



Repositorios digitales: Tecnologías Git y GitHub

Alfredo Abad

ISO-04-071-git-github.pptx

5-oct-2023

1
Alfredo Abad



Repositorios digitales

Un repositorio almacena datos que pueden recuperarse y modificarse posteriormente.

Existen distintos tipos de repositorios, que pueden utilizarse para el control de versiones, metadatos y otros fines.

Un tipo de repositorio digital es GIT

2
Alfredo Abad



¿Qué es un repositorio?

- “Repositorio” significa “almacenamiento” y procede de la palabra latina *repositorium*.
 - En tecnología de software, un repositorio es un archivo digital en el que se pueden **almacenar y compartir datos, documentos, avances de desarrollo, metadatos y programas**.
 - El control de versiones es otra característica de los repositorios.
 - Dependiendo del uso previsto, esta tecnología permite a grandes equipos o comunidades que trabajan en todo el mundo colaborar en proyectos compartidos.
 - Los tipos de repositorios disponibles difieren en cuanto a su enfoque y estructura.
 - Los repositorios más conocidos son [GitHub](#) y Google Repository.
- Normalmente, la base de un repositorio es una base de datos, que, según los requisitos, puede instalarse en un disco duro local o en un servidor, o también puede distribuirse entre numerosos servidores de una [CDN](#) (red de distribución de contenidos).
 - Se crean **catálogos de datos** que contienen las formas y representaciones de varios objetos almacenados y proporcionan información sobre su relación entre sí.
 - Toda esta información se almacena en forma de metadatos y se puede buscar, recuperar, modificar y adaptar en cualquier momento con los permisos adecuados.

3
Alfredo Abad



¿Cómo se estructura un repositorio?

- Para ilustrar cómo se estructura un repositorio, puedes visualizar un árbol. En el desarrollo de software, puedes verlo reflejado incluso en la terminología.
- Aquí se distingue entre el **tronco**, que contiene la versión actual de un proyecto y el código fuente, y las **ramas**, donde se almacenan las modificaciones.
- Los cambios se vuelven a añadir posteriormente al tronco para que todos los participantes tengan acceso a ellos.
- El almacenamiento se realiza mediante etiquetas.

4
Alfredo Abad



Tipos de repositorios (I): Repositorio de gestión de versiones

- En el sistema de gestión de versiones, el objetivo es **almacenar los datos de forma clara, mientras se elaboran lógicamente los pasos y las relaciones en un archivo común**.
 - Los archivos de código fuente y otros datos se almacenan y archivan.
 - Los datos pueden copiarse del repositorio a un disco duro local para que los desarrolladores sigan trabajando con ellos; proceso que se conoce como "checking out".
 - A continuación, el desarrollador trabaja con los datos locales, realizando cambios o descartando los cambios anteriores.
 - Una vez completado el trabajo, el último estado del proyecto se carga de nuevo en el repositorio, lo que se denomina "checking in".
 - Todos los cambios y comentarios se registran durante este proceso.
- Este enfoque tiene varias ventajas.
 - Por un lado, los usuarios pueden colaborar en un proyecto sin sobrescribir versiones anteriores.
 - En cambio, todas las actualizaciones de estado se registran, lo que permite regresar a una versión anterior.
 - Un repositorio **permite a equipos pequeños y grandes colaborar en el mismo proyecto**.
 - Las actualizaciones pueden hacerse simultáneamente sin sobrescribir estados ni perder cambios.
 - Teóricamente, todos los usuarios pueden continuar un proyecto en cualquier estado sin ningún riesgo.
- Los sistemas de control de versiones más populares son CVS, GitHub y SVN.

5
Alfredo Abad



Tipos de repositorios (II): Repositorio de gestión de versiones

- Un repositorio de metadatos suele utilizarse en infraestructuras informáticas muy complejas.
- Un repositorio de este tipo contiene **los datos de todo el sistema, así como información sobre el contexto y el entorno de la infraestructura**.
- La ventaja de este tipo de repositorio es que se pueden hacer cambios sin alterar el código fuente ni tener que implementar programas adicionales.
- En su lugar, la tabla de la base de datos, que es la base del sistema correspondiente, se adapta de forma sencilla.
- El repositorio de metadatos suele utilizarse en la integración de aplicaciones empresariales (UAI) y el almacenamiento de datos.

6
Alfredo Abad



Tipos de repositorios (III): Repositorio de software

- Un repositorio de software es especialmente importante para los usuarios de Linux.
- Un repositorio de software contiene **paquetes de aplicaciones y los metadatos correspondientes**, como explicaciones, anotaciones, dependencias y cambios.
- La instalación y las actualizaciones se realizan mediante un gestor de paquetes.
 - De este modo, los usuarios no tienen que preocuparse de actualizar sus aplicaciones.
- Normalmente es la comunidad la que proporciona las actualizaciones y el sistema, el que se actualiza automáticamente.
- Los usuarios que mantienen los paquetes, conocidos como mantenedores de paquetes, suelen proporcionar los datos actualizados y llevar a cabo el mantenimiento del respectivo repositorio de software.

7
Alfredo Abad



Tipos de repositorios (IV): Repositorio de servidores de documentos

- El término repositorio también se aplica a las publicaciones extensas en red y a los servidores de documentos, al menos en sentido figurado.
- Aunque **algunas características especiales del principio de repositorio no se aplican una a una**, el procedimiento sí se ha adaptado para su uso.
- Servidores de documentos muy conocidos, como arXiv, publican artículos de biología, informática, matemáticas, física y estadística.
- Un experto revisa los nuevos artículos y los aprueba o rechaza.
- A continuación, los trabajos científicos pueden estar disponibles para su descarga.
 - Sin embargo, a diferencia de un repositorio de control de versiones, no es posible editar los documentos.

8
Alfredo Abad



Tipos de repositorios (y V): Repositorio CASE

- Un repositorio también se utiliza con frecuencia en la **ingeniería de software asistida por ordenador (CASE)**.
- Se utiliza principalmente para almacenar datos del proyecto, documentación y código fuente.

9

Alfredo Abad



¿Qué repositorios son más útiles?

- Existen numerosos tipos de repositorios para **distintos fines**.
 - Hay que distinguir entre las soluciones que son de código abierto y las que se ofrecen con fines comerciales.
 - El repositorio de código abierto más popular es GitHub. Sin embargo, existen varias [alternativas a GitHub](#) como Apache Allura, Bazaar, Gitolite, Mercurial o SourceForge.
 - En nuestra Digital Guide encontrarás una [comparación entre GitHub y GitLab](#) detallada.
 - Entre los repositorios de software de pago más conocidos están Alienbrain, Bitkeeper, IBM Rational Synergy y MySQL Yum.
- Que un repositorio sea más o menos adecuado para tu proyecto **depende de tus necesidades y de tu forma de trabajar**.
 - Para el trabajo en equipo, un repositorio puede mejorar los procesos y optimizar el flujo de trabajo.
 - Aunque los colaboradores accedan a un proyecto y realicen cambios en diferentes momentos y desde distintos lugares, el tronco siempre está a salvo.
 - Se pueden probar soluciones sin poner en peligro los avances anteriores.
 - Siempre es una buena idea probar una solución de código abierto antes de comprar una versión comercial.

10

Alfredo Abad



¿Cómo funciona un repositorio?

- Utilizado correctamente, un repositorio ofrece varias ventajas. GitHub es un gran ejemplo de ello.
- Una vez que hayas instalado y configurado GitHub, puedes utilizar la intuitiva interfaz de usuario para asignar y procesar tareas.
- Los cambios listados se realizan mediante los comandos commit y pull.
- De este modo, un responsable de equipo puede hacer un seguimiento de los pasos de progreso individuales y los miembros pueden seguir el proyecto hasta el más mínimo detalle.
- Para saber más sobre GitHub, echa un vistazo a nuestro [tutorial de Git](#).

11
Alfredo Abad



Sistemas de control de versiones

12
Alfredo Abad



Sistemas de control de versiones

- Un sistema de control de versiones es un conjunto de herramientas que nos permite registrar los cambios que se producen en los archivos de un proyecto a lo largo del tiempo, y que por tanto nos permite volver a cualquier estado del mismo a nuestra voluntad
- Podemos encontrar distintos tipos de sistemas de control de versiones:
 - **Sistemas locales**, que sólo existen en nuestros propios equipos de trabajo.
 - Un ejemplo de este tipo sería por ejemplo RCS
 - **Sistemas centralizados**, que dependen de un servidor centralizado que almacena toda la información, encargándose de distribuir distintas copias entre los distintos usuarios y guardar todos los cambios que se van produciendo.
 - Algunos ejemplos de este tipo serían CVS, Subversion, o Perforce
 - **Sistemas de control distribuido**, en los que cada participante en el proyecto tiene una copia completa del mismo sin depender de un servidor que almacene toda la información.
 - Algunos ejemplos serían GIT o Mercurial

13
Alfredo Abad



¿Qué es Git y GitHub?

- Son tecnologías que se encargan de los procesos de gestión de versiones de software
- GitHub, además, añade la publicación del software en la sede de GitHub con dos posibilidades:
 - Gratuito: solo para publicar software libre
 - De pago
- Son herramientas bastante sofisticadas y ampliamente utilizadas

14
Alfredo Abad





Fundamentos básicos de git

15
Alfredo Abad



¿Por qué GIT?

