Nombre y Apellido: Roldan Claudio Nicolás
Grupo 3 PI : Trabajo Practico #1: Introducción al IoT
Usuario GitHub: ClaudioNR22
Email: claudio.n.roldan09@gmail.com

Parte 2: Introducción a GitHub:

1. **Creación de una cuenta en GitHub:**
   - **Usuario: ClaudioNR22**
   - **Email: claudio.n.roldan09@gmail.com**

2. **Informe de Aprendizaje**:

   Resumen:

Git es un **sistema de control de versiones (VCS) distribuido**, lo que significa que es una herramienta útil para rastrear fácilmente los cambios en su código, colaborar y compartir. Con Git, puedes realizar un seguimiento de los cambios que realizas en tu proyecto para que siempre tengas un registro de lo que has trabajado y puedas volver fácilmente a una versión anterior si es necesario. También facilita el trabajo con otros: grupos de personas pueden trabajar juntas en el mismo proyecto y fusionar sus cambios en una fuente final.

GitHub es una forma de usar el mismo poder de Git en línea con una interfaz fácil de usar. Se utiliza en todo el mundo del software y más allá para colaborar y mantener el historial de proyectos.

GitHub es la plataforma líder mundial para el desarrollo de software, la colaboración y la seguridad.

Desarrollo de Software:

GitHub Actions: Automatiza tus flujos de trabajo de compilación, pruebas y despliegue con CI/CD (Integración Continua/Entrega Continua) simple y seguro.

GitHub Codespaces: Ofrece un entorno de desarrollo completo en segundos. Codifica, compila, prueba y crea solicitudes de extracción desde cualquier repositorio.

GitHub Mobile: Lleva tus proyectos en el bolsillo para no perderte nada mientras estás en movimiento.

Nombre y Apellido: Roldan Claudio Nicolás
Grupo 3 PI : Trabajo Practico #1: Introducción al IoT
Usuario GitHub: ClaudioNR22
Email: claudio.n.roldan09@gmail.com

Colaboración:

Discussions: Crea espacios para hacer preguntas y tener conversaciones abiertas.

Pull Requests: Permite la comunicación y colaboración en tiempo real sobre cambios de código.

Seguridad:

Code scanning: Nuestra herramienta de análisis de código te ayuda a solucionar problemas en tu código.

Dependabot: Facilita la búsqueda y corrección de dependencias vulnerables en tu cadena de suministro.

Secret scanning: Busca automáticamente patrones de secretos y previene el uso fraudulento de secretos accidentalmente comprometidos.

CI/CD (Integración Continua/Entrega Continua) en **GitHub** es un conjunto de prácticas y herramientas que automatizan el proceso de desarrollo de software desde la escritura del código hasta su implementación en producción. Aquí tienes una descripción más detallada:

Integración Continua (CI):

Automatización de Compilación: Cada vez que se realiza un *commit* al repositorio, se ejecutan automáticamente pruebas de compilación y se generan artefactos (como binarios o paquetes).

Pruebas Automatizadas: Se ejecutan pruebas unitarias, de integración y otras pruebas automáticamente para detectar errores temprano.

Retroalimentación Rápida: Los desarrolladores reciben comentarios inmediatos sobre la calidad del código.

Entrega Continua (CD):

Automatización de Despliegue: Una vez que las pruebas pasan, el código se despliega automáticamente en un entorno de preparación o de producción.

Gestión de Versiones: Se manejan las versiones de la aplicación y se asegura que las actualizaciones sean coherentes y controladas.

Monitoreo y Rollbacks: Se monitorea el despliegue y, si es necesario, se revierte a una versión anterior.

GitHub Actions:

Nombre y Apellido: Roldan Claudio Nicolás
Grupo 3 PI : Trabajo Practico #1: Introducción al IoT
Usuario GitHub: ClaudioNR22
Email: claudio.n.roldan09@gmail.com

Automatización de Flujos de Trabajo: GitHub Actions permite definir flujos de trabajo personalizados en archivos YAML. Puedes configurar acciones para compilar, probar y desplegar tu código automáticamente.

Seguridad y Calidad del Código: Puedes integrar análisis de seguridad, pruebas de calidad y otras verificaciones en tus flujos de trabajo.

En resumen, CI/CD en GitHub te ayuda a acelerar el desarrollo, mejorar la calidad del código y garantizar una entrega confiable de tus aplicaciones.

