

Git y GitHub: Un primer contacto



Alberto Montoro Gavilán
Curso: 2024-2025

Índice de contenidos

1. Introducción
2. Conceptos
3. Flujo de trabajo
4. Instalación de Git
5. Primeros pasos
6. Uso básico de Git
7. Uso básico de GitHub
8. Ramas en Git y GitHub
9. Trabajo colaborativo
10. Algunos comandos útiles

Índice de contenidos

- 1. Introducción**
2. Conceptos
3. Flujo de trabajo
4. Instalación de Git
5. Primeros pasos
6. Uso básico de Git
7. Uso básico de GitHub
8. Ramas en Git y GitHub
9. Trabajo colaborativo
10. Algunos comandos útiles

1. Introducción

Sistema de Control de versiones

- Un Sistema de Control de Versiones (**VCS, Version Control System**) es una herramienta que permite gestionar los cambios en archivos y proyectos a lo largo del tiempo.
- Facilita el seguimiento de modificaciones, la recuperación de versiones anteriores y la colaboración entre múltiples desarrolladores.
- Los VCS son esenciales en el desarrollo de software, ya que permiten trabajar en equipo sin el riesgo de sobrescribir código, mantener un historial de cambios y gestionar diferentes versiones de un proyecto.

1. Introducción

Tipos de Sistemas de Control de versiones

■ Sistemas de Control de Versiones **Locales**

- Se almacenan en un solo equipo.
- No permiten colaboración en red.
 - Ejemplo: [Copias manuales con fechas o versiones en carpetas](#).

■ Sistemas de Control de Versiones **Centralizados** (CVCS)

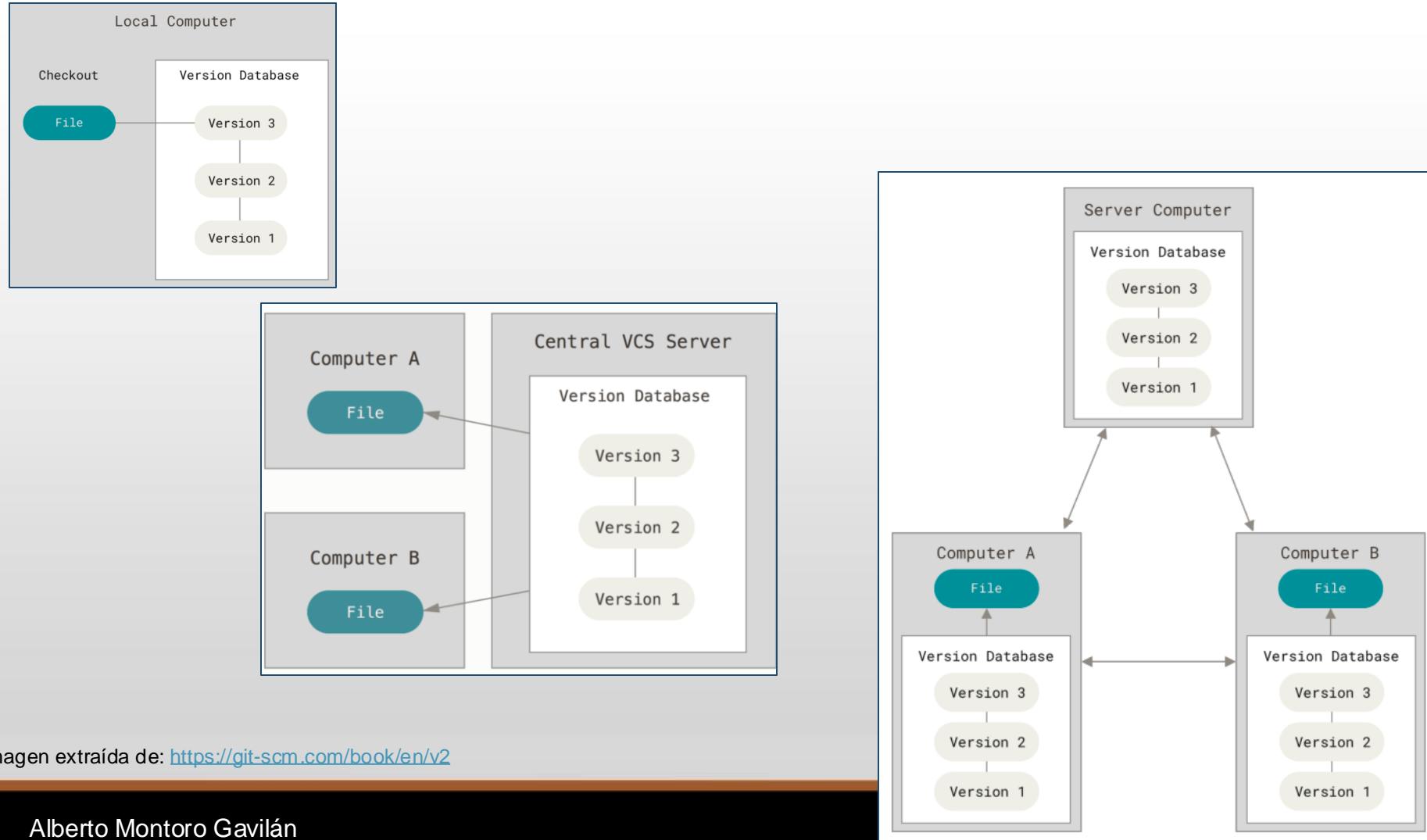
- Tienen un servidor central donde se almacenan todas las versiones.
- Los usuarios descargan los archivos y trabajan sobre ellos, los modifican y lo vuelven a subir. En un momento concreto el desarrollador solo tiene la última copia de un fichero.
- Desventaja: Si el servidor falla, se pierde el historial.
 - Ejemplo: [Subversion \(SVN\), Perforce, CVS](#).

■ Sistemas de Control de Versiones **Distribuidos** (DVCS)

- Cada usuario tiene una copia completa del historial del proyecto.
- No depende de un servidor central para recuperar el historial.
- Permite trabajar sin conexión y realizar múltiples ramas de desarrollo.
 - Ejemplo: [Git, Mercurial](#).

1. Introducción

Tipos de Sistemas de Control de versiones



1. Introducción

Ejemplos de Sistemas de Control de Versiones

- **Copia manual de archivos** (Local) → Método rudimentario donde los usuarios crean copias de seguridad de los archivos con distintos nombres.
- **Subversion (SVN)** (Centralizado) → Usado en empresas con control estricto.
- **Perforce** (Centralizado) → Utilizado en la industria de videojuegos.
- **Git** (Distribuido) → Es el más popular y usado en proyectos de software.
- **Mercurial** (Distribuido) → Similar a Git, pero con enfoque en facilidad de uso.



Imagen de: <https://medium.com/@derya.cortuk/version-control-software-comparison-git-mercurial-cvs-svn-21b2a71226e4>

1. Introducción

Plataformas de Alojamiento de Repositorios

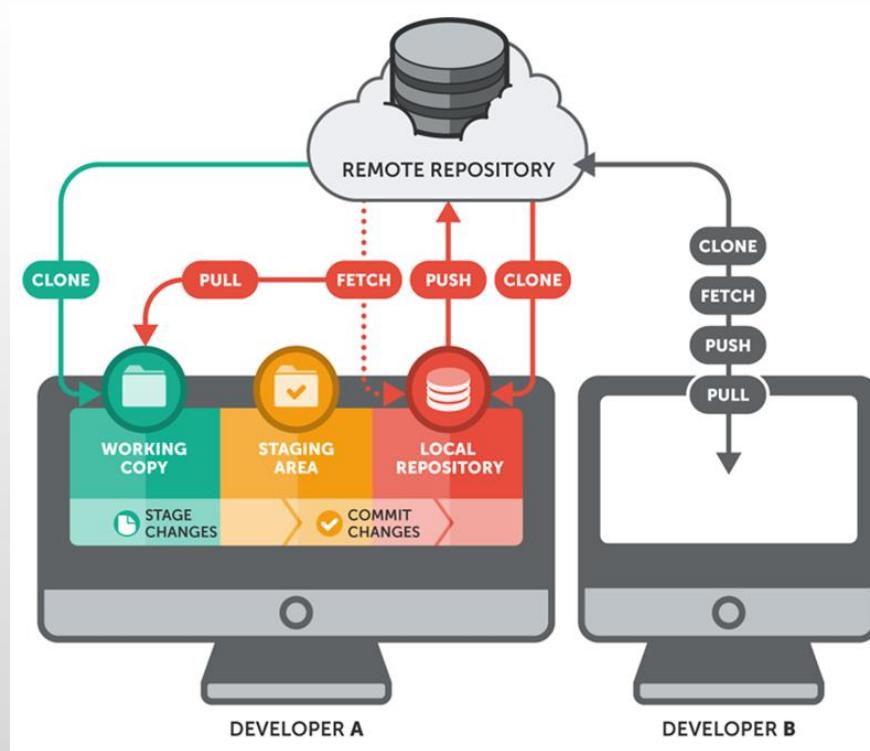


Imagen de: <https://diegobersano.wordpress.com/2017/06/13/introduccion-a-git-repaso-a-los-conceptos-generales/>

1. Introducción

Plataformas de Alojamiento de Repositorios

- Los VCS pueden gestionarse de manera local, pero en entornos colaborativos es clave contar con plataformas en la nube que faciliten el trabajo en equipo, la seguridad y la integración con otras herramientas.

- Plataformas basadas en Git

- GitHub → La más popular para proyectos open-source y empresariales.
- GitLab → Enfocada en DevOps y autoalojamiento.
- Bitbucket → Integración con Atlassian (Jira, Trello).
- Azure DevOps Repos → Solución de Microsoft para desarrollo empresarial.
- AWS CodeCommit → Alternativa de Amazon en la nube.

- Plataformas para otros VCS

Subversion (SVN)

- Alojado en Apache Subversion, Assembla o RhodeCode.

Mercurial

- Soportado en RhodeCode y antiguamente en Bitbucket.

1. Introducción

Plataformas de Alojamiento de Repositorios

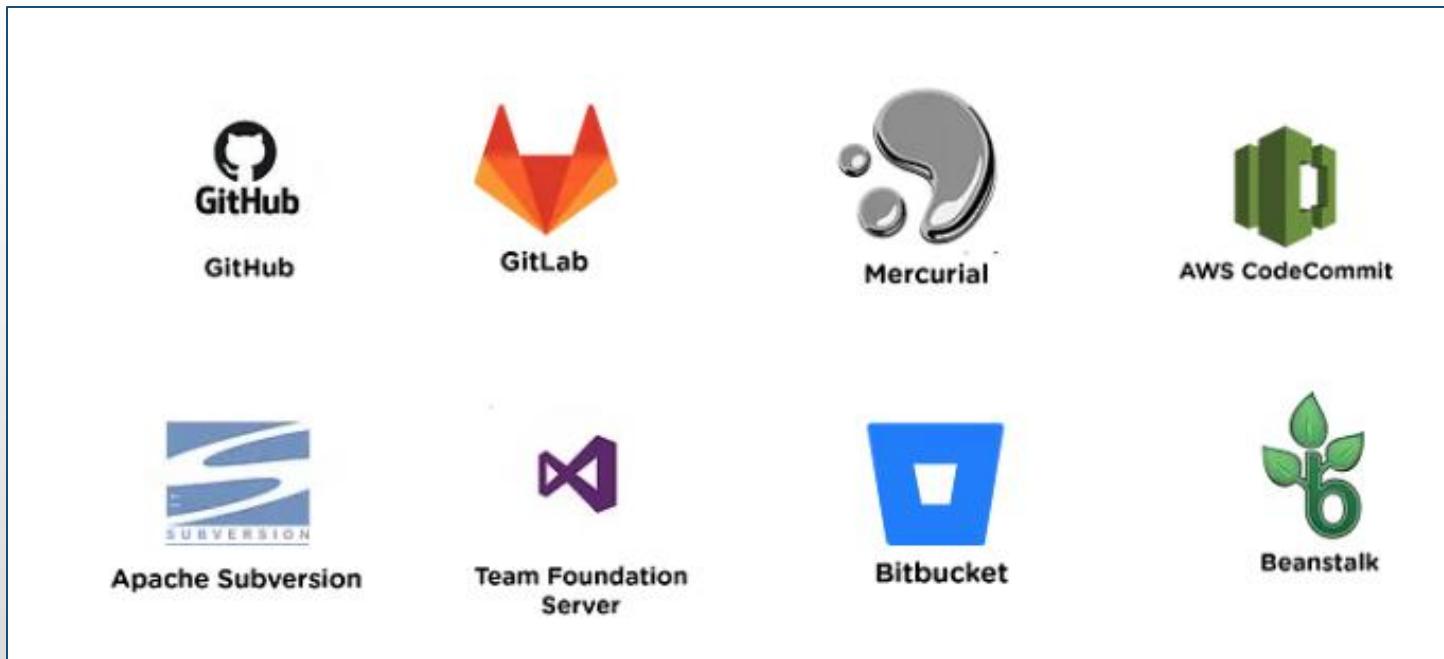


Imagen de <https://blog.stackademic.com/what-is-a-version-control-system-2f3509066b72>

1. Introducción

Git

- Git es un software de control de versiones (VCS) **distribuido** gratuito y open source muy fácil de aprender, ligero y con un rendimiento increíble.
- El sitio web oficial de Git es: <https://git-scm.com/>. Aquí encontrarás las forma de conseguirlo, una amplia documentación y una comunidad dispuesta a ayudar en todo momento.

The screenshot shows the homepage of the official Git website (<https://git-scm.com/>). At the top left is the Git logo, which consists of a red diamond shape containing a white icon that looks like a stylized 'g' or a pair of pliers. To the right of the logo is the word "git" in a large, bold, black sans-serif font. Below the logo, the tagline "--distributed-is-the-new-centralized" is written in a smaller, gray sans-serif font. To the right of the tagline is a search bar with a magnifying glass icon and the placeholder text "Search entire site...". The main content area features two sections of text. The first section, titled "What is Git?", describes Git as a "free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency." The second section, titled "Why Git?", highlights Git's ease of learning, fast performance, and superior features compared to other SCM tools like Subversion, CVS, and ClearCase. It mentions "cheap local branching", "convenient staging areas", and "multiple workflows". To the right of the text is a diagram illustrating the distributed nature of Git. It shows seven white server icons, each representing a repository, connected by a network of colored lines (red, blue, and yellow) forming a mesh-like structure. The background of the page has a light gray grid pattern.

Imagen extraída del sitio web: <https://git-scm.com/>

1. Introducción

Git

- Git es un software de control de versiones (VCS) diseñado por Linus Torvalds y Junio Hamano.
- Su objetivo es mantener versiones de aplicaciones de cualquier tipo con un gran número de archivos de código fuente. Para ello permite:
 - Registrar los cambios en archivos y volver a estados anteriores.
 - Coordinar el trabajo en equipo sobre archivos del repositorio.
 - Llevar a cabo una gestión distribuida.
 - Realizar un desarrollo no lineal mediante ramas.
- Se distribuye bajo los términos de la Licencia Pública General de GNU versión 2.
- Accede al sitio web oficial de Git para aprender más: <https://git-scm.com/>.

1. Introducción

Git

- Git es un VCS distribuido a diferencia de los VCS tradicionales centralizados o de un control de versiones local.
 - En este tipo, los desarrolladores hacen siempre una replica completa del repositorio. No solo descargan la última copia instantánea de los archivos.

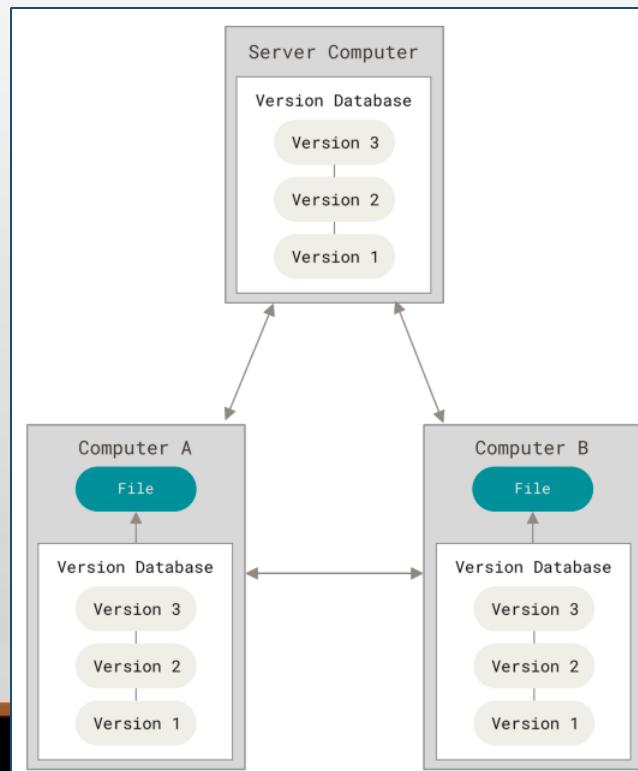


Imagen extraída de: <https://git-scm.com/book/en/v2>

1. Introducción

GitHub

- GitHub es plataforma de desarrollo colaborativo pensada para alojar repositorios (o proyectos) usando Git.
- El sitio web oficial de GitHub es: <https://github.com/>.

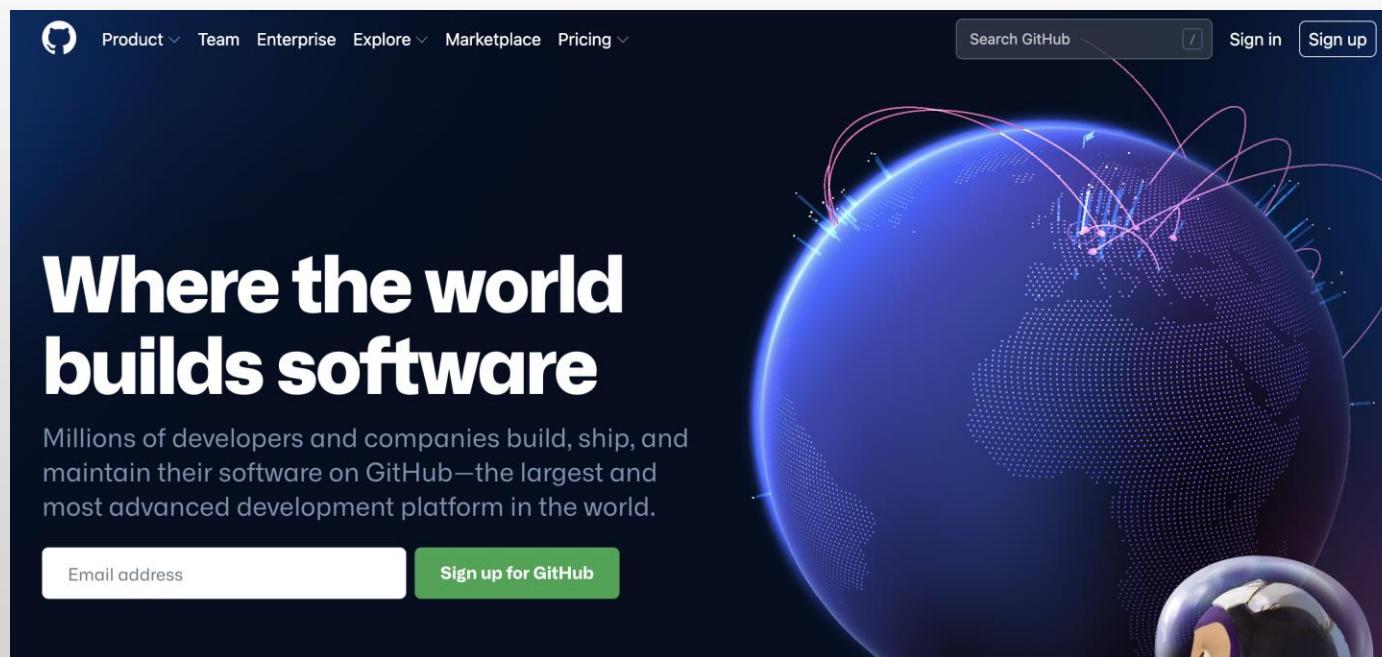


Imagen extraída del sitio web: <https://github.com/>

1. Introducción

GitHub

- GitHub es la plataforma más importante de este tipo para proyectos de código abierto.
- Actualmente pertenece a Microsoft (desde 2018).
- El sitio web oficial de GitHub es: <https://github.com/>.



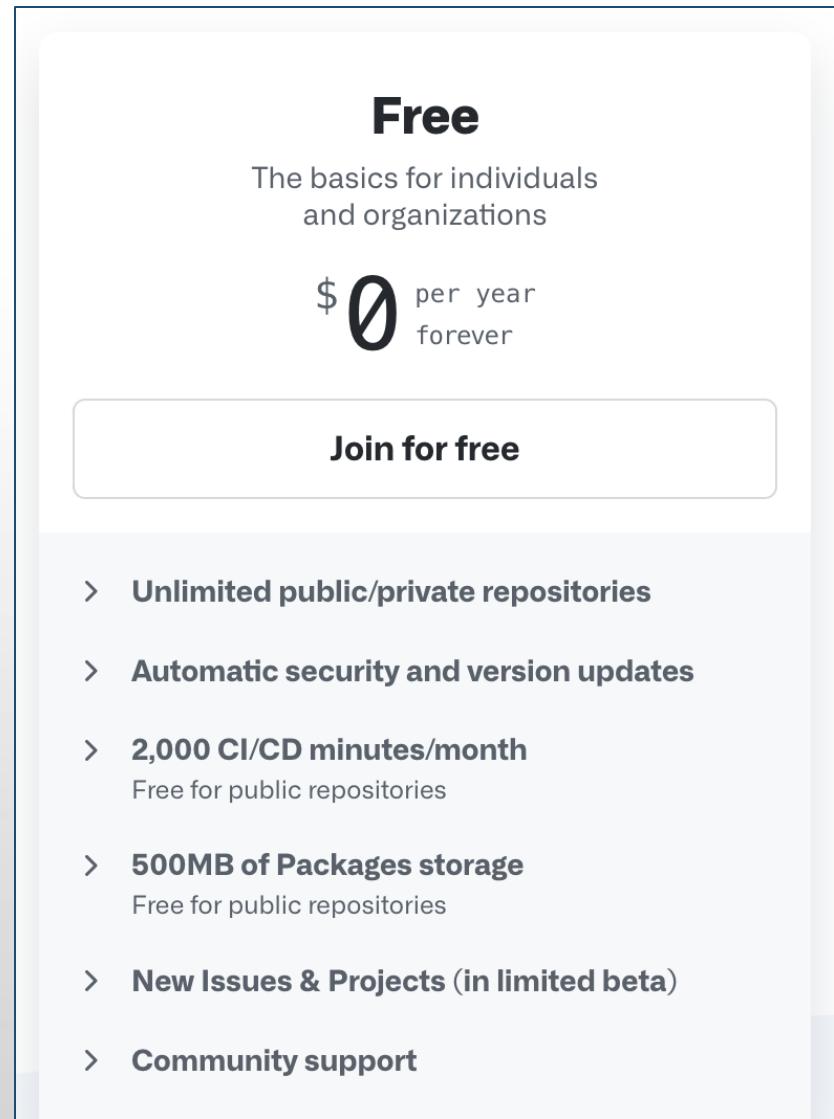
Imagen extraída del sitio web: <https://github.com/>

1. Introducción

GitHub

- La opción gratuita de GitHub nos ofrece:
- Para más información sobre precios:

<https://github.com/pricing>



The screenshot shows the GitHub pricing page for the 'Free' plan. The title 'Free' is at the top, followed by the subtitle 'The basics for individuals and organizations'. It features a large '\$ 0 per year forever' price indicator. A prominent 'Join for free' button is centered below the price. To the right, a list of benefits is displayed, each preceded by a right-pointing arrow.

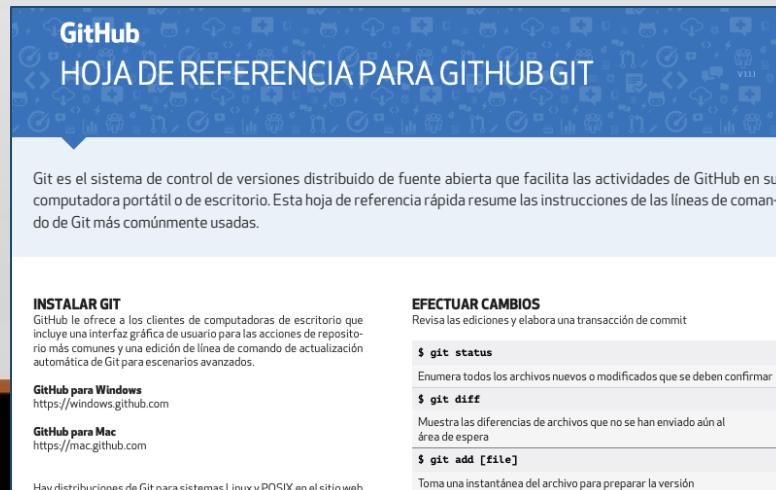
- **Unlimited public/private repositories**
- **Automatic security and version updates**
- **2,000 CI/CD minutes/month**
Free for public repositories
- **500MB of Packages storage**
Free for public repositories
- **New Issues & Projects (in limited beta)**
- **Community support**

Imagen extraída de: <https://github.com/pricing>

1. Introducción

Tarea 1

- Usando una IA y un motor de búsqueda realiza las siguientes tareas.
 - Apartado 1.1. Antes de comenzar a usar Git en profundidad, es importante conocer dónde encontrar la documentación y recursos oficiales.
 - Accede a la web oficial de Git: <https://git-scm.com/>
 - Explora y responde:
 - ¿Dónde puedes encontrar la documentación oficial de Git?
 - Existe una "cheat sheet" o hoja de referencia rápida en español. Descárgala y revísala.
 - ¿Dónde puedes encontrar información sobre los comandos de Git y sus opciones avanzadas?



1. Introducción

Tarea 1

- Usando una IA y un motor de búsqueda realiza las siguientes tareas.
 - Apartado 1.2. GitHub no es solo una plataforma para alojar código, también ofrece varios servicios que pueden ser útiles para desarrolladores, estudiantes y empresas.
 - Accede a la web oficial de GitHub: <https://github.com/>
 - Explora y responde:
 - ¿Dónde puedes encontrar la documentación oficial?
 - Explica brevemente en qué consisten los siguientes servicios relacionados con GitHub:
 - a) GitHub Pages
 - b) Patrocinación (GitHub Sponsors)
 - c) GitHub Gist
 - d) GitHub Codespaces
 - e) GitHub Education

NOTA: Instalación de Git

- Mientras continuamos con las explicaciones teóricas, puedes aprovechar para descargar la última versión de Git para tu Sistema Operativo Anfitrión: <https://git-scm.com/downloads>
- Despues debes instalarlo siguiendo los pasos descritos en el punto 4. Instalación de Git y hacer así la tarea 3.

Índice de contenidos

1. Introducción
- 2. Conceptos**
3. Flujo de trabajo
4. Instalación de Git
5. Primeros pasos
6. Uso básico de Git
7. Uso básico de GitHub
8. Ramas en Git y GitHub
9. Trabajo colaborativo
10. Algunos comandos útiles

2. Conceptos

- Asociado al control de versiones existen una serie de conceptos o términos fundamentales:
 - Directorio de trabajo o Working directory.
 - Área de preparación o Staging area.
 - Repositorio.
 - Inicio o init.
 - Seguimiento o track.
 - Publicar o commit
 - Revisión o versión (Hash ID).
 - Head.
 - Etiquetar o tag.



Imagen extraída de: <https://git-scm.com/about/>

2. Conceptos

- Asociado al control de versiones existen una serie de conceptos o términos fundamentales:
 - Ramas o branch.
 - Master/Main Branch.
 - Fusionar o Merge
 - Revertir o Revert
 - Estado o Status
 - Desplegar o Checkout
 - Cambio o Diff
 - Conflicto: suele ocurrir cuando se trabaja en equipo o entre ramas.
 - Resolver



Imagen extraída de: <https://git-scm.com/about/>

2. Conceptos

- Asociado al control de versiones existen una serie de conceptos o términos fundamentales:
 - **Remote**
 - Origin: nombre por defecto de la URL.
 - **Push**: commits locales al repositorio remoto.
 - **Fetch**: descarga los cambios más recientes desde el repositorio remoto, pero sin fusionarlos con tu código local.
 - **Pull**: fetch + merge: primero trae los cambios desde el remoto (fetch) y luego los combina con la rama local.

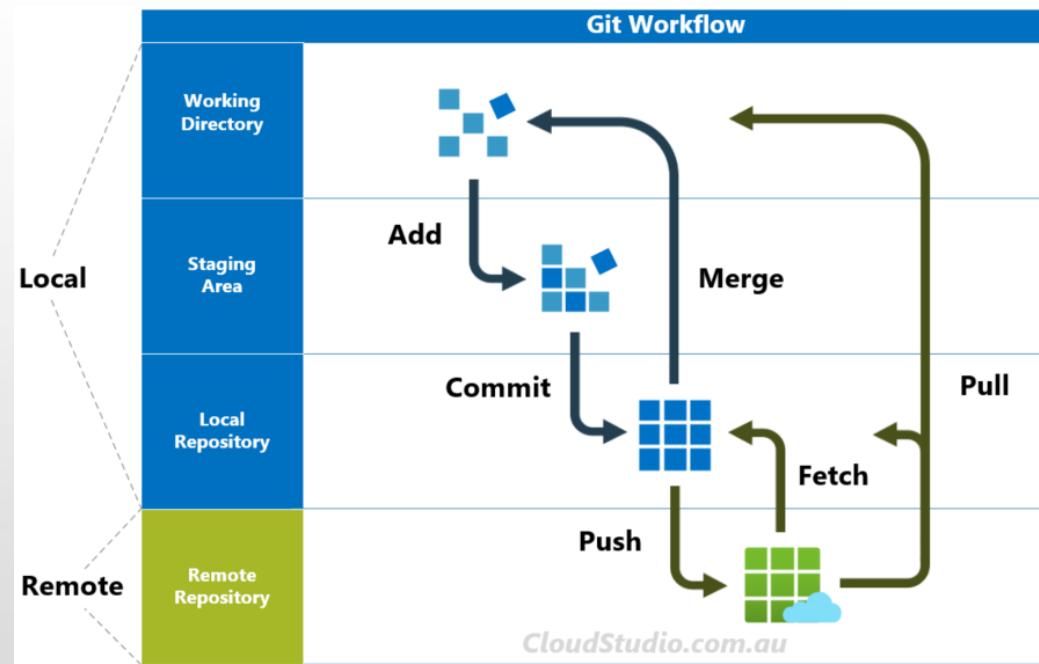


Imagen de <https://cloudstudio.com.au/2021/06/26/git-command/>

2. Conceptos

Tarea 2

- Crea un esquema donde relaciones los conceptos previos y los acompaña de una pequeña descripción.
 - Debes usar una aplicación como MindMeister, Draw.io o Miro
 - Una vez finalizado debe exportar el mapa mental a formato pdf o jpg.
 - Renombre el fichero con pisa3_git_tarea2_apellido1_nombre.
 - Sube el fichero a la carpeta colaborativa en Google Drive.

