



UNIDAD 3

- **Control de versiones del software:**
 - **Git.**
 - **Github.**



Git

Acerca del Control de Versiones

¿Qué es un control de versiones, y por qué es importante?

Un control de versiones es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante.



Git

Sistemas de Control de Versiones Locales

Un método de control de versiones, usado por muchas personas, es copiar los archivos a otro directorio (quizás indicando la fecha y hora en que lo hicieron, si son ingeniosos). Este método es muy común porque es muy sencillo, pero también es tremendamente propenso a errores. Es fácil olvidar en qué directorio te encuentras y guardar accidentalmente en el archivo equivocado o sobrescribir archivos que no querías.

Para afrontar este problema los programadores desarrollaron hace tiempo VCS locales que contenían una simple base de datos, en la que se llevaba el registro de todos los cambios realizados a los archivos.



Git

Sistemas de Control de Versiones Centralizados

Las personas que necesitan colaborar con desarrolladores en otros sistemas. Los sistemas de Control de Versiones Centralizados (CVCS por sus siglas en inglés) fueron desarrollados para solucionar este problema. Tienen un único servidor que contiene todos los archivos versionados y varios clientes que 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. Cuando tienes toda la historia del proyecto en un mismo lugar, te arriesgas a perderlo todo.



Git

Sistemas de Control de Versiones Distribuidos

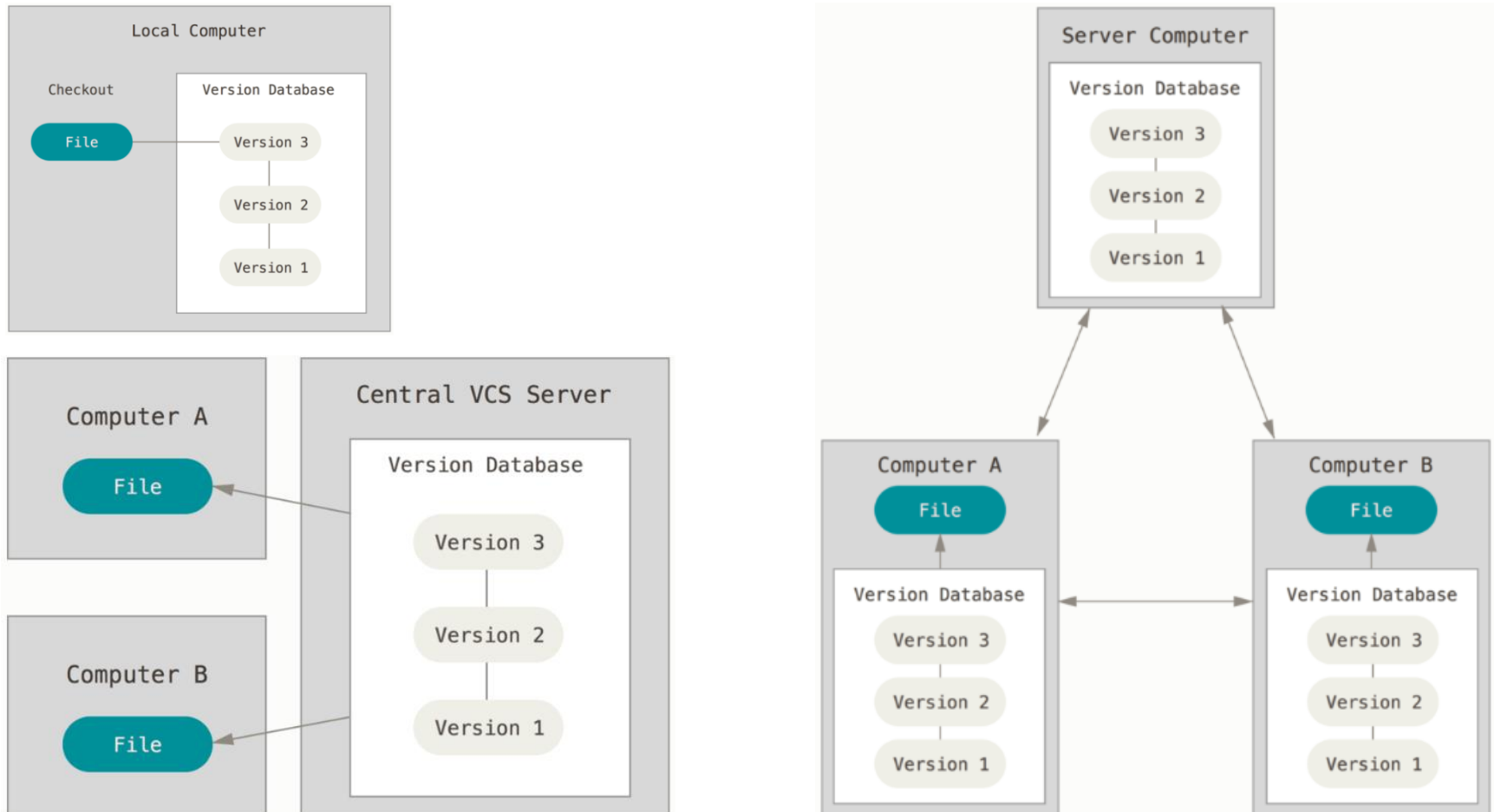
Los sistemas de Control de Versiones Distribuidos (DVCS por sus siglas en inglés) ofrecen soluciones para los problemas que han sido mencionados. En un DVCS (como Git, Mercurial, Bazaar o Darcs), los clientes no solo descargan la última copia instantánea de los archivos, sino que se replica completamente 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.

Además, muchos de estos sistemas se encargan de manejar numerosos repositorios remotos con los cuales pueden trabajar, de tal forma que puedes colaborar simultáneamente con diferentes grupos de personas en distintas maneras dentro del mismo proyecto. Esto permite establecer varios flujos de trabajo que no son posibles en sistemas centralizados, como pueden ser los modelos jerárquicos.



Git

Sistemas de Control de Versiones Locales, Centralizados y Distribuidos.





Git

¿Qué es y para qué sirve Git?

Git es un software de control de versiones diseñado por Linus Torvalds. Fue pensado para que el mantenimiento de versiones de aplicaciones - cuando tienen un gran número de archivos de código fuente - sea eficiente y confiable.

Su propósito es llevar registro de los cambios en archivos de computadora y coordinar el trabajo que varias personas realizan sobre archivos compartidos.



Git

Fundamentos de Git

Git almacena y maneja la información de forma muy diferente a esos otros sistemas.

Copias instantáneas, no diferencias

La principal diferencia entre Git y cualquier otro VCS es la forma en la que manejan sus datos. Conceptualmente, la mayoría de los otros sistemas almacenan la información como una lista de cambios en los archivos. Estos sistemas manejan la información que almacenan como un conjunto de archivos y las modificaciones hechas a cada uno de ellos a través del tiempo.



Git

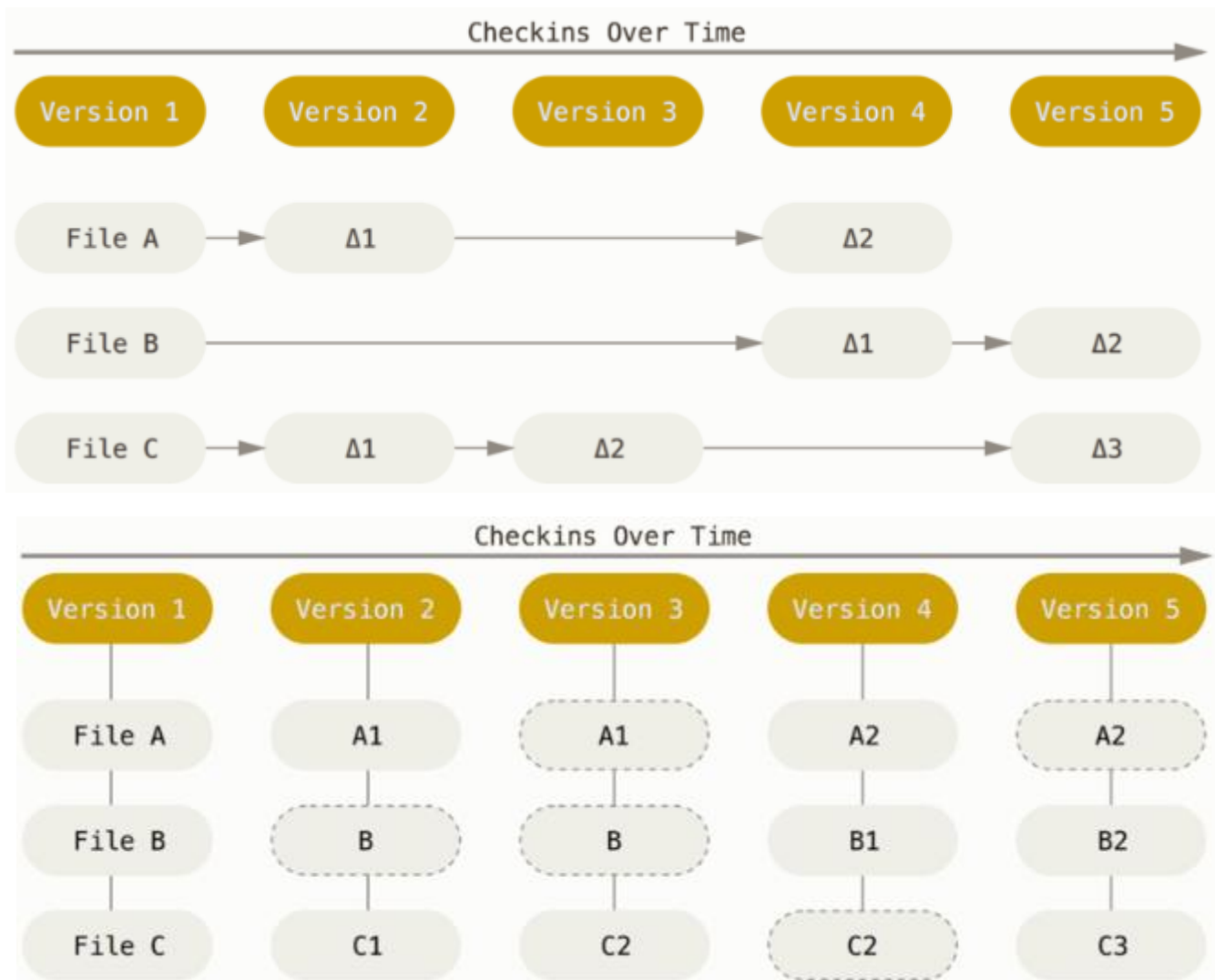
Fundamentos de Git

Git no maneja ni almacena sus datos de esta forma. 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.



Git





Git

Los Tres Estados

Git tiene tres estados principales en los que se pueden encontrar tus archivos: confirmado (**committed**), modificado (**modified**), y preparado (**staged**).

Confirmado: significa que los datos están almacenados de manera segura en tu base de datos local.

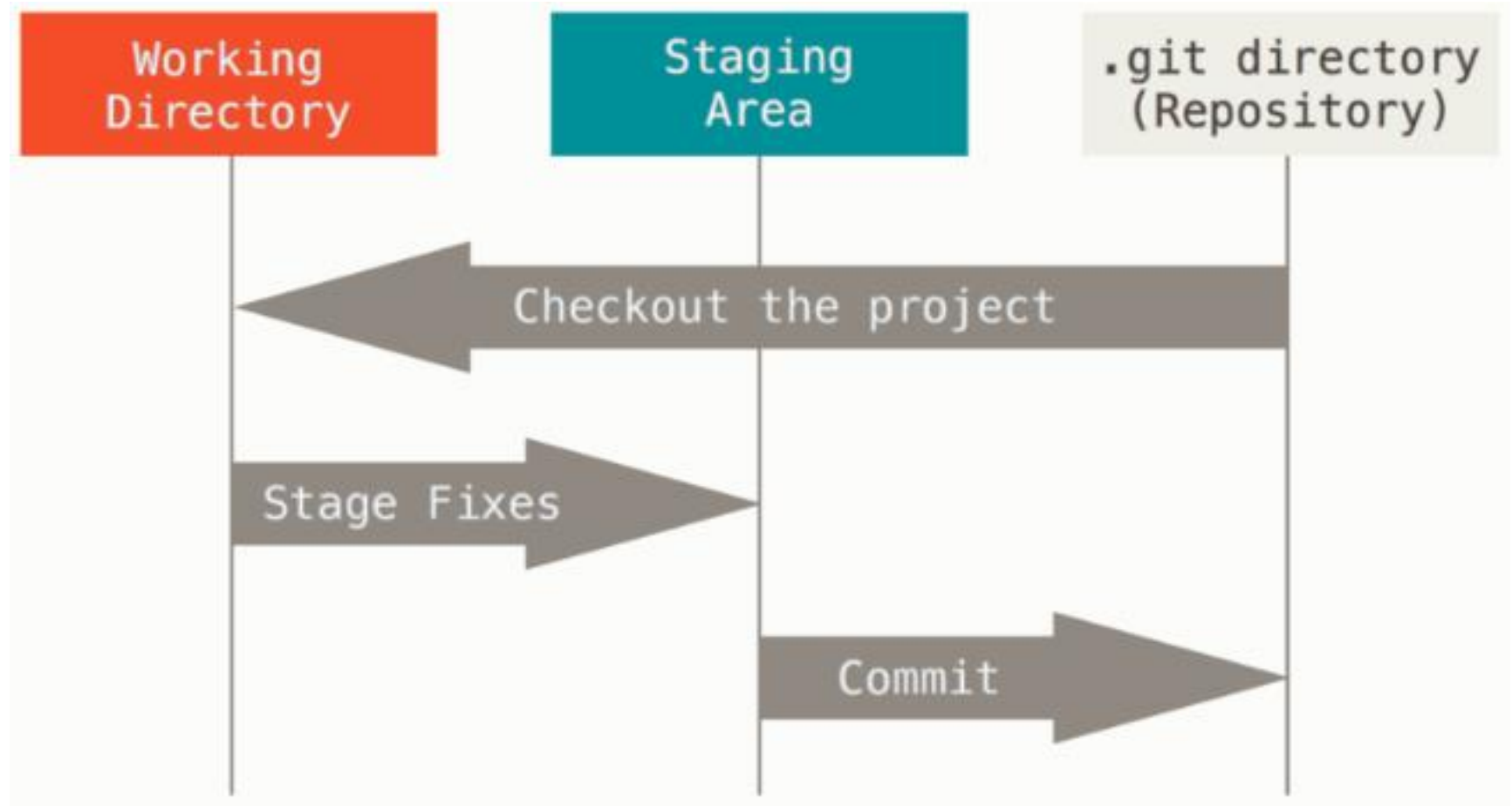
Modificado: significa que has modificado el archivo pero todavía no lo has confirmado a tu base de datos.

