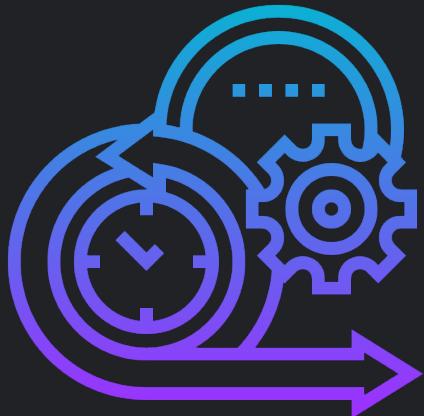


Buenas prácticas y agilidad

Control de versiones y Git



Josu Gorostegui

¿Quiénes impartimos este módulo?

- **Josu Gorostegui** Software Architect. ARK Team
- **Juan Ignacio Forcén** Team leader. Verificación Documental

Módulo Agile, buenas prácticas de desarrollo y conocimientos transversales.

¿Por qué?

Para conocer **prácticas transversales** del desarrollo software

Se profundizará en:

- Cuestiones transversales, git, control de versiones, ...
- Generación de Código con Inteligencia Artificial:
- Frameworks de Trabajo
- Cómo trabajar en equipo, qué me motiva...



Josu Gorostegui



<https://linkedin.com/in/josugorostegui>



<https://jgorostegui.github.io/>

Ingeniero de software y ML

Formado en Ingeniería de Telecomunicaciones, inició su trayectoria en Donostia dedicándose a la investigación en el ámbito de imagen y vídeo.

Motivado por el avance de la inteligencia artificial, se trasladó a Pamplona para especializarse en biometría y reconocimiento facial. Posteriormente, asumió roles de Product Manager en biometría facial y de voz.

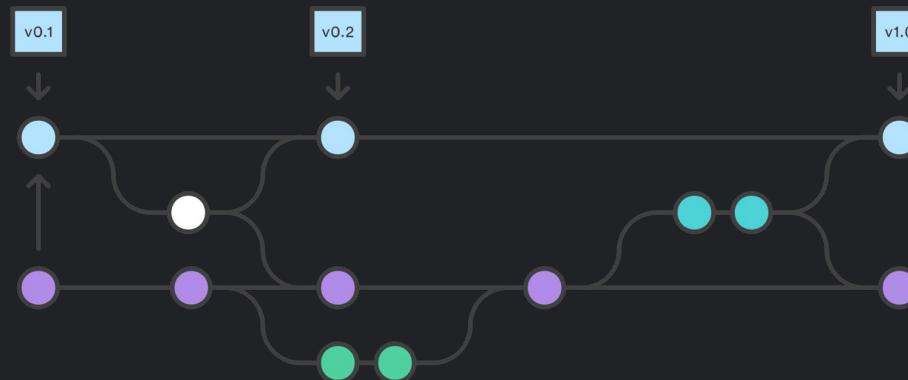
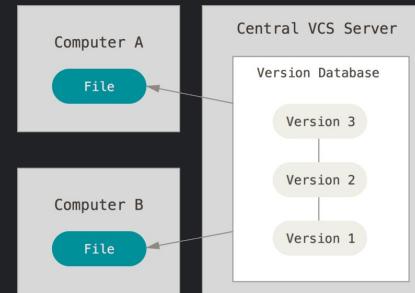
Actualmente, desempeña funciones como Arquitecto de Software en Tecnología.

- ▶ Introducción al Control de Versiones
- ▶ Qué es Git
- ▶ Fundamentos de Git
- ▶ Flujo de trabajo básico con Git
- ▶ Trabajar con Ramas
- ▶ Trabajar con repositorios remotos
- ▶ Buenas prácticas en Git

- El control de versiones es como una máquina del tiempo para nuestro código.
- A menudo es denominado "control de código fuente", es una metodología que rastrea y administra los cambios en el código de software a lo largo del tiempo.
- Es necesario que nuestro código mantenga un histórico de cambios para poder trabajar en un entorno **colaborativo**.
- Un sistema de control de versiones le permite realizar un **seguimiento** de los cambios en sus documentos.
- Permite ir a una **versión anterior** del código anterior.

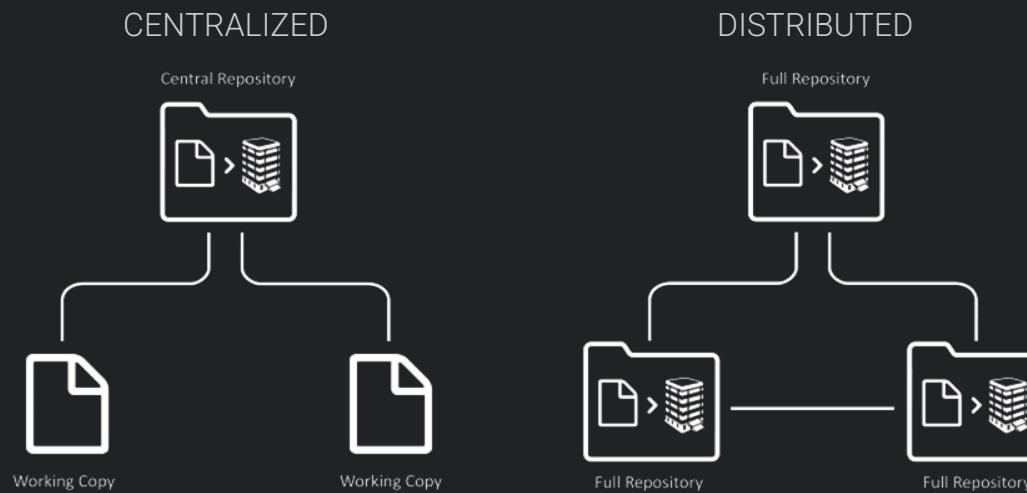
¿Por qué es necesario?

1. Protección del Código Fuente
2. Trabajo Colaborativo
3. Historial Completo
4. Creación de Ramas y Fusiones
5. Trazabilidad
6. Reducción de Errores y Problemas



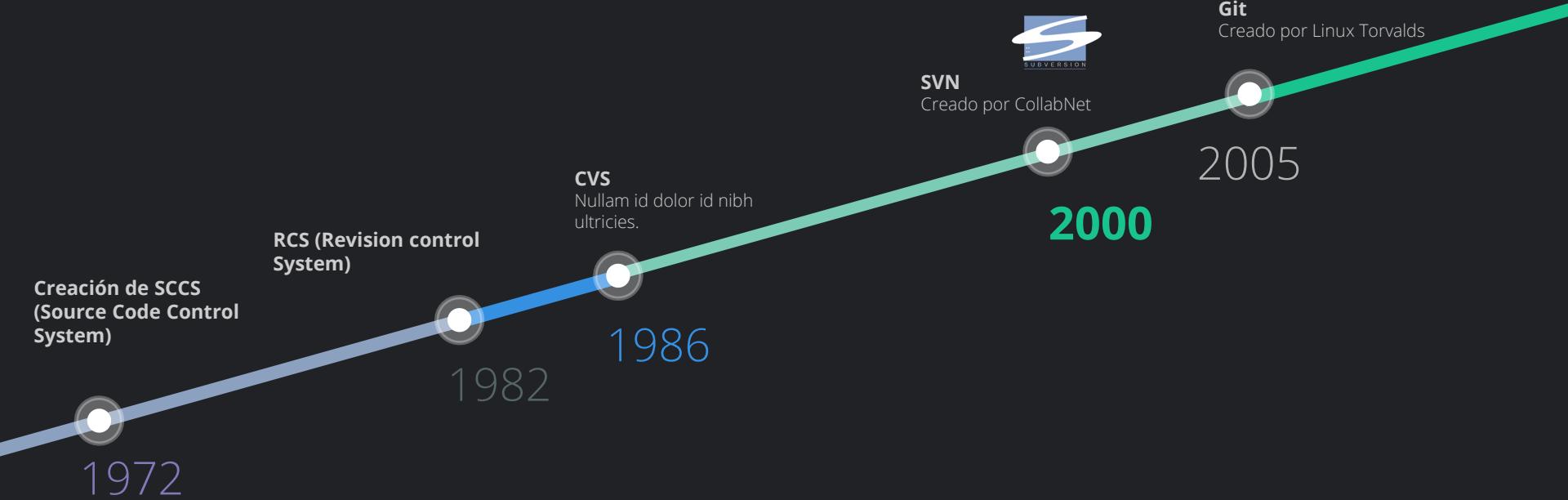
Central: Se tiene una copia del proyecto en un servidor central y los integrantes del equipo realizan una actualización de estos archivos de acuerdo a los cambios que realizan.

Distribuido: Los desarrolladores trabajan en su repositorio local y los cambios se actualizan entre repositorios.



▶ Introducción al control de versiones

Historia



RCS (Revision Control System)

Fue uno de los primeros sistemas de control de versiones. Guardaba conjuntos de cambios en archivos especiales.

Ventajas

- Mejor que copiar carpetas manualmente.

Desventajas

- Solo funcionaba localmente.
- Si dos desarrolladores intentaban hacer cambios al mismo tiempo, surgían complicaciones.

Los sistemas centralizados como **CVS** y posteriormente **Subversion (SVN)**

Características

- Todos los cambios se registran en un servidor central.
- Los desarrolladores obtienen la última versión del servidor y envían sus cambios.

Ventajas

- Facilita la colaboración.
- Control centralizado.

Desventajas

- Si el servidor falla y no hay copias de seguridad, se pierde todo.
- Todos los desarrolladores dependen de la accesibilidad del servidor central.

Dentro de sistemas distribuidos se encuentran Git y Mercurial

Características

Cada desarrollador tiene una copia local completa del historial del proyecto.

Los cambios se pueden compartir entre diferentes repositorios.

Ventajas

- Funciona sin conexión.
- Los repositorios pueden ser respaldados por múltiples personas, reduciendo el riesgo de pérdida.
- Permite colaboraciones más flexibles.

Desventajas

Curva de aprendizaje inicialmente más empinada.

¿Qué es Git?

Git es un sistema de control de versiones (VCS) y gestor de código fuente (SCM).

Principales características:

- Software libre y gratuito.
- Control de versiones distribuido: cada clon es un repositorio completo.
- Diseñado para manejar desde pequeños a grandes proyectos con rapidez y eficiencia.
- Ampliamente adoptado en la industria y respaldado por una amplia comunidad.
- GitHub, una plataforma popular, ofrece un servicio "hosted" basado en Git.
- Documentación disponible oficial en múltiples idiomas



Orígenes

- El desarrollo del kernel de Linux: un proyecto de código abierto de gran envergadura.
- Hasta 2002: cambios mediante parches y archivos.
- 2002-2005: uso de BitKeeper, un DVCS propietario.

Nacimiento de Git

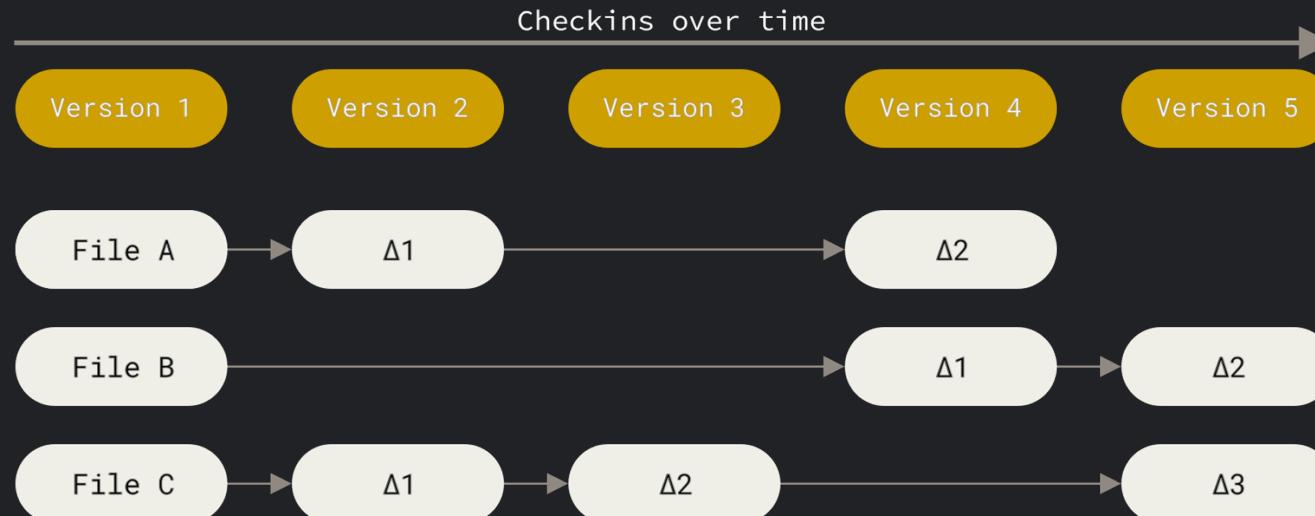
Creado por Linus Torvalds con objetivos claros:

- Rapidez.
- Diseño sencillo.
- Soporte para desarrollo no lineal.
- Totalmente distribuido.
- Eficiencia en proyectos grandes.
- Desde 2005: Evolución y maduración, manteniendo sus características fundamentales.