Índice de contenidos

1. Introducción
2. Conceptos
- 3. Flujo de trabajo**
4. Instalación de Git
5. Primeros pasos
6. Uso básico de Git
7. Uso básico de GitHub
8. Ramas en Git y GitHub
9. Trabajo colaborativo
10. Algunos comandos útiles

3. Flujo de trabajo

- Cuando varias personas trabajan en un mismo proyecto es necesario establecer unas pautas o acuerdos a la hora de trabajar con Git para organizar los cambios y evitar conflictos. A esto se le conoce como flujo de trabajo (Git Workflow).
- Un buen flujo de trabajo ayuda a:
 - Mantener un historial limpio.
 - Evitar conflictos entre desarrolladores.
 - Controlar qué cambios llegan a producción.
- Existen diversos flujos de trabajo y aquí nos vamos a detener en el propuesto de por GitHub en 2011 el cual habla de 6 principios.

3. Flujo de trabajo

"4 branching workflows for Git". De Patrick Porto.

<https://medium.com/@patrickporto/4-branching-workflows-for-git-30d0aaee7bf>

1. Anything in the `master` branch is deployable
 2. To work on something new, create a branch off from `master` and give it a descriptively name(ie: `new-oauth2-scopes`)
 3. Commit to that branch locally and regularly push your work to the same named branch on the server
 4. When you need feedback or help, or you think the branch is ready for merging, open a [pull request](#)
 5. After someone else has reviewed and signed off on the feature, you can merge it into `master`
 6. Once it is merged and pushed to `master`, you can and *should* deploy immediately

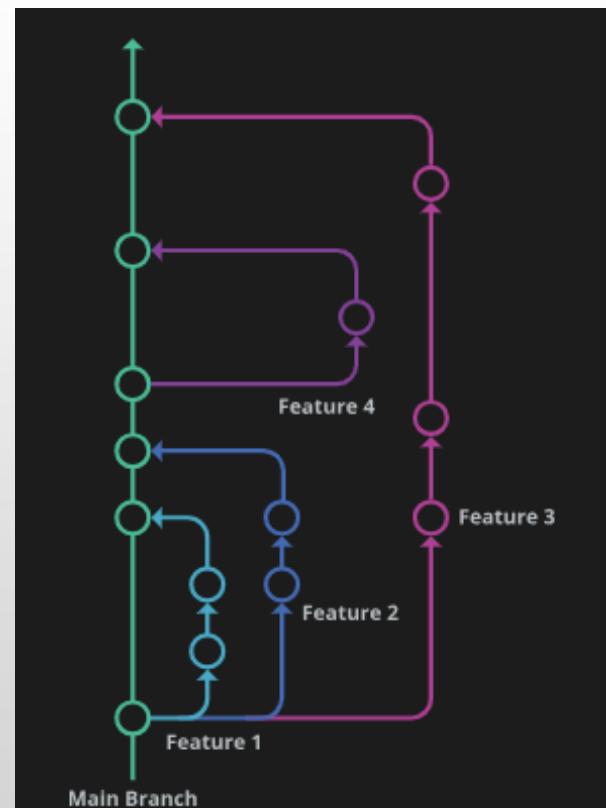


Imagen de <https://www.gitkraken.com/learn>

3. Flujo de trabajo

El **GitHub Flow** es un flujo de trabajo simple y flexible, ideal para desarrollo ágil y despliegue continuo.

Se basa en 6 principios:

1. Todo en la rama principal (main) debe ser desplegable.
2. Para trabajar en algo nuevo, crea una rama desde main con un nombre descriptivo.
3. Haz commits regularmente y sube los cambios a la misma rama en el servidor.
4. Cuando necesites feedback o fusión, abre un Pull Request (PR).
5. Después de la revisión, si la funcionalidad está lista, se fusiona en main.
6. Tras la fusión en main, se puede desplegar inmediatamente.

Texto extraído del artículo: "4 branching workflows for Git". De Patrick Porto. Revisado en mayo de 2022. [Enlace al artículo.](#)

3. Flujo de trabajo

- Además de GitHub Flow, existen otros flujos de trabajo que se adaptan a diferentes necesidades:
 - **Git Flow:** Uso de ramas dedicadas para desarrollo (develop), producción (main), y ramas temporales para características, hotfixes y releases. Adecuado para proyectos con versiones definidas: <https://www.gitkraken.com/learn/git/git-flow>
 - **GitLab Flow:** Similar a GitHub Flow, pero con integración directa con CI/CD y uso de ramas de entorno (staging, production). Diseñado para integración continua.
 - **Forking Workflow:** Usado en proyectos open-source. Cada colaborador trabaja en su propio "fork" y propone cambios mediante Pull Requests.
- Cada flujo de trabajo tiene ventajas y desventajas según el tipo de proyecto y equipo: <https://www.gitkraken.com/learn/git/best-practices/git-branch-strategy>

Índice de contenidos

1. Introducción
2. Conceptos
3. Flujo de trabajo
- 4. Instalación de Git**
5. Primeros pasos
6. Uso básico de Git
7. Uso básico de GitHub
8. Ramas en Git y GitHub
9. Trabajo colaborativo
10. Algunos comandos útiles

4. Instalación de Git

- A continuación veremos cómo se hace la instalación de Git en nuestra máquina anfitriona con un sistema operativo de Microsoft Windows.
- Los pasos para realizar la instalación de Git en distintos SSOO se encuentra en:
<https://git-scm.com/downloads>



Imagen extraída de: <https://git-scm.com/downloads>

Download for Windows

4. Instalación de Git

[Click here to download](#) the latest (2.47.1(2)) 64-bit version of Git for Windows. This is the most recent [maintained build](#). It was released [29 days ago](#), on 2025-01-14.

- Descargar el instalador y sigue los pasos descritos en el asistente realizando la instalación por defecto y fijándote especialmente en:
 - La licencia. [¿Es para software libre?](#)
 - El directorio de instalación. No es necesario cambiarlo.
 - La selección de componentes a instalar predeterminados. No es necesario cambiarlos.
 - Marca usar Visual Studio Code o Notepad++ como editor. **Cambio**.
 - Marca usar la rama 'main'. **Cambio**.
 - Usa MinTTY. [¿Qué es esto?](#)
 - Finalmente lanza Git Bash.

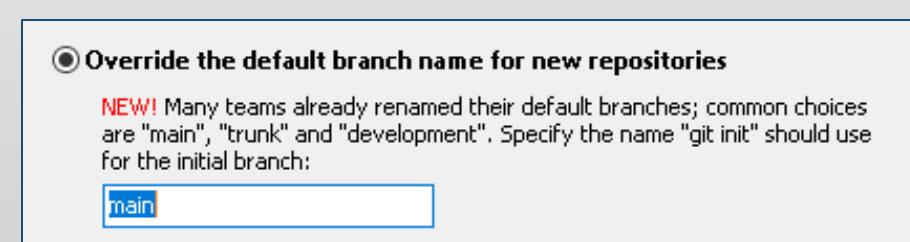


Imagen extraída de: <https://git-scm.com/downloads>

4. Instalación de Git

4.1. Instalación en GNU/Linux

- A continuación veremos cómo se hace la instalación de Git en una máquina virtual donde tenemos instalada la distribución de Xubuntu.
- Los pasos para realizar la instalación de Git en distintos SSOO se encuentra en:
<https://git-scm.com/downloads>

Imagen extraída de: <https://git-scm.com/downloads>

It is easiest to install Git on Linux using the preferred package manager of your Linux distribution. If you prefer to build from source, you can find tarballs on [kernel.org](#). The latest version is [2.36.0](#).

Debian/Ubuntu

For the latest stable version for your release of Debian/Ubuntu

```
# apt-get install git
```

For Ubuntu, this PPA provides the latest stable upstream Git version

```
# add-apt-repository ppa:git-core/ppa # apt update; apt install git
```

4. Instalación de Git

4.1. Instalación en GNU/Linux

- Para instalar en distribuciones tipo Debian es muy sencillo.

- Abre un terminal y ejecuta:

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

- El comando anterior instala la versión de Git incluida en los repositorios oficiales de tu distribución. Es la opción más **segura y estable**, pero puede no ser la última versión disponible.

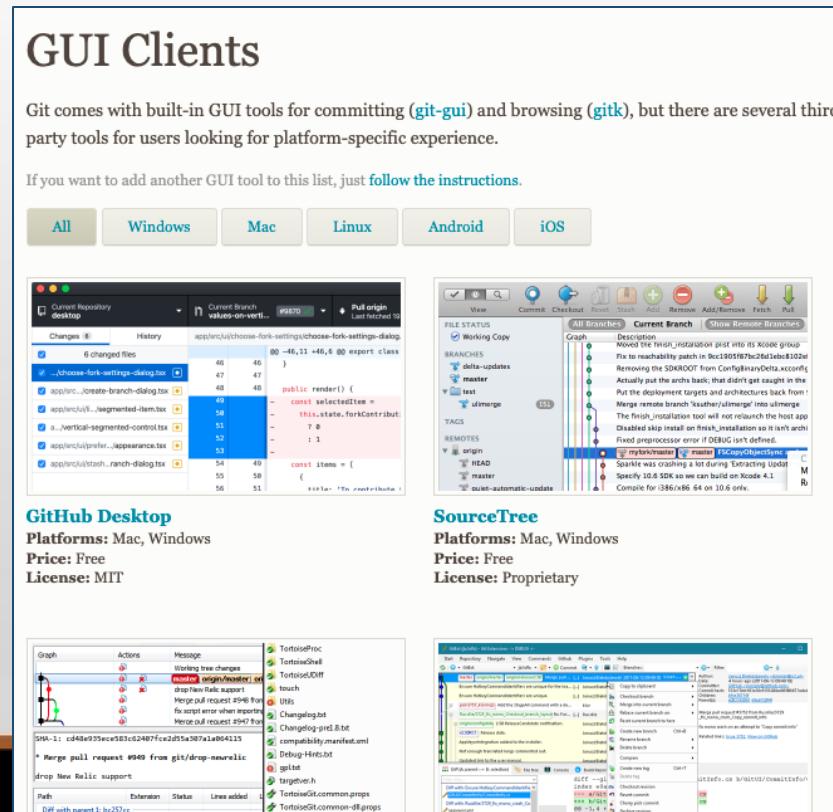
```
sudo add-apt-repository ppa:git-core/ppa
```

```
sudo apt update
```

```
sudo apt install git
```

4. Instalación de Git

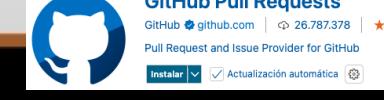
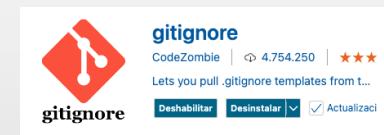
- Nosotros en estas prácticas vamos a trabajar con Git en modo línea de comandos o usando Visual Studio Code.
- Aunque en la web de Git en la zona de descargas podemos encontrar clientes de Git con interfaz gráfica para Git: <https://git-scm.com/downloads/guis/>



4. Instalación de Git

Tarea 3

- Apartado 3.1. Realiza la instalación de Git en tu sistema operativo anfitrión siguiendo las indicaciones dada en el punto anterior.
- Apartado 3.2. Dado que en un futuro también vamos a usar Git dentro de VS Code vamos a aprovechar ya para instalar una serie de extensiones:
 - gitignore
 - Git History
 - Git Graph
 - Open in GitHub, Bitbucket, Gitlab, VisualStudio.com !
 - GitHub Pull Requests
 - GitLens - **Si nos deja, la vamos a dejar deshabilitada.**
Si no nos deja, la vamos a desinstalar.

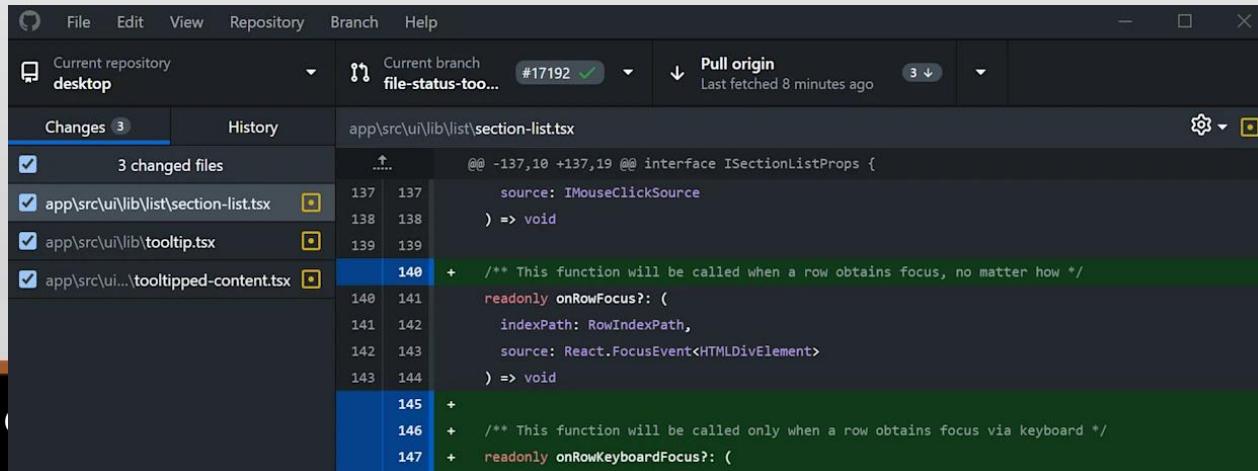


4. Instalación de Git

Tarea 3

- Apartado 3.3. En esta tarea, vamos a instalar **GitHub Desktop**, una aplicación gráfica que facilita la interacción con GitHub.
 - Aunque como he indicado en las prácticas trabajaremos principalmente con Git desde la línea de comandos y Visual Studio Code, es bueno que tengas esta herramienta instalada en caso de que necesites:
 - Visualizar cambios en un repositorio de forma gráfica.
 - Gestionar commits y ramas de manera más intuitiva.
 - Resolver conflictos de forma visual.
 - Sincronizar tu repositorio local con GitHub fácilmente.

<https://github.com/apps/desktop>

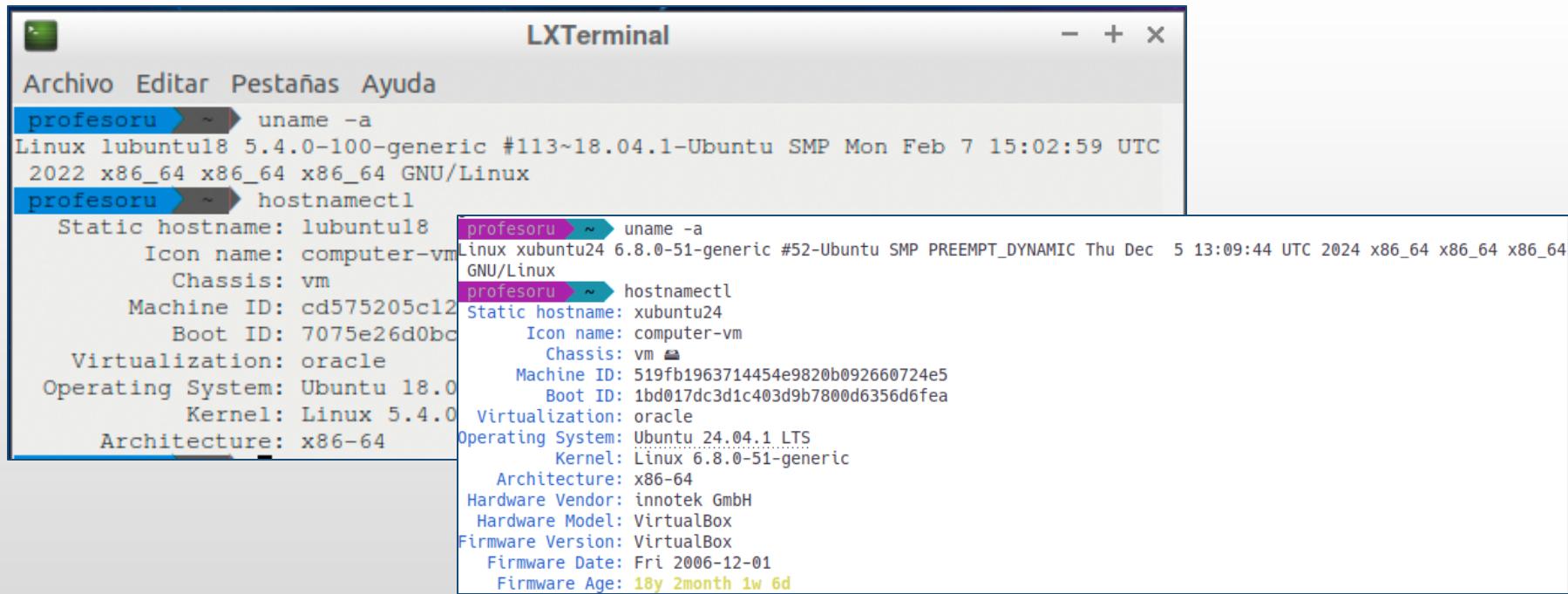


Índice de contenidos

1. Introducción
2. Conceptos
3. Flujo de trabajo
4. Instalación de Git
- 5. Primeros pasos**
6. Uso básico de Git
7. Uso básico de GitHub
8. Ramas en Git y GitHub
9. Trabajo colaborativo
10. Algunos comandos útiles

5. Primeros pasos

Los siguientes ejercicios se han realizado sobre diversas máquinas virtuales creadas con Oracle VirtualBox y con un sistema GNU/Linux.



The screenshot shows an LXTerminal window with two separate sessions. The top session is for a machine named 'profesoru' (Ubuntu 18.04) and the bottom session is for a machine named 'profesoru' (Ubuntu 24.04). Both sessions run the 'uname -a' command and the 'hostnamectl' command. The output for 'uname -a' shows the kernel version, architecture, and boot time. The output for 'hostnamectl' shows the static hostname, icon name, chassis, machine ID, boot ID, virtualization host, operating system, kernel, and architecture. The hardware vendor and model are also listed at the end of the 'hostnamectl' output.

```
profesoru ~ ➤ uname -a
Linux lubuntu18 5.4.0-100-generic #113~18.04.1-Ubuntu SMP Mon Feb 7 15:02:59 UTC
2022 x86_64 x86_64 x86_64 GNU/Linux
profesoru ~ ➤ hostnamectl
Static hostname: lubuntu18
Icon name: computer-vm
Chassis: vm
Machine ID: cd575205c12
Boot ID: 7075e26d0bc
Virtualization: oracle
Operating System: Ubuntu 18.0
Kernel: Linux 5.4.0
Architecture: x86-64
profesoru ~ ➤ uname -a
Linux xubuntu24 6.8.0-51-generic #52-Ubuntu SMP PREEMPT_DYNAMIC Thu Dec 5 13:09:44 UTC 2024 x86_64 x86_64 x86_64
GNU/Linux
profesoru ~ ➤ hostnamectl
Static hostname: xubuntu24
Icon name: computer-vm
Chassis: vm
Machine ID: 519fb1963714454e9820b092660724e5
Boot ID: 1bd017dc3d1c403d9b7800d6356d6fea
Virtualization: oracle
Operating System: Ubuntu 24.04.1 LTS
Kernel: Linux 6.8.0-51-generic
Architecture: x86-64
Hardware Vendor: innotek GmbH
Hardware Model: VirtualBox
Firmware Version: VirtualBox
Firmware Date: Fri 2006-12-01
Firmware Age: 18y 2month 1w 6d
```

Nota: Este terminal tiene instalado OhMyBash: <https://ohmybash.nntoan.com/>

Aviso: Las capturas de pantalla de a continuación, se han tomado en distintos pruebas y puede ser que los Hash ID de los commit de Git o las fechas no coincidan. Debes tener cuidado con eso.

5. Primeros pasos

Documentación oficial de Git: <https://git-scm.com/docs>

Reference

Quick reference guides: [GitHub Cheat Sheet](#) | [Visual Git Cheat Sheet](#)

[Complete list of all commands](#)

- Setup and Config**
 - git
 - config
 - help
 - bugreport
- Getting and Creating Projects**
 - init
 - clone
- Basic Snapshotting**
 - add
 - status
 - diff
 - commit
 - notes
 - restore
 - reset
 - rm
 - mv
- Branching and Merging**
 - branch
 - checkout
 - switch
 - merge
- Guides**
 - gitattributes
 - Command-line interface conventions
 - Everyday Git
 - Frequently Asked Questions (FAQ)
 - Glossary
 - Hooks
 - gitignore
 - gitmodules
 - Revisions
 - Submodules
 - Tutorial
 - Workflows
 - All guides...
- Email**
 - am
 - apply
 - format-patch
 - send-email
 - request-pull
- External Systems**
 - svn
 - fast-import

5. Primeros pasos

- Comprueba que tienes Git instalado y su versión:

```
git --version
```

```
profesoru ~ git --version  
git version 2.36.0
```

```
profesoru ~ git --version  
git version 2.43.0
```



Advertencia

Cuidado al copiar y pegar comandos desde un PDF

Si estás trabajando con Windows, puedes usar Git Bash para hacer estos ejercicios. Git Bash es un **emulador** de terminal que permite ejecutar comandos de Git en **Windows**, proporcionando un entorno similar al de **Linux** o **macOS**. Git Bash usa MinTTY como su terminal predeterminada en Windows. MinTTY es un emulador de terminal liviano y personalizable que proporciona una mejor experiencia en comparación con cmd.exe o PowerShell.

Comprueba que al iniciar lo estás **/c/User/tuUsuario**

```
MINGW64:/c/Users/profesorw ~  
profesorw@MV2W1020 MINGW64 ~  
$ git --version  
git version 2.47.1.windows.2  
profesorw@MV2W1020 MINGW64 ~  
$ |
```

5. Primeros pasos

- Configura la información del usuario para todos los repositorios locales:

```
git config --global user.name "Tu nombre"
```

```
git config --global user.email "Tu email profesional"
```

¿Qué utilidad tiene esto?

```
profesoru ~ ➔ git config --global user.name "A. Montoro"  
profesoru ~ ➔ git config --global user.email "amontoro.sistemasinformaticos@g  
mail.com"
```

- Comprueba que la configuración se ha establecido correctamente:

```
git config --global --list
```

```
profesoru ~ ➔ git config --global --list
```

Nota: Para ver toda la configuración: `git config --list`

5. Primeros pasos

- Establece el uso de colores para facilitar las revisiones en Git:

```
git config --global color.ui auto
```

```
profesoru ~ ➔ git config --global color.ui auto
```

- Establece el nombre de la rama por defecto a 'main' (en lugar de usar 'master'):

```
git config --global init.defaultBranch main
```

```
profesoru ~ ➔ git config --global init.defaultBranch main
```

- Comprueba que la configuración se ha establecido correctamente:

```
git config --global --list
```

Nota: Si te equivocas, edita el fichero `~/.gitconfig`

En Windows: code C:\Users\%USERNAME%\gitconfig

5. Primeros pasos

- Opcional en GNU/Linux:

```
git config --global core.editor nano
```

```
git config --global core.editor "code --wait"
```

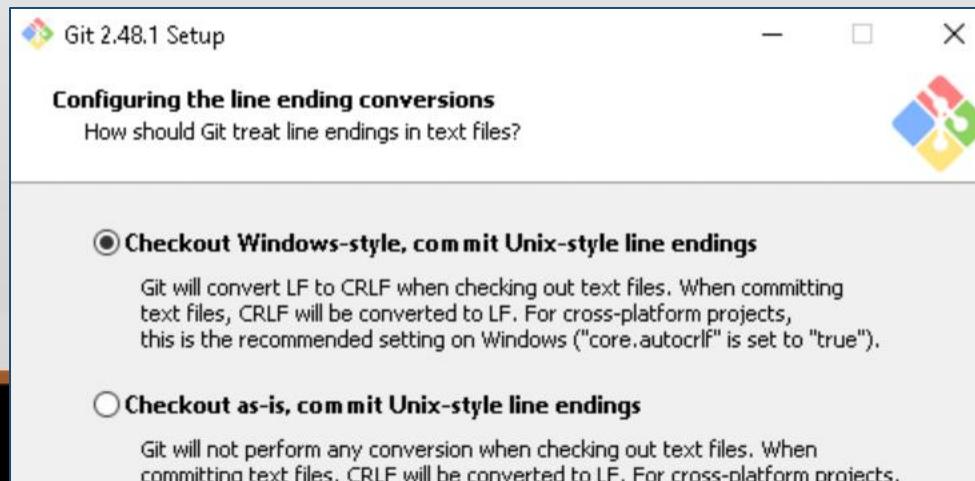
- Si trabajamos con un equipo de desarrollo mixto:

- Si usas Linux: Mantenemos LF, sin cambios en local.

```
git config --global core.autocrlf input
```

- Si usas Windows: Convertimos LF a CRLF al **descargar**. Y CRLF a LF al subir.

```
git config --global core.autocrlf true
```



5. Primeros pasos

- Opcional en GNU/Linux:



¿Qué pasa si el equipo usa herramientas que necesitan LF en Windows?

Si tu sistema anfitrión es Windows y en el repositorio se tienen ficheros manejados por herramientas que sí necesitan LF en Windows (como scripts en bash para WSL o un entorno Linux Dockerizado), entonces usar este comando `git config --global core.autocrlf true` puede traer problemas.

En ese caso, usaremos un fichero `.gitattributes`. (Ver anexo)

5. Primeros pasos

Tarea 4.

- Realiza los primeros pasos explicados en este apartado sobre tu sistema anfitrión.

Nota: Si te equivocas, edita el fichero `~/.gitconfig`

En Windows: `code C:\Users\%USERNAME%\gitconfig`

5.1. GitHub: Creación de cuenta

- Para empezar a trabajar con repositorios remotos más adelante, necesitaremos una cuenta en GitHub.

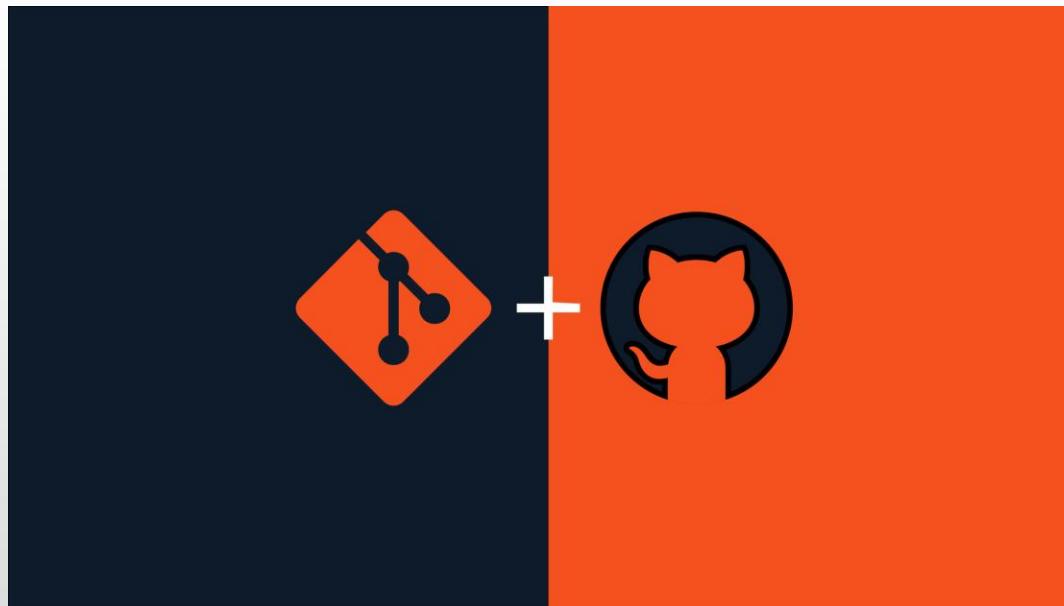


Imagen de: <https://www.google.com/url?sa=i&url=https%3A%2F%2Fseracoder.com%2Fdecoding-git-and-github-an-introductory-handbook-on-version-control-and-collaborative-coding%2F&psig=AOvVaw1SUV0FPVISrceZncFu5vb5&ust=1739486428279000&source=images&cd=vfe&opi=89978449&ved=>

5.1. GitHub: Creación de cuenta

Tarea 5.

- Si no te has creado ya una cuenta en GitHub, créala y verifica tu correo.
 - Usa un correo profesional y anota la contraseña. (Usa el mismo correo que configuraste en Git).
 - Edita tu perfil con tu nombre y foto de perfil, una breve biografía y configura tu ubicación. Añade tus enlaces si lo deseas.
 - Después anota la URL de tu perfil de usuario de GitHub en el documento colaborativo: [pisa aprendizaje 03 e01 grupos](#)
 - Mi perfil es: <https://github.com/amontorosi>

Es recomendable, **más adelante**, activar 2FA (Autenticación en dos pasos) por seguridad. Se activa en: Settings > Password and authentication > Enable two-factor authentication.

5.1. GitHub: Creación de cuenta

Tarea 6.

- En esta tarea, vamos a mejorar nuestro perfil de GitHub agregando un README de presentación.
 - Crear un README en tu perfil:
 - Crea un nuevo repositorio con el mismo nombre que tu usuario (Ejemplo: si tu usuario es gregorioprieto-dev, el repositorio debe llamarse gregorioprieto-dev).
 - Marca la opción "Initialize this repository with a README" antes de crearlo.
 - Edita el archivo README.md y agrega información sobre ti usando MarkDown:
 - ¿Has considerado hacerlo en inglés?
 - Quién eres y qué estudias/trabajas.
 - Agrega una fotografía profesional (o un avatar hasta que decidas qué foto usar).
 - Lenguajes o tecnologías que dominas o estás aprendiendo.
 - Un enlace a tu LinkedIn o portafolio (si tienes).
 - Información de contacto.
 - Muestra TODAS tus contribuciones.
 - Añade Badges: <https://shields.io/>
 - Revisa la siguiente documentación: <https://midu.dev/como-crear-tu-perfil-de-github-con-readme/>
 - Añade algún elemento avanzado siquieres.

