# Manual de Git y GitHub

## Temas selectos de estadística


Alejandra Mariela López Núñez
270611

# Índice

*¿Qué es Git?* ..... 2
*¿Qué es GitHub?* ..... 2
**¿Cómo funcionan Git y GitHub juntos?** ..... 2
**Proyectos en Git** ..... 3
**Instalar Git:** ..... 3
**Crear una cuenta en GitHub*:*** ..... 4
***Subir documentos a GitHub*** ..... 7
**Comandos primarios** ..... 8
***Comandos avanzados*** ..... 11
**Indicadores** ..... 13
**Bibliografías** ..... 16

## *¿Qué es Git?*

Es un sistema de control de versiones que realiza un seguimiento de los cambios en los archivos. Git es especialmente útil cuando un grupo de personas y tú estáis haciendo cambios en los mismos archivos al mismo tiempo.

Para hacerlo en un flujo de trabajo basado en Git, se hace lo siguiente:

- Crear una rama a partir de la copia principal de archivos en los que tú (y tus colaboradores) estáis trabajando.
- Realizar modificaciones en los archivos de forma independiente y segura en tu propia rama personal.
- Dejar que Git fusione mediante combinación y de forma inteligente los cambios específicos en la copia principal de archivos, de modo que los cambios no afecten a las actualizaciones de otras personas.
- Dejar que Git realice un seguimiento de tus cambios y los de otras personas, por lo que todos siguen trabajando en la versión más actualizada del proyecto.

## *¿Qué es GitHub?*

Es una plataforma en la web que utiliza Git donde puedes almacenar, compartir y trabajar junto con otros usuarios para escribir código.

Almacenar tu código en un "repositorio" en GitHub te permite:

- Presentar o compartir el trabajo.
- Seguir y administrar los cambios en el código a lo largo del tiempo.
- Dejar que otros usuarios revisen el código y realicen sugerencias para mejorarlo.
- Colaborar en un proyecto compartido, sin preocuparse de que los cambios afectarán al trabajo de los colaboradores antes de que esté listo para integrarlos.
- El trabajo colaborativo, una de las características fundamentales de GitHub, es posible gracias al software de código abierto Git, en el que se basa GitHub.

## ¿Cómo funcionan Git y GitHub juntos?

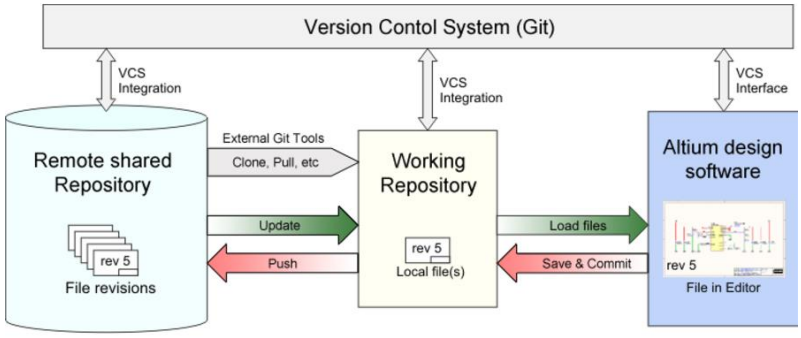
Al cargar archivos en GitHub, los almacenas en un "repositorio de Git". Esto significa que al realizar cambios (o "compromisos") en los archivos de GitHub, Git se iniciará automáticamente para realizar un seguimiento y administrar los cambios.

## *Proyectos en Git*

Los proyectos de la vida real generalmente tienen múltiples desarrolladores trabajando en paralelo. Así que necesitan un sistema de control de versiones como Git para asegurarse de que no hay conflictos de código entre ellos.

Además, los requerimientos en este tipo de proyectos cambian constantemente. Así que un sistema de control de versiones permite a los desarrolladores revertir y regresar a una versión anterior de su código.

El sistema de ramas en Git permite a los desarrolladores trabajar individualmente en una tarea (Por ejemplo: una rama -> una tarea O una Rama -> un desarrollador). Básicamente, se puede pensar en Git como una aplicación de software pequeña que controla tu código base, si eres un desarrollador.




Muestra cómo funciona GitHub

## Instalar Git:

1. Descarga desde git-scm.com.


```
sudo apt update
```

2. Instalación de herramientas de Git.

a) Actualización de paquetes

b) Instala Git y GitHub con apt-get.