- GIT es un sistema de control de versiones que surge tras la necesidad de dar soporte a la gran cantidad de cambios que se producían en el Kernel de Linux y tras las diversas discrepancias que se produjeron allá por el año 2005 con BitKeeper, herramienta que se usaba por aquel entonces para mantener todo el código
 - Esta situación impulsó a la comunidad de desarrolladores de Linux y en especial a Linus Torvalds a desarrollar su propio sistema de control de versiones. De esa iniciativa surgió GIT
- En la actualidad GIT es una de las mejores opciones que existen por las siguientes razones:
 - Tiene un diseño sencillo y prácticamente transparente para sus usuarios
 - Es veloz y sobre todo eficiente
 - Cuenta con un increíble sistema de ramificaciones convirtiéndolo en una fabulosa herramienta para el desarrollo no lineal de proyectos
 - Es completamente distribuido y por tanto capaz de manejar grandes proyectos

16
Alfredo Abad



Instalación del software

- Se requiere una copia de Git instalada en el sistema que se vaya a utilizar
 - Existen versiones para distintos sistemas operativos: Windows, Linux, etc.
 - El software de instalación se puede descargar a través de los repositorios de las distros o desde la web de Git
- Git se utiliza a través de la línea de comandos, aunque también hay disponibles interfaces gráficas
- Para nuestras pruebas confeccionaremos un repositorio sencillo en el que incluiremos un fichero con código fuente y un README
 - Después se estudiarán y practicarán los comandos típicos de git: init, clone, add, commit, diff, log, etc.

17
Alfredo Abad



Los tres estados de GIT

- Cuando trabajamos con GIT, nuestra información puede estar en una de las siguientes situaciones:
 - **Información modificada (*modified*)**
 - Este estado implica que hemos modificado nuestra información, pero aún no está siendo trackeada por GIT
 - **Información preparada (*staged*)**
 - Este estado implica que hemos marcado nuestra información para posteriormente ser confirmada y por tanto trackeada como nueva versión
 - **Información confirmada (*committed*)**
 - Este estado implica que nuestra información ha sido almacenada en la base de datos local de GIT

18
Alfredo Abad



Inicio de un repositorio Git

- Preparación:
 - Crear un directorio de trabajo sobre el que crear el repositorio (en nuestro caso: video-lister)
 - Nos posicionamos en ese directorio por defecto
- Creación e inicio del repositorio: **git init**
 - Esto crea un subdirectorio oculto (.git), que es donde Git almacenará sus bases de datos y configuración
 - También crea una rama del proyecto por defecto que es la rama MASTER

```
video-lister — mattsetter@Matts-Mac: ~/Workspace/PHP/video-lister — zsh
../video-lister
→ video-lister git init .
Initialized empty Git repository in /Users/mattsetter/Workspace/PHP/video-lister/.git/
→ video-lister git:(master)
```

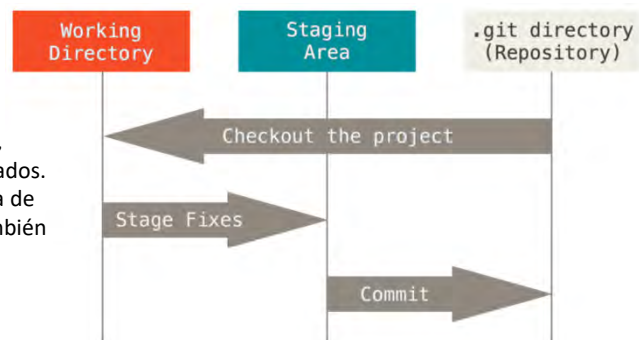
19
Alfredo Abad



Relaciones entre componentes: los tres estados de GIT

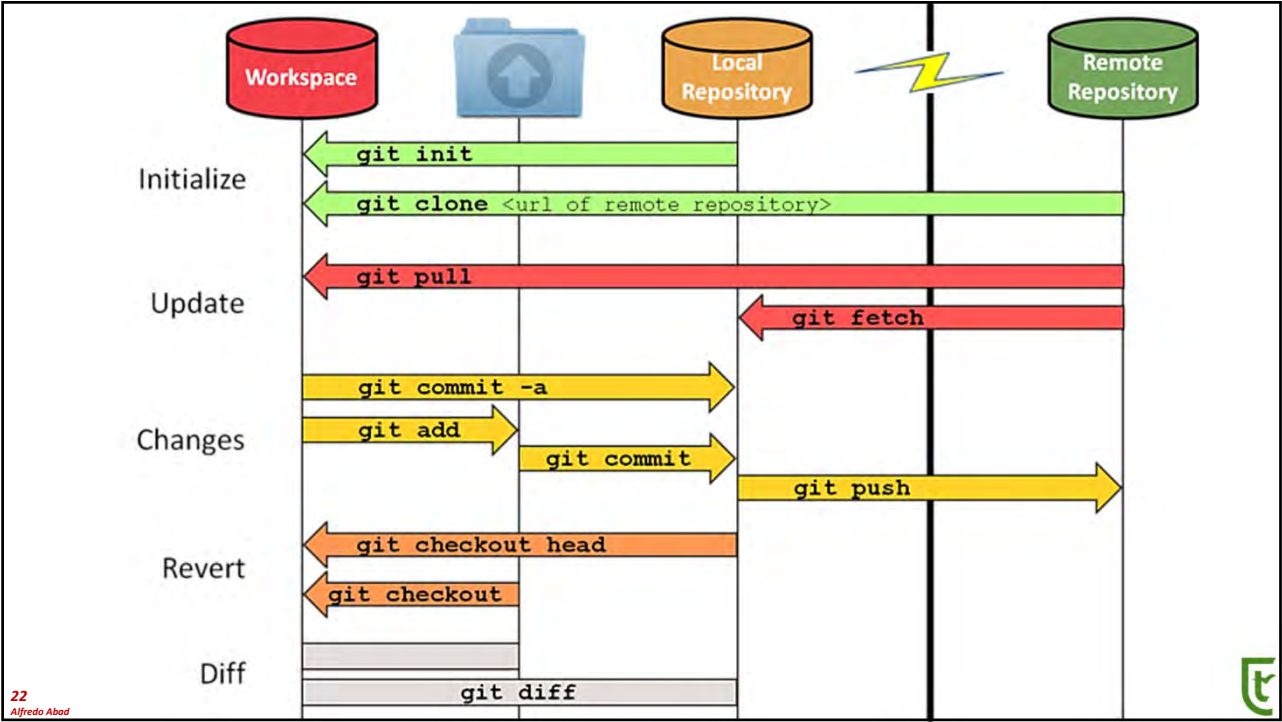
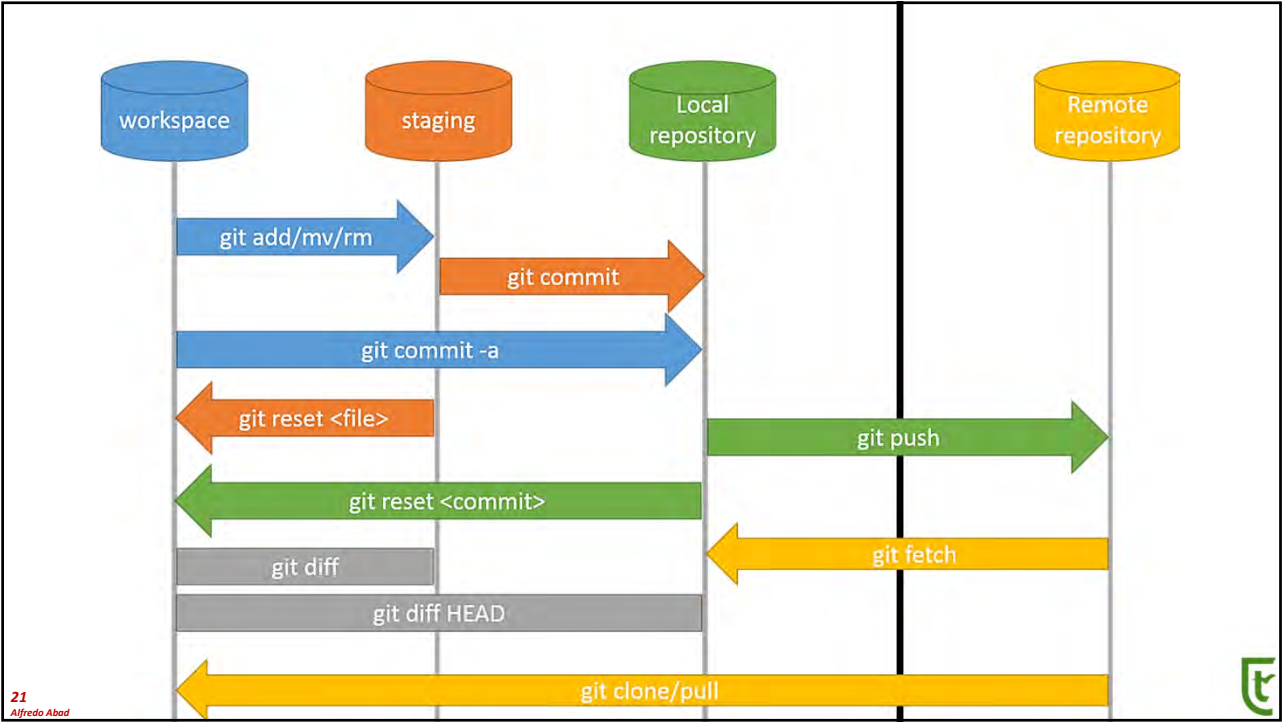
- Estas situaciones dan lugar a lo que se conoce como los tres estados de GIT:
 - El **working directory**, que es nuestra área de trabajo en la que hacemos los cambios en nuestros ficheros
 - El **staging area**, que es el espacio donde colocaremos aquellos ficheros listos para ser colocados en el repositorio
 - Y el **repositorio**, que es el área donde GIT irá guardando las distintas versiones de nuestra información

Todo nuestro trabajo con GIT se reduce a ir moviendo la información de un área a otra, gestionando de esa forma sus distintos estados. Para ello usaremos preferiblemente la línea de comandos de nuestro terminal aunque también existen utilidades gráficas que nos facilitan nuestro trabajo diario con GIT.