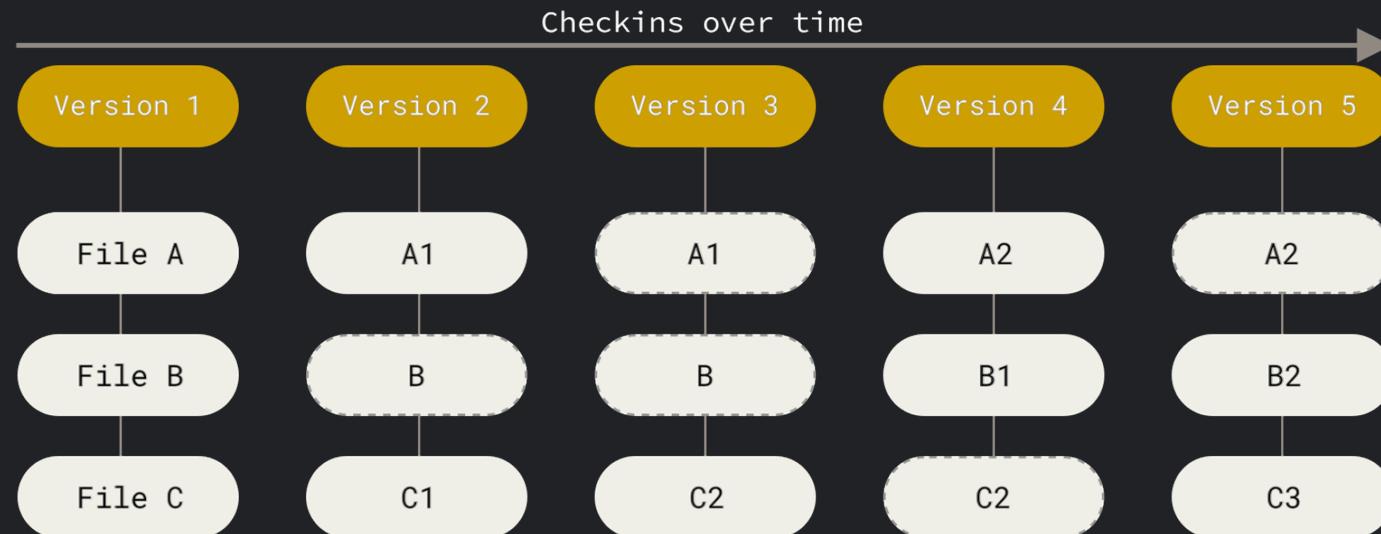
Copias instantáneas, no diferencias

- Otros sistemas almacenan la información como un conjunto de archivos y modificaciones de ellos a través del tiempo.



Copias instantáneas, no diferencias

- Git maneja sus datos como un conjunto de copias instantáneas de un sistema de archivos miniature.



Git Internals

Git, es un sistema de archivos direccional por contenido con una interfaz de usuario de sistema de control de versiones (VCS) construida sobre él.

Esto significa que Git, es un almacén de datos clave-valor que permite insertar cualquier tipo de contenido y recuperar ese contenido más tarde utilizando una clave única proporcionada por Git.

- Comandos Plumbing: Bajo nivel, utilizados para operaciones internas y scripts.
 - Ejemplo: `git hash-object`, `git cat-file`
- Comandos Porcelain: Interfaz de usuario amigable para las operaciones diarias.
 - Ejemplo: `git add`, `git commit`, `git branch`

El directorio `.git`

Esta carpeta se crea nada más hacer `git init`, donde se encuentra todo lo almacenado/manipulado por git.

```
$ git init
$ ls -F1 .git/
config  description  HEAD  hooks/  info/  objects/  refs/
```

Git Internals

El directorio .git

Esta carpeta se crea nada más hacer `git init`, donde se encuentra todo lo almacenado/manipulado por git.

```
$ git init  
$ ls -F1 .git/  
config  description  HEAD  hooks/  info/  objects/  refs/
```

git y GitHub



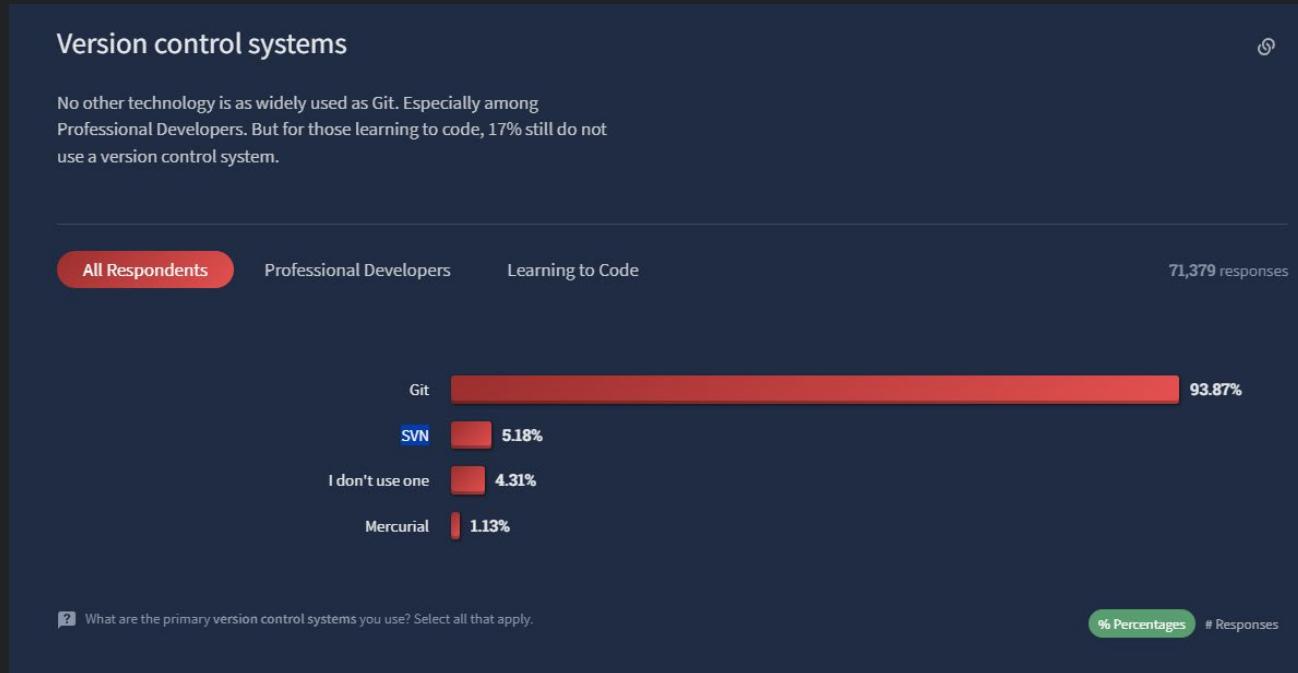
Herramienta de control
de versiones



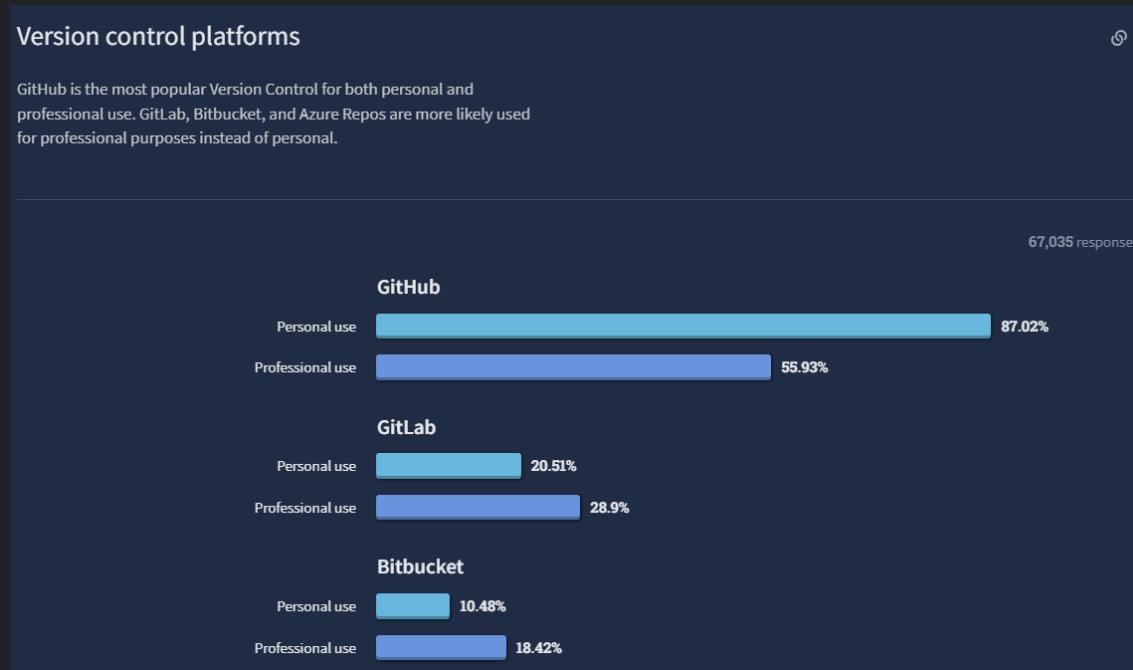
GitHub

Servicio de hosting para
proyectos de Git

git y GitHub



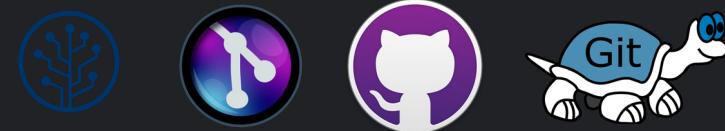
git y GitHub



Diferentes plataformas y herramientas, algunas con GUI stand-alone o integradas en el editor.

Plataformas

- GitHub permite trabajar con las herramientas de CI/CD de su elección, pero deberá integrarlas usted mismo. Los usuarios de GitHub suelen trabajar con un programa de CI de terceros
- GitLab tiene integrados flujos de trabajo de integración continua/entrega continua (CI/CD) y DevOps.



En Linux/Mac OS:

- Ya viene instalado por defecto, aunque no última versión
 - Ubuntu: `apt-get install git`
 - Mac OS: `brew install git`
- Última versión disponible en: <https://git-scm.com/downloads>

En Windows:

- Se descarga desde la web oficial: <https://git-scm.com/download/win>

Funciona exactamente igual en todos los sistemas operativos.

Git

- El libro es gratuito y está traducido a todos los idiomas
- En la web también hay documentación y otros recursos

The screenshot shows the official Git website at <https://git-scm.com>. The page features a large banner with the Git logo and the tagline "git --distributed-is-the-new-centralized". Below the banner, there's a search bar and a diagram illustrating a distributed version control system where multiple repositories are interconnected. The main content area includes sections for "About", "Documentation", "Downloads", and "Community". A prominent "Latest source Release" section highlights "2.42.0" with a "Download for Windows" button. At the bottom, links are provided for "Windows GUIs", "Tarballs", "Mac Build", and "Source Code".

Clientes GUI

- Fuente: <https://git-scm.com/downloads/guis>
- Más populares:
 - Clientes desde VS Code
 - GitHub desktop
 - GitKraken
 - SourceTree

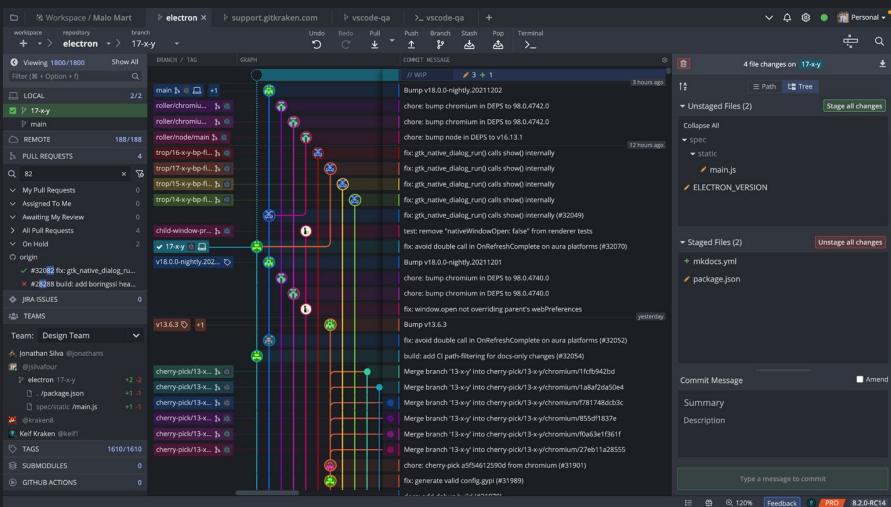
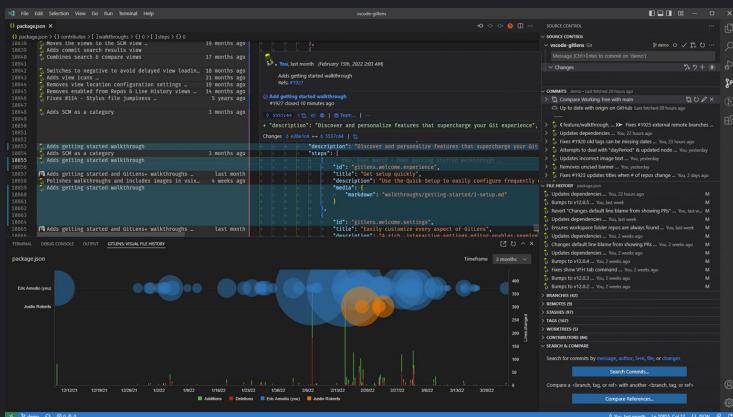
The screenshot shows the official Git website's "GUI Clients" section. At the top, there's a navigation bar with links for About, Documentation, Downloads, and Community. Below that, a sidebar lists "Downloads" and "GUI Clients". A note about the Pro Git book is present. The main content area features a heading "25 Linux GUIs are shown below" followed by five examples:

- GitKraken**: Platforms: Linux, Mac, Windows. Price: Free / \$59+/user annually. License: Proprietary.
- Magit**: Platforms: Linux, Mac, Windows. Price: Free. License: GNU GPL.
- SmartGit**: Platforms: Linux, Mac, Windows. Price: Free for non-commercial use / \$59/user annually. License: Proprietary.
- MeGit (based on EGit)**: Platforms: Linux, Mac, Windows. Price: Free. License: EPL2.0.
- SourceTree**: Platforms: All. Price: \$49.99. License: Proprietary.