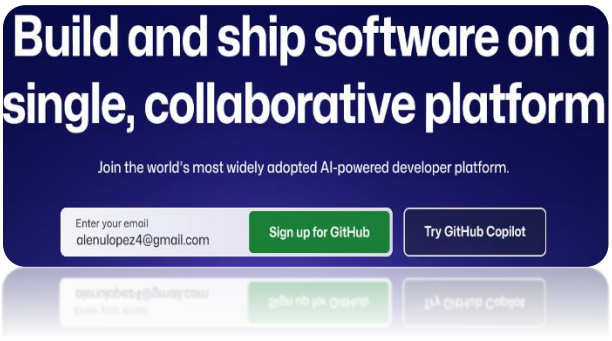
```
sudo apt-get install git
```

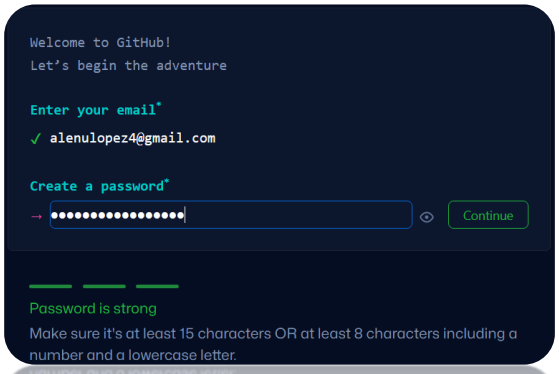
c) Verifica que Git se instaló correctamente.

```
git --version
```

## Crear una cuenta en GitHub*:*

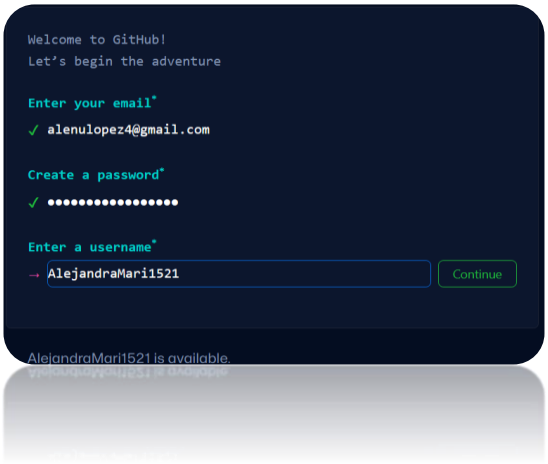
- ✓ Ve a github.com y regístrate.
- ✓ Configura una clave SSH para autenticación segura, colocando tu correo electrónico:




3.Crea una contraseña.

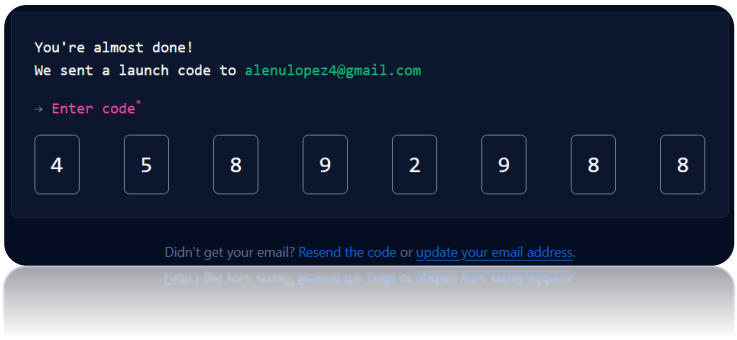
4.Construye tu nombre de usuario.