Índice de contenidos

1. Introducción
2. Conceptos
3. Flujo de trabajo
4. Instalación de Git
5. Primeros pasos
- 6. Uso básico de Git**
7. Uso básico de GitHub
8. Ramas en Git y GitHub
9. Trabajo colaborativo
10. Algunos comandos útiles

6. Uso básico de Git

- Comprueba que estás en tu directorio de trabajo:

```
profesoru ~ cd ~  
profesoru ~ pwd  
/home/profesoru
```

- Crea un directorio de trabajo, colócate dentro de él y haz un listado largo incluyendo archivos y directorios ocultos:

```
mkdir si_sh_repositorio_01  
cd si_sh_repositorio_01  
ls -la
```

```
profesoru ~ mkdir si_sh_repositorio_01  
mkdir: se ha creado el directorio 'si_sh_repositorio_01'  
profesoru ~ cd si_sh_repositorio_01/  
/home/profesoru/si_sh_repositorio_01  
profesoru ~/si_sh_repositorio_01 ls -la  
total 8  
drwxrwxr-x 2 profesoru profesoru 4096 Feb 15 20:21 .  
drwxr-x--- 17 profesoru profesoru 4096 Feb 15 20:21 ..  
profesoru ~/si_sh_repositorio_01
```

Si estás en Windows:

```
profesorw@MV2W1020 MINGW64 ~  
$ cd ~  
  
profesorw@MV2W1020 MINGW64 ~  
$ pwd  
/c/Users/profesorw  
  
profesorw@MV2W1020 MINGW64 ~  
$ mkdir si_sh_repositorio_01  
  
profesorw@MV2W1020 MINGW64 ~  
$ cd si_sh_repositorio_01  
  
profesorw@MV2W1020 MINGW64 ~/si_sh_repositorio_01  
$ ls -la  
total 8  
drwxrwxr-x 1 profesorw 197121 0 Feb 15 20:25 ./  
drwxrwxr-x 1 profesorw 197121 0 Feb 15 20:25 ../
```

6. Uso básico de Git

- Inicia un repositorio local en ese directorio de trabajo y haz un listado largo incluyendo archivos y directorios ocultos:

```
git init  
ls -la
```

```
profesoru ~ /si_sh_repositorio_01 git init  
Inicializado repositorio Git vacío en /home/profesoru/si_sh_repositorio_01/.git/  
profesoru ~ /si_sh_repositorio_01 ls -la  
total 12  
drwxrwxr-x 3 profesoru profesoru 4096 abr 19 19:21 .  
drwxr-xr-x 25 profesoru profesoru 4096 abr 19 19:18 ..  
drwxrwxr-x 7 profesoru profesoru 4096 abr 19 19:21 .git  
profesoru ~
```

6. Uso básico de Git

- Crea dos ficheros en blanco dentro de ese directorio y ya que estamos vamos a darle permisos de ejecución a ambos:

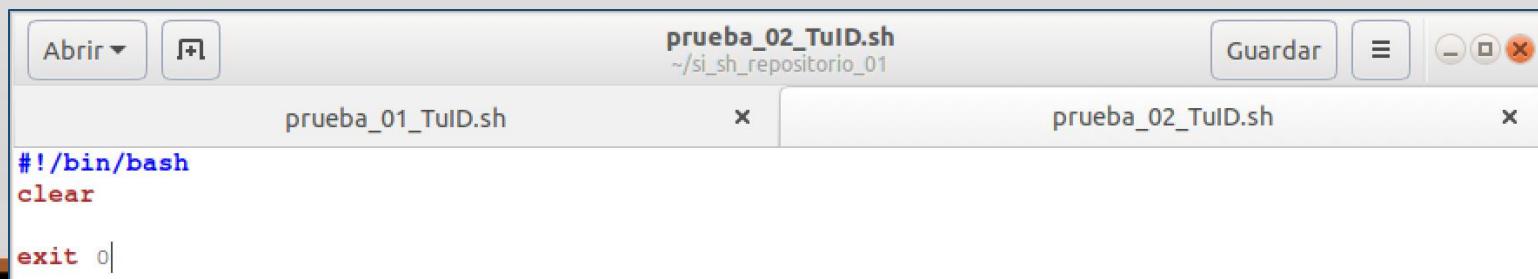
```
touch prueba_01_TuID.sh prueba_02_TuID.sh  
chmod u+x p*.sh
```

Nota: En Git Bash de Windows los permisos son emulados.
Por tanto, chmod no hace nada en Git Bash.

- Edita el contenido de esos dos ficheros con el 'esqueleto' básico de un script de bash:

```
#!/bin/bash  
  
clear  
  
exit 0
```

Nota: En Git Bash de Windows al comenzar el fichero con #!/bin/bash, asigna automáticamente el permiso de ejecución.



6. Uso básico de Git

- Comprueba el estado del seguimiento que se está haciendo a los ficheros creados. Analiza la información que te aparece:

```
git status
```

```
git status -s
```

```
profesoru ~ main 7:2 ~/si_sh_repositorio_01 git status
En la rama main

No hay commits todavía

Archivos sin seguimiento:
(usa "git add <archivo>..." para incluirlo a lo que se será confirmado)
    prueba_01_TuID.sh
    prueba_02_TuID.sh

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

```
profesoru ~ main ?:2 ~/si_sh_repositorio_01 git status -s
?? prueba_01_TuID.sh
?? prueba_02_TuID.sh
```

6. Uso básico de Git

- Recuerda:

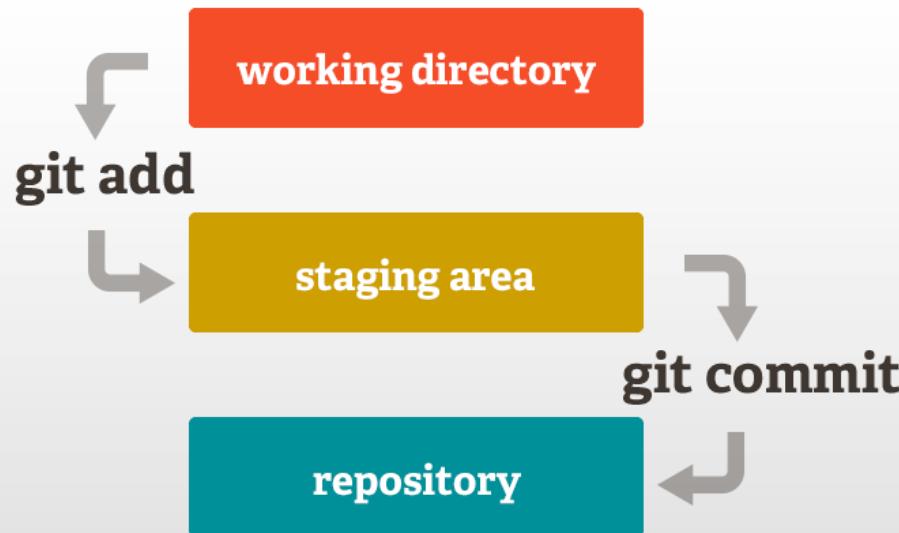


Imagen extraída de: <https://git-scm.com/>

6. Uso básico de Git

- Realiza un seguimiento del primer script añadiéndolo al staging area y comprueba el estado del seguimiento:

```
git add prueba_01_TuID.sh
```

```
git status
```

```
git status -s
```

```
profesoru ~/si_sh_repositorio_01 git add prueba_01_TuID.sh
profesoru ~/si_sh_repositorio_01 git status -s
A prueba_01_TuID.sh
?? prueba_02_TuID.sh
profesoru ~/si_sh_repositorio_01 git status
En la rama main

No hay commits todavía

Cambios a ser confirmados:
(usa "git rm --cached <archivo>..." para sacar del área de stage)
    nuevos archivos: prueba_01_TuID.sh

Archivos sin seguimiento:
(usa "git add <archivo>..." para incluirlo a lo que se será confirmado)
    prueba_02_TuID.sh
```

¿Cómo podría dejar de seguir al fichero prueba_01_TuID.sh y sacarlo de staging area?
¿A qué área iría?

6. Uso básico de Git

- Confirma esos cambios añadiendo los ficheros modificados del seguimiento al repositorio:

```
git commit -m "Incluyendo primer script"
```

```
git status -s
```

```
git status
```

```
profesoru ➤ main S:1 ?:1 ➤ ~/si_sh_repositorio_01 ➤ git commit -m "Incluyendo
primer script"
[main (commit-raiz) c95315c] Incluyendo primer script
 1 file changed, 4 insertions(+)
 create mode 100755 prueba_01_TUID.sh
profesoru ➤ main ?:1 ➤ ~/si_sh_repositorio_01 ➤ git status -s
?? prueba_02_TUID.sh
profesoru ➤ main ?:1 ➤ ~/si_sh_repositorio_01 ➤ git status
En la rama main
Archivos sin seguimiento:
  (usa "git add <archivo>..." para incluirlo a lo que se será confirmado)
    prueba_02_TUID.sh

no hay nada agregado al commit pero hay archivos sin seguimiento presentes (usa
"git add" para hacerles seguimiento)
profesoru ➤ main ?:1 ➤ ~/si_sh_repositorio_01 ➤
```

6. Uso básico de Git

- Analiza la información que nos da por pantalla.

¿Qué es el número c95315c? ¿es el número completo?

¿Qué ocurre con prueba_02_TuID.sh

```
profesoru ➤ main S:1 ?:1 ➤ ~/si_sh_repositorio_01 ➤ git commit -m "Incluyendo
primer script"
[main (commit-raíz) c95315c] Incluyendo primer script
 1 file changed, 4 insertions(+)
 create mode 100755 prueba_01_TuID.sh
profesoru ➤ main ?:1 ➤ ~/si_sh_repositorio_01 ➤ git status -s
?? prueba_02_TuID.sh
profesoru ➤ main ?:1 ➤ ~/si_sh_repositorio_01 ➤ git status
En la rama main
Archivos sin seguimiento:
  (usa "git add <archivo>..." para incluirlo a lo que se será confirmado)
    prueba_02_TuID.sh

no hay nada agregado al commit pero hay archivos sin seguimiento presentes (usa
"git add" para hacerles seguimiento)
profesoru ➤ main ?:1 ➤ ~/si_sh_repositorio_01 ➤
```

6. Uso básico de Git

- Si quisieras editar el mensaje de ese último commit puedes hacerlo con :

```
git commit --amend
```

```
/home/profesoru/si_sh_repositorio_01/.git/COMMIT_EDITMSG      Modificado

Incluyendo primer script

# Por favor ingresa el mensaje del commit para tus cambios. Las
# líneas que comiencen con '#' serán ignoradas, y un mensaje
# vacío aborta el commit.
#
# Fecha:      Fri May 20 22:38:53 2022 +0200
#
# En la rama main
#
# Confirmación inicial
#
# Cambios a ser confirmados:
#     nuevos archivos: prueba_01_TuID.sh
#
# Cambios no rastreados para el commit:
#     modificados:     prueba_01_TuID.sh
#
^G Ver ayuda ^O Guardar ^W Buscar ^K Cortar Tex^J Justificar^C Posición
^X Salir      ^R Leer fich.^` Reemplazar^U Pegar txt ^T Ortografia^_ Ir a línea
```

6. Uso básico de Git

- Muestra el historial de todos los commit que hemos realizado:

```
git log
```

```
commit c69979e7b3d990138226f8e97e431d6a66a57b5b (HEAD -> main)
Author: A. Montoro <amontoro.sistemasinformaticos@gmail.com>
Date:   Fri May 20 22:38:53 2022 +0200

    Incluyendo primer script
(END)
```

6. Uso básico de Git

- ¿Cómo funciona esto?:

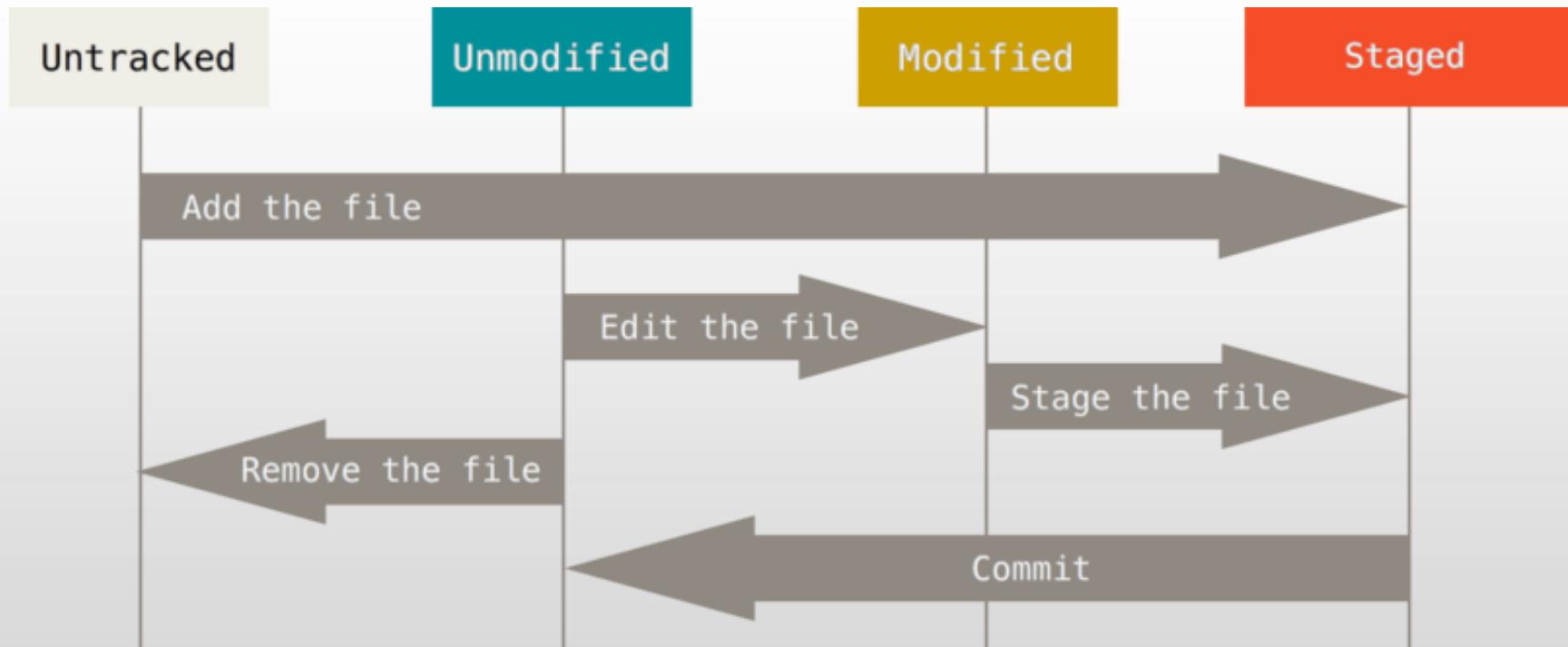
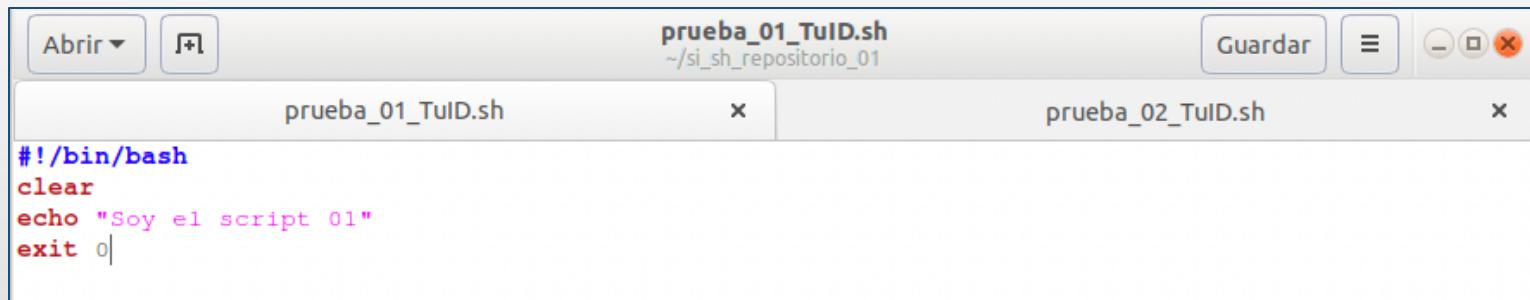


Imagen extraída de: <https://git-scm.com/about/>

6. Uso de básico de Git

- Edita de nuevo ese primer script:

```
#!/bin/bash
clear
echo "Soy el script 01"
exit 0
```



A screenshot of a terminal window with two tabs. The left tab is titled 'prueba_01_TuID.sh' and contains the script code shown above. The right tab is titled 'prueba_02_TuID.sh' and is currently empty. The window has standard OS X-style controls at the top.

```
#!/bin/bash
clear
echo "Soy el script 01"
exit 0
```

6. Uso básico de Git

- Comprueba el estado del seguimiento:

```
git status -s
```

```
git status
```

```
profesoru ~ main ?:1 ~/si_sh_repositorio_01 git status -s
M prueba_01_TuID.sh
?? prueba_02_TuID.sh
profesoru ~ main U:1 ?:1 ~/si_sh_repositorio_01 git status
En la rama main
Cambios no rastreados para el commit:
  (usa "git add <archivo>..." para actualizar lo que será confirmado)
  (usa "git restore <archivo>..." para descartar los cambios en el directorio de trabajo)
      modificados:     prueba_01_TuID.sh

Archivos sin seguimiento:
  (usa "git add <archivo>..." para incluirlo a lo que se será confirmado)
      prueba_02_TuID.sh

sin cambios agregados al commit (usa "git add" y/o "git commit -a")
```

6. Uso básico de Git

- Consulta las diferencias de ese fichero en el **working directory** con la versión en el **staging area**, es decir los cambios que aún no han sido añadidos con git add:

```
git diff prueba_01_TuID.sh
```

```
diff --git a/prueba_01_TuID.sh b/prueba_01_TuID.sh
index e31e908..0d51392 100755
--- a/prueba_01_TuID.sh
+++ b/prueba_01_TuID.sh
@@ -1,4 +1,4 @@
#!/bin/bash
clear
-
+echo "Soy el script 01"
 exit 0
(END)
```

- Consulta las diferencias de todos los ficheros (que aún no han sido añadidos con git add):

```
git diff
```

6. Uso básico de Git

- Para mostrar los cambios que serán incluidos en el próximo commit, es decir, las diferencias entre el staging area y el último commit:

```
git diff --staged prueba_01_TuID.sh
```

Nota: Si ya hiciste git commit, entonces git diff no mostrará nada porque no hay diferencias pendientes. Para ver la diferencia entre commits, usa:

```
git diff HEAD~1
```

Conclusión:

git diff: solo muestra los cambios que aún no han sido añadidos con git add. Si no ves ninguna salida en la terminal, significa que no hay diferencias entre el working directory y el área de staging.

git diff --staged: muestra los que sí están en staging pero aún no se han comiteado

6. Uso básico de Git

- Intenta hacer un commit:

```
git commit -m "Modificado el primer script"
```

```
profesoru ➜ main 0:1 7:1 ➤ ~/si_sh_repositorio_01 ➤ git commit -m "Modificado  
el primer script"  
En la rama main  
Cambios no rastreados para el commit:  
  (usa "git add <archivo>..." para actualizar lo que será confirmado)  
  (usa "git restore <archivo>..." para descartar los cambios en el directorio de  
trabajo)  
    modificados:     prueba_01_TuID.sh  
  
Archivos sin seguimiento:  
  (usa "git add <archivo>..." para incluirlo a lo que se será confirmado)  
    prueba_02_TuID.sh  
  
sin cambios agregados al commit (usa "git add" y/o "git commit -a")
```

¿Qué ocurre?

6. Uso básico de Git

- Muestra el historial del commit que **NO** hemos realizado:

```
git log
```

```
commit c69979e7b3d990138226f8e97e431d6a66a57b5b (HEAD -> main)
Author: A. Montoro <amontoro.sistemasinformaticos@gmail.com>
Date:   Fri May 20 22:38:53 2022 +0200

    Incluyendo primer script
(END)
```

6. Uso básico de Git

- Añade de nuevo el primer script al staging area:

```
git add prueba_01_TuID.sh
```

```
git status -s
```

```
git status
```

```
profesoru ~ main U:1 ?:1 ~/si_sh_repositorio_01 git add prueba_01_TuID.sh
profesoru ~ main S:1 ?:1 ~/si_sh_repositorio_01 git status -s
M prueba_01_TuID.sh
?? prueba_02_TuID.sh
profesoru ~ main S:1 ?:1 ~/si_sh_repositorio_01 git status
En la rama main
Cambios a ser confirmados:
(usa "git restore --staged <archivo>..." para sacar del área de stage)
    modificados: prueba_01_TuID.sh

Archivos sin seguimiento:
(usa "git add <archivo>..." para incluirlo a lo que se será confirmado)
    prueba_02_TuID.sh
```

6. Uso básico de Git

- Intenta hacer un commit de nuevo:

```
git commit -m "Modificado el primer script"
```

```
profesoru ~ main S:1 ?:1 ~/si_sh_repositorio_01 git commit -m "Modificado el primer script"
[main f0f2c7b] Modificado el primer script
 1 file changed, 1 insertion(+), 1 deletion(-)
profesoru ~ main ?:1 ~/si_sh_repositorio_01
```

6. Uso básico de Git

- Muestra el historial de todos los commit que hemos realizado:

```
git log
```

```
git log --oneline
```

```
git log --graph
```

```
git log --graph --oneline --all --decorate
```

```
commit f0f2c7b0c3bc293ea810ca9842ea4f3a84722a7f (HEAD -> main)
Author: A. Montoro <amontoro.sistemasinformaticos@gmail.com>
Date:   Tue Apr 19 20:08:05 2022 +0200
```

```
    Modificado el primer script
```

```
commit c95315c81c58d082b640d35dfdde7c7ea2566de3
Author: A. Montoro <amontoro.sistemasinformaticos@gmail.com>
Date:   Tue Apr 19 19:47:46 2022 +0200
```

```
    Incluyendo primer script
```

```
(END)
```

```
f0f2c7b (HEAD -> main) Modificado el primer script
c95315c Incluyendo primer script
(END)
```

6. Uso de básico de Git

- Imagina que probamos el script 1 y nos damos cuenta de que las líneas que hemos añadido han empeorado el rendimiento del mismo y que queremos volver al commit anterior descartando esos cambios y que además, queremos borrar todos esos cambios que hicimos.
- Para ello haremos uso del comando:

```
git reset --hard hash_id_al_que_queremos_ir
```

6. Uso básico de Git

- Recupera el estando en el primer commit, muestra el contenido del fichero y muestra el historial de todos los commit (en versión corta):

```
git reset --hard c95315c  
cat prueba_01_TuID.sh  
git log --oneline
```

```
profesoru ~ main ?:1 ~/si_sh_repositorio_01 cat prueba_01_TuID.sh  
#!/bin/bash  
clear  
echo "Soy el script 01"  
exit 0  
profesoru ~ main ?:1 ~/si_sh_repositorio_01 git reset --hard c95315c  
HEAD está ahora en c95315c Incluyendo primer script  
profesoru ~ main ?:1 ~/si_sh_repositorio_01 cat prueba_01_TuID.sh  
#!/bin/bash  
clear  
  
exit 0
```

```
profesoru ~ main ?:1 ~/si_sh_repositorio_01 git log --oneline
```

```
c95315c (HEAD -> main) Incluyendo primer script  
(END)
```

6. Uso básico de Git

- Recupera el estado del primer commit:

- **MUY IMPORTANTE:** El comando git reset tiene tres modos de funcionamiento:
 - **HARD (--hard):** En este caso, se vuelve al commit anterior y todos los cambios que habíamos hecho después de ese commit se pierden. Deja el working directory completamente limpio.
 - **SOFT (--soft):** En este caso, se vuelve al commit anterior y todos los ficheros que han cambiado entre ese commit y el estado actual, pasan al staging area. Esos cambios no desaparecen. Por decirlo de alguna manera, los cambios los 'sacamos' del commit.
 - **MIX (sin argumentos o --mixed):** Lo mismo que SOFT pero dejando los cambios directamente en el working directory.

6. Uso básico de Git

- Muestra todos los movimientos de HEAD, incluyendo commits eliminados o movidos:

```
git reflog
```

También se pueden ver solo los commits de una rama específica:

```
git reflog show main
```

```
c5a36c5 (HEAD -> main) HEAD@{0}: reset: moving to c5a36c  
f52f0e2 HEAD@{1}: commit: Modificado el primer script  
c5a36c5 (HEAD -> main) HEAD@{2}: commit (amend): Incluyendo primer script  
3e6e63e HEAD@{3}: commit (initial): Incluyendo primer script  
(END)
```

6. Uso básico de Git

- Modifica de nuevo el fichero prueba_01_TuID.sh y guarda cambios:

```
code prueba_01_TuID.sh &
```

```
1 #!/bin/bash
2
3 clear
4 nombre="Tu nombre"
5
6 exit 0
```

- Prueba los siguientes comandos. ¿Qué estamos haciendo?:

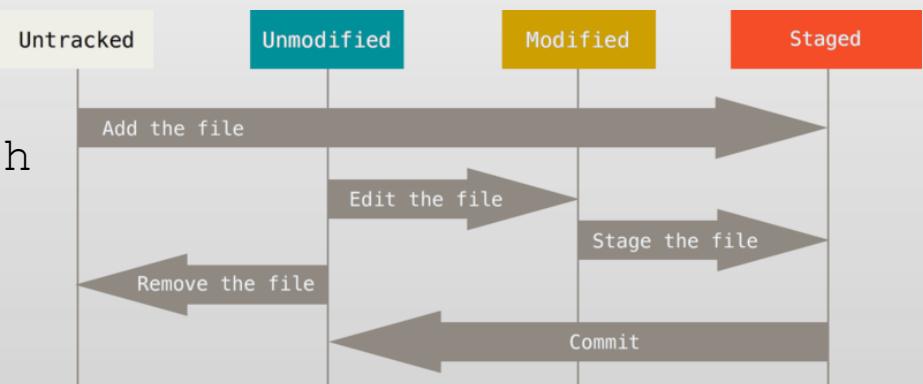
```
git diff
```

```
git status
```

```
git restore prueba_01_TuID.sh
```

```
cat prueba_01_TuID.sh
```

```
git status
```



git restore solo funciona si el fichero **NO** está en el Stage Area

6. Uso básico de Git

- Vuelve a modificar de nuevo el fichero prueba_01_TuID.sh:

```
code prueba_01_TuID.sh &
```

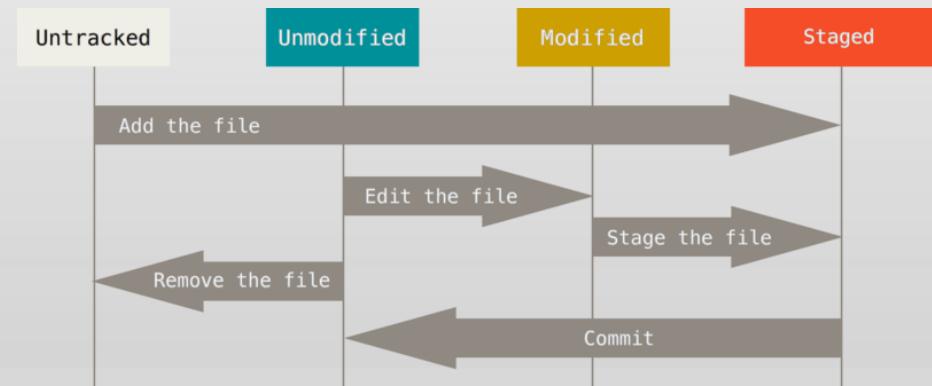
```
1 #!/bin/bash
2
3 clear
4 nombre="Tu nombre"
5 echo "Hola $nombre"
6 exit 0
```

- Prueba los siguientes comandos. ¿Qué ocurre?:

```
git diff
git status
git add prueba_01_TuID.sh
git status -s
git restore prueba_01_TuID.sh
cat prueba_01_TuID.sh
```

#**¿Qué ocurre?**

```
git reset HEAD prueba_01_TuID.sh
git restore prueba_01_TuID.sh
cat prueba_01_TuID.sh
```



6. Uso básico de Git

- Añade todos los ficheros al staging area:

```
git add .
```

También se consigue lo mismo con: git add -A

```
git status -s
```

```
profesoru ~ main ?:1 ~/si_sh_repositorio_01 git add .
profesoru ~ main S:1 ~/si_sh_repositorio_01 git status -s
A prueba_02_TuID.sh
```

Aviso: Ten en cuenta que las carpetas vacías se ignoran siempre.

Nota: Si no querías añadir todos los documentos puede deshacer esto con:

```
git reset .
```

Si solo quieres descartar un fichero concreto:

```
git reset nombre_fichero
```

6. Uso básico de Git

- Haz un commit con el siguiente mensaje:

```
git commit -m "Dos scripts"
```

```
profesoru ~ main S:1 ~/si_sh_repositorio_01 git commit -m "Dos script"
[main cef1d5e] Dos script
 1 file changed, 4 insertions(+)
 create mode 100755 prueba_02_TuID.sh
profesoru ~ main ~/si_sh_repositorio_01
```

6. Uso de básico de Git

- Modifica los dos ficheros en la línea tres con estos mensajes respectivamente:

```
echo "Soy el script 02"
```

```
echo "Soy el script 02"
```

The screenshot shows a dual-pane terminal window. The left pane contains the file `prueba_01_TuID.sh` with the following content:

```
#!/bin/bash
clear
echo "Soy el script 01"
exit 0
```

The right pane contains the file `prueba_02_TuID.sh` with the following content:

```
#!/bin/bash
clear
echo "Soy el script 02"
exit 0
```

This screenshot shows the same dual-pane terminal window after the modifications have been made. The left pane (`prueba_01_TuID.sh`) now displays:

```
#!/bin/bash
clear
echo "Soy el script 01"
exit 0
```

The right pane (`prueba_02_TuID.sh`) now displays:

```
#!/bin/bash
clear
echo "Soy el script 02"
exit 0
```

6. Uso básico de Git

- Comprueba el estado del seguimiento:

```
git status -s
```

```
git status
```

```
profesoru ~/main ~/si_sh_repositorio_01 git status -s
M prueba_01_TuID.sh
M prueba_02_TuID.sh
profesoru ~/main U:2 ~/si_sh_repositorio_01 git status
En la rama main
Cambios no rastreados para el commit:
  (usa "git add <archivo>..." para actualizar lo que será confirmado)
  (usa "git restore <archivo>..." para descartar los cambios en el directorio de trabajo)
    modificados: prueba_01_TuID.sh
    modificados: prueba_02_TuID.sh

sin cambios agregados al commit (usa "git add" y/o "git commit -a")
```

6. Uso básico de Git

- Prueba el comando git commit con la opción -a:

```
git commit -am "Añadidos mensajes a los dos scripts"
```

```
profesoru ➜ main U:2 ➜ ~/si_sh_repositorio_01 ➜ git commit -am "Añadidos mensajes a los dos script"
[main 9c6bdcaa] Añadidos mensajes a los dos script
 2 files changed, 2 insertions(+), 2 deletions(-)
```

¿Qué estamos haciendo?

6. Uso básico de Git

- Muestra el historial de todos los commit que hemos realizado:

```
git log
```

```
git log --oneline
```

```
9c6bdaa (HEAD -> main) Añadidos mensajes a los dos script
cef1d5e Dos script
c95315c Incluyendo primer script
```

- Muestra los detalles de un commit específico para ver exactamente qué cambió en el commit:

```
git show 9c6bdaa
```

6. Uso básico de Git

- También podemos movernos entre commits en modo "solo lectura". A diferencia de lo que hemos hecho con git --reset.

```
git checkout b7067c
```

```
ls -la
```

```
profesoru ~/si_sh_repositorio_01 main git checkout b7067cd
Nota: cambiando a 'b7067cd'.

Te encuentras en estado 'detached HEAD'. Puedes revisar por aquí, hacer cambios experimentales y hacer commits, y puedes descartar cualquier commit que hayas hecho en este estado sin impactar a tu rama realizando otro checkout.

