

SISTEMA DE CONTROL DE VERSIONES

Contenido

Sistemas de control de versiones VCS	2
VCS Centralizados CVCS	3
VCS Distribuidos DVCS	4
Git	5
Fundamentos de Git	5
Git tiene integridad	6
Git generalmente solo añade información	7
Los Tres Estados	7
La Línea de Comandos	8
Github	10
Instalación de Git	12
git config	13
Laboratorio: Probemos Git	13
git init	14
git status	14
git add	15
git commit	15
git log	16
Untracked files	17
git revert	18
git remote / git fetch / git push / git pull	18
Qué es una rama?	18
git branch	19
git checkout	19
Editor Visual Studio Code	20
Carguemos lo realizado a nuestro repositorio remoto de github	21
Conociendo a github	24
< > Code	24
Settings (acá encuentras la opción para invitar a colaboradores)	24
Issues	24
Projects	25

ANEXOS

27

Sistemas de control de versiones VCS

Los sistemas de control de versiones o VCS registran los cambios realizados en uno o varios archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante o regresar a versiones anteriores de tus archivos, regresar a una versión anterior del proyecto completo, comparar cambios a lo largo del tiempo, ver quién modificó por última vez algo que pueda estar causando problemas, ver quién introdujo un problema y cuándo, recuperar archivos borrados y mucho más

VCS Centralizados CVCS

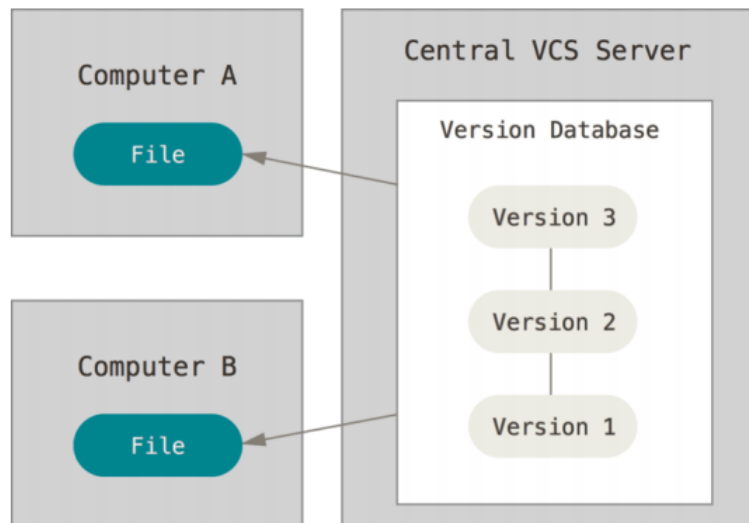


Figure 2. Control de versiones centralizado.

Tomado de <https://git-scm.com/book/es/v2>

Utilizado por equipo de desarrolladores que necesitan colaborar en otros sistemas, donde en único servidor contiene todos los archivos versionados y varios clientes descargan los archivos desde ese lugar central. Este ha sido el estándar para el control de versiones por muchos años.

Esta configuración ofrece muchas ventajas, especialmente frente a VCS locales. Por ejemplo, todas las personas saben hasta cierto punto en qué están trabajando los otros colaboradores del proyecto. Los administradores tienen control detallado sobre qué puede hacer cada usuario, y es mucho más fácil administrar un CVCS que tener que lidiar con bases de datos locales en cada cliente.

Sin embargo, esta configuración también tiene serias desventajas. La más obvia es el punto único de fallo que representa el servidor centralizado. Si ese servidor se cae durante una hora, entonces durante esa hora nadie podrá colaborar o guardar cambios en archivos en los que hayan estado trabajando. Si el disco duro en el que se encuentra la base de datos central se corrompe, y no se han realizado copias de seguridad adecuadamente, se perderá toda la información del proyecto, con excepción de las copias instantáneas que las personas tengan en sus máquinas locales. Los VCS locales sufren de este mismo problema: Cuando tienes toda la historia del proyecto en un mismo lugar, te arriesgas a perderlo todo.

VCS Distribuidos DVCS

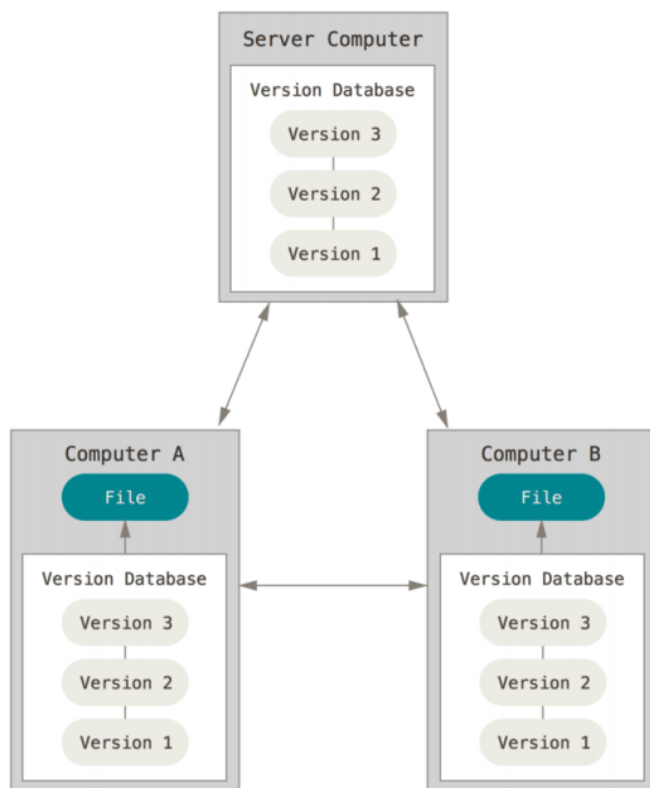


Figure 3. Control de versiones distribuido.

Tomado de <https://git-scm.com/book/es/v2>

En un DVCS (como Git, Mercurial, Bazaar o Darcs), los clientes no solo descargan la última copia instantánea de los archivos, sino que se replica completamente el repositorio. De esta manera, si un servidor deja de funcionar y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios disponibles en los clientes

puede ser copiado al servidor con el fin de restaurarlo. Cada clon es realmente una copia completa de todos los datos.

