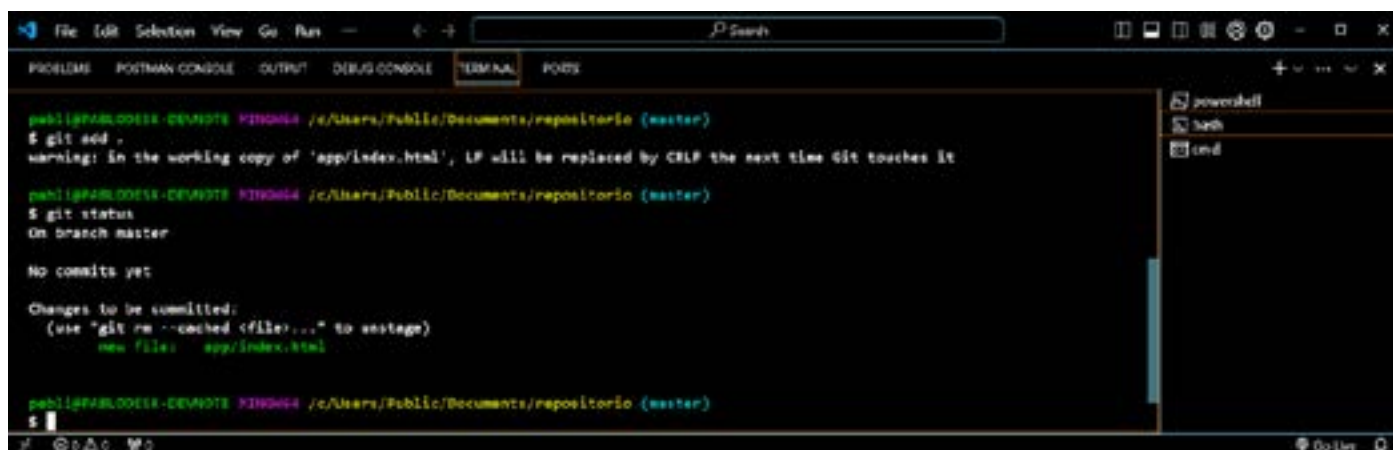
publi@PS-C0615-DE-VARDE /c:/Users/Public/Documents/repositorio (master)
$
  
```

b) ESCENARIO 2 (CAMBIOS EN EL ÁREA DE STAGING):

► Descripción: Destacar la diferencia después de agregar archivos al área de staging.

► Comando: **git add <archivo>** y luego **git status**

► Captura de Pantalla: (Captura que muestra archivos en el área de staging).



```

publi@PABLODEIX-DEVNOTE: ~$ git add .
warning: in the working copy of 'app/index.html', LF will be replaced by CRLF the next time Git touches it

publi@PABLODEIX-DEVNOTE: ~$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   app/index.html

publi@PABLODEIX-DEVNOTE: ~$
  
```

2. GIT ADD:

a) ESCENARIO 1 (AGREGAR ARCHIVOS ESPECÍFICOS):

► Descripción: Ejemplo de agregar archivos específicos al área de staging.

► Comando: **git add <archivo>**

► Captura de Pantalla: (Captura que muestra un archivo específico en el área de staging).



```

On branch master
PS C:\Users\Public\Documents\repositorio> git add app/main.js
PS C:\Users\Public\Documents\repositorio> git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   app/main.js

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


PS C:\Users\Public\Documents\repositorio>
  
```

b) ESCENARIO 2 (AGREGAR TODOS LOS ARCHIVOS):

► Descripción: Mostrar cómo agregar todos los archivos al área de staging.

► Comando: **git add .**

► Captura de Pantalla: (Captura que muestra todos los archivos en el área de staging).