Si quieres crear una nueva rama para mantener los commits que has creado, puedes hacerlo (ahora o después) usando -c con el comando checkout. Ejemplo:

git switch -c <nombre-de-nueva-rama>

O deshacer la operación con:

git switch -

Desactiva este aviso poniendo la variable de config advice.detachedHead en false

HEAD está ahora en b7067cd Añadido ficheros_conf.txt
profesoru ~/si_sh_repositorio_01 - b7067cd
```

- Para volver:

```
git checkout main
```

```
ls -la
```

6. Uso básico de Git

- Prueba los siguientes comandos y determina qué está pasando:

```
echo "Copiar: cp ORIGEN DESTINO" > comandos.txt  
echo "Fichero redes: netplan" > ficheros_conf.txt
```

```
ls -l  
git add comandos.txt  
git commit -m "Añadido comandos.txt"  
git rm comandos.txt #Elimina el archivo del repo y del disco  
git status  
git commit -m "Eliminado comandos.txt"  
git status  
ls -l
```

¿Podría recuperar el fichero borrado si vuelvo al commit anterior? SÍ: git restore --source=HEAD~1 comandos.txt

6. Uso básico de Git

- Prueba los siguientes comandos y determina qué está pasando:

```
git add ficheros_conf.txt
```

```
git commit -m "Añadido ficheros_conf.txt"
```

```
git rm --cached ficheros_conf.txt #Elimina el archivo solo  
del repo. Lo mantienen en el disco
```

```
git status
```

```
git commit -m "Eliminado ficheros_conf.txt"
```

```
git status
```

```
ls -l
```

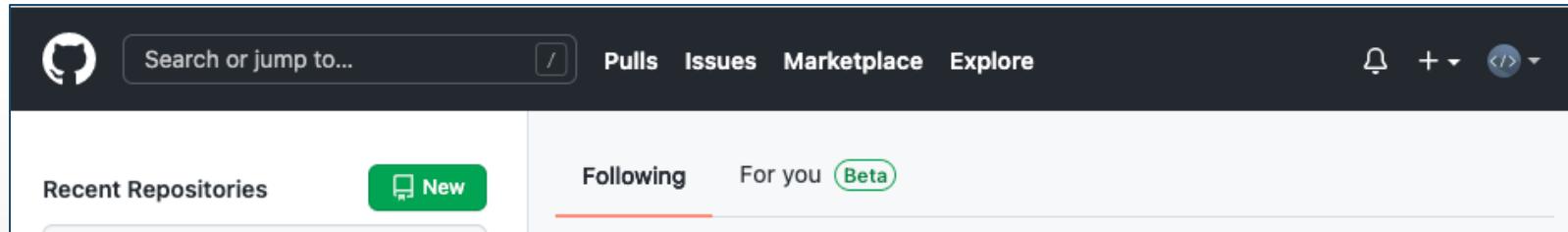
```
rm -f "ficheros_conf.txt" #No es git
```

Índice de contenidos

1. Introducción
2. Conceptos
3. Flujo de trabajo
4. Instalación de Git
5. Primeros pasos
6. Uso básico de Git
- 7. Uso básico de GitHub**
8. Ramas en Git y GitHub
9. Trabajo colaborativo
10. Algunos comandos útiles

7. Uso básico de GitHub

- Accede a GitHub y crea un nuevo repositorio:



- Vamos a crear con el nombre `si_sh_repositorio_01`, público, sin `readme`, sin `.gitignore` y sin licencia.

Interesante: El nombre del repositorio en GitHub no tiene por qué coincidir con el nombre del repositorio en local. Aunque sea lo más habitual.

7. Uso básico de GitHub

- Accede a GitHub y crea un nuevo repositorio:

The screenshot shows the GitHub 'Create repository' interface. The 'Owner' field is set to 'amontorosi'. The 'Repository name' field contains 'si_sh_repositorio_01', which is highlighted with a green checkmark. A note below suggests using short and memorable names like 'ubiquitous-telegram'. The 'Description (optional)' field is empty. The 'Visibility' section shows 'Public' selected (indicated by a purple dot) and 'Private' (indicated by an empty circle). The 'Public' option is described as allowing anyone on the internet to see the repository. The 'Initialize this repository with:' section includes a checkbox for 'Add a README file', which is unchecked. Below it, a note says 'Skip this step if you're importing an existing repository.' The 'Add .gitignore' section shows a dropdown menu set to 'None'. The 'Choose a license' section also has a dropdown menu set to 'None'. A blue info box at the bottom states, 'You are creating a public repository in your personal account.' At the bottom right is a green 'Create repository' button.

Owner * Repository name *

amontorosi / si_sh_repositorio_01 ✓

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

Description (optional)

 **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

7. Uso básico de GitHub

- Analiza la venta que te aparece a continuación:

The screenshot shows a GitHub repository page for 'amontorosi/si_sh_repositorio_01'. The repository is public. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. A 'Quick setup' section is highlighted in blue, containing instructions for cloning the repository via Desktop (with icons for Set up in Desktop, HTTPS, and SSH) or cloning it via a provided URL (https://github.com/amontorosi/si_sh_repositorio_01.git). Below this, there's a note about creating a new file or uploading an existing one, and a recommendation to include README, LICENSE, and .gitignore files. Three sections provide command-line instructions: 1) '...or create a new repository on the command line' with the following commands:
echo "# si_sh_repositorio_01" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/amontorosi/si_sh_repositorio_01.git
git push -u origin main
2) '...or push an existing repository from the command line' with the following commands:
git remote add origin https://github.com/amontorosi/si_sh_repositorio_01.git
git branch -M main
git push -u origin main

7. Uso básico de GitHub

- Analiza la venta que te aparece a continuación:

¿Qué es origin? ¿Y el main?

¿Qué hace el comando git remote?

¿Qué hace el comando git push?

¿Qué hace la opción -u en el comando anterior?

¿Cuál sería el comando inverso a git push?

7. Uso de básico de GitHub

- Como nosotros ya tenemos un repositorio en local, tenemos que optar por lo que nos indica aquí:

...or push an existing repository from the command line

```
git remote add origin https://github.com/amontorosi/si_sh_repositorio_01.git  
git branch -M main  
git push -u origin main
```

Copiar y ejecutar línea a línea.

7. Uso básico de GitHub

- Como nosotros ya tenemos un repositorio en local, tenemos que optar por lo que nos indica aquí:

```
git remote add origin
```

```
https://github.com/amontorosi/si_sh_repositorio_01.git
```

```
git branch -M main
```

```
git push -u origin main
```

```
profesoru ~ si_sh_repositorio_01 git remote add origin https://github.com/amontorosi/si_sh_repositorio_01.git
```

```
profesoru ~ si_sh_repositorio_01 git branch -M main  
profesoru ~ si_sh_repositorio_01 git push -u origin main  
Username for 'https://github.com':
```

Si tuvieras que modificar el repositorio remoto:

```
git remote set-url origin https://github.com/amontorosi/si_sh_repositorio_03.git
```

7. Uso básico de GitHub

- Como nosotros ya tenemos un repositorio en local, tenemos que optar por lo que nos indica aquí:

```
Password for 'https://amontorosi@github.com':  
remote: Support for password authentication was removed on August 13, 2021. Please  
use a personal access token instead.  
remote: Please see https://github.blog/2020-12-15-token-authentication-requirements-for-git-operations/ for more information.  
fatal: Autenticación falló para 'https://github.com/amontorosi/si_sh_repositorio_01.git'  
profesoru ➜ main ➜ ~/si_sh_repositorio_01 ➜ 128 ➜
```

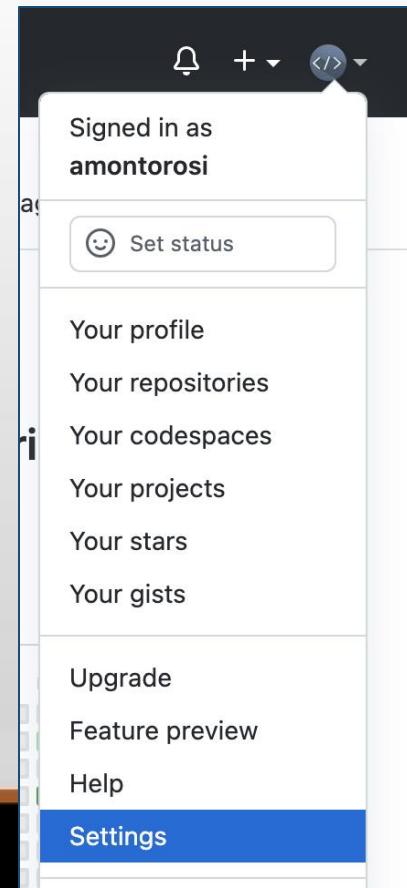
¿Por qué falla?

Nota: Si estás haciendo las prácticas en tu equipo anfitrión y ya has trabajado con Git y GitHub desde un IDE como VS Code en algún momento habrás almacenado tus credenciales y esto no pasará.

7. Uso básico de GitHub

- Vamos crear el token de autenticación en GitHub.
- En el siguiente enlace se explicar cómo hacerlo:

<https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token>



7. Uso básico de GitHub

- Vamos crear el token de autenticación en GitHub.
- Para ello vamos a 'Settings' y a 'Developer settings':

The screenshot shows the 'Personal access tokens' section of the GitHub Developer settings. On the left, there is a sidebar with three options: 'GitHub Apps', 'OAuth Apps', and 'Personal access tokens'. The 'Personal access tokens' option is highlighted with a red border. The main area has a heading 'Personal access tokens' and a 'Generate new token' button. Below the heading, there is a paragraph explaining what personal access tokens are and how they can be used. A link to generate a token is also provided.

Settings / Developer settings

GitHub Apps

OAuth Apps

Personal access tokens

Personal access tokens

Generate new token

Need an API token for scripts or testing? [Generate a personal access token](#) for quick access to the [GitHub API](#).

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

<https://github.com/settings/tokens>

Usa *classic*

7. Uso básico de GitHub

- Después pulsamos en generar nuevo un nuevo token.
- Le damos un nombre, una fecha de expiración (final de curso), marcamos todo y finalmente generamos el token:

New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Note

What's this token for?

Expiration *

Custom...

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input type="checkbox"/> repo:status	Access commit status
<input type="checkbox"/> repo_deployment	Access deployment status
<input type="checkbox"/> public_repo	Access public repositories
<input type="checkbox"/> repo:invite	Access repository invitations
<input type="checkbox"/> security_events	Read and write security events
<input checked="" type="checkbox"/> workflow	Update GitHub Action workflows
<input checked="" type="checkbox"/> write:packages	Upload packages to GitHub Package Registry
<input type="checkbox"/> read:packages	Download packages from GitHub Package Registry
<input checked="" type="checkbox"/> delete:packages	Delete packages from GitHub Package Registry

7. Uso básico de GitHub

- Después pulsamos en generar nuevo un nuevo token.
- Le damos un nombre, una fecha de expiración (final de curso), marcamos todo y finalmente generamos el token:

New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Note
si_21_22

What's this token for?

Expire date
Custom

Select scopes
Scopes

workflow Update GitHub Action workflows

write:packages Upload packages to GitHub Package Registry

read:packages Download packages from GitHub Package Registry

delete:packages Delete packages from GitHub Package Registry

Muy importante: Copia y pega ese token en un fichero de texto al que tengas acceso inmediato y guarda los cambios puesto que debemos tenerlo disponible dado que nos va a hacer falta más adelante.

7. Uso básico de GitHub

- Y volvemos al paso anterior:

```
profesoru ~ main ~/si_sh_repositorio_01 128 git push -u origin main
Username for 'https://github.com': amontorosi
Password for 'https://amontorosi@github.com':
Enumerando objetos: 9, listo.
Contando objetos: 100% (9/9), listo.
Compresión delta usando hasta 2 hilos
Comprimiendo objetos: 100% (5/5), listo.
Escribiendo objetos: 100% (9/9), 880 bytes | 880.00 KiB/s, listo.
Total 9 (delta 0), reusados 0 (delta 0), pack-reusados 0
To https://github.com/amontorosi/si_sh_repositorio_01.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
profesoru ~ main ~/si_sh_repositorio_01
```

Recuerda: Almacena ese token en un fichero de texto porque nos va a hacer falta más adelante.

7. Uso básico de GitHub

- Y ahora accede a GitHub para comprobar que nuestro repositorio se ha sincronizado:

The screenshot shows a GitHub repository page for 'amontorosi / si_sh_repositorio_01'. The repository is private. At the top, there are buttons for 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Security', 'Insights', and 'Settings'. Below these are buttons for 'main' (branch), '1 branch', '0 tags', 'Go to file', 'Add file', and 'Code'. The 'Code' button is highlighted in green. To the right is an 'About' section with the message 'No description, website, or topics provided.' Below this are statistics: '0 stars', '1 watching', and '0 forks'. A blue call-to-action button at the bottom left says 'Add a README'.

amontorosi / si_sh_repositorio_01 Private

Code Issues Pull requests Actions Projects Security Insights Settings

main 1 branch 0 tags Go to file Add file Code

About

No description, website, or topics provided.

0 stars 1 watching 0 forks

Add a README

7. Uso básico de GitHub

- Navega por GitHub y observa como esta herramienta web gestiona el repositorio Git:

The screenshot shows a GitHub repository page for 'amontorosi / si_sh_repositorio_01'. The repository is private. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Security, Insights, and Settings. The main content area displays the 'Code' tab, specifically the 'main' branch. Three recent commits are listed:

- Añadidos mensajes a los dos script
amontorosi committed 26 minutes ago (commit [9c6bdaa](#))
- Dos script
amontorosi committed 33 minutes ago (commit [cef1d5e](#))
- Incluyendo primer script
amontorosi committed 1 hour ago (commit [c95315c](#))

At the bottom of the commit list are 'Newer' and 'Older' buttons.

7. Uso básico de GitHub

Tarea 7

- **Parte 1.** Ya que estamos en este punto tienes que investigar sobre el fichero `.gitignore`:

<https://docs.github.com/es/get-started/getting-started-with-git/ignoring-files>

- ¿Para qué sirve este fichero?
- ¿Por qué es tan útil?
- ¿Qué son los ficheros 'plantilla' de `.gitignore` para un determinado framework o lenguaje de programación?
- ¿Qué ocurriría si tenemos el siguiente fichero `.gitignore`?

```
*.log
```

```
temp/
```

Nota: Git no hace seguimiento de las carpetas vacías.

7. Uso básico de GitHub

Tarea 7

- **Parte 1.** Investiga sobre el fichero `.gitignore`:

Advertencia: El archivo `.gitignore` sirve para evitar que Git rastree archivos no deseados, pero **NO borra archivos que ya han sido añadidos al repositorio.**

Si un archivo ya fue añadido (`git add`) y comiteado (`git commit`) antes de crear el `.gitignore`, ese archivo seguirá en el repositorio aunque lo agregues a `.gitignore` después. Aún así, si quieres ignorar en los futuros commit un archivo que ya está en Git tiene que hacer lo siguiente:

```
echo "archivo_que_quiero_ignorar.txt" >> .gitignore
git rm --cached archivo_que_quiero_ignorar.txt
git commit -m "Ignorando archivo_que_quiero_ignorar.txt"
git push origin main
```

7. Uso básico de GitHub

Tarea 7

- **Parte 1.** Investiga sobre el fichero **.gitignore**:

Advertencia:  Existe una forma de eliminar completamente un archivo del historial de Git, pero NO es recomendada en la mayoría de los casos, porque puede afectar la integridad del historial y causar problemas a otros colaboradores.

Es decir, es muy importante que **planifiques tu .gitignore desde el principio**.

7. Uso básico de GitHub

Tarea 7

- **Parte 2.** Y también tienes que investigar sobre la licencia: *Los repositorios públicos de GitHub se suelen utilizar para compartir software de código abierto. Para que tu repositorio sea verdaderamente de código abierto, tendrás que generarle una licencia. De este modo, las demás personas podrán usar, modificar y distribuir el software con libertad.*

<https://docs.github.com/es/enterprise-server@3.12/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/licensing-a-repository>

<https://choosealicense.com>

Choose an open source license

An open source license protects contributors and users. Businesses and savvy developers won't touch a project without this protection.

{ Which of the following best describes your situation? }



I need to work in a community.



I want it simple and permissive.



I care about sharing improvements.

Use the [license preferred by the community](#) you're contributing to or depending on. Your project will fit right in.

The [MIT License](#) is short and to the point. It lets people do almost anything they want with your project, like making and distributing closed source versions.

The [GNU GPLv3](#) also lets people do almost anything they want with your project, except distributing closed source versions.

7. Uso básico de GitHub

Tarea 7

■ Parte 2. Investiga sobre las licencias en GitHub:

- ¿Por qué es importante añadir una licencia a un repositorio? Si no agregas una licencia, tu código es privado por defecto (aunque sea público en GitHub).
- ¿Qué pasa si un proyecto no tiene una licencia?
- ¿Cómo se añade una licencia en GitHub?
- Elige una licencia para tu próximo proyecto en GitHub. ¿Cuál crees que se adapta mejor a lo que haces en el ciclo y por qué?

Licencias más comunes son MIT y GPLv3:

- MIT: Es una de las más populares porque permite usar, modificar y distribuir el código libremente, con la única condición de incluir la licencia original (solo deben mantener el aviso de la licencia y pueden incluirlo en software propietario). Suele usarse en desarrollo web o software académico.
- GPLv3: Cualquiera puede usar, modificar y distribuir, pero el código derivado debe ser también GPLv3. Garantiza que cualquier software derivado sea de código abierto. Nadie lo hará propietario.

Otras licencias interesantes:

- Apache 2.0: Similar a MIT, pero con protección contra patentes.
- Creative Commons (CC0): Para proyectos sin código, permite liberar el contenido al dominio público.

7. Uso básico de GitHub

Tarea 7

- **Parte 3. En tu equipo anfitrión (no en GitHub)**, crea un fichero readme.md donde pruebas las diferentes marcas de Markdown para explicar qué estamos haciendo en este repositorio.
- Cuando lo termines de editar, haz un commit (el mensaje del commit puede ser "Añadido readme.md"), publica ese fichero en tu repositorio remoto (push) y accede a la web de GitHub para ver cómo se visualiza.

7.1. Más usos de GitHub

- Hecha esta pausa, retomamos el tema y vamos a comprobar que nuestro repositorio local tiene asociado un repositorio remoto en GitHub. Este se llama por defecto **origin**:

```
git remote
```

```
git remote -v
```

```
profesoru ~ main ↑2 ↓1 ~/si_sh_repositorio_01 git remote
origin
profesoru ~ main ↑2 ↓1 ~/si_sh_repositorio_01 git remote -v
origin https://github.com/amontorosi/si_sh_repositorio_01.git (fetch)
origin https://github.com/amontorosi/si_sh_repositorio_01.git (push)
```

7.1. Más usos de GitHub

Nota: Si estás haciendo las prácticas en tu equipo anfitrión y ya has trabajado con Git y GitHub desde un IDE como VS Code en algún momento habrás almacenado tus credenciales y esto no pasará.

- Para obtener más detalles de esto:

```
git remote show origin
```

```
profesoru ~ main 12 ↓ ~ / si_sh_repositorio_01 git remote show origin
Username for 'https://github.com': amontoro_si
Password for 'https://amontoro_si@github.com':
* remoto origin
  URL para obtener: https://github.com/amontorosi/si_sh_repositorio_01.git
  URL para publicar: https://github.com/amontorosi/si_sh_repositorio_01.git
  Rama HEAD: main
  Rama remota:
    main rastreada
  Rama local configurada para 'git pull':
    main fusiona con remoto main
  Referencia local configurada para 'git push':
    main publica a main (desactualizado local)
```

¿Te pide de nuevo las credenciales?

<https://docs.github.com/es/get-started/getting-started-with-git/caching-your-github-credentials-in-git>

7.1. Más usos de GitHub

¿Te pide de nuevo las credenciales?

<https://docs.github.com/es/get-started/getting-started-with-git/caching-your-github-credentials-in-git>

- Si GitHub te pide el token cada vez que haces algo contra el repositorio remoto, significa que Git no está almacenando las credenciales. Esto puede solucionarse de varias maneras:
 - Usar gt aut login con GitHub CLI. Es una opción muy recomendada.
 - Usar SSH en lugar de HTTPS. Es el método más seguro y permanente:
<https://github.com/settings/keys>
 - Usar el "Credential Store" para guardar el token permanentemente. Esto guardará el token en texto plano en `~/.git-credentials` (no es la opción más segura).

Nota importante: En Windows, Git también usa **Git Credential Manager (GCM)** para almacenar credenciales de GitHub. La primera vez que hagas git push, GitHub te pedirá autenticación en el navegador.

7.1. Más usos de GitHub

- Otra forma (no es la opción más segura):

```
git remote git remote set-url origin
```

```
https://<TOKEN>@github.com/tuusuario/tu-repo.git
```

7.1. Más usos de GitHub

- Ahora en **Ubuntu**, vamos a almacenar nuestras credenciales de GitHub en la caché dentro de Git. Para ello voy a usar el [CLI de GitHub](#).

Para ello tenemos que instalarlo (son 4 pasos en Ubuntu):

```
curl -fsSL https://cli.github.com/packages/githubcli-archive-keyring.gpg |  
sudo dd of=/usr/share/keyrings/githubcli-archive-keyring.gpg
```

```
echo "deb [arch=$(dpkg --print-architecture) signed-  
by=/usr/share/keyrings/githubcli-archive-keyring.gpg]  
https://cli.github.com/packages stable main" | sudo tee  
/etc/apt/sources.list.d/github-cli.list > /dev/null
```

```
sudo apt update
```

```
sudo apt install gh
```

Windows

`gh` is available via [WinGet](#), [scoop](#), [Chocolatey](#), [Conda](#), [Webi](#), and as downloadable MSI.

7.1. Más usos de GitHub

- Una vez instalado ejecuta el siguiente mensaje y sigue los pasos:

```
gh auth login
```

```
profesoru ~ main ~ /si_sh_repositorio_01 127 gh auth login
? What account do you want to log into? GitHub.com
? What is your preferred protocol for Git operations? HTTPS
? Authenticate Git with your GitHub credentials? Yes
? How would you like to authenticate GitHub CLI? Paste an authentication token
Tip: you can generate a Personal Access Token here https://github.com/settings/tokens
The minimum required scopes are 'repo', 'read:org', 'workflow'.
? Paste your authentication token: *****
- gh config set -h github.com git_protocol https
✓ Configured git protocol
✓ Logged in as amontorosi
```

7.1. Más usos de GitHub

Tarea 8

- En tu equipo de trabajo tienes que **almacenar tus credenciales de GitHub en la caché dentro de Git.**
- Elige el método que más te guste, investiga sobre él y úsalo:
 - Usar el Git Credential Manager (GCM)
 - Usar GitHub CLI
 - Usar SSH en lugar de HTTPS

7.1. Más usos de GitHub

- **Antes de seguir:**

Verifica que estás en la rama main

```
git pull origin main
```

```
git push origin main
```

Buenas prácticas, sobre todo cuando trabajas en equipo

Es fundamental siempre hacer `git pull` antes de `git push` para evitar conflictos y asegurarte de que tienes la versión más actualizada del código antes de subir tus cambios.

Nota: Con `git push -u origin main`, además de hacer un `git push origin main`, se estableció `origin/main` como el repositorio remoto predeterminado para futuras subidas. ¿Esto qué permite?

7.1. Más usos de GitHub

- Crea una nueva carpeta que se llame pruebas y dentro un fichero vacío con nombre pruebas.txt. Después confirma los cambios y publica esos cambios en tu repositorio de GitHub:

```
mkdir pruebas
```

```
touch pruebas/pruebas.txt
```

```
git add .
```

```
git commit -m "Carpeta para pruebas"
```

```
git push (equivale a git push origin main)
```

```
profesoru ~/main 7:1 ~ /si_sh_repositorio_01 ➔ git add .
profesoru ~/main S:1 ~ /si_sh_repositorio_01 ➔ git commit -m "Carpeta para pruebas"
[main 2aae7fd] Carpeta para pruebas
 1 file changed, 0 insertions(+), 0 deletions(-)
   create mode 100644 pruebas/pruebas.txt
profesoru ~/main ↑ ~ /si_sh_repositorio_01 ➔ git push
Username for 'https://github.com': amontoro_si
Password for 'https://amontoro_si@github.com':
Enumerando objetos: 5, listo.
Contando objetos: 100% (5/5), listo.
Compresión delta usando hasta 2 hilos
Comprimiendo objetos: 100% (2/2), listo.
Escribiendo objetos: 100% (4/4), 334 bytes | 334.00 KiB/s, listo.
Total 4 (delta 1), reusados 0 (delta 0), pack-reusados 0
```

7.1. Más usos de GitHub

- Comprueba los cambios en GitHub y desde allí añade tu nombre a ese fichero y confirma los cambios:

The screenshot shows a GitHub repository interface. At the top left, there is a dropdown menu set to 'main'. Next to it is the repository name 'si_sh_repositorio_01 / pruebas /'. Below this, there is a list of files and folders. Under the 'amontorosi Carpeta para pruebas' folder, there is a file named 'pruebas.txt'. To the right of the file name, the description 'Carpeta para pruebas' is visible.

This screenshot shows the content of the 'pruebas.txt' file. The file path 'si_sh_repositorio_01 / pruebas / pruebas.txt' is at the top, followed by the text 'in main'. Below this, there are two buttons: 'Edit file' and 'Preview changes'. The file content itself shows a single line of text: '1 Alberto M.|'.

This screenshot shows the 'Commit changes' dialog box. It has a title 'Commit changes' and a sub-instruction 'Update pruebas.txt'. Below this is a text input field containing the placeholder 'Add an optional extended description...'. At the bottom of the dialog, there are two radio button options: one selected with the label 'Commit directly to the main branch.' and another unselected with the label 'Create a new branch for this commit and start a pull request. Learn more about pull requests.'. At the very bottom of the dialog are two buttons: 'Commit changes' in green and 'Cancel' in red.

7.1. Más usos de GitHub

- Muestra el histórico de todos los commit realizados:

```
git log
```

¿Aparece el último commit realizado en GitHub?

```
commit 3e3244066abeabbda720c2496ab913ebf44dd6a8 (HEAD -> main, origin/main)
Author: A. Montoro <amontoro.sistemasinformaticos@gmail.com>
Date:   Fri May 20 23:46:12 2022 +0200
```

Carpeta para pruebas

```
commit 6b06alfelf10c1b2e9ba16c6dff21600d210b88f
Author: A. Montoro <amontoro.sistemasinformaticos@gmail.com>
Date:   Fri May 20 23:16:30 2022 +0200
```

Añadidos mensajes a los dos scripts

```
commit 4e0b71e4e7cd8ddb40cdelb00f33d4e5822b3fe
Author: A. Montoro <amontoro.sistemasinformaticos@gmail.com>
Date:   Fri May 20 23:13:05 2022 +0200
```

Dos script

```
commit 7a892890bb8353804b11c433cd122a016454788f
Author: A. Montoro <amontoro.sistemasinformaticos@gmail.com>
Date:   Fri May 20 22:38:53 2022 +0200
```

Incluyendo primer script

7.1. Más usos de GitHub

- Tráete esos cambios e inclúyelos directamente en tu repositorio:

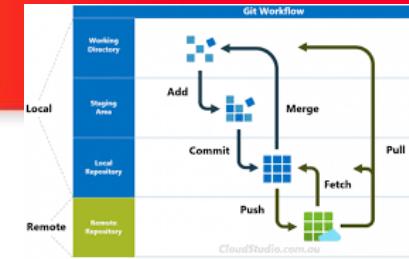
```
git pull
```

Equivale a: `git pull origin main`

```
cat pruebas/pruebas.txt
```

```
profesoru ~ main ~/si_sh_repositorio_01 git pull
Username for 'https://github.com': amontoro_si
Password for 'https://amontoro_si@github.com':
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 1), reused 0 (delta 0), pack-reused 0
Desempaquetando objetos: 100% (4/4), 694 bytes | 694.00 KiB/s, listo.
Desde https://github.com/amontorosi/si_sh_repositorio_01
 2aae7fd..67f2395 main      -> origin/main
 * [nueva rama]    funciones -> origin/funciones
Actualizando 2aae7fd..67f2395
Fast-forward
  pruebas/pruebas.txt | 1 +
  1 file changed, 1 insertion(+)
```

```
profesoru ~ main ~/si_sh_repositorio_01 cat pruebas/pruebas.txt
Alberto M.
```



7.1. Más usos de GitHub

¿Qué estamos haciendo exactamente con pull?

Cuando ejecutamos git pull, estamos trayendo los cambios del repositorio remoto y **fusionándolos directamente con nuestra rama local**. Es como descargar una actualización y aplicarla al instante.

¿Y si solo quiero bajar los cambios pero decidir después si los consolido o no?

En este caso, en lugar de git pull, usamos git fetch. Esto solo descarga los cambios del remoto, pero no los fusiona automáticamente.