Git

En 2005 inicia Git como un nuevo sistema DVCS que cumpliera con los siguientes objetivos:

- Velocidad
- Diseño sencillo
- Gran soporte para desarrollo no lineal (miles de ramas paralelas)
- Completamente distribuido
- Capaz de manejar grandes proyectos (como el kernel de Linux) eficientemente (velocidad y tamaño de los datos)

Desde entonces Git ha evolucionado y madurado para ser fácil de usar y conservar sus características iniciales. Es tremendamente rápido, muy eficiente con grandes proyectos y tiene un increíble sistema de ramificación (branching) para desarrollo no lineal

Fundamentos de Git

Git maneja sus datos como un conjunto de copias instantáneas de un sistema de archivos miniatura. Cada vez que confirmas un cambio, o guardas el estado de tu proyecto en Git, él básicamente toma una foto del aspecto de todos tus archivos en ese momento y guarda una referencia a esa copia instantánea. Para ser eficiente, si los archivos no se han modificado Git no almacena el archivo de nuevo, sino un enlace al archivo anterior idéntico que ya tiene almacenado. Git maneja sus datos como una secuencia de copias instantáneas.

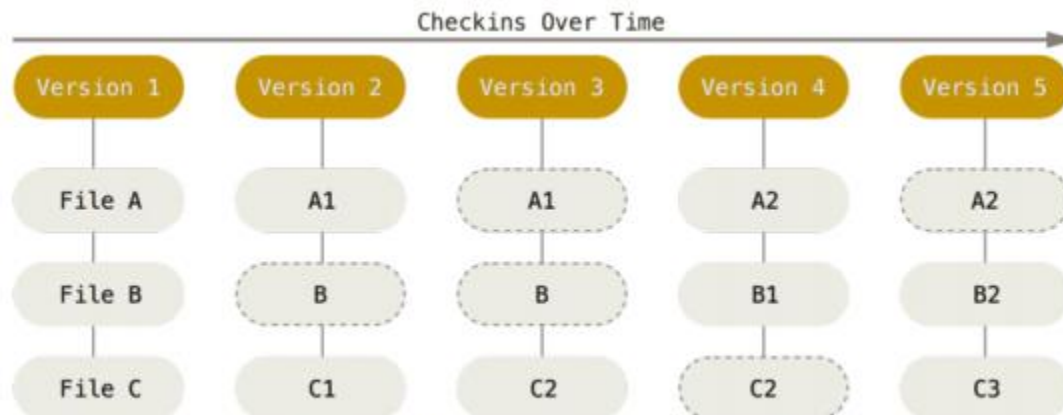


Figure 5. Almacenamiento de datos como instantáneas del proyecto a través del tiempo.

Tomado de <https://git-scm.com/book/es/v2>

La mayoría de las operaciones en Git sólo necesitan archivos y recursos locales para funcionar. Por lo general no se necesita información de ningún otro computador de tu red.

Por ejemplo, para navegar por la historia del proyecto, Git no necesita conectarse al servidor para obtener la historia y mostrarla - simplemente la lee directamente de tu base de datos local.

Git tiene integridad

Todo en Git es verificado mediante una suma de comprobación (checksum en inglés) antes de ser almacenado, y es identificado a partir de ese momento mediante dicha suma. Esto significa que es imposible cambiar los contenidos de cualquier archivo o directorio sin que Git lo sepa. Esta funcionalidad está integrada en Git al más bajo nivel y es parte integral de su filosofía. No puedes perder información durante su transmisión o sufrir corrupción de archivos sin que Git sea capaz de detectarlo.

El mecanismo que usa Git para generar esta suma de comprobación se conoce como hash SHA-1. Se trata de una cadena de 40 caracteres hexadecimales (0-9 y a-f), y se calcula con base en los contenidos del archivo o estructura del directorio en Git.

Un hash SHA-1 se ve de la siguiente forma:

24b9da6552252987aa493b52f8696cd6d3b00373

Verás estos valores hash por todos lados en Git, porque son usados con mucha frecuencia. De hecho, Git guarda todo no por nombre de archivo, sino por el valor hash de sus contenidos

Git generalmente solo añade información

Cuando realizas acciones en Git, casi todas ellas sólo añaden información a la base de datos de Git. Es muy difícil conseguir que el sistema haga algo que no se pueda enmendar, o que de algún modo borre información. Como en cualquier VCS, puedes perder o estropear cambios que no has confirmado todavía. Pero después de confirmar una copia instantánea en Git es muy difícil perderla, especialmente si envías tu base de datos a otro repositorio con regularidad. Esto hace que usar Git sea un placer, porque sabemos que podemos experimentar sin peligro de estropear gravemente las cosas.

Los Tres Estados

Git tiene tres estados principales en los que se pueden encontrar tus archivos:

- confirmado (committed): significa que los datos están almacenados de manera segura en tu base de datos local
- modificado (modified): significa que has modificado el archivo pero todavía no lo has confirmado a tu base de datos
- preparado (staged): significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación

Esto nos lleva a las tres secciones principales de un proyecto de Git:

- El directorio de Git (Git directory): es donde se almacenan los metadatos y la base de datos de objetos para tu proyecto. Es la parte más importante de Git, y es lo que se copia cuando clonas un repositorio desde otra computadora.
- el directorio de trabajo (working directory): es una copia de una versión del proyecto. Estos archivos se sacan de la base de datos comprimida en el directorio de Git, y se colocan en disco para que los puedas usar o modificar
- el área de preparación (staging area): es un archivo, generalmente contenido en tu directorio de Git, que almacena información acerca de lo que va a ir en tu próxima confirmación.