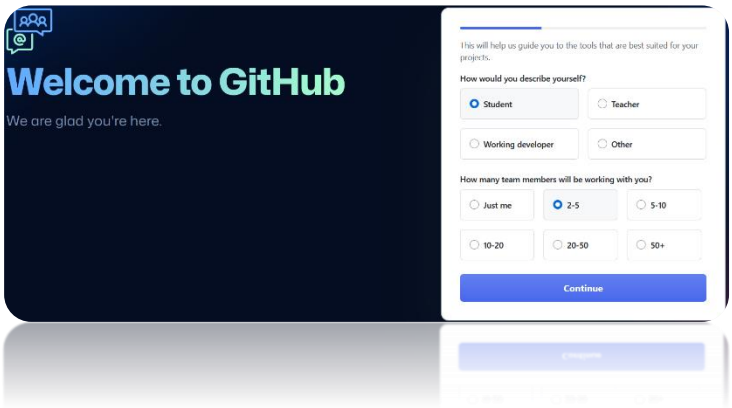




5. Verifica tu identidad.

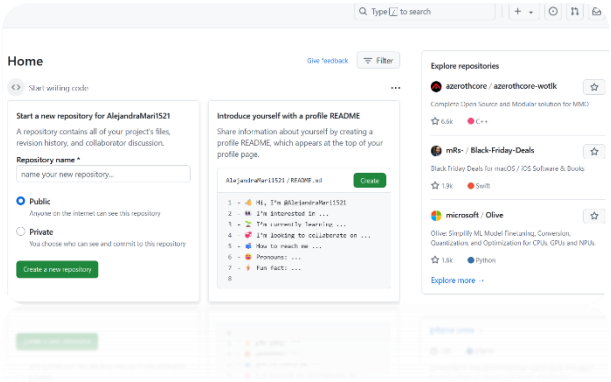
6. Registra el código que llegó a tu correo electrónico.



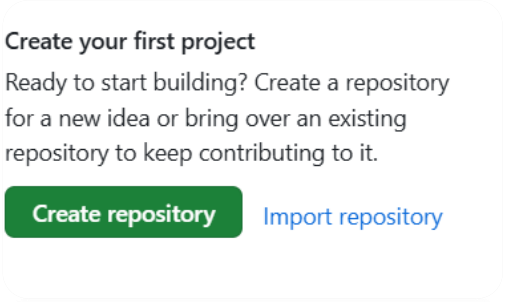




7. Registra tus datos.

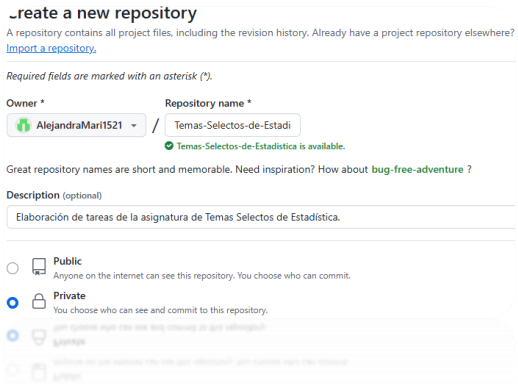
8. Coloca como quieres que tu perfil sea visible.




9. Da clic en crear repositorio.




10. Nombra tu repositorio y llena los datos.

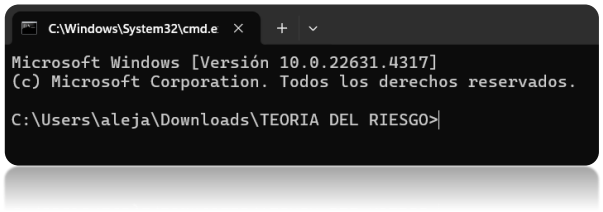



11. Copia el link.

## *Subir documentos a GitHub*

1. Una vez que tienes la dirección del repositorio, necesitas utilizar tu terminal. Usa el siguiente comando en tu terminal. Este comando crea un archivo copia de un repositorio existente en el equipo local.

```
git clone [DIRECCION HTTPS
```




2. Ahora, se debe de abrir una carpeta en archivos y la terminal.

```
git status
```

3. Colocar " git status" para revisar los archivos que has modificado. Muestra el estado actual del repositorio. Este comando muestra el estado del repositorio y los cambios que se tienen actualmente.

4. Luego "git add" seguido del nombre del archivo o trabajo que se subirá entre comillas y con la terminación del tipo de archivo que es. Este comando prepara los cambios para el siguiente commit. Añade la archivo o directorio especificado en el índice.

```
git add [NOMBRE DE ARCHIVO
```

```
git commit -m
```

5. Después el "git commit -m" seguido del nombre del archivo o proyecto que aparecerá en el repositorio. Este comando crea una nueva commit con los cambios que has realizado en tu repositorio local.

6. Ejecuta los siguientes comandos con tu información para establecer un nombre de usuario y un correo electrónico predeterminados para cuando vayas a salvar tu trabajo.

```
git config --global user.name "MV Thanoshan"
git config --global user.email "example@mail.com"
```

```
git push
```

7.Ahora podemos poner nuestro trabajo en GitHub. Ponemos "git push" para lograrlo. Este comando envía los cambios locales al control remoto depósito.

## *Comandos primarios*

- Elimina archivo totalmente

Comando que elimina el archivo tanto del repositorio de Git como del sistema de archivos local:

“git rm archivo.txt

git commit -m "Eliminar archivo.txt"

git push”.

- Elimina archivo parcialmente

Elimina el archivo del índice de Git (repositorio), pero lo mantiene en tu sistema local. Es útil cuando quieres dejar de rastrear un archivo sin borrarlo de tu computadora:

“git rm --cached archivo.txt

git commit -m "Dejar de rastrear archivo.txt"

git push”.