Fundamentos de Git

GitHub, GitLab y Bitbucket

Herramientas con GUI para trabajar con Git



Referencias

- <https://courses.cs.washington.edu/courses/cse390a/14wi/bash.html>

Bash Shell Reference (manual)

category	command	description
basic shell	clear	clear all previous commands' output text from the terminal
	exit (or logout)	quits the shell
	alias, unalias	give a pseudonym to another command (you may need to enclose the command in quotes if it contains spaces or operators)
	history	show a list of all past commands you have typed into this shell
directories	ls	list files in a directory
	pwd	displays the shell's current working directory
	cd	changes the shell's working directory to the given directory; can be a relative or absolute path
	mkdir	creates a new directory with the given name
	rmdir	removes the directory with the given name (the directory must be empty)
file operations	cp	copies a file/directory
	mv	moves (or renames) a file/directory
	rm	deletes a file
	touch	update the last-modified time of a file (or create an empty file)
file examination	cat	output the contents of a file
	more (or less)	output the contents of a file, one page at a time
	head, tail	output the beginning or ending of a file
	wc	output a count of the number of characters, lines, words, etc. in a file
	du	report disk space used by a file/directory
	diff	output differences between two files

Terminología general

Commit

- Como sustantivo, un punto en la historia de Git
- Como verbo: La acción de crear un nuevo snapshot en un proyecto Git, y el hecho de añadirlo a la historia.

Repositorio

- El directorio que contiene el proyecto git, así como sus archivos.
- Si es un repositorio, tendrá una carpeta oculta con algunos archivos llamada .git

SHA

- Un SHA es básicamente un número de identificación para cada confirmación.

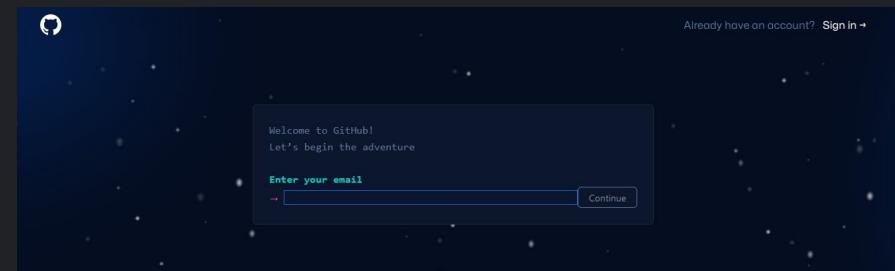
SSH

- Protocolo de seguridad que permite conectar y autenticarse con servicios y servidores remotos.

Configurando git

Para configurar git es necesario realizar los siguientes pasos:

1. Crearse una cuenta en [GitHub](#).
2. Generar claves SSH, siguiendo el [tutorial de github](#).
3. Añadir/vincular las llaves SSH a la cuenta de GitHub
4. Configurar git



Comandos

```
$ ssh-keygen -t ed25519 -C "tunombre@gmail.com"
```

```
$ cat ~/.ssh/id_ed25519.pub
```

Configurando git

Para configurar git es necesario realizar los siguientes pasos:

1. Crearse una cuenta en [GitHub](#).
- 2. Generar claves SSH, siguiendo el [tutorial de github](#).**
3. Añadir/vincular las llaves SSH a la cuenta de GitHub
4. Configurar git

Comandos

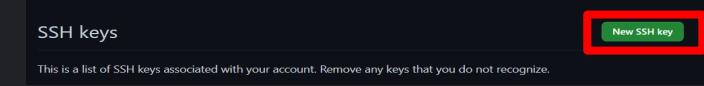
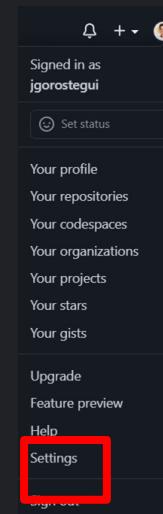
```
$ ssh-keygen -t ed25519 -C  
"tunombre@gmail.com"  
$ cat ~/.ssh/id_ed25519.pub
```



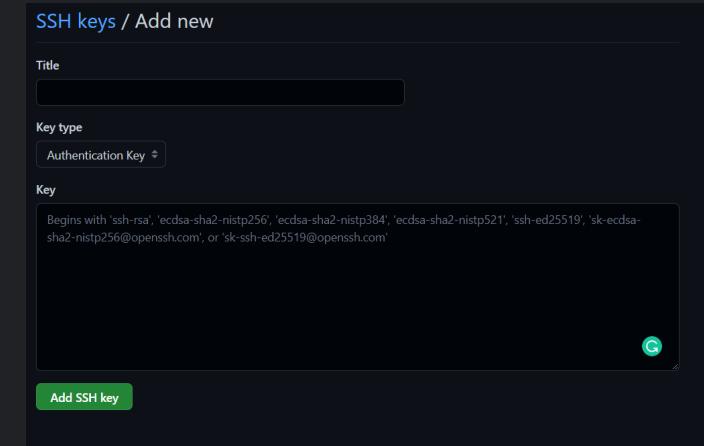
Configurando git

Para configurar git es necesario realizar los siguientes pasos:

1. Crearse una cuenta en [GitHub](#).
2. Generar claves SSH, siguiendo el [tutorial de github](#).
- 3. Añadir/vincular las llaves SSH a la cuenta de GitHub**
4. Configurar git



This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.



SSH keys / Add new

Title

Key type

Key
Begins with 'ssh-rsa', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521', 'ssh-ed25519', 'sk-ecdsa-sha2-nistp256@openssh.com', or 'sk-ssh-ed25519@openssh.com'

Add SSH key

Configurando git

Para configurar git es necesario realizar los siguientes pasos:

1. Crearse una cuenta en [GitHub](#).
2. Generar claves SSH, siguiendo el [tutorial de github](#).
3. Añadir/vincular las llaves SSH a la cuenta de GitHub
- 4. Configurar git**



```
# Git configuration
# Sets up Git with your name
git config --global user.name "Josu Gorostegui"
# Sets up Git with your email
git config --global user.email "gorosjosu@gmail.com"
```

```
$ cat .gitconfig
[user]
email = gorosjosu@gmail.com
name = Josu Gorostegui
```

git init

Iniciarizar repositorio (utilizando `git init`)

```
$ git init
```

- Inicializa un repositorio Git vacío.
- Crea una nueva carpeta `.git` en el directorio actual.
- Sólo se necesita hacer una vez por repositorio.

git add

Añadiendo los cambios con `git add`

```
$ git add README.md
```

- Puedes agregar archivos específicos o todos los cambios.
- `git add .` agrega todos los cambios del directorio actual.
- Prepara los cambios para el próximo "commit".

git status

Ver el estado del repo (utilizando `git status`)

```
$ git status
```

- Muestra archivos modificados, agregados o eliminados.
- Indica si los archivos están listos para ser "commit".
- Esencial para el flujo de trabajo y se utiliza frecuentemente.

git commit

Guardando Cambios con `git commit`

```
$ git commit -m "Añadiendo el archivo README.md"
```

- Crea una instantánea del proyecto.
- El parámetro `-m` permite añadir un mensaje descriptivo.
- Es fundamental para mantener un registro del progreso.

git commit --amend

Permite modificar el último commit realizado.

- ¿Como se usa?

```
$ git commit --amend -m "nuevo mensaje de commit" # Cambia solo el mensaje
```

```
$ git commit --amend # cambios en el último commit
```

- Corregir un error tipográfico en el mensaje del commit.
- Añadir archivos olvidados al commit anterior.
- Realizar pequeños cambios adicionales antes de compartir el commit.



¡CUIDADO!

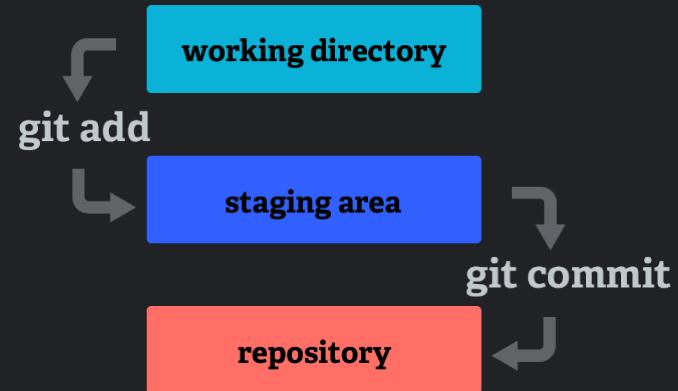
- **¡No modifiques commits que ya hayas enviado a un repositorio remoto!** Esto puede causar problemas de sincronización con otros colaboradores.
- Úsalo con cuidado y solo en commits locales.

Añadiendo archivos

Cuando se añaden (`git add`) archivos estos pasan a estar en un área llamada *staging area* o *índice*.

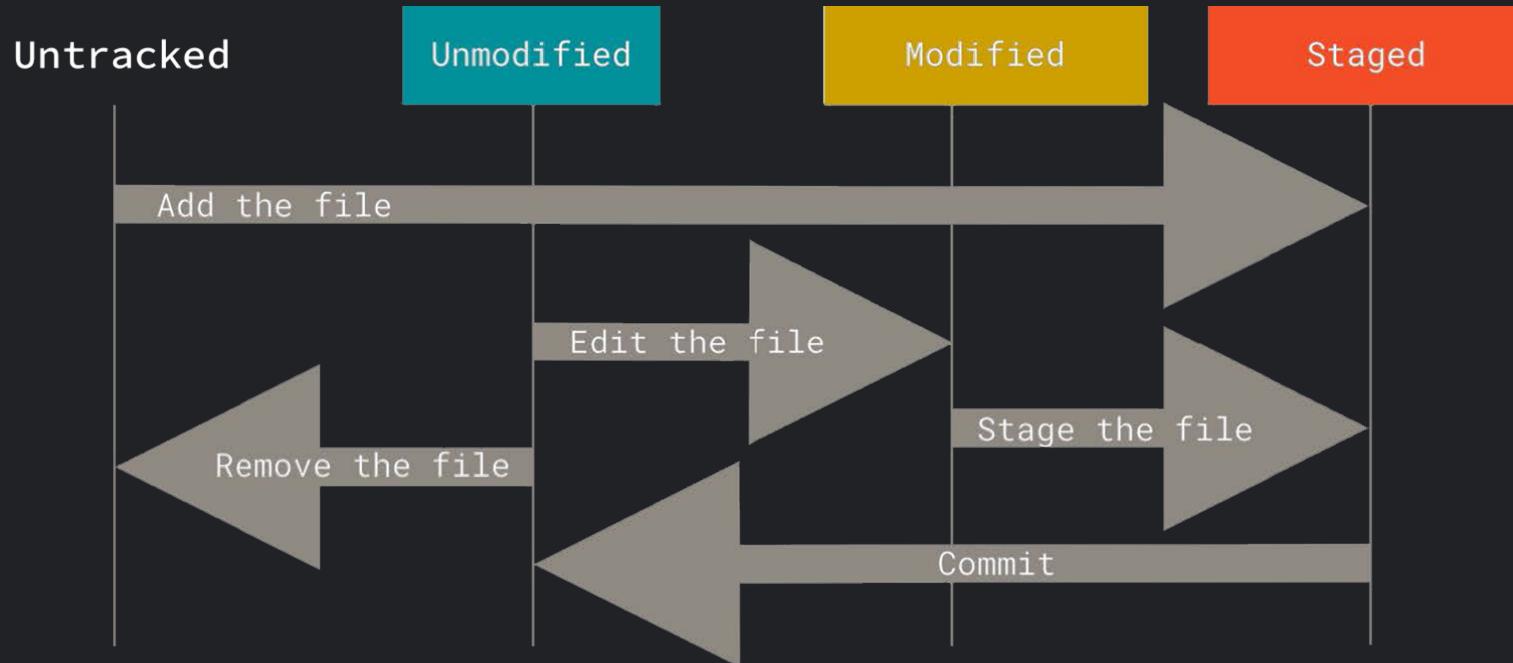
Los cambios no se registran hasta que no se ejecuta `git commit`

El commit representa la fotografía del proyecto en ese momento.



Flujo de trabajo básico con Git

Ciclo de vida del estado de los archivos



git log

Visualizando el historial con `git log`

```
$ git log
```

- Muestra los "commits" en orden cronológico inverso.
- Incluye autor, fecha y mensaje de cada "commit".
- Útil para rastrear cambios y la evolución del proyecto.

Para limitar la salida del historial

```
$ git log --since=2.weeks
```

git log (2)

`git log` ofrece opciones para personalizar la visualización del historial de commits.

Formato:

`--oneline`: Muestra cada commit en una sola línea, ideal para una vista rápida.

`--graph`: Visualiza el historial como un gráfico de ramas y fusiones.

`--pretty=format:"%h - %an, %ar : %s"`: Personaliza la información mostrada para cada commit (hash, autor, fecha, mensaje).

Ejemplos

`git log --oneline --graph`: Vista compacta y gráfica del historial.