Preparado: 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**), el directorio de trabajo (**working directory**), y el área de preparación (**staging area**).



Git





Git

Los Tres Estados

El **directorio de Git** 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** 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** es un archivo, generalmente contenido en tu directorio de Git, que almacena información acerca de lo que va a ir en tu próxima confirmación. A veces se le denomina índice (“index”), pero se está convirtiendo en estándar el referirse a ella como el área de preparación.



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, añadiéndolos a tu área de preparación.
3. Confirmas 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).



Git

¿Qué es el desarrollo colaborativo de software?

Es un modelo de desarrollo basado en la disponibilidad pública del código y la comunicación vía Internet. Este modelo se hizo popular a raíz de su uso para el desarrollo de Linux en 1991.

Tomando como contexto a Git, podríamos decir que el desarrollo colaborativo proporciona herramientas para que un gran número de individuos puedan hacer desarrollos en conjunto de una manera más fácil, menos propensa a errores y rápida de implementar.

De esta manera, siempre tenemos la opción de, por medio de algún cliente, publicar nuestro código junto con todas las etapas y versiones que nos llevó el proyecto para que otras personas puedan sumar y aportar nuevas ideas a nuestro repositorio.



Git

Instalación

The screenshot shows a Google search for 'git'. The search bar contains 'git' and the results show 'Cerca de 1.330.000.000 resultados (0,33 segundos)'. The first result is from 'https://git-scm.com' with the title 'Git' and a description: 'Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.' Below this, there are links for 'Downloads', 'Download for Windows', 'Mac Build', and 'Documentation'. A featured snippet on the right side of the page shows the Git logo and the text 'git' in a large font, with a link to 'Más imágenes'.

git - Buscar con Google

https://www.google.com/search?q=git&source=hp&ei=JGjeY6jfBZHd1sQPouyCsAU&iflsig=AK50M_UAAAAAY952NKVx03F6pfPlcwL6xDouFJolEmc&ved=0ah...

Google

git

Todo Imágenes Vídeos Noticias Maps Más Herramientas

Cerca de 1.330.000.000 resultados (0,33 segundos)

https://git-scm.com Traducir esta página

Git

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Downloads

Windows - Brew install git - GUI Clients - ...

Download for Windows

Click here to download the latest (2.39.1) 32-bit version of Git for ...

Mac Build

There are several options for installing Git on macOS. Note ...

Documentation

Reference - What is Version Control? - Get Going with Git - ...

Más resultados de git-scm.com »

Git

Software

Traducción del inglés - Git es un sistema de control de versiones distribuido que realiza un seguimiento de los cambios en cualquier conjunto de archivos de computadora, generalmente se usa para coordinar el trabajo entre programadores que desarrollan código fuente en colaboración durante el desarrollo de



Git

Instalación

The screenshot shows the Git website homepage. At the top, the browser address bar displays "https://git-scm.com". The main heading is "git --distributed-is-the-new-centralized". Below this, a paragraph describes Git as a "free and open source" distributed version control system. To the right, there is a search bar and a diagram illustrating distributed version control with multiple repositories connected by lines. Below the main text, there are four sections: "About" (advantages of Git), "Documentation" (command reference, Pro Git book), "Downloads" (GUI clients, binary releases), and "Community" (bug reporting, mailing list). On the right side, a monitor displays the "Latest source Release 2.39.1" with a "Download for Windows" button.

Git

https://git-scm.com

git --distributed-is-the-new-centralized

Search entire site...

Git is a **free and open source** distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is **easy to learn** and has a **tiny footprint with lightning fast performance**. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like **cheap local branching**, convenient **staging areas**, and **multiple workflows**.

About
The advantages of Git compared to other source control systems.

Documentation
Command reference pages, Pro Git book content, videos and other material.

Downloads
GUI clients and binary releases for all major platforms.

Community
Get involved! Bug reporting, mailing list, chat, development and more.

Latest source Release
2.39.1
[Release Notes \(2022-12-13\)](#)
[Download for Windows](#)



Git

Instalación

Git - Downloading Package x +

https://git-scm.com/download/win

git --distributed-even-if-your-workflow-isnt

Search entire site...

About

Documentation

Downloads

- GUI Clients
- Logos

Community

The entire **Pro Git book** written by Scott Chacon and Ben Straub is available to [read online for free](#). Dead tree versions are available on [Amazon.com](#).

Download for Windows

[Click here to download](#) the latest (**2.39.1**) **64-bit** version of **Git for Windows**. This is the most recent **maintained build**. It was released **18 days ago**, on 2023-01-17.

Other Git for Windows downloads

Standalone Installer

- [32-bit Git for Windows Setup.](#)
- [64-bit Git for Windows Setup.](#)

Portable ("thumbdrive edition")

- [32-bit Git for Windows Portable.](#)
- [64-bit Git for Windows Portable.](#)

Using winget tool

Install [winget tool](#) if you don't already have it, then type this command in command prompt or Powershell.

```
winget install --id Git.Git -e --source winget
```

The current source code release is version **2.39.1**. If you want the newer version, you can build it from [the source code](#).

Abrir al finalizar

Abrir siempre archivos de este tipo

Pausa

Mostrar en carpeta

Cancelar

Git-2.39.1-64-bit.exe

Mostrar todo



Git

Instalación

Git 2.39.1 Setup

Information

Please read the following important information before continuing.

When you are ready to continue with Setup, click Next.

GNU General Public License

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your
freedom to share and change it. By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change

<https://gitforwindows.org/>

Next

Cancel

Git 2.39.1 Setup

Select Destination Location

Where should Git be installed?



Setup will install Git into the following folder.

To continue, click Next. If you would like to select a different folder, click Browse.

C:\Program Files\Git

Browse...

At least 293,4 MB of free disk space is required.

<https://gitforwindows.org/>

Back

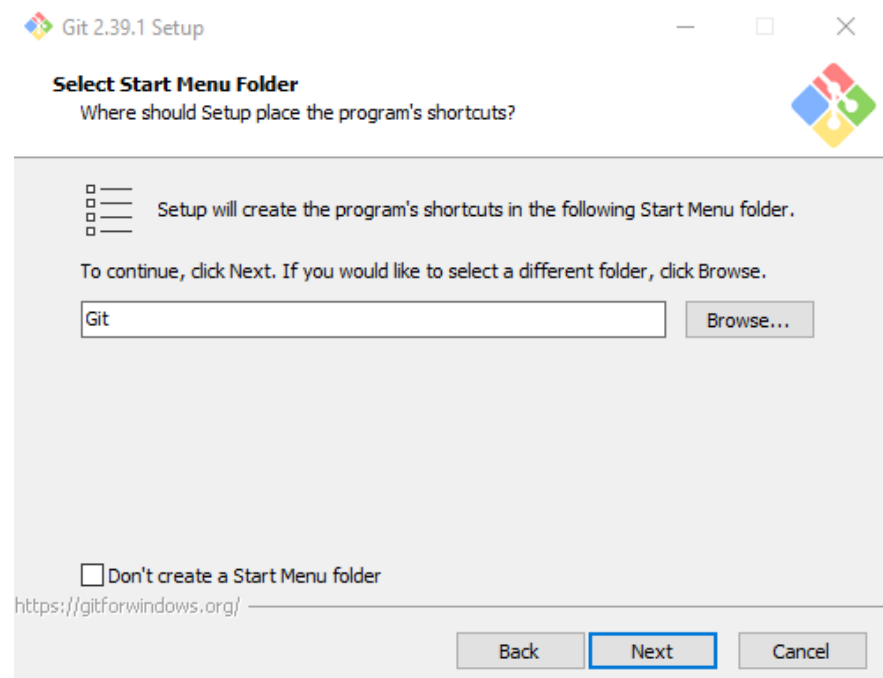
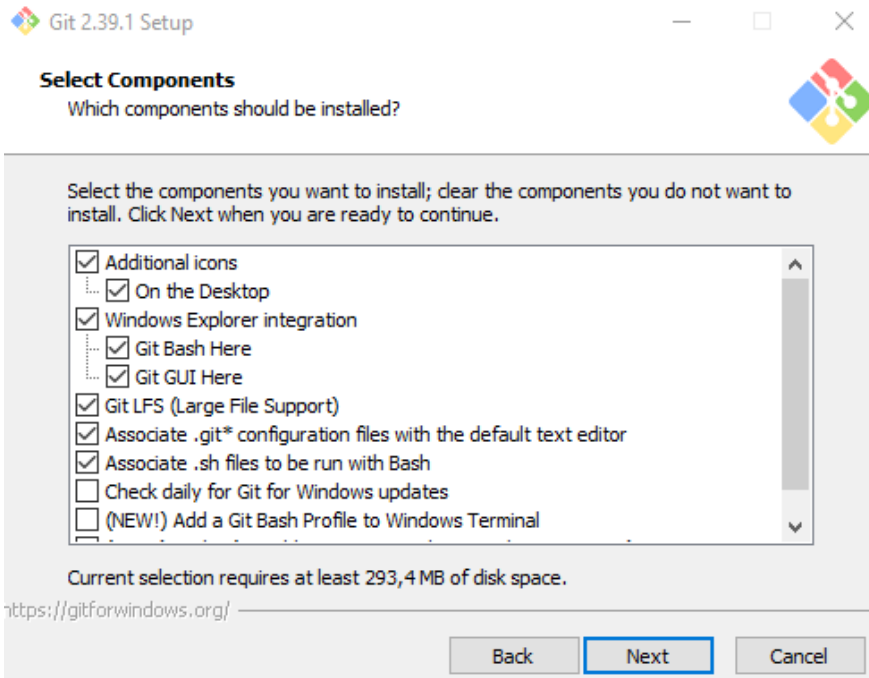
Next

Cancel



Git

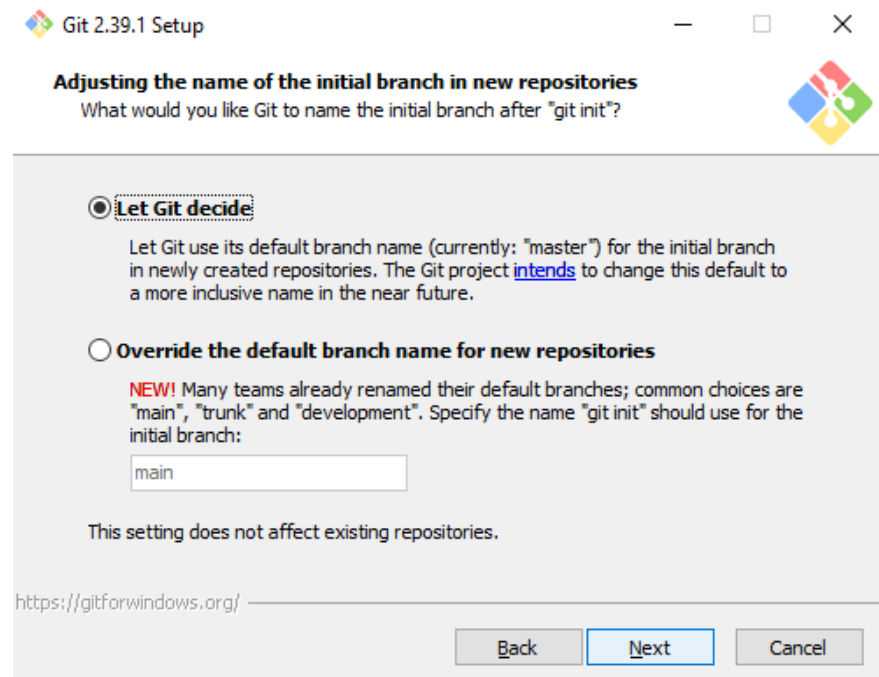
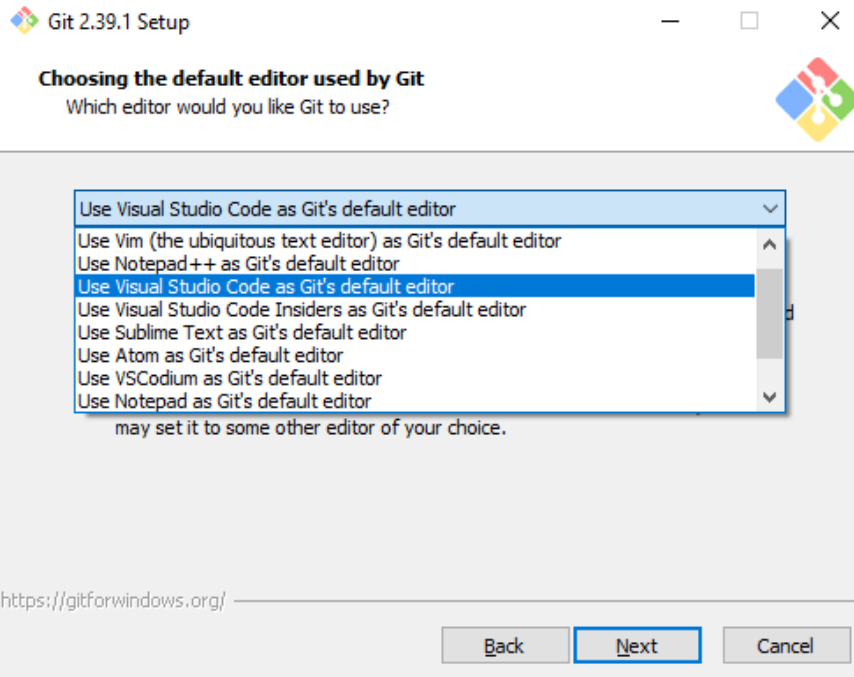
Instalación





Git

Instalación





Git

Instalación

Git 2.39.1 Setup

Adjusting your PATH environment

How would you like to use Git from the command line?