```

PS C:\Users\Public\Documents\repositorio> git add .
PS C:\Users\Public\Documents\repositorio> git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   .gitignore
        new file:   app/main.js
PS C:\Users\Public\Documents\repositorio>
  
```

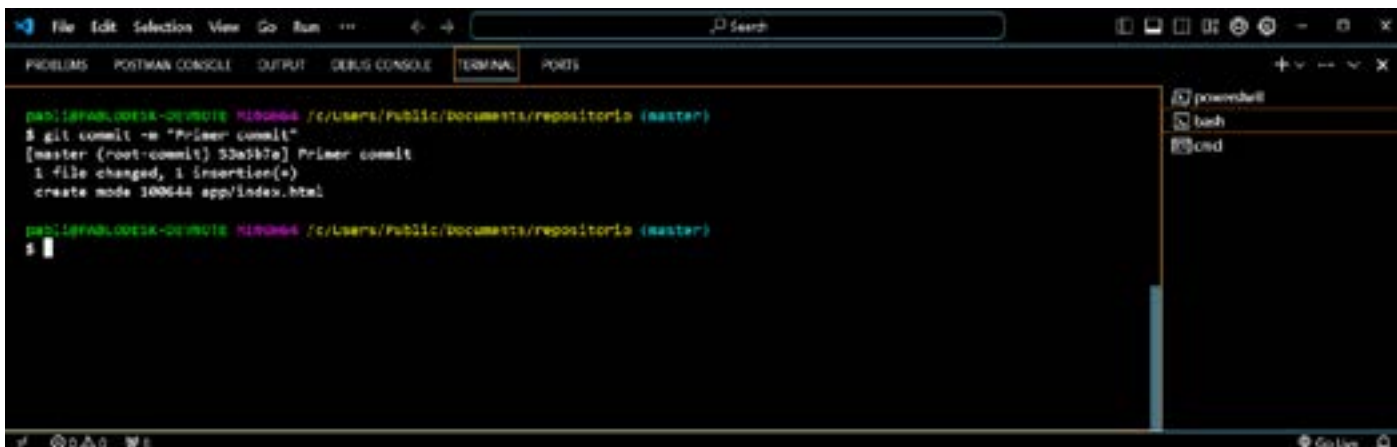
3. GIT COMMIT -m "MENSAJE DEL COMMIT":

a) ESCENARIO 1 (REALIZAR UN COMMIT):

► Descripción: Ejemplo de realizar un commit con un mensaje descriptivo.

► Comando: **git commit -m "Mensaje del commit"**

► Captura de Pantalla: (Captura que muestra el resultado después de un commit).



```

pablo@PABLODESK-DEV0016: ~/Documents/repositorio (master)
$ git commit -m "Primer commit"
[master (root-commit) 53a5b7a] Primer commit
1 file changed, 1 insertion(+)
 create mode 100644 app/index.html

pablo@PABLODESK-DEV0016: ~/Documents/repositorio (master)
$
  
```

4. GIT PUSH:

a) ESCENARIO 1 (ENVIAR CAMBIOS AL REPOSITORIO REMOTO):

- Descripción: Mostrar cómo enviar los cambios al repositorio remoto en Github.
- Comando: **git push**
- Captura de Pantalla: (Captura que muestra el resultado después de un push).

```

publi@PABLODESA-DEVNOTE: /c/Users/Public/Documents/repositorio (master)
$ git push -u origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Writing objects: 100% (4/4), 275 bytes | 275.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/CursosInformatario/ejemplo.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.

publi@PABLODESA-DEVNOTE: /c/Users/Public/Documents/repositorio (master)
$
  
```

VISUALIZACIÓN DE LAS ÁREAS



CAMBIOS No RASTREADOS:

Visual: (Imagen que representa cambios no rastreados).



```

PS C:\Users\Public\Documents\repositorio> git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        app/README.md
        app/archivo_ignorado.txt
        app/main.js

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\Public\Documents\repositorio>
  
```

CAMBIOS MODIFICADOS:

Visual: (Imagen que representa cambios modificados)

Aquí un archivo **.gitignore** nos ayudará a no rastrear, o como en su nombre lo dice, ignorar archivos. Es especialmente útil en situaciones donde tenemos archivos generados automáticamente (por ejemplo, archivos compilados, archivos temporales) o configuraciones locales que no deberían ser compartidos en el repositorio.



```

PS C:\Users\Public\Documents\repositorio> git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        app/README.md
        app/main.js

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\Public\Documents\repositorio>
  
```


ÁREA DE STAGING:

Visual: (Imagen que representa el área de staging con archivos listos para el commit).