```
git fetch origin
```

Opción a) Si después de revisar los cambios quieres aplicarlos, usamos:

```
git merge origin/main
```

Opción b) Si decides que no quieres esos cambios aún, no haces nada.

7.1. Más usos de GitHub

- Edita el fichero pruebas.txt añadiendo tu ID a la última línea. Después confirma los cambios y publica los cambios en tu repositorio de GitHub:

```
code pruebas/pruebas.txt
```

```
git add .
```

```
git commit -m "Cambios en pruebas.txt en local"
```

```
git push
```

```
profesoru ~ main ~/si_sh_repositorio_01 gedit pruebas/pruebas.txt
profesoru ~ main ~/si_sh_repositorio_01 git add .
profesoru ~ main S:1 ~/si_sh_repositorio_01 git commit -m "Cambios en prue
pas.txt"
[main 90a061a] Cambios en pruebas.txt
 1 file changed, 1 insertion(+)
profesoru ~ main ↑ ~/si_sh_repositorio_01 git push
Enumerando objetos: 7, listo.
Contando objetos: 100% (7/7), listo.
Compresión delta usando hasta 2 hilos
Comprimiendo objetos: 100% (2/2), listo.
Escribiendo objetos: 100% (4/4), 345 bytes | 345.00 KiB/s, listo.
Total 4 (delta 1), reusados 0 (delta 0), pack-reusados 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/amontorosi/si_sh_repositorio_01.git
  67f2395..90a061a  main -> main
profesoru ~ main ~/si_sh_repositorio_01
```

7.1. Más usos de GitHub

Tarea 9

- En el módulo Entornos de Desarrollo, prueba en Visual Studio Code las siguientes extensiones:
 - gitignore
 - Git History
 - Git Graph
 - Open in GitHub, Bitbucket, Gitlab, VisualStudio.com
- Crea una carpeta con nombre si_sh_repositorio_02 y repite todos los pasos realizados hasta ahora, pero ahora usando la interfaz gráfica de VS Code.
- Mientras realizas este ejercicio en Visual Studio Code, te recomiendo que abras GitHub Desktop en paralelo para observar cómo se reflejan los cambios en una interfaz visual.
 - Observa cómo se actualiza la vista cuando realizas acciones en VS Code (como git add, commit, push, etc.).
 - Explora la pestaña "Changes" para ver qué archivos han sido modificados.
 - Mira el historial de commits en la pestaña "History" para comparar con git log en la terminal.

7.2. Clonar un repositorio

- Imagina que vas a otro equipo y quieres seguir trabajando con este repositorio o suponte que ves un repositorio en GitHub y te lo quieres descargar para trabajar con el en local o piensa que acabas de incorporarte a un equipo de trabajo que ya tiene publicado un repositorio en GitHub.
- Esto es muy sencillo gracias a `git clone url_repositorio`.

7.2. Clonar un repositorio

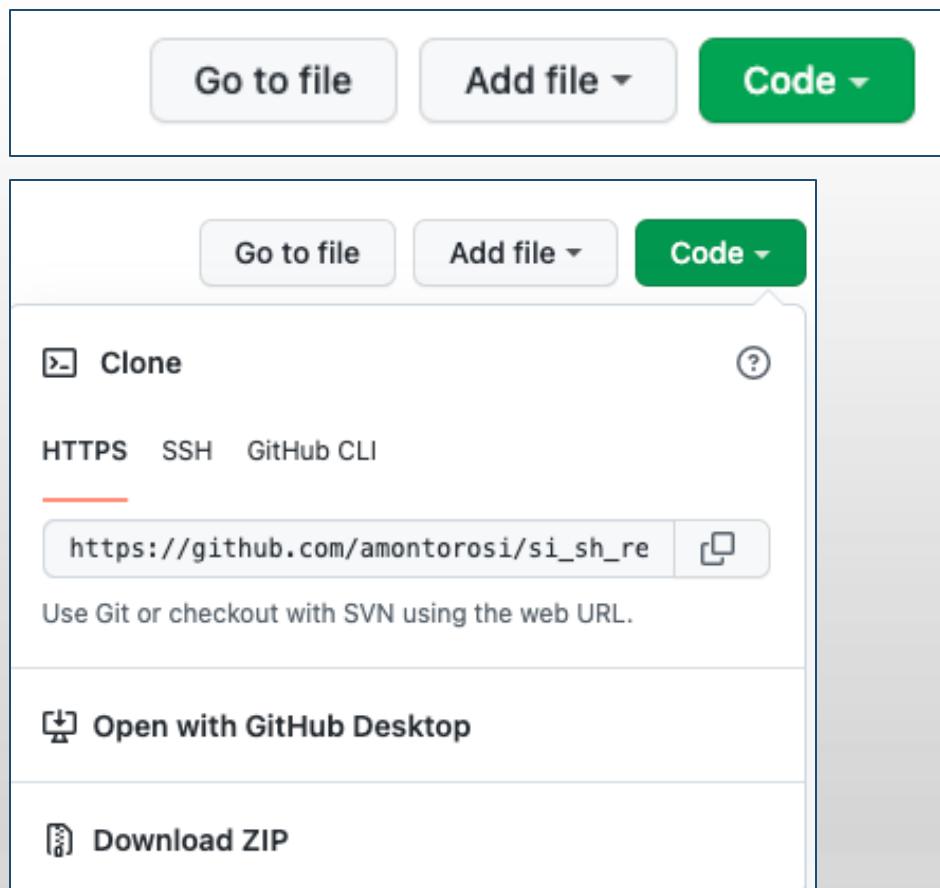
- Antes de seguir:

git pull

git push

7.2. Clonar un repositorio

- Accede al repositorio de GitHub y comprueba las opciones disponibles en el botón 'Code' del repositorio que hemos creado. Finalmente copia la URL https:



7.2. Clonar un repositorio

- En el terminal, sal de tu repositorio y crea una carpeta con nombre `si_en_la_oficina` y entra en ella:

```
cd ..
```

```
mkdir si_en_la_oficina
```

```
cd si_en_la_oficina
```

- Clona el repositorio de GitHub en tu equipo:

```
git clone https://github.com/amontorosi/si_sh_repositorio_01.git
```

- Haz un listado del contenido del directorio:

```
ls
```

- Entra dentro de la carpeta que te has clonado:

```
cd si_sh_repositorio_01
```

7.2. Clonar un repositorio

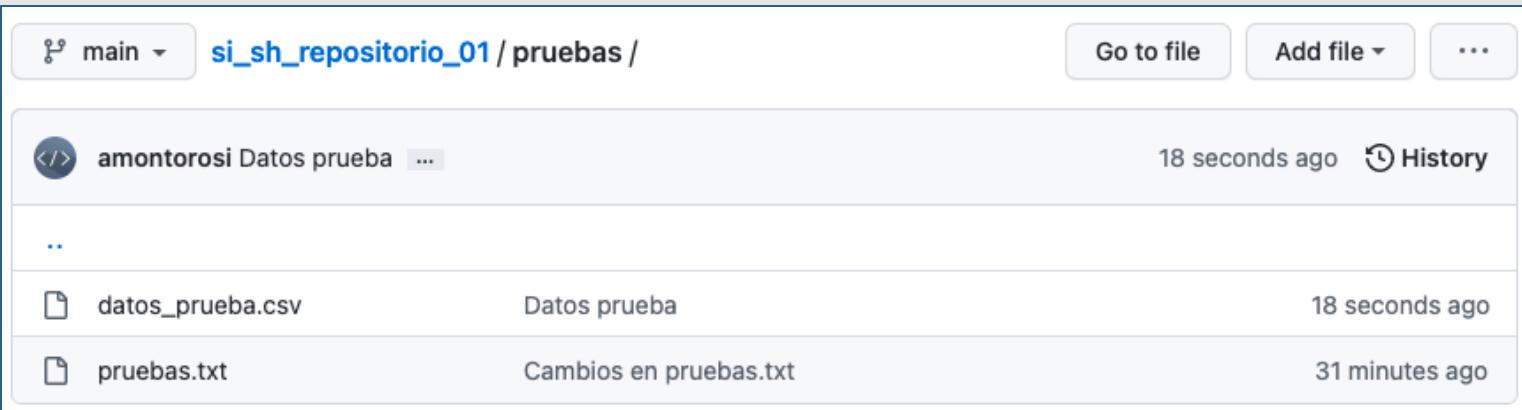
- **MUY IMPORTANTE:** Ahora tenemos dos repositorios locales (en este caso en la misma máquina porque estamos probando, pero lo lógico es que estén en dos equipos distintos). Y ambos están apuntando **al mismo repositorio remoto en GitHub**. Es por ello que tenemos que tener todo siempre actualizado y trabajar con cuidado antes de editar nada porque en caso contrario podrías generar conflictos más difíciles de gestionar.

7.2. Clonar un repositorio

- Crea dentro de la carpeta pruebas un fichero vacío con nombre datos_prueba.csv. Después confirma los cambios y publica estos cambios en tu repositorio de GitHub:

```
touch pruebas/datos_prueba.csv  
git add .  
git commit -m "Datos prueba"  
git push
```

- Comprueba los cambios en GitHub:



The screenshot shows a GitHub repository interface. At the top, there's a navigation bar with 'main' and a dropdown, the repository name 'si_sh_repositorio_01 / pruebas /', and buttons for 'Go to file', 'Add file', and more. Below this is a list of recent commits:

Commit	Message	Time
 amontorosi	Datos prueba	18 seconds ago
..		
 datos_prueba.csv	Datos prueba	18 seconds ago
 pruebas.txt	Cambios en pruebas.txt	31 minutes ago

7.2. Clonar un repositorio

- Ahora vuelve al repositorio original y tráete estos cambios:

```
cd ../../
```

```
cd si_sh_repositorio_01
```

```
ls pruebas/
```

```
profesoru ➜ main ➤ ~/si_sh_repositorio_01 ➤ ls pruebas/  
pruebas.txt
```

```
git pull
```

```
ls pruebas/
```

```
profesoru ➜ main ➤ ~/si_sh_repositorio_01 ➤ git pull  
remote: Enumerating objects: 6, done.  
remote: Counting objects: 100% (6/6), done.  
remote: Compressing objects: 100% (2/2), done.  
remote: Total 4 (delta 1), reused 4 (delta 1), pack-reused 0  
Desempaquetando objetos: 100% (4/4), 331 bytes | 331.00 KiB/s, listo.  
Desde https://github.com/amontorosi/si_sh_repositorio_01  
 00ed54c..6d6e871 main      -> origin/main  
Actualizando 00ed54c..6d6e871  
Fast-forward  
  pruebas/datos_prueba.csv | 0  
  1 file changed, 0 insertions(+), 0 deletions(-)  
  create mode 100644 pruebas/datos_prueba.csv
```

```
profesoru ➜ main ➤ ~/si_sh_repositorio_01 ➤ ls pruebas/  
datos_prueba.csv  pruebas.txt
```

7.2. Clonar un repositorio

■ Problemas típicos con pull:

- **Problema 1.** Hay cambios en el main en GitHub y nuestra versión local de main está "atrasada" respecto al remoto, pero no hay conflictos. En este caso Git no permite sobrescribir cambios remotos sin antes traerlos.
 - **Solución recomendada:** Antes de hacer git push, primero hay que hacer git pull para traer los cambios del remoto.

- **Problema 2.** Tu main local y el main en GitHub tienen cambios diferentes. Git no sabe si debe hacer un merge o un rebase para combinar los cambios. Esto lo veremos en el siguiente apartado.
 - **Solución recomendada:** Git sugiere 3 opciones que podemos configurar con git config.

1. Hacer un merge automáticamente (más sencillo para principiantes). Esto creará un commit de merge que combina ambas versiones de main.

```
git config pull.rebase false # Usa merge en los pull
```

2. Hacer un rebase automáticamente (historial más limpio). Esto moverá los commits locales al final de los commits remotos, evitando un commit de merge.

```
git config pull.rebase true # Usa rebase en los pull
```

3. Solo permitir fast-forward si no hay conflictos

```
git config pull.ff only
```

Índice de contenidos

1. Introducción
2. Conceptos
3. Flujo de trabajo
4. Instalación de Git
5. Primeros pasos
6. Uso básico de Git
7. Uso básico de GitHub
- 8. Ramas en Git y GitHub**
9. Trabajo colaborativo
10. Algunos comandos útiles

8. Ramas en Git y GitHub

- La creación de ramas nos permite tener líneas independientes de desarrollo en un mismo proyecto:

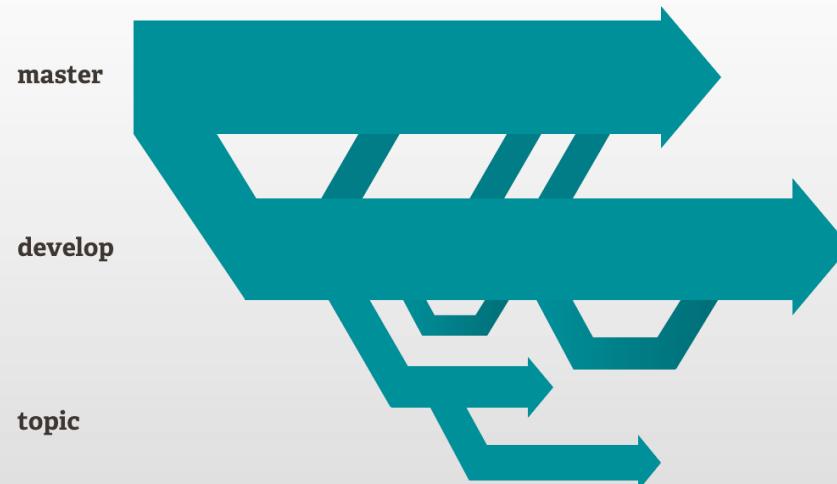


Imagen extraída de: <https://git-scm.com/about/>

8. Ramas en Git y GitHub

- **Antes de seguir:**

Verifica que estás en la rama main

```
git pull
```

```
git push
```

8. Ramas en Git y GitHub

- Comprueba los commits que hemos hecho y la rama en la que te encuentras:

```
git log --oneline
```

```
aa12a3d (HEAD -> main, origin/main) Datos prueba
5866a8c Cambio en pruebas.txt en local
8562fde Update prubas.txt
3805bb9 Carpeta para pruebas
fb4384c Eliminado ficheros_conf.txt
b7067cd Añadido ficheros_conf.txt
1d4b7cc Eliminado comandos.txt
7d99036 Añadido comandos.txt
e3c26be Añadidos mensajes a los dos scripts
49b06d4 Dos scripts
c5a36c5 Incluyendo primer script
(END)
```

¿Qué era HEAD?

```
git log --all
```

Muestra todos los commits de todas las ramas, incluso aquellos que no están en la rama actual.

8. Ramas en Git y GitHub

- Crea una rama que se llama funciones:

```
git branch funciones:
```

A screenshot of a terminal window with a blue header bar. The terminal shows the command 'git branch funciones' being run. The output shows the current branch is 'main', and a new branch 'funciones' is created.

```
profesoru ~profesoru main ~profesoru main ~si_sh_repositorio_01 git branch funciones
```

- Ejecuta de nuevo el comando anterior:

```
git log --oneline
```

```
aa12a3d (HEAD -> main, origin/main, funciones) Datos prueba
5866a8c Cambio en pruebas.txt en local
8562fde Update pruebas.txt
3805bb9 Carpeta para pruebas
fb4384c Eliminado ficheros_conf.txt
b7067cd Añadido ficheros_conf.txt
1d4b7cc Eliminado comandos.txt
7d99036 Añadido comandos.txt
e3c26be Añadidos mensajes a los dos scripts
49b06d4 Dos scripts
c5a36c5 Incluyendo primer script
(END)
```

8. Ramas en Git y GitHub

- Consulta las ramas disponibles **en local** (el asterisco te indica la rama en la que estás):

```
git branch
```

```
funciones
* main
(END)
```

¿Qué ocurre si ahora empiezo a hacer cambios y commits?

- Consulta todas las ramas disponibles, tanto **en local** como **en remoto**.

```
git branch -a
```

```
funciones
* main
remotes/origin/main
(END)
```

```
git branch -r
```

8. Ramas en Git y GitHub

- Cambia a la rama funciones:

```
git checkout funciones
```

También puedes usar: `git switch funciones`

git checkout tiene además otros usos y git switch solo cambia de rama.

```
git branch
```

```
profesoru ~ main ↑↓ ~/si_sh_repositorio_01 git checkout funciones
Cambiado a rama 'funciones'
profesoru ~ funciones ~/si_sh_repositorio_01 git branch
```

Nota 1: Si queremos crear una rama nueva y saltar directamente a ella:

```
git checkout -b rama_nueva
```

Nota 2: Si queremos borrar una rama:

```
git branch -d nombre-de-la-rama (si fue fusionada)
```

```
git branch -D nombre-de-la-rama (si no ha sido fusionada)
```

```
git push origin --delete nombre-de-la-rama (rama remota)
```

```
git fetch --prune (limpiar ramas remotas eliminadas)
```

8. Ramas en Git y GitHub

- Crea una carpeta que se llame funciones y un fichero que se llame funciones.sh. Dale permisos de ejecución y luego editalo con el siguiente contenido:

```
mkdir funciones  
touch funciones/funciones.sh  
chmod u+x funciones/funciones.sh
```

Recuerda: En Git Bash de Windows los permisos son emulados.

Por tanto, chmod no hace nada en Git Bash.

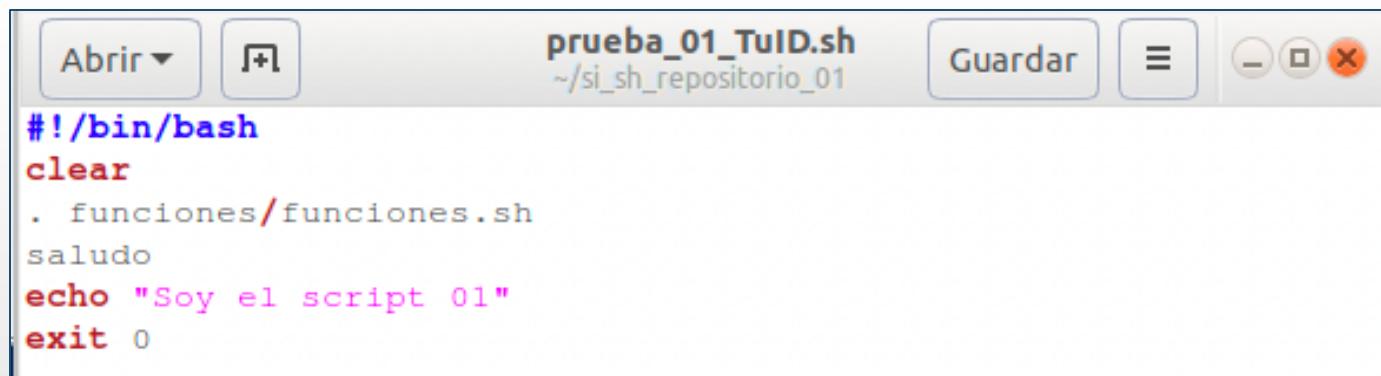
```
code funciones/funciones.sh &
```

```
#!/bin/bash
function saludo() {
    echo "Hola desde funciones"
}
```

Recuerda: En Git Bash de Windows al comenzar el fichero con #!/bin/bash, asigna automáticamente el permiso de ejecución.

8. Ramas en Git y GitHub

- Edita el fichero prueba_01_TuID.sh con el siguiente contenido y ejecútalo para ver que funciona:



A screenshot of a terminal window titled "prueba_01_TuID.sh" located at "~/si_sh_repositorio_01". The window has standard OS X-style controls (Minimize, Maximize, Close) at the top right. The script content is as follows:

```
#!/bin/bash
clear
. funciones/funciones.sh
saludo
echo "Soy el script 01"
exit 0
```

8. Ramas en Git y GitHub

- Agrega la modificaciones al staging area y confirma los cambios:

```
git add .
```

```
git commit -m "Creado fichero con funciones"
```

```
profesoru 2: funciones 0:1 ?:1 ~ /si_sh_repositorio_01 git add .
profesoru 3: funciones S:3 ~ /si_sh_repositorio_01 git commit -m "Creado fichero con funciones"
```

- Muestra el histórico de commit y analiza el resultado:

```
git log --oneline
```

```
972e49f (HEAD -> funciones) Creado fichero funciones
aa12a3d (origin/main, main) Datos prueba
5866a8c Cambio en pruebas.txt en local
8562fde Update pruebas.txt
3805bb9 Carpeta para pruebas
fb4384c Eliminado ficheros_conf.txt
b7067cd Añadido ficheros_conf.txt
1d4b7cc Eliminado comandos.txt
7d99036 Añadido comandos.txt
e3c26be Añadidos mensajes a los dos scripts
49b06d4 Dos scripts
c5a36c5 Incluyendo primer script
(END)
```

```
git log --oneline --all
```

8. Ramas en Git y GitHub

- Ejecuta el siguiente comando:

```
git push
```

Recuerda que equivale a: git push origin main

```
profesoru ~> funciones ~> ~/si_sh_repositorio_01 > git push  
fatal: La rama actual funciones no tiene una rama upstream.  
Para realizar un push de la rama actual y configurar el remoto como upstream, usa
```

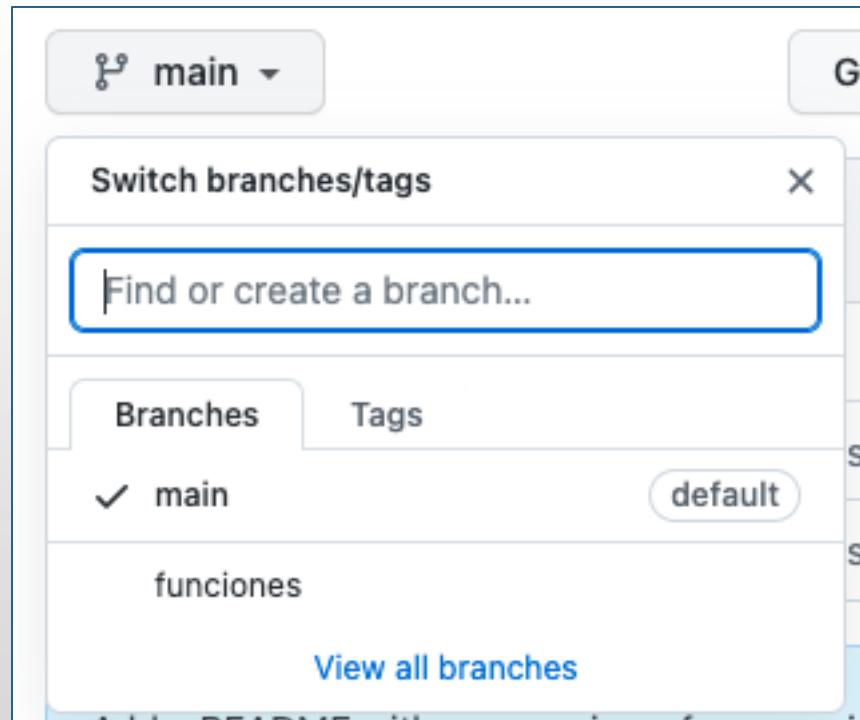
```
git push --set-upstream origin funciones
```

```
profesoru ~> funciones ~> ~/si_sh_repositorio_01 > 128 >
```

¿Qué ocurre?

```
git push origin funciones
```

8. Ramas en Git y GitHub



8. Ramas en Git y GitHub

- Consulta todas las ramas disponibles, tanto **en local** como **en remoto**.

```
git branch -a
```

```
* funciones
  main
  remotes/origin/functions
  remotes/origin/main
(END)
```

8. Ramas en Git y GitHub

- Ejecuta el siguiente comando:

```
git push --set-upstream origin funciones
```

Equivale a: git push -u origin funciones

Estableciste un vínculo entre la rama local `funciones` y su versión remota en GitHub. Ahora, cada vez que hagas `git push` o `git pull`, Git sabrá automáticamente a qué remoto enviar o traer cambios.

8. Ramas en Git y GitHub

Interesante: Git permite moverte a un commit anterior, crear una rama desde ahí y seguir trabajando como si fuera una línea alternativa de desarrollo.

Verifica el historial para encontrar el hash del commit:

```
git log --oneline --all --graph
```

Crea la rama

```
git checkout -b nueva-rama hash_commit
```

Ahora puedes hacer cambios y commits en esta nueva rama sin afectar main.

Commit1 -> Commit2 -> Commit3 -> Commit4 (main)

```
\  
  Commit2.1 -> Commit2.2 -> Commit2.3 (nueva-rama)
```

8. Ramas en Git y GitHub

Supuesto 1.

- Cuando terminamos nuestro trabajo en la rama funciones y está todo testeado y se han realizado los pull y push pertinentes, se puede fusionar esa rama con al principal.
- Si no existen conflictos se hará de forma inmediata. Dado que ahora mismo estamos trabajando solo, no debería haber conflictos.
- Si existen conflictos, git intentará repararlos y si no puede, te pedirá intervención en aquellos ficheros en los que existe conflicto. Sin embargo, sí integrará los ficheros en los que no haya conflicto.

8. Ramas en Git y GitHub

Supuesto 1.

- Recuerda que en este caso Git solo genera conflictos cuando editamos las mismas líneas de un archivo (en local y en remoto a la vez, o en ramas diferentes y las queremos fusionar). Si cada trabajamos en secciones diferentes de un mismo archivo, Git puede **combinar los cambios automáticamente sin problema**.

8. Ramas en Git y GitHub

Supuesto 1.

- Ahora mismo no existe ningún conflicto. Así que podemos realizar la integración sin problemas.

- Muévete a la rama principal:

```
git checkout main
```

Asegúrate de que la rama main está actualizado con los últimos cambios del remoto: git pull origin main

- Haz un ls y un cat al fichero modificado en la otra rama para comprobar que se mantiene intacto en la rama main:

```
ls  
cat prueba_01_TuID.sh
```

```
profesoru ~ main ~/si_sh_repositorio_01 cat prueba_01_TuID.sh  
#!/bin/bash  
clear  
echo "Soy el script 01"  
exit 0  
profesoru ~ main ~/si_sh_repositorio_01
```

8. Ramas en Git y GitHub

Supuesto 1.

- Prueba el siguiente comando para conocer las ramas que se han fusionado con la rama en la que nos encontramos. De momento ninguna, así que solo debe aparecer main:

```
git branch --merged
```

```
* main  
(END)
```

8. Ramas en Git y GitHub

Supuesto 1.

- Comprueba las diferencias entre las dos ramas antes de fusionarlas.

```
git diff main funciones
```

Otros casos útiles

Comparar una rama remota con una local

```
git diff origin/main main
```

Ver cambios específicos en un archivo entre ramas

```
git diff main funciones -- archivo.txt
```

8. Ramas en Git y GitHub

Supuesto 1.

- Ahora mismo no existe ningún conflicto. Así que podemos realizar la integración sin problemas.
- Fusiona los cambios de la rama funciones en la rama en la que te encuentras:

```
git merge funciones  
ls  
cat prueba_01_TuID.sh
```

```
profesoru ~ main ~ /si_sh_repositorio_01 1 git merge funciones  
Actualizando 6d6e871..5ddc1ea  
Fast-forward  
  funciones/funciones.sh | 4 +  
  prueba_01_TuID.sh      | 2 +  
  2 files changed, 6 insertions(+)  
  create mode 100755 funciones/funciones.sh  
profesoru ~ main ↑ ~ /si_sh_repositorio_01 1 ls  
funciones prueba_01_TuID.sh prueba_02_TuID.sh pruebas  
profesoru ~ main ↑ ~ /si_sh_repositorio_01 1
```

```
git status
```

8. Ramas en Git y GitHub

Supuesto 1.

- Prueba el siguiente comando para conocer las ramas que se han fusionado con la rama en la que nos encontramos.

```
git branch --merged
```

8. Ramas en Git y GitHub

Supuesto 1.

- Esto que acabamos de hacer se conoce como:

Fast-Forward Merge (Fusión rápida)

- Ocurre cuando Git puede avanzar directamente sin crear un nuevo commit de merge.
 - Sigue sucediendo cuando la rama a fusionar no tiene commits nuevos en común con main.
 - No hay conflictos porque la rama es simplemente una continuación.

8. Ramas en Git y GitHub

Supuesto 1.

- Prueba el siguiente comando para conocer las ramas que se han fusionado en la rama en la que nos encontramos:

```
git branch --merged
```

```
funciones
* main
(END)
```

- Publica los cambios en GitHub (por si hubiera alguno):

```
git push
```

```
profesoru ➜ main ↑i ➤ ~/si_sh_repositorio_01 ➤ git push
Total 0 (delta 0), reusados 0 (delta 0), pack-reusados 0
To https://github.com/amontorosi/si_sh_repositorio_01.git
  6d6e871..5ddclea  main -> main
profesoru ➜ main ➤ ~/si_sh_repositorio_01 ➤
```

8. Ramas en Git y GitHub

Supuesto 1.

- Muestra el histórico de los commit realizado en esta rama:

```
git log --oneline
```

```
972e49f (HEAD -> main, origin/main, origin/funciones, funciones) Creado fichero
funciones
aa12a3d Datos prueba
5866a8c Cambio en pruebas.txt en local
8562fde Update prubas.txt
3805bb9 Carpeta para pruebas
fb4384c Eliminado ficheros_conf.txt
b7067cd Añadido ficheros_conf.txt
1d4b7cc Eliminado comandos.txt
7d99036 Añadido comandos.txt
e3c26be Añadidos mensajes a los dos scripts
49b06d4 Dos scripts
c5a36c5 Incluyendo primer script
(END)
```

```
git log --graph --oneline --all --decorate
```

8. Ramas en Git y GitHub

Supuesto 1.

- Muestra el histórico de los commit realizado en esta rama mostrando los cambios realizado:

```
git log --stat
```

```
commit 5ddc1ea0f1644ad1ff6abca5e713570748da6a28 (HEAD -> funciones, origin/main,
origin/functions, main)
Author: A. Montoro <amontoro.sistemasinformaticos@gmail.com>
Date:   Sat May 21 00:51:55 2022 +0200

    Creado fichero con funciones

    funciones/functions.sh | 4 +++
    prueba_01_TuID.sh     | 2 ++
    2 files changed, 6 insertions(+)

commit 6d6e871054302f5199cf2e1c1b6b02605a9cbe60
Author: A. Montoro <amontoro.sistemasinformaticos@gmail.com>
Date:   Sat May 21 00:29:20 2022 +0200

    Datos prueba

    pruebas/datos_prueba.csv | 0
    1 file changed, 0 insertions(+), 0 deletions(-)
```

```
git log --oneline --stat
```

8. Ramas en Git y GitHub

Supuesto 1.

- Si la rama funciones no fuera ya necesaria podríamos eliminarla ahora que está fusionada en main. Aunque NO lo vamos a hacer:

```
git branch -d funciones (en local)
```

```
git push origin -d funciones (en remoto)
```

8. Ramas en Git y GitHub

Supuesto 2.

- Imagina que estamos trabajando en funciones y acabamos de hacer un commit con los último cambios y nuestro push pertinente, **y aún nos queda trabajo en esta rama**. Sin embargo, nos damos cuenta de que hay nuevas actualizaciones en main que queremos traer a la rama funciones. Sabemos que no habrá conflictos. ¿Cómo hacemos el merge correctamente?
- **Siempre verifica en qué rama estás antes de hacer cualquier acción.**

```
git checkout funciones
```

```
git branch
```

```
git merge main
```

```
git status
```

Entendemos que la rama main está actualizado con los últimos cambios del remoto

8. Ramas en Git y GitHub

Supuesto 2.

- En estos casos, **se puede** también reescribir el historial moviendo los commits de una rama para que aparezcan después de otra. De este modo **se mantiene un historial más limpio**.
- Para ello, hacemos un pequeño cambio en los pasos anteriores:

```
git checkout funciones
```

```
git branch
```

```
git rebase main
```

- Git tomará todos los commits de nueva-funcionalidad, aplicará los nuevos cambios de main, y luego los "pegará" al final.

8. Ramas en Git y GitHub

Supuesto 3.

- Ahora imagina que hemos creado una rama nueva en la que hemos creado un fichero. A continuación, hemos hecho un commit y nuestro push pertinente, pero aún nos queda trabajo en esa rama antes de fusionarla con la rama main.

Mientras seguimos trabajando en nuestra nueva rama, la rama main ha cambiado, pero lo que ha cambiado no genera **ningún conflicto** con lo que nosotros estamos haciendo y por tanto nosotros seguimos trabajando tranquilamente.

Cuando terminamos nuestro trabajo en la nueva rama, hacemos un commit y ya sí queremos llevar esos cambios a la rama main.

¿Cómo hacemos el merge correctamente? ¿Qué ocurre?

8. Ramas en Git y GitHub

Supuesto 3.

```
git branch arrays
```

```
git checkout arrays
```

```
mkdir arrays
```

```
touch arrays/arrays.sh
```

```
chmod u+x arrays/arrays.sh
```

Entendemos que la rama main está actualizado con los últimos cambios del remoto

Recuerda: En Git Bash de Windows los permisos son emulados.

Por tanto, chmod no hace nada en Git Bash

```
code arrays/arrays.sh &
```

```
$ arrays.sh x
home > profesor > si_sh_repositorio_01 > $ arrays.sh
1 #!/bin/bash
2 archivos=("a1.txt" "a2.txt" "a3.txt")
3 exit 0
```

8. Ramas en Git y GitHub

Supuesto 3.

```
git add .  
git commit -m "Creado fichero con arrays"  
git push origin arrays  
git log --oneline --all
```

8. Ramas en Git y GitHub

Supuesto 3.

```
git ckeckout main  
touch prueba_03_TuID.sh  
git add .  
git commit -m "Tres scripts"  
git push origin main
```

Entendemos que la rama main está actualizado con los últimos cambios del remoto

8. Ramas en Git y GitHub

Supuesto 3.

```
git checkout arrays  
code arrays/arrays.sh &
```

```
$ arrays.sh x  
home > profesoru > si_sh_repositorio_01 > arrays > $ arrays.sh  
1 #!/bin/bash  
2 archivos=("a1.txt" "a2.txt" "a3.txt")  
3 echo ${archivos[1]}  
4 echo 0
```

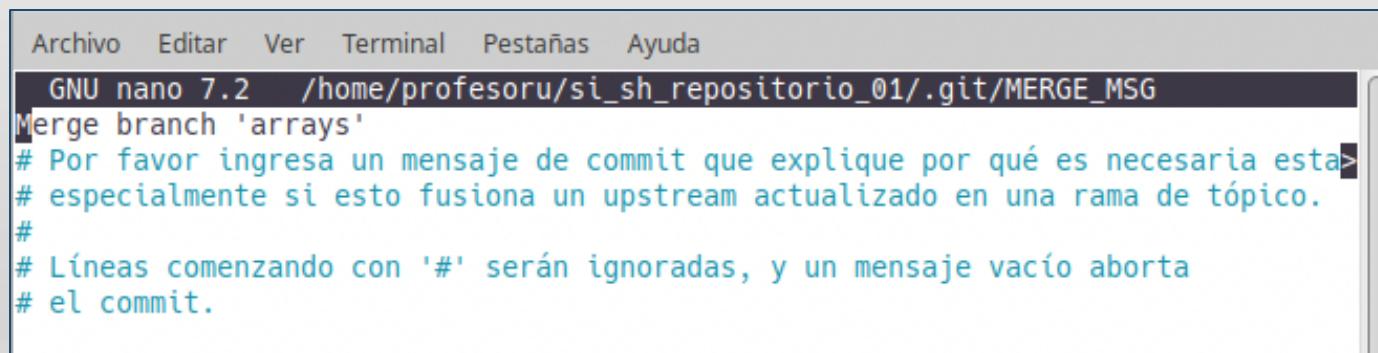
8. Ramas en Git y GitHub

Supuesto 3.