☐ Use Git from Git Bash only

This is the most cautious choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.

☒ **Git from the command line and also from 3rd-party software**

(Recommended) This option adds only some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from Git Bash, the Command Prompt and the Windows PowerShell as well as any third-party software looking for Git in PATH.

☐ Use Git and optional Unix tools from the Command Prompt

Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.

<https://gitforwindows.org/>

Back

Next

Cancel

Git 2.39.1 Setup

Choosing the SSH executable

Which Secure Shell client program would you like Git to use?

☒ **Use bundled OpenSSH**

This uses ssh.exe that comes with Git.

☐ Use (Tortoise)Plink

To use PuTTY, specify the path to an existing copy of (Tortoise)Plink.exe:

C:\Program Files\PuTTY\plink.exe ...

☐ Set ssh.variant for Tortoise Plink

☐ Use external OpenSSH

NEW! This uses an external ssh.exe. Git will not install its own OpenSSH (and related) binaries but use them as found on the PATH.

<https://gitforwindows.org/>

Back

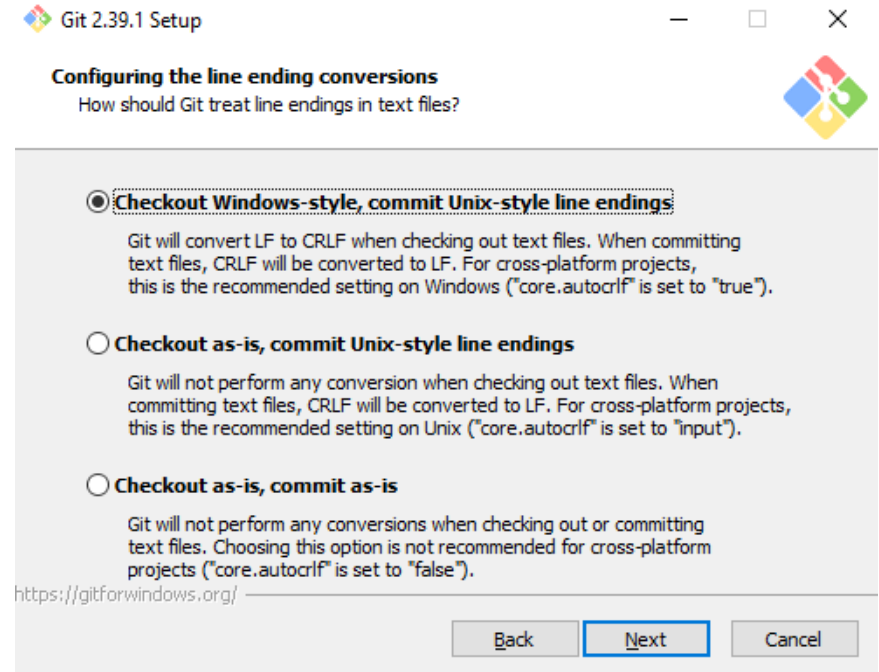
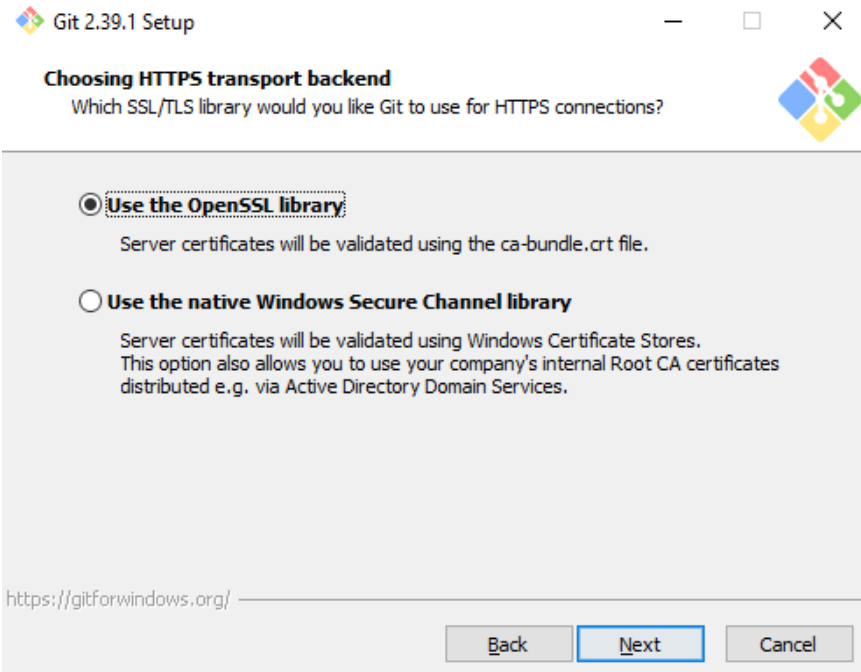
Next

Cancel



Git

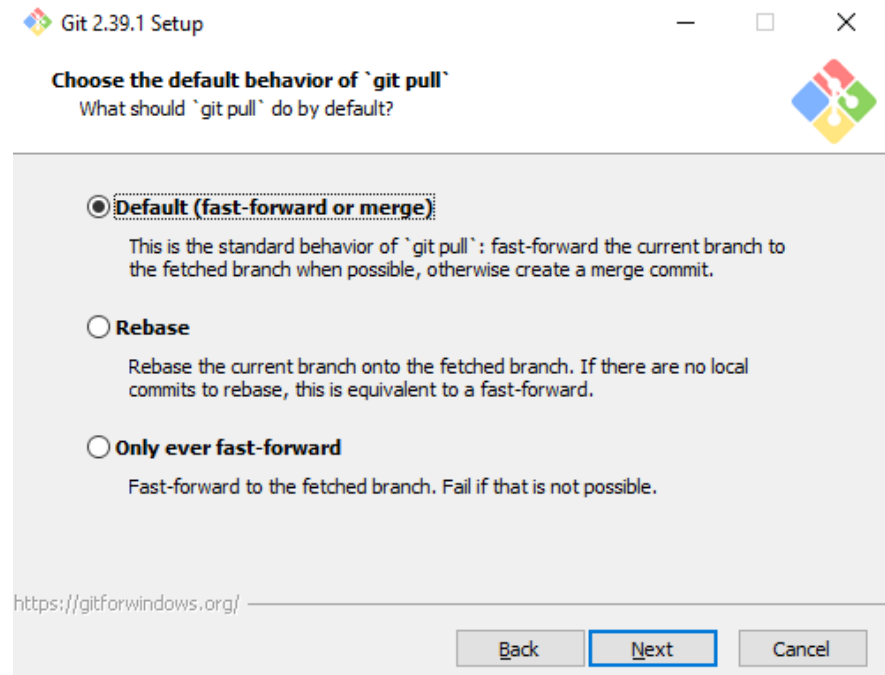
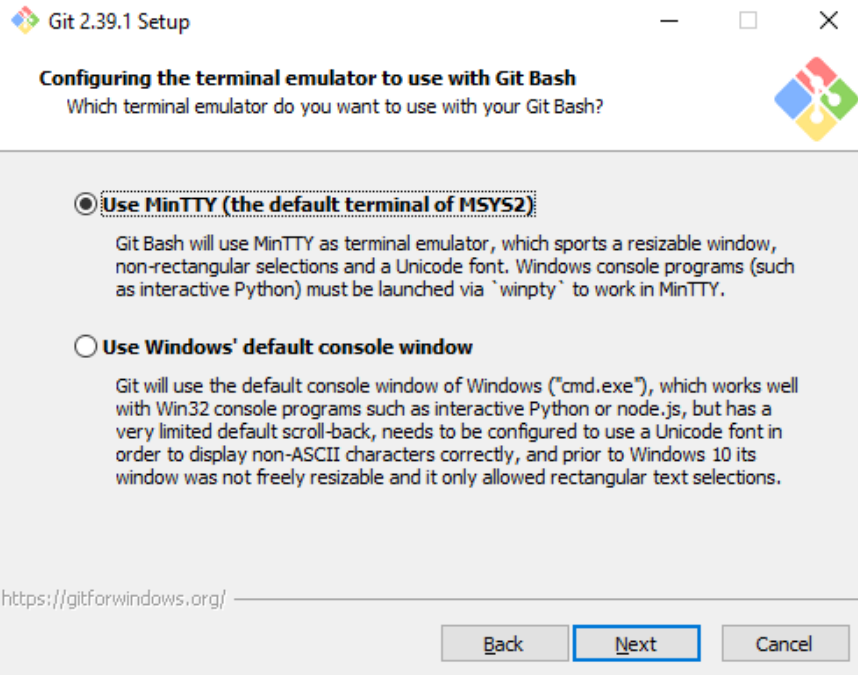
Instalación





Git

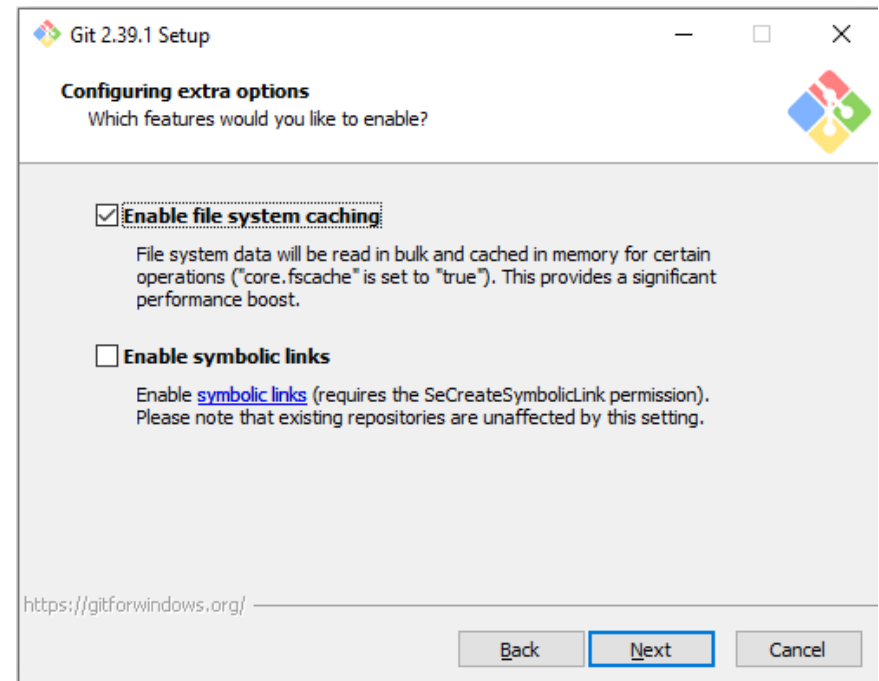
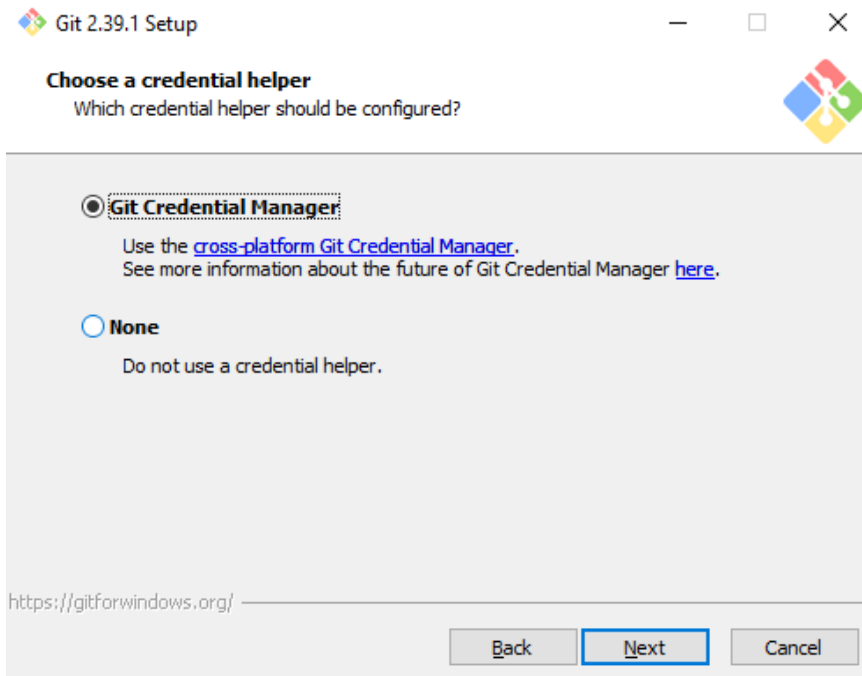
Instalación