`git log --author="Juan Pérez" --since="2023-01-01"`:

Muestra los commits de Juan Pérez desde el 1 de enero de 2023.

```
$ git log --pretty=format:"%h %s" --graph
* 2d3acf9 Ignore errors from SIGCHLD on trap
* 5e3ee11 Merge branch 'master' of https://github.com/dustin
  \
  | * 420eac9 Add method for getting the current branch
  * | 30e367c Timeout code and tests
  * | 5a09431 Add timeout protection to grit
  * | e1193f8 Support for heads with slashes in them
  /
* d6016bc Require time for xschema
* 11d191e Merge branch 'defunkt' into local
```

Crear el primer repositorio

Se va a crear repositorio y se utilizarán los comandos

1. Crear un directorio (utilizando `mkdir`)
2. Inicializar repositorio (utilizando `git init`)
3. Ver el estado del repo (utilizando `git status`)

Se puede ver como existen ficheros ocultos tras la inicialización.

git diff

Para visualizar los cambios que no añadidos se utiliza: **git diff**

```
$ git diff
```

git diff muestra las diferencias entre lo que tienes en tu directorio de trabajo y lo que está guardado en tu área de preparación (staging area) o entre dos commits, ramas, etc.



Primer commit

Dentro del repo (creado en la diapositiva anterior), se va a añadir un archivo y realizaremos el primer commit.

Para esto es necesario recordar:

- Añadir un archivo con:
 - `git add <file>`
- Para añadir todos los archivos presentes:
 - `git add .`
- Realizar el commit con:
 - `git commit -m <message>`

git tag

Marcando puntos importantes de nuestro historial con `git tag`

\$ `git tag v1.0`

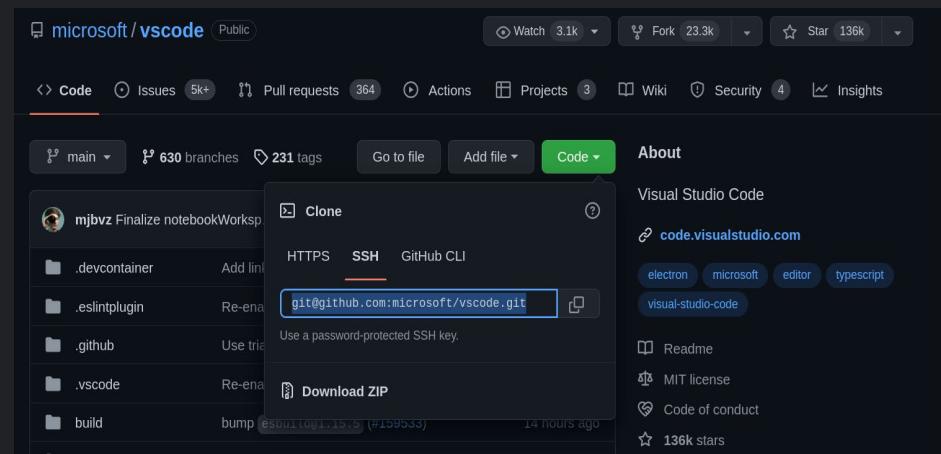
- Ideal para marcar versiones.
- Las etiquetas pueden ser ligeras o anotadas.
- Las etiquetas anotadas incluyen información adicional y se recomiendan para lanzamientos.

```
$ git tag  
v0.1  
v1.3  
$ git tag -l 'v1.8.5*'  
v1.8.5  
v1.8.5-rc0  
v1.8.5-rc1
```

Clonar y Fork

En GitHub existen más de 100M de repositorios, es la forma colaborativa de trabajar en el mundo del software.

- **Clonar:** Crea una copia del repositorio en la máquina local. Simplemente es el proceso de descargar un proyecto de GitHub desde un repositorio en línea a su máquina local usando Git.
- **Fork:** Hace una copia de un repositorio de GitHub para ser usado como base para un nuevo proyecto, pudiendo enviar los cambios al repositorio original y/o realizar unos cambios de forma independiente. Crea una copia del repositorio original (repositorio ascendente), pero el repositorio se copia en su cuenta de GitHub, no en su máquina local.



Branch y Tag

Branch o rama es una *línea de desarrollo*.

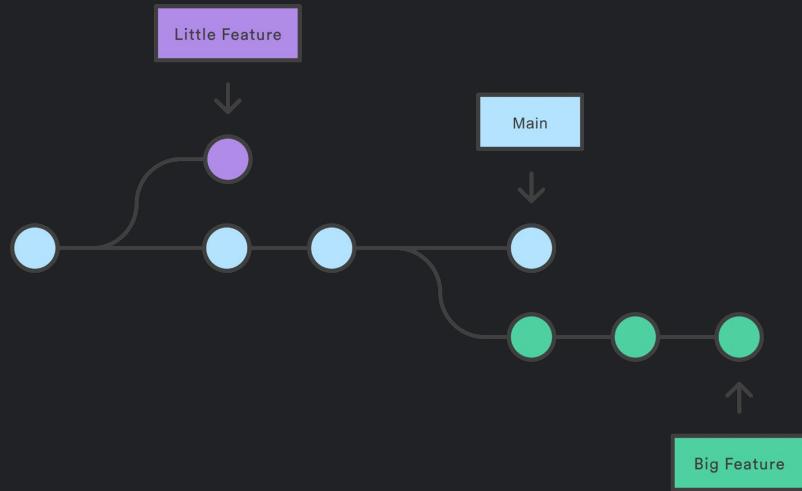
Esta línea nueva permite continuar sin alterar la línea principal.

Un repositorio git tiene las ramas main o master por defecto cuando se crea el repositorio.

En un entorno colaborativo, es habitual que varios desarrolladores compartan y trabajen sobre el mismo código fuente.

Algunos desarrolladores corregirán errores, otros implementarán nuevas features, etc.

Tag o etiqueta son referencias que apuntan a commits concretos en el historial de Git.



git branch

Gestión de Ramas con `git branch`

\$ `git branch nueva_rama`

- `git branch` solo listará todas las ramas.
- Usar `git branch -d nombre_rama` para eliminar una rama.
- Las ramas son cruciales para el desarrollo paralelo y la organización de características o fixes.

git checkout

Cambiar de Estado o Rama con **git checkout**.

El comando git checkout permite cambiar entre diferentes "commits", ramas o incluso restaurar archivos.

```
$ git checkout nombre_rama
```

- Usado tradicionalmente para cambiar entre ramas o "commits".
- **git checkout -- nombre_archivo** puede descartar cambios en ese archivo.
- En versiones más recientes de Git, **git switch** y **git restore** se recomiendan para algunas de estas tareas.

git switch

- Introducido en **Git 2.23**
- Enfocado en operaciones de ramas y más intuitivo para principiantes
- Tiene salvaguardas incorporadas

Comparación con git checkout

- Más claro y específico
- Sólo para ramas
- Más seguro con cambios sin confirmar

Sintaxis:

```
$ git switch <nombre-rama>
```

Ramas en git

Para crear rama y mergear los comandos son los siguientes:

- `git branch`
- `git checkout <featureX>`
- `git branch -d <featureX>`
- `git merge branch <featureX>`
- `git tag`
- `git tag <tagname>`



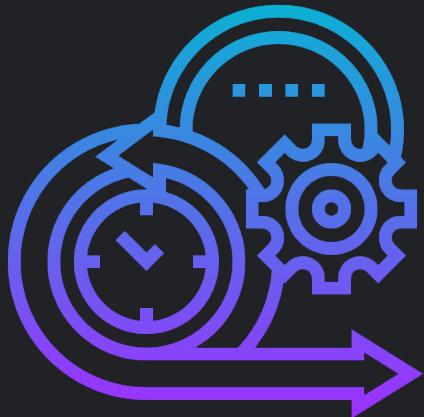
Las tareas a realizar constan de dos partes, para lo cual hay que descargar el .zip desde la [plataforma moddle](#).

1. Operaciones con archivos. Se tiene un .zip comprimido de partida, se descomprime y se realizan ciertas operaciones.
2. La carpeta del ejercicio anterior es el punto de partida del ejercicio 2 y siguientes, que trata realizar ciertas operaciones con Git.

Una vez realizadas las tareas, será necesario subir el resultado comprimido en .zip a la [plataforma moddle](#).

Buenas prácticas y agilidad

Control de versiones y Git



Josu Gorostegui

Preguntas/Respuestas

Utilidades para el terminal

- **tldr:** <https://github.com/tldr-pages/tldr>
- <https://cheat.sh/> (desde el navegador es accesible)
 - Existe aplicación de terminal instalable mediante:
 - `curl -s https://cht.sh/:cht.sh | sudo tee /usr/local/bin/cht.sh && sudo chmod +x /usr/local/bin/cht.sh`

Recursos bash/terminal

- <https://github.com/phyver/GameShell>
- <https://github.com/denysdovhan/learnyoubash>
- Awesome list: <https://github.com/awesome-lists/awesome-bash>
- Errores comunes y cómo evitarlos: <http://mywiki.wooledge.org/BashPitfalls>

Preguntas/Respuestas

Visualizando y añadiendo remotos

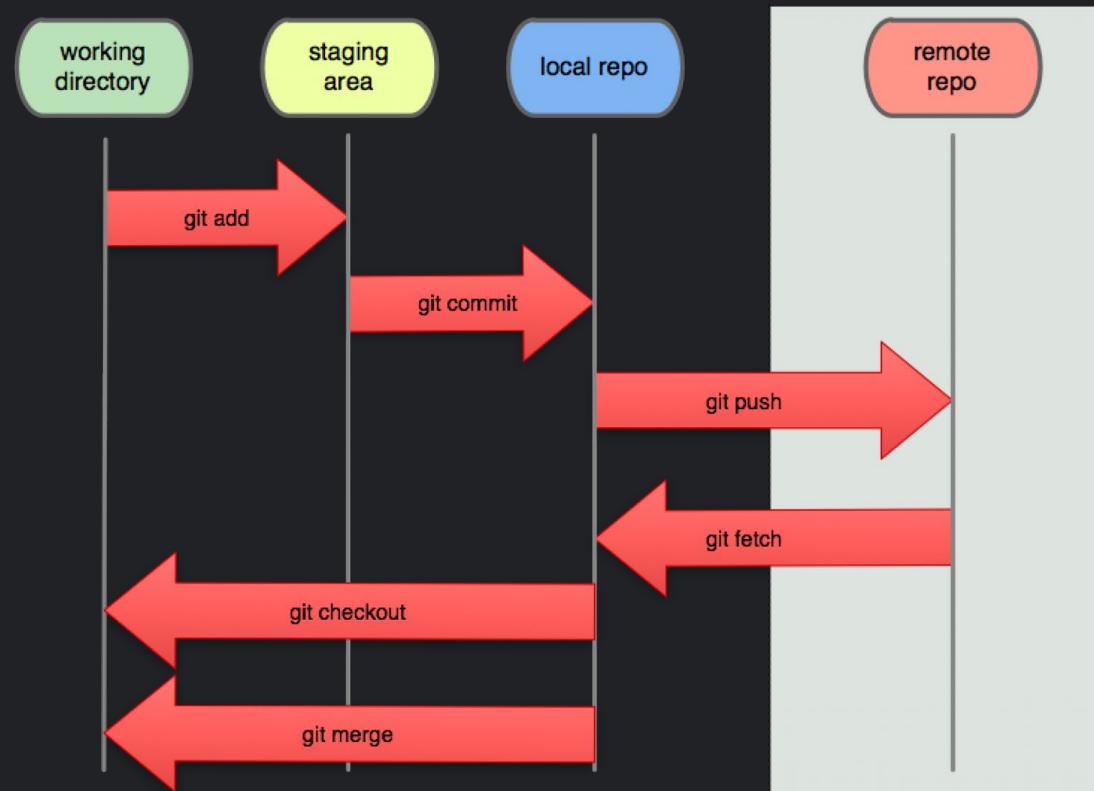
- Si creamos un repositorio en local:
- Usando git init, inicializamos un nuevo repositorio, pero no tiene ningún remoto configurado por defecto.
- Comandos para añadir el remoto

Comandos

- git remote add origin [URL-de-GitHub]
 - Añade un nuevo remoto llamado origin que apunta al repositorio de GitHub especificado.

Flujo de trabajo básico con Git

Trabajando con repositorios



Preguntas/Respuestas

Visualizando y añadiendo los remotos

- Si clonamos un repositorio desde Git
 - Al clonar, automáticamente se configura un "remoto" llamado origin.

Comandos

- `git remote -v`
 - Muestra todos los remotos configurados y sus URLs.

```
ubuntuveridas@ubuntu:~$ git clone git@github.com:jgorostegui.github.io.git
Cloning into 'jgorostegui.github.io'...
remote: Enumerating objects: 610, done.
remote: Counting objects: 100% (610/610), done.
remote: Compressing objects: 100% (351/351), done.
remote: Total 610 (delta 265), reused 522 (delta 188), pack-received objects: 100% (610/610), 2.14 MiB | 4.26 MiB/s, done.
Resolving deltas: 100% (265/265), done.
ubuntuveridas@ubuntu:~$ cd jgorostegui.github.io/
ubuntuveridas@ubuntu:~/jgorostegui.github.io$ git remote -v
origin  git@github.com:jgorostegui/jgorostegui.github.io.git
origin  git@github.com:jgorostegui/jgorostegui.github.io.git
```

git clone

El comando **git clone** se utiliza para copiar un repositorio desde un origen remoto a tu máquina local.

```
$ git clone https://github.com/usuario/repo.git
```

- Crea una copia local del repositorio remoto.
- Incluye todo el historial de cambios del repositorio.
- Por defecto, también configura el origen remoto para futuras interacciones.

git pull

Actualizando tu Repositorio Local con git pull **git pull**

\$ git pull origin master

- Combina automáticamente los cambios descargados.
- Equivalente a hacer **git fetch** seguido de **git merge**.
- Mantén tu repositorio local actualizado regularmente usando git pull.

git remote

Permite gestionar conexiones con repositorios remotos

Comandos principales:

- `git remote add <nombre> <url>`: Añade un nuevo remoto
- `git remote -v`: Lista todos los remotos
- `git remote remove <nombre>`: Elimina un remoto
- `git remote rename <viejo> <nuevo>`: Renombra un remoto

Ejemplo:

- `git remote add origin https://github.com/usuario/repo.git`
- `git remote -v`

git fetch

El comando **git fetch** descarga cambios desde un repositorio remoto, pero no los combina automáticamente en tu rama actual.