```

PS C:\Users\Public\Documents\repositorio> git add app/README.md
PS C:\Users\Public\Documents\repositorio> git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   app/README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        app/main.js
  
```

COMMIT:

Visual: (Imagen que representa el resultado después de realizar un commit).



```

PS C:\Users\Public\Documents\repositorio> git commit -m "add README.md"
[master 868dcd1] add README.md
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 app/README.md
PS C:\Users\Public\Documents\repositorio> git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        app/main.js

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

Consejos Prácticos:

a). Eficiencia en el Área de Staging:

Utilizar el área de staging para agrupar cambios relacionados y realizar commits más coherentes.

b). Mensajes de Commit Descriptivos:

Escribir mensajes de commit claros y descriptivos para que cualquier colaborador pueda entender los cambios realizados.

c). Revisión Antes de Hacer Commit:

Antes de hacer un commit, revisar los cambios en el área de staging para asegurarse de incluir todo lo necesario y mantener la coherencia.

d). Uso de Ramas:

Considera el uso de ramas para desarrollar nuevas características o correcciones, manteniendo la rama principal (main/master) limpia y estable.

Otros Comandos de Git y Ejemplos Prácticos

1. GIT FETCH:

a). El comando **git fetch** descarga cambios desde el repositorio remoto, pero no realiza fusiones automáticas ni modifica el trabajo actual.

b). Por ejemplo – Descarga los cambios del repositorio remoto llamado "origin": **git fetch origin**

Capturas de pantalla que ilustren el resultado después de ejecutar **git fetch**

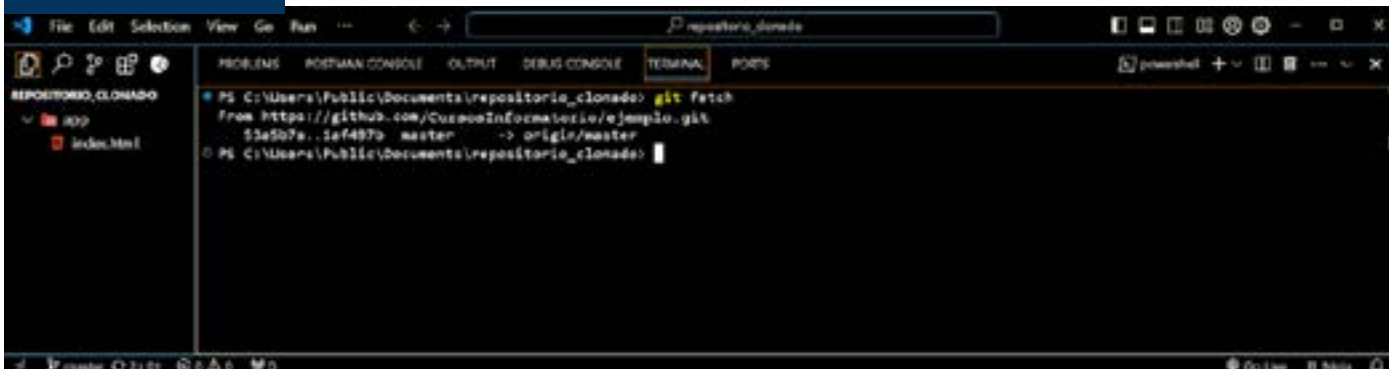
2. GIT PULL:

a). **git pull** combina **git fetch** y **git merge**. Descarga cambios del repositorio remoto y fusiona automáticamente los cambios en la rama local.

b). Por ejemplo – Descarga cambios de la rama principal del repositorio remoto y fusiona en la rama local: **git pull origin master**