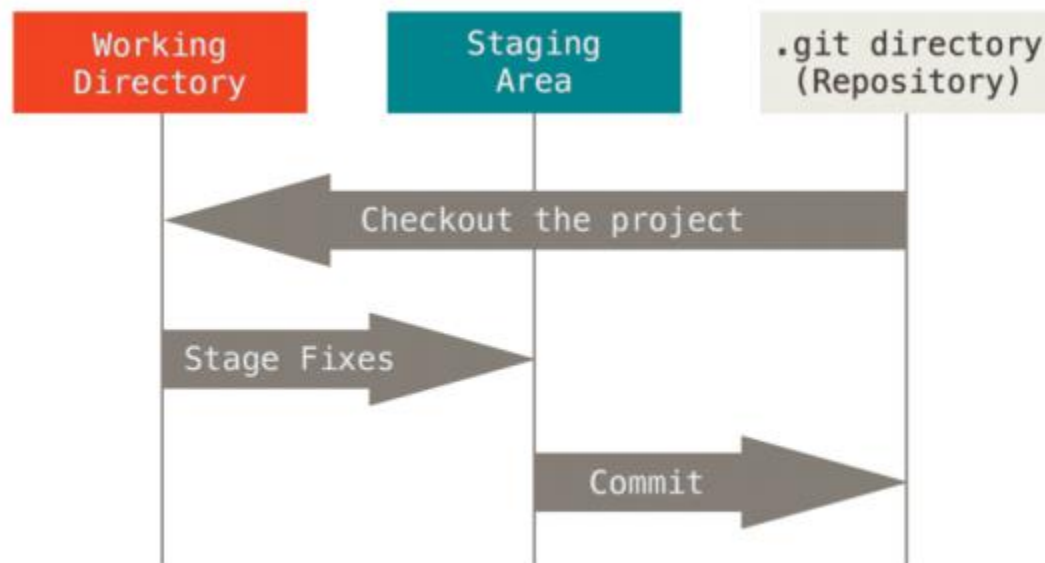


Figure 6. Directorio de trabajo, área de almacenamiento y el directorio Git.

El flujo de trabajo básico en Git es algo así:

1. Modificas una serie de archivos en tu directorio de trabajo.
2. Preparas los archivos, añadiendo a tu área de preparación.
3. Confirmar los cambios, lo que toma los archivos tal y como están en el área de preparación y almacena esa copia instantánea de manera permanente en tu directorio de Git.

Si una versión concreta de un archivo está en el directorio de Git, se considera confirmada (committed). Si ha sufrido cambios desde que se obtuvo del repositorio, pero ha sido añadida al área de preparación, está preparada (staged). Y si ha sufrido cambios desde que se obtuvo del repositorio, pero no se ha preparado, está modificada (modified).

La Línea de Comandos

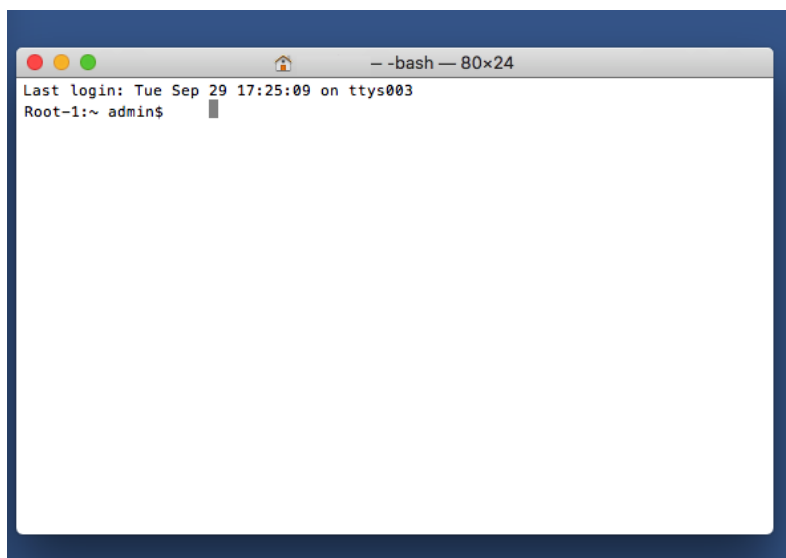
La línea de comandos es el único lugar en donde puedes ejecutar todos los comandos de Git - la mayoría de interfaces gráficas de usuario solo implementan una parte de las características de Git por motivos de simplicidad

Terminal en Mac, o el "Command Prompt" o "Powershell" en Windows



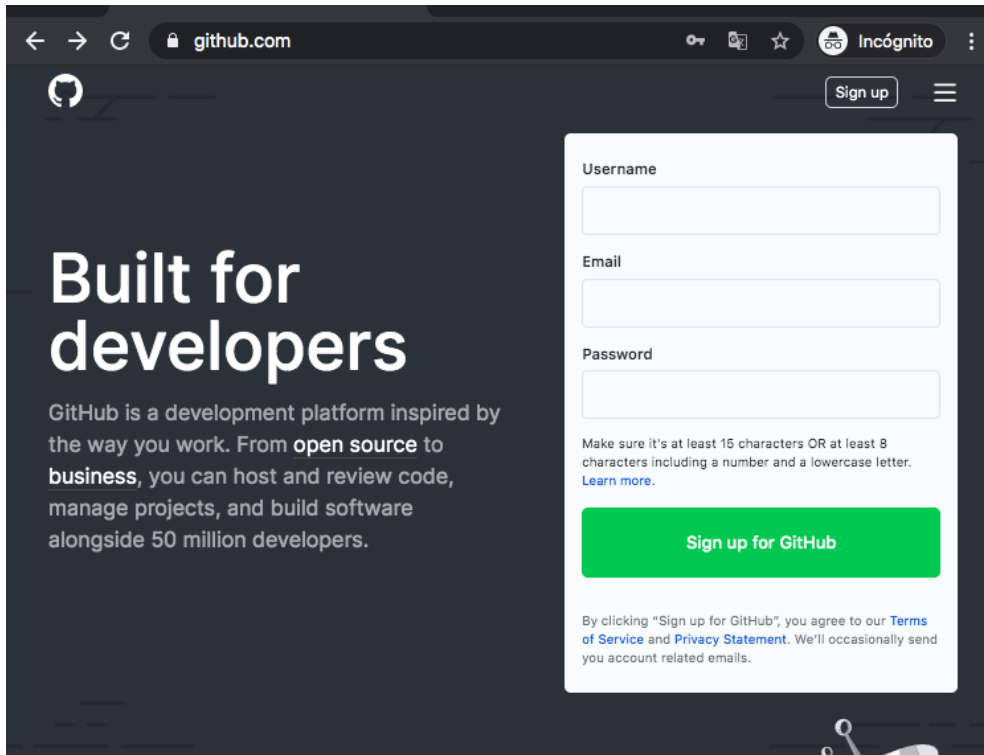
El futuro digital
es de todos

Gobierno
de Colombia
MinTIC



Github

Antes de configurar Git en tu dispositivo, ingresa a <https://github.com/> y regístrate:



Username

Email

Password

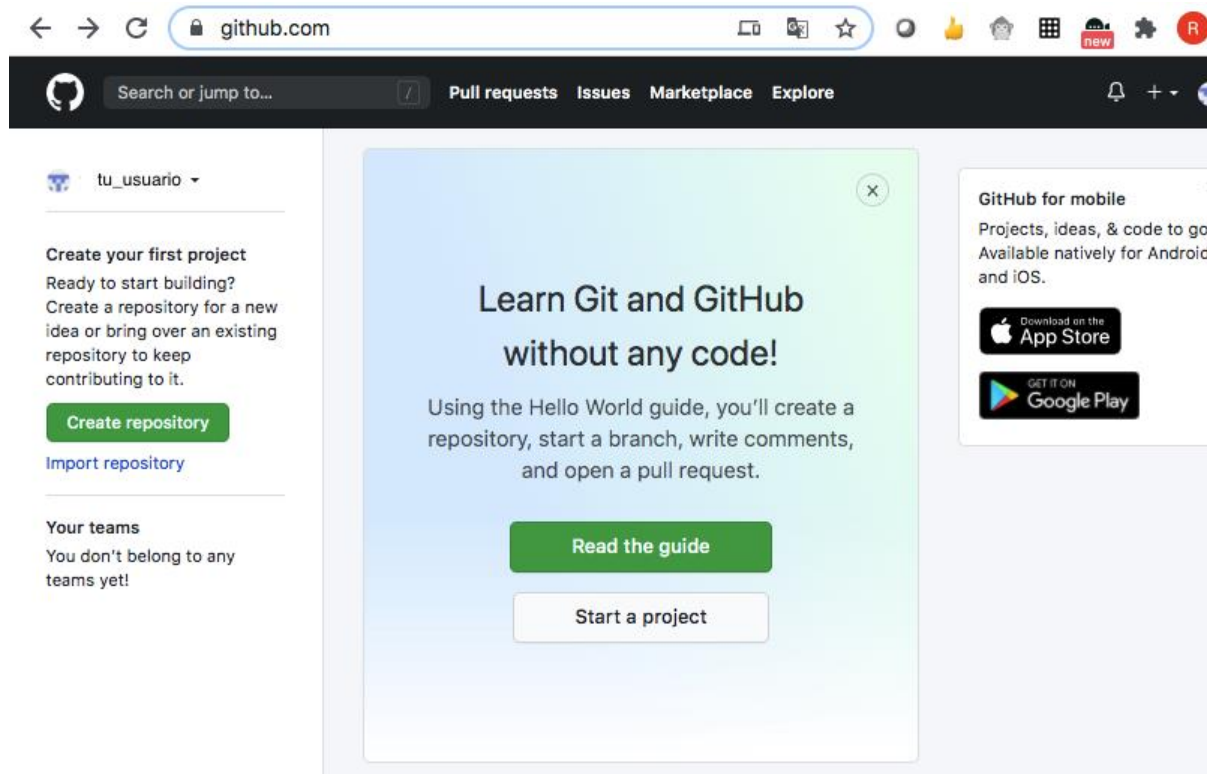
Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more.](#)

Sign up for GitHub

By clicking "Sign up for GitHub", you agree to our [Terms of Service](#) and [Privacy Statement](#). We'll occasionally send you account related emails.

Recuerda muy bien estos 3 datos: El Username es tu nombre de usuario *tu_nombre_de_usuario* ; Email, tu correo electrónico al que tengas acceso fácilmente *tu_email* ; Password: una contraseña con al menos 8 caracteres que incluyan un número y una letra mayúscula.

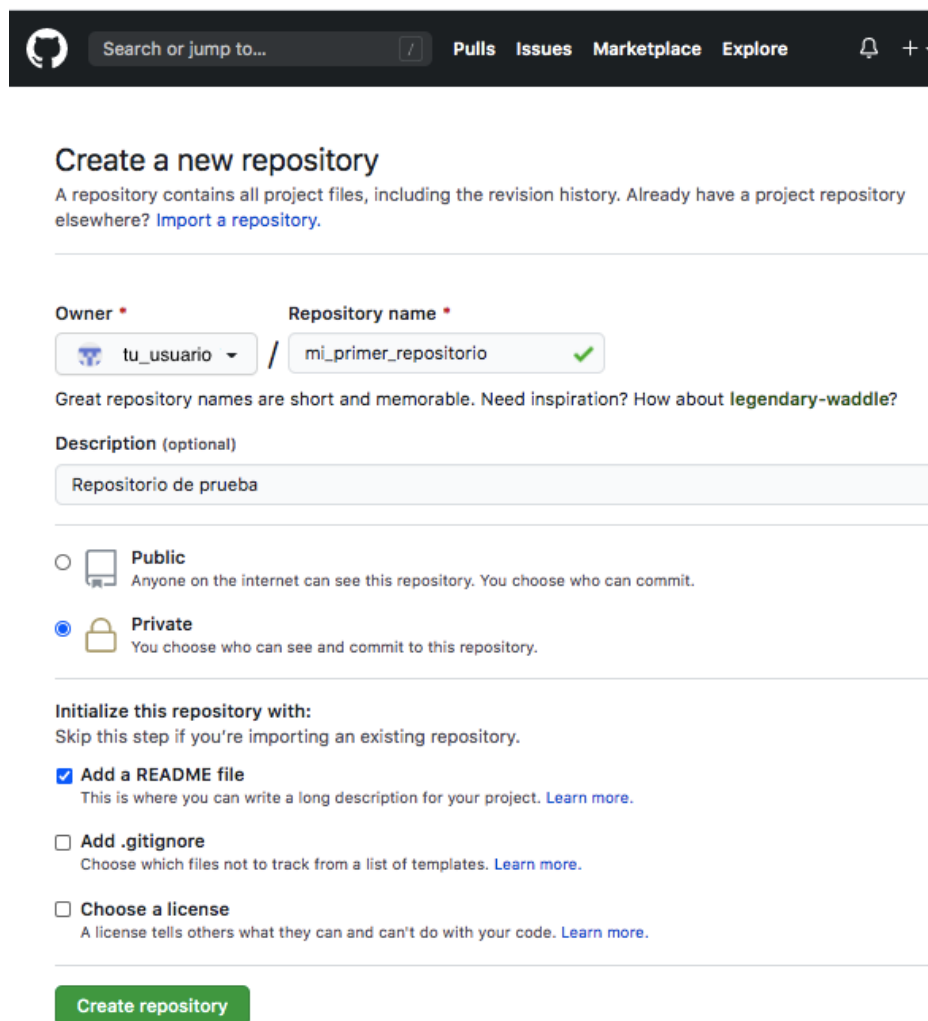
Recuerda leer los Términos del servicio.