Git

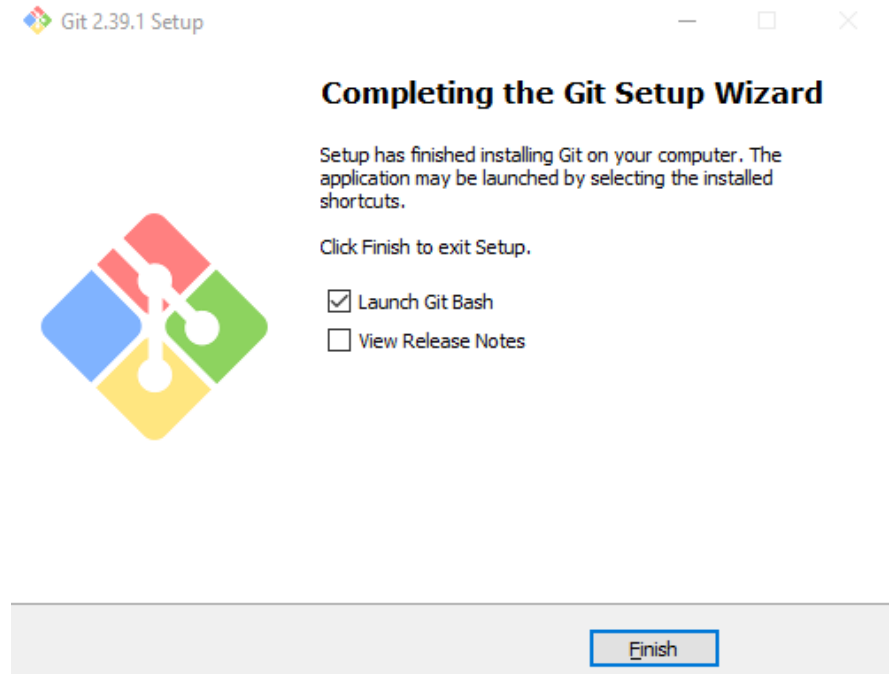
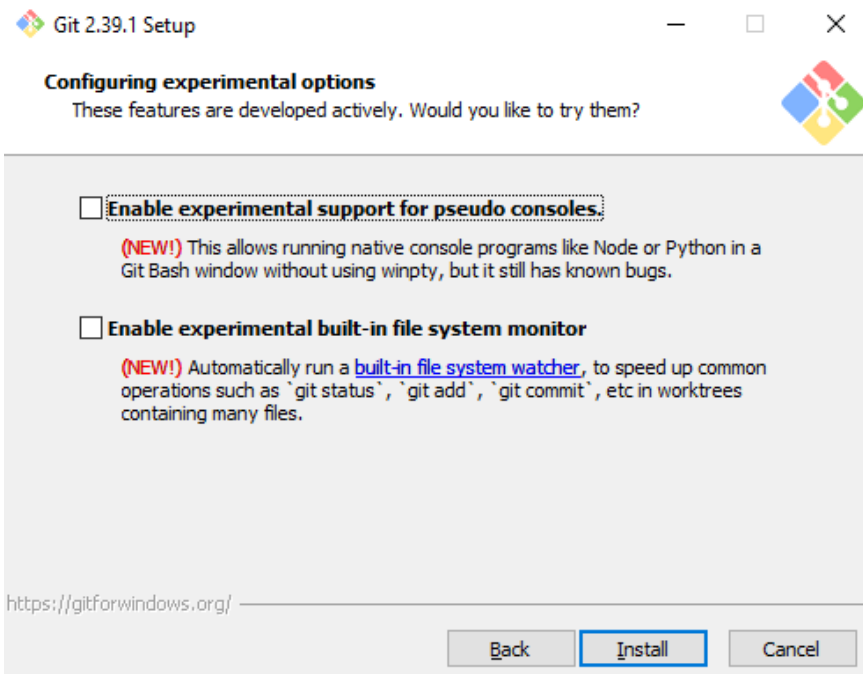
Instalación





Git

Instalación





Git

Configuración:

Git trae una herramienta llamada git config, que te permite obtener y establecer variables de configuración que controlan el aspecto y funcionamiento de Git.

Lo primero que deberás hacer cuando instales Git es establecer tu nombre de usuario y dirección de correo electrónico. Esto es importante porque los "commits" de Git usan esta información, y es introducida de manera inmutable en los commits que envías.

```
$ git config --global user.name "John Doe"
```

```
$ git config --global user.email johndoe@example.com
```

Tu editor por defecto:

```
$ git config --global core.editor "code –wite"
```



Git

Configuración:

Comprobando tu Configuración

Si quieres comprobar tu configuración, puedes usar el comando `git config --list` para mostrar todas las propiedades que Git ha configurado

The screenshot shows a terminal window on the left and a Visual Studio Code editor on the right. The terminal window, titled 'MINGW64/c/Users/diego', shows the following commands and output:

```
diego@DML10 MINGW64 ~  
$ git --version  
git version 2.39.1.windows.1  
  
diego@DML10 MINGW64 ~  
$ git config --global user.name "Diego Luna"  
  
diego@DML10 MINGW64 ~  
$ git config --global user.email profdiegoluna@gmail.com  
  
diego@DML10 MINGW64 ~  
$ git config --global -e  
  
diego@DML10 MINGW64 ~  
$ git config --global core.editor "code --wait"  
  
diego@DML10 MINGW64 ~  
$ git config --global -e  
hint: Waiting for your editor to close the file...
```

The Visual Studio Code editor, titled '.gitconfig', shows the contents of the `.gitconfig` file:

```
1 [filesystem "Oracle Corporation|15.0.1|-1025916383"]  
2     timestampResolution = 2003 microseconds  
3     minRacyThreshold = 0 nanoseconds  
4 [user]  
5     email = profdiegoluna@gmail.com  
6     name = Diego Luna  
7 [filesystem "Oracle Corporation|1.8.0_281|-1025916383"]  
8     timestampResolution = 1001 microseconds  
9     minRacyThreshold = 0 nanoseconds  
10 [filesystem "Azul Systems, Inc.|11.0.10|-1025916383"]  
11     timestampResolution = 343 milliseconds  
12     minRacyThreshold = 0 nanoseconds  
13 [filesystem "Oracle Corporation|11|-1025916383"]  
14     timestampResolution = 93748 microseconds  
15     minRacyThreshold = 0 nanoseconds  
16 [core]  
17     editor = code --wait  
18
```



Git

Obteniendo un repositorio Git

Puedes obtener un proyecto Git de dos maneras. La primera es tomar un proyecto o directorio existente e importarlo en Git. La segunda es clonar un repositorio existente en Git desde otro servidor.

Inicializando un repositorio en un directorio existente

Si estás empezando a seguir un proyecto existente en Git, debes ir al directorio del proyecto y usar el siguiente comando:

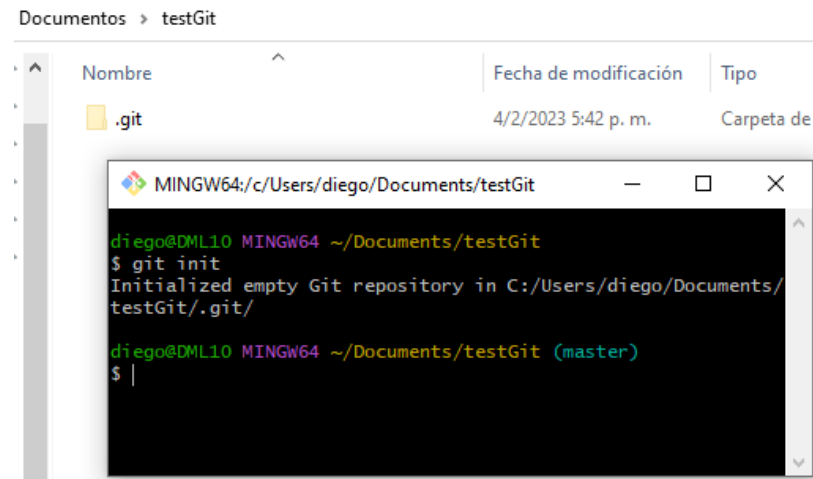
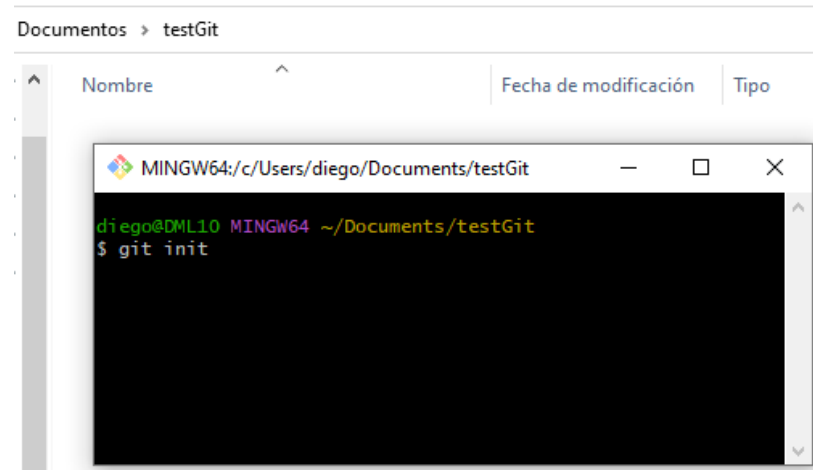
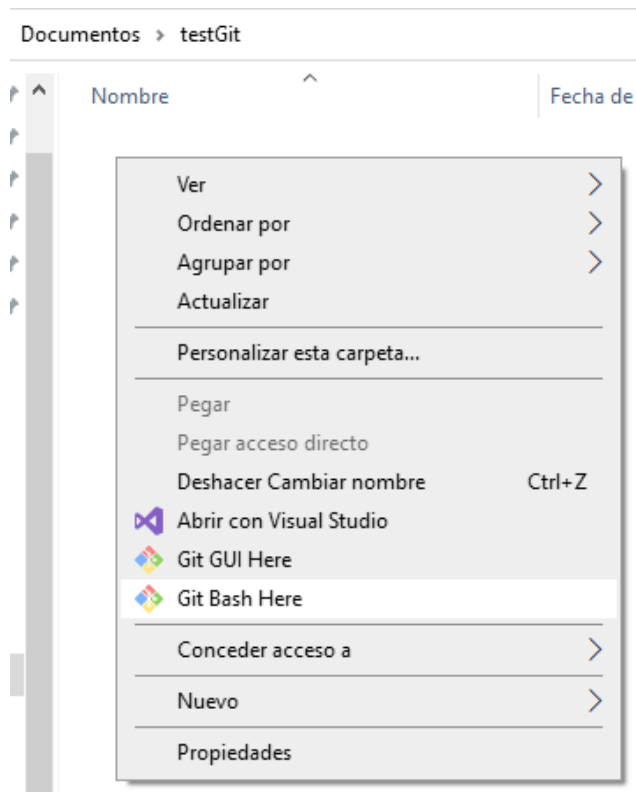
\$ git init

Esto crea un subdirectorio nuevo llamado `.git`, el cual contiene todos los archivos necesarios del repositorio – un esqueleto de un repositorio de Git. Todavía no hay nada en tu proyecto que esté bajo seguimiento.



Git

Obteniendo un repositorio Git

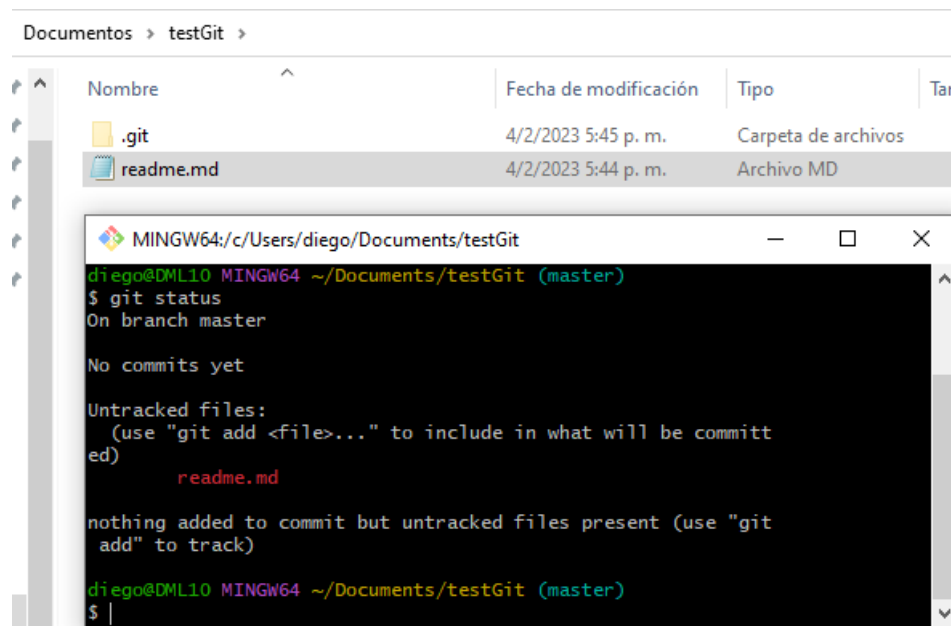




Git

Obteniendo un repositorio Git

Si deseas empezar a controlar versiones de archivos existentes (a diferencia de un directorio vacío), probablemente deberías comenzar el seguimiento de esos archivos y hacer una confirmación inicial. Puedes conseguirlo con unos pocos comandos **git add** para especificar qué archivos quieres controlar, seguidos de un **git commit** para confirmar los cambios.



```
Documentos > testGit >
```

Nombre	Fecha de modificación	Tipo	Tam
.git	4/2/2023 5:45 p. m.	Carpeta de archivos	
readme.md	4/2/2023 5:44 p. m.	Archivo MD	

```
MINGW64:/c/Users/diego/Documents/testGit
diego@DML10 MINGW64 ~/Documents/testGit (master)
$ git status
On branch master

No commits yet

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

        readme.md

nothing added to commit but untracked files present (use "git add" to track)
diego@DML10 MINGW64 ~/Documents/testGit (master)
$
```



Git

Obteniendo un repositorio Git

Documentos > testGit >

Nombre	Fecha de modificación	Tipo	Tamaño
.git	4/2/2023 5:47 p. m.	Carpeta de archivos	
readme.md	4/2/2023 5:44 p. m.	Archivo MD	

```
MINGW64:/c/Users/diego/Documents/testGit
diego@DML10 MINGW64 ~/Documents/testGit (master)
$ git add readme.md

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   readme.md

diego@DML10 MINGW64 ~/Documents/testGit (master)
$
```

Documentos > testGit >

Nombre	Fecha de modificación	Tipo	Tamaño
.git	4/2/2023 5:49 p. m.	Carpeta de archivos	
readme.md	4/2/2023 5:44 p. m.	Archivo MD	

```
MINGW64:/c/Users/diego/Documents/testGit

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   readme.md

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ git commit -m "version inicial del proyecto"
[master (root-commit) fca333e] version inicial del proyecto
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 readme.md

diego@DML10 MINGW64 ~/Documents/testGit (master)
$
```




Git

Clonando un repositorio existente

Para obtener una copia de un repositorio Git existente — por ejemplo, un proyecto en el que te gustaría contribuir — el comando que necesitas es *git clone*.

