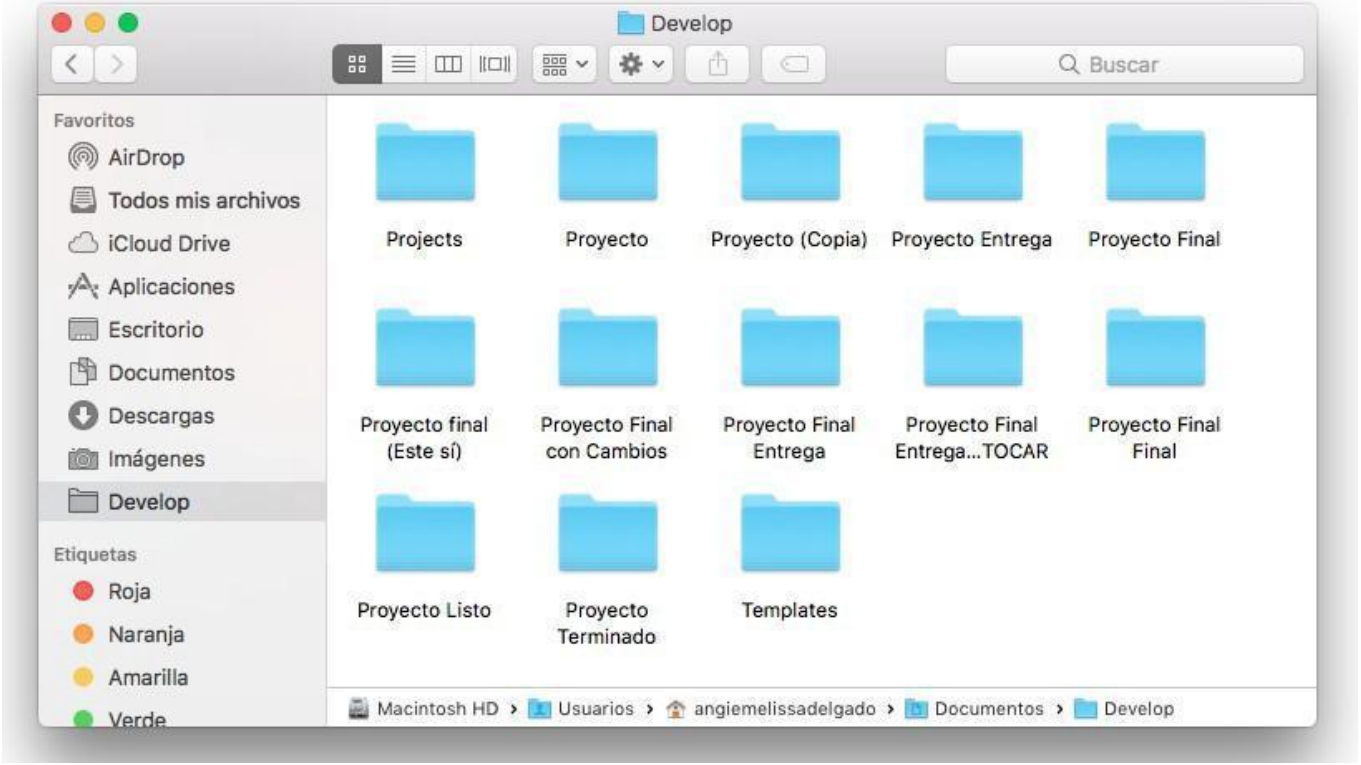
Sistemas de control de versiones o SCV

El problema


Develop
Buscar
Favoritos
AirDrop
Todos mis archivos
iCloud Drive
Aplicaciones
Escritorio
Documentos
Descargas
Imágenes
Develop
Etiquetas
Roja
Naranja
Amarilla
Verde
Projects
Proyecto
Proyecto (Copia)
Proyecto Entrega
Proyecto Final
Proyecto final (Este sí)
Proyecto Final con Cambios
Proyecto Final Entrega
Proyecto Final Entrega...TOCAR
Proyecto Final Final
Proyecto Listo
Proyecto Terminado
Templates
Macintosh HD › Usuarios › angiemelissadelgado › Documentos › Develop

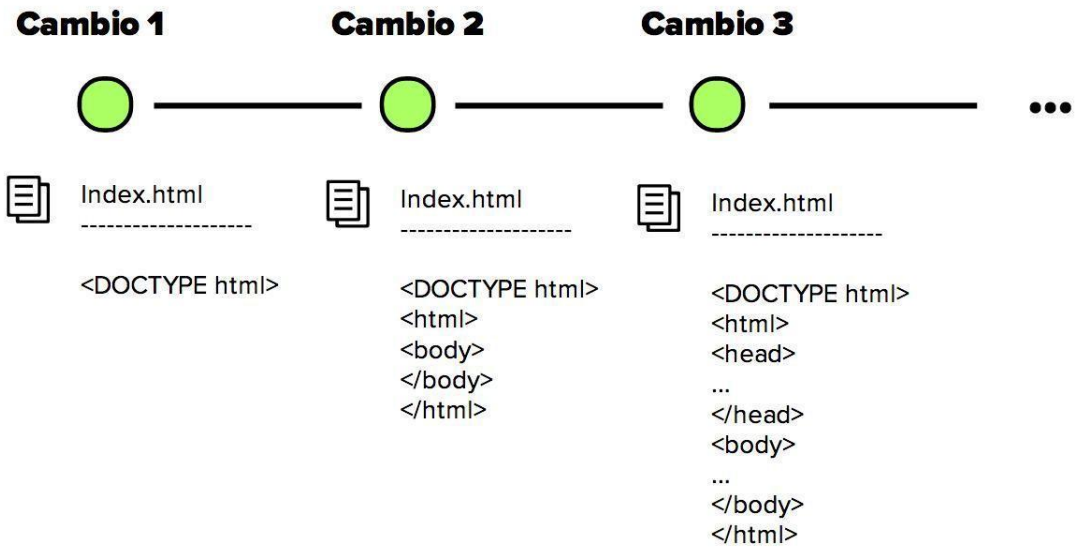
Qué es un SCV

Es un software que controla y organiza las distintas revisiones que se realizan sobre uno o varios documentos de cualquier orígen y/o tipo.

Qué es una revisión

Es un cambio realizado sobre un documento, por ejemplo añadir un párrafo, borrar un fragmento, una línea o algo similar.