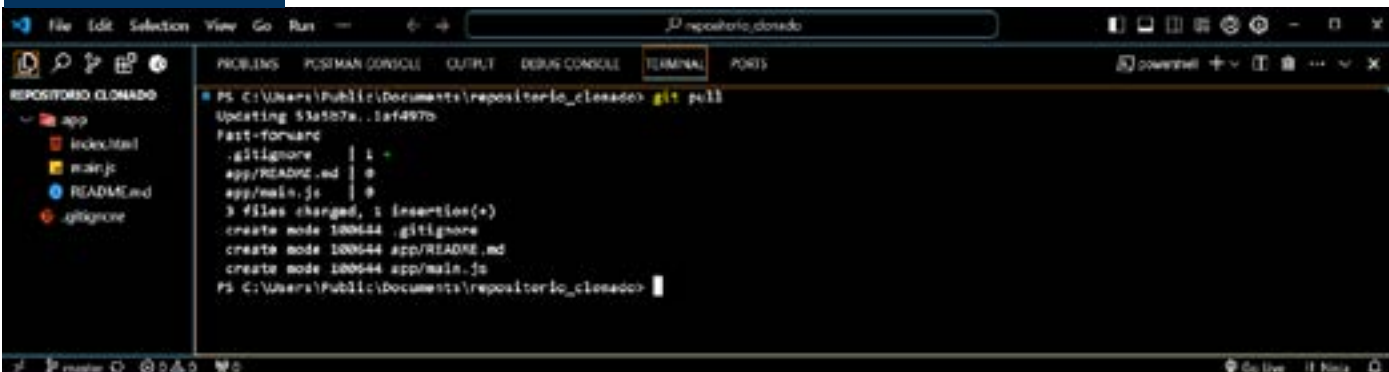
Capturas y ejemplos prácticos que demuestren la diferencia entre **git pull** y **git fetch**

1. GIT FETCH:



```
PS C:\Users\Public\Documents\repositorio_clonado> git fetch
From https://github.com/CursoeInformatorio/ejemplo.git
 53a5b7a..1af497b master -> origin/master
PS C:\Users\Public\Documents\repositorio_clonado>
```

2. GIT PULL:



```
PS C:\Users\Public\Documents\repositorio_clonado> git pull
Updating 53a5b7a..1af497b
Fast-forward
 .gitignore      | 1 +
 app/README.md   | 1
 app/main.js     | 1
 3 files changed, 3 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 app/README.md
 create mode 100644 app/main.js
PS C:\Users\Public\Documents\repositorio_clonado>
```

3. GIT CLONE:

- a). **git clone** se utiliza para clonar un repositorio remoto en una nueva carpeta local.
- b). Por ejemplo - Clona un repositorio remoto en la ubicación actual: **git clone URL_del_repositorio**

Capturas de pantalla que muestran el proceso de clonación de un repositorio remoto.

3. GIT CLONE:

4. GIT BRANCH:

5. GIT CHECKOUT:

- a). **git checkout** se utiliza para cambiar entre ramas o puntos en la historia del proyecto, crear una nueva rama, entre otras funcionalidades.
- b). Por ejemplo - Si estamos trabajando en la rama "rama_con_nuevas_funcionalidades" y deseamos cambiar a la rama "master", utilizaremos: **git checkout master**

Para volver a la otra rama: **git checkout rama_con_nuevas_funcionalidades**

Demostración del proceso de cambio de ramas con **git checkout**



5. GIT CHECKOUT:

```

PS C:\Users\Public\Documents\repositorio> git branch
* master
* rama_con_nuevas_funcionalidades
PS C:\Users\Public\Documents\repositorio> git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 1 commit.
(usa 'git push' to publish your local commits)
PS C:\Users\Public\Documents\repositorio> git branch
* master
* rama_con_nuevas_funcionalidades
PS C:\Users\Public\Documents\repositorio> git checkout rama_con_nuevas_funcionalidades
Switched to branch 'rama_con_nuevas_funcionalidades'
PS C:\Users\Public\Documents\repositorio> git branch
* master
* rama_con_nuevas_funcionalidades
PS C:\Users\Public\Documents\repositorio>

```

5. GIT CHECKOUT:

- c). Por ejemplo - Para crear una nueva rama llamada "nueva_rama" y automáticamente cambia a ella:

git checkout -b nueva_rama

Demostración del proceso de creación y cambio de ramas con **git checkout -b**

```

PS C:\Users\Public\Documents\repositorio> git branch
* master
PS C:\Users\Public\Documents\repositorio> git checkout -b nueva_rama
Switched to a new branch 'nueva_rama'
PS C:\Users\Public\Documents\repositorio> git branch
* master
* nueva_rama
PS C:\Users\Public\Documents\repositorio>

```

6. GIT MERGE:

- a). **git merge** se utiliza para combinar los cambios de una rama en otra. Este proceso puede resultar en la creación de un nuevo commit que incorpora los cambios de ambas ramas.