Cada versión de cada archivo de la historia del proyecto es descargada por defecto cuando ejecutas *git clone*. De hecho, si el disco de tu servidor se corrompe, puedes usar cualquiera de los clones en cualquiera de los clientes para devolver el servidor al estado en el que estaba cuando fue clonado.



Git

Clonando un repositorio existente

The screenshot illustrates the steps to clone an existing repository. It shows the GitHub repository page for 'profdiegoluna/testGitClone', a file explorer window displaying the 'testGitClone' folder in the 'Documentos' directory, and a terminal window where the repository is cloned using the command:

```
diego@DML10 MINGW64 ~/Documents/testGitClone
$ git clone https://github.com/profdiegoluna/testGitClone.git
Cloning into 'testGitClone'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (6/6), done.

diego@DML10 MINGW64 ~/Documents/testGitClone
$
```



Git

Registro de cambios en el repositorio

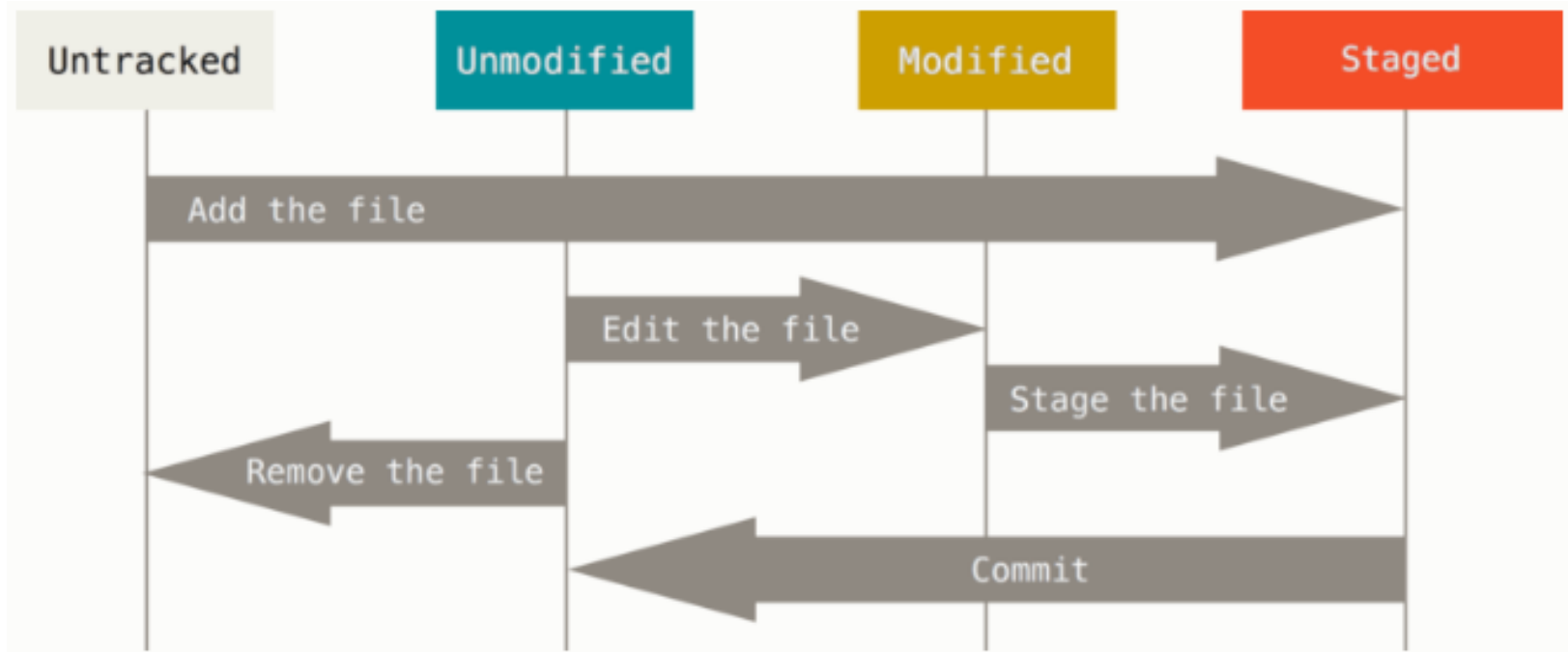
Cada archivo en su directorio de trabajo puede estar en uno de dos estados: **rastreado** o **no rastreado**. Los archivos rastreados son archivos que estaban en la última instantánea, así como cualquier archivo recién preparado; pueden ser sin modificar, modificados o escalonados. En resumen, los archivos rastreados son archivos que Git conoce.

Los archivos sin seguimiento son todo lo demás: cualquier archivo en su directorio de trabajo que no estaba en su última instantánea y que no está en su área de preparación. Cuando clone un repositorio por primera vez, todos sus archivos serán rastreados y no modificados porque Git los revisó y no ha editado nada.

A medida que edita archivos, Git los ve como modificados, porque los ha cambiado desde su última confirmación. A medida que trabaja, prepara selectivamente estos archivos modificados y luego confirma todos esos cambios preparados, y el ciclo se repite.



Git





Git

Comprobación del estado de sus archivos

La herramienta principal que utiliza para determinar qué archivos están en qué estado es el comando **git status**. Si ejecuta este comando directamente después de un clon, debería ver algo como esto:

```
MINGW64:/c/Users/diego/Documents/testGitClone/testGitClone
Receiving objects: 100% (6/6), done.

diego@DML10 MINGW64 ~/Documents/testGitClone
$ cd testGitClone/
ls
diego@DML10 MINGW64 ~/Documents/testGitClone/testGitClone (main)
$ ls
README.md

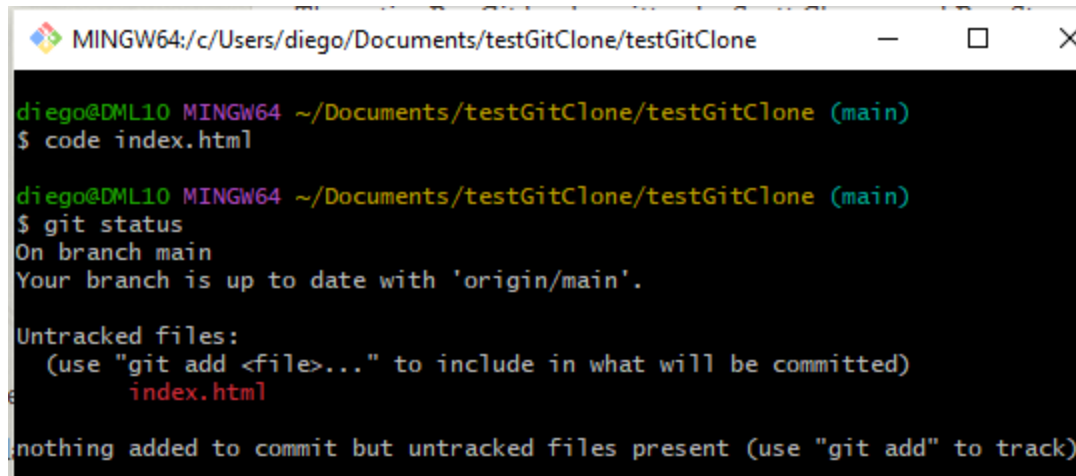
diego@DML10 MINGW64 ~/Documents/testGitClone/testGitClone (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```



Git

Digamos que agrega un nuevo archivo a su proyecto, un index.html archivo simple. Si el archivo no existía antes y ejecuta `git status`, verá su archivo sin seguimiento así:



```
MINGW64:/c:/Users/diego/Documents/testGitClone/testGitClone
diego@DML10 MINGW64 ~/Documents/testGitClone/testGitClone (main)
$ code index.html

diego@DML10 MINGW64 ~/Documents/testGitClone/testGitClone (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

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

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

Básicamente, sin seguimiento significa que Git ve un archivo que no tenía en la instantánea anterior (confirmación) y que aún no se ha preparado; Git no comenzará a incluirlo en tus instantáneas de confirmación hasta que le indiques explícitamente que lo haga.



Git

Seguimiento de nuevos archivos

Para comenzar a rastrear un nuevo archivo, use el comando `git add`. Para comenzar a rastrear el `index.html` archivo, puede ejecutar esto:

```
MINGW64:/c:/Users/diego/Documents/testGitClone/testGitClone
diego@DML10 MINGW64 ~/Documents/testGitClone/testGitClone (main)
$ git add index.html

diego@DML10 MINGW64 ~/Documents/testGitClone/testGitClone (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   index.html
```

El comando `git add` toma un nombre de ruta para un archivo o un directorio; si es un directorio, el comando agrega todos los archivos en ese directorio recursivamente. La versión del archivo en el momento en que lo ejecutó `git add` es la que estará en la siguiente instantánea histórica.



Git

Archivos modificados provisionales

Si cambia un archivo previamente rastreado llamado index.html luego ejecuta su [git status](#) comando nuevamente, obtendrá algo similar a esto:

```
MINGW64:/c/Users/diego/Documents/testGitClone/testGitClone
diego@DML10 MINGW64 ~/Documents/testGitClone/testGitClone (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   index.html

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   index.html
```




Git

Ignorar archivos

A menudo, tendrás una clase de archivos que no deseas que Git agregue automáticamente o que incluso te muestre como sin seguimiento. Por lo general, estos son archivos generados automáticamente, como archivos de registro o archivos producidos por su sistema de compilación. En tales casos, puede crear un archivo que enumere los patrones para que coincidan con el nombre [.gitignore](#).

```
$ cat .gitignore
*.[oa]
*~
```

La primera línea le dice a Git que ignore cualquier archivo que termine en ".o" o ".a": archivos de objetos y archivos que pueden ser el producto de la construcción de su código. La segunda línea le dice a Git que ignore todos los archivos cuyos nombres terminen con una tilde (~),



Git

Confirmar sus cambios

Ahora que su área de preparación está configurada de la manera que desea, puede confirmar sus cambios. Recuerde que cualquier cosa que aún no esté preparada (cualquier archivo que haya creado o modificado que no haya ejecutado *git add* desde que los editó) no entrará en esta confirmación. Permanecerán como archivos modificados en su disco.

La forma más sencilla de confirmar es escribir *git commit*

Alternativamente, puede escribir su mensaje de confirmación en línea con el *commit* comando especificándolo después de una *-m* bandera

```
MINGW64:/c/Users/diego/Documents/testGitClone/testGitClone  —  [
diego@DML10 MINGW64 ~/Documents/testGitClone/testGitClone (main)
$ git commit -m "Instantanea inicial"
[main 529506f] Instantanea inicial
1 file changed, 29 insertions(+)
create mode 100644 index.html
```



Git

Saltar el Área de Preparación

A pesar de que puede resultar muy útil para ajustar los commits tal como quieres, el área de preparación es a veces un paso más complejo de lo que necesitas para tu flujo de trabajo. Si quieres saltarte el área de preparación, Git te ofrece un atajo sencillo.

Añadiendo la opción **-a** al comando **git commit** harás que Git prepare automáticamente todos los archivos rastreados antes de confirmarlos, ahorrándote el paso de **git add**.

```
MINGW64:/c/Users/diego/Documents/testGit

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   readme.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   readme.md

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ git commit -a -m "saltando preparación"
[master b539b0d] saltando preparación
1 file changed, 49 insertions(+)

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ |
```



Git

Eliminar Archivos

Para eliminar archivos de Git, debes eliminarlos de tus archivos rastreados (o mejor dicho, eliminarlos del área de preparación) y luego confirmar. Para ello existe el comando **git rm**, que además elimina el archivo de tu directorio de trabajo de manera que no aparezca la próxima vez como un archivo no rastreado.

```
MINGW64:/c/Users/diego/Documents/testGit

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ git rm readme.md
rm 'readme.md'

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    readme.md


diego@DML10 MINGW64 ~/Documents/testGit (master)
$
```



Git

Cambiar el Nombre de los Archivos

Git tiene un comando **mv**. Si quieres renombrar un archivo en Git, puedes ejecutar algo como **git mv file_from file_to**

 MINGW64:/c/Users/diego/Documents/testGit

```
diego@DML10 MINGW64 ~/Documents/testGit (master)
$ git mv readme.md readme1.md

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:    readme.md -> readme1.md

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ |
```



Git

Ver el Historial de Confirmaciones

Después de haber hecho varias confirmaciones, o si has clonado un repositorio que ya tenía un histórico de confirmaciones, probablemente quieras mirar atrás para ver qué modificaciones se han llevado a cabo. La herramienta más básica y potente para hacer esto es el comando **git log**.