20
Alfredo Abad





Configurando GIT

- En primer lugar nos aseguraremos que lo tenemos instalado en nuestro equipo
 - Para verificar si está o no instalado, podemos ejecutar este comando **git version**
 - Si obtenemos respuesta nos indicará la versión de GIT que tenemos instalada, y en caso de que no lo esté, tendrás que instalarlo en tu equipo
 - Aquí hay una guía de instalación:
 - <https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Instalaci%C3%B3n-de-Git>

23
Alfredo Abad



Configuraciones básicas iniciales

- Todas las operaciones en GIT van firmadas con la información básica del usuario: **user.name** y **user.email**
 - Para configurar dicha información en GIT ejecutaremos los siguientes comandos en nuestro terminal:
 - `git config --global user.name "name"`
 - `git config --global user.email "email"`
- El modificador **--global** indica a GIT que esta configuración se usará por defecto siempre que trabajemos con GIT
 - Estos parámetros de configuración podemos personalizarlos a nivel de proyecto tan sólo quitando el modificador **--global** del comando
- Si queremos conocer cuál es nuestra configuración actual podemos ejecutar los siguientes comandos:
 - `git config user.name`, o bien `git config user.email`

24
Alfredo Abad



Otras configuraciones iniciales

- Vamos a configurar un par de parámetros más de forma global:
 - **git config --global color.ui true**
 - Indicamos a GIT que use color para mostrarnos distintos tipos de información
 - **git config --global core.editor "nano"**
 - Configura el editor de texto por defecto que usará GIT cuando necesitemos por ejemplo, anotar información acerca de los cambios que se van a confirmar
- Podríamos hacer más configuraciones, pero en principio para comenzar tenemos suficiente
- Si queremos ver nuestra configuración global, ejecutaremos el comando:
 - **git config --list**



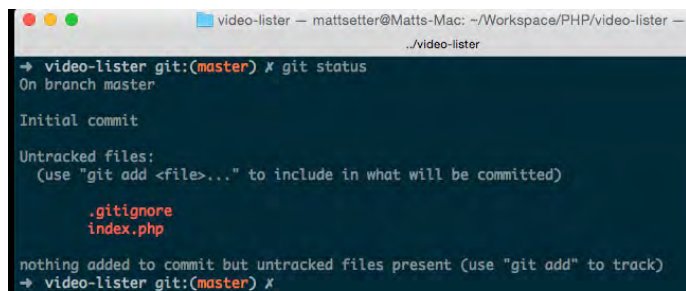
Clonar un repositorio

- Hay un segundo modo de acceder a un repositorio que es mediante clonación de otro previamente existente
- Se clona con **git clone <repository URL (remoto)>**
 - Que creará una copia exacta del repositorio indicado (remoto) en el repositorio local
 - Después de esto se puede trabajar con la copia local y luego reintegrar los cambios de nuevo al remoto



Añadir un nuevo fichero a un repositorio

- Crearemos un fichero local en código fuente (**index.php**, en nuestro caso) en el directorio raíz de nuestro repositorio local
 - El código fuente (no significativo) es:
<?php print "Hello World">;
- Después de salvar el código del fichero ejecutaremos **git status**
 - Que mostrará el estado del repositorio local de trabajo



```
video-listener — mattsetter@Matts-Mac: ~/Workspace/PHP/video-listener —  
./video-listener  
→ video-listener git:(master) # git status  
On branch master  
  
Initial commit  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
  
    .gitignore  
    index.php  
  
nothing added to commit but untracked files present (use "git add" to track)  
→ video-listener git:(master) #
```

27
Alfredo Abad



Seguimos trabajando sobre el repositorio de trabajo

- Podemos añadir u operar con más ficheros como lo haríamos normalmente en cualquier directorio
 - Añadiremos un nuevo fichero **README.md**
 - (Todo nuevo buen proyecto debería tener un README.md con su descripción)
- Ahora ejecutamos de nuevo **git status** para evaluar el estado del repositorio (diapo siguiente)

28
Alfredo Abad



Estatus de los dos ficheros (untrack)

```
Git-GitHub — mattsetter@Matts-Mac: ~/Workspace/Clients/Contently/Git-GitHub —  
../Git-GitHub  
→ Git-GitHub git:(master) x git status  
On branch master  
  
Initial commit  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
  
    README.md  
    index.php  
  
nothing added to commit but untracked files present (use "git add" to track)  
→ Git-GitHub git:(master) x
```

29
Alfredo Abad



Gestión de índice del fichero index.php

- De momento, nos olvidamos de REAME.md y nos fijamos en index.php
- Ejecutamos **git add index.php**
 - Y después un git status para ver el estado, que habrá cambiado (ver diapo siguiente)
 - index.php habrá sido registrado en el control del repositorio (mientras que README.md seguirá alojado pero sin sometimiento a ningún control)

```
Git-GitHub — mattsetter@Matts-Mac: ~/Workspace/Clients/Contently/Git-GitHub —  
../Git-GitHub  
→ Git-GitHub git:(master) x git status  
On branch master  
  
Initial commit  
  
Changes to be committed:  
  (use "git rm --cached <file>..." to unstage)  
  
    new file:   index.php  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
  
    README.md  
  
→ Git-GitHub git:(master) x
```

30
Alfredo Abad



Actualizando la configuración de Git

- Una vez indexado el fichero index.php, ahora podremos realizar sobre él un **commit** cuyos mensajes se escribirán con un editor por defecto
- Git usa por defecto el editor indicado en las variables **\$VISUAL** o **\$EDITOR**, que normalmente será pico, vi, vim o emacs
- Si queremos especificar un editor concreto (Notepad, TextEdit, Gedit, etc.), lo podemos hacer con:
 - `git config --global core.editor <nombre-de-aplicación>`

31
Alfredo Abad



Realizar el primer commit

- **Commit** es fijar los cambios realizados asociándolos a una versión concreta de la que se gestionará su control
- Se utilizará el comando **git commit**
 - Que abrirá automáticamente el editor seleccionado con una plantilla (diapo siguiente) para rellenar en donde especificaremos los cambios realizados
- El **commit** solo afectará a los ficheros del proyecto que hayan sido indexados previamente (en nuestro caso, index.php; README.md no será afectado)

32
Alfredo Abad



Ejemplo de plantilla de documentación de commit

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   new file:   index.php
#
# Untracked files:
#   README.md
#
```

33
Alfredo Abad



El mensaje de commit

- Un buen mensaje de **commit** consta de dos partes:
 - **Un mensaje corto** (máximo de 72 caracteres) con una breve descripción de los cambios
 - **Un mensaje mucho más largo** (opcional) con una descripción detallada de los cambios
- Git status, después de commit:

```
Git-GitHub — mattsetter@Matts-Mac: ~/Workspace/Clients/Contently/Git-GitHub
..ly/Git-GitHub
→ Git-GitHub git:(master) x git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    README.md