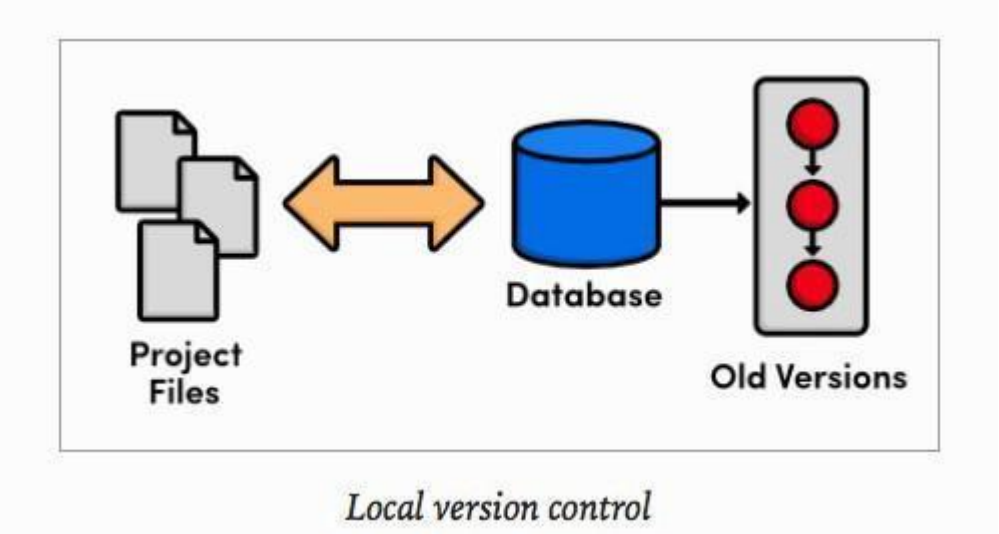
Ejemplo de cambios o revisiones



Tipos de SCVs

SCVs Locales

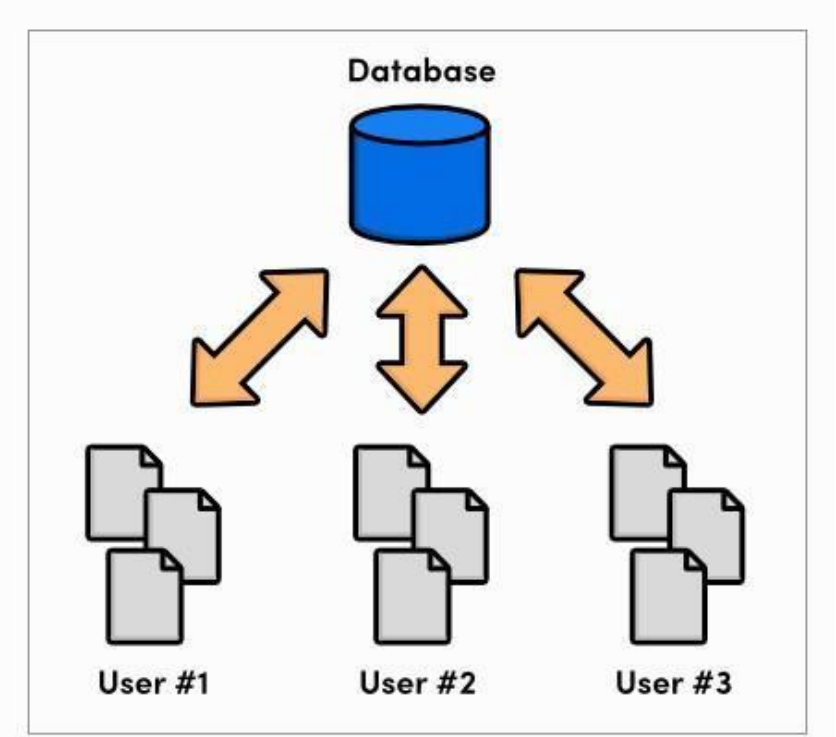
Ejemplo:
SourceSafe
(Microsoft)