Por defecto, si no pasas ningún parámetro, **git log** lista las confirmaciones hechas sobre ese repositorio en orden cronológico inverso.

```
MINGW64:/c/Users/diego/Documents/testGit

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ git log
commit b65b1147db0b23c11c50f480727125f45b4dac50 (HEAD -> master)
Author: Diego Luna <profdiegoluna@gmail.com>
Date: Sat Feb 25 21:36:14 2023 -0300

    nuevo readme.md

commit 13264690e068b3f484bb9b3a5b5eb7c9bc0b44
Author: Diego Luna <profdiegoluna@gmail.com>
Date: Sat Feb 25 21:34:52 2023 -0300

    eliminando archivo

commit b539b0d565864ca0a91e9b7d61bfc58adc764e
Author: Diego Luna <profdiegoluna@gmail.com>
Date: Sat Feb 25 21:25:58 2023 -0300

    saltando preparación

commit fca333ef9bce41beccf0ddc084b56ea962683ec7
Author: Diego Luna <profdiegoluna@gmail.com>
Date: Sat Feb 4 17:49:19 2023 -0300

    version inicial del proyecto

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ |
```



Git

Tabla 2. Opciones típicas de `git log`

Opción	Descripción
<code>-p</code>	Muestra el parche introducido en cada confirmación.
<code>--stat</code>	Muestra estadísticas sobre los archivos modificados en cada confirmación.
<code>--shortstat</code>	Muestra solamente la línea de resumen de la opción <code>--stat</code> .
<code>--name-only</code>	Muestra la lista de archivos afectados.
<code>--name-status</code>	Muestra la lista de archivos afectados, indicando además si fueron añadidos, modificados o eliminados.
<code>--abbrev-commit</code>	Muestra solamente los primeros caracteres de la suma SHA-1, en vez de los 40 caracteres de que se compone.
<code>--relative-date</code>	Muestra la fecha en formato relativo (por ejemplo, “2 weeks ago” (“hace 2 semanas”)) en lugar del formato completo.
<code>--graph</code>	Muestra un gráfico ASCII con la historia de ramificaciones y uniones.
<code>--pretty</code>	Muestra las confirmaciones usando un formato alternativo. Posibles opciones son <code>oneline</code> , <code>short</code> , <code>full</code> , <code>fuller</code> y <code>format</code> (mediante el cual puedes especificar tu propio formato).



Git

Deshacer Cosas

Uno de las acciones más comunes a deshacer es cuando confirmas un cambio antes de tiempo y olvidas agregar algún archivo, o te equivocas en el mensaje de confirmación.

Si quieres rehacer la confirmación, puedes reconfirmar con la opción **--amend**

```
MINGW64:/c/Users/diego/Documents/testGit

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ git log -1
commit ceab488ebe0420590a0656f2e0e57a0168f2be36 (HEAD -> master)
Author: Diego Luna <profdiegoluna@gmail.com>
Date: Sat Feb 25 21:36:14 2023 -0300

    Nuevo readme.md

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ git commit --amend -m "Nuevo readme.md completo"
[master fa0272b] Nuevo readme.md completo
Date: Sat Feb 25 21:36:14 2023 -0300
1 file changed, 48 insertions(+)
create mode 100644 readme1.md

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ git log -1
commit fa0272b8a5a988e31f853216f18886a2da42074c (HEAD -> master)
Author: Diego Luna <profdiegoluna@gmail.com>
Date: Sat Feb 25 21:36:14 2023 -0300

    Nuevo readme.md completo

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ |
```




Git

Deshacer un Archivo Preparado

Por ejemplo, supongamos que has cambiado dos archivos y que quieres confirmarlos como dos cambios separados, pero accidentalmente has escrito **git add** y has preparado ambos.

¿Cómo puedes sacar del área de preparación uno de ellos?

El comando **git status** te recuerda cómo:

git restore --staged <file>

```
MINGW64:/c/Users/diego/Documents/testGit

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   contribuciones.md
    renamed:    readme1.md -> readme.md

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ git restore --staged contribuciones.md

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    renamed:    readme1.md -> readme.md

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

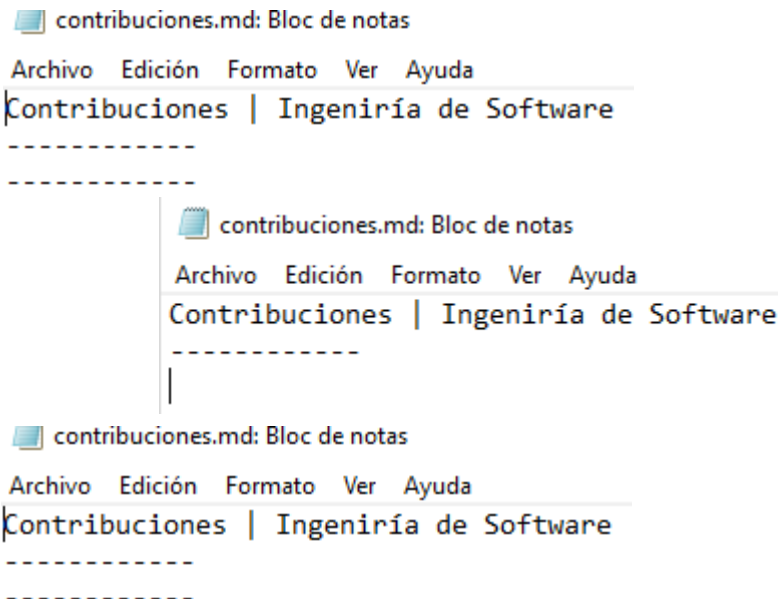
diego@DML10 MINGW64 ~/Documents/testGit (master)
$
```



Git

Deshacer un Archivo Modificado

¿Qué tal si te das cuenta que no quieres mantener los cambios del archivo *contribuciones.md*? ¿Cómo puedes restaurarlo fácilmente - volver al estado en el que estaba en la última confirmación (o cuando estaba recién clonado, o como sea que haya llegado a tu directorio de trabajo)?: ***git checkout -- contribuciones.md***



```
MINGW64:/c/Users/diego/Documents/testGit

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   contribuciones.md

no changes added to commit (use "git add" and/or "git commit -a")

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ git checkout -- contribuciones.md

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ git status
On branch master
nothing to commit, working tree clean

diego@DML10 MINGW64 ~/Documents/testGit (master)
$
```



Git

Deshacer un Archivo Modificado

Recupero la versión de contribuciones.md de un **commit** anterior.

git checkout 16c33c1 contribuciones.md

contribuciones.md: Bloc de notas

Archivo Edición Formato Ver Ayuda

Contribuciones | Ingeniería de Software

contribuciones.md: Bloc de notas

Archivo Edición Formato Ver Ayuda

Contribuciones | Ingeniería de Software

También podemos llevar no sólo un archivo a un punto predeterminado, si no todos los archivos del repositorio, para ello escribimos: ***git checkout 16c33c1***

```
MINGW64:/c/Users/diego/Documents/testGit

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ git log --oneline
03d4d8c (HEAD -> master) Versión de contribuciones.md con 2 líneas
1aa654b Elimino línea en contribuciones.md
f6fb16d Agrego línea en contribuciones.md
16c33c1 Modifico contribuciones.md
fa0272b Nuevo readme.md completo
1326469 eliminando archivo
b539b0d saltando preparación
fca333e version inicial del proyecto

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ git checkout 16c33c1 contribuciones.md
Updated 1 path from 742d206

diego@DML10 MINGW64 ~/Documents/testGit (master)
$ |
```



Git

Ramificaciones en Git

Cuando hablamos de ramificaciones, significa que tú has tomado la rama principal de desarrollo (master) y a partir de ahí has continuado trabajando sin seguir la rama principal de desarrollo.

¿Qué es una rama?

Para entender realmente cómo ramifica Git, previamente hemos de examinar la forma en que almacena sus datos.

Recordando lo citado en Inicio - Sobre el Control de Versiones, Git no los almacena de forma incremental (guardando solo diferencias), sino que los almacena como una serie de instantáneas (copias puntuales de los archivos completos, tal y como se encuentran en ese momento).



Git

¿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).

Vamos a suponer, por ejemplo, que tienes una carpeta con **tres archivos**, que preparas (**stage**) todos ellos y los confirmas (**commit**).

Al preparar los archivos, Git realiza una **suma de control de cada uno** de ellos (un resumen SHA-1), **almacena una copia de cada** uno en el repositorio (estas copias se denominan "**blobs**"), y guarda cada suma de control en el área de preparación (**staging area**).



Git

Cuando creas una confirmación con el comando **git commit**, Git realiza **sumas de control de cada subdirectorio** (en el ejemplo, solamente tenemos el directorio principal del proyecto), y las guarda **como objetos árbol** en el repositorio Git. Después, Git crea un **objeto de confirmación con los metadatos** pertinentes y un **apuntador al objeto árbol raíz** del proyecto.

En este momento, el repositorio de Git contendrá **cinco objetos**: un "blob" para cada uno de los **tres archivos**, un **árbol**

con la lista de contenidos del directorio (más sus respectivas relaciones con los "blobs"), y **una confirmación** de cambios (commit) apuntando a la raíz de ese árbol y conteniendo el resto de metadatos pertinentes.

```
MINGW64:/c:/Users/diego/Documents/GIT/testBranch

diego@DML10 MINGW64 ~/Documents/GIT/testBranch
$ git init
Initialized empty Git repository in C:/Users/diego/Documents/GIT/testBranch/.git/

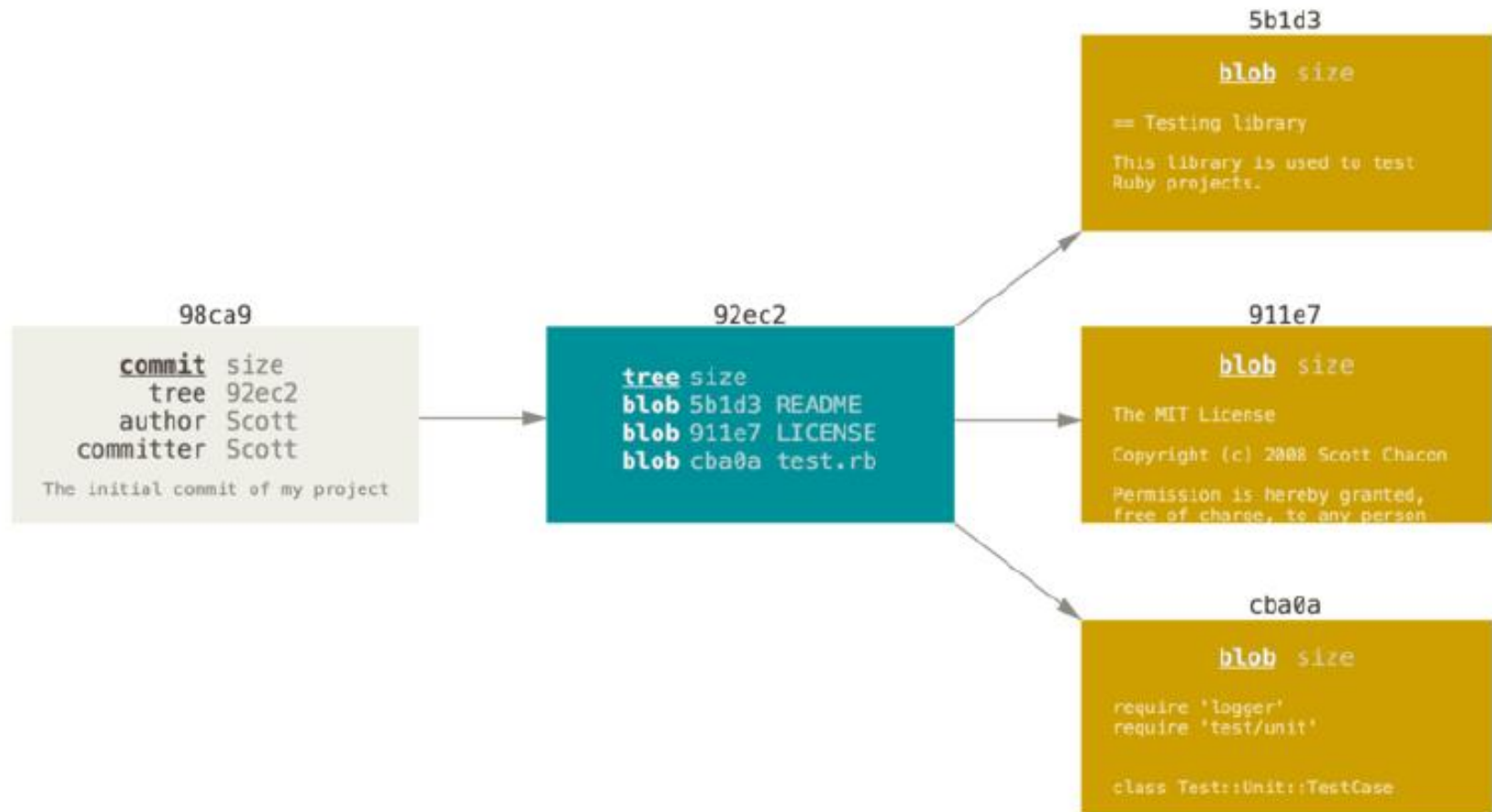
diego@DML10 MINGW64 ~/Documents/GIT/testBranch (master)
$ git add readme.md licencia.txt index.html

diego@DML10 MINGW64 ~/Documents/GIT/testBranch (master)
$ git commit -m "inicio del proyecto"
[master (root-commit) 5dd8e76] inicio del proyecto
3 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 index.html
create mode 100644 licencia.txt
create mode 100644 readme.md

diego@DML10 MINGW64 ~/Documents/GIT/testBranch (master)
$ |
```



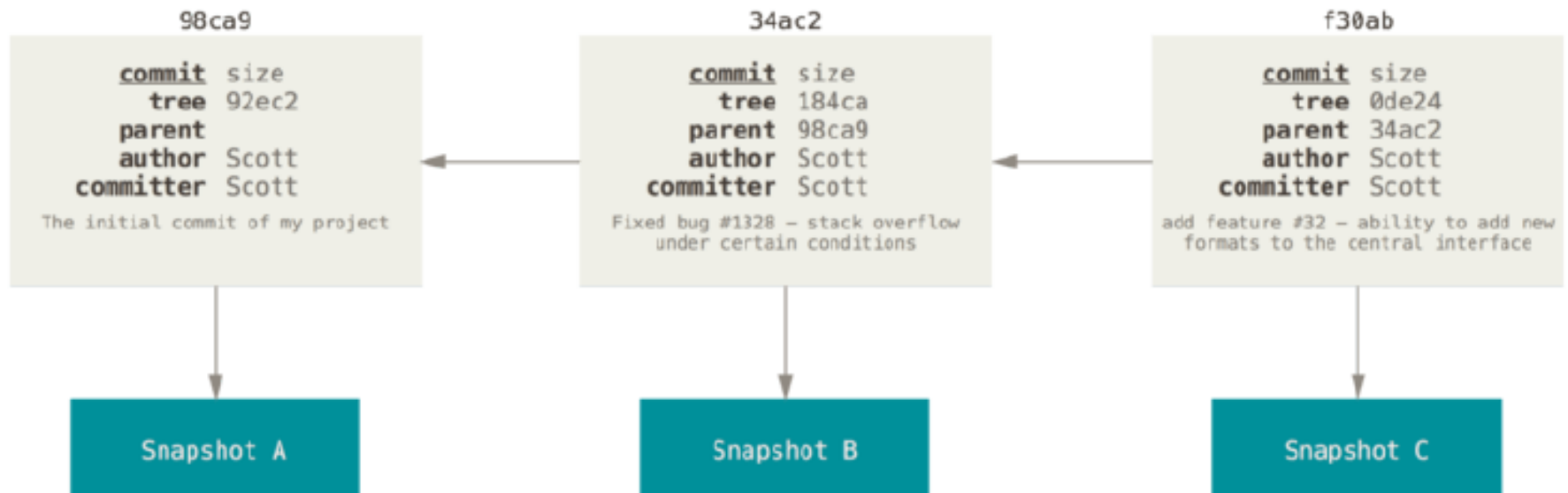
Git





Git

Si haces más cambios y vuelves a confirmar, la siguiente confirmación guardará un apuntador a su confirmación precedente.