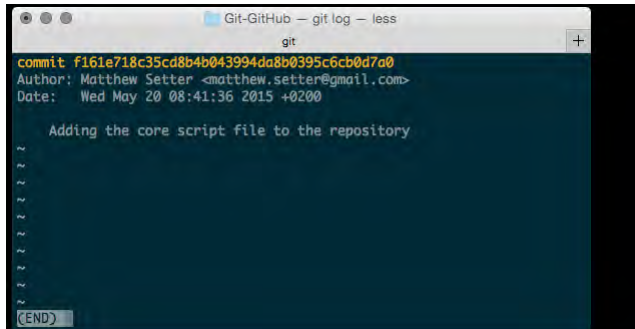
nothing added to commit but untracked files present (use "git add" to track)
→ Git-GitHub git:(master) x
```

34
Alfredo Abad



Examinar diferencias y vista de la historia de cambios

- Se pueden examinar las diferencias (local – remota) de un fichero con **git diff**
 - Más información en <https://git-scm.com/docs/git-diff>
- El histórico de cambios se irá escribiendo automáticamente y se puede consultar con **git log**
 - Más información con **git help log**

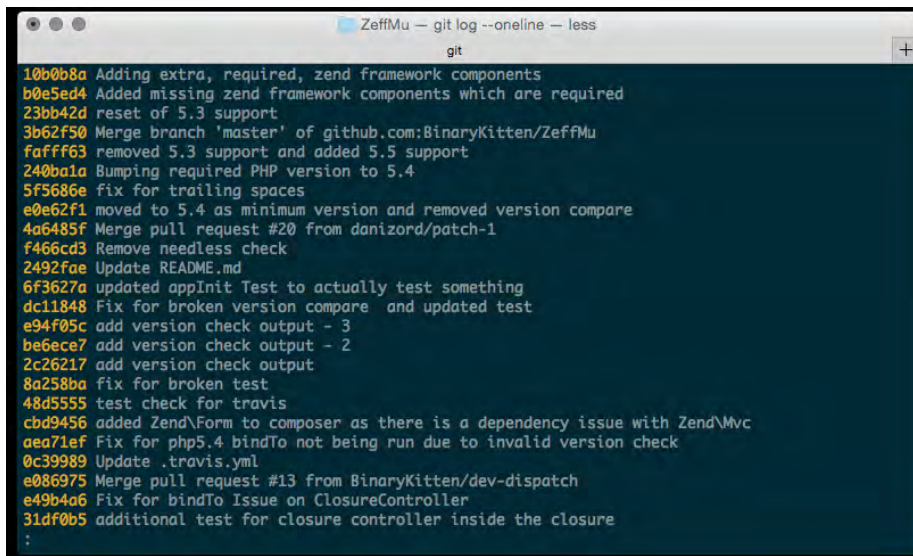


35

Alfredo Abad



Visualizar el hash y el mensaje del comit con **git log --oneline**



36

Alfredo Abad



Las ramas de un proyecto (branching) Mainline trunk (en terminología Git)

- Todo proyecto Git tiene necesariamente una rama **MASTER**, que es la que utiliza por defecto (en todos los anteriores ejemplos)
 - Después se pueden abrir nuevas ramas para hacer derivaciones del proyecto o para hacer pruebas o para dividir el conjunto total
- Para crear una nueva rama (develop) se utiliza el comando **git checkout -b develop**
 - Que en su inicio será una copia del master
- Se puede reintegrar una rama (**develop**) en la rama de que deriva (**master**) con el comando **git merge develop**

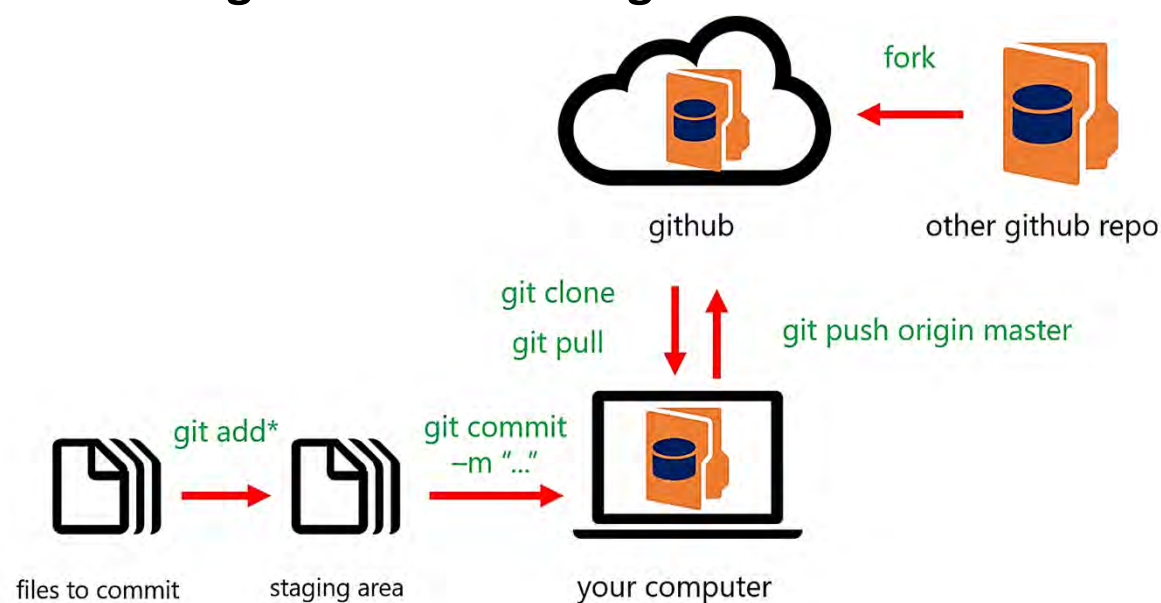
```
Git-GitHub — mattsetter@Matts-Mac: ~/Workspace/Clients/Contently/Git-GitHub — zsh
..ly/Git-GitHub
→ Git-GitHub git:(master) git merge develop
Updating f161e71..b0f2c89
Fast-forward
 README.md | 9 ++++++++
 1 file changed, 9 insertions(+)
 create mode 100644 README.md
→ Git-GitHub git:(master)
```

37

Alfredo Abad



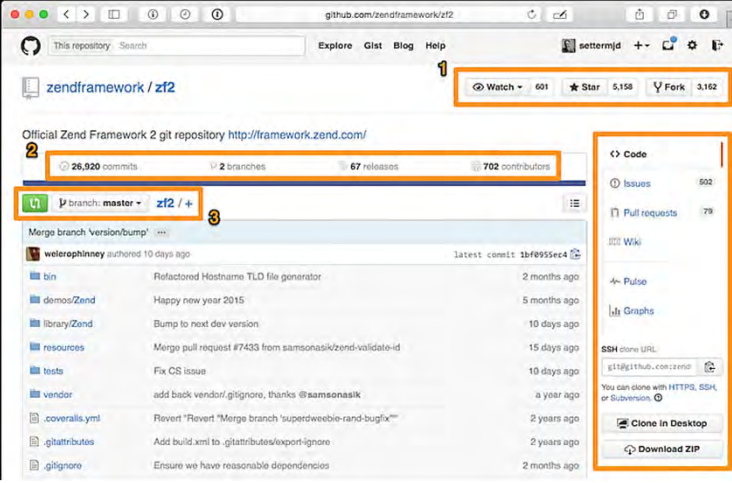
Resumen gráfico del uso de git



38

Alfredo Abad



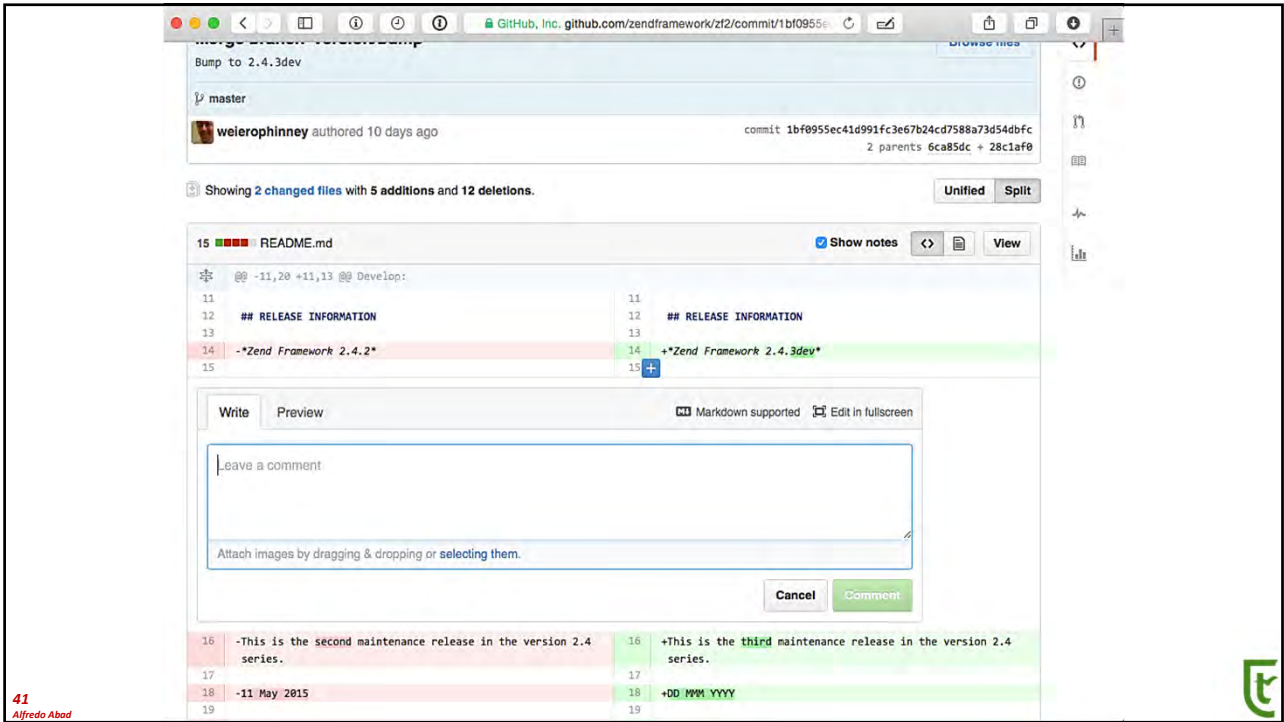


Github

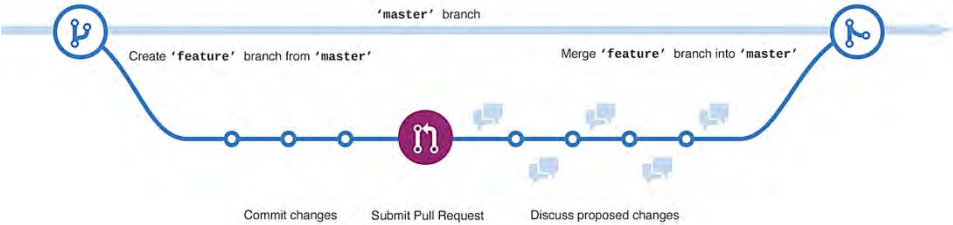
39
Alfredo Abad

¿Qué es GitHub?

- Es una plataforma de hosting de código que aloja en la nube los proyectos Git de sus usuarios registrados
 - De forma que proporciona una plataforma web para la gestión de un repositorio Git público a través de un homepage de proyecto
- Para clonar un proyecto basta con hacer clic en el botón web correspondiente de la página de proyecto (equivalente a git clone)
 - Hay otras opciones disponibles desde la homepage como commit, diff, visualizar historial, etc.



Ejemplo de branch en GitHub



Barra de navegación de GitHub

- **Code:** The view you're on by default, showing the files in the project.
- **Issues:** A simple but effective issue tracker, whether you and the team want to report bugs and problems, make requests for new features, or other such tasks.
- **Wiki:** A simple but effective wiki for documenting the project in more detail than a standard **README** file allows.
- **Pulse:** A summary of statistics about the project, including open and closed issues. Here is where you find out how active the project is.
- **Graphs:** A timeline of commits, followed by a breakdown of commits by individual contributor. You can then use the available tabs to look at the project activity in detail, based on a series of key metrics including code frequency, least to most active days for contributions, and so on.

43
Alfredo Abad



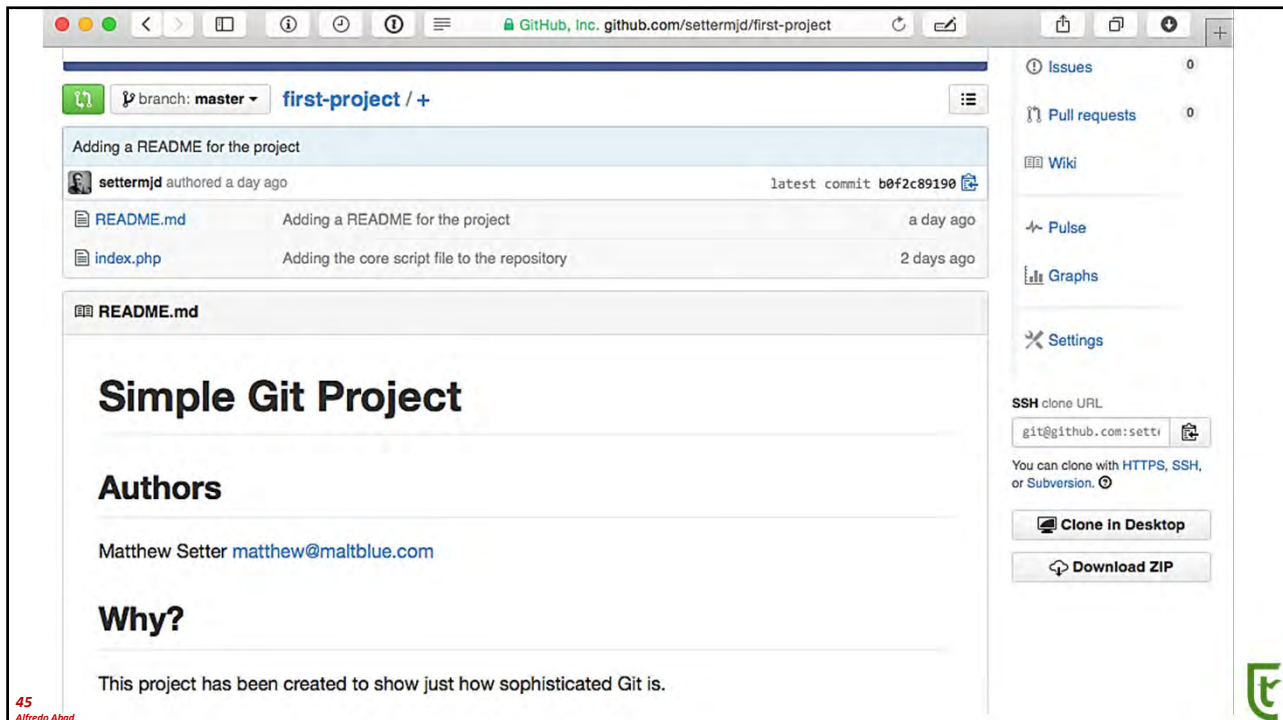
Añadir un proyecto a GitHub

The screenshot shows the GitHub 'New repository' page. The top navigation bar has a 'New repository' button highlighted. The main form includes the following fields and options:

- Owner:** settermjd
- Repository name:** (empty field)
- Description (optional):** (empty text area)
- Public/Private:** The 'Public' option is selected. The 'Private' option is also visible.
- Initialize this repository with a README:** (checkbox, unchecked)
- Add .gitignore:** None
- Add a license:** None
- Create repository:** (green button)

44
Alfredo Abad





Gitea:

Una alternativa open source y gratuita a GitHub

<https://atarea.es/podcast/mi-propio-github-de-la-mano-de-gitea/>

46
Alfredo Abad



Resumen de comandos útiles para gestionar git

47
Alfredo Abad



crea un repositorio nuevo

Crea un directorio nuevo, ábrelo y ejecuta