Descripción del Flujo de GitHub

Repositories (Repositorios):

Un repositorio es el lugar donde se realiza el trabajo del proyecto, considéralo como la carpeta del proyecto. Contiene todos los archivos y el historial de revisiones de su proyecto. Puede trabajar solo dentro de un repositorio o invitar a otros a colaborar con usted en esos archivos.

Cloning (Clonación):

Cuando se crea un repositorio con GitHub, se almacena de forma remota en el archivo. Puede clonar un repositorio para crear una copia local en su computadora y luego usar Git para sincronizar los dos. Esto hace que sea más fácil solucionar problemas, agregar o eliminar archivos e impulsar confirmaciones más grandes. También puedes usar la herramienta de edición que elijas en lugar de la interfaz de usuario de GitHub. La clonación de un repositorio también extrae todos los datos del repositorio que GitHub tiene en ese momento, incluidas todas las versiones de todos los archivos y carpetas del proyecto.

Commiting and Pushing (Comprometerse y Empujar):

Commiting and Pushing: son la forma en que puedes agregar los cambios que realizaste en tu máquina local al repositorio remoto en GitHub. De esa manera, tu instructor y/o compañeros de equipo pueden ver tu último trabajo cuando estés listo para compartirlo. Puede hacer una confirmación cuando haya realizado cambios en su proyecto que desee "punto de control". También puedes añadir un **mensaje de confirmación** útil para

Nombre y Apellido: Roldan Claudio Nicolás
Grupo 3 PI : Trabajo Practico #1: Introducción al IoT
Usuario GitHub: ClaudioNR22
Email: claudio.n.roldan09@gmail.com

recordarte a ti mismo o a tus compañeros de equipo el trabajo que hiciste (por ejemplo, "Se ha añadido un archivo LÉAME con información sobre nuestro proyecto").

Una vez que tengas una confirmación o varias confirmaciones que estés listo para agregar a tu repositorio, puedes usar el comando push para agregar esos cambios a tu repositorio remoto.

Términos de GitHub

Repositories (Repositorio): son donde ocurre el trabajo de tu proyecto. Los repositorios también contienen **archivos README**. Puedes agregar un archivo README a tu repositorio para decirle a otras personas por qué tu proyecto es útil, qué pueden hacer con él y cómo pueden usarlo

Branches (Rama): Puedes usar ramas en GitHub para aislar el trabajo que aún no quieres fusionar en tu proyecto final. Las ramas te permiten desarrollar características, corregir errores o experimentar de forma segura con nuevas ideas en un área contenida de tu repositorio. Normalmente, puedes crear una nueva rama a partir de la rama predeterminada de tu repositorio: main. Esto crea una nueva copia de trabajo de tu repositorio para que experimentes con ella. Una vez que un compañero de equipo haya revisado los nuevos cambios, o esté satisfecho con ellos, puede fusionar los cambios en la rama predeterminada del repositorio

Forks (Tenedores): Una bifurcación es otra forma de copiar un repositorio, pero generalmente se usa cuando desea contribuir al proyecto de otra persona. La bifurcación de un repositorio le permite experimentar libremente con los cambios sin afectar al proyecto original y es muy popular cuando se contribuye a proyectos de software de código abierto.

Pull Request (Solicitud de incorporación de cambios): Al trabajar con ramas, puedes usar una solicitud de incorporación de cambios para informar a otros sobre los cambios que deseas realizar y pedirles sus comentarios. Una vez que se abre una solicitud de extracción, puede discutir y revisar los posibles cambios con los colaboradores y agregar más cambios si es necesario. Puedes agregar personas específicas como revisores de tu solicitud de extracción, lo que demuestra que quieres sus comentarios sobre tus cambios. Una vez que una solicitud de incorporación de cambios está lista para funcionar, se puede fusionar con la rama principal.

Nombre y Apellido: Roldan Claudio Nicolás
Grupo 3 PI : Trabajo Practico #1: Introducción al IoT
Usuario GitHub: ClaudioNR22
Email: claudio.n.roldan09@gmail.com

Issues (Cuestiones/Incidencia): Las incidencias son una forma de realizar un seguimiento de las mejoras, las tareas o los errores de tu trabajo en GitHub. Los problemas son una excelente manera de realizar un seguimiento de todas las tareas en las que desea trabajar para su proyecto y dejar que otros sepan en qué planea trabajar. También puede usar problemas para informar a un proyecto de código abierto favorito sobre un error que encontró o una característica que cree que sería genial agregar. En el caso de proyectos más grandes, puede realizar un seguimiento de muchos problemas en un tablero de proyecto.