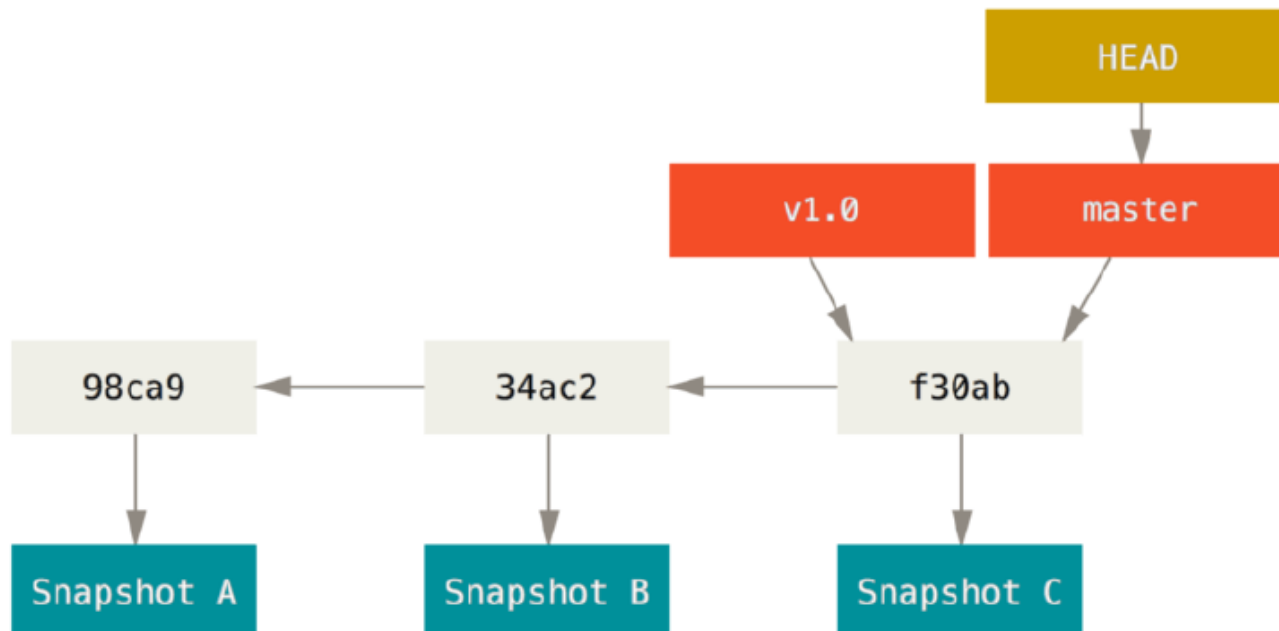




Git

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.**





Git

Crear una Rama Nueva

Por ejemplo, supongamos que quieres crear una rama nueva denominada "testing". Para ello, usarás el comando **git Branch**

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

MINGW64:/c/Users/diego/Documents/GIT/testBranch

```
diego@DML10 MINGW64 ~/Documents/GIT/testBranch (master)
$ git branch testing

diego@DML10 MINGW64 ~/Documents/GIT/testBranch (master)
$ git log --oneline
5dd8e76 (HEAD -> master, testing) inicio del proyecto

diego@DML10 MINGW64 ~/Documents/GIT/testBranch (master)
$ git branch
* master
  testing

diego@DML10 MINGW64 ~/Documents/GIT/testBranch (master)
$ |
```

MINGW64:/c/Users/diego/Documents/GIT/testBranch

```
diego@DML10 MINGW64 ~/Documents/GIT/testBranch (master)
$ git branch testing

diego@DML10 MINGW64 ~/Documents/GIT/testBranch (master)
$ |
```

¿cómo sabe Git en qué rama estás en este momento? Lo hace mediante un apuntador especial denominado **HEAD**.

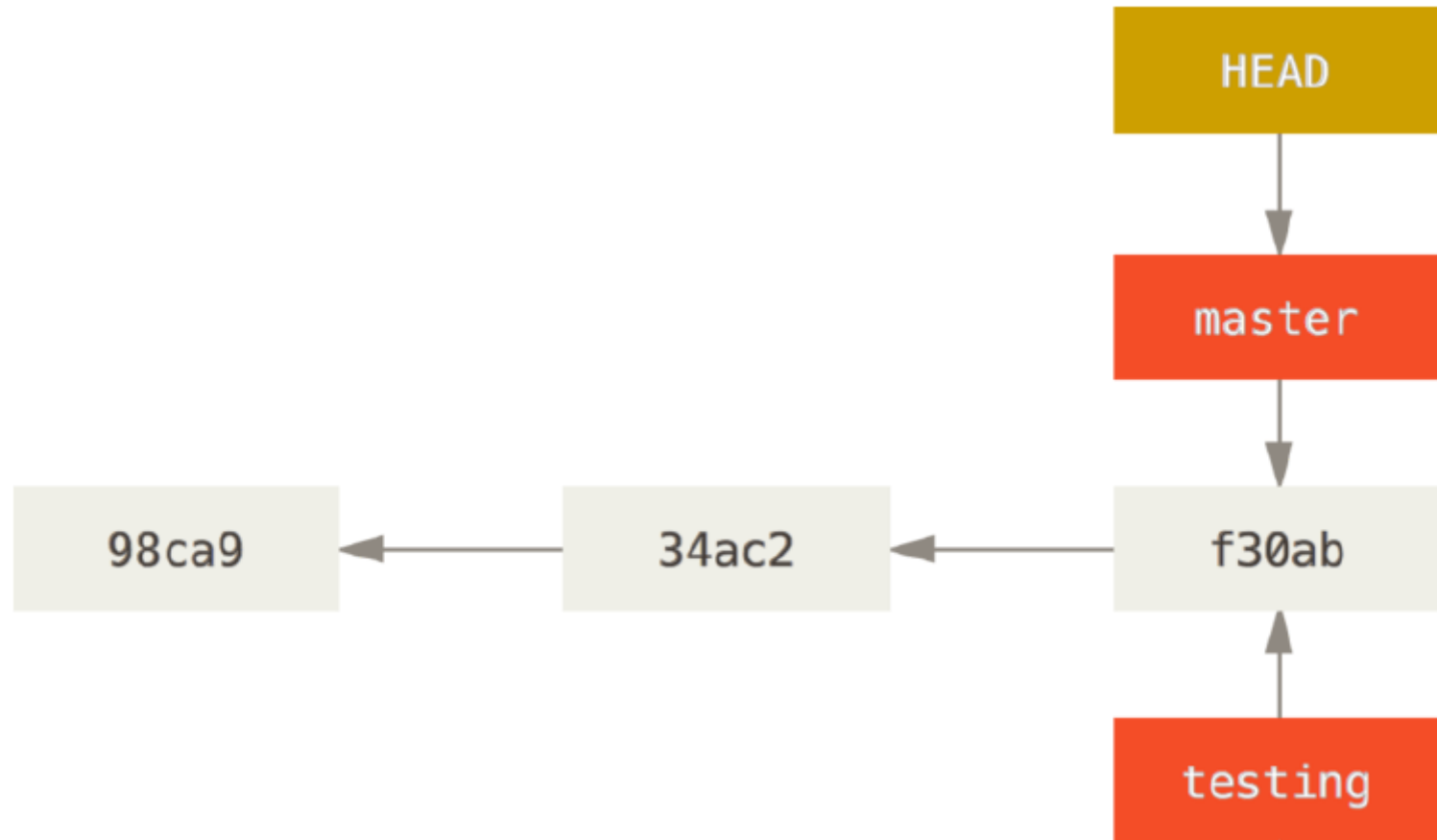
MINGW64:/c/Users/diego/Documents/GIT/testBranch

```
diego@DML10 MINGW64 ~/Documents/GIT/testBranch (master)
$ git log --oneline --decorate
5dd8e76 (HEAD -> master, testing) inicio del proyecto

diego@DML10 MINGW64 ~/Documents/GIT/testBranch (master)
$ |
```



Git





Git

Cambiar de Rama

Para saltar de una rama a otra, tienes que utilizar el comando **git checkout**. Hagamos una prueba, saltando a la rama testing recién creada.

Esto mueve el apuntador HEAD a la rama testing.



```
MINGW64:/c:/Users/diego/Documents/GIT/testBranch

diego@DML10 MINGW64 ~/Documents/GIT/testBranch (master)
$ git log --oneline --decorate
5dd8e76 (HEAD -> master, testing) inicio del proyecto

diego@DML10 MINGW64 ~/Documents/GIT/testBranch (master)
$ git checkout testing
Switched to branch 'testing'

diego@DML10 MINGW64 ~/Documents/GIT/testBranch (testing)
$ git log --oneline --decorate
5dd8e76 (HEAD -> testing, master) inicio del proyecto

diego@DML10 MINGW64 ~/Documents/GIT/testBranch (testing)
$ |
```



Git

¿Cuál es el significado de todo esto? Bueno..., lo veremos tras realizar otra confirmación de cambios:

```
MINGW64:/c:/Users/diego/Documents/GIT/testBranch

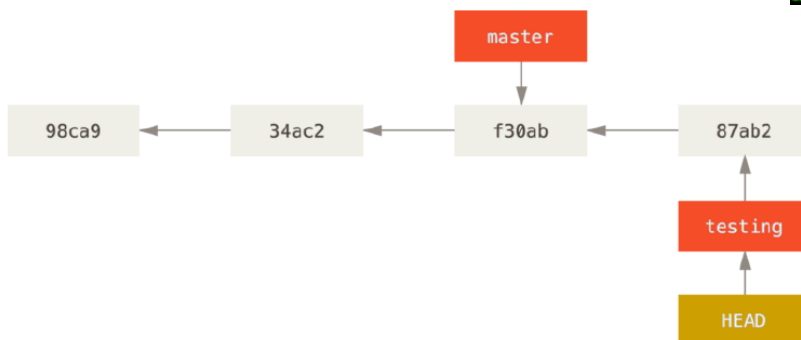
diego@DML10 MINGW64 ~/Documents/GIT/testBranch (testing)
$ git status
On branch testing
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   index.html

no changes added to commit (use "git add" and/or "git commit -a")

diego@DML10 MINGW64 ~/Documents/GIT/testBranch (testing)
$ git add index.html

diego@DML10 MINGW64 ~/Documents/GIT/testBranch (testing)
$ git commit -m "Cambios en index.html - geolocalización"
[testing bb5e8f9] Cambios en index.html - geolocalización
1 file changed, 29 insertions(+)

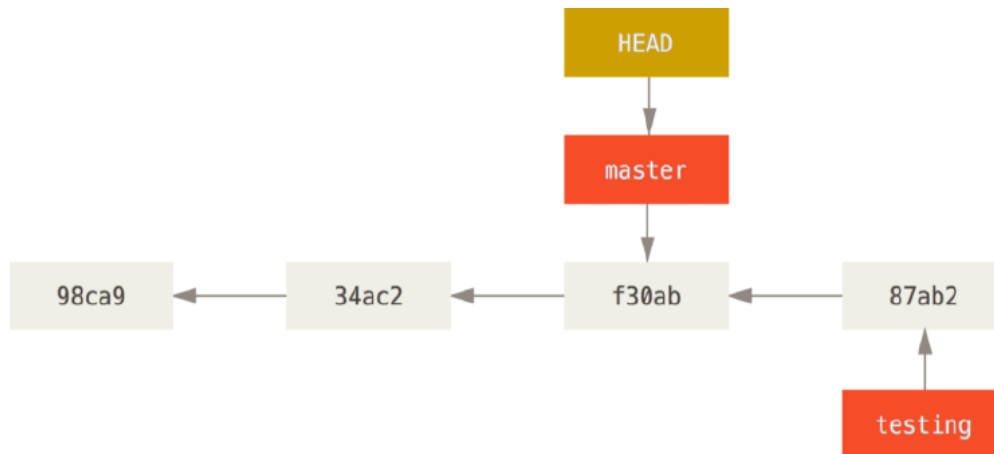
diego@DML10 MINGW64 ~/Documents/GIT/testBranch (testing)
```





Git

La rama testing avanza, mientras que la rama master permanece en la confirmación donde estaba cuando lanzaste el comando **git checkout** para saltar. Volvamos ahora a la rama master



```
MINGW64:/c/Users/diego/Documents/GIT/testBranch

diego@DML10 MINGW64 ~/Documents/GIT/testBranch (testing)
$ git checkout master
Switched to branch 'master'

diego@DML10 MINGW64 ~/Documents/GIT/testBranch (master)
$ git log --oneline --decorate
5dd8e76 (HEAD -> master) inicio del proyecto

diego@DML10 MINGW64 ~/Documents/GIT/testBranch (master)
$
```

Este comando realiza dos acciones: Mueve el apuntador HEAD de nuevo a la rama master, y **revierte los archivos de tu directorio de trabajo**; dejándolos tal y como estaban en la última instantánea confirmada en dicha rama master. Esto supone que los cambios que hagas desde este momento en adelante, divergirán de la antigua versión del proyecto. Básicamente, lo que se está haciendo **es rebobinar el trabajo** que habías hecho temporalmente en la rama testing; de tal forma que puedas avanzar en otra dirección diferente.



