



4.2 – Repositorios y modelos de desarrollo

Tema 1 – Git



Universidad
Rey Juan Carlos

Micael Gallego
micael.gallego@urjc.es
@micael_gallego

Francisco Gortázar
francisco.gortazar@urjc.es
@fgortazar



Desarrollo colaborativo con Git y GitHub

- **Introducción: Git y GitHub**
- Git: trabajando en local
- Git y GitHub: trabajando en equipo

Introducción

- Los programadores desarrollan código de forma **colaborativa**
- Las herramientas **populares** para compartir ficheros **no son adecuadas** para el código fuente
- Los **sistemas de control de versiones** (*Version control system, VCS*) son herramientas para desarrollo colaborativo



Dropbox

Introducción

- Tienen muchos nombres
 - **Sistema de control de versiones (VCS)**
 - *Version Control System*
 - **Gestor de código fuente (SCM)**
 - *Source Code Manager*
 - **Repositorios de código**
 - *Code repository*

Introducción

- Hay muchos diferentes



¿Por qué Git?

- Software libre
- Muy popular
- Múltiples herramientas para trabajar con él
- **Muy rápido** en realizar las operaciones
- Posibilidad de trabajar offline y luego sincronizar (**distribuido**)
- Funcionalidades que facilitan el **trabajo colaborativo**



Introducción

- **Git** es uno de los más utilizados en la actualidad
- Desarrollado por **Linus Torvals** en 2005 para el desarrollo del **Kernel de Linux**