\$ git fetch origin

- Solo descarga los datos, no los combina.
- Te permite revisar los cambios antes de fusionar.
- Se usa a menudo en combinación con **git merge**.

git push

El comando **git push** envía los commits realizados en tu rama local al repositorio remoto.

\$ git push origin nombre_rama

- Transfiere tus cambios locales al servidor.
- Si la rama remota no está actualizada, puedes enfrentar conflictos que necesitarás resolver antes de hacer push.
- Usado frecuentemente en colaboraciones y para mantener el repositorio remoto al día.

git stash

Muchas veces, cuando has estado trabajando en una parte de tu proyecto, las cosas se encuentran desordenadas y quieres cambiar de ramas por un momento para trabajar en algo más. El problema es que no quieres hacer un “commit” de un trabajo que va por la mitad, así puedes volver a ese punto más tarde. La respuesta a ese problema es el comando **git stash**.

El guardado rápido toma el desorden de tu directorio de trabajo – que es, tus archivos controlados por la versión modificados y cambios almacenados – y lo guarda en un saco de cambios sin terminar que puedes volver a usar en cualquier momento.

git stash

Almacena temporalmente todos los archivos modificados de los cuales se tiene al menos una versión guardada

```
$ git stash
```

Restaura los archivos guardados más recientemente

```
$ git stash pop
```

```
$ git stash pop stash@{1} # para elegir un stash específico
```

Enumera todos los grupos de cambios que están guardados temporalmente

```
$ git stash list
```

Elimina el grupo de cambios más reciente que se encuentra guardado temporalmente

```
$ git stash drop
```

Si quieres aplicar un stash sin eliminarlo de la lista

```
$ git stash apply
```

git merge

Combinando Ramas con git merge

```
$ git checkout master
```

```
$ git merge nombre_rama
```

- Une las historias de dos ramas.
- Si hay conflictos, Git señalará y necesitarás resolverlos manualmente.
- Es una herramienta esencial en el flujo de trabajo con ramas.

git blame

- Muestra quién modificó cada línea de un archivo
- Permite visualizar los metadatos de autor adjuntos a líneas específicas confirmadas en un archivo.

```
facebook/react Public
Code Issues 808 Pull requests 309 Actions Projects Wiki Security Insights
react / README.md Raw Normal view History
188644 75 lines (45 sloc) 5.08 KB
Updated scripts and config to replace "master" with "main" 16 months ago
Initial public release 10 years ago
# (React)[https://reactjs.org/] ReactDOM[!GitHub license](https://img.shields.io/badge/MIT-blue.svg)](https://github.com/facebook/react/blob/main/LICENSE)
React is a JavaScript library for building user interfaces.

**Declarative:** React makes it painless to create interactive UIs. Design simple views for each state in your application, and React will efficiently update them when your data changes.
**Component-Based:** Build encapsulated components that manage their own state, then compose them to make complex UIs. Since component logic is written in JavaScript, you can reuse it across the stack.
**Learn Once, Write Anywhere:** We don't make assumptions about the rest of your technology stack, so you can develop new features in React without rewriting existing ones.
[Learn how to use React in your project](https://reactjs.org/docs/getting-started.html).

## Installation

React has been designed for gradual adoption from the start, and you can use as little or as much React as you need:
* Use [Online Playgrounds](https://reactjs.org/docs/getting-started.html#online-playgrounds) to get a taste of React.
* [Add React to a Website](https://reactjs.org/docs/add-react-to-a-website.html) as a <script> tag in one minute.
* [Create a New React App](https://reactjs.org/docs/create-a-new-react-app.html) if you're looking for a powerful JavaScript toolchain.

You can use React as a <script> tag from a [CDN](https://reactjs.org/docs/cdn-links.html), or as a "react" package on [npm](https://www.npmjs.com/package/react).
## Documentation

You can find the React documentation [on the website](https://reactjs.org/).

Check out the [Getting Started](https://reactjs.org/docs/getting-started.html) page for a quick overview.

The documentation is divided into several sections:
```

git show

- Muestra información detallada sobre un objeto Git (commit, tag, etc.)
- Útil para revisar cambios específicos

Ejemplo:

- `git show <commit>`

git merge

Flujo básico de trabajo con Git

- Trabajas en el código de una página web y has hecho algunos commits en la rama principal (master).
- De repente, surge la necesidad de corregir un error urgente (hotfix) en producción.

git merge

1. Crear una rama para una nueva funcionalidad

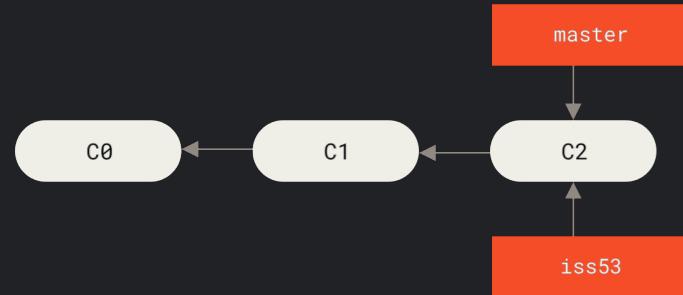
```
$ git checkout -b iss53
```

1. Trabaja en tu nueva funcionalidad

```
$ git commit -a -m 'Crear nuevo footer  
[issue 53]'
```

1. Trabaja en tu nueva funcionalidad

```
$ git checkout master
```



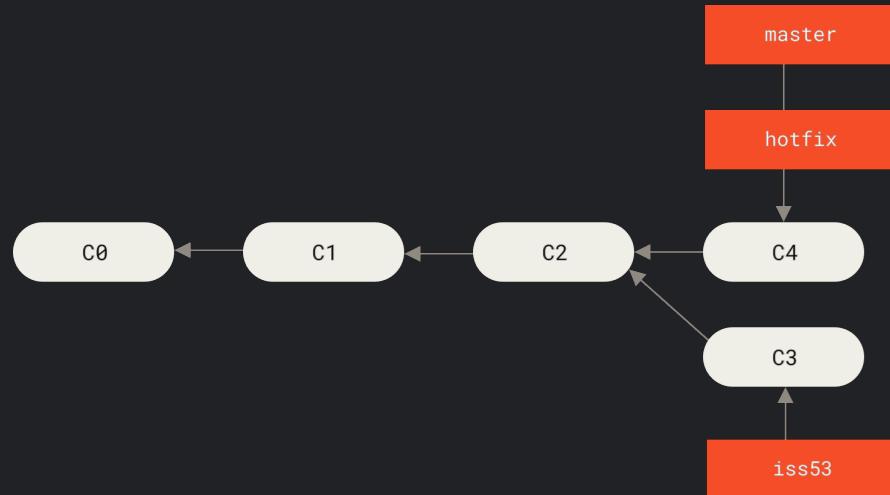
git merge

4. Crear una rama para el Hotfix

```
$ git checkout -b hotfix  
# cambios en los ficheros necesarios  
$ git commit -a -m 'Arreglar dirección de  
correo'
```

4. Fusionar el Hotfix a master

```
$ git checkout master  
$ git merge hotfix
```



git merge

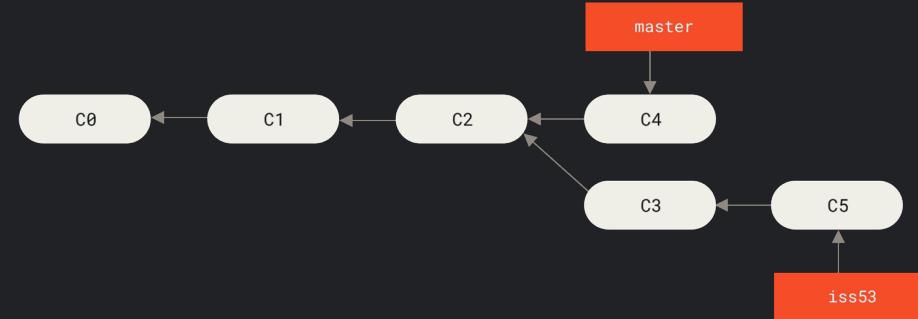
6. Volver al trabajo en la Rama iss53

```
$ git checkout iss53
```

6. Volver al Trabajo en la Rama iss53

```
$ git checkout master
```

```
$ git merge iss53
```



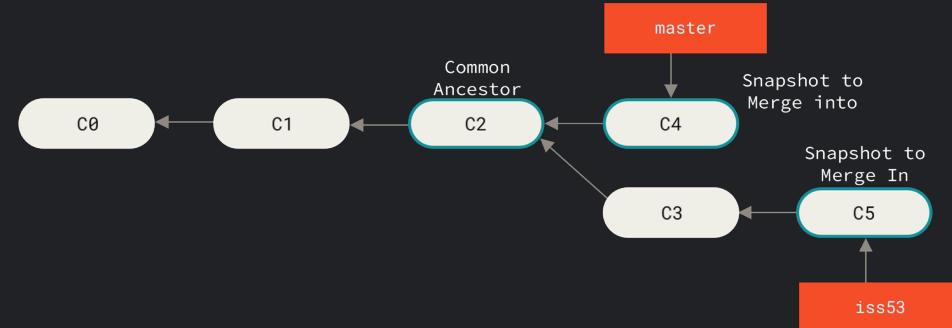
git merge

- Volver al trabajo en la Rama iss53

```
$ git checkout iss53
```

- Volver al Trabajo en la Rama iss53

```
$ git checkout master  
$ git merge iss53
```



git rebase

git rebase es una alternativa a git merge que permite aplicar commits de una rama sobre otra sin crear un merge commit.

Ventajas:

- Historial más limpio y lineal.
- Facilita el seguimiento de cambios en proyectos grandes.

git cherry-pick

- Aplica los cambios de commits específicos a la rama actual
- Útil para mover cambios selectivos entre ramas

Ejemplo

```
$ git cherry-pick <commit-hash>
```

git reset

El comando **git reset** te permite mover o reiniciar el **HEAD** a un estado específico. Puede ser usado para descartar cambios en el área de preparación o incluso descartar commits.

```
$ git reset --hard HEAD~1
```

Usado para deshacer cambios.

git reflog

git reflog registra todos los cambios en las referencias (ramas, HEAD), incluso aquellos que se han perdido con comandos como git reset o git rebase.

Cuándo se utiliza

- Recuperar commits eliminados accidentalmente.
- Volver a un estado anterior después de un rebase problemático.
- Identificar el origen de un cambio inesperado.

```
$ git reflog
1a0b9ec HEAD@{0}: reset: moving to HEAD~3
34ac2e2 HEAD@{1}: commit: Añadido análisis de errores
00fc2a3 HEAD@{2}: commit: Mejora en la interfaz de usuario
b1d216d HEAD@{3}: commit: Corrección de bugs críticos
...
...
```

Resumen de comandos

Comandos básicos

- `git status`: Permite visualizar en que branch me encuentro
- `git log`: Muestra el histórico de commits
- `git add fichero.txt`: Añadir <fichero> o <carpeta>
- `git commit -m "Mi mensaje de commit"`: Crea la instantaneas de los cambios.
- `git diff`: Visualiza los lista de cambios y conflictos
- `git pull`: Sincroniza todos los cambios del repositorio remoto

Ramas

- `git branch nuevarama1`: Crea la rama
- `git checkout nuevarama1`: Se cambia de una rama a otra
- `git merge branch nuevarama1`
- `git tag`
- `git tag nuevotag1`

Comandos de git

Comandos GIT: https://training.github.com/downloads/es_ES/github-git-cheat-sheet/

git cheat sheet



This image is a screenshot of a GitHub cheat sheet for Git commands. It is organized into five main sections: Initialization, Commits, Change Review, Branch and Rebase, and Advanced. Each section contains a list of Git commands with their descriptions. The background is dark gray, and the text is white.