GitHub es una plataforma de desarrollo colaborativo de software para alojar proyectos usando el sistema de control de versiones Git. El código se almacena de forma pública, aunque también se puede hacer de forma privada, creando una cuenta de pago. También se pueden obtener repositorios privados (de pago) si se es estudiante.

GitHub no sólo ofrece alojamiento del código si no muchas más posibilidades asociadas a los repos como son, forks, issues, pull requests, diffs, etc. Se verán todos con detalle más adelante.

Da clic en Crear repositorio, aparece la siguiente imagen:



La imagen muestra la interfaz de GitHub para crear un nuevo repositorio. En la parte superior, hay una barra de navegación con el logo de GitHub, un campo de búsqueda "Search or jump to...", y enlaces a "Pulls", "Issues", "Marketplace" y "Explore". Debajo, el título "Create a new repository" está seguido de una descripción: "A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)".

En la sección de configuración, se muestra el "Owner" como "tu_usuario" y el "Repository name" como "mi_primer_repositorio" con una marca de verificación verde. Debajo, se sugiere: "Great repository names are short and memorable. Need inspiration? How about **legendary-waddle**?".

La "Description (optional)" contiene el texto "Repositorio de prueba".

Se muestran dos opciones de visibilidad: "Public" (desactivada) y "Private" (activada). La descripción para "Private" es: "You choose who can see and commit to this repository."

En la sección "Initialize this repository with:", se indica: "Skip this step if you're importing an existing repository.".

Hay tres opciones de inicialización: "Add a README file" (seleccionada con una casilla marcada), "Add .gitignore" (desseleccionada) y "Choose a license" (desseleccionada). Cada opción tiene un enlace "Learn more."

En la parte inferior, hay un botón verde que dice "Create repository".

Diligencia con los datos que prefieras

Instalación de Git

Antes de empezar a utilizar Git, tienes que instalarlo en tu computadora. Incluso si ya está instalado, este es posiblemente un buen momento para actualizarlo a su última versión. Puedes instalarlo como un paquete, a partir de un archivo instalador o bajando el código fuente y compilándolo tú mismo

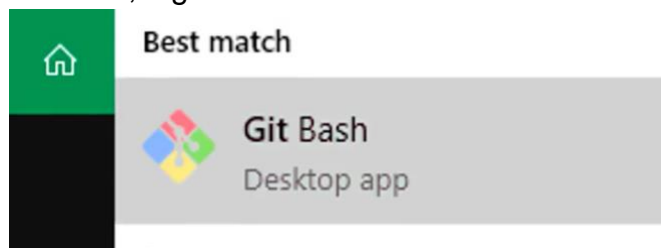
Instalación en Linux: <https://git-scm.com/download/linux>

Instalación en Mac: <https://git-scm.com/download/mac>

Instalación en Windows: <https://git-scm.com/download/win>

git config

Ahora sí, ingresa a la terminal de Git:



Digita en la terminal:

```
$ git config --global user.name tu_nombre_de_usuario  
$ git config --global user.email tu_email
```

Puedes verificar la configuración así:

```
$ git config --list
```

```
usuario-computador$ git config --list  
credential.helper=osxkeychain  
core.excludesfile=/Users/compute/.gitignore_global  
difftool.sourcetree.cmd=opendiff "$LOCAL" "$REMOTE"  
difftool.sourcetree.path=  
mergetool.sourcetree.cmd=/Users/compute/Applications/Sourcetree.app/Contents/Resources/opendiff-w.sh "$LOCAL" "$REMOTE"  
ASE" -merge "$MERGED"  
mergetool.sourcetree.trustexitcode=true  
user.name=tu_nombre_de_usuario  
user.email=tu_email@gmail.com  
commit.template=/Users/compute/.stCommitMsg  
usuario-computador$
```

Laboratorio: Probemos Git

Creemos una carpeta en el escritorio llamada pruebaGit.

Con la terminal nos ubicamos dentro de la carpeta y digitamos

git init

```
$ git init
```

vemos el siguiente mensaje

```
Inicializado repositorio Git vacío en /Users/computer/Desktop/pruebagit/.git/
```

si listamos los archivos ocultos, veremos una carpeta con el nombre .git

Ahora creemos un archivo dentro de la carpeta pruebagit: leeme.txt

En la terminal escribamos

git status

```
$ git status
```

sale lo siguiente

```
En          la          rama          master
No          hay          commits          todavía
Archivos          sin          seguimiento:
(usa "git add <archivo>..." para incluirlo a lo que se será confirmado)
leeme.txt

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

Agregamos leeme.txt al commit

git add

```
$ git add leeme.txt
```

Ahora, si volvemos a escribir

```
$ git status
```

sale:

```
En la rama master
No hay commits todavía
Cambios a ser confirmados:
(usa "git rm --cached <archivo>..." para sacar del área de stage)
nuevo archivo: leeme.txt
```

Primer Commit

Ahora, vamos a hacer el commit

git commit

```
$git commit
```

vemos :

```
# Por favor ingresa el mensaje del commit para tus cambios. Las
# líneas que comiencen con '#' serán ignoradas, y un mensaje
# vacío aborta el commit.
#
# En la rama master
#
```

```
# Confirmación inicial
#
# Cambios a ser confirmados:
# nuevo archivo: leeme.txt
#
~
```

Agregamos un mensaje al inicio así:

```
Commit inicial
# Por favor ingresa el mensaje del commit para tus cambios. Las
# líneas que comiencen con '#' serán ignoradas, y un mensaje
# vacío aborta el commit.
#
# En la rama master
#
# Confirmación inicial
#
# Cambios a ser confirmados:
# nuevo archivo: leeme.txt
#
~
~
```

Ahora revisemos

```
$git status
```

vemos:

```
En la rama master
nada para hacer commit, el árbol de trabajo está limpio
```

Ahora revisemos las versiones

git log


```
$git log
```

vemos lo siguiente:

```
commit 18bc1a77ff64558a37deca3153ad0ed77ead6aa5 (HEAD -> master)
Author:      tu_usuario <tu_correo@gmail.com>
Date:        Thu Oct 1 21:39:59 2020 -0500

    Commit inicial
```

Modificar el archivo leeme.txt y verificar qué sale con git status

Untracked files

Cambios no rastreados o cambios no registrados por git. Para que queden en el stage o listos para ser confirmados, damos git add leeme.txt

Realizamos nuevamente git commit para registrar los cambios realizados

Se recomienda escribir un mensaje claro de los cambios realizados, es posible que requieras volver a un estado previo del archivo y si está bien “documentado” te será fácil encontrar la versión requerida.

Si se quiere recuperar el último trabajo no reportado o el último commit:

```
git commit --amend
```

Deshacer cambios

Para deshacer cambios en el historial de commits

Ejecutando

git revert

```
git revert HEAD
```

Se crea un commit con lo opuesto al último commit

git remote / git fetch / git push / git pull

El comando git pull se emplea para extraer y descargar contenido desde un repositorio remoto y actualizar al instante el repositorio local para reflejar ese contenido. La fusión de cambios remotos de nivel superior en tu repositorio local es una tarea habitual de los flujos de trabajo de colaboración basados en Git. El comando git pull es, en realidad, una combinación de dos comandos, git fetch seguido de git merge. En la primera etapa de la operación git pull ejecutará un git fetch en la rama local a la que apunta HEAD. Una vez descargado el contenido, git pull entrará en un flujo de trabajo de fusión. Se creará una nueva confirmación de fusión y se actualizará HEAD para que apunte a la nueva confirmación.

Qué es una rama?

En cada confirmación de cambios (commit), Git almacena una instantánea de tu trabajo preparado. Dicha instantánea contiene además unos metadatos con el autor y el mensaje explicativo, y uno o varios apuntadores a las confirmaciones (commit) que sean padres directos de esta (un padre en los casos de confirmación normal, y múltiples padres en los casos de estar confirmando una fusión (merge) de dos o más ramas).

Una rama Git es simplemente un apuntador móvil apuntando a una de esas confirmaciones. La rama por defecto de Git es la rama master. Con la primera confirmación de cambios que realicemos, se creará esta rama principal master apuntando a dicha confirmación. En cada confirmación de cambios que realicemos, la rama irá avanzando automáticamente.

Crear una Rama Nueva

¿Qué sucede cuando creas una nueva rama? Bueno..., simplemente se crea un nuevo apuntador para que lo puedas mover libremente. Por ejemplo, supongamos que quieres crear una rama nueva denominada "testing". Para ello, usarás el comando

git branch