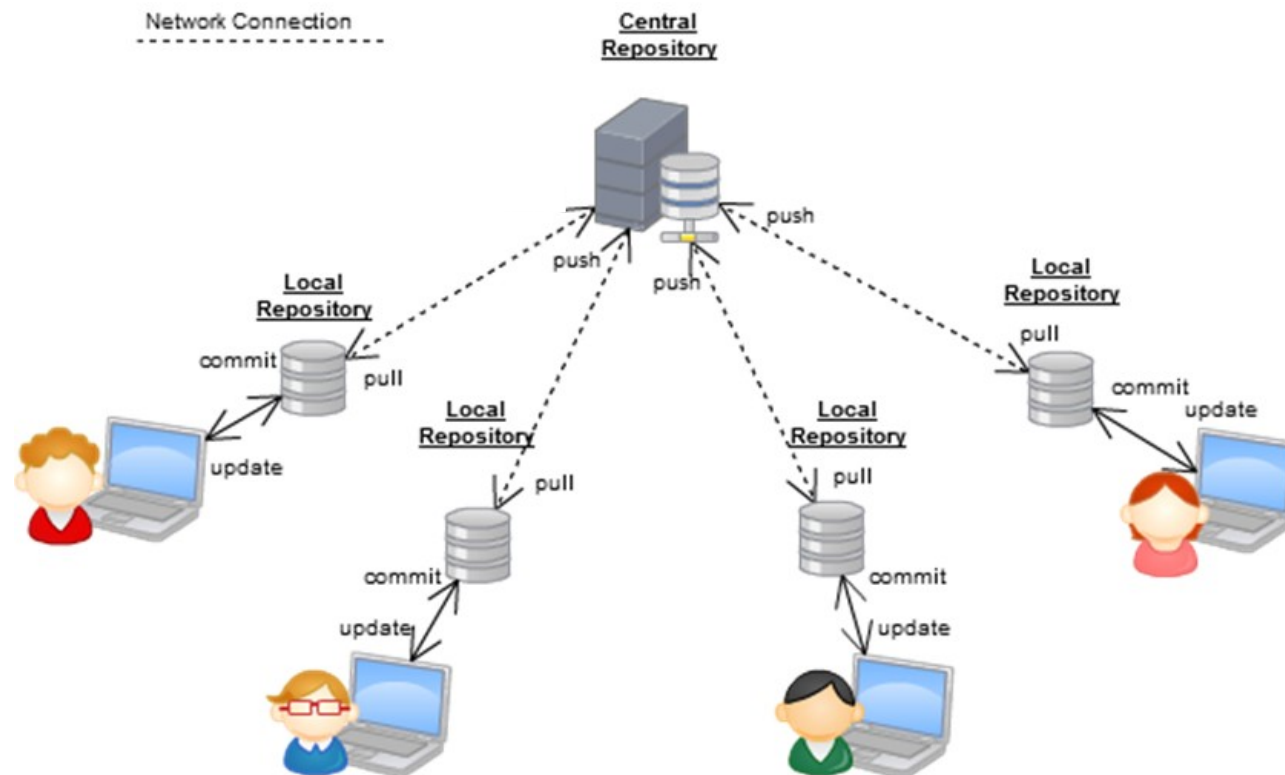
<https://git-scm.com/>

<https://en.wikipedia.org/wiki/Git>



Introducción

- Git es un SCM distribuido
- Cada desarrollador tiene un **repositorio en local** que se sincroniza con el **repositorio compartido**