- Eliminar múltiples archivos con una extensión específica

Para eliminar todos los archivos con una extensión específica, como .ext, usa el comodín *.ext:

“git rm --cached *.ext

git commit -m "Eliminar archivos con extensión .ext del repositorio"

git push”.

Elimina todos los archivos con esa extensión del índice, pero los mantiene localmente.

- Renombrar archivos

Usar el comando del sistema operativo mv para renombrar archivos:

“mv nombre_archivo_antiguo nombre_archivo_nuevo

git add nombre_archivo_nuevo

git commit -m "Renombrar archivo: nombre_archivo_antiguo a nombre_archivo_nuevo"

git push”.

- Inicializar un repositorio de Git

Este comando crea un archivo nuevo repositorio Git en el directorio actual: “git init”.

- ✓ Cambios de extracción

Actualiza el repositorio local con los cambios desde el remoto depósito. Extrae los cambios del repositorio remoto y los fusiona con los cambios locales: “git pull”.

- ✓ Listar ramas

Muestra la rama actual en la que se encuentra y la resalta con un asterisco: “git branch”.

- ✓ Nueva rama

Se utiliza para elaborar una nueva rama: “git branch <nombre de la rama>”.

- ✓ Renombrar la rama

Se usa para cambiar el nombre de la rama: “git branch -m <nombre viejo> <nombre nuevo>”.

- ✓ Cambiar de rama

Este comando cambia a la rama especificada. Actualiza el directorio de trabajo para que coincida con el contenido de la rama especificada: “git checkout <nombre de la rama>”.

- ✓ Fusionar rama

Este comando fusiona la rama especificada en la rama actual: “git merge <nombre de la rama>”.

- ✓ Listar todos los commits

Este comando muestra una lista de todos las commits en el repositorio. Muestra el autor, la fecha y el mensaje de confirmación de cada commit que tiene el repositorio: “git log”.

- ✓ Listar repositorios remotos

Este comando lista todos los repositorios remotos asociados con el repositorio local. Muestra la URL de cada repositorio remoto: “git remote -v”.

### ✓ Comprobar diferencia

Este comando muestra las diferencias entre el directorio de trabajo y el área de preparación o el repositorio: “git diff <archivo>.

### ✓ Cambios

Este comando descarga los cambios realizados en el repositorio remoto y actualiza su repositorio local, pero no no fusiona los cambios con su rama local: “git fetch”.

### ✓ Restablecer cambios

Esto elimina el archivo especificado del área de preparación, deshaciendo efectivamente cualquier cambio hecho al archivo desde el ultimo commit. No borra los cambios realizados en un archivo: “git reset <archivo>”.

### ✓ Revertir cambios

Crea un nuevo commit que deshace los cambios realizados en el commit especificado: “git revert <commit>”.

## *Comandos avanzados*

### ✓ Almacenar

Este comando le permite guardar sus cambios sin confirmarlos, lo que puede ser útil cuando necesite cambiar de rama o trabajar en una tarea diferente: “git stash”.

### ✓ Seleccionar

Este comando le permite aplicar selectivamente los cambios realizados en un commit específico a su rama actual, que puede ser útil cuando se necesita incorporar los cambios realizados en otra rama sin fusionar toda la rama: “git cherry-pick <commit>”.

### ✓ Bisecar

Este comando te permite realizar una búsqueda binaria en el historial de commits para encontrar el que introdujo un error: “git bisect”.

### ✓ Buscar culpable

Este comando le permite ver quién modificó por última vez cada línea de un archivo y cuándo lo hizo, lo que puede ser útil cuando necesita averiguar quién introdujo un cambio específico: “git blame”.

### ✓ Registro

Este comando te permite ver un registro de todos los cambios realizados en las referencias Git, lo que puede ser útil cuando necesites recuperar commits perdidos: “git reflog”.

### ✓ Árbol de trabajo

Este comando te permite trabajar en múltiples ramas al mismo tiempo en directorios de trabajo separados: “git worktree”.

### ✓ Filtrar rama

Este comando te permite reescribir la historia de Git aplicando filtros a las ramas, lo que puede ser útil cuando necesitas eliminar datos sensibles de tu repositorio Git: “git filter-branch”.

### ✓ Fusionar

Este comando te permite fusionar cambios de una rama en otra rama como un único commit, lo que puede ser útil cuando quieres mantener un historial de limpio: “git merge - -squash”.

### ✓ Submódulo

Este comando te permite incluir un repositorio Git dentro de otro repositorio Git: “git submodule”.