```
$ git branch testing
```

Esto creará un nuevo apuntador apuntando a la misma confirmación donde estés actualmente.

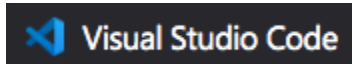
Para trabajar en nuestra nueva rama y no afectar a la rama Master, damos

git checkout

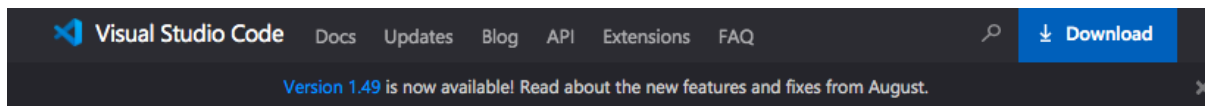
```
git checkout testing
```

Editor Visual Studio Code

Visual studio Code - Descargar el software desde el sitio web oficial
<https://code.visualstudio.com/>

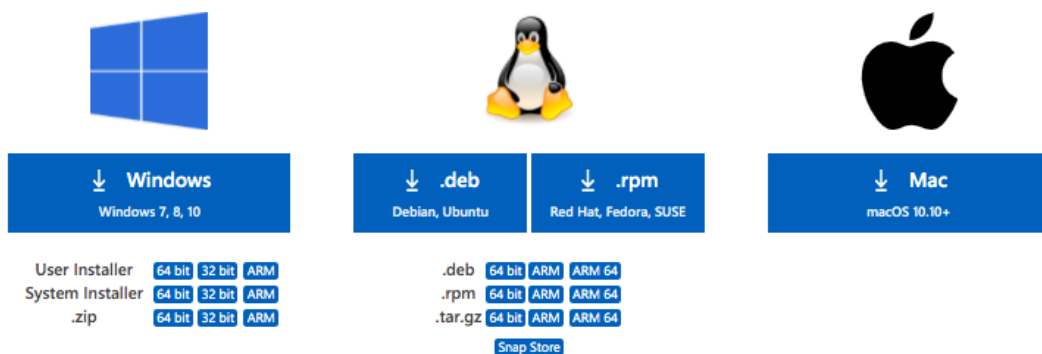


<https://code.visualstudio.com/download>



Download Visual Studio Code

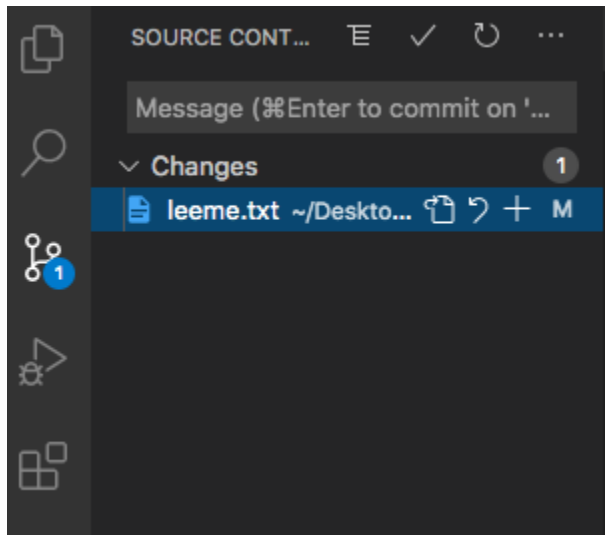
Free and built on open source. Integrated Git, debugging and extensions.



<https://github.com/Hispano/Guia-sobre-Git-Github-y-Metodologia-de-Desarrollo-de-Software-usando-Git-y-Github>

Para editar el archivo que tenemos en el repositorio, damos clic derecho, abrir con Visual Studio Code.

Dentro de este editor se encuentra una Terminal



En la imagen se visualiza el logo de Git; también al final del nombre y ruta del archivo se ve una M, al ubicar el mouse encima, aparece Modified, indicando que el archivo se ha modificado.

Otras letras que aparecen:

- U** untracked, archivo nuevo o modificado que no se ha añadido al repositorio
- A** Added Nuevo archivo que se ha añadido al repositorio
- D** Deleted Archivo borrado
- C** Conflict Hay conflicto en el archivo
- R** Renamed El archivo ha sido renombrado

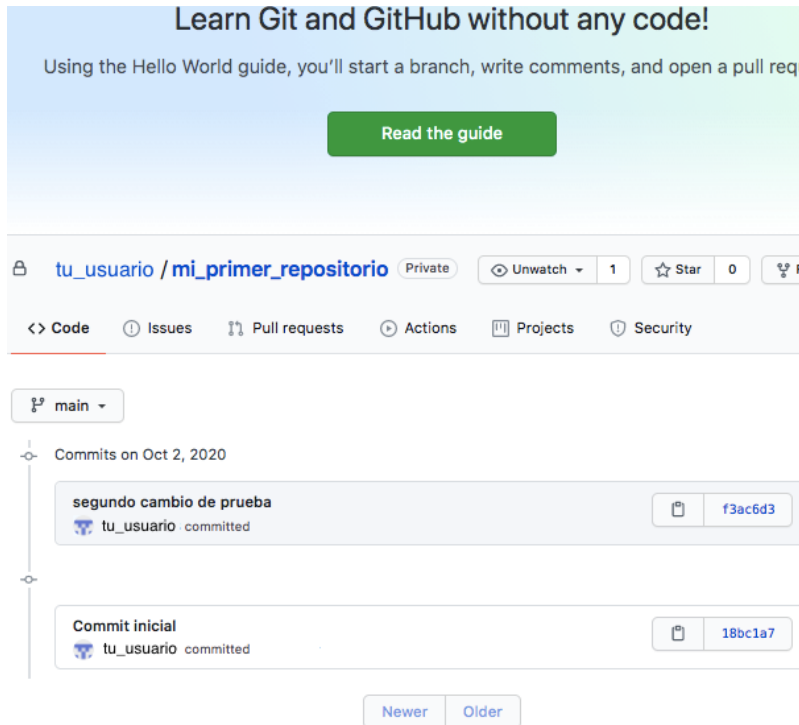
Carguemos lo realizado a nuestro repositorio remoto de github

Abramos una terminal desde VSC y escribamos

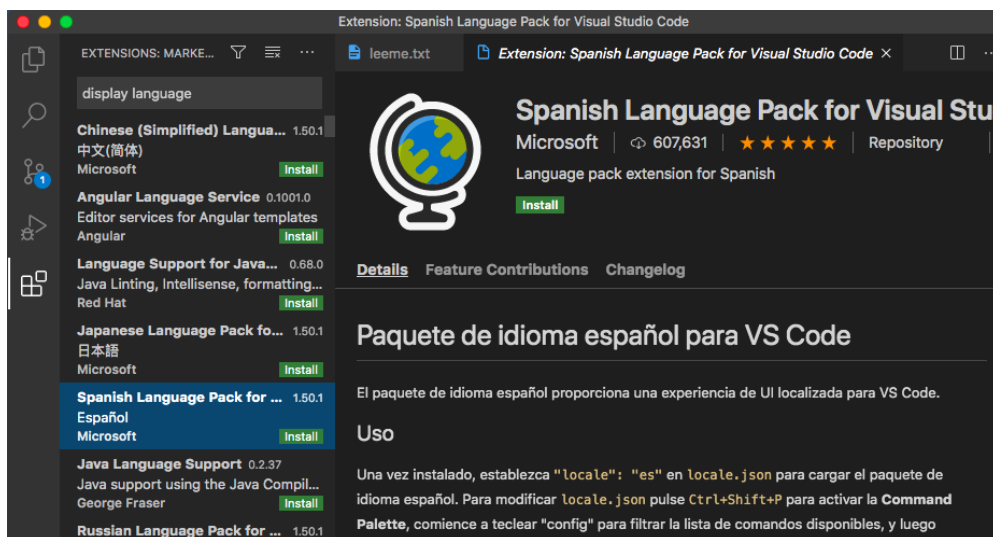
```
git branch -M main
```

git remote