Local version control

SCVs centralizados

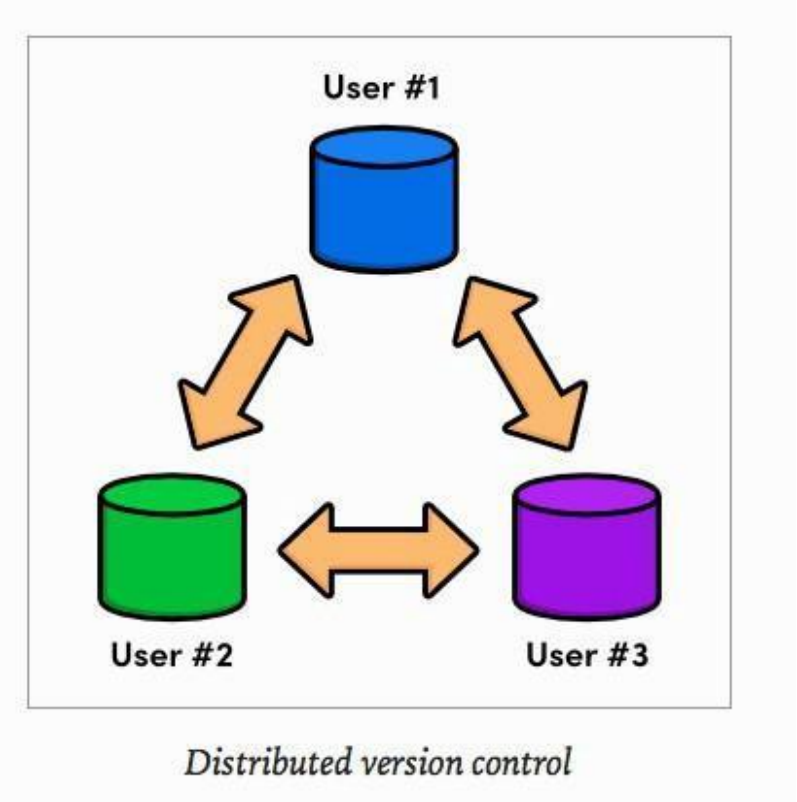
Ejemplo:
CVS
Subversion
Perforce



Centralized version control

SCVs distribuidos

Ejemplo:
GIT
Mercurial
Baazar

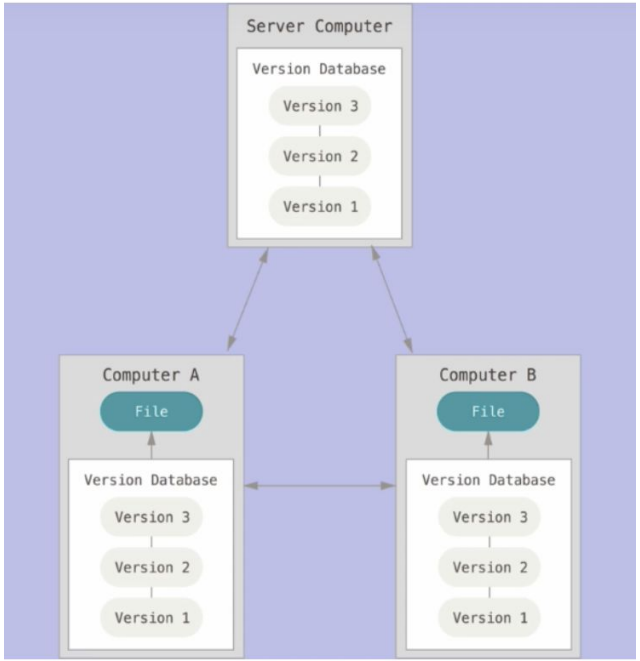


Distributed version control

GIT

¿Qué es GIT?

Es una herramienta de software que tiene como función realizar el control de versiones de archivos y/o documentos de forma **distribuida**.



Características

Es rápido

Es muy potente

No depende de un repositorio central

Es software libre

Fue diseñado por Linus Torvalds

Podemos mantener un historial completo de versiones

Características

Podemos movernos, como si tuviéramos un puntero en el tiempo, por todas las revisiones de código y desplazarnos de una manera muy ágil.

Tiene un sistema de trabajo con ramas que lo hace especialmente potente

En cuanto a la funcionalidad de las ramas, las mismas están destinadas a provocar proyectos divergentes de un proyecto principal, para hacer experimentos o para probar nuevas funcionalidades.

Las ramas pueden tener una línea de progreso diferente de la rama principal donde está el core de nuestro desarrollo.

Suma de comprobación (checksum)

- Hash SHA-1
- 40 caracteres hexadecimales (0-9 y a-f)
- Ejemplo: 24b9da6552252987aa493b52f8696cd6d3b00373