### ✓ Submódulo foreach

Este comando le permite ejecutar un comando Git en cada submódulo, lo que puede ser útil cuando necesite actualizar múltiples submódulos a la vez: “git submodule foreach”

### ✓ Rebase 1

Este comando te permite reescribir interactivamente la historia de Git reordenando, editando o eliminando commits, lo que puede ser útil cuando necesitas limpiar tu historial de commits: “git rebase -i”.

### ✓ Rebase 2

Este comando te permite aplicar los cambios realizados en una rama en otra rama, lo que puede ser útil cuando necesita actualizar una rama de características con los cambios realizados en la rama maestra: “git rebase”.

## *Indicadores*

Los indicadores (u opciones) en los comandos Git modifican el comportamiento del comando o proporcionan funcionalidad adicional. Suelen empezar con un guión simple (-) para las opciones cortas, o guiones dobles (--) para las opciones largas. El uso de flags te permite personalizar los comandos de acuerdo a sus necesidades, haciendo su flujo de trabajo más flujo de trabajo.

Banderas útiles:

- **-m**

Le permite proporcionar un mensaje de commits directamente en la línea de comandos, evitando la necesidad de abrir un editor: git commit -m “Nombre del commit”.

- **-u**

Usado con git add. Ejecuta los cambios en los archivos rastreados ignorando los archivos sin seguimiento.

- **–amend**

Modifica el último commit añadiendo nuevos cambios o actualizando el mensaje del commit: “git commit  -- amend”.

- **–oneline**

Muestrar cada commit en una línea, haciendo el registro más fácil de leer: “git log -- oneline”.

- **–graph**

Visualiza el historial de commits como un gráfico, ayudándote a entender la estructura de ramas: “git log --graph”.

- **-b**

Crea una nueva rama y cambia a ella en un solo comando: “git checkout -b <nombre de la rama”.

- **–no-ff**

Fuerza un commit de combinación incluso si la fusión se podría realizar con un avance rápido: “git merge --no-ff<nombre de la rama>”.

- **–rebase**

Reaplica los commits locales sobre de los cambios obtenidos, creando un historial más limpio: “git pull --rebase”.

- **–force**

Fuerza un push al repositorio remoto, sobreescribiendo los cambios: “git push  --force”.

- **–hard**

Restablece el directorio de trabajo y el área de preparación a un commit específico, descartando todos los cambios:“git reset --hard <commit>

- **–no-commit**

Se usa con git revert para aplicar los cambios sin confirmarlos inmediatamente:  “git revert <commit_hash> --no-commit”.

- **–no-edit**

Impide que Git abra el editor de mensajes del commit al revertir un commit: “git revert <commit_hash>  --no-edit --soft”.

- **–soft**

Para mover el puntero HEAD a un commit específico sin cambiar el área de preparación o el directorio de trabajo. Este mantiene los cambios preparados para un futuro commit: “git reset --soft <commit_hash>

- **–hard**

Restablece el puntero HEAD, el área de preparación y el directorio al commit especificado. Este comando borra todos los cambios: “git reset --hard <commit_hash>”.

- **–mixed**

Mueve el puntero HEAD y actualiza el área de preparación para que coincida con el commit especificado, pero deja el directorio de trabajo de trabajo sin cambios. Esto es útil para desestablecer archivos: “git reset --mixed <commit_hash>”.

- **Push**

Guarda los cambios locales en un stash y borra el directorio de trabajo directorio de trabajo. Más tarde puede aplicar estos cambios: “git stash push”.

- **Stash apply stash@{index}**

Este comando aplica los cambios de un stash sin eliminarlo. Sustituye {index} por el número de stash : “git  stash apply stash@{2}”.

- **List**

Enumera todos los cambios almacenados, lo que le da una visión general de lo que ha guardado: “git stash list”.

- **Pop**

Aplica los cambios guardados en el último stash y elimina esa entrada de la lista:“git stash pop”.

- **Drop**

Elimina una entrada específica del stash, permitiéndote limpiar

la lista de stash: “git stash drop stash@{0}”.

## *Bibliografías*


- FreeCodeCamp. (n.d.). Guía para principiantes de Git y GitHub. FreeCodeCamp. Recuperado de https://www.freecodecamp.org/espanol/news/guia-para-principiantes-de-git-y-github/

- Documentación de GitHub: GitHub. (n.d.). Hello World. Recuperado de https://docs.github.com/es/get-started/start-your-journey/hello-world

- JS Mastery. (n.d.). Git & GitHub Handbook [Imagen]. Recuperado de https://jsmastery.pro