```
git init
```

para crear un nuevo repositorio de git.

hacer checkout a un repositorio

Crea una copia local del repositorio ejecutando

```
git clone /path/to/repository
```

Si utilizas un servidor remoto, ejecuta

```
git clone username@host:/path/to/repository
```

48
Alfredo Abad



flujo de trabajo

Tu repositorio local esta compuesto por tres "árboles" administrados por git. El primero es tu **Directorio de trabajo** que contiene los archivos, el segundo es el **Index** que actua como una zona intermedia, y el último es el **HEAD** que apunta al último commit realizado.



49
Alfredo Abad



add & commit

Puedes registrar cambios (añadirlos al **Index**) usando

```
git add <filename>
```

```
git add .
```

Este es el primer paso en el flujo de trabajo básico. Para hacer commit a estos cambios usa

```
git commit -m "Commit message"
```

Ahora el archivo esta incluído en el **HEAD**, pero aún no en tu repositorio remoto.

50
Alfredo Abad



envío de cambios

Tus cambios están ahora en el **HEAD** de tu copia local. Para enviar estos cambios a tu repositorio remoto ejecuta

```
git push origin master
```

Reemplaza *master* por la rama a la que quieres enviar tus cambios.

Si no has clonado un repositorio ya existente y quieres conectar tu repositorio local a un repositorio remoto, usa

```
git remote add origin <server>
```

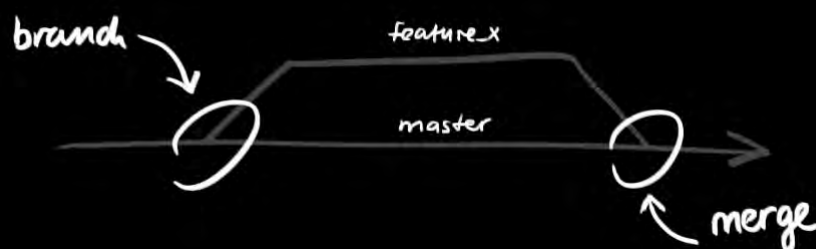
Ahora podrás subir tus cambios al repositorio remoto seleccionado.

51
Alfredo Abad



ramas

Las ramas son utilizadas para desarrollar funcionalidades aisladas unas de otras. La rama *master* es la rama "por defecto" cuando creas un repositorio. Crea nuevas ramas durante el desarrollo y fusiónalas a la rama principal cuando termines.



52
Alfredo Abad



Crea una nueva rama llamada "feature_x" y cámbiate a ella usando

```
git checkout -b feature_x
```

vuelve a la rama principal

```
git checkout master
```

y borra la rama

```
git branch -d feature_x
```

Una rama nueva *no estará disponible para los demás* a menos que subas (push) la rama a tu repositorio remoto

```
git push origin <branch>
```

53
Alfredo Abad



actualiza & fusiona

Para actualizar tu repositorio local al commit más nuevo, ejecuta

```
git pull
```

en tu directorio de trabajo para *bajar* y *fusionar* los cambios remotos.

Para fusionar otra rama a tu rama activa (por ejemplo master), utiliza

```
git merge <branch>
```

en ambos casos git intentará fusionar automáticamente los cambios.

Desafortunadamente, no siempre será posible y se podrán producir *conflictos*. Tú eres responsable de fusionar esos *conflictos* manualmente

al editar los archivos mostrados por git. Después de modificarlos,

necesitas marcarlos como fusionados con

```
git add <filename>
```

Antes de fusionar los cambios, puedes revisarlos usando

```
git diff <source_branch> <target_branch>
```

54
Alfredo Abad



etiquetas

Se recomienda crear etiquetas para cada nueva versión publicada de un software. Este concepto no es nuevo, ya que estaba disponible en SVN.

Puedes crear una nueva etiqueta llamada *1.0.0* ejecutando

```
git tag 1.0.0 1b2e1d63ff
```

1b2e1d63ff se refiere a los 10 caracteres del commit id al cual quieres referirte con tu etiqueta. Puedes obtener el commit id con

```
git log
```

también puedes usar menos caracteres que el commit id, pero debe ser un valor único.

55
Alfredo Abad



reemplaza cambios locales

En caso de que hagas algo mal (lo que seguramente nunca suceda ;) puedes reemplazar cambios locales usando el comando

```
git checkout -- <filename>
```

Este comando reemplaza los cambios en tu directorio de trabajo con el último contenido de HEAD. Los cambios que ya han sido agregados al Index, así como también los nuevos archivos, se mantendrán sin cambio.

Por otro lado, si quieres deshacer todos los cambios locales y commits, puedes traer la última versión del servidor y apuntar a tu copia local principal de esta forma

```
git fetch origin
```

```
git reset --hard origin/master
```

56
Alfredo Abad



datos útiles

Interfaz gráfica por defecto

```
gitk
```

Colores especiales para la consola

```
git config color.ui true
```

Mostrar sólo una línea por cada commit en la traza

```
git config format.pretty oneline
```

Agregar archivos de forma interactiva

```
git add -i
```

57
Alfredo Abad



Enlaces y recursos

• Clientes gráficos

- [Tower \(OSX\)](#)
- [Source Tree \(OSX, free\)](#)
- [GitHub for Mac \(OSX, free\)](#)
- [GitBox \(OSX\)](#)

• Guías

- [Git Community Book](#)
- [Think like a git](#)
- [GitHub Help](#)

• Conectar Git y GitHub

- <https://www.htcmania.com/showthread.php?t=674598>

58
Alfredo Abad





Recurso de consulta intensiva sobre git (estructurado y en español)

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

59
Alfredo Abad



Manual de GIT y ejercicios resueltos

Estudiar las siguientes páginas web:

Manuales:

<https://aprendeconalf.es/docencia/git/manual/>

<https://www.howtoforge.com/advanced-git-tutorial/>

Ejercicios: <https://aprendeconalf.es/docencia/git/ejercicios/>

Branching: https://learngitbranching.js.org/?locale=es_ES

60
Alfredo Abad



Ejemplo de sesión en GIT

<https://www.howtoforge.com/git-basics/>

61
Alfredo Abad



Initialize a local repository and perform basic operations on it.

Before you proceed check the operating system you are using and if Git is available on it.

I already have it installed on my Ubuntu 20.04 LTS server.

```
cat /etc/issue
```

```
git --version
```

```
rahul@rahul:~$ cat /etc/issue
Ubuntu 20.04.1 LTS \n \l

rahul@rahul:~$ git --version
git version 2.25.1
rahul@rahul:~$
```

62
Alfredo Abad



If you are using the same operating system as mine then you can install it using the following command if you do not have it on to your server.

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

Now let's get started with basic operations in Git.

Check the current directory and create a new directory in it. Change your working directory to the directory you created.

```
pwd
```

```
mkdir my-first-git-repo
```

```
cd my-first-git-repo/
```

```
ls -la
```

```
rahul@rahul:~$ pwd
/home/rahul
rahul@rahul:~$ mkdir my-first-git-repo
rahul@rahul:~$ cd my-first-git-repo/
rahul@rahul:~/my-first-git-repo$ ls -la
total 8
drwxrwxr-x 2 rahul rahul 4096 Dec 16 13:23 .
drwxr-xr-x 7 rahul rahul 4096 Dec 16 13:23 ..
rahul@rahul:~/my-first-git-repo$
```

63

Alfredo Abad



Till this point, we just have a directory that does not contain any files in it and is a simple directory in the Linux system. Use the following command to convert the simple directory into a Git repository.

```
git init
```

```
ls -la
```

Now you can see that a new hidden folder has been created with Git configurations in it.