Elementos de GIT

Instantáneas

Suma de comprobación

Áreas o Secciones

Estados

Ciclo de vida de los archivos

Confirmaciones o commits

Ramas o branch

Repositorio

Fusiones con merge

Fusiones con rebase

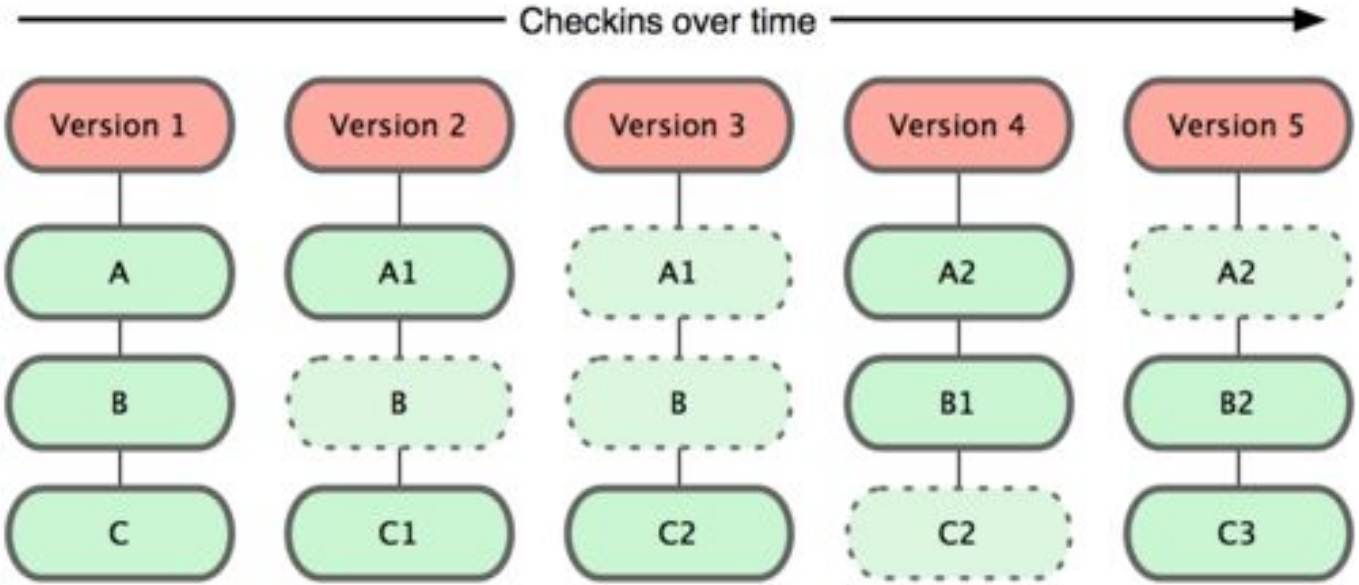
GitFlow

Repos remotos

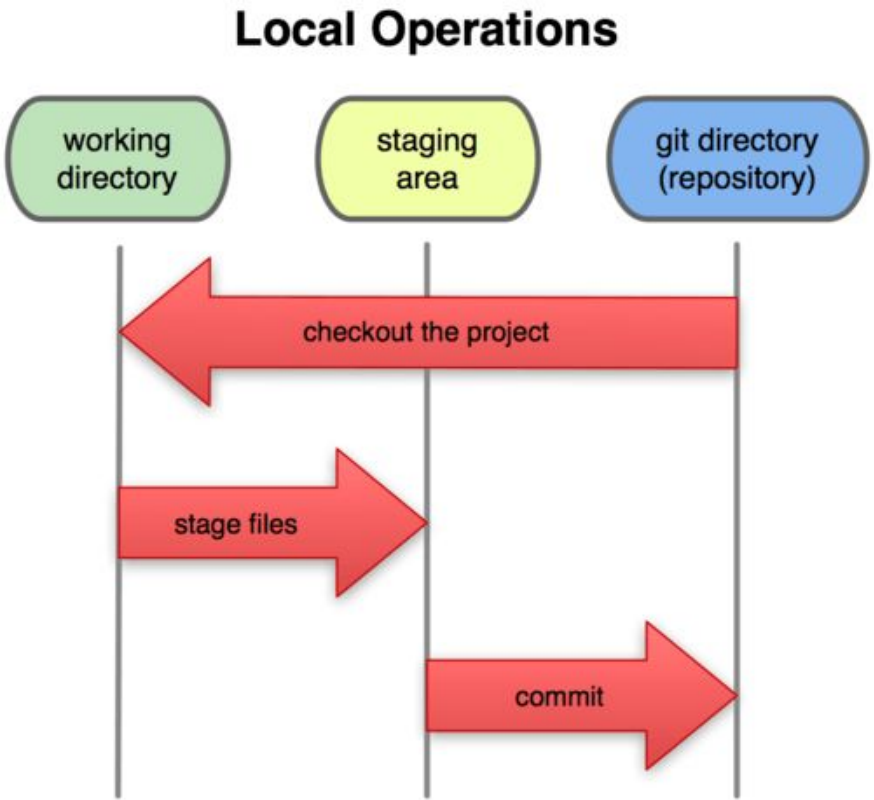
Ramas remotas

GitLab

Cómo trabaja Git - Instantáneas



Áreas de un proyecto Git



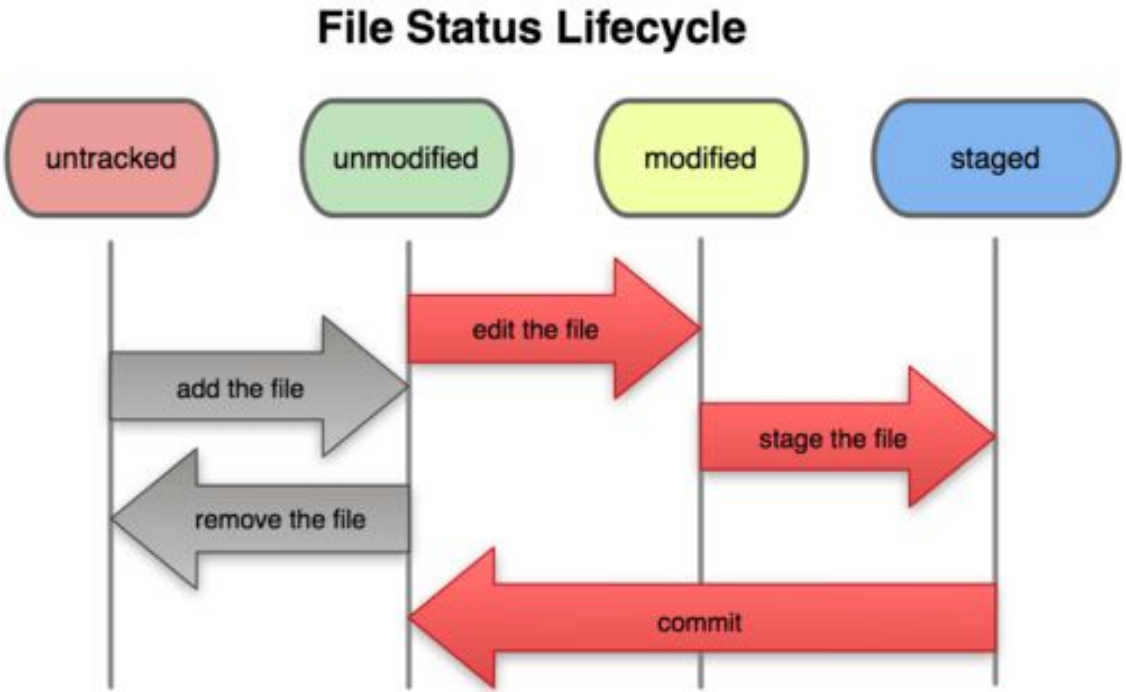
Estados

- Modified (modificado)
- Staged (preparado)
- Committed (confirmado)

Flujo básico de trabajo

1. **Modificar** una serie de archivos en tu directorio de trabajo.
2. **Preparar** los archivos, añadiendoles a tu área de preparación.
3. **Confirmar** los cambios, lo que toma los archivos tal y como están en el área de preparación, y almacena esas instantáneas de manera permanente en tu directorio de .git.

Ciclo de vida del estado de un archivo




¡Hora de ponerse a trabajar...!

Instalación

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

Linux: sudo apt-get install git

Mac:

Opción 1:

http://sourceforge.net/projects/git-osx-installer/

Opción 2:

brew install git

Comprobar instalación

En Mac o Linux abrir una terminal de comandos

En Windows abrir interprete de comando o CMD o gitbash

En cualquier caso ejecutar el comando:

$ git --version

git version 2.20.1

Configurar Git

Se usa el comando: **git config**

$ git config --global user.name “Johnny Guzman”

$ git config --global user.email “jcguzman@gmail.com”

$ git config --global core.editor nano

Repositorios

Es el lugar donde vas a colocar físicamente tu proyecto git.

Repos local: lo creas con el comando git init dentro de un directorio

Repos remoto: lo bajas (clone) a tu pc con el comando git clone

Repositorio local

Crear un directorio llamado “**mi-proyecto-local**”

Dentro de él ejecutar el comando :

$ git init

Initialized empty Git repository in /Users/johnnyguzman/Documents/projects/rollingcode/mi-proyecto-local/.git/

Repositorio remoto

Se usa el comando: **git clone**

Situarse en algún directorio dentro de tu sistema de archivos. Ej en Documentos

y ejecuta el siguiente comando:

$ git clone https://gitlab.com/rollingcodeschool/mi-proyecto.git

Esto va a crear un dir llamada **mi-proyecto** y dentro del él estará una copia de los archivos del repos remoto que acabas de descargar

Directorio .git

Es la parte más importante de un repositorio de git. Contiene metadatos, la base de datos donde gestiona todo los archivos confirmados, apuntadores, etc.

git init y git clone crean automáticamente el dir .git . En Mac o Linux lo pueden ver

dentro de **mi-proyecto** con el siguiente comando:

```
$ ls -la

total 0

ddrwxr-xr-x  10 johnnyguzman  staff  320 Apr  4 17:24 .git
```