You User Profiles (Su perfil de Usuario): Tu perfil cuenta a la gente la historia de tu trabajo a través de los repositorios que te interesan, las contribuciones que has hecho y las conversaciones que has tenido. También puedes darle al mundo una visión única de quién eres con el archivo README de tu perfil.

Using markdown on GitHub (Uso de markdown en GitHub): puedes agregar un estilo divertido a tus propuestas, solicitudes de incorporación de cambios y archivos. "Markdown" es una manera fácil de diseñar tus propuestas, solicitudes de incorporación de cambios y archivos con una sintaxis simple. Esto puede ser útil para organizar su información y facilitar la lectura de los demás. ¡También puedes colocar gifs e imágenes para ayudar a transmitir tu punto de vista! Para obtener más información sobre el uso de la versión de Markdown de GitHub.

Comando mas utilizados:

git init

El comando git init es utilizado para inicializar un nuevo repositorio Git en un directorio vacío o en un directorio existente que aún no esté siendo rastreado por Git. Cuando ejecutas git init, Git crea una estructura de directorios y archivos en la ubicación actual para comenzar a rastrear los cambios en el proyecto.

Pasos básicos para utilizar el comando git init

Abre una terminal o línea de comandos en la ubicación donde deseas crear el repositorio, o bien en el directorio del proyecto existente que deseas convertir en un repositorio Git.

Ejecuta el comando git init.

Una vez que has inicializado el repositorio con git init, puedes empezar a realizar cambios en los archivos de tu proyecto y utilizar otros comandos de Git como git add, git commit, git status, etc., para gestionar y controlar versiones de tu código.

Nombre y Apellido: Roldan Claudio Nicolás
Grupo 3 PI : Trabajo Practico #1: Introducción al IoT
Usuario GitHub: ClaudioNR22
Email: claudio.n.roldan09@gmail.com

Es importante mencionar que git init se usa generalmente solo una vez al principio del proyecto, cuando se desea iniciar el seguimiento de versiones con Git. Si ya existe un repositorio Git (por ejemplo, si clonaste uno existente con git clone), no necesitas usar git init.

git clone

El comando git clone se utiliza para crear una copia exacta de un repositorio Git existente en un servidor remoto y así descargarlo en tu máquina local. Al clonar un repositorio, obtienes una réplica completa del historial de cambios, ramas y archivos del proyecto.

Pasos básicos para utilizar el comando git clone

La sintaxis básica del comando git clone es la siguiente:

git clone [URL del repositorio remoto]

Al ejecutar git clone, Git descargará todo el historial del repositorio remoto y creará un directorio nuevo con el mismo nombre que el repositorio en tu máquina local. Todos los archivos, ramas y commits estarán disponibles para que puedas trabajar en el proyecto localmente.

El comando git clone es una de las principales formas de comenzar a trabajar en un proyecto colaborativo o de obtener una copia de un proyecto de código abierto para contribuir o explorar su código.

git add

git add se utiliza para agregar cambios realizados en archivos específicos o en todos los archivos modificados al área de preparación (staging area) en Git. El área de preparación es una etapa intermedia donde Git registra los cambios que estarán incluidos en el próximo commit.

Pasos básicos para utilizar el comando git add

La sintaxis básica del comando git add es la siguiente:

- **git add** [archivo]. Añade cambios en el archivo específico al área de preparación.
- **git add**. Añade todos los cambios de todos los archivos modificados al área de preparación.

Nombre y Apellido: Roldan Claudio Nicolás
Grupo 3 PI : Trabajo Practico #1: Introducción al IoT
Usuario GitHub: ClaudioNR22
Email: claudio.n.roldan09@gmail.com

Después de ejecutar git add, los cambios en los archivos especificados quedan listos para ser incluidos en el próximo commit. Luego, puedes usar el comando git commit para crear un commit que registre esos cambios en el historial del repositorio.

Es necesario destacar que git add solo prepara los cambios para el commit, pero no los guarda permanentemente en el repositorio. Para que los cambios se almacenen de forma permanente, es necesario realizar un commit con git commit.

git commit

El comando git commit se utiliza para guardar de manera permanente los cambios que han sido previamente agregados al área de preparación (staging area) en Git. Un commit representa un registro inmutable de los cambios realizados en los archivos del proyecto, en un momento específico.