```
cat .git/config
```

```
rahul@rahul:~/my-first-git-repo$ git init
Initialized empty Git repository in /home/rahul/my-first-git-repo/.git/
rahul@rahul:~/my-first-git-repo$ ls -la
total 12
drwxrwxr-x 3 rahul rahul 4096 Dec 16 13:23 .
drwxr-xr-x 7 rahul rahul 4096 Dec 16 13:23 ..
drwxrwxr-x 7 rahul rahul 4096 Dec 16 13:23 .git
rahul@rahul:~/my-first-git-repo$ cat .git/
HEAD      branches/  config      description hooks/      info/      objects/    refs/
rahul@rahul:~/my-first-git-repo$ cat .git/config
[core]
  repositoryformatversion = 0
  filemode = true
  bare = false
  logallrefupdates = true
rahul@rahul:~/my-first-git-repo$
```

64

Alfredo Abad



Upon initializing a Git repository it does not contain the user identity. To set a user name and email ID as user identity use the following commands. This identity is appended in the Git log messages.

```
git config --list
```

```
git config user.name rahul
```

```
git config user.email rahul@example.com
```

```
git config --list
```

Once you set the user identity and and list the config you can see that the username and email have been set.

```
rahul@rahul:~/my-first-git-repo$ git config --list
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
rahul@rahul:~/my-first-git-repo$ git config user.name rahul
rahul@rahul:~/my-first-git-repo$ git config user.email rahul@example.com
rahul@rahul:~/my-first-git-repo$ git config --list
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
user.name=rahul
user.email=rahul@example.com
rahul@rahul:~/my-first-git-repo$
```

65

Alfredo Abad



You can even check the username and email ID which has been set as a user identity using the following commands.

```
git config user.name
```

```
git config user.email
```

```
rahul@rahul:~/my-first-git-repo$ git config user.name
rahul
rahul@rahul:~/my-first-git-repo$ git config user.email
rahul@example.com
rahul@rahul:~/my-first-git-repo$
```

The above commands set user identity which is limited to a particular repository.

Can even set user identity which will be used by all the repositories on your machine and it is known as global identity. If you set the global identity, the repositories which do not have a local identity will use this global identity while committing the messages.

66

Alfredo Abad



If the global identity is not set and you try to list the username and email you won't get any information.

```
git config --global user.name
```

```
git config --global user.email
```

But once you set the username and email ID and after that you check the identity you can see it being displayed.

```
git config --global user.name rahul
```

```
git config --global user.email rahul@example.com
```

```
git config --global user.name
```

```
git config --global user.email
```

67
Alfredo Abad



Anyone and state the global identity using the following commands.

```
git config --global --unset user.name
```

```
git config --global --unset user.email
```

```
git config --global user.name
```

```
git config --global user.email
```

Check the following screenshot for your reference purpose

```
rahul@rahul:~/my-first-git-repo$ git config --global user.name
rahul@rahul:~/my-first-git-repo$ git config --global user.email
rahul@rahul:~/my-first-git-repo$ git config --global user.name rahul
rahul@rahul:~/my-first-git-repo$ git config --global user.email rahul@example.com
rahul@rahul:~/my-first-git-repo$ git config --global user.name
rahul
rahul@rahul:~/my-first-git-repo$ git config --global user.email
rahul@example.com
rahul@rahul:~/my-first-git-repo$ git config --global --unset user.name
rahul@rahul:~/my-first-git-repo$ git config --global --unset user.email
rahul@rahul:~/my-first-git-repo$ git config --global user.name
rahul@rahul:~/my-first-git-repo$ git config --global user.email
rahul@rahul:~/my-first-git-repo$
```

68
Alfredo Abad



Now you know that you can check your local identity using the commands but you can even check what has been set by reading `.git/config` file.

```
git config user.name
```

```
git config user.email
```

If you read the content of `.git/config` file, you will see that the username and email have been set in this file. And this is from where the local identity is used. Can even change the identity by editing the same file.

```
vim .git/config
```

```
cat .git/config
```

```
git config user.name
```

```
git config user.email
```

69
Alfredo Abad



```
rahul@rahul:~/my-first-git-repo$ git config user.name
rahul
rahul@rahul:~/my-first-git-repo$ git config user.email
rahul@example.com
rahul@rahul:~/my-first-git-repo$ vim .git/config
rahul@rahul:~/my-first-git-repo$ cat .git/config
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
[user]
    name = rahul-s
    email = rahul-s@example.com
rahul@rahul:~/my-first-git-repo$ git config user.name
rahul-s
rahul@rahul:~/my-first-git-repo$ git config user.email
rahul-s@example.com
rahul@rahul:~/my-first-git-repo$
```

Once you have your commit identity set for your local repository, the next step is to create a file and add it to the repository.

Let's create an empty file in the local repository.

```
touch README.md
```

70
Alfredo Abad



After adding the file if you check the status of the repository you will see that the file is now untracked by Git.

```
git status
```

```
git branch
```

Before you commit your file to Git you need to add it first. Use the following command to add your file to Git so that it will be ready for the commit.

```
git add README.md
```

```
git status
```

```
git log
```

While committing a file to the Git repository you need to add a message to it.

```
git commit -m "my first commit - added README.md"
```

Now, upon checking the logs you will see that the file has been committed with the message that we specified in the commit.

71
Alfredo Abad



```
git log
```

By default your commits are added to the master branch.

```
git branch
```

```
rahul@rahul:~/my-first-git-page$ touch README.md
rahul@rahul:~/my-first-git-page$ git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
rahul@rahul:~/my-first-git-page$ git branch
rahul@rahul:~/my-first-git-page$ git add README.md
rahul@rahul:~/my-first-git-page$ git status
On branch master

No commits yet

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

rahul@rahul:~/my-first-git-page$ git log
fatal: your current branch 'master' does not have any commits yet
rahul@rahul:~/my-first-git-page$ git commit -m "my first commit - added README.md"
[master (root-commit) 7fc4ddb] my first commit - added README.md
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md
rahul@rahul:~/my-first-git-page$
rahul@rahul:~/my-first-git-page$ git log
commit 7fc4ddb737bdcda33bf4f41026cc9ce7c1e57e6 (HEAD -> master)
Author: rahul-s <rahul-s@example.com>
Date:   Wed Dec 16 13:35:17 2020 +0000

    my first commit - added README.md
rahul@rahul:~/my-first-git-page$ git branch
* master
```

72
Alfredo Abad



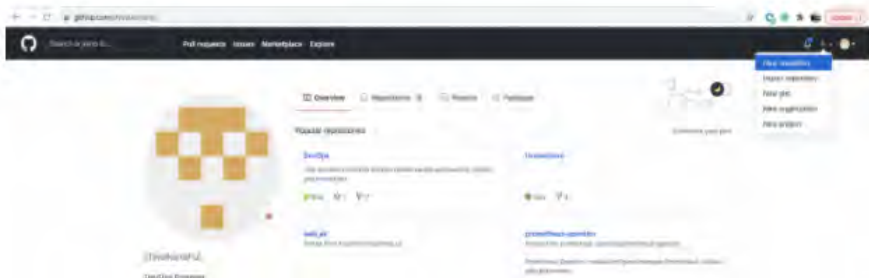
Clone an existing repository and perform basic operations on it

Till this point, we worked with a repository on our local system.

Now, we will see performing a few operations on an existing repository. If you do not have an existing repository on a code hosting platform, such as Github, for version control and collaboration then first create an account on Github to create a repository.

You can create an account on Github [here](#).

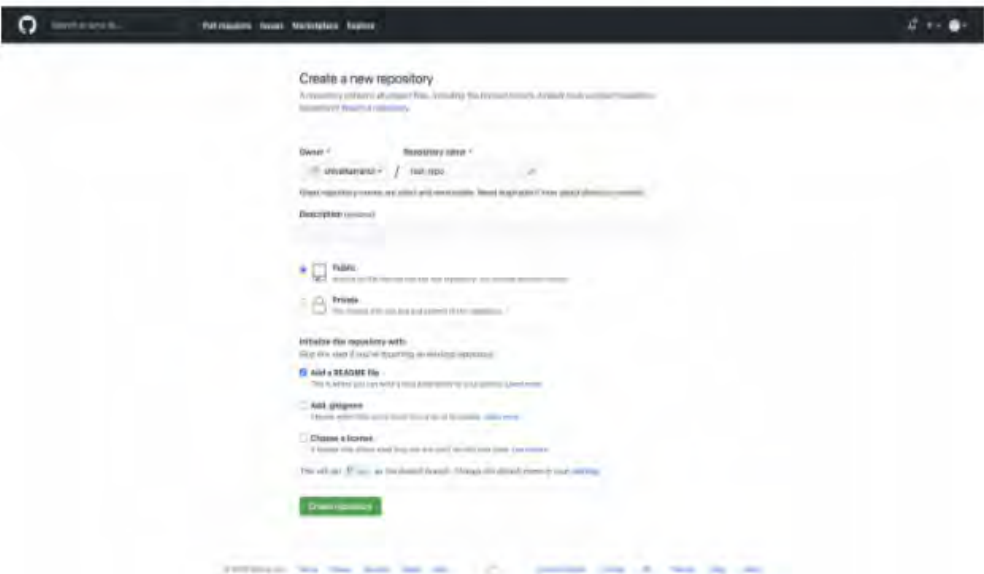
Once you create an account on Github it's time to create a new repository. To create a repository click on the "+" icon in the top right of the screen and click on the "New repository" option.



73
Alfredo Abad



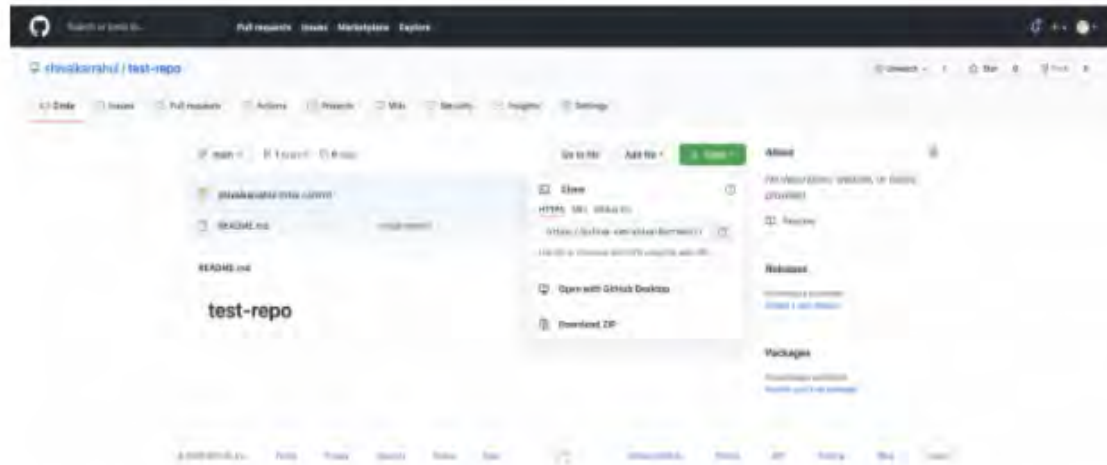
Give a name to the repository to be created and set its access level as private or public based on your requirement. You can even add a readme file while creating the repository. Click on the "Create repository" button to create a repository with the configuration you specified.



74
Alfredo Abad



Once you create a repository the next step is to clone it on your local system. To get its URL click on "Code" and copy the URL from HTTPS section.



Go back to your system and change your working directory

75
Alfredo Abad



Go back to your system and change your working directory