Seguimiento de archivos

git add file_name

Ejemplo: Dentro de **mi-proyecto**, crea un archivo index.html y README.md y ejecuta los siguientes comandos:

$ git add index.html

$ git add README.md

Comprobar estado de los archivos

Dentro de **mi-proyecto** ejecuta:

```
$ git status

On branch master

No commits yet

Changes to be committed:

  (use "git rm --cached <file>..." to unstage)

        new file:   README.md

        new file:   index.html
```

Aplicar confirmación o commit

Dentro de **mi-proyecto** ejecuta:

```
$ git commit -m "primera confirmación enviada"

[master (root-commit) 405d39e] primera confirmación enviada

 2 files changed, 13 insertions(+)

 create mode 100644 README.md

 create mode 100644 index.html
```

Práctica - git add, git status y git commit

1. Modificar el archivo index.html y agregarlo al área de preparación con git add
2. Comprobar el estado del mismo con git status
3. Agregar el archivo al área de confirmaciones con git commit
4. Comprobar nuevamente el estado
5. Repetir los pasos 1-4 al menos 2 veces

Ver diferencias con git diff

Con **git diff** podrás responder la siguientes preguntas:

¿Qué has cambiado pero aún no has preparado?

¿Qué has preparado y estás a punto de confirmar?

Ejemplo git diff

Crea o edita el archivo index.html para que quede de la siguiente forma:

```
<!DOCTYPE html>
<html lang="en">
<body>
   <header>
       <h1>Mi primera pagina</h1>
   </header>
   </body>
</html>
```

usa **git add** y luego **git commit** para preparar y confirmar los cambios.

Ejemplo git diff

Vuelve a editar el archivo index.html como se muestra a continuación:

```
<!DOCTYPE html>
<html lang="en">
<body>
   <header>
       <h1>Mi primera pagina</h1>
   </header>
   <nav>
       <ul>
           <li>Principal</li>
           <li>Seguidores</li>
       </ul>
   </nav>
</body>
</html>
```

Ejemplo git diff

Usa git diff para ver las diferencias de index.html desde la último commit con los cambios que aún no han sido preparados:

```
$ git diff
diff --git a/index.html b/index.html
index 404bb2e..73df529 100644
--- a/index.html
+++ b/index.html
@@ -10,5 +10,11 @@
     <header>
         <h1>Mi primera pagina</h1>
     </header>
+    <nav>
+        <ul>
+            <li>Principal</li>
+            <li>Seguidores</li>
+        </ul>
+    </nav>
 </body>
 </html>
```

Historial de confirmaciones - git log

```
$ git log
commit b42e936d131246adbbbab350cd6458858a362115 (HEAD -> master)
Author: Johnny Guzman <guzmanjhonny@gmail.com>
Date:   Fri Apr 5 17:28:57 2019 -0300

    se agrego seccion de menu principal

commit 73679210fa9e3432d6c0aec9ffbce15b89031da8
Author: Johnny Guzman <guzmanjhonny@gmail.com>
Date:   Fri Apr 5 16:53:35 2019 -0300

    se agrego cabecera con titulo

commit 405d39e1f42a4452a7130ee9e684d281bdb3c660
Author: Johnny Guzman <guzmanjhonny@gmail.com>
Date:   Fri Apr 5 13:17:29 2019 -0300

    primera confirmacion enviada
```

Deshacer acciones de git add y git commit

El comando a usar para quitar cambios aplicados al area de preparacion es:

$ git reset HEAD nombre_archivo

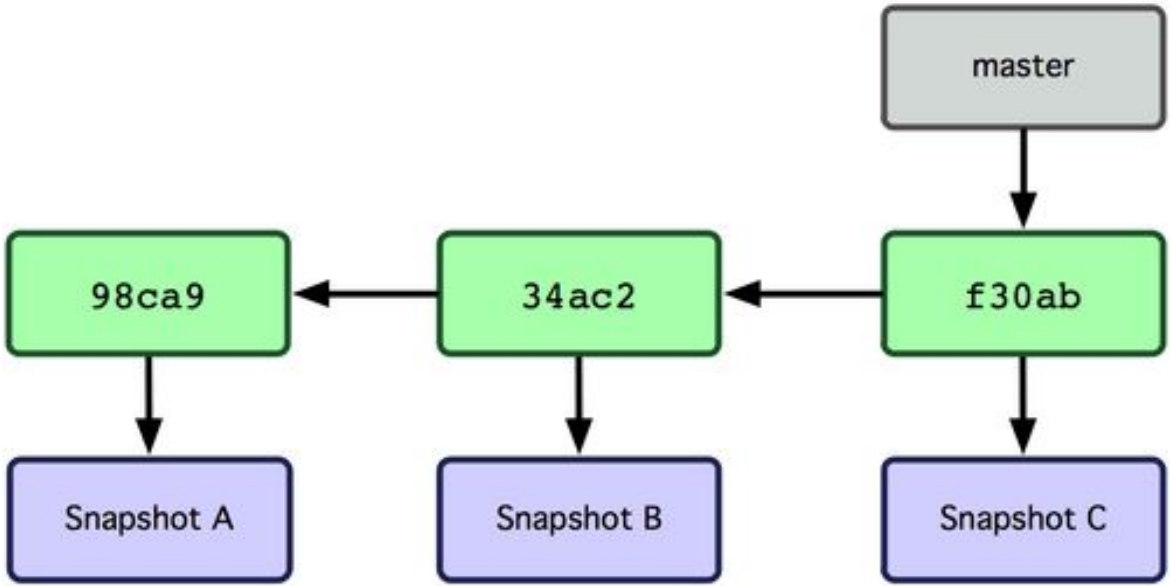
El comando a usar para sacar cambios confirmados es:

$ git checkout -- nombre_archivo

Práctica - git diff y deshacer cambios

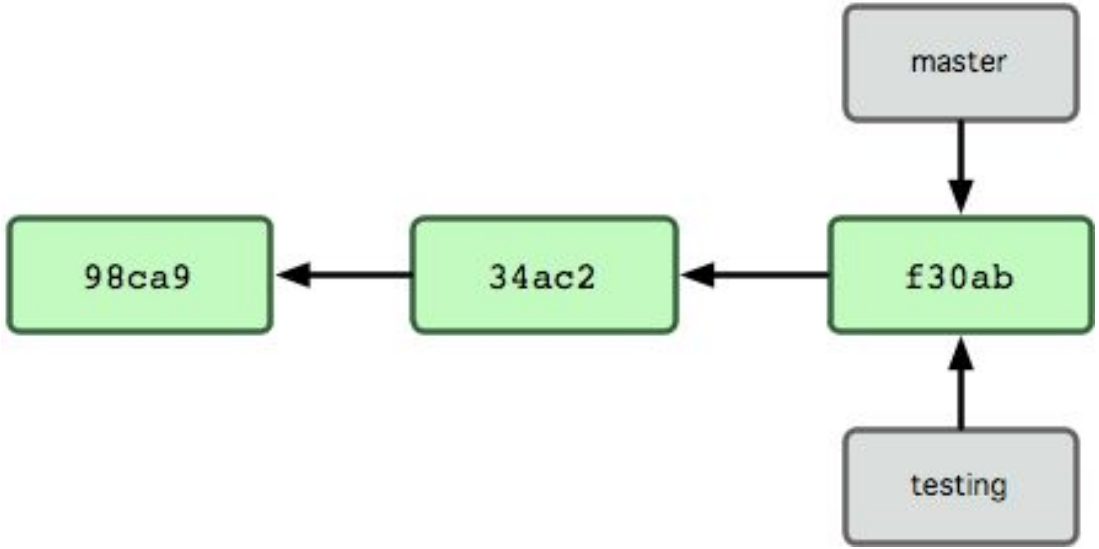
1. Modifica el archivo index.html y agregalo al area de preparación y luego confirmarlo.
2. Modifica nuevamente el archivo index.html y agregalo al area de preparacion
3. Usa el comando git diff para ver las diferencias del archivo index.html
4. Quita los cambios aplicados en index.html del area de preparacion
5. Quita los cambios aplicados en index.html del area de confirmación
6. Tu archivo debe quedar como estaba antes de ejecutar el paso 1

Ramas o branch



Creando ramas

$ git branch testing



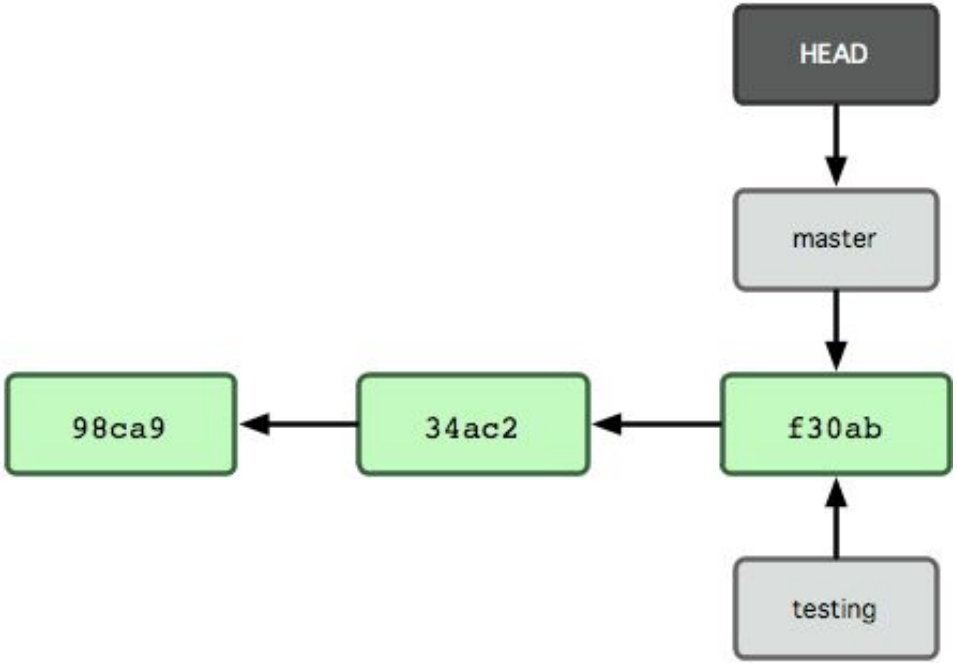
Listar ramas

```
$ git branch

master

* testing
```

El (*) indica en qué rama estamos situados actualmente.

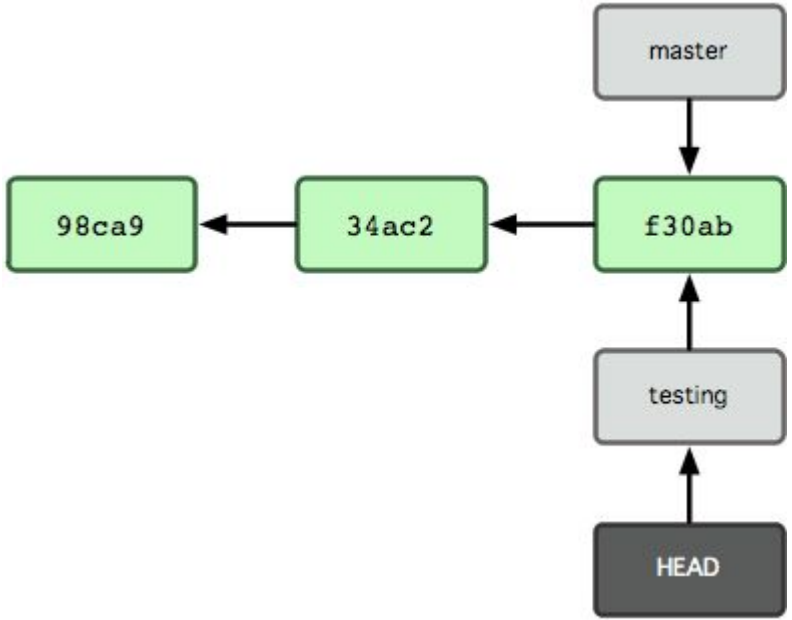
HEAD

¿Cómo sabe Git en qué rama estás en este momento?