Pasos básicos para utilizar el comando git commit

La sintaxis básica del comando git commit es la siguiente:

git commit -m "mensaje del commit"

Después de ejecutar git commit, los cambios en el área de preparación se guardarán permanentemente en el historial del repositorio. Cada commit tiene un identificador único y se almacena en una secuencia cronológica, lo que permite rastrear la evolución del proyecto a lo largo del tiempo.

Es importante realizar commits frecuentes y con mensajes descriptivos, ya que esto facilita el seguimiento de los cambios, la colaboración con otros desarrolladores y la resolución de problemas en el código.

Además, ten en cuenta que el comando git commit solo guarda los cambios en tu repositorio local. Para sincronizar los commits con un repositorio remoto, es necesario usar el comando git push, que sube los commits locales al repositorio remoto.

git status

El comando git status es una herramienta útil en Git, ya que muestra el estado actual del repositorio en tu máquina local. Proporciona información sobre los cambios realizados en los archivos, las ramas actuales y otra información relevante sobre el estado del repositorio.

Nombre y Apellido: Roldan Claudio Nicolás
Grupo 3 PI : Trabajo Practico #1: Introducción al IoT
Usuario GitHub: ClaudioNR22
Email: claudio.n.roldan09@gmail.com

Pasos básicos para utilizar el comando git status

La sintaxis del comando git status es bastante simple:

git status

Cuando ejecutas git status, recibirás información sobre los siguientes aspectos:

- **Rama actual**. Git te mostrará la rama en la que te encuentras trabajando actualmente. Por ejemplo, si estás en la rama principal, verás algo como «En la rama main».
- **Cambios sin preparar** (Untracked files / Changes not staged for commit). Git te mostrará una lista de los archivos que han sido modificados pero que aún no han sido agregados al área de preparación (staging area) utilizando git add.
- **Cambios preparados** (Changes to be committed). Git te mostrará una lista de los archivos que han sido agregados al área de preparación y que están listos para ser incluidos en el próximo commit.
- **Ramas y commits**. Puedes ver el estado de las ramas, los commits pendientes de enviar al repositorio remoto y otra información sobre el historial de cambios.

El comando git status es una herramienta valiosa para tener una visión general de cómo se encuentra tu repositorio local en términos de cambios y versiones. Te ayuda a entender qué archivos han sido modificados y qué cambios están listos para ser commiteados. Utilizar git status, frecuentemente, te permitirá mantener un flujo de trabajo organizado y mantener un control sobre los cambios realizados en tu proyecto.

git log

El comando git log se utiliza para mostrar el historial completo de commits realizados en un repositorio Git. Al ejecutar git log, obtendrás una lista cronológica de todos los commits realizados, lo que te permite ver quién hizo los cambios, cuándo se realizaron y los mensajes asociados a cada commit.

Pasos básicos para utilizar el comando git log

La sintaxis básica del comando git log es la siguiente:

git log

Cuando ejecutas git log, verás una lista de commits que incluye la siguiente información:

- SHA: es el identificador único de cada commit (Secure Hash Algorithm). Se utiliza para referirse a un commit específico.

Nombre y Apellido: Roldan Claudio Nicolás
Grupo 3 PI : Trabajo Practico #1: Introducción al IoT
Usuario GitHub: ClaudioNR22
Email: claudio.n.roldan09@gmail.com

- Autor: nombre y dirección de correo electrónico del autor del commit.
- Fecha: fecha y hora en la que se realizó el commit.
- Mensaje: el mensaje descriptivo que se agregó al realizar el commit, que describe los cambios realizados.

git log es una herramienta muy útil para revisar la historia del repositorio, rastrear los cambios realizados y comprender la evolución del proyecto. Además, te proporciona la información necesaria para identificar y resolver problemas en el código y facilita la colaboración entre desarrolladores al mantener un registro detallado de las contribuciones.

git pull

El comando git pull se utiliza para descargar y fusionar los cambios desde un repositorio remoto a tu repositorio local. Es una combinación de dos comandos en Git: git fetch, que descarga los cambios desde el repositorio remoto sin aplicarlos y git merge, que fusiona los cambios descargados en tu rama actual.
Pasos básicos para utilizar el comando git pull

La sintaxis básica del comando git pull es la siguiente:

git pull [repositorio-remoto] [rama]