Git

Realizamos un cambio a index.html lo preparamos y confirmamos:

Ahora el historial de tu proyecto diverge. Has creado una rama y saltado a ella, has trabajado sobre ella; has vuelto a la rama original, y has trabajado también sobre ella. Los cambios realizados en ambas sesiones de trabajo

están aislados en ramas independientes: puedes saltar libremente de una a otra según estimes oportuno. Y todo ello simplemente con tres comandos: **git branch**, **git checkout** y **git commit**.

```
MINGW64:/c/Users/diego/Documents/GIT/testBranch

diego@DML10 MINGW64 ~/Documents/GIT/testBranch (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   index.html

no changes added to commit (use "git add" and/or "git commit -a")

diego@DML10 MINGW64 ~/Documents/GIT/testBranch (master)
$ git add index.html

diego@DML10 MINGW64 ~/Documents/GIT/testBranch (master)
$ git commit -m "otros cambios a index"
[master 1462c7c] otros cambios a index
1 file changed, 9 insertions(+)

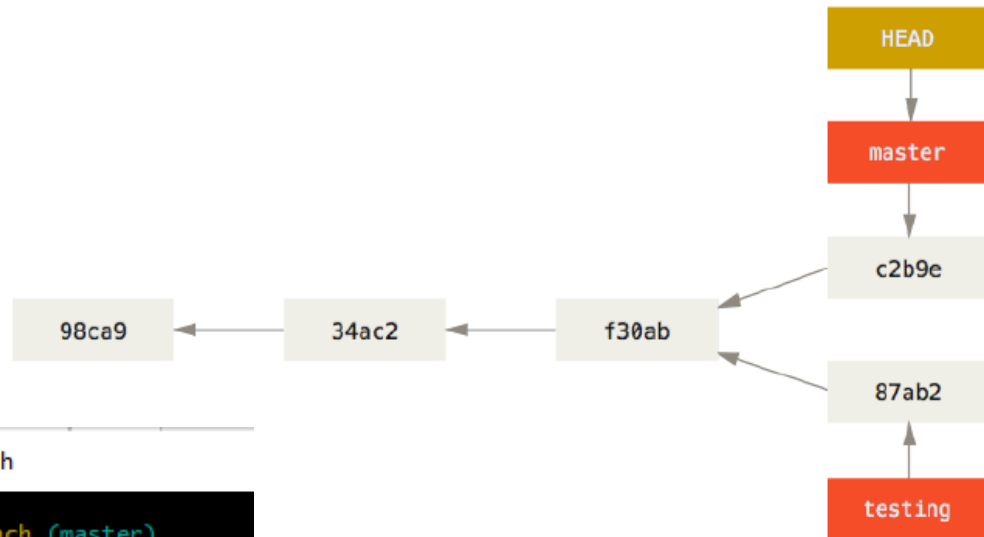
diego@DML10 MINGW64 ~/Documents/GIT/testBranch (master)
$
```



Git

También puedes ver esto fácilmente utilizando el comando `git log`.

Si ejecutas `git log --oneline --decorate --graph --all` te mostrará el historial de tus confirmaciones, indicando dónde están los apuntadores de tus ramas y como ha divergido tu historial.



MINGW64:/c/Users/diego/Documents/GIT/testBranch

```
diego@DML10 MINGW64 ~/Documents/GIT/testBranch (master)
$ git log --oneline --decorate --graph --all
* 1462c7c (HEAD -> master) otros cambios a index
| * bb5e8f9 (testing) Cambios en index.html - geolocalización
|/
* 5dd8e76 inicio del proyecto

diego@DML10 MINGW64 ~/Documents/GIT/testBranch (master)
$
```




Git

Procedimiento Básico de Ramificación y Fusión

Imagina que estas trabajando en un proyecto y tienes un par de confirmaciones

```
MINGW64:/c:/Users/diego/Documents/GIT/testBranchMerge

diego@DML10 MINGW64 ~/Documents/GIT/testBranchMerge
$ git init
Initialized empty Git repository in C:/Users/diego/Documents/GIT/testBranchMerge/.git/

diego@DML10 MINGW64 ~/Documents/GIT/testBranchMerge (master)
$ git add index.html

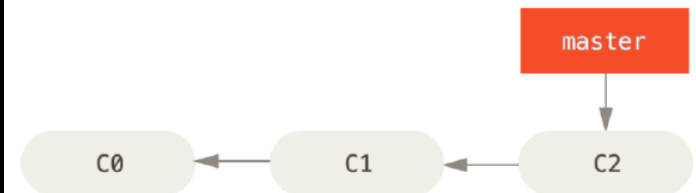
diego@DML10 MINGW64 ~/Documents/GIT/testBranchMerge (master)
$ git commit -m "inicio del proyecto"
[master (root-commit) a337352] inicio del proyecto
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 index.html

diego@DML10 MINGW64 ~/Documents/GIT/testBranchMerge (master)
$ git add index.html

diego@DML10 MINGW64 ~/Documents/GIT/testBranchMerge (master)
$ git commit -m "Modificación de index.html"
[master 4009231] Modificación de index.html
1 file changed, 12 insertions(+)

diego@DML10 MINGW64 ~/Documents/GIT/testBranchMerge (master)
$ git log --oneline
4009231 (HEAD -> master) Modificación de index.html
a337352 inicio del proyecto

diego@DML10 MINGW64 ~/Documents/GIT/testBranchMerge (master)
$
```





Git

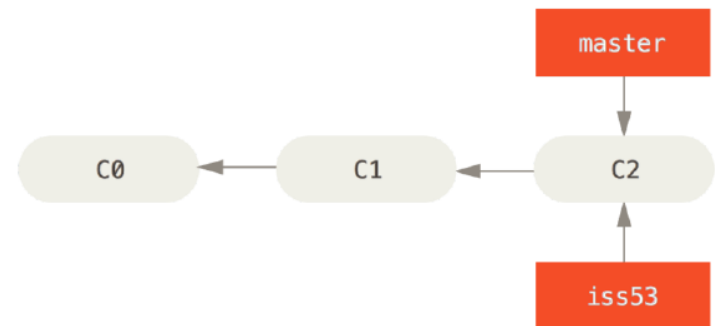
Decides trabajar en el problema #53, según el sistema que tu compañía utiliza para llevar el seguimiento de los problemas. Para crear una nueva rama y saltar a ella, en un solo paso, puedes utilizar el comando **git checkout** con la opción **-b** (-b es un atajo para: **git branch iss53**, luego **git checkout iss53**)

```
MINGW64:/c/Users/diego/Documents/GIT/testBranchMerge

diego@DML10 MINGW64 ~/Documents/GIT/testBranchMerge (master)
$ git checkout -b iss53
Switched to a new branch 'iss53'

diego@DML10 MINGW64 ~/Documents/GIT/testBranchMerge (iss53)
$ git log --oneline
4009231 (HEAD -> iss53, master) Modificación de index.html
a337352 inicio del proyecto

diego@DML10 MINGW64 ~/Documents/GIT/testBranchMerge (iss53)
$
```





Git

Trabajas en el sitio web y haces algunas confirmaciones de cambios (commits). Con ello avanzas la rama iss53, que es la que tienes activada (checked out) en este momento (es decir, a la que apunta HEAD):

MINGW64:/c:/Users/diego/Documents/GIT/testBranchMerge

```
diego@DML10 MINGW64 ~/Documents/GIT/testBranchMerge (master)
$ git checkout -b iss53
Switched to a new branch 'iss53'

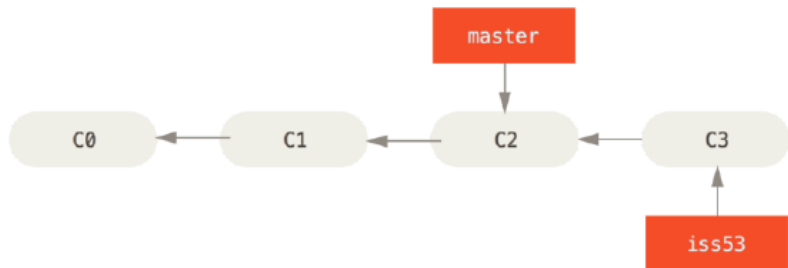
diego@DML10 MINGW64 ~/Documents/GIT/testBranchMerge (iss53)
$ git log --oneline
4009231 (HEAD -> iss53, master) Modificación de index.html
a337352 inicio del proyecto

diego@DML10 MINGW64 ~/Documents/GIT/testBranchMerge (iss53)
$ git add index.html

diego@DML10 MINGW64 ~/Documents/GIT/testBranchMerge (iss53)
$ git commit -m "index.html con Lorem ipsum"
[iss53 0504864] index.html con Lorem ipsum
1 file changed, 7 insertions(+), 2 deletions(-)

diego@DML10 MINGW64 ~/Documents/GIT/testBranchMerge (iss53)
$ git log --oneline --decorate --graph --all
* 0504864 (HEAD -> iss53) index.html con Lorem ipsum
* 4009231 (master) Modificación de index.html
* a337352 inicio del proyecto

diego@DML10 MINGW64 ~/Documents/GIT/testBranchMerge (iss53)
$
```





Git

Puedes realizar las pruebas oportunas, asegurarte de que la solución es correcta, e incorporar los cambios a la rama master para ponerlos en producción.

Para fusionar la rama iss53 simplemente activa (checkout) la rama donde deseas fusionar y lanza el comando **git merge**



MINGW64:/c:/Users/diego/Documents/GIT/testBranchMerge

```
diego@DML10 MINGW64 ~/Documents/GIT/testBranchMerge (iss53)
$ git checkout master
Switched to branch 'master'

diego@DML10 MINGW64 ~/Documents/GIT/testBranchMerge (master)
$ git merge iss53
Updating 4009231..0504864
Fast-forward
 index.html | 9 ++++++--
 1 file changed, 7 insertions(+), 2 deletions(-)

diego@DML10 MINGW64 ~/Documents/GIT/testBranchMerge (master)
$ git log --oneline --decorate --graph --all
* 0504864 (HEAD -> master, iss53) index.html con Lorem ipsum
* 4009231 Modificación de index.html
* a337352 inicio del proyecto

diego@DML10 MINGW64 ~/Documents/GIT/testBranchMerge (master)
$ git branch -d iss53
Deleted branch iss53 (was 0504864).

diego@DML10 MINGW64 ~/Documents/GIT/testBranchMerge (master)
$ git log --oneline --decorate --graph --all
* 0504864 (HEAD -> master) index.html con Lorem ipsum
* 4009231 Modificación de index.html
* a337352 inicio del proyecto

diego@DML10 MINGW64 ~/Documents/GIT/testBranchMerge (master)
$
```



Git

La frase “Fast forward” (“Avance rápido”) que aparece en la salida del comando. Git ha movido el apuntador hacia adelante, ya que la confirmación apuntada en la rama donde has fusionado estaba directamente arriba respecto a la confirmación actual.

Dicho de otro modo: cuando intentas fusionar una confirmación con otra confirmación accesible siguiendo directamente el historial de la primera; Git simplifica las cosas avanzando el puntero, ya que no hay ningún otro trabajo divergente a fusionar.

Ahora, los cambios realizados están ya en la instantánea de la confirmación (commit) apuntada por la rama master.

Es importante borrar la rama iss53, ya que no la vamos a necesitar más, puesto que apunta exactamente al mismo sitio que la rama master.

Esto lo puedes hacer con la opción **-d** del comando **git branch**



Git

Glosario de términos:

- **Branch o Rama:** Es un apuntador móvil que apunta a las confirmaciones. La rama por defecto es la “master” y se crea con la primera confirmación de cambios que se realice.
- **Control de versiones:** Principal característica de GIT, que ofrece herramientas para poder gestionar cada una de las etapas y versiones por las que va transitando un proyecto de desarrollo. Favorece el desarrollo colaborativo.
- **Desarrollo colaborativo:** Modelo basado en la disponibilidad pública del código y la comunicación vía Internet. Proporciona herramientas para que un gran número de individuos puedan hacer desarrollos en conjunto.
- **Fork de un proyecto:** Se realiza para poder colaborar en proyectos cuando no se tienen permisos de escritura (push access). GitHub intentará hacer una copia entera del proyecto en nuestra cuenta de usuario, de esa manera, se podrán realizar cambios.



Git

Glosario de términos:

- **GIT:** Software de control de versiones diseñado por Linus Torvalds. Su propósito es llevar registro de los cambios en archivos de computadora y coordinar el trabajo que varias personas realizan sobre archivos compartidos.
- **GitHub:** Host de almacenamiento de repositorios GIT más grande y punto central de colaboración de millones de proyectos y desarrolladores. Un gran porcentaje de repositorios GIT están alojados en GitHub, y muchos proyectos open-source lo usan como almacenamiento, registro de problemas (issue tracking), revisión de código, y otros.
- **Grafo:** En matemáticas y ciencias de la computación, es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos que permiten representar relaciones binarias entre elementos de un conjunto.



Git

Glosario de términos:

- **Hooks:** Es una herramienta de GitHub que permite disparar peticiones HTTP POST a un dominio especificado al ocurrir ciertos eventos en el repositorio (por ejemplo: push).
- **Repositorio de GIT:** es la carpeta .git / dentro de un proyecto. Rastrea todos los cambios realizados y crea un historial a lo largo del tiempo.