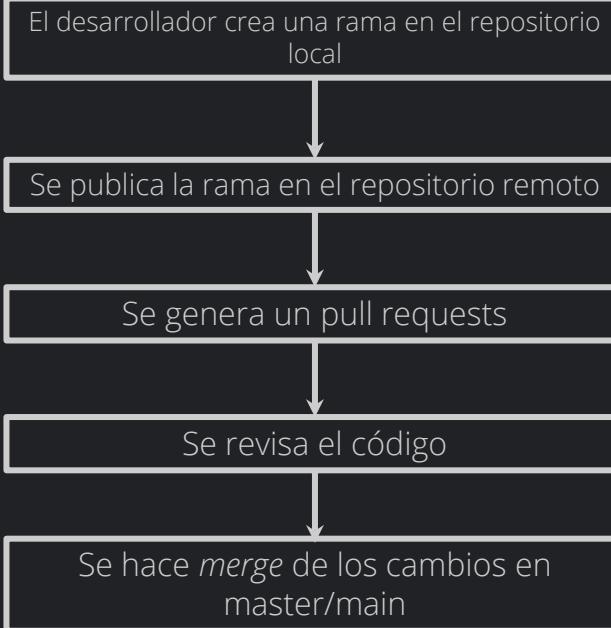
Initialization	Commits	Change Review	Branch and Rebase	Advanced
<pre>\$ git init <directory></pre> <p>Creates new repo in specified directory</p> <pre>\$ git clone curl</pre> <p>Copies repo from specified url</p> <pre>\$ git config user.name <user_name></pre> <p>Sets username for commits in current repo</p> <pre>Use --global to apply it globally</pre> <pre>\$ git config user.email <user_email></pre> <p>Sets email for commit in current repo</p> <pre>Use --global to apply it globally</pre> <pre>\$ git config color.ui auto</pre> <p>Enables helpful colorization of command line output<pre>\$ git config --global --edit</pre><p>Opens the global configuration file in text editor for manual editing<pre>\$ git remote add <remote> <link></pre><p>Connects your local repo to the remote one. Usually the default value for <remote> is origin<pre>\$ vi .gitignore</pre><p>Opens .gitignore file. This file is used for a list of files that have to be excluded. Ensure that this file is in the root of the local repo. You can change vi to your favorite text editor</p></p></p></p>	<pre>\$ git add <path></pre> <p>Adds path into staging. Path can be file or directory</p> <pre>\$ git restore --staged <path></pre> <p>Removes path from staging back to unstaged area<pre>\$ git rm -r <path></pre><p>Removes path and adds that change into staging<pre>\$ git commit -m <message></pre><p>Commits the stage with specified message<pre>\$ git commit --amend -m <message></pre><p>Repairs last commit with specified new message. Change -m <message> to --no-edit to repair without editing commit message<pre>\$ git status</pre><p>Lists which files are staged, unstaged, or untracked<pre>\$ git push <remotes> <branch></pre><p>Uploads <branch> branch to same branch in <remotes><pre>\$ git pull -r</pre><p>Updates local branch with all new commits from remote branch with rebasing, avoiding the conflict with changes from remote</p></p></p></p></p></p></p>	<pre>\$ git log</pre> <p>Lists version history for the current branch. Add --pretty=oneline to show commit hashes and messages only</p> <pre>\$ git diff <commit1> <commit2></pre> <p>Shows difference between two commits. It is also used to comparing two branches. Add --name-only to show the file names only<pre>\$ git stash</pre><p>Saves current changes into stash stack. Usually used when current changes don't want to be committed<pre>\$ git stash pop</pre><p>Applies last changes stored in stash stack onto current working HEAD<pre>\$ git stash list</pre><p>Shows stash stack<pre>\$ git revert <commit></pre><p>Creates new commit that undoes all of the changes in <commit><pre>\$ git reset <commit></pre><p>Undoes the commits after <commit>, keep the changes locally. Add --hard to discard the changes<pre>\$ git blame -- <file></pre><p>Shows revision in <file> line by line</p></p></p></p></p></p></p>	<pre>\$ git checkout <branch></pre> <p>Switches to the specified branch</p> <pre>\$ git checkout -</pre> <p>Switches to the previous visited branch</p> <pre>\$ git checkout -b <name></pre> <p>Creates a new branch with specified name and switch in that branch<pre>\$ git checkout <path></pre><p>Restores changes of <path> back into latest revision<pre>\$ git branch</pre><p>Lists all branches<pre>\$ git branch -a <old> <new></pre><p>Renames branch from <old> to <new><pre>\$ git branch -d <branch></pre><pre>\$ git push <remote> --delete <branch></pre><p>Deletes the specified branch in local and remote correspondingly<pre>\$ git remote set-url <remote> <url></pre><p>Changes remote url. Usually used after repository migration<pre>\$ git cherry-pick <commit></pre><p>Creates new commit by applying changes in <commit> into current working HEAD<pre>\$ git gc --prune=now --aggressive</pre><p>Cleanups and optimizes all files in local repository<pre>\$ git bisect (start,good,bad)</pre><p>Finds the commit that contains bug with binary search. Use start to begin the bisect, good to mark good commits, bad to mark bad commits</p></p></p></p></p></p></p></p></p>	<pre>\$ git checkout -R <old_branch></pre> <pre>\$ git push <remote> :<old_branch> <new_branch></pre> <p>Rename branch in local and remote correspondingly</p> <pre>\$ git tag <tag_name></pre> <pre>\$ git push <remote> --tags</pre> <p>Create tag and push all created tags<pre>\$ git tag -d <tag_name></pre><pre>\$ git push <remote> --delete <tag_name></pre><p>Delete tag and push deleted tags<pre>\$ git remote set-url <remote> <url></pre><p>Changes remote url. Usually used after repository migration<pre>\$ git cherry-pick <commit></pre><p>Creates new commit by applying changes in <commit> into current working HEAD<pre>\$ git gc --prune=now --aggressive</pre><p>Cleanups and optimizes all files in local repository<pre>\$ git bisect (start,good,bad)</pre><p>Finds the commit that contains bug with binary search. Use start to begin the bisect, good to mark good commits, bad to mark bad commits</p></p></p></p></p></p>

- Haz commits pequeños y frecuentes
- Usa ramas para nuevas características o experimentos
- Escribe mensajes de commit claros y descriptivos
- Revisa tu código antes de hacer push
- Actualiza tu repositorio local frecuentemente

Pull Request y merge

Es una petición para integrar nuestras propuestas o cambios de código a un proyecto.

Estos permiten no solo llevar de forma más ordenada las tareas en la etapa del desarrollo, sino también crear propuestas o cambios que puedan ser integrados posteriormente a dicho proyecto.



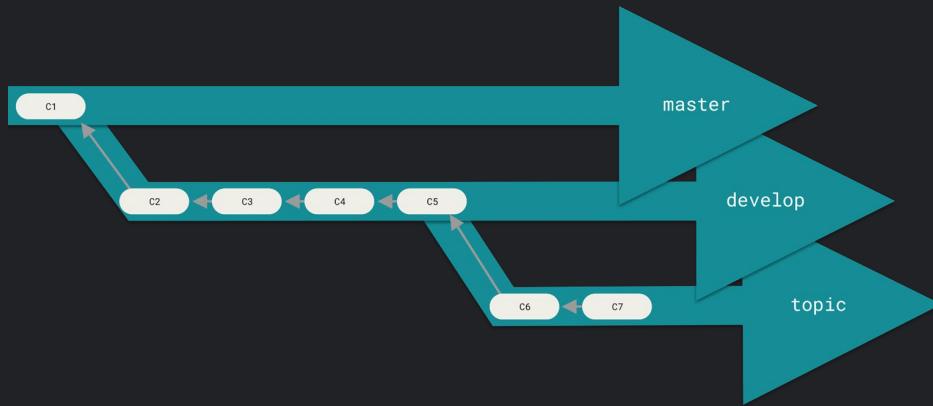
Ramas de Largo Recorrido

En proyectos grandes, se usan ramas de larga duración para diferentes niveles de estabilidad:

- **master**: Solo contiene código completamente estable.
- **develop o next**: Usado para el desarrollo diario y para pruebas de estabilidad antes de integrarse en master.

Se fusionan regularmente entre sí

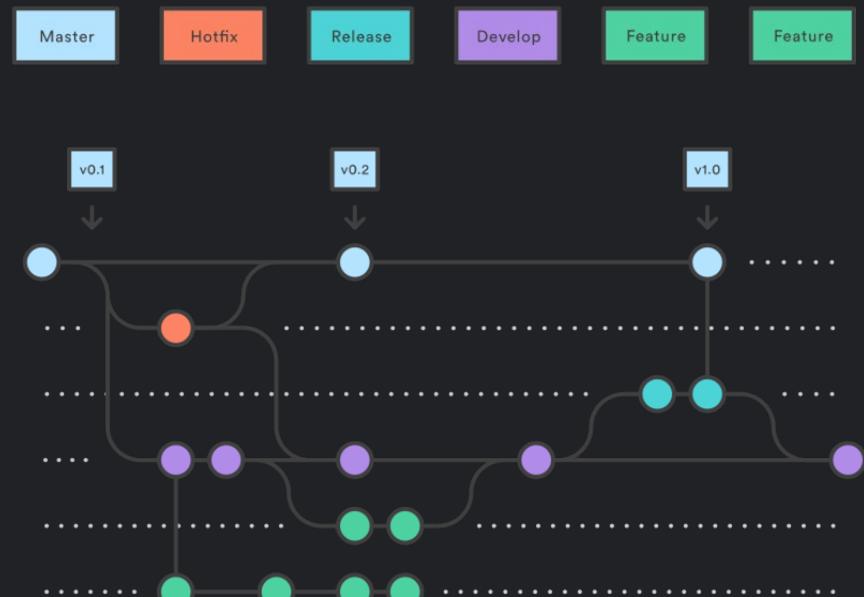
Proporcionan estabilidad y permiten desarrollo continuo



Flujos de Trabajo Ramificados

Git Flow

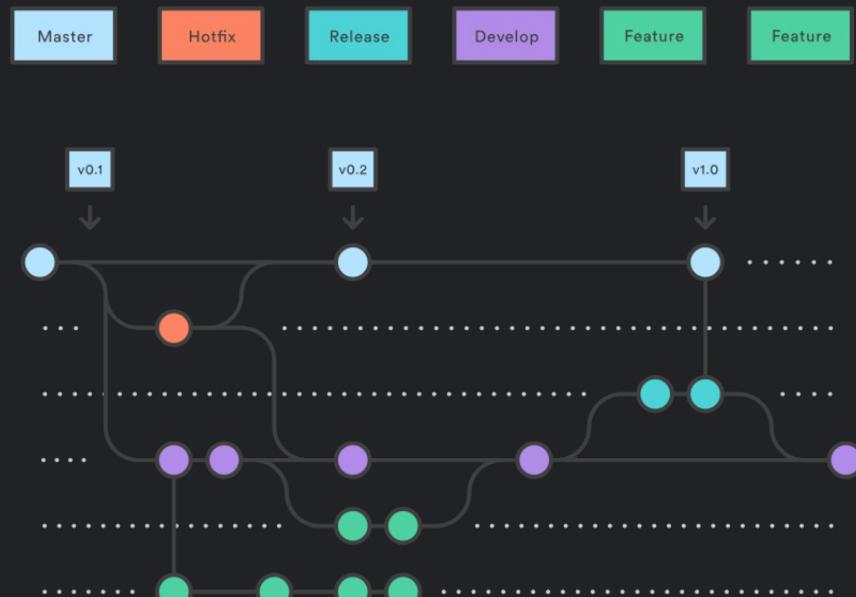
- Introducido por Vincent Driessen en 2010.
- Ramas definidas:
 - master
 - develop
 - feature/*
 - release/*
 - hotfix/*
- Ideal para:
 - Proyectos con ciclos de lanzamiento programados.
 - Necesidad de correcciones urgentes.



Flujos de Trabajo Ramificados

Git Flow

- Introducido por Vincent Driessen en 2010.
- Ramas definidas:
 - master
 - develop
 - feature/*
 - release/*
 - hotfix/*
- Ideal para:
 - Proyectos con ciclos de lanzamiento programados.
 - Necesidad de correcciones urgentes.

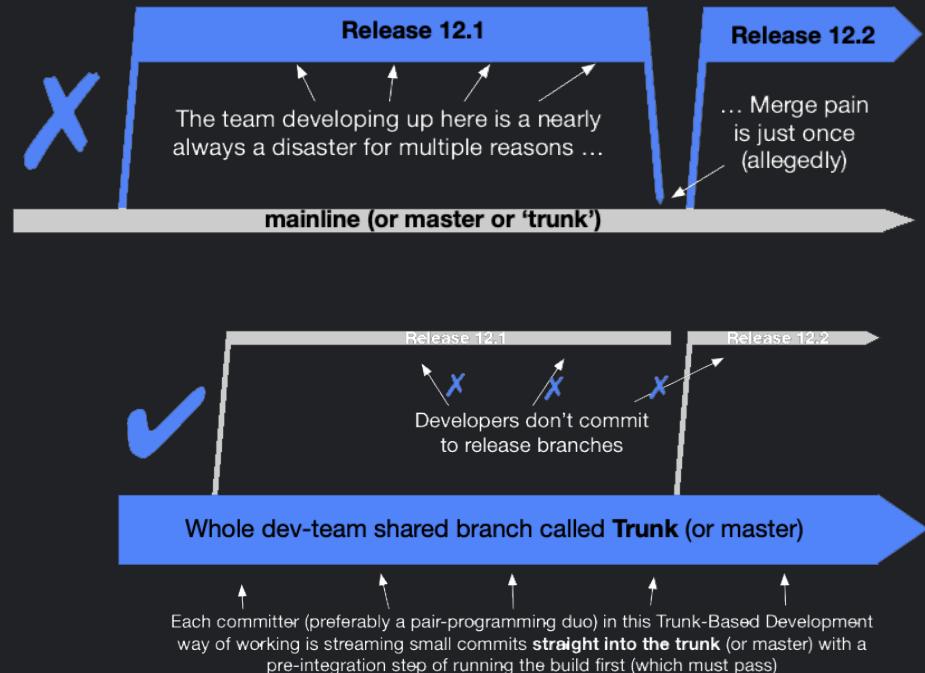


Flujos de Trabajo Ramificados

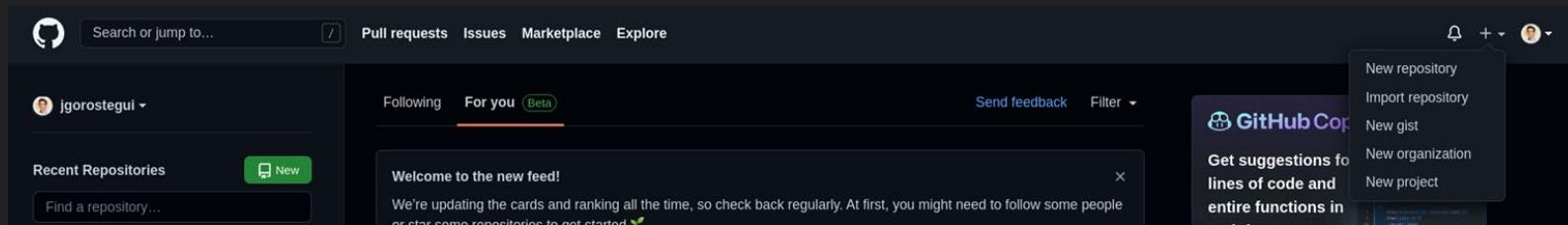
- Desarrolladores trabajan en ramas de corta duración o directamente en el tronco.
- Se mantiene el tronco siempre en estado lanzable.
- Depende de la Integración Continua (CI).
- Uso de "feature flags" para gestionar características incompletas.