Cambiar de branch

$ git checkout testing



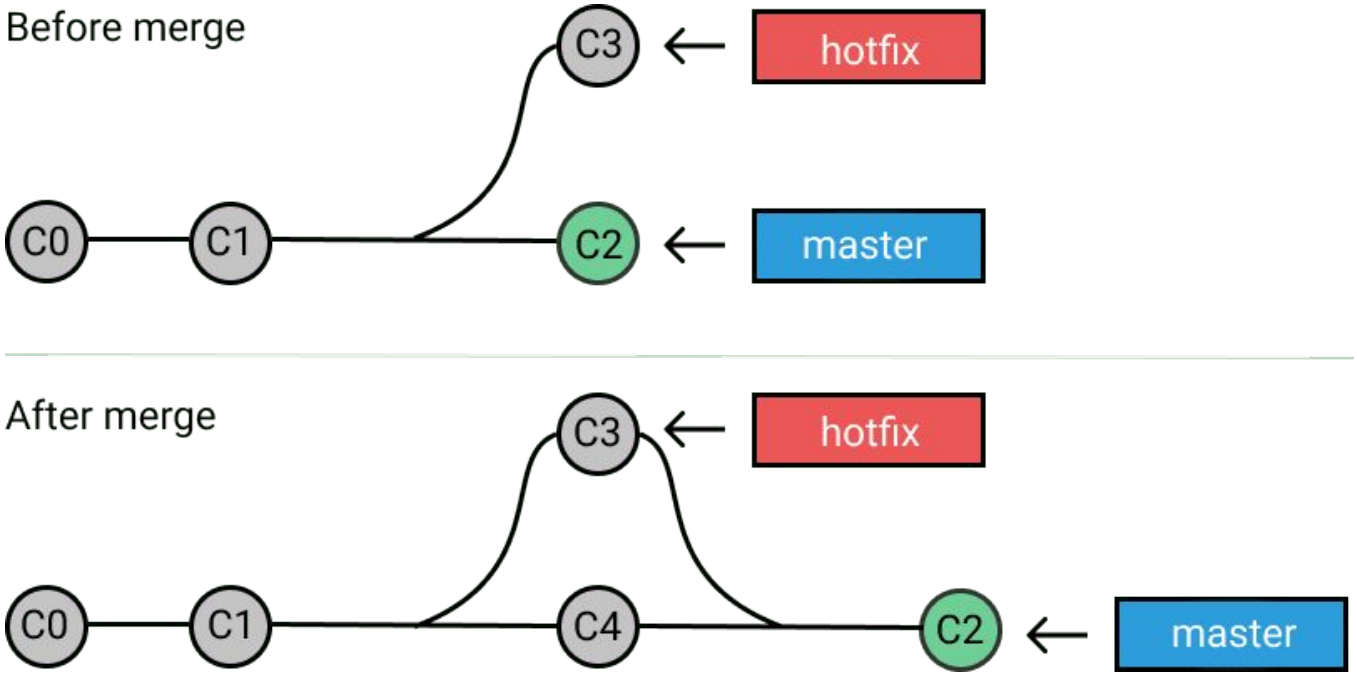
Practica git branch y git checkout

Dentro de **mi-proyecto** realiza las siguientes acciones:

1. Revisa en que rama estas situado
2. Crear una rama nueva llamada **testing** y cambiate a ella.
3. Haz modificaciones sobre el archivo index.html y agregalo al área de preparación y luego al area de confirmación.
4. Repite el paso 3 al menos 2 veces.
5. Cambiate a la rama master y compara con lo que tenías en la rama testing

Fusiones con git merge

$ git merge testing



Conflictos con git merge

```
$ git merge testing
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

Conflictos con git merge

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   index.html
```

Marcadores de conflicto y solución

Ejemplo:

```
<<<<<<< HEAD:index.html
<p>prueba</p>
=======
<p>prueba 2</p>
>>>>>>> testing:index.html
```

Pasos para la solución de un conflicto:

1. Elegir un bloque
2. Remover

<<<<<<< ======= >>>>>>>

3. git add index.html

4. git commit -m “conflicto solucionado”

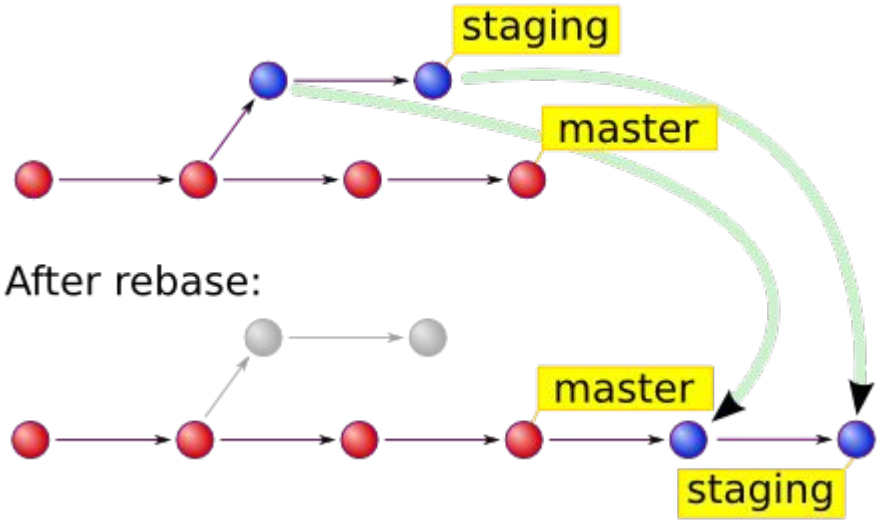
Práctica git merge y Conflictos

Dentro de **mi-proyecto** realiza las siguientes acciones:

1. Situarse en la rama testing
2. Haz cambios sobre el archivo index.html
3. Agregar index.html al área de preparación y luego al área de confirmaciones
4. Situarse sobre la rama master y ejecuta el comando **git merge testing**
5. Si hay conflictos, solucionalos
6. En caso de conflicto luego de la resolución, agregar index.html al área de preparación y luego al de confirmación.