```
pwd
```

```
cd ..
```

```
pwd
```

Clone the repository on your local machine using the following command, You need to specify your Git URL.

```
git clone <your-repository-URL>
```

```
cd test-repo/
```

```
ls -lt
```

The above command will clone your repository from Github to your local machine and now if you check the Git log you will see one log message with the initial commit which I added a README.md file.

76
Alfredo Abad



```
git log
```

```
rahul@rahul:~/my-first-git-repo$ pwd
/home/rahul/my-first-git-repo
rahul@rahul:~/my-first-git-repo$ cd ..
rahul@rahul:~/my-first-git-repo$ pwd
/home/rahul
rahul@rahul:~/my-first-git-repo$ git clone https://github.com/shivakarrahul/test-repo.git
Cloning into 'test-repo'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 591 bytes | 591.00 KiB/s, done.
rahul@rahul:~/my-first-git-repo$ cd test-repo/
rahul@rahul:~/test-repo$ ls -lt
total 4
-rw-rw-r-- 1 rahul rahul 11 Dec 17 05:39 README.md
rahul@rahul:~/test-repo$ git log
commit cb39c69233238ac9f8867f6b8a87ac2a0979abd (HEAD -> main, origin/main, origin/XXXX)
Author: shivakarrahul <rahulshivakar@rediffmail.com>
Date: Thu Dec 17 11:09:25 2020 +0530

Initial commit
rahul@rahul:~/test-repo$
```

Now let's create a new file, add it to Git and commit it with a message.

```
pwd
```

```
touch first-file-to-push
```

```
git status
```

77
Alfredo Abad



```
git add first-file-to-push
```

```
git status
```

```
git commit -m "first commit to push to the remote repository"
```

Upon committing, you'll be asked to set your user identity

```
git config --list
```

```
git config user.name rahul
```

```
git config user.email rahul@example.com
```

```
git config --list
```

```
git log
```

78
Alfredo Abad

After you set your user identity you can fix the identity in the commit we did.



```

[rahul@rahul:~/test-repo$ pwd
/home/rahul/test-repo
[rahul@rahul:~/test-repo$ touch first-file-to-push
[rahul@rahul:~/test-repo$ git status
On branch main
Your branch is up to date with 'origin/main'.

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

nothing added to commit but untracked files present (use "git add" to track)
[rahul@rahul:~/test-repo$ git add first-file-to-push
[rahul@rahul:~/test-repo$ git status
On branch main
Your branch is up to date with 'origin/main'.

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

[rahul@rahul:~/test-repo$ git commit -m "first commit to push to the remote repository"
[main 9c2436e] first commit to push to the remote repository
Committer: rahul <rahul@rahul>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

```

79

Alfredo Abad



```

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 first-file-to-push
[rahul@rahul:~/test-repo$ git config --list
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
remote.origin.url=https://github.com/shivalkarrahul/test-repo.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
branch.main.remote=origin
branch.main.merge=refs/heads/main
[rahul@rahul:~/test-repo$ git config user.name rahul
[rahul@rahul:~/test-repo$ git config user.email rahul@example.com
[rahul@rahul:~/test-repo$ git config --list
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
remote.origin.url=https://github.com/shivalkarrahul/test-repo.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
branch.main.remote=origin
branch.main.merge=refs/heads/main
user.name=rahul
user.email=rahul@example.com
[rahul@rahul:~/test-repo$ git log
commit 9c2436e0e583b3268f4de74641bade5673b1694 (HEAD -> main)
Author: rahul <rahul@rahul>
Date: Thu Dec 17 05:41:13 2020 +0000

    first commit to push to the remote repository

commit cb39cb692333bac9f0887fd588a07ac26b9f9abd (origin/main, origin/HEAD)
Author: shivalkarrahul <rahulshivalkar@rediffmail.com>
Date: Thu Dec 17 11:09:25 2020 +0530

    Initial commit

```

80

Alfredo Abad



To fix the identity, execute the following command. You will get an editor and there you can change the commit if required.

```
git commit --amend --reset-author
```

Upon checking the Git log, you will see the log has the identity you set and the commit you fixed.

```
git log
```

```
git status
```

You are all set to push your local changes to the remote repository. The following command to push changes to Github in your repository.

```
git push
```

81
Alfredo Abad



```

rahul@rahul:~/test-repo$ git commit --amend --reset-author
[main d811342] :wq!first commit to push to the remote repository
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 first-file-to-push
rahul@rahul:~/test-repo$ git log
commit d811342a2aeb06faad1986fa35ec73655838ad1 (HEAD -> main)
Author: rahul <rahul@example.com>
Date: Thu Dec 17 05:42:17 2020 +0000

    :wq!first commit to push to the remote repository

commit 5a39cbb92333bac9fa887f0588e07ac26b9f9abd (origin/main, origin/HEAD)
Author: shivalkarrahal <rahulshivalkar@rediffmail.com>
Date: Thu Dec 17 11:09:25 2020 +0530

    Initial commit
rahul@rahul:~/test-repo$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)


nothing to commit, working tree clean
rahul@rahul:~/test-repo$ git push
Username for 'https://github.com': rahulshivalkar@rediffmail.com
Password for 'https://rahulshivalkar@rediffmail.com@github.com':
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 303 bytes | 303.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/shivalkarrahal/test-repo.git
cb39cbb..d811342 main -> main
rahul@rahul:~/test-repo$

```

82
Alfredo Abad

You can now find your changes in the remote repository. Go to the repository and you can find the commit and the file you added.





Sometimes you may need to exclude particular files from being added to the repository.

```
pwd
```

```
ls -la
```

83
Alfredo Abad

You can create a .gitignore file and add the pattern of files to be ignored by Git.

```
touch .gitignore
```

```
vim .gitignore
```

Here, I have added "*.txt" which will ignore all files ending with ".txt".

```
cat .gitignore
```

After you add a pattern to the .gitignore file and create a file of that pattern in the repository, the file will be ignored by Git.

Since we have added *.txt and now if we try to create a file ending with .txt, it will be ignored by Git in the "git add" operation.

```
touch ignore-this-file.txt
```

```
git status
```

84
Alfredo Abad

The files which do not match the pattern will not be ignored by Git

```
touch dont-ignore-this-file.doc
```

```
git status
```

You can then add all the files in the repository and commit them with a message. You will notice that the files which have been ignored will not get added.

```
git add .
```

```
git status
```

```
git commit -m "added .gitignore and a sample doc file"
```

```
git status
```

Once the files have been added and committed with a message, they are ready to be pushed to the remote repository.

```
git push
```

85
Alfredo Abad



```
rahul@rahul:~/test-repo$ pwd
/home/rahul/test-repo
rahul@rahul:~/test-repo$ ls -la
total 16
drwxrwxr-x 3 rahul rahul 4096 Dec 17 05:40 .
drwxr-xr-x 9 rahul rahul 4096 Dec 17 05:42 ..
drwxrwxr-x 8 rahul rahul 4096 Dec 17 05:42 .git
-rw-rw-r-- 1 rahul rahul 11 Dec 17 05:39 README.md
-rw-rw-r-- 1 rahul rahul 0 Dec 17 05:40 first-file-to-push
rahul@rahul:~/test-repo$ touch .gitignore
rahul@rahul:~/test-repo$ vim .gitignore
rahul@rahul:~/test-repo$ cat .gitignore
*.txt
rahul@rahul:~/test-repo$ touch ignore-this-file.txt
rahul@rahul:~/test-repo$ git status
On branch main
Your branch is up to date with 'origin/main'.