```
git remote add origin https://github.com/tu_usuario/mi_primer_repositorio.git
```



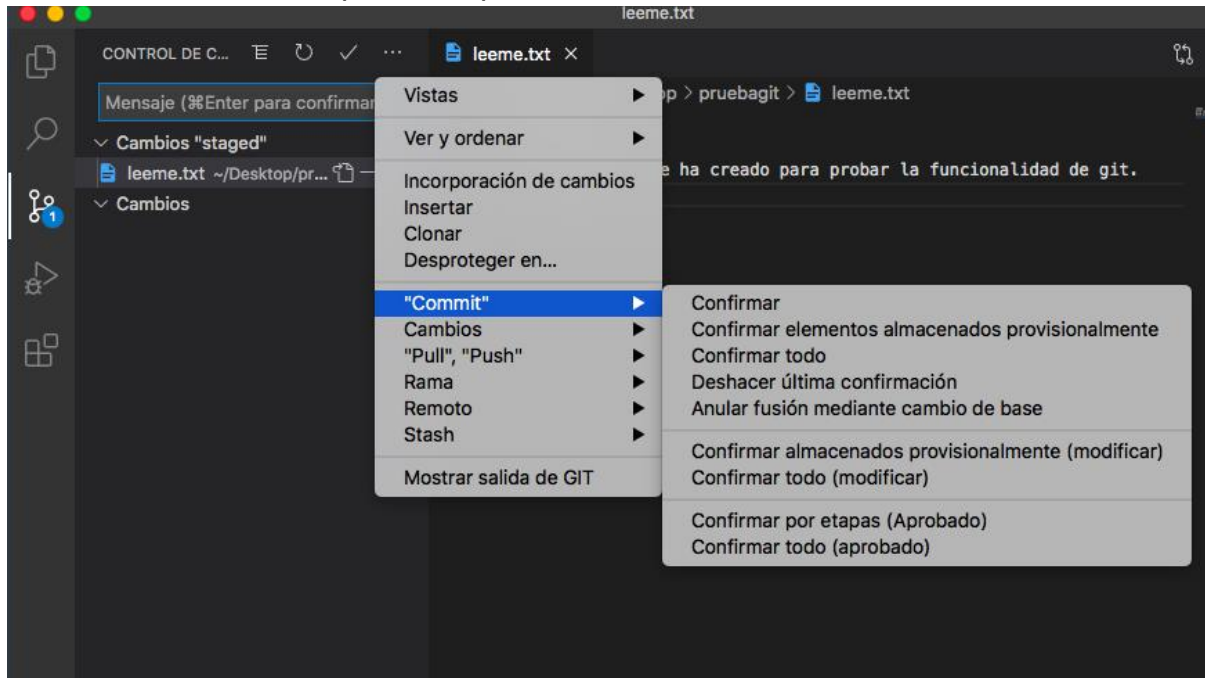
Si deseas configurar tu VSC a español, ve a preferencias -> extensiones y busca display language, e instala Spanish Language Pack



Realicemos un cambio con VSC, utilicemos la terminal para ver los cambios realizados sin preparar con gif diff

Se dejan preparados con git add leeme.txt

Ubica el menú en los 3 puntos superiores



Conociendo a github

Al ingresar a github, es importante que reconozcas el entorno del sitio:

< > Code

Encuentras lo básico revisado en las primeras páginas de este documento

Settings (acá encuentras la opción para invitar a colaboradores)

Acá se configura el acceso al repositorio, ingresando por Manage access, y dando clic en el botón Invite a collaborator

Issues

Permite llevar las tareas a realizar en el repositorio o errores o publicación de errores o temas relacionados en el repositorio on seguimiento por medio de comentarios. Tiene 2 estados: abierto o cerrado.

Por defecto encuentras las siguientes etiquetas o Label:



9 labels	
bug	Something isn't working
documentation	Improvements or additions to documentation
duplicate	This issue or pull request already exists
enhancement	New feature or request
good first issue	Good for newcomers
help wanted	Extra attention is needed
invalid	This doesn't seem right
question	Further information is requested
wontfix	This will not be worked on

Tomado de github

Se usan Milestones para crear colecciones o agrupar issues

Projects

Ayudan a organizar tu trabajo. Los tableros de proyectos o project boards se manejan como tarjetas (para seleccionar issues) dentro de tablas, por ejemplo la plantillas o template: Basic kanban con columnas: To do ; In progress; Done



🏠 proyecto de ejemplo
Updated 1 minute ago

1 To do + ...

📄 Welcome to GitHub Projects ✨ ...

We're so excited that you've decided to create a new project! Now that you're here, let's make sure you know how to get the most out of GitHub Projects.

☒ Create a new project

☒ Give your project a name

☐ Press the (?) key to see available keyboard shortcuts

☐ Add a new column

☐ Drag and drop this card to the new column

☐ Search for and add issues or PRs to your project

☐ Manage automation on columns

☐ [Archive a card](#) or archive all cards in a column

Added by tu_usuario

1 In progress + ...

📄 Cards ...

Cards can be added to your board to track the progress of issues and pull requests. You can also add note cards, like this one!

Added by tu_usuario

1 Done + ...

📄 Automation ...

Automatically move your cards to the right place based on the status and activity of your issues and pull requests.

Added by tu_usuario

🔍 Filter cards

+ Add column



ANEXOS

Encabezados

Para crear un encabezado, agrega uno a seis símbolos # antes del encabezado del texto. La cantidad de # que usas determinará el tamaño del encabezado.

El encabezado más largo
El segundo encabezado más largo
El encabezado más pequeño

The largest heading

The second largest heading

The smallest heading

Estilo de texto

Puedes indicar énfasis con texto en negrita, cursiva o tachado.

Estilo	Sintaxis	Atajo del teclado	Ejemplo	Resultado
Negrita	** ** o _ _	command/control + b	**Este texto está en negrita**	Este texto está en negrita
Cursiva	* * o _ _	command/control + i	*Este texto está en cursiva*	<i>Este texto está en cursiva</i>
Tachado	~ ~ ~		~Este texto está equivocado~	Este texto está equivocado
Cursiva en negrita y anidada	** ** y _ _		**Este texto es _extremadamente_ importante**	Este texto es extremadamente importante
Todo en negrita y cursiva	*** ***		***Todo este texto es importante***	<i>Todo este texto es importante</i>



Cita de texto

Puedes citar texto con un `>`.

Tal como dice Abraham Lincoln:

`> Con perdón de la expresión`

In the words of Abraham Lincoln:

`| Pardon my French`

Enlaces

Puedes crear un enlace en línea al encerrar el texto del enlace entre corchetes `[]`, y luego encerrar la URL entre paréntesis `()`. También puedes usar el atajo del teclado `command + k` para crear un enlace.

Este sitio se construyó usando `[GitHub Pages](https://pages.github.com/)`.

This site was built using [GitHub Pages](https://pages.github.com/).



Código de cita

Puedes indicar un código o un comando dentro de un enunciado con comillas simples. El texto dentro de las comillas simples no será formateado.

```
Usa `git status` para enumerar todos los archivos nuevos o modificados que aún no han
```

```
Use git status to list all new or modified files that haven't yet been committed.
```

Para formatear código o texto en su propio bloque distintivo, usa comillas triples.

Algunos de los comandos de Git básicos son:

```
```\n\ngit status\n\ngit add\n\ngit commit\n\\`
```

Some basic Git commands are:

```
git status\n\ngit add\n\ngit commit
```