Es importante tener en cuenta que antes de ejecutar git pull, es recomendable asegurarse de que no tienes cambios locales sin commitear que puedan entrar en conflicto con los cambios del repositorio remoto. Si tienes cambios sin commitear, es mejor commitearlos o guardarlos temporalmente con git stash antes de hacer el pull.

git pull es una operación común en el flujo de trabajo de Git, ya que permite mantener tu repositorio local actualizado con los cambios del repositorio remoto y facilita la colaboración con otros miembros del equipo.

git push

El comando git push se utiliza para enviar los commits locales de tu repositorio a un repositorio remoto. Después de realizar cambios y commits en tu repositorio local, puedes usar git push para sincronizar tus cambios con el repositorio remoto y compartirlos con otros colaboradores.

Pasos básicos para utilizar el comando git push

La sintaxis básica del comando git push es la siguiente:

Nombre y Apellido: Roldan Claudio Nicolás
Grupo 3 PI : Trabajo Practico #1: Introducción al IoT
Usuario GitHub: ClaudioNR22
Email: claudio.n.roldan09@gmail.com

git push [repositorio-remoto] [rama]

Es importante tener en cuenta que antes de realizar un git push, asegúrate de haber realizado commits locales con mensajes descriptivos y de haber actualizado tu rama local con git pull si hay cambios nuevos en el repositorio remoto. Esto evita conflictos y garantiza que los cambios se envíen de manera adecuada.

git push es una operación esencial para colaborar con otros desarrolladores en un proyecto Git y para mantener el repositorio remoto actualizado con los cambios realizados en tu repositorio local.

git branch

El comando git branch se utiliza en Git para mostrar una lista de todas las ramas presentes en el repositorio. Una rama en Git es simplemente una línea independiente de desarrollo que permite trabajar en diferentes características o versiones del proyecto sin afectar la rama principal (generalmente llamada «master» o «main»).

Pasos básicos para utilizar el comando git branch

La sintaxis básica del comando git branch es la siguiente:

git branch

Cuando ejecutas git branch, obtendrás una lista de todas las ramas existentes en el repositorio, y la rama activa (en la que te encuentras trabajando) estará resaltada con un asterisco (*).

git checkout

El comando git checkout en Git se utiliza para cambiar de rama o desplazarse entre distintos commits en tu repositorio. También se puede ejecutar para crear nuevas ramas. La acción que realiza git checkout depende del argumento que le proporciones.

Pasos básicos para utilizar el comando git checkout

La sintaxis básica del comando git checkout es la siguiente:

git checkout [rama o commit]git checkout b3a2d97

Esto te llevará al commit con el identificador único (SHA) b3a2d97. En este estado, te situarás en un modo «desconectado» (detached HEAD), lo que significa que no estás en una rama específica. Puedes revisar el código en ese commit, pero ten en cuenta que

Nombre y Apellido: Roldan Claudio Nicolás
Grupo 3 PI : Trabajo Practico #1: Introducción al IoT
Usuario GitHub: ClaudioNR22
Email: claudio.n.roldan09@gmail.com

cualquier nuevo commit creado en esta situación no estará en una rama y podría perderse fácilmente.

git checkout -b new-feature

Este comando creará una nueva rama llamada new-feature y cambiará a esa rama. A partir de este punto, todos los cambios y commits que realices se aplicarán a la nueva rama.

Es importante mencionar que a partir de Git 2.23, el comando git switch se introdujo como una alternativa más segura para cambiar entre ramas, mientras que git checkout se recomienda para casos de «desconexión» o para casos más avanzados como deshacer cambios.

git merge

El comando git merge se utiliza en Git para combinar los cambios de una rama con otra. Al fusionar dos ramas con git merge, los cambios realizados en una rama se incorporan a otra rama.

Pasos básicos para utilizar el comando git merge

La sintaxis básica del comando git merge es la siguiente:

git merge [rama]

Es necesario tener en cuenta que antes de realizar un git merge, es una buena práctica asegurarse de que tus cambios estén commiteados en la rama actual. Además, es recomendable realizar un git pull para obtener los últimos cambios del repositorio remoto antes de la fusión, especialmente si hay otros colaboradores que están trabajando en la misma rama.

git merge es una herramienta poderosa para combinar el trabajo realizado en diferentes ramas y es fundamental para el desarrollo colaborativo en Git, permitiendo a los equipos trabajar en paralelo en características y luego integrarlas en la rama principal cuando estén listas.

git remote

El comando git remote se utiliza para ver y gestionar los repositorios remotos asociados con tu repositorio local en Git. Un repositorio remoto es una copia del repositorio que se

Nombre y Apellido: Roldan Claudio Nicolás
Grupo 3 PI : Trabajo Practico #1: Introducción al IoT
Usuario GitHub: ClaudioNR22
Email: claudio.n.roldan09@gmail.com

encuentra en un servidor externo o en la nube, y es utilizado para colaborar con otros desarrolladores y sincronizar cambios entre diferentes máquinas.