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

nothing added to commit but untracked files present (use "git add" to track)
rahul@rahul:~/test-repo$ touch dont-ignore-this-file.doc
rahul@rahul:~/test-repo$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  .gitignore
  dont-ignore-this-file.doc
```

86
Alfredo Abad



87

Alfredo Abad

```

Untracked files:
(use "git add <file>..." to include in what will be committed)
.gitignore
dont-ignore-this-file.doc


nothing added to commit but untracked files present (use "git add" to track)
rahul@rahul:~/test-repo$ git add .
rahul@rahul:~/test-repo$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
(use "git restore --staged <file>..." to unstage)
new file:   .gitignore
new file:   dont-ignore-this-file.doc

rahul@rahul:~/test-repo$ git commit -m "added .gitignore and a sample doc file"
[main d671dbd] added .gitignore and a sample doc file
2 files changed, 1 insertion(+)
create mode 100644 .gitignore
create mode 100644 dont-ignore-this-file.doc
rahul@rahul:~/test-repo$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)

nothing to commit, working tree clean
rahul@rahul:~/test-repo$ git push
Username for 'https://github.com': rahulshivalkar@rediffmail.com
Password for 'https://rahulshivalkar@rediffmail.com@github.com':
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 355 bytes | 355.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/shivalkarraahul/test-repo.git
d011342..d671dbd main -> main
rahul@rahul:~/test-repo$

```



88

Alfredo Abad

If you ever feel like removing your files from the repository, you can use the "git rm" command.

pwd

ls -lt

git status

git rm dont-ignore-this-file.doc

git status


ls -la

After removing the file you can then commit and push your changes to the repository.

git commit -m "git removed dont-ignore-this-file.doc"

git status

git push



```

[rahul@rahul:~/test-repo$ pwd
/home/rahul/test-repo
[rahul@rahul:~/test-repo$ ls -la
total 16
drwxrwxr-x 3 rahul rahul 4096 Dec 17 05:40 .
drwxr-xr-x 9 rahul rahul 4096 Dec 17 05:42 ..
drwxrwxr-x 8 rahul rahul 4096 Dec 17 05:42 .git
-rw-rw-r-- 1 rahul rahul 11 Dec 17 05:39 README.md
-rw-rw-r-- 1 rahul rahul  0 Dec 17 05:40 first-file-to-push
[rahul@rahul:~/test-repo$ touch .gitignore
[rahul@rahul:~/test-repo$ vim .gitignore
[rahul@rahul:~/test-repo$ cat .gitignore
*.txt
[rahul@rahul:~/test-repo$ touch ignore-this-file.txt
[rahul@rahul:~/test-repo$ git status
On branch main
Your branch is up to date with 'origin/main'.

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

nothing added to commit but untracked files present (use "git add" to track)
[rahul@rahul:~/test-repo$ touch dont-ignore-this-file.doc
[rahul@rahul:~/test-repo$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        dont-ignore-this-file.doc

```

89

Alfredo Abad



```

  (use "git add <file>..." to include in what will be committed)
        .gitignore
        dont-ignore-this-file.doc

nothing added to commit but untracked files present (use "git add" to track)
[rahul@rahul:~/test-repo$ git add .
[rahul@rahul:~/test-repo$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   .gitignore
        new file:   dont-ignore-this-file.doc

[rahul@rahul:~/test-repo$ git commit -m "added .gitignore and a sample doc file"
[main d671dbd] added .gitignore and a sample doc file
 2 files changed, 1 insertion(+)
 create mode 100644 .gitignore
 create mode 100644 dont-ignore-this-file.doc
[rahul@rahul:~/test-repo$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
[rahul@rahul:~/test-repo$ git push
Username for 'https://github.com': rahulshivalkar@rediffmail.com
Password for 'https://rahulshivalkar@rediffmail.com@github.com':
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 355 bytes | 355.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/shivalkarrahul/test-repo.git
 d011342..d671dbd main -> main
[rahul@rahul:~/test-repo$

```

90

Alfredo Abad



COMANDOS BÁSICOS DE git

GIT es un sistema de control de versiones que nos ayuda a llevar el historial completo de modificaciones de un proyecto.

Todo desarrollador sin importar el lenguaje **debe dominar Git**.
Prof. Beto Quiroga

GIT INIT

Inicia un nuevo repositorio.

Añade un archivo a la zona de montaje. **git add *** añade uno o más archivos a la zona de montaje.

Se utiliza para listar el historial de versiones de la rama actual.

Descompone el archivo, pero conserva el contenido del mismo.

GIT CLONE

Clona un repositorio existente.

Establece el nombre del autor, el correo y demás **parámetros que Git utiliza por defecto**.

Enumera todos los archivos que deben ser confirmados.

Muestra las diferencias de archivo que aún no se ponen en escena.

GIT ADD

GIT LOG

GIT RESET

GIT CONFIG

GIT STATUS

GIT DIFF

¿Qué comandos faltó? Haremos la 2da parte con tus sugerencias

ed.team/cursos/git

EDteam

91

Alfredo Abad

Instalación de GIT en Windows

92

Alfredo Abad

tajamar.

Se descarga la versión apropiada para el sistema

- Sitio de descarga: <https://git-scm.com/download/win>

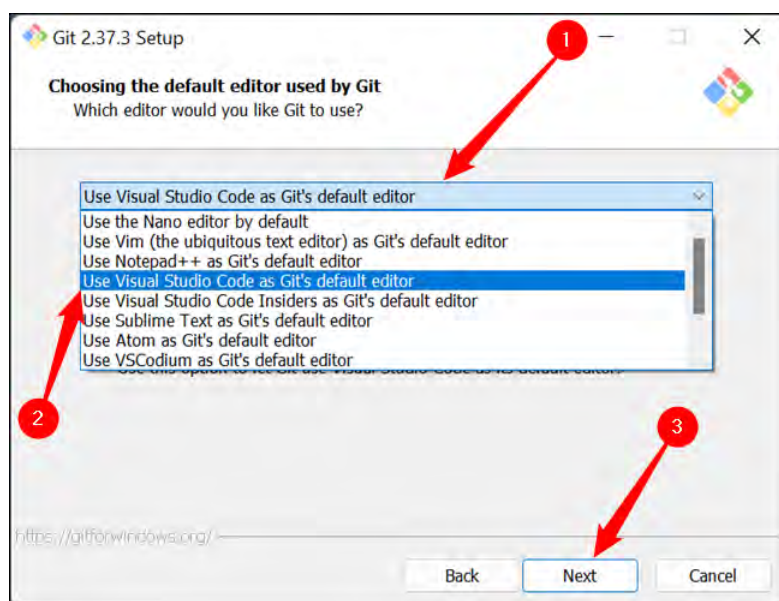


93

Alfredo Abad



Se elige el editor que utilizará GIT por defecto

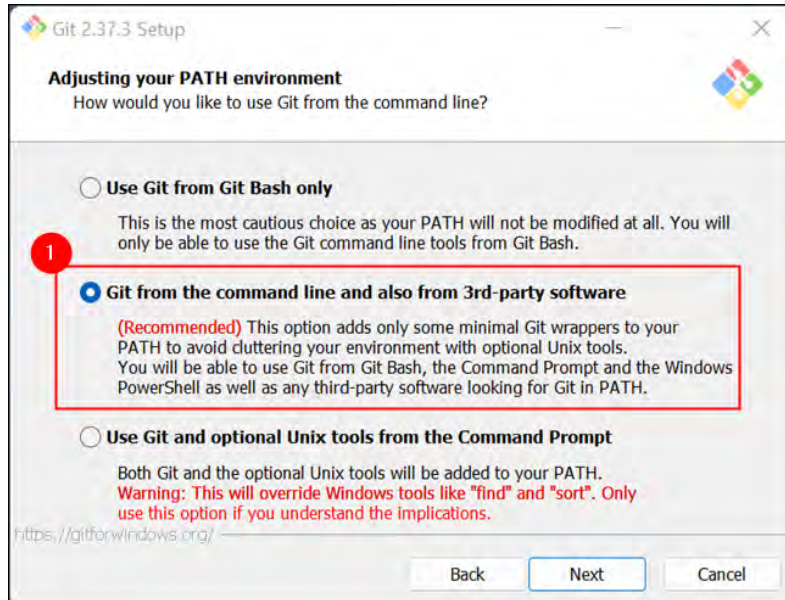


94

Alfredo Abad



Decisión de si incluir o no la ruta de instalación en el PATH



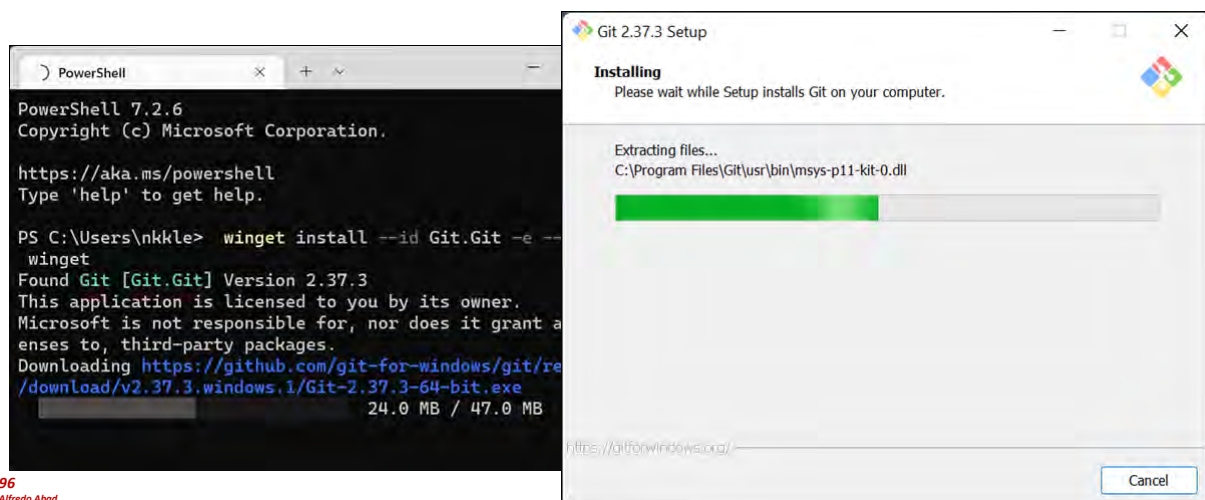
95

Alfredo Abad



También se puede instalar desde la línea de comandos

- Descarga con winget: `winget install --id Git.Git -e --source winget`



96

Alfredo Abad





Cómo instalar GIT en Ubuntu 22.04 LTS

<https://wowgold-seller.com/es/stories/8692-how-to-install-git-on-ubuntu-22-04-lts-jammy-jellyfish-linux>

97
Alfredo Abad



11 Best Graphical Git Clients and Git Repository Viewers for Linux

<https://www.tecmint.com/best-gui-git-clients-git-repository-viewers-for-linux/>

98
Alfredo Abad