Introducción

- Clientes



Tortoise Git

<https://tortoisegit.org>

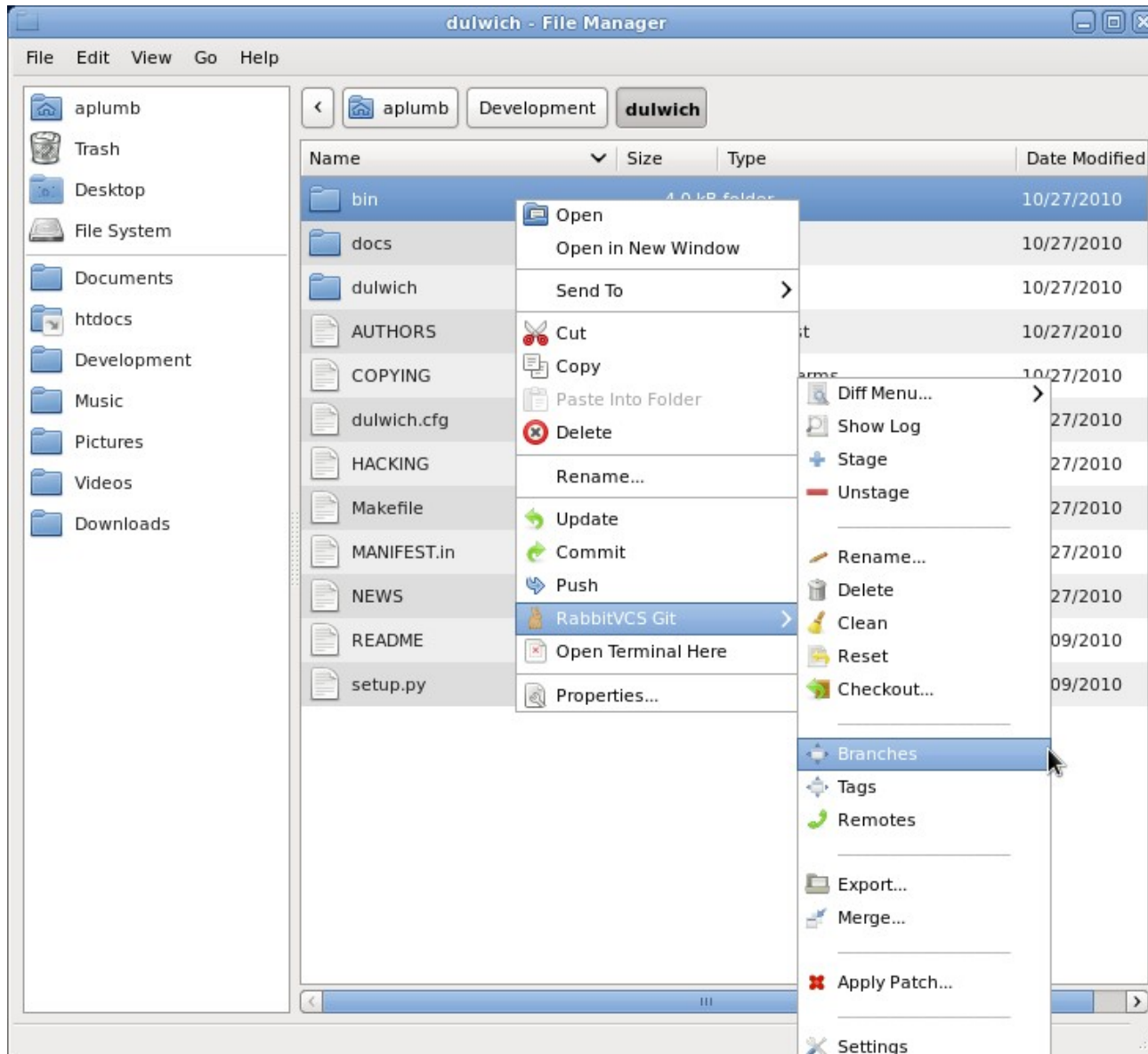


RabbitVCS

<http://wiki.rabbitvcs.org/wiki/install/ubuntu>

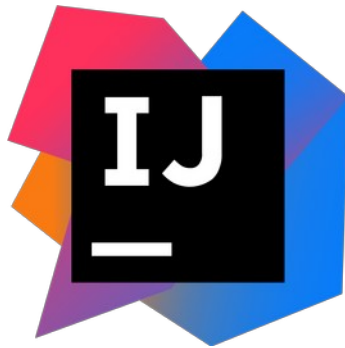
Integrado en el explorador de ficheros

Introducción



Introducción

- Clientes



Visual Studio Code



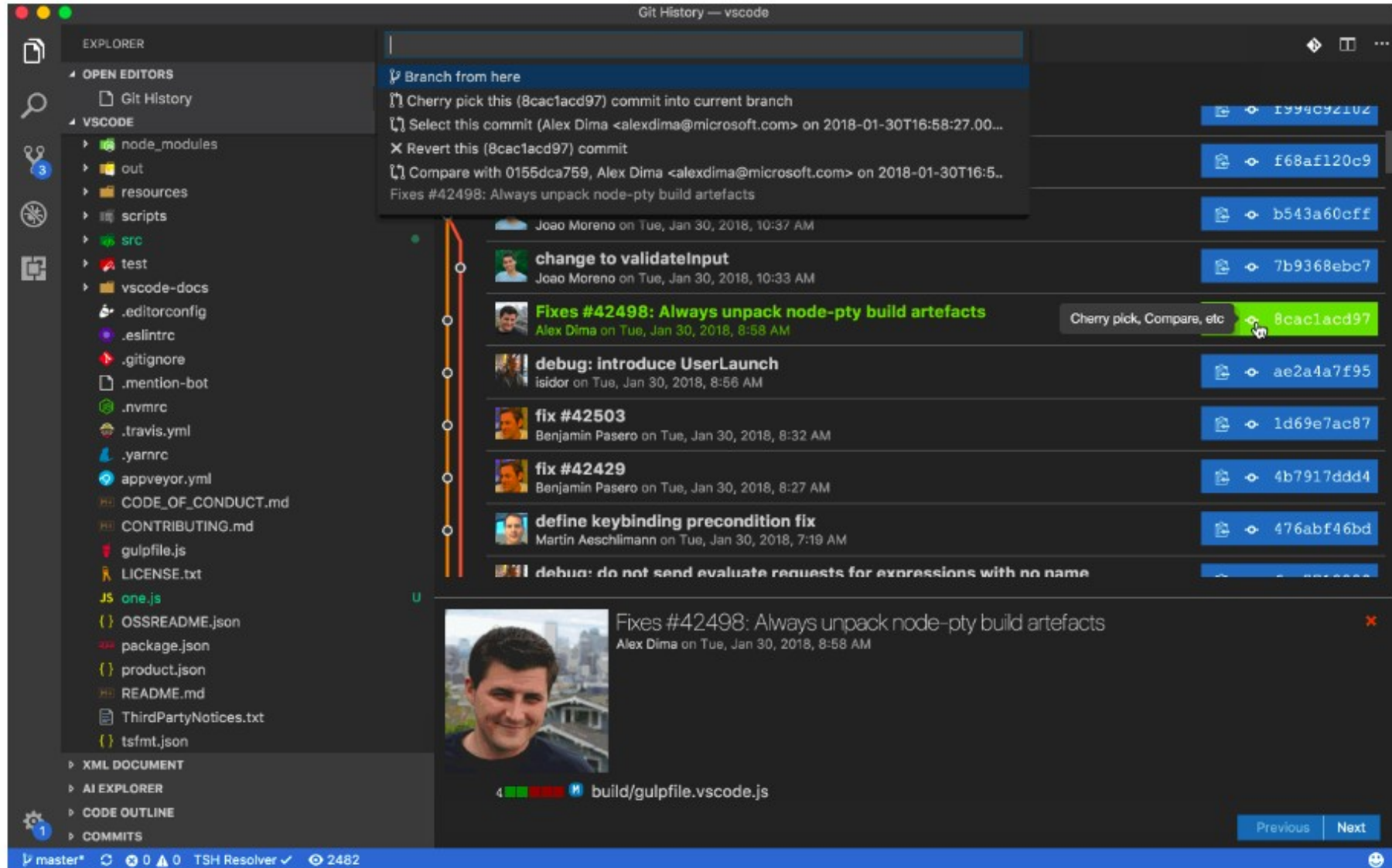
NetBeans



eclipse

Integrado en los editores / IDEs

Introducción



Introducción

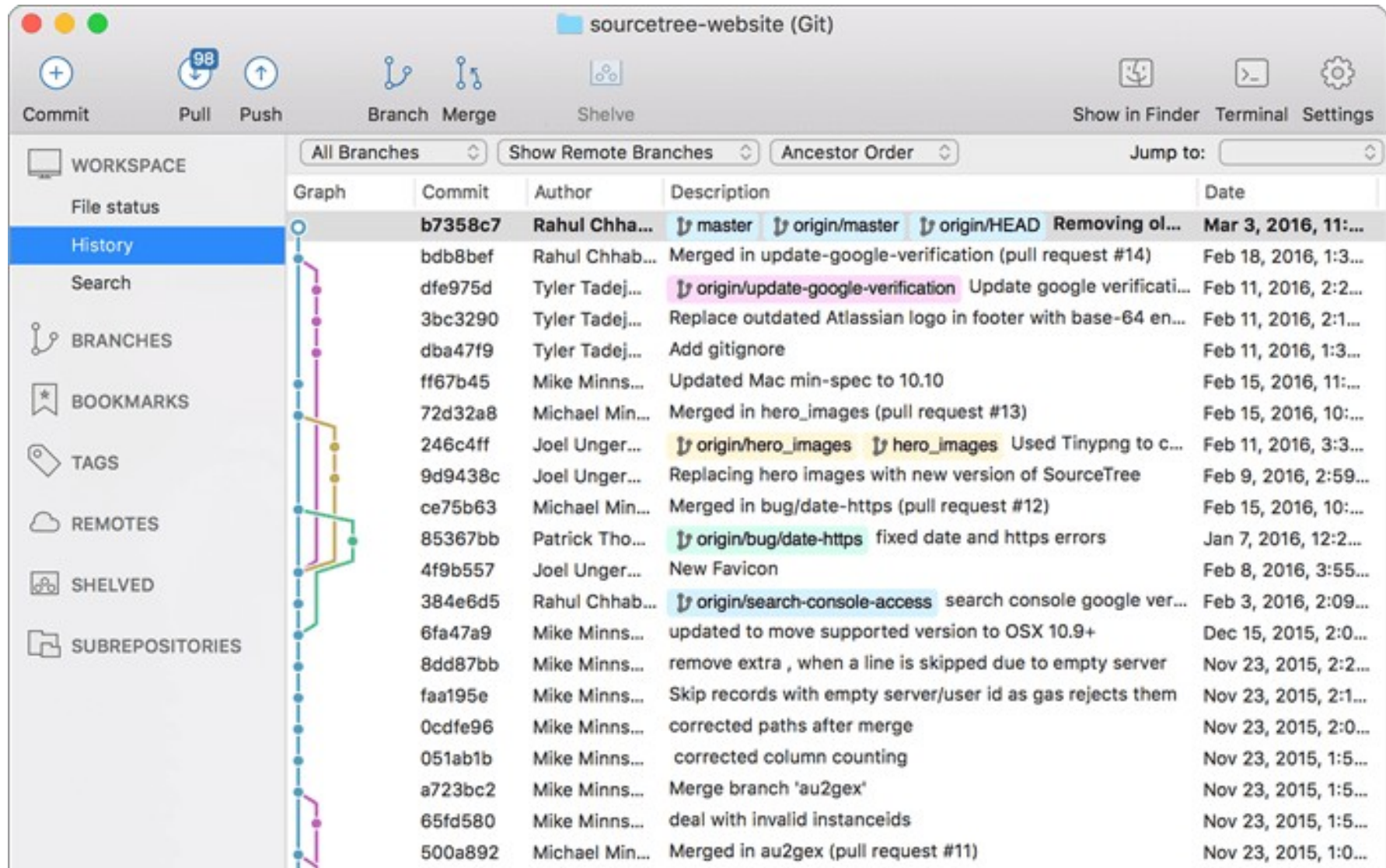
- Clientes



<https://boostlog.io/@nixus89896/top-10-git-gui-clients-5b3336b244deba0054047685>

Clientes gráficos

Introducción



sourcetree-website (Git)

Commit Pull Push Branch Merge Shelf Show in Finder Terminal Settings

WORKSPACE
File status
History
Search

BRANCHES

BOOKMARKS

TAGS

REMOTES

SHELVED