- b). Por ejemplo - Supongamos que estamos en la rama "rama_con_nuevas_funcionalidades" y

deseamos incorporar los cambios en la rama "master".

- 1). Asegurémonos de estar en la rama de destino, en este caso, la rama "master":

git checkout master

6. GIT MERGE:

```

PS C:\Users\Public\Documents\repositorio> git branch
master
* rama_con_nuevas_funcionalidades
PS C:\Users\Public\Documents\repositorio> git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)
PS C:\Users\Public\Documents\repositorio> git branch
* master
  rama_con_nuevas_funcionalidades
PS C:\Users\Public\Documents\repositorio>
  
```

6. GIT MERGE:

2). Luego, ejecutamos el comando **git merge** seguido del nombre de la rama que deseamos fusionar, en este caso, "rama_con_nuevas_funcionalidades":

git merge rama_con_nuevas_funcionalidades

```

PS C:\Users\Public\Documents\repositorio> git merge rama_con_nuevas_funcionalidades
Updating 468dcd1..1af497b
Fast-forward
 1 file changed, 1 insertion(+)
 create mode 100644 .gitignore
 create mode 100644 app/main.js
PS C:\Users\Public\Documents\repositorio>
  
```

podemos realizar un commit si lo creemos necesario previo al push

```

PS C:\Users\Public\Documents\repositorio> git push origin master
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/CursosInformatario/ejemplo.git
 55ab7a..1af497b master -> master
PS C:\Users\Public\Documents\repositorio>
  
```


3). Git intentará automáticamente fusionar los cambios. Si hay conflictos (cambios conflictivos en ambas ramas), Git solicitara resolver esos conflictos manualmente antes de completar la fusión.

Despliegue en GitHub Pages

1. Despliegue en GitHub Pages de un HTML Básico:

a). Crear un archivo HTML básico.



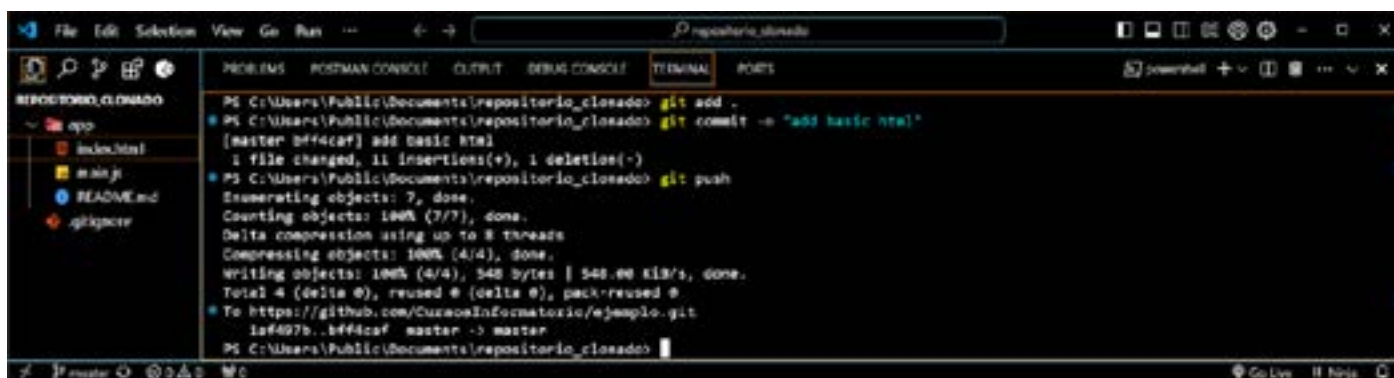
```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Prueba de HTML</title>
7 </head>
8 <body>
9   <h1>Hola mundo</h1>
10 </body>
11 </html>
12

```

b). Inicializar un repositorio Git en ese proyecto.

c). Hacer un commit y enviarlo al repositorio remoto en GitHub.