Reorganización con git rebase

$ git rebase staging



Ejemplo git rebase

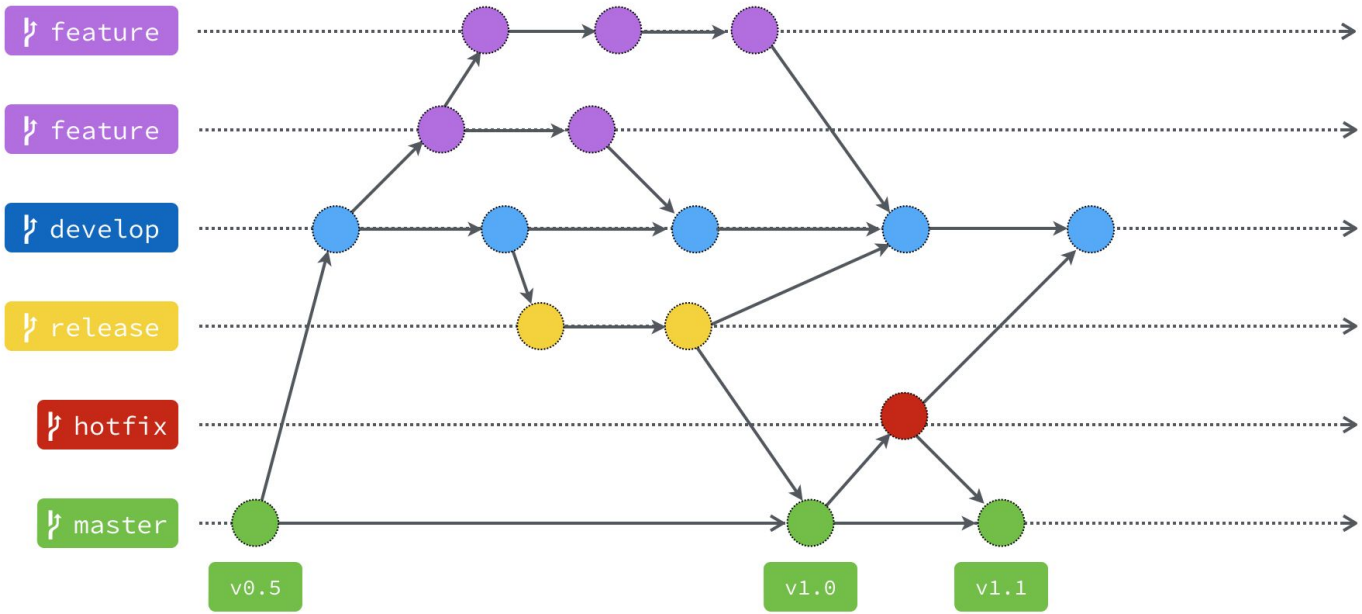
```
$ git rebase testing
First, rewinding head to replay your work on top of it...
Applying: se agrego etiqueta h2
Using index info to reconstruct a base tree...
M       index.html
Falling back to patching base and 3-way merge...
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
error: Failed to merge in the changes.
Patch failed at 0001 se agrego etiqueta h2
hint: Use 'git am --show-current-patch' to see the failed patch

Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort".
```

Práctica git rebase y resolución de conflictos

Dentro de **mi-proyecto** realiza las siguientes acciones:

1. Situarse en la rama testing
2. Haz cambios sobre el archivo index.html
3. Agregar index.html al área de preparación y luego al área de confirmaciones
4. Situarse en la rama **master**, haz algunos cambios y luego prepara y confirma los cambios.
5. Ejecuta el comando **git rebase testing**
6. Si hay conflictos, solucionalos
7. En caso de conflicto luego de la resolución, prepara y confirma los cambios

Flujo de trabajos ramificados o GitFlow



Ramas de largo recorrido

- Master
- Release
- Develop
- Unification (rama que contiene las fusiones de varios features)

Ramas puntuales

- Hotfix
- Features
  - feature
  - refactor
  - fix

Ejemplos:

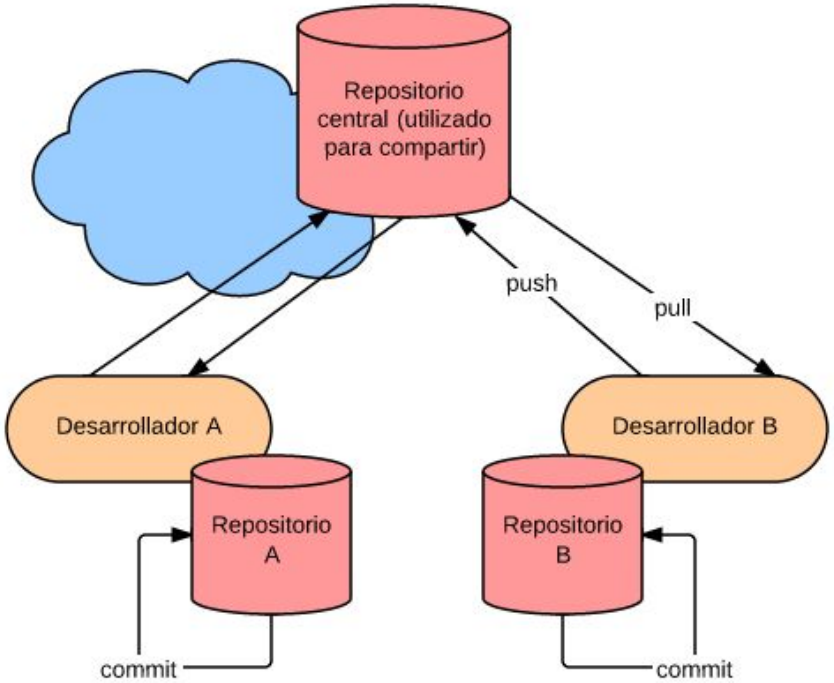
fix/loginUser

feat/registerUser

ref/updateUserInfo

Repositorio remotos

- Son versiones de tu proyecto que se encuentran alojados en Internet.
- Tienen permisos de: lectura o lectura-escritura
- Operaciones: push y pull
- Ejemplos de servicios para alojar repositorios : Github, Gitlab, Bitbucket, etc



Repositorios remotos - Gestión

La gestión de un repositorio remoto significa:

- Añadir repositorios nuevos
- Eliminar aquellos que ya no son válidos
- Gestionar ramas remotas e indicar si están bajo seguimiento o no
- Subir cambios (push) o actualizar repos local (pull)
- y muchas más cosas

Repositorios remotos - Listado con git remote

```
$ git clone git@gitlab.com:rollingcodeschool/fullstack-modulo1-git.git
Cloning into 'fullstack-modulo1-git'...
remote: Enumerating objects: 17, done.
remote: Counting objects: 100% (17/17), done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 17 (delta 3), reused 0 (delta 0)
Receiving objects: 100% (17/17), done.
Resolving deltas: 100% (3/3), done.

$ cd fullstack-modulo1-git/
$ git remote
origin
```

Repositorios remotos - Añadir

$ git remote add origin url_repos_remoto