Pasos básicos para utilizar el comando git remote

La sintaxis básica del comando git remote es:

git remote [subcomando]

Donde [subcomando] puede ser:

- **add:** agrega un nuevo repositorio remoto al repositorio local.
- **remove o rm:** elimina un repositorio remoto previamente asociado.
- **rename:** cambia el nombre de un repositorio remoto.
- **show:** muestra información detallada sobre los repositorios remotos configurados.
- **set-url:** configura o modifica la URL del repositorio remoto.

Recuerda que los repositorios remotos permiten la colaboración y el intercambio de código entre diferentes miembros del equipo o entre diferentes máquinas. Puedes utilizar **git remote** para gestionar tus repositorios remotos y mantener tu flujo de trabajo de manera organizada y eficiente.

git diff

El comando git diff se utiliza en Git para mostrar las diferencias entre el estado actual del directorio de trabajo y el último commit registrado en el repositorio. Es una herramienta muy útil para revisar los cambios realizados en los archivos antes de agregarlos al área de preparación (staging area) o antes de hacer un commit.

Pasos básicos para utilizar el comando git diff

La sintaxis básica es:

git diff

Cuando ejecutas git diff, Git muestra una lista de las diferencias entre los archivos en el directorio de trabajo (es decir, los cambios no agregados) y los archivos en el último commit.

Además, puedes utilizar git diff con otros argumentos para comparar diferentes estados en el repositorio, como por ejemplo:

Comparar cambios entre el directorio de trabajo y el área de preparación (staging area):

Nombre y Apellido: Roldan Claudio Nicolás
Grupo 3 PI : Trabajo Practico #1: Introducción al IoT
Usuario GitHub: ClaudioNR22
Email: claudio.n.roldan09@gmail.com

git diff --staged

Comparar cambios entre dos commits específicos:

git diff commit1 commit2

Comparar cambios entre dos ramas:

git diff rama1 rama2

git stash

El comando git stash se utiliza en Git para guardar temporalmente los cambios que no están listos para ser commiteados en una «pila» de cambios temporales, conocida como «stash». Esta función es útil cuando necesitas cambiar de rama o realizar alguna acción que afectaría tus cambios locales, pero no deseas hacer commit de esos cambios aún.

Pasos básicos para utilizar el comando git stash

La sintaxis básica del comando es:

git stash

Al ejecutar git stash, Git guardará todos los cambios que no están commiteados en una pila temporal y dejará tu directorio de trabajo limpio, como si no hubieras realizado cambios. Esto te permite cambiar de rama o realizar otras operaciones sin tener que hacer commit de los cambios en tu rama actual.

Además, puedes utilizar git stash con otros argumentos para realizar acciones adicionales, como las siguientes:

- **git stash list:** muestra una lista de todos los cambios guardados en la pila de stashes.
- **git stash apply:** aplica el cambio más reciente del stash a tu directorio de trabajo sin eliminarlo del stash.
- **git stash pop:** aplica el cambio más reciente del stash a tu directorio de trabajo y elimina el stash.
- **git stash drop:** elimina el cambio más reciente del stash sin aplicarlo al directorio de trabajo.

Es importante mencionar que los stashes son específicos de tu repositorio local y no se comparten automáticamente con otros colaboradores. Puedes utilizarlos para guardar

Nombre y Apellido: Roldan Claudio Nicolás
Grupo 3 PI : Trabajo Practico #1: Introducción al IoT
Usuario GitHub: ClaudioNR22
Email: claudio.n.roldan09@gmail.com

temporalmente modificaciones mientras trabajas en una característica o para evitar conflictos cuando cambias de rama.

Recuerda que cuando estés listo para retomar los cambios guardados en un stash, puedes usar **git stash apply o git stash pop** para restaurar esos cambios en tu directorio de trabajo.