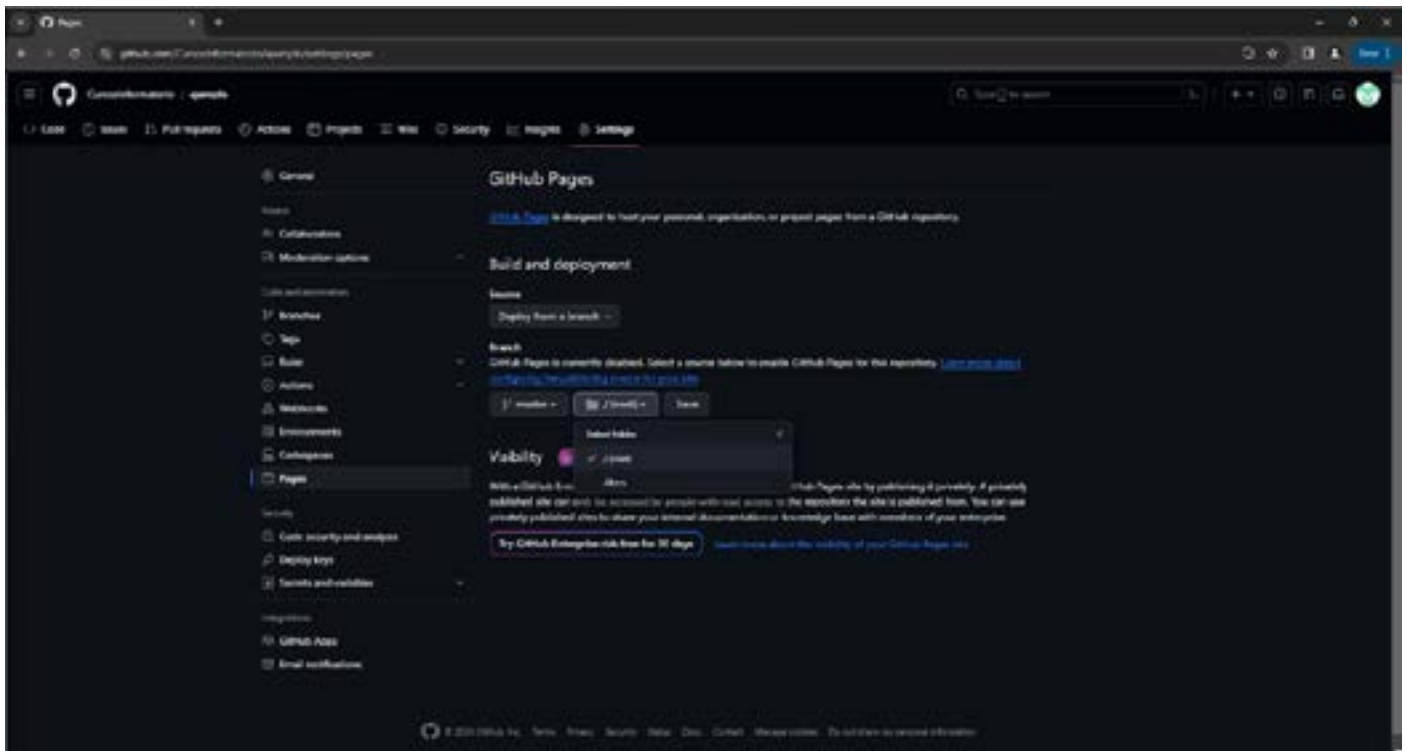
```

PS C:\Users\Public\Documents\repositorio_creado> git add .
PS C:\Users\Public\Documents\repositorio_creado> git commit -m "add basic html"
[master 0ff4c0f] add basic html
1 file changed, 11 insertions(+), 1 deletion(-)
PS C:\Users\Public\Documents\repositorio_creado> git push
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), 548 bytes | 548.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/CuzcosInformatorio/ejemplo.git
 1a497b..0ff4c0f master -> master
PS C:\Users\Public\Documents\repositorio_creado>

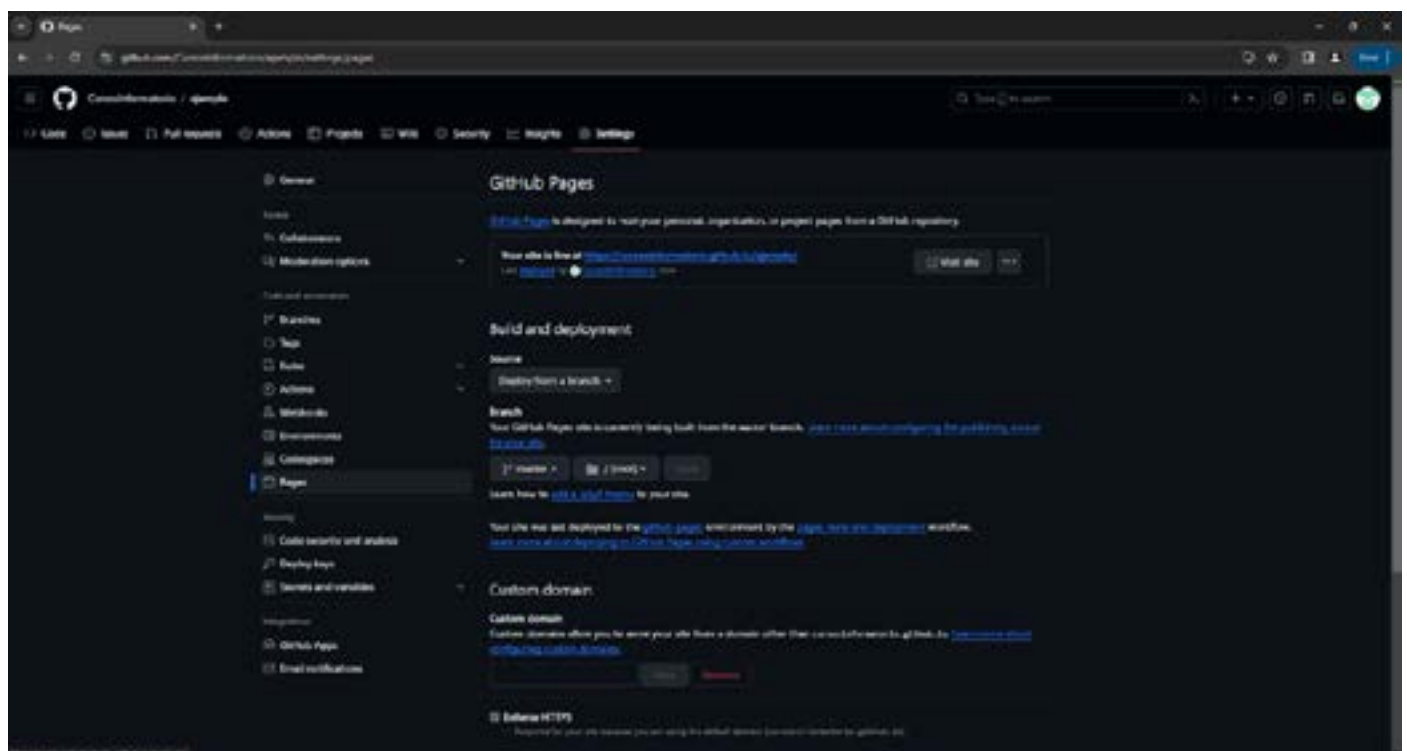
```

d). Ir a la sección de configuración del repositorio en GitHub.

e). En la sección de GitHub Pages, seleccionar la rama y carpeta a desplegar (generalmente master/main y /root).



f). Mostrar cómo el sitio está ahora disponible en <https://tu-usuario.github.io/nombre-repositorio>





1. Explicación y Consideraciones Importantes:

a). GitHub Pages:

Se utiliza para alojar sitios web estáticos, sin procesamiento del lado del servidor, directamente desde repositorios de GitHub.

b). Configuración del Repositorio:

En el video explicamos configuración para la rama y la carpeta de despliegue en GitHub Pages.

En el video mostramos cómo los archivos en esa carpeta serán accesibles públicamente.

c). Actualizaciones Automáticas:

Cada cambio que se envía a la rama seleccionada se refleja automáticamente en el sitio desplegado.

b). Consideraciones Importantes:

Asegurarse de que la estructura de archivos y enlaces en el HTML sea correcta para evitar problemas de carga.

Créditos y atribuciones

* **IMAGEN 1** Imagen de vectorjuice en Freepik