Ideal para:

- Integraciones frecuentes.
- Proyectos con muchos colaboradores.
- Uso intenso de CI/CD.



Creando un repositorio en GitHub



Creando un repositorio en GitHub

GitHub permite crear de forma gratuita repositorios tanto públicos como privados. También tiene un editor online

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * Repository name *

jgorostegui / testrepo ✓

Great repository names are short and memorable. Need inspiration? How about [ubiquitous-fiesta](#)?

Description (optional)
test repo

Public Anyone on the internet can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

Add a README file This is where you can write a long description for your project. [Learn more](#).

Add .gitignore
Choose which files not to track from a list of templates. [Learn more](#).

.gitignore template: None ▾

Choose a license
A license tells others what they can and can't do with your code. [Learn more](#).

License: None ▾

You are creating a public repository in your personal account.

Create repository

Trabajando en el repositorio remoto

Para trabajar con repositorios remotos necesitaremos poder clonar para copiar un repositorio de GitHub. Para ello se utiliza:

- **git clone <url>**

Una vez se tengan los cambios realizados, para poder cargar el contenido a un repositorio remoto se utiliza:

- **git push**
- **git push origin master**



Trabajando en el repositorio remoto

Creando un repositorio en GitHub

GitHub permite crear de forma gratuita repositorios tanto públicos como privados. También tiene un editor online

The screenshot shows the GitHub interface for creating a new repository. At the top, it says 'Create a new repository'. Below that, it asks if the user already has a project repository elsewhere and provides a link to 'Import a repository'. The 'Owner' field is set to 'igorostegui' and the 'Repository name' field is 'testrepo'. A note below suggests using short and memorable names like 'ubiquitous-fiesta'. The 'Description (optional)' field contains 'test repo'. There are two radio button options for visibility: 'Public' (selected) and 'Private'. Under 'Initialize this repository with:', there is a checkbox for 'Add a README file' which is checked, and a note explaining it's where to write a long description. There is also a section for '.gitignore' with a dropdown menu showing 'None'. A 'Choose a license' section follows, with a note about creating a public repository in a personal account and a dropdown for 'License: None'. At the bottom is a large green 'Create repository' button.

Trabajando en el repositorio remoto

Para trabajar con repositorios remotos necesitaremos poder clonar para copiar un repositorio de GitHub. Para ello se utiliza:

- **git clone <url>**

Una vez se tengan los cambios realizados, para poder cargar el contenido a un repositorio remoto se utiliza:

- **git push**
- **git push origin master**

The screenshot shows a GitHub repository interface. At the top, there are navigation buttons for 'master', '1 branch', '0 tags', and a 'Code' button. Below that is a commit history card for 'jgorostegui' with the commit message 'First commit' and timestamp '21e9a42 13 seconds ago'. This commit is shown as '1 commit'. Below the commit history is a file listing for 'first.txt' with the same details ('First commit', '13 seconds ago').

```
$ git clone git@github.com:jgorostegui/test.git
Cloning into 'test'...
warning: You appear to have cloned an empty repository.
$ cd test/
$ ls
$ echo "hello" >> first.txt
$ git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
$ git add first.txt
$ git commit -m "First commit"
[master (root-commit) 21e9a42] First commit
  1 file changed, 1 insertion(+)
  create mode 100644 first.txt
$ git push
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 222 bytes | 222.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:jgorostegui/test.git
 * [new branch]      master -> master
```

▶ Trabajar con Ramas

Ejemplo práctico

Clonando el repositorio <https://github.com/microsoft/vscode>

Para ello se ejecuta: `git clone https://github.com/microsoft/vscode`

Es posible ver los archivos se ejecuta `git log`

```
commit 1895178d0009566b2ae04a58cd9a9a51e125eff8 (HEAD -> main, origin/main, origin/HEAD)
Author: Matt Bierner <matb@microsoft.com>
Date: Tue Aug 30 15:36:40 2022 -0700

    Finalize notebookWorkspaceEdit api (#159613)

    Fixes #155245

commit ffc2374a824f337d5816b51a49b61f1477cf70af
Author: David Dossett <djossett@microsoft.com>
Date: Tue Aug 30 14:28:16 2022 -0700

    Remove border radius on selected text in SCM view (#159612)

    Remove border radius on selected text

commit dbfbef5c022859b0baae385434c9bbbc4d68208
Author: Logan Ramos <lramos15@gmail.com>
Date: Tue Aug 30 15:07:38 2022 -0400

    Fix x button overlapping (#159603)

commit 65b53f2aa3a63f7f919ba29613c08c22c2dbbbca
Author: Robo <hop2deep@gmail.com>
Date: Wed Aug 31 04:02:14 2022 +0900

    fix: notify windows 7 support eol (#159546)
```

The screenshot shows the GitHub repository page for `microsoft / vscode`. The top navigation bar includes options like Watch, Fork, Star, and Security. Below the header, there are tabs for Code, Issues, Pull requests, Actions, Projects, Wiki, and Security. The main content area displays the commit history for the `main` branch. Two commits are visible for August 30, 2022:

- Finalize notebookWorkspaceEdit api (#159613)** by `mjbvz` committed 2 hours ago. This commit is verified.
- Remove border radius on selected text in SCM view (#159612)** by `daviddossett` committed 3 hours ago. This commit is verified.

Below these, there are two more commits from August 31, 2022:

- Fix x button overlapping (#159603)** by `lramos15` committed 6 hours ago. This commit is verified.
- fix: notify windows 7 support eol (#159546)** by `deepak1556` committed 6 hours ago. This commit is verified.

Further down, there are two more commits from August 31, 2022:

- Revert "Queue all parts of debug start that can require user input (#... (** by `roblourens` committed 6 hours ago. This commit is verified.
- Merge pull request #158609 from microsoft/tyriar/156179-options** by `Tyriar` committed 8 hours ago. This commit is verified.

- **README.md**
 - Descripción general del proyecto.
 - Instrucciones de instalación, uso y contribución.
- **.gitignore**
 - Lista de archivos y directorios que Git debería ignorar.
- **LICENSE**
 - Define cómo otros pueden usar, contribuir o distribuir tu proyecto.
- **src/ o app/**
 - Directorio que contiene el código fuente del proyecto.
- **test/ o tests/**
 - Contiene los tests unitarios o de integración.
- **docs/**
 - Documentación adicional, guías, etc.
- **assets/ o media/**
 - Imágenes, videos y otros recursos multimedia.
- **CONTRIBUTING.md**
 - Guía para quienes deseen contribuir al proyecto.
- **CHANGELOG.md o HISTORY.md**
 - Historial de versiones y cambios del proyecto.
- **package.json, package-lock.json, etc. (para proyectos Node.js)**
 - Detalla las dependencias y scripts del proyecto.
- **Diversos archivos de configuración**
- Dependiendo del proyecto y tecnología: **.travis.yml** (para integración continua con Travis CI), **Dockerfile** (para contenedores Docker), etc.

Añadiendo colaboradores

The screenshot shows a GitHub repository page for 'jgorostegui/testrepo'. A red arrow points from the top-left to the 'Private' button in the header. Another red arrow points from the bottom-left to the 'Collaborators' tab in the 'Who has access' section. A third red box highlights the 'Settings' tab in the header.

Who has access

- Access**: Shows 'Collaborators' selected (highlighted by a red arrow).
- PRIVATE REPOSITORY**: Shows 'Only those with access to this repository can view it.' with a 'Manage' link.
- DIRECT ACCESS**: Shows '0 collaborators have access to this repository. Only you can contribute to this repository.'

Manage access

You haven't invited any collaborators yet

Add people

Realizar un PR

Para trabajar con repositorios remotos necesitaremos poder clonar para copiar un repositorio de GitHub. Para ello se utiliza:

- `git clone <url>`

Una vez se tengan los cambios realizados, para poder cargar el contenido a un repositorio remoto se utiliza:

- `git push`
- `git push origin master`

Resolviendo conflictos en un Merge request

- Los conflictos pueden ocurrir porque no estamos sincronizados con remotos

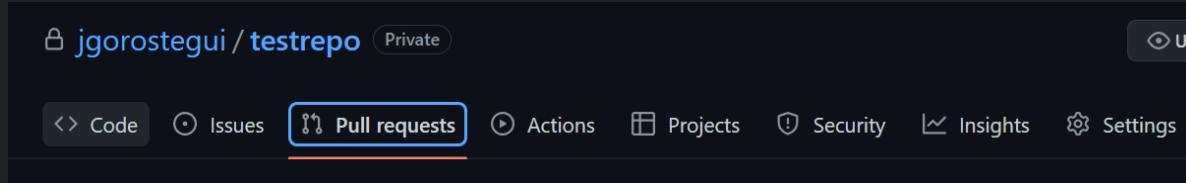
```
ubuntueridas@ubuntu:~/tmp/testrepo$ git add second.txt
ubuntueridas@ubuntu:~/tmp/testrepo$ git cm "Update second.txt"
^[[3-[main 7a61144] Update second.txt
 1 file changed, 2 insertions(+), 1 deletion(-)
ubuntueridas@ubuntu:~/tmp/testrepo$ git push
To github.com:jgorostegui/testrepo.git
  ! [rejected]      main -> main (fetch first)
error: failed to push some refs to 'git@github.com:jgorostegui/testrepo.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
ubuntueridas@ubuntu:~/tmp/testrepo$ S
```

```
[GNOME nano 4.8] /home/ubuntueridas/tmp/testrepo/.git/MERGE_MSG
Merge branch 'main' of github.com:jgorostegui/testrepo into main
# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
```

```
ubuntueridas@ubuntu:~/tmp/testrepo$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
Unpacking objects: 100% (3/3), 684 bytes | 684.00 KiB/s, done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
From github.com:jgorostegui/testrepo
  4c00b0a..3591f6c main -> origin/main
Merge made by the 'recursive' strategy.
 first.txt | 1 +
 1 file changed, 1 insertion(+)
ubuntueridas@ubuntu:~/tmp/testrepo$ git push
Enumerating objects: 9, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 12 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 606 bytes | 606.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0)
To github.com:jgorostegui/testrepo.git
  3591f6c..de8a2b1 main -> main
ubuntueridas@ubuntu:~/tmp/testrepo$ S
```

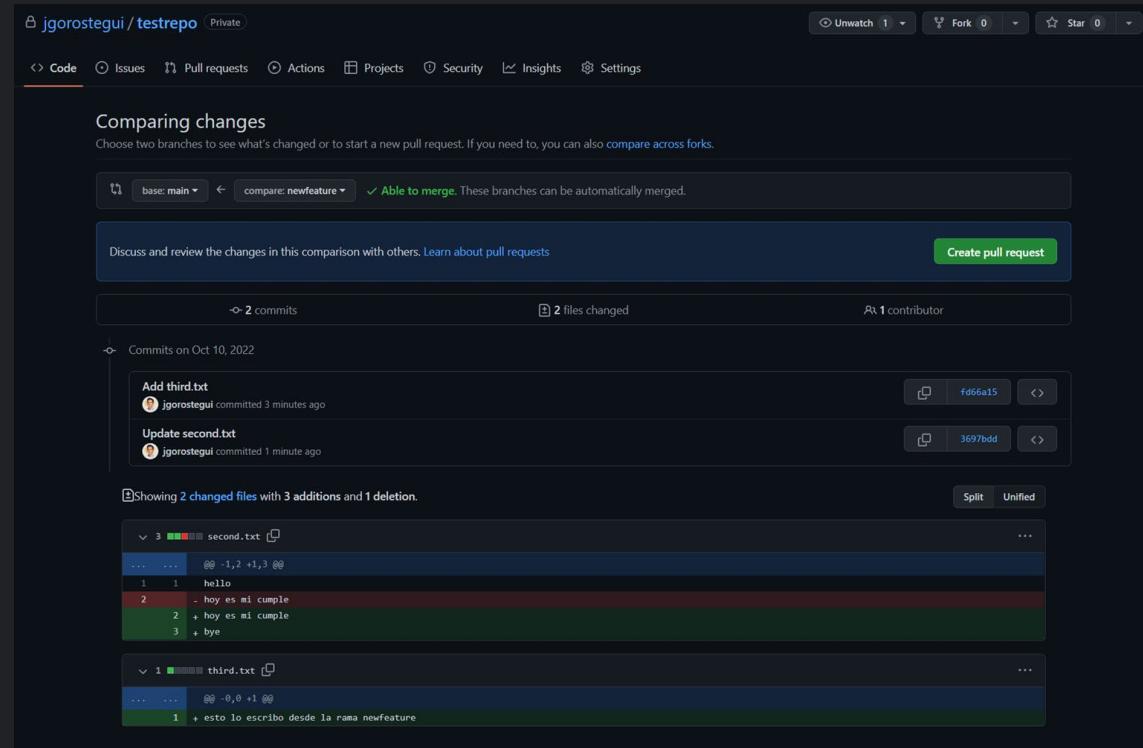
Realizando un Pull Request con GitHub

- En la pestaña de Pull Requests nos permiten seleccionar qué rama se quiere fusionar.