```
git add .  
git commit -m "Modificado fichero con arrays"  
git push origin arrays  
git log --oneline --all  
git checkout main  
git merge arrays
```

¿Qué ocurre?

Tenemos que hacer un commit que explique lo que ha pasado



```
Archivo Editar Ver Terminal Pestañas Ayuda  
GNU nano 7.2 /home/profesoru/si_sh_repositorio_01/.git/MERGE_MSG  
Merge branch 'arrays'  
# Por favor ingresa un mensaje de commit que explique por qué es necesaria esta  
# especialmente si esto fusiona un upstream actualizado en una rama de tópico.  
#  
# Líneas comenzando con '#' serán ignoradas, y un mensaje vacío aborta  
# el commit.
```

8. Ramas en Git y GitHub

Supuesto 3.

Editamos el mensaje

```
GNU nano 7.2  /home/profesoru/si_sh_repositorio_01/.git/MERGE_MSG *
Merge branch 'arrays' sin conflicto
# Por favor ingresa un mensaje de commit que explique por qué es necesaria esta>
# especialmente si esto fusiona un upstream actualizado en una rama de tópico.
#
# Líneas comenzando con '#' serán ignoradas, y un mensaje vacío aborta
# el commit.
```

Y salimos guardando cambios.

```
git log --oneline --all
```

```
39eed4c (HEAD -> main) Merge branch 'arrays' sin conflicto
2e00665 (origin/arrays, arrays) Modificado fichero con arrays
ec72c66 (origin/main) Tres scripts
dfbf265 Creado fichero con arrays
```

8. Ramas en Git y GitHub

Supuesto 3.

- Esto que acabamos de hacer se conoce como:

Merge Automático (Sin conflictos)

Ocurre cuando ambas ramas han avanzado y Git necesita unirlas con un commit extra.

8. Ramas en Git y GitHub

Supuesto 4.

- Ahora vamos a suponer que editamos el fichero prueba_01_TuID.sh en la rama funciones con la siguiente información y que confirmamos cambios y los publicamos en GitHub:

```
git checkout funciones  
code prueba_01_TuID.sh &
```

```
#!/bin/bash  
clear  
. funciones/funciones.sh  
saludo  
echo "Soy el script 01 en la rama funciones"  
echo "Volvemos a saludar":  
saludo  
exit 0
```

```
git commit -am "Saludamos dos veces en rama funciones"  
git push
```

8. Ramas en Git y GitHub

Supuesto 4.

- Y que en la rama main editamos el fichero prueba_01_TuID.sh con la siguiente información creando un conflicto porque el commit anterior solo pertenece a la funciones:

```
git checkout main
```

```
git log --oneline (Observa que lo anterior aún no está aquí)
```

```
5ddc1ea (HEAD -> main, origin/main) Creado fichero con funciones
6d6e871 Datos prueba
00ed54c Cambios en pruebas.txt
```

Editamos el fichero para crear el conflicto:

8. Ramas en Git y GitHub

- **Supuesto 4.** Y que en la rama main editamos el fichero prueba_01_TuID.sh con la siguiente información creando un conflicto porque el commit anterior solo pertenece a la funciones:

```
code prueba_01_TuID.sh &
```

A screenshot of a terminal window titled "prueba_01_TuID.sh" located at "~/si_sh_repositorio_01". The window has standard OS X-style controls (Minimize, Maximize, Close) at the top right. The code in the terminal is as follows:

```
#!/bin/bash
clear
. funciones/funciones.sh
#saludo
echo "Soy el script 01 en la rama main."
echo "Decidimos no saludar"
exit 0
```

Nota: Deja el fichero abierto en segundo plano.

En la rama funciones

```
#!/bin/bash
clear
. funciones/funciones.sh
saludo
echo "Soy el script 01 en la rama funciones"
echo "Volvemos a saludar":
saludo
exit 0
```

8. Ramas en Git y GitHub

- **Supuesto 4.** Confirmamos los cambios y publicamos en GitHub:

```
git commit -am "No saludamos en rama main"
```

```
git push
```

```
profesoru ~ main U:1 ~/si_sh_repositorio_01 git commit -am "No saludamos  
en la rama main"  
[main b4bf859] No saludamos en la rama main  
 1 file changed, 3 insertions(+), 2 deletions(-)  
profesoru ~ main ↑ ~/si_sh_repositorio_01 git push  
Enumerando objetos: 5, listo.  
Contando objetos: 100% (5/5), listo.  
Compresión delta usando hasta 2 hilos  
Comprimiendo objetos: 100% (3/3), listo.  
Escribiendo objetos: 100% (3/3), 379 bytes | 379.00 KiB/s, listo.  
Total 3 (delta 1), reusados 0 (delta 0), pack-reusados 0  
remote: Resolving deltas: 100% (1/1), completed with 1 local object.  
To https://github.com/amontorosi/si_sh_repositorio_01.git  
 5ddclea..b4bf859 main -> main
```

8. Ramas en Git y GitHub

- **Supuesto 4.** Antes de seguir, si vamos GitHub:

A screenshot of a GitHub repository page. At the top, there's a yellow banner with the message "funciones had recent pushes 7 minutes ago" and a green "Compare & pull request" button. Below the banner, there are navigation buttons for "main" (selected), "2 branches", "0 tags", and "Code". On the right, there are "Go to file", "Add file", and "Code" buttons. The main area shows a list of commits:

Author	Commit Message	Date
amontorosi	No saludamos en la rama main	b4bf859 1 minute ago
funciones	Creado fichero con funciones	9 hours ago
pruebas	Datos prueba	9 hours ago
prueba_01_TuID.sh	No saludamos en la rama main	1 minute ago
prueba_02_TuID.sh	Añadidos mensajes a los dos scripts	11 hours ago

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).



base: main ▾



compare: funciones ▾



✗ **Can't automatically merge.** Don't worry, you can still create the pull request.

8. Ramas en Git y GitHub

- **Supuesto 4.** Vamos a intentar fusionar. Muévete a la rama principal (aunque ya deberías estar en ella) y muestra el histórico de commit para comprobar que el último commit de la rama funciones no está aquí, pero sí el último que hemos hecho en la última edición del fichero sh:

```
git checkout main  
git log --oneline
```

```
b4bf859 (HEAD -> main, origin/main) No saludamos en la rama main  
5ddc1ea Creado fichero con funciones  
6d6e871 Datos prueba  
00ed54c Cambios en pruebas.txt  
4cfelab Update pruebas.txt  
3e32440 Carpeta para pruebas  
6b06alf Añadidos mensajes a los dos scripts  
4e0b71e Dos script  
7a89289 Incluyendo primer script  
(END)
```

```
git log --oneline --all
```

8. Ramas en Git y GitHub

- **Supuesto 4.** Ahora combinamos la rama funciones en la rama main y observamos el mensaje:

```
git merge funciones
```

```
profesoru ~ main ~ /si_sh_repositorio_01 git merge funciones
Auto-fusionando prueba_01_TuID.sh
CONFLICTO (contenido): Conflicto de fusión en prueba_01_TuID.sh
Fusión automática falló; arregle los conflictos y luego realice un commit con el
resultado.
profesoru ~ main S:1 U:1 ~ /si_sh_repositorio_01 1
```

```
git status
```

Nota: Si hubiera que fusionar otros ficheros y esos no presentarán conflictos se fusionarían directamente y solo estaríamos pendientes de corregir los conflictos en el fichero o ficheros que nos indique Git.

8. Ramas en Git y GitHub

- **Supuesto 4.** Recarga el fichero que tiene el conflicto y observa cómo Git ha intentado fusionar los cambios de dos ramas diferentes y como no ha podido te muestra lo que ha intentado:

El archivo «/home/profesoru/si_sh_re...orio_01/prueba_01_TuID.sh» ha cambiado en el disco.

Recargar



```
#!/bin/bash
clear
. funciones/funciones.sh
<<<<< HEAD
#saludo
echo "Soy el script 01 en la rama main."
echo "Decidimos no saludar"
=====
saludo
echo "Soy el script 01 en la rama funcines"
echo "Volvemos a saludar":
saludo
>>>>> funciones
exit 0
```

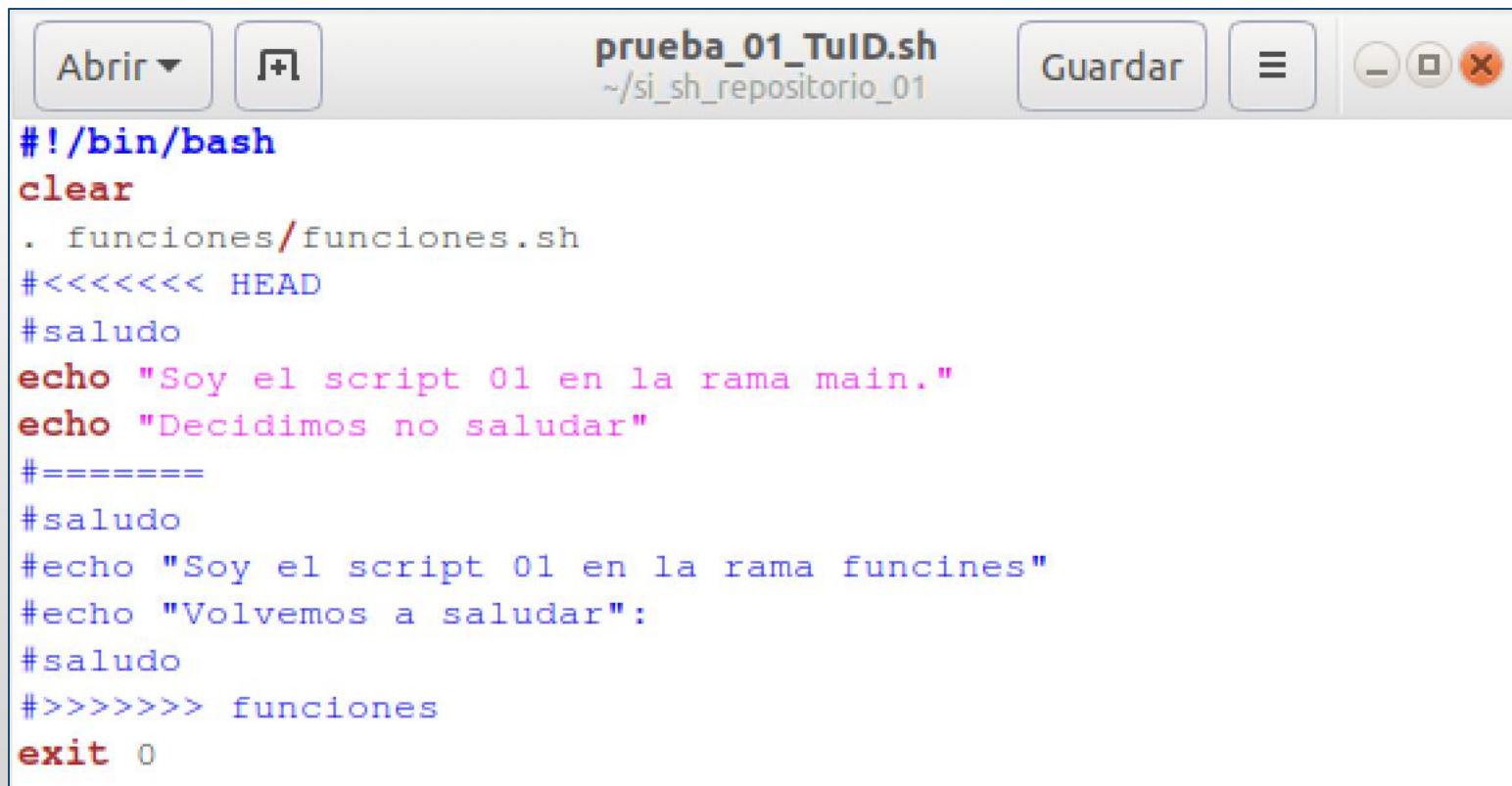
8. Ramas en Git y GitHub

- **Supuesto 4.** En code se verá así:

```
home > profesor > si_sh_repositorio_01 > $ prueba_01_TuID.sh
1  #!/bin/bash
2  clear
3  . funciones/funciones.sh
4  saludo
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
5  <<<<< HEAD (Current Change)
6  echo "Soy el script 01 en la rama main."
7  echo "Decidimos no saludar"
8  =====
9  echo "Soy el script 01 en la rama funciones"
10 echo "Volvemos a saludar"
11 >>>>> funciones (Incoming Change)
12 exit 0
```

8. Ramas en Git y GitHub

- **Supuesto 4.** En este caso, tenemos que intervenir nosotros manualmente y arreglar el conflicto tras analizar el fichero.



The screenshot shows a terminal window with the following details:

- File name: prueba_01_TuID.sh
- Path: ~si_sh_repositorio_01
- Buttons: Abrir, Guardar, Minimize, Maximize, Close.

The content of the script is as follows:

```
#!/bin/bash
clear
. funciones/funciones.sh
#<<<<< HEAD
#saludo
echo "Soy el script 01 en la rama main."
echo "Decidimos no saludar"
=====
#saludo
#echo "Soy el script 01 en la rama funciones"
#echo "Volvemos a saludar":
#saludo
#>>>>> funciones
exit 0
```

The script contains a merge conflict, indicated by the line '#<<<<< HEAD' which is part of the code from the 'main' branch. The code from the 'funciones' branch is shown below it, starting with '#saludo' and ending with '#>>>>> funciones'.

8. Ramas en Git y GitHub

- **Supuesto 4.** Una vez arreglado hacemos un commit y comprobamos el histórico de commit y las ramas fusionadas.

```
git commit -am "Solucionado conflicto prueba_01_TuID.sh"  
git log --oneline --all
```

```
b067ed8 (HEAD -> main) Solucionado conflicto prueba_01_TuID.sh  
b4bf859 (origin/main) No saludamos en la rama main  
742679e (origin/funciones, funciones) Saludamos dos veces en rama funciones  
5ddc1ea Creado fichero con funciones  
6d6e871 Datos prueba  
00ed54c Cambios en pruebas.txt
```

```
git branch --merged
```

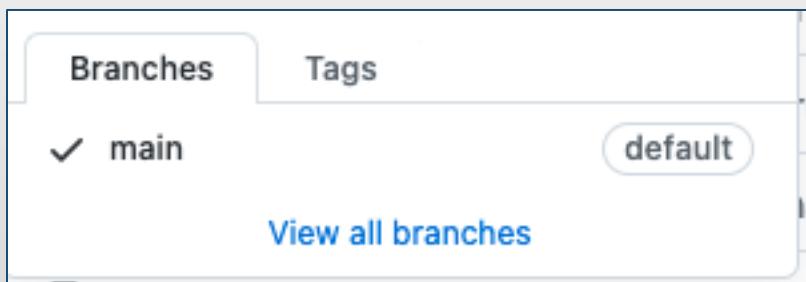
```
    funciones  
* main  
(END)
```

8. Ramas en Git y GitHub

- **Supuesto 4.** Ahora vamos a publicar los cambios en GitHub y dado que la rama funciones ya no la necesitamos más, la vamos a eliminar en GitHub. Solo en GitHub, en local la vamos a dejar de momento, aunque Git dejará de tenerla en cuenta también en local al borrar en remoto:

```
git push origin --delete funciones
```

```
profesoru ~ main ~ /si_sh_repositorio_01 129 git push origin --delete fu  
nciones  
To https://github.com/amontorosi/si_sh_repositorio_01.git  
 - [deleted]      funciones
```



```
git branch -a
```

```
funciones  
* main  
remotes/origin/main  
(END)
```

8. Ramas en Git y GitHub

Supuesto 4.

- Esto que acabamos de hacer se conoce como:

Merge Manual (Con conflictos)

Ocurre cuando ambas ramas han avanzado y Git necesita unirlas con un commit extra.

8. Ramas en Git y GitHub

- **Supuesto 4.** Recuerda si finalmente decidimos borrar la rama funciones también en local porque estamos seguro que no la necesitamos haremos. Aunque no lo vamos a hacer:

```
git branch -d funciones
```

```
git branch
```

8. Ramas en Git y GitHub

- **Supuesto 5.** Estamos trabajando en la rama funciones, creamos un fichero nuevo y estamos modificando un fichero que ya había sido comiteado, por ejemplo el fichero funciones.sh. De forma temporal tenemos que ir a la rama main a solucionar un problema urgente, pero dado que el fichero funciones.sh aún está a medias, no queremos hacer un commit aún. Para esto, existe la **pila stash**

\$ git stash

Almacena temporalmente todos los archivos tracked modificados

\$ git stash pop

Restaura los archivos guardados más recientemente

\$ git stash list

Enumera todos los sets de cambios en guardado rápido

\$ git stash drop

Elimina el set de cambios en guardado rápido más reciente

8. Ramas en Git y GitHub

■ Supuesto 5.

```
git checkout funciones
echo "exit 0" > funciones/funciones_aux.sh
echo "#Un comentario" >> funciones/funciones.sh
git add .
git stash
git status -s
git stash list
git checkout main
git ... #Lo que tengamos que hacer urgente en main.
git checkout funciones
git stash list
git stash show -p stash@{0}
git stash pop
git add .
git commit -m "Finalizadas nuevas funciones y arreglos"
```

\$ git stash

Almacena temporalmente todos los archivos tracked modificados

\$ git stash pop

Restaura los archivos guardados más recientemente

\$ git stash list

Enumera todos los sets de cambios en guardado rápido

\$ git stash drop

Elimina el set de cambios en guardado rápido más reciente

8. Ramas en Git y GitHub

- **Supuesto 6.** Volvemos al supuesto 2. Imagina que estamos trabajando en una rama nueva, pero mientras estabamos trabajando, la rama main ha recibido nuevos cambios que no entran en conflicto con nuestra rama.

Ahora queremos actualizar nueva-funcionalidad para que esté al día con main, pero sin generar un commit de merge innecesario.

- Aquí es donde usamos git rebase.

8. Ramas en Git y GitHub

Tarea 10

- Usando Visual Studio Code y la carpeta y el repositorio `si_sh_repositorio_02`, repite todos los pasos realizados en este capítulo, pero ahora usando la interfaz gráfica de VS Code.

Índice de contenidos

1. Introducción
2. Conceptos
3. Flujo de trabajo
4. Instalación de Git
5. Primeros pasos
6. Uso básico de Git
7. Uso básico de GitHub
8. Ramas en Git y GitHub
- 9. Trabajo colaborativo**
10. Algunos comandos útiles

9. Trabajo colaborativo

COLABORADOR

- La forma habitual de contribuir a un proyecto en el que trabajamos en equipo es que nos agreguen como colaboradores:

The screenshot shows a GitHub repository page for 'amontorosi/si_sh_repositorio_01'. The repository is public. At the top, there are buttons for Pin, Unwatch (1), Fork (1), and Star (0). Below the repository name, there are links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The Settings link is underlined, indicating it is active. On the left, a sidebar has tabs for General, Access, Collaborators, and Moderation options. The General tab is selected. The main content area shows the repository name 'si_sh_repositorio_01' in a field with a 'Rename' button. There is also a checkbox for 'Template repository' with a descriptive note below it.

amontorosi / si_sh_repositorio_01 Public

Pin Unwatch 1 Fork 1 Star 0

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

General

Access

Collaborators

Moderation options

Repository name: si_sh_repositorio_01

Template repository

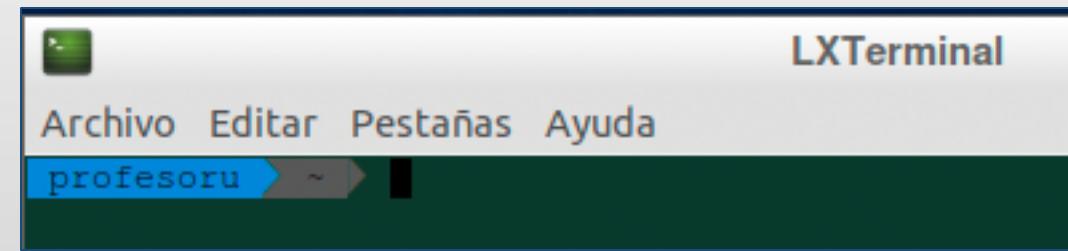
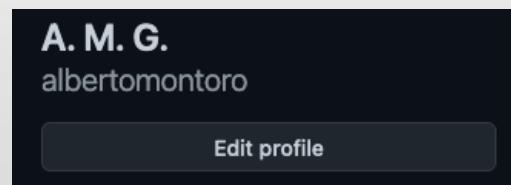
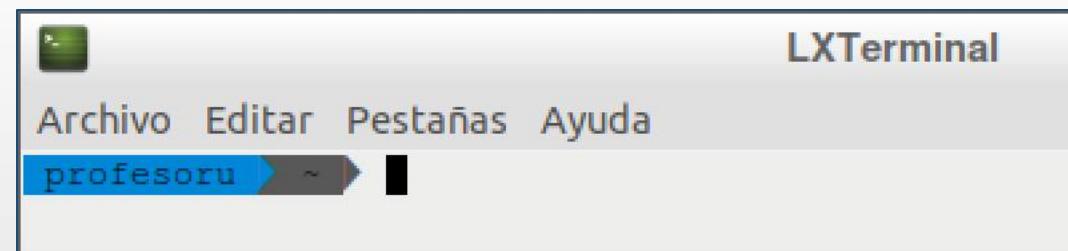
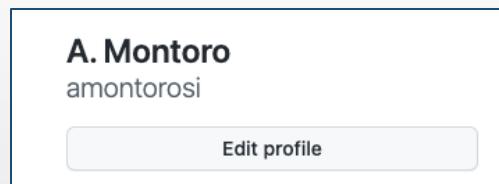
Template repositories let users generate new repositories with the same directory structure and files. [Learn more.](#)

El propietario del repositorio debe decidir qué permisos nos asigna y nos puede permitir crear y fusionar ramas sin necesidad de "aprobación" y gestionar issues y pull requests.

9. Trabajo colaborativo

COLABORADOR

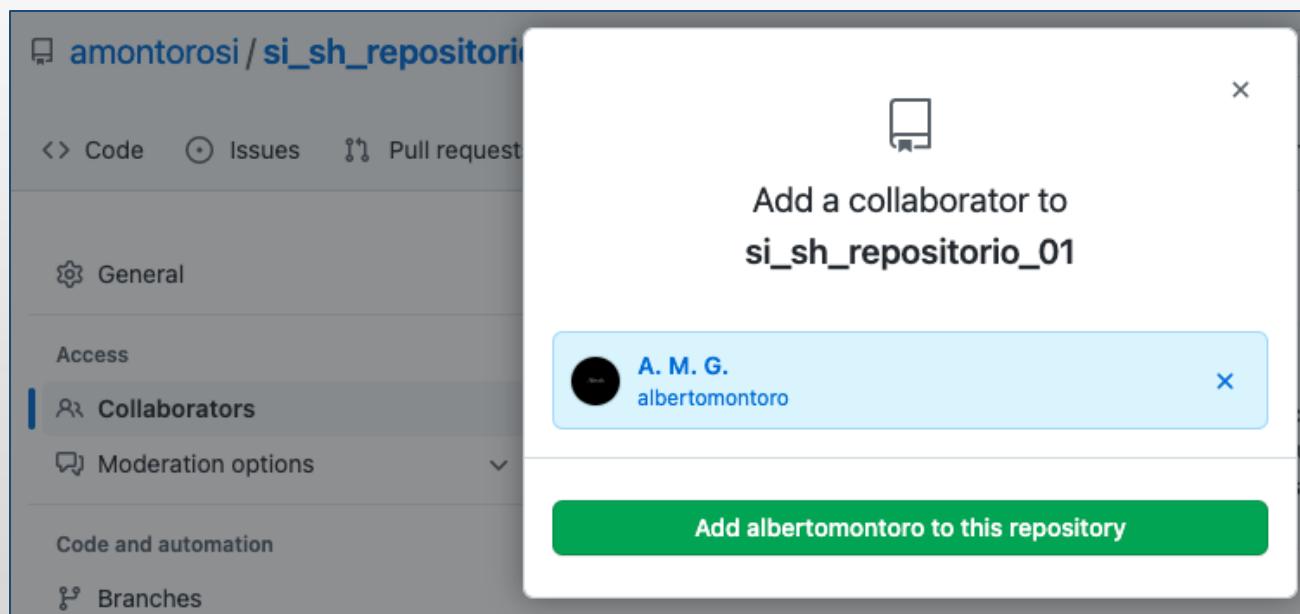
- Aquí para estos ejemplos, voy a trabajar con dos usuarios y dos equipos diferentes:



9. Trabajo colaborativo

COLABORADOR

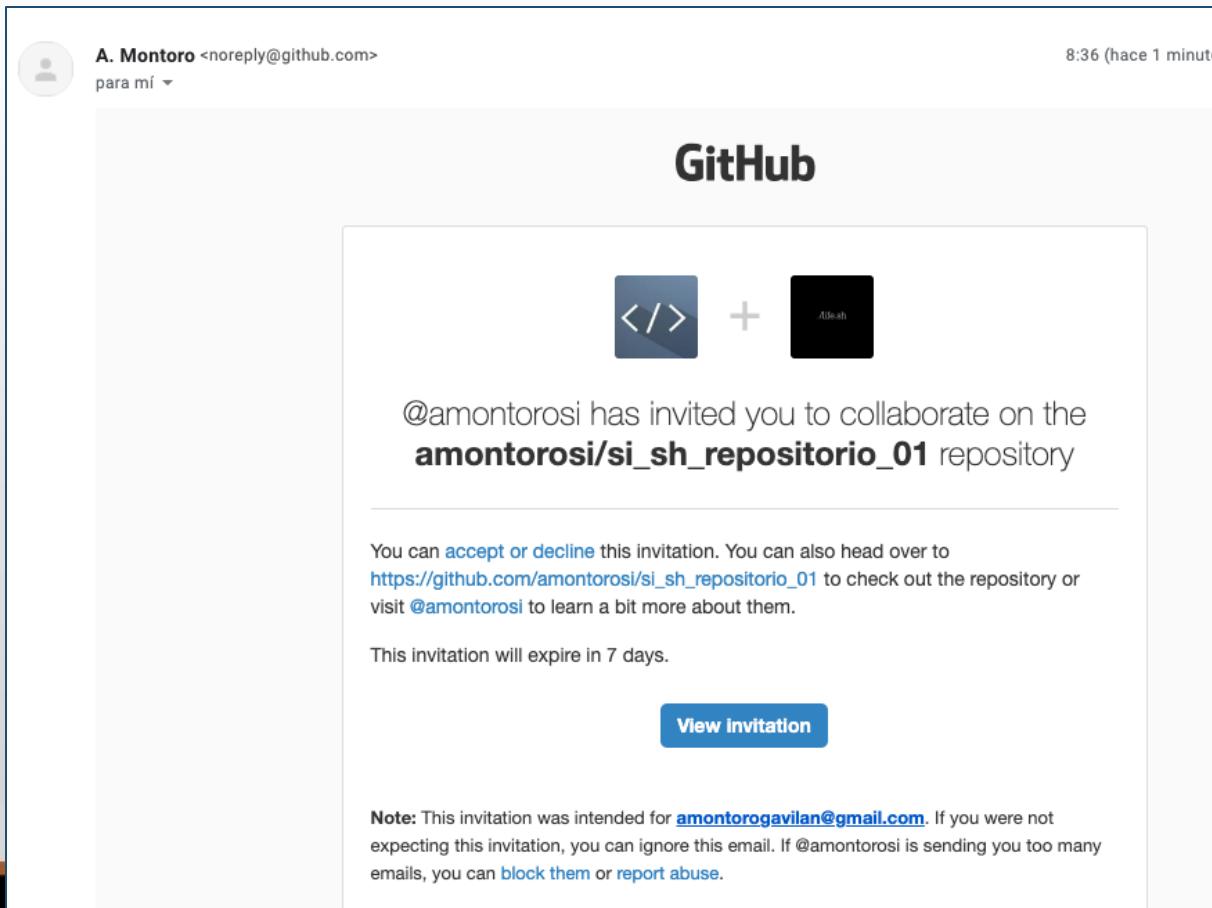
- En este ejemplo vamos a agregar al desarrollador 2 como COLABORADOR del repositorio si_sh_repositorio_01 del desarrollador 1:



9. Trabajo colaborativo

COLABORADOR

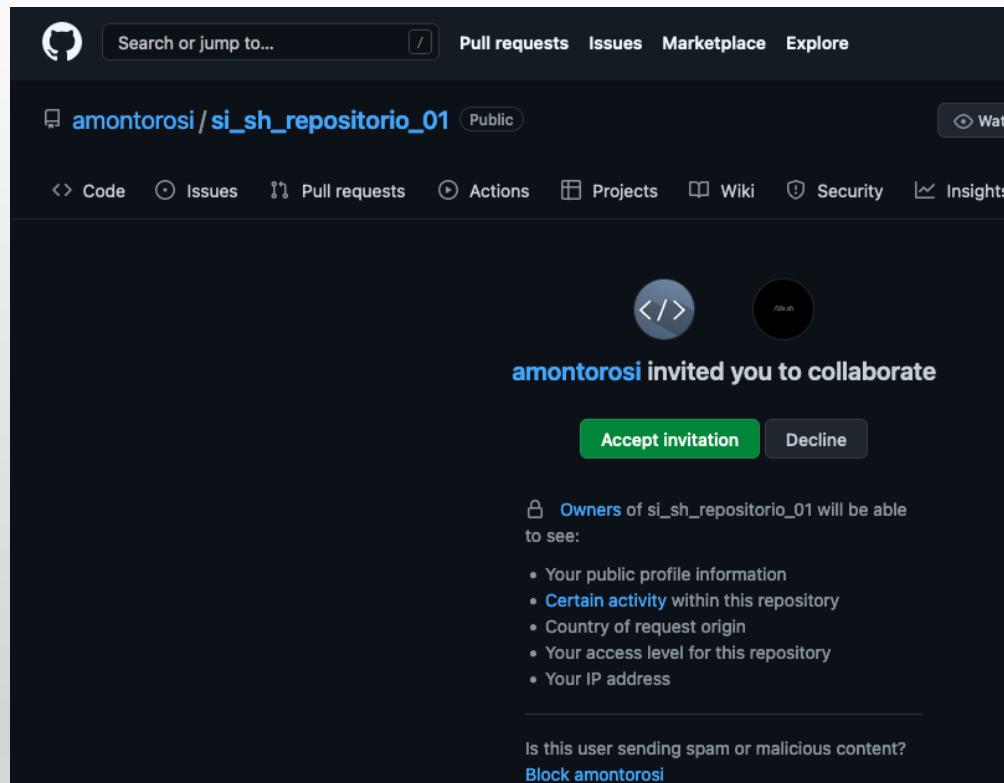
- En ese momento el desarrollador 2 recibe una invitación en su correo electrónico que debe aceptar antes de 7 días (**esto es muy importante**):



9. Trabajo colaborativo

COLABORADOR

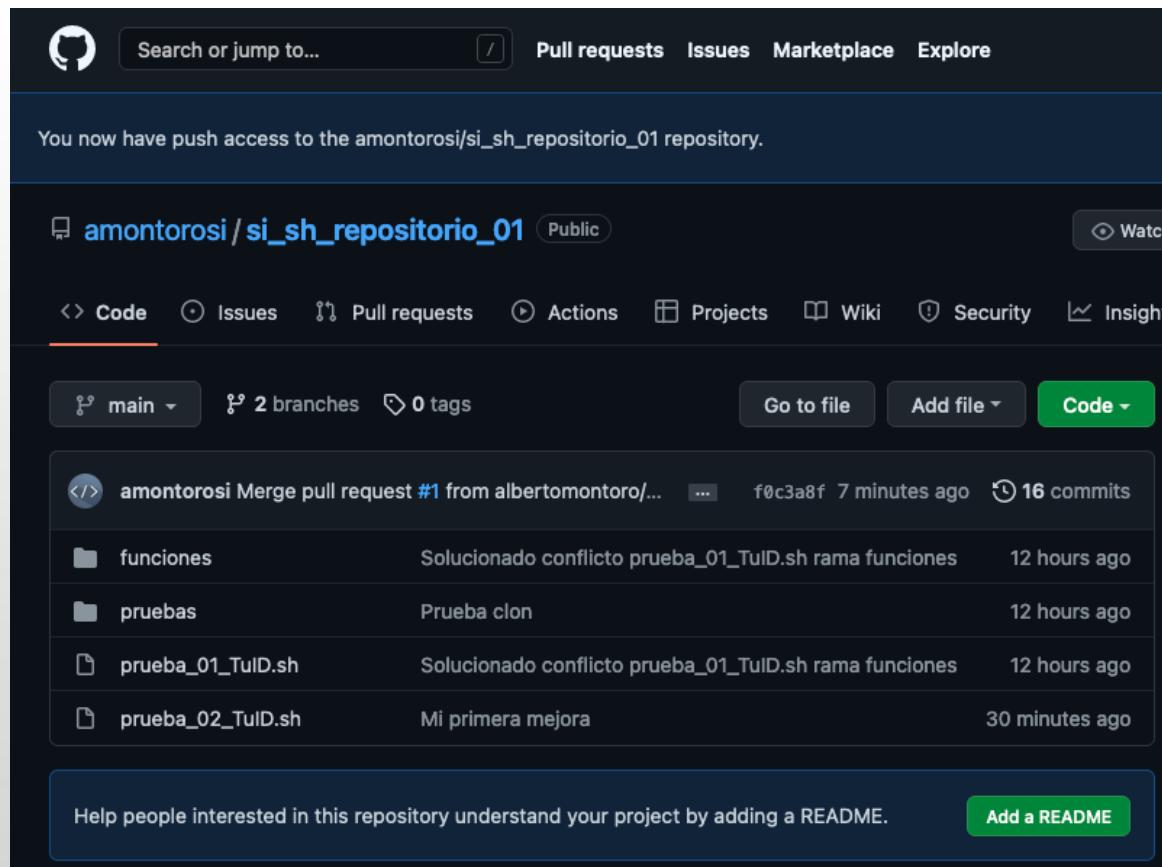
- Al pulsar la invitación del correo, nos aparece una ventana en GitHub y tengo que aceptar o rechazar:



9. Trabajo colaborativo

COLABORADOR

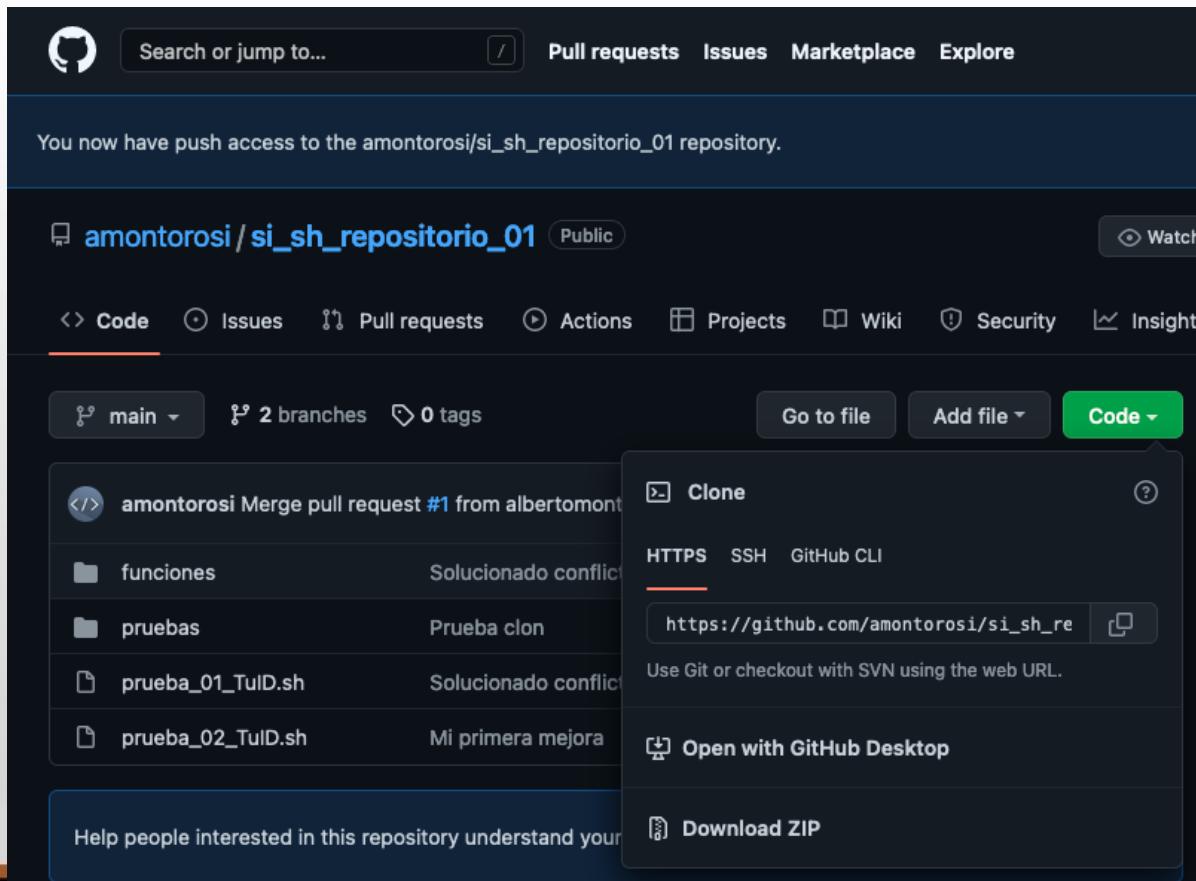
- Y al aceptarla:



9. Trabajo colaborativo

COLABORADOR

- Ahora este desarrollador 2 puede clonarse el repositorio del desarrollador 1 en su máquina local y hacer aportaciones:



9. Trabajo colaborativo

COLABORADOR

- Esta aportaciones irán directamente al repositorio en GitHub del desarrollador 1 al hacer push:

```
mkdir mis_proyectos_colaborativos  
cd mis_proyectos_colaborativos  
git clone https://github.com/amontorosi/si_sh_repositorio_01.git
```

```
profesoru ➤ ~/mis_proyectos_colaborativos ➤ git clone https://github.com/amonto  
rosi/si_sh_repositorio_01.git  
Clonando en 'si_sh_repositorio_01'...  
remote: Enumerating objects: 53, done.  
remote: Counting objects: 100% (53/53), done.  
remote: Compressing objects: 100% (34/34), done.  
remote: Total 53 (delta 14), reused 42 (delta 7), pack-reused 0  
Desempaquetando objetos: 100% (53/53), listo.  
profesoru ➤ ~/mis_proyectos_colaborativos ➤
```

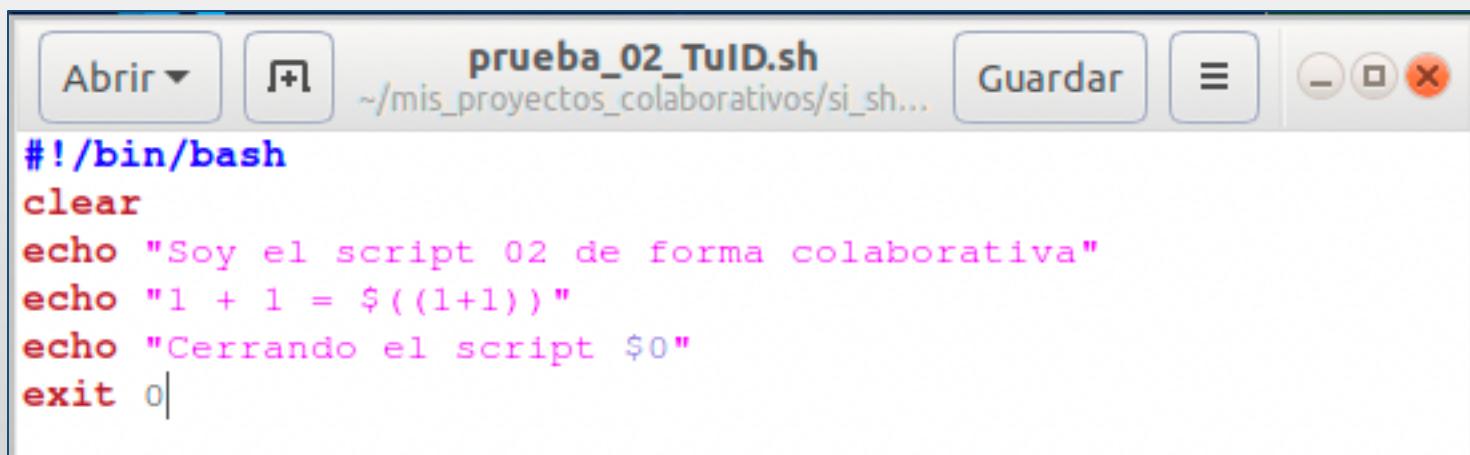
Entendemos que el desarrollador 2 también tiene configurado git y sus credenciales de GitHub en su equipo local.

9. Trabajo colaborativo

COLABORADOR

- Esta aportaciones irán directamente al repositorio en GitHub del desarrollador 1 al hacer push:

```
cd si_sh_repositorio_01  
git pull (Solo por seguridad)  
code prueba_02_TuID.sh &
```



The screenshot shows a terminal window with the following details:

- File menu: "Abrir" (Open) with a dropdown arrow.
- New file icon.
- Title bar: "prueba_02_TuID.sh" and the path "~/mis_proyectos_colaborativos/si_sh...".
- Save button: "Guardar".
- Minimize, maximize, and close buttons.
- Script content:

```
#!/bin/bash  
clear  
echo "Soy el script 02 de forma colaborativa"  
echo "1 + 1 = $((1+1))"  
echo "Cerrando el script $0"  
exit 0|
```

9. Trabajo colaborativo

COLABORADOR

- Esta aportaciones irán directamente al repositorio en GitHub del desarrollador 1 al hacer push:

```
git add .
```

```
git commit -m "Arreglando el script 02"
```

```
git push
```

```
profesoru ~ main ~/mis_proyectos_colaborativos/si_sh_repositorio_01 git add .
[2]+  Hecho                  gedit prueba_02_TuID.sh
profesoru ~ main S:1 ~/mis_proyectos_colaborativos/si_sh_repositorio_01 git commit -m "Arreglando el script 2"
[main a8f3de4] Arreglando el script 2
 1 file changed, 1 insertion(+), 1 deletion(-)
profesoru ~ main ↑ ~/mis_proyectos_colaborativos/si_sh_repositorio_01 git push
Username for 'https://github.com': albertomontoro
Password for 'https://albertomontoro@github.com':
Contando objetos: 3, listo.
Delta compression using up to 2 threads.
Comprimiendo objetos: 100% (3/3), listo.
Escribiendo objetos: 100% (3/3), 374 bytes | 374.00 KiB/s, listo.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/amontorosi/si_sh_repositorio_01.git
  f0c3a8f..a8f3de4  main -> main
```

9. Trabajo colaborativo

COLABORADOR

- Apareciendo dichos cambios en el repositorio original:

The screenshot shows a GitHub repository page for 'amontorosi/si_sh_repositorio_01'. The repository is public. The main tab is selected, showing the file 'prueba_02_TuID.sh' which has been updated by 'A. M. G' with the commit message 'Arreglando el script 2'. The file has 1 contributor. It is an executable file with 6 lines (6 sloc) and 124 Bytes. The code content is:

```
1 #!/bin/bash
2 clear
3 echo "Soy el script 02 de forma colaborativa"
4 echo "1 + 1 = $((1+1))"
5 echo "Cerrando el script $0"
6 exit 0
```

9. Trabajo colaborativo

COLABORADOR

- Cuando varios colaboradores trabajan en el mismo repositorio, pueden ocurrir conflictos de Git si dos personas modifican el mismo archivo y suben cambios al repositorio. Si Git detecta que hay diferencias incompatibles entre las versiones, te pedirá que resuelvas el conflicto manualmente antes de hacer un merge o pull. Para solucionarlo, revisa los cambios en el archivo, elige qué versión conservar y confirma el merge con un commit.

9. Trabajo colaborativo

COLABORADOR

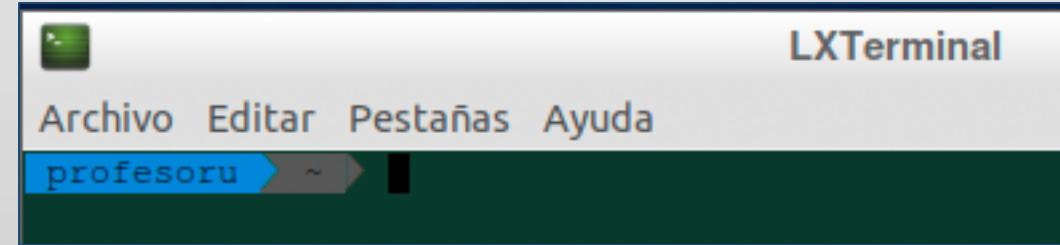
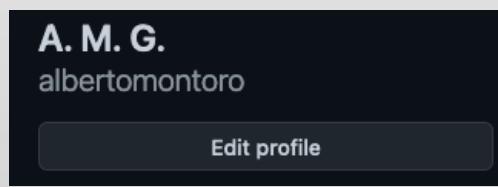
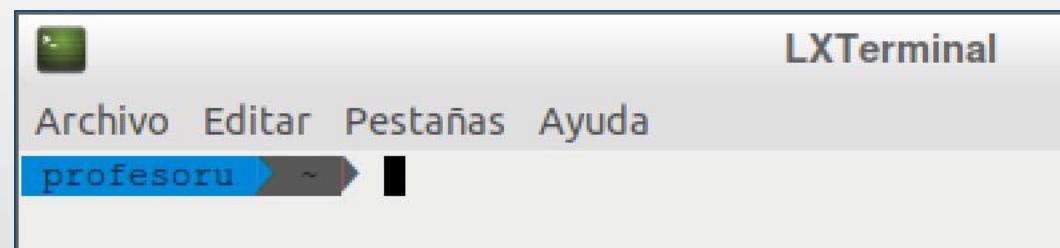
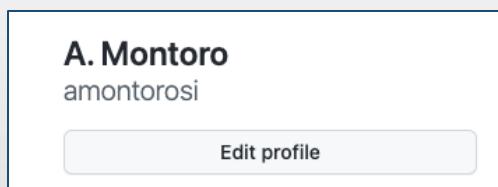
■ Tarea 11:

- En esta tarea vamos a formar parejas y usando la interfaz gráfica de Visual Studio Code, vamos a realizar **los pasos de este capítulo relativos a Colaborador** sobre el repositorio `si_sh_repositorio_02` de uno de los miembros de la pareja.
- Pruebas a realizar:
 - Crear archivos diferentes → No debe haber conflictos.
 - Editar el mismo archivo en líneas diferentes → Git debe fusionar los cambios automáticamente.
 - Editar exactamente la misma línea → Se generará un conflicto, que deberás resolver usando la interfaz gráfica de VS Code.
 - Subir los cambios y revisar el historial en GitHub.
- Extras: Investiga el uso `git pull --rebase` en vez de `git pull` para ver la diferencia en el historial.

9. Trabajo colaborativo

FORK

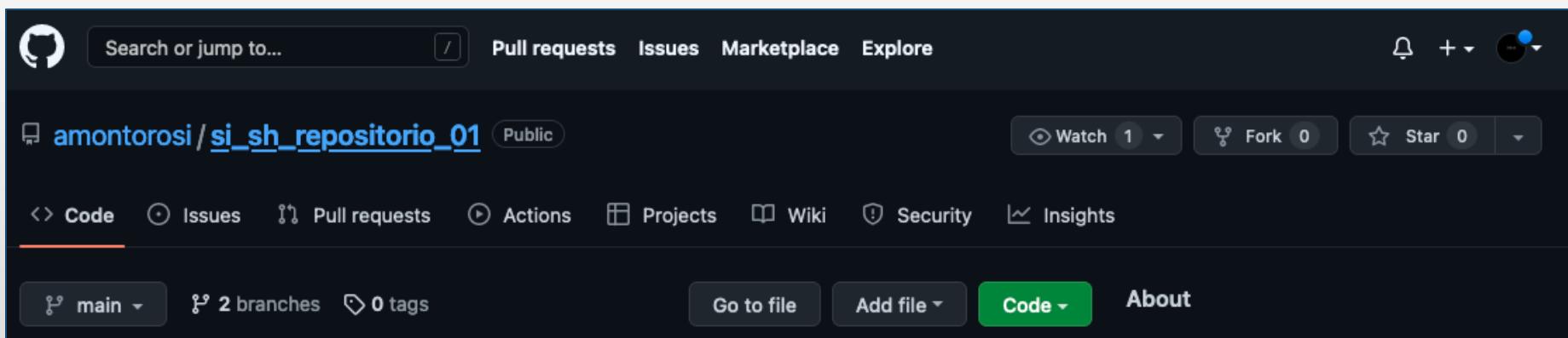
- En este punto vamos a suponer que un segundo desarrollador quiere desarrollar su propio proyecto a partir del nuestro y luego además quiere aportar algo nuevo a nuestro proyecto. Esto se hace a través de un fork.
- Por tanto, aquí volvemos a trabajar con dos usuarios y dos equipos diferentes:



9. Trabajo colaborativo

FORK

- El segundo usuario, encuentra nuestro proyecto, le parece interesante y decide crear su propio proyecto a partir del nuestro.
- Esto en GitHub es muy sencillo:



9. Trabajo colaborativo

Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

The screenshot shows the GitHub fork creation interface and the resulting repository page.

Fork Creation Interface:

- Owner ***: albertomontoro
- Repository name ***: si_sh_repositorio_01
- Description (optional)**: (empty)
- Note**: You are creating a fork in your personal account.
- Create fork** button

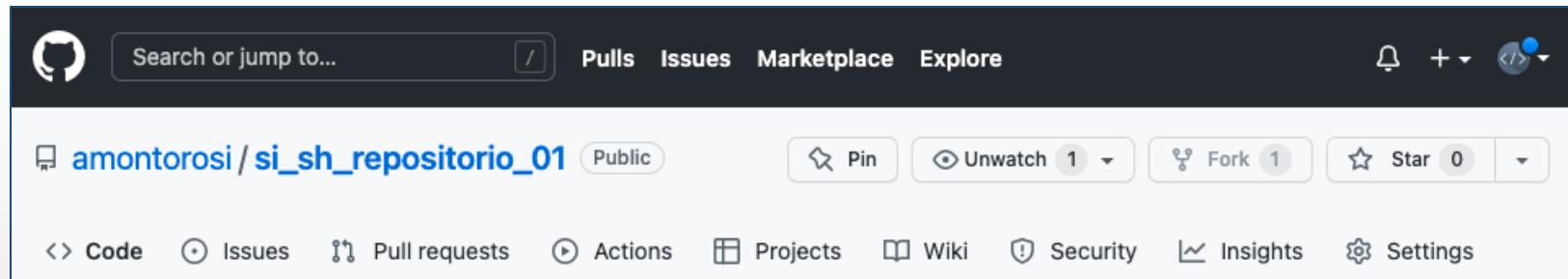
Repository Page:

- Repository URL**: albertomontoro / si_sh_repositorio_01 (Public)
- Source**: forked from amontorosi/si_sh_repositorio_01
- Navigation**: Code, Pull requests, Actions, Projects, Wiki, Security, Insights, Settings
- Branches**: main, 2 branches, 0 tags
- Status**: This branch is up to date with amontorosi/si_sh_repositorio_01:main.
- Commits**:
 - amontorosi Prueba clon (03d2344, 11 hours ago, 14 commits)
 - funciones: Solucionado conflicto prueba_01_TuID.sh rama funciones (11 hours ago)
 - pruebas: Prueba clon (11 hours ago)
 - prueba_01_TuID.sh: Solucionado conflicto prueba_01_TuID.sh rama funciones (11 hours ago)
 - prueba_02_TuID.sh: Añadidos mensajes a los dos script (24 days ago)
- README**: Help people interested in this repository understand your project by adding a README. Add a README button

9. Trabajo colaborativo

FORK

- El primer desarrollador verá que ha ocurrido esto:



9. Trabajo colaborativo

FORK

- Hecho esto, el segundo desarrollador clona este repositorio (que se ha creado tras un fork en su GitHub) en su equipo local y añade su propios cambios al fichero prueba_02_TuID.sh.

```
git clone https://github.com/albertomontoro/si_sh_repositorio_01.git
```

```
profesoru ~ / mis_proyectos > git clone https://github.com/albertomontoro/si_sh_repositorio_01.git
```

```
git pull
```

```
code prueba_02_TuID.sh &
```

A screenshot of a terminal window titled 'prueba_02_TuID.sh'. The window has a toolbar with buttons for 'Abrir', 'Guardar', and others. The script content is as follows:

```
#!/bin/bash
clear
echo "Soy el script 0"
echo "1 + 1 = $((1+1))"
echo "Cerrando el script $0"
exit 0
```

9. Trabajo colaborativo

FORK

- Finalmente sube los cambios a su repositorio:

```
git add .
```

```
git commit -m "Mi primera mejora"
```

```
profesoru ~ main U:l ~/mis_proyectos/si_sh_repositorio_01 git add .
profesoru ~ main S:l ~/mis_proyectos/si_sh_repositorio_01 git commit -m "
Mi primera mejora"
[main 8149d22] Mi primera mejora
 1 file changed, 3 insertions(+), 1 deletion(-)
```

```
git log --oneline
```

```
8149d22 (HEAD -> main) Mi primera mejora
03d2344 (origin/main, origin/HEAD) Prueba clon
527d283 Solucionado conflicto prueba_01_TUID.sh rama funciones
37b6807 Creando un conflicto entre ramas
90a061a Cambios en pruebas.txt
67f2395 Update pruebas.txt
2aae7fd Carpeta para pruebas
94a9417 (origin/funciones) Creado fichero con funciones
```

9. Trabajo colaborativo

FORK

- Finalmente sube los cambios a su repositorio:

```
git push
```

```
profesoru ~ main ↑ ~ /mis_proyectos/si_sh_repositorio_01 git push
Username for 'https://github.com': albertomontoro
Password for 'https://albertomontoro@github.com':
Contando objetos: 3, listo.
Delta compression using up to 2 threads.
Comprimiendo objetos: 100% (3/3), listo.
Escribiendo objetos: 100% (3/3), 349 bytes | 349.00 KiB/s, listo.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/albertomontoro/si_sh_repositorio_01.git
  03d2344..8149d22  main -> main
profesoru ~ main ~ /mis_proyectos/si_sh_repositorio_01
```

- En su GitHub ve esto:

A. M. G Mi primera mejora	8149d22 7 minutes ago	15 commits
funciones	Solucionado conflicto prueba_01_TuiD.sh rama funciones	12 hours ago
pruebas	Prueba clon	11 hours ago
prueba_01_TuiD.sh	Solucionado conflicto prueba_01_TuiD.sh rama funciones	12 hours ago
prueba_02_TuiD.sh	Mi primera mejora	7 minutes ago

9. Trabajo colaborativo

FORK

- Como es lógico esto no afecta al desarrollador original:

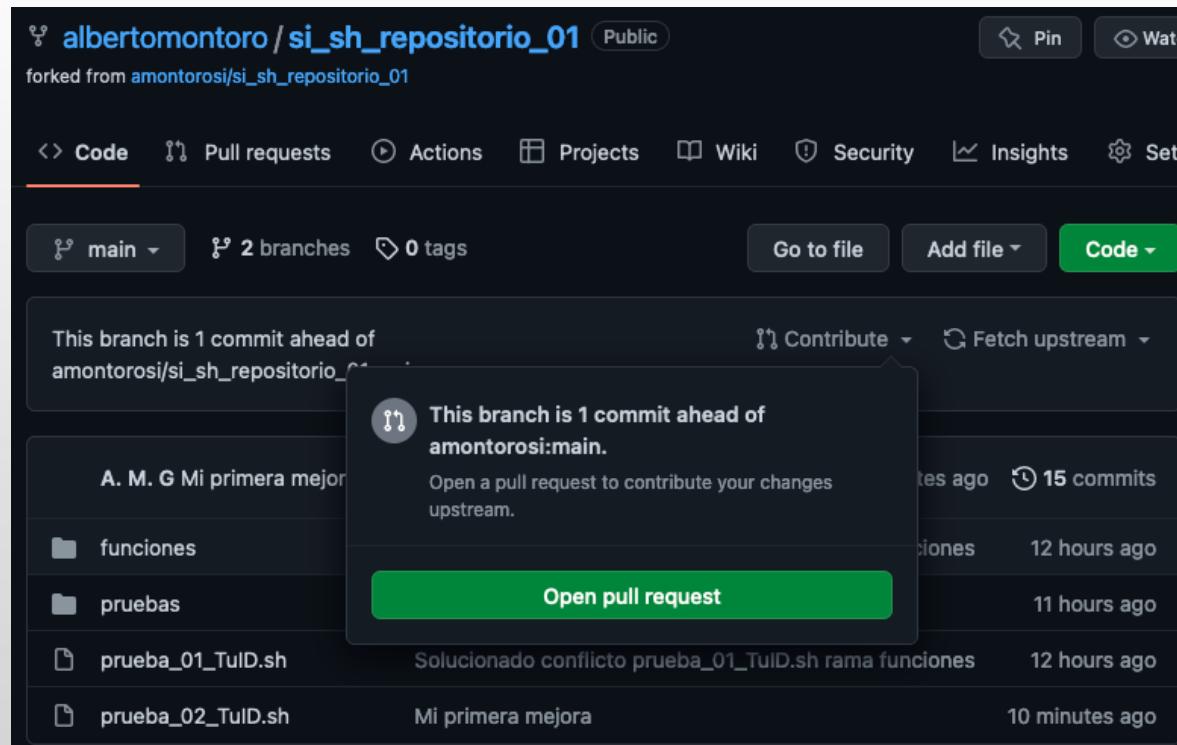
The screenshot shows a GitHub repository page for 'amontorosi / si_sh_repositorio_01'. The repository is public. At the top, there are navigation links: Code (which is underlined), Issues, Pull requests, Actions, Projects, Wiki, and Security. Below these are buttons for main (selected), Go to file, Add file, and Code. A dropdown menu for the 'main' branch is open. The main content area displays a list of commits from the 'main' branch. The commits are:

Author	Commit Message	Time Ago	Reactions
amontorosi	Prueba clon	11 hours ago	14
	funciones Solucionado conflicto prueba_01_TulD.sh ram...	12 hours ago	
	pruebas Prueba clon	11 hours ago	
	prueba_01_TulD.sh Solucionado conflicto prueba_01_TulD.sh ram...	12 hours ago	
	prueba_02_TulD.sh Añadidos mensajes a los dos script	24 days ago	

9. Trabajo colaborativo

FORK

- Supongamos que ahora el segundo desarrollador quiere compartir estos cambios con el desarrollador original:



9. Trabajo colaborativo

FORK

- Supongamos que ahora el segundo desarrollador quiere compartir estos cambios con el desarrollador original:

The screenshot shows a GitHub interface for comparing branches. At the top, it displays the base repository as `amontorosi/si_sh_repositorio...`, the base branch as `main`, the head repository as `albertomontoro/si_sh_repositorio...`, and the compare branch as `main`. A green checkmark indicates that the branches are able to be merged automatically. Below this, there's a message encouraging users to discuss and review the changes, with a link to learn about pull requests and a prominent green "Create pull request" button.

Key statistics shown are 1 commit, 1 file changed, and 1 contributor. The commit details show a single commit made on May 13, 2022, by user "A. M. G" with the message "Mi primera mejora". The commit hash is `8149d22`.

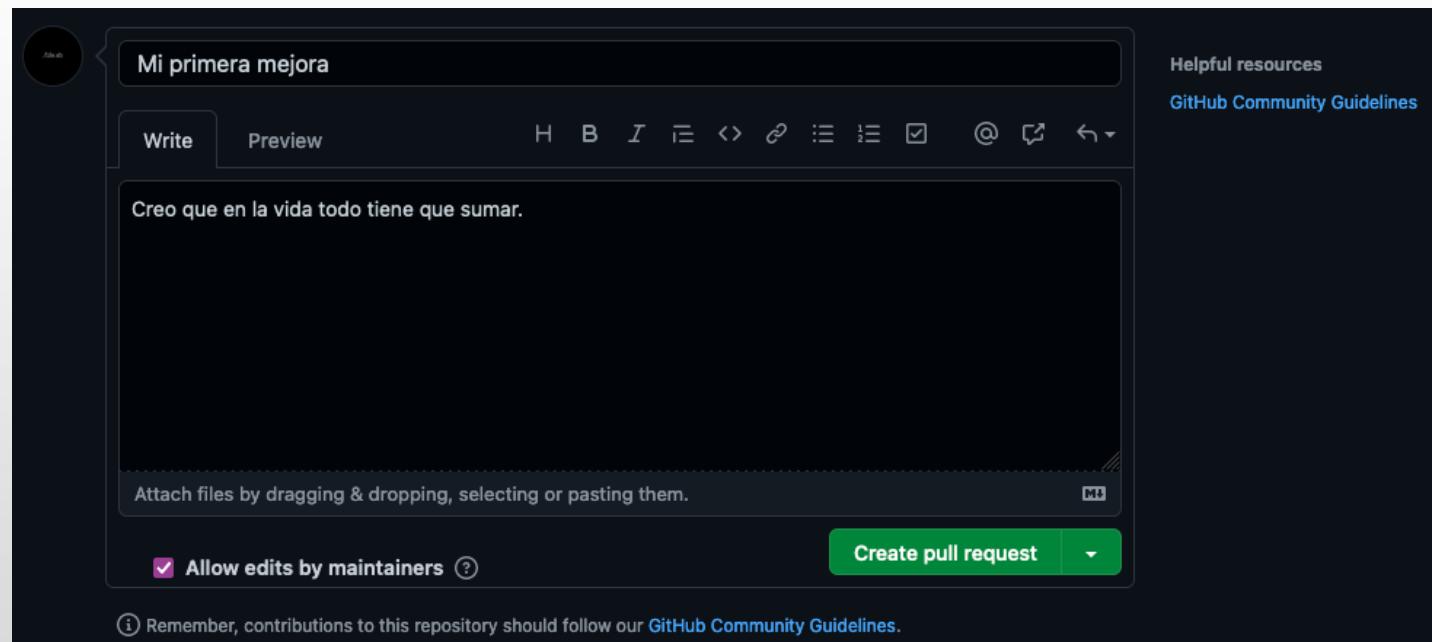
The "File changes" section shows a diff for the file `prueba_02_TuID.sh`. The diff highlights three additions and one deletion. The additions are:

```
@@ -1,4 +1,6 @@
 1      1      #!/bin/bash
 2      2      clear
 3      - echo "Soy el script 02"
 3      + echo "Soy el script 0"
 4      + echo "1 + 1 = $((1+1))"
 5      + echo "Cerrando el script $0"
 4      6      exit 0
```

9. Trabajo colaborativo

FORK

- Supongamos que ahora el segundo desarrollador quiere compartir estos cambios con el desarrollador original:



9. Trabajo colaborativo

FORK

- Supongamos que ahora el segundo desarrollador quiere compartir estos cambios con el desarrollador original:

The screenshot shows a GitHub pull request interface. At the top, a green button says "Open" and a message states "albertomontoro wants to merge 1 commit into `amontorosi:main` from `albertomontoro:main`". Below this, there are tabs for "Conversation" (0), "Commits" (1), "Checks" (0), and "Files changed" (1). A comment from "albertomontoro" is displayed, reading "Creo que en la vida todo tiene que sumar." Below the comment, a commit titled "Mi primera mejora" is shown with the commit hash "8149d22". A note below the commit says "Add more commits by pushing to the `main` branch on `albertomontoro/si_sh_repositorio_01`". A green box highlights the message "This branch has no conflicts with the base branch" with a checkmark icon. It also notes "Only those with [write access](#) to this repository can merge pull requests." At the bottom, there's a text input field "Leave a comment" and a "Comment" button.

albertomontoro wants to merge 1 commit into `amontorosi:main` from `albertomontoro:main`

Conversation 0 Commits 1 Checks 0 Files changed 1

albertomontoro commented now

Creo que en la vida todo tiene que sumar.

-o Mi primera mejora 8149d22

Add more commits by pushing to the `main` branch on `albertomontoro/si_sh_repositorio_01`.

This branch has no conflicts with the base branch
Only those with [write access](#) to this repository can merge pull requests.

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

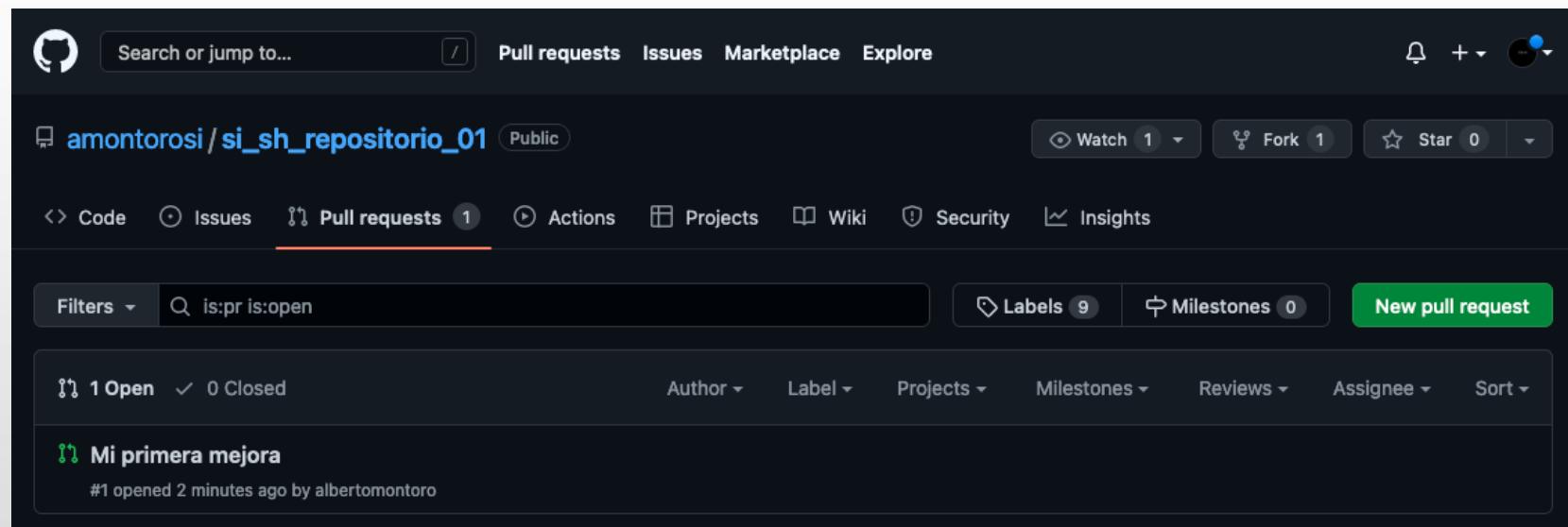
Close pull request Comment

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

9. Trabajo colaborativo

FORK

- Supongamos que ahora el segundo desarrollador quiere compartir estos cambios con el desarrollador original:



9. Trabajo colaborativo

FORK

- Finalmente, el desarrollador original se encuentra esto:

The screenshot shows a GitHub repository page for 'amontorosi/si_sh_repositorio_01'. The repository is public. At the top, there are buttons for Pin, Unwatch (1), Fork (1), and Star (0). Below the header, there are navigation links: Code, Issues, Pull requests (1, highlighted), Actions, Projects, Wiki, Security, Insights, and Settings. A modal window titled 'Label issues and pull requests for new contributors' is open, stating 'Now, GitHub will help potential first-time contributors discover issues labeled with good first issue' and has a 'Dismiss' button. Below the modal, there are filters: 'Filters' dropdown, search bar ('is:pr is:open'), 'Labels' (9), 'Milestones' (0), and a 'New pull request' button. Under the filters, there are checkboxes for '1 Open' and '0 Closed', and dropdown menus for Author, Label, Projects, Milestones, Reviews, Assignee, and Sort. A specific pull request is listed: '#1 Mi primera mejora' by albertomontoro, opened 3 minutes ago.

9. Trabajo colaborativo

FORK

- Finalmente, el desarrollador original se encuentra esto:

Mi primera mejora #1

[Edit](#) [Code](#)

[Open](#) albertomontoro wants to merge 1 commit into `amontorosi:main` from `albertomontoro:main`

Conversation 0 Commits 1 Checks 0 Files changed 1 +3 -1

Changes from all commits ▾ File filter ▾ Conversations ▾ Jump to ▾ Review changes ▾

0 / 1 files viewed

prueba_02_TuID.sh

Viewed ...

```
@@ -1,4 +1,6 @@
 1   1 #!/bin/bash
 2   2 clear
 3 - echo "Soy el script 02"
 3 + echo "Soy el script 0"
 4 + echo "1 + 1 = $((1+1))"
 5 + echo "Cerrando el script $0"
 4   6 exit 0
```

9. Trabajo colaborativo

FORK

- Finalmente, el desarrollador original se encuentra esto:

9. Trabajo colaborativo

FORK

- Finalmente, el desarrollador original se encuentra esto:

The screenshot shows a GitHub pull request interface. At the top, there are tabs for Conversation (0), Commits (1), Checks (0), Files changed (1), and a status bar showing +3 -1. Below these are filters for Changes from all commits, File filter, Conversations, Jump to, and a review summary: 0 / 1 files viewed with a link to Review changes.

The main area displays a diff of a file named `prueba_02_TuID.sh`. The diff shows the following changes:

```
@@ -1,4 +1,6 @@
1 1 #!/bin/bash
2 2 clear
3 - echo "Soy el script 02"
3 + echo "Soy el script 0"
4 + echo "1 + 1 = $((1+1))"
5 + echo "Cerrando el script $0"
4 6 exit 0
```

A review comment is open over line 3, reading: "Creo que la línea 3 debería quedarse como estaba, pero el resto me gusta. Gracias por tu aportación." Below the comment is a section for attaching files and three radio button options for responding to the review:

- Comment: Submit general feedback without explicit approval.
- Approve: Submit feedback and approve merging these changes.
- Request changes: Submit feedback that must be addressed before merging.

At the bottom right of the review box is a green "Submit review" button.

9. Trabajo colaborativo

FORK

- Finalmente, el desarrollador original se encuentra esto:

The screenshot shows a GitHub pull request interface for a pull request titled "Mi primera mejora #1". The pull request is from the user "albertomontoro" into the branch "main" from the repository "albertomontoro:si_sh_repositorio_01". The status bar indicates "Changes approved" with 1 approval, and the message "Continuous integration has not been set up". It also states that "This branch has no conflicts with the base branch". A prominent green button at the bottom left says "Merge pull request". Below the main content, there is a comment section with a "Leave a comment" input field and a "Comment" button.

Mi primera mejora #1

albertomontoro wants to merge 1 commit into [amontorosi:main](#) from [albertomontoro:main](#)

Add more commits by pushing to the **main** branch on [albertomontoro/si_sh_repositorio_01](#).

Changes approved
1 approving review [Learn more](#).

1 approval

Continuous integration has not been set up
[GitHub Actions](#) and [several other apps](#) can be used to automatically catch bugs and enforce style.

This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request

You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

Leave a comment

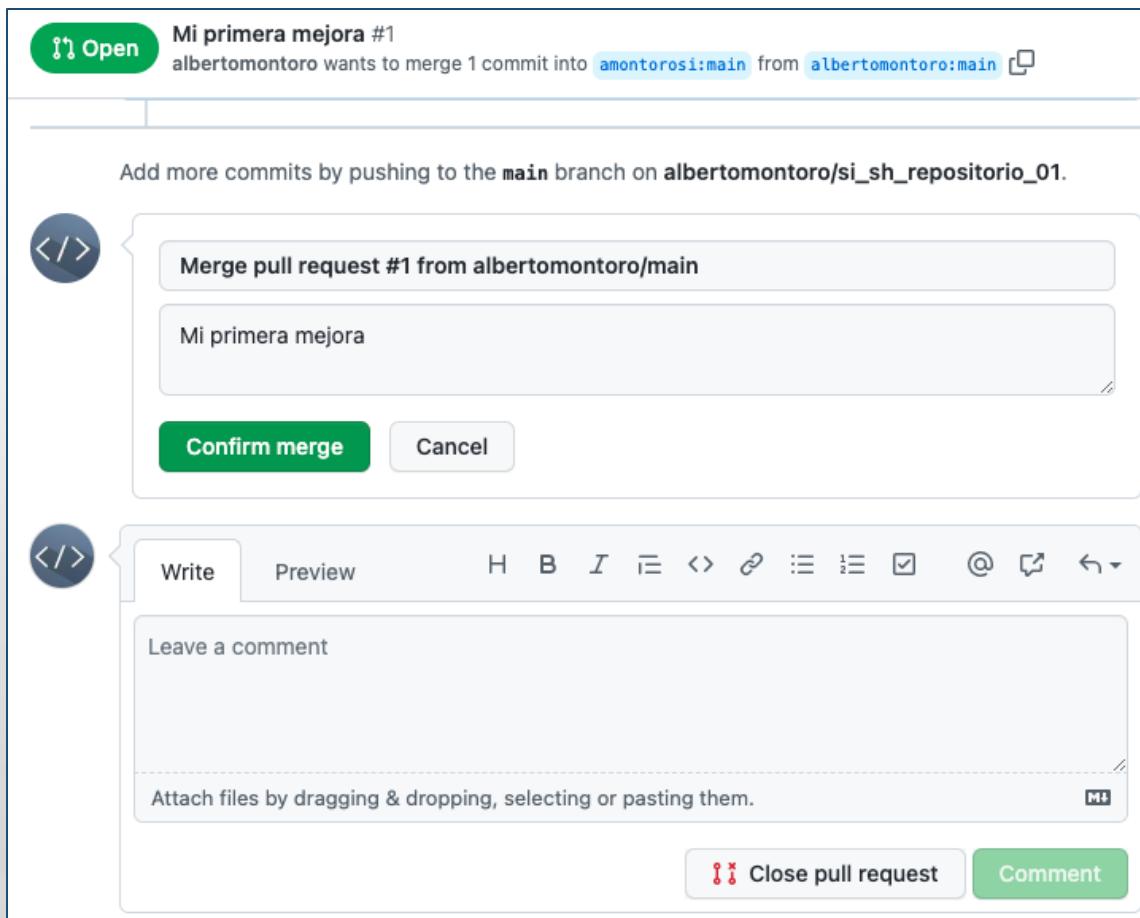
Attach files by dragging & dropping, selecting or pasting them.

[Close pull request](#) [Comment](#)

9. Trabajo colaborativo

FORK

- Finalmente, el desarrollador original se encuentra esto:



9. Trabajo colaborativo

FORK

■ Tarea 12:

- Formar parejas y usando la terminal y GitHub realizar los pasos de este capítulo relativos al Fork sobre el repositorio si_sh_repositorio_01 de uno de los miembros de la pareja.
- Pasos a seguir:
 - Crear un fork
 - Realizar cambios en el código (crear/modificar archivos).
 - Guardar y subir los cambios a su fork en GitHub.
 - Desde GitHub, abrir un Pull Request (PR)
 - Revisra el PR y aceptarlo.
- Extra: ¿Qué pasa si el repo original cambia? Tu fork no se actualiza automáticamente. Investiga cómo sincronizar manualmente un fork con el original.
 - Para ello, hay que decirle a Git dónde está el repositorio original. Para eso, debemos agregar un "remoto" llamado upstream. Después nos traemos los cambios sin aplicarlos (fetch) y finalmente hacemos el merge para fusionar.

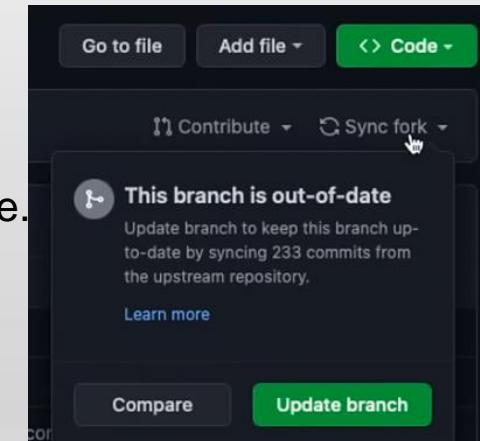
9. Trabajo colaborativo

FORK

- Cuando haces un fork de un repositorio en GitHub, creas una copia independiente del proyecto original en tu cuenta. Sin embargo, **si el repositorio original sigue evolucionando con nuevos commits, tu fork no se actualiza automáticamente**. Si has estado avanzando por tu cuenta en tu fork, podrías querer traer los cambios del repositorio original sin perder tu trabajo.

Esto se puede hacer desde GitHub o desde terminal. La forma más sencilla es ir a la página de tu fork en GitHub:

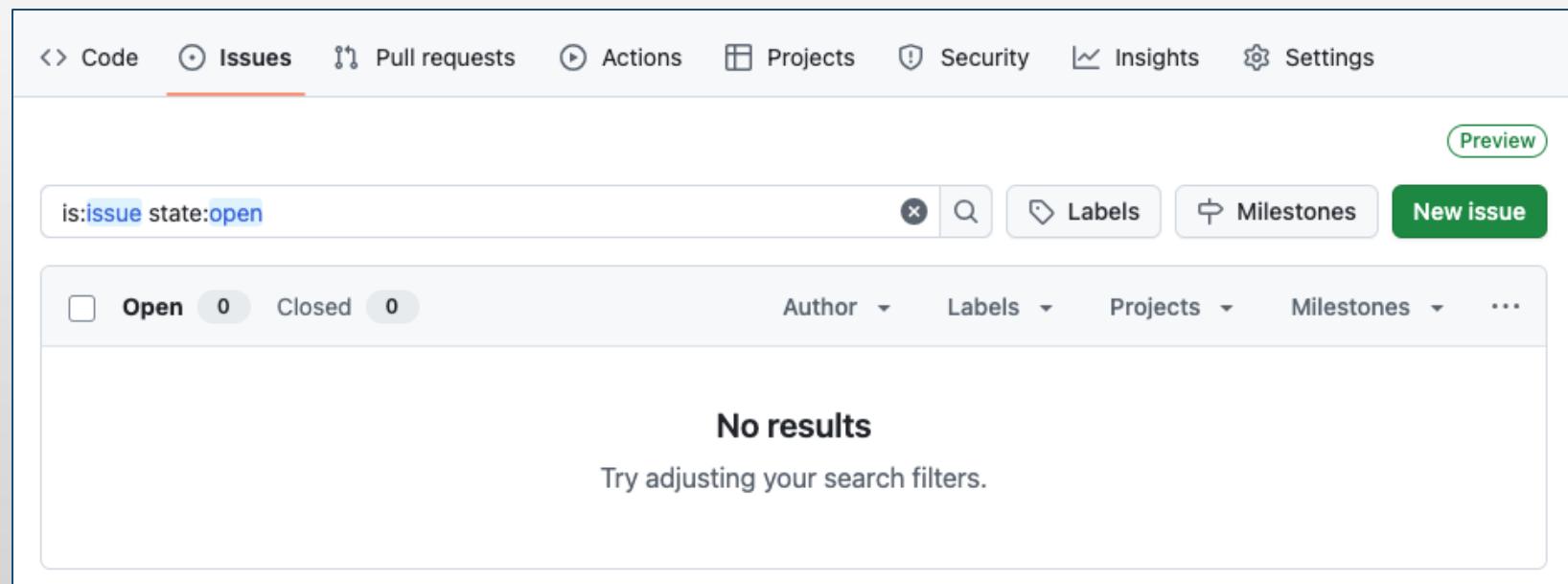
- Haz clic en "Sync fork" (puede estar en la pestaña "Pull requests").
- GitHub te dará la opción de "Update branch".
- Si no hay conflictos, GitHub fusionará los cambios automáticamente.
- Si hay conflictos, GitHub te pedirá que los resuelvas manualmente.



9. Trabajo colaborativo

ISSUES

- Los Issues en GitHub Son una herramienta para reportar errores, solicitar mejoras o discutir ideas dentro de un repositorio. Funcionan como un sistema de seguimiento de tareas en el proyecto. Pueden **ser utilizados tanto por los colaboradores del proyecto como por otros usuarios.**



9. Trabajo colaborativo

ISSUES

- ¿Cómo funcionan?

- 1. Crear un Issue:** Describe un problema o una mejora en el repositorio.
- 2. Asignar etiquetas (labels):** Para categorizar el Issue (bug, mejora, documentación...).
- 3. Mencionar a colaboradores:** Se puede etiquetar a personas con @usuario para que participen.
- 4. Comentar y discutir:** Se pueden agregar comentarios, adjuntar imágenes o referenciar código.
- 5. Cerrar el Issue:** Cuando el problema se resuelve o la mejora se implementa.

The screenshot shows a 'Create new issue' form with the following fields and options:

- Title:** A text input field with placeholder text 'Title'.
- Add a description:** A rich text editor with 'Write' and 'Preview' tabs, and a toolbar with bold (B), italic (I), and other text styling icons.
- Assignees:** A section showing 'No one - Assign yourself' with a link and a settings icon.
- Labels:** A section showing 'No labels' with a settings icon.
- Type:** A section showing 'No type' with a settings icon.
- Projects:** A section showing 'No projects' with a settings icon.
- Milestone:** A section showing 'No milestone' with a settings icon.

Índice de contenidos

1. Introducción
2. Conceptos
3. Flujo de trabajo
4. Instalación de Git
5. Primeros pasos
6. Uso básico de Git
7. Uso básico de GitHub
8. Ramas en Git y GitHub
9. Trabajo colaborativo
- 10. Algunos comandos útiles**

10. Algunos comandos útiles: 1.

- El comando git fetch descarga commits y archivos de un repositorio remoto a tu repositorio local, pero los posibles cambios que pueda haber entre el repositorio remoto y el local **no se integran** en tu repositorio local. NO los fusiona con tu código local.

```
git fetch
```

- Git aísla esos cambios y nos permite revisarlos antes de fusionarlos con nuestro repositorio local. **Solo comprueba si hay algún cambio disponible:**

```
git log origin/main --oneline
```

- Cuando realizamos un git pull nos traemos todos los cambios y los fusionamos con nuestro repositorio local directamente. Automáticamente realiza el git fetch y el git merge:

```
git pull origin main
```

10. Algunos comandos útiles: 2.

- Las etiquetas nos permiten definir hitos de versiones finales y estables de nuestro proyecto. Si en el futuro necesitas revisar o corregir esa versión, puedes acceder a ella sin problemas con: `git checkout tag_creado`
- Para crear una etiqueta solo debes escribir:

```
git tag v1 -m "Versión 1"
```

```
git push origin v1
```

The image shows a screenshot of a GitHub repository interface. On the left, there's a summary bar with the main branch (main), 2 branches, and 1 tag. The main content area shows a 'Tags' tab selected under 'Releases'. A single tag named 'v1' is listed, with a timestamp of '25 minutes ago', a commit hash 'a8f3de4', download links for 'zip' and 'tar.gz', and a 'Notes' link.

Releases Tags

Tags

v1 ...

25 minutes ago · a8f3de4 · zip · tar.gz · Notes

10. Algunos comandos útiles: 3.

- Imagina que acabamos de hacer un commit y nos damos cuenta que un fichero que queríamos incluir en ese commit no lo hemos modificado.

En ese caso podemos modificar dicho fichero, añadirlo al staging area y en lugar de hacer un nuevo commit podemos hacer:

```
git commit --amend
```

Esto nos abre un editor que nos permite modificar el último commit. Nos permite incluir ese fichero en ese último commit y nos permite también modificar el mensaje del último commit.

Al guardar los cambios se sobrescribe el commit anterior y se genera un nuevo hash, pero no se crea un commit adicional.

10. Algunos comandos útiles: 3.

- Si hiciste un commit por error pero quieres mantener los archivos modificados, usa lo siguiente. Revierte el commit, pero mantiene los cambios listos para corregirlos.

```
git reset HEAD~1
```

10. Algunos comandos útiles: 4.

- El comando checkout permite cambiar entre diferentes versiones de una entidad: archivos, commits y ramas. Ya hemos visto su uso en ramas.

También podemos usarlo también para volver los ficheros al estado del último commit descartando todos los cambios que hayamos realizado hasta ese momento.

Imagina que después del último commit modificamos el fichero 'prueba_01_TuID.sh', probamos esos cambios y no nos gusta el resultado. Para restaurar ese fichero al último commit basta con hacer:

```
git checkout prueba_01_TuID.sh
```

Si hemos editado más de un fichero y queremos que todos vuelvan a ese estado previo:

```
git checkout -- .
```

10. Algunos comandos útiles: 5.

- Otra forma de descartar cambios en ficheros que se han modificado y se encuentra en el working directory sería:

```
git restore nombre_fichero
```

- Del mismo modo, para descartar cambios en ficheros que se han modificado y se han pasado al staging area sería:

```
git restore --staged nombre_fichero
```

10. Algunos comandos útiles: 6.

- Existe un comando que permite tomar un commit específico de una rama y aplicarlo a otra, sin necesidad de fusionar toda la rama.

```
git cherry-pick hash_commit
```

- Se usa cuando:
 - Hiciste un cambio en la rama equivocada y quieres mover solo ese commit a la correcta.
 - Necesitas traer un commit específico de una rama sin traer todos sus cambios.
 - Quieres aplicar un hotfix (corrección rápida) desde una rama de desarrollo a producción.

10. Algunos comandos útiles: 6.

- Vemos el ejemplo cuando hacemos un commit en la rama equivocada.
- Para **copiar** ese commit a la rama que queremos, necesitamos conocer el hash completo del commit y hacer lo siguiente:

```
git checkout rama_destino_commit  
git cherry-pick hash_id_commit  
git log
```

- Para **mover** ese commit a la rama que queremos, haríamos exactamente lo mismo y luego nos moveríamos a la rama inicial y haríamos un reset:

```
git checkout rama_error_commit  
git reset --hard hash_id_commit
```

10. Algunos comandos útiles: 7.

- En esta línea podría elegir dos commit y ver sus diferencias:

```
git diff hash_id_commit1 hash_id_commit2
```

- Recuerda. Comando para conocer el histórico de todos los commit, incluidos los reset y los amend:

```
git reflog
```

10. Algunos comandos útiles: 8.

- Si hacemos un reset a un commit en local que ya habíamos publicado en GitHub, cuando intentemos sincronizar con GitHub vamos a tener un conflicto porque el último commit de GitHub va 'delante' del commit actual que tenemos.

- En ese caso podemos forzar a que se descarte el commit de GitHub y que el nuestro actual en local se convierta en el principal

```
git push origin -f
```

- La última propuesta seguramente elimine ficheros. Si estamos seguro lo podemos hacer sin ningún tipo de problemas. En cambio, si no lo estamos, en ese caso puede ser recomendable hacer un `git revert`.

10. Algunos comandos útiles: 9.

- Podemos crear alias y usarlos con git

```
git config --global alias.mylog "log --graph --oneline --all --decorate"  
git mylog
```

10. Algunos comandos útiles: 10.

- Si usas git stash, puedes añadir un mensaje para recordar qué guardaste:

```
git stash push -m "Guardando cambios antes de cambiar de rama"
```

- Existe también un comando que te permite ver quién modificó cada línea de un archivo y en qué commit se hizo el cambio. Es muy útil para rastrear cambios en el código y entender quién escribió qué y cuándo lo hizo.

```
git blame nombre_fichero
```

Anexos

A. Markdown

- Markdown es un lenguaje de marcas ligero muy usado en la elaboración de la documentación web que nos permite con marcas simples en nuestro texto generar un documento web válido.
- Fue creado por John Gruber y Aaron Swartz.
- Se distribuye con licencia BSD.

A. Markdown

- En el siguiente enlace puedes encontrar las marcas básicas que puedes emplear en Markdown: <https://www.markdownguide.org/cheat-sheet/>

Basic Syntax

These are the elements outlined in John Gruber's original design document. All Markdown applications support these elements.

Element	Markdown Syntax
Heading	# H1 ## H2 ### H3
Bold	bold text
Italic	<i>italicized text</i>
Blockquote	>blockquote

Imagen extraída de: <https://www.markdownguide.org/cheat-sheet/>

A. Markdown

- Y en el siguiente enlace puedes descubrir cómo usar este lenguaje de marcado directamente en Visual Studio Code:

<https://code.visualstudio.com/docs/languages/markdown>

B. Configuración en Windows para evitar problemas con Docker y scripts

- Si tu sistema anfitrión es Windows y en el repositorio se tienen ficheros manejados por herramientas que sí necesitan LF en Windows (como scripts en bash para WSL o un entorno Linux Dockerizado), entonces usar este comando puede traer problemas:

```
git config --global core.autocrlf true ← ¿Problemas?
```

- En Windows, algunos archivos NO deben convertirse a CRLF, porque pueden dejar de funcionar. Para evitarlo, es necesario forzar LF en algunos archivos específicos usando el fichero .gitattributes. Para ello,
 - Paso 1. En el repositorio del proyecto, creamos un archivo llamado .gitattribute
 - Paso 2. Hacemos un commit con este archivo para que todos los sistemas respeten estas reglas.
 - Paso 2.1 si los ficheros ya existían antes del paso 1 y el paso 2 hay que renormalizar y hacer un push:

```
git add --renormalize .
```

B. Configuración en Windows para evitar problemas con Docker y scripts

- Paso 1. Fichero .gitattributes

```
# Asegurar que los scripts `*.sh` siempre usen LF
*.sh text eol=lf

# Mantener archivos de Docker en LF
Dockerfile text eol=lf
*.dockerrc ignore text eol=lf
*.yml text eol=lf
*.yaml text eol=lf

# Mantener archivos de configuración de Unix en LF
*.conf text eol=lf
*.service text eol=lf

# Permitir que los archivos de Windows usen CRLF si es necesario
*.bat text eol=crlf
*.cmd text eol=crlf
```

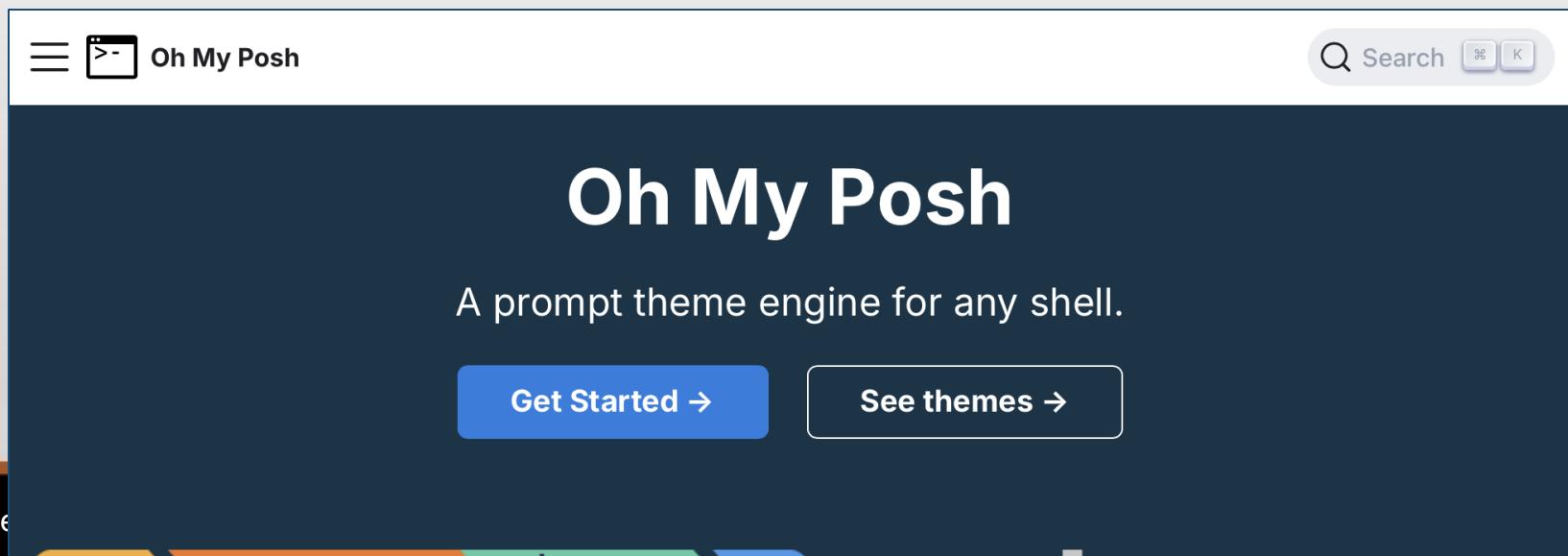
B. Configuración en Windows para evitar problemas con Docker y scripts

- Paso 2. Añadimos al repositorio

```
git add .gitattributes  
git commit -m "Añadir configuración de EOL en .gitattributes"  
git push
```

C. Oh My Posh para Windows

- Para PowerShell, existe Oh My Posh, una herramienta que permite personalizar el prompt de PowerShell con temas y funcionalidades avanzadas.
- Los detalles y opciones de configuración están disponibles en el sitio oficial: <https://ohmyposh.dev> y <https://ohmyposh.dev/docs/installation/windows>
- Aquí tenéis un pequeño manual: <https://platzi.com/tutoriales/2042-prework-windows/9265-personaliza-tu-powershell-con-oh-my-posh/>



C. Oh My Posh para Windows

- Tanto Oh My Posh como Oh My Bash ofrecen una variedad de temas que utilizan iconos especiales. Para que estos iconos se muestren correctamente, es necesario instalar fuentes específicas en vuestro sistema. Podéis descargar e instalar las Nerd Fonts desde: <https://www.nerdfonts.com/>

C1. Oh My Bash en Windows con Git Bash

- Si utilizáis Git Bash en Windows, podéis instalar **Oh My Bash**.
 - Sin embargo, Git Bash en Windows no soporta todas las características de Bash en Linux, por lo que algunas funciones pueden no funcionar correctamente.
- Podéis encontrar más información en su repositorio oficial:
<https://github.com/ohmybash/oh-my-bash>
- También puedes instalar Zsh en Git Bash y usar Oh My Zsh. Es más rebuscado, pero aquí tenéis un enlace: <https://renan-costaalencar.medium.com/zsh-and-oh-my-zsh-on-windows-via-git-bash-a2f21b85b51c>

Setting up Zsh and oh-my-zsh on Windows via Git Bash

```
~ mkdir zshhh
~ cd zshhh
~/zshhh git init
Initialized empty Git repository in /home/michiel/zshhh/.git/
~/zshhh master touch zshhh.txt
~/zshhh master gaa
~/zshhh master + gcam "first commit"
```