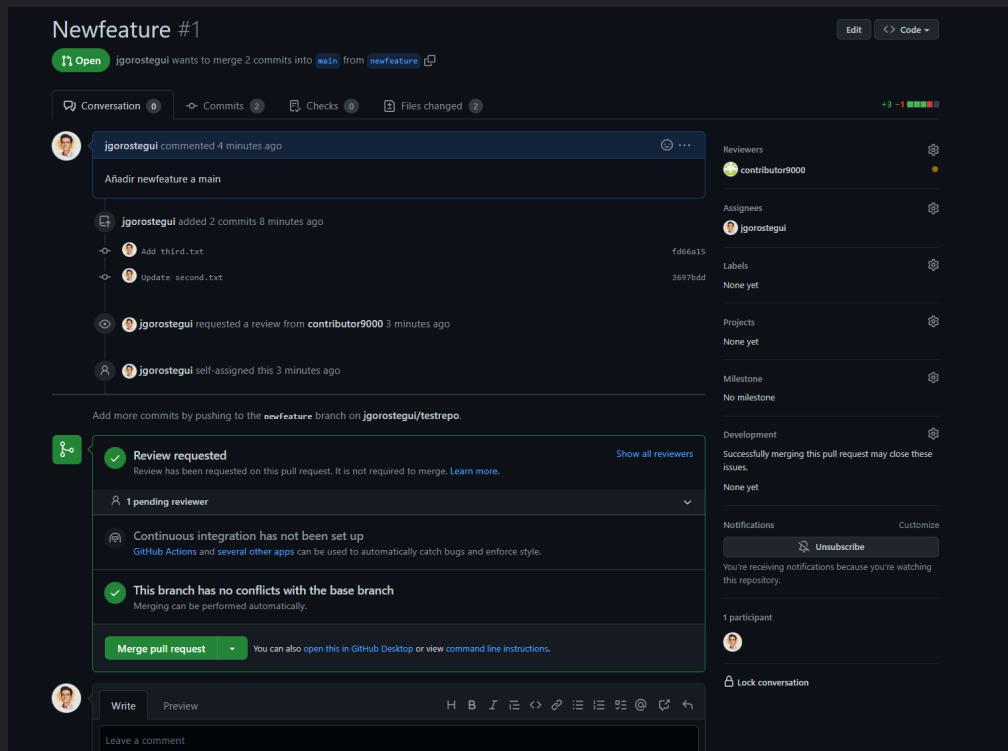
Realizando un Pull Request con GitHub

- Los pull requests son la forma de contribuir al proyecto de forma colaborativa.



Realizando un Pull Request con GitHub

- Los pull requests son la forma de contribuir al proyecto de forma colaborativa.



Volviendo atrás en git

- Los comandos para revertir cambios son:
- `git checkout` / `git clean` / `git revert` / `git reset` `git rm`

```
ubuntuveridas@ubuntu:~/tmp/testrepo$ git log --oneline
811fa77 (HEAD -> main, origin/main, origin/HEAD) Merge pull request #1 from jgorostegui/newfeature
3697bdd (origin/newfeature, newfeature) Update second.txt
fde6a15 Add third.txt
deba2b1 Merge branch 'main' of github.com:jgorostegui/testrepo into main
4f73e9a Update second.txt
3591f6c Update first.txt
4c00b0a create second.txt
116943f Create first.txt
Note: switching to '4f73e9a'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:
  git switch -c <new-branch-name>
Or undo this operation with:
  git switch -
Turn off this advice by setting config variable advice.detachedHead to false
HEAD is now at 4f73e9a Update second.txt
ubuntuveridas@ubuntu:~/tmp/testrepo$ ls -lh
total 8.0K
-rw-rw-r-- 1 ubuntuveridas ubuntuveridas 5 Oct 10 12:23 first.txt
-rw-rw-r-- 1 ubuntuveridas ubuntuveridas 22 Oct 10 12:23 second.txt
ubuntuveridas@ubuntu:~/tmp/testrepo$
```

Fichero .gitignore

- Ignora archivos no deseados en Git (trazas, temporales, compilados).
- Evita la adición automática de archivos no relevantes.
- Previene confirmaciones accidentales.
- Ejemplo Visual: Imagen de un archivo .gitignore y varios archivos a su alrededor con algunos resaltados y otros atenuados para representar los archivos ignorados.

Fichero .gitignore

Ejemplo

```
● ● ●

# ignora los archivos terminados en .a
*.a
# pero no lib.a, aun cuando había ignorado los archivos terminados en .a en
# la linea anterior
!lib.a
# ignora únicamente el archivo TODO de la raiz, no subdir/TODO
/TODo
# ignora todos los archivos del directorio build/
build/
# ignora doc/notes.txt, pero no este: doc/server/arch.txt
doc/*.txt
# ignora todos los archivos .txt del directorio doc/
doc/**/*.txt
```

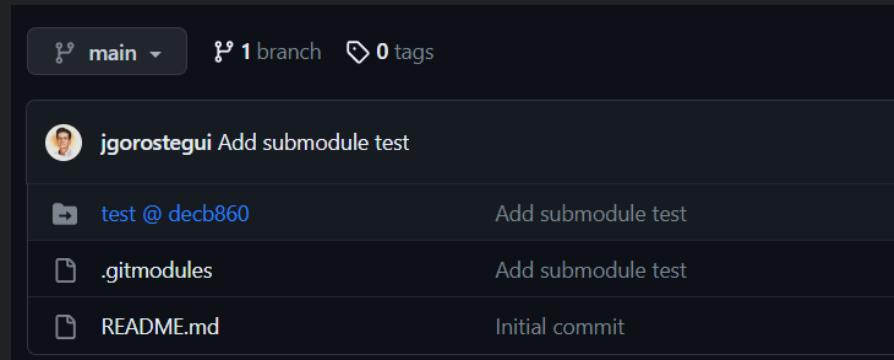
Submódulos

¿Qué es un Submódulo?

- Un repositorio Git incrustado dentro de otro repositorio.
- Permite trabajar en un proyecto dentro de otro, manteniendo su historial separado.

Utilidad

- Reutilizar código entre múltiples proyectos.
- Mantener librerías o componentes separados pero integrados en un proyecto principal.



Submódulos

Añadiendo un submódulo

- git submodule add [URL]
- git commit -m "Mensaje"

El archivo `.gitmodules` se crea automáticamente

```
ubuntuveridas@ubuntu:~/father$ cat .gitmodules
[submodule "test"]
    path = test
    url = git@github.com:jgorostegui/test.git
```

Git LFS

- Una extensión para Git que maneja archivos grandes.
 - Soluciona el problema de versionar archivos que son demasiado grandes para ser manejados eficientemente por Git.
- ¿Por qué es necesario?
 - Git fue diseñado para código fuente, no para archivos binarios grandes.
 - Repositorios con archivos grandes se vuelven lentos y difíciles de clonar.
- Git LFS utiliza el archivo `.gitattributes` para determinar qué archivos deben ser manejados por LFS.



```
*.tflite filter=lfs diff=lfs merge=lfs -text  
*.tar filter=lfs diff=lfs merge=lfs -text  
*.joblib filter=lfs diff=lfs merge=lfs -text  
*.npy filter=lfs diff=lfs merge=lfs -text  
*.json filter=lfs diff=lfs merge=lfs -text  
*.jpg filter=lfs diff=lfs merge=lfs -text  
*.csv filter=lfs diff=lfs merge=lfs -text
```

Git LFS

Instalación

- A menudo disponible como un paquete separado.
- Ejemplo: `apt install git-lfs`

Configuración

- Iniciar LFS para un repositorio:
 - `git lfs install`

```
ubuntuveridas@ubuntu:~/test$ git add test.jpg  
ubuntuveridas@ubuntu:~/test$ git lfs status
```

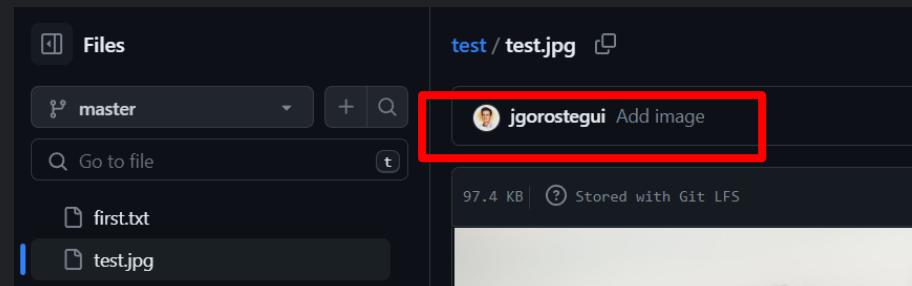
On branch master

Git LFS objects to be committed:

test.jpg (LFS: 904f78c)

Git LFS objects not staged for commit:

```
ubuntuveridas@ubuntu:~/test$ git cm "Add image"  
[master decb860] Add image  
 1 file changed, 3 insertions(+)  
 create mode 100644 test.jpg
```



Alias de Git

- Permiten definir atajos para comandos Git largos o complicados.
- Mejora la eficiencia al interactuar con Git.
- Personaliza y adapta Git a tu flujo de trabajo.

Configuración

Aunque se puede configurar mediante línea de comandos, lo más sencillo es añadir la configuración a `~/.gitconfig`

```
ubuntuveridas@ubuntu:~$ cat .gitconfig
[user]
    email = jgorostegui@veridas.com
    name = Josu Gorostegui

[filter "lfs"]
    clean = git-lfs clean -- %f
    smudge = git-lfs smudge -- %f
    process = git-lfs filter-process
    required = true

[include]
    path = gitalias.txt
```

```
[alias]
    s = status
    ci = commit
    co = checkout
    br = branch
```

Git Hooks

Git Hooks son scripts que se ejecutan automáticamente antes o después de ciertos eventos en un repositorio de Git.

- Se almacenan en el directorio `.git/hooks/`.
- Permiten automatizar tareas, como validación de código, formateo automático o despliegue.

Tipos de Git Hooks

Client-side hooks: Se ejecutan en acciones locales como commits y merges.

- pre-commit: Antes de realizar un commit.
- post-commit: Despues de realizar un commit.
- pre-merge: Antes de realizar un merge.
- post-merge: Despues de realizar un merge.

Server-side hooks: Se ejecutan en acciones del servidor como recibir push y actualizar referencias.

- pre-receive: Antes de aceptar un push en el servidor.
- post-receive: Despues de aceptar un push en el servidor.
- update: Antes de actualizar una referencia en el servidor.

▶ Buenas prácticas

Escribiendo mensajes de commit descriptivos

Un error habitual es no prestar suficiente atención a los mensajes de commit.

Los contribuidores de repositorios como *Linux* y *Git*, conocen que la comunicación a la hora de dar contexto es relevante.

En otras palabras:

- Un **diff** te dirá **que** ha cambiado.
- El **commit** te dirá **porqué**.

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDFKLJF	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

¿Qué es GitHub Pages?

- Servicio de alojamiento de sitios web estáticos directamente desde repositorios de GitHub
 - Gratis para proyectos de código abierto
 - Ideal para documentación de proyectos, portafolios, blogs, etc.

Opciones de publicación:

- Desde una rama específica (ej. gh-pages)
- Desde la carpeta /docs en la rama principal
- Desde la rama principal

¿Qué es GitHub Actions?

- Herramienta de automatización de flujos de trabajo
- Integrada directamente en GitHub
- Permite CI/CD y automatización de tareas

Componentes Clave

1. **Workflows**: Procesos automatizados configurables
2. **Jobs**: Conjunto de pasos que se ejecutan en un runner
3. **Steps**: Tareas individuales dentro de un job
4. **Actions**: Comandos reutilizables para los steps
5. **Runners**: Servidores que ejecutan los workflows

Casos de Uso Comunes

- Integración continua (CI)
- Despliegue continuo (CD)
- Pruebas automatizadas
- Empaquetado y publicación de releases
- Automatización de tareas de mantenimiento

```
name: CI

on: [push]

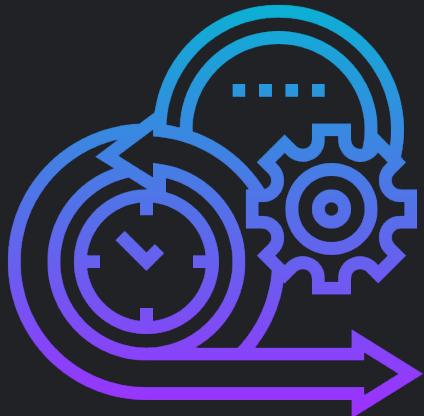
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Run tests
        run: make test
      - name: Deploy
        run: make deploy
```

Git & Github

- <https://git-scm.com/>
- <https://git-scm.com/about>
- <https://mirrors.edge.kernel.org/pub/software/scm/git/docs/user-manual.html>
- https://learngitbranching.js.org/?locale=es_ES
- Libro Pro Git <https://book.git-scm.com/book/en/v2>
- Version Control with Git | udacity
- Git Katas: <https://github.com/eficode-academy/git-katas>
- Guías para escribir un buen mensaje de commit
 - https://wiki.openstack.org/wiki/GitCommitMessages#Information_in_commit_messages
 - <https://cbea.ms/git-commit/>
 - <https://thoughtbot.com/blog/5-useful-tips-for-a-better-commit-message>

Buenas prácticas y agilidad

Control de versiones y Git



Josu Gorostegui