SUBREPOSITORIES

All Branches Show Remote Branches Ancestor Order Jump to:

| Graph | Commit | Author | Description | Date |
|-------|----------------|----------------------|---|----------------------------|
| | b7358c7 | Rahul Chha... | master origin/master origin/HEAD Removing ol... | Mar 3, 2016, 11:... |
| | bdb8bef | Rahul Chhab... | Merged in update-google-verification (pull request #14) | Feb 18, 2016, 1:3... |
| | dfe975d | Tyler Tadej... | origin/update-google-verification Update google verificati... | Feb 11, 2016, 2:2... |
| | 3bc3290 | Tyler Tadej... | Replace outdated Atlassian logo in footer with base-64 en... | Feb 11, 2016, 2:1... |
| | dba47f9 | Tyler Tadej... | Add gitignore | Feb 11, 2016, 1:3... |
| | ff67b45 | Mike Minns... | Updated Mac min-spec to 10.10 | Feb 15, 2016, 11:... |
| | 72d32a8 | Michael Min... | Merged in hero_images (pull request #13) | Feb 15, 2016, 10:... |
| | 246c4ff | Joel Unger... | origin/hero_images hero_images Used Tinypng to c... | Feb 11, 2016, 3:3... |
| | 9d9438c | Joel Unger... | Replacing hero images with new version of SourceTree | Feb 9, 2016, 2:59... |
| | ce75b63 | Michael Min... | Merged in bug/date-https (pull request #12) | Feb 15, 2016, 10:... |
| | 85367bb | Patrick Tho... | origin/bug/date-https fixed date and https errors | Jan 7, 2016, 12:2... |
| | 4f9b557 | Joel Unger... | New Favicon | Feb 8, 2016, 3:55... |
| | 384e6d5 | Rahul Chhab... | origin/search-console-access search console google ver... | Feb 3, 2016, 2:09... |
| | 6fa47a9 | Mike Minns... | updated to move supported version to OSX 10.9+ | Dec 15, 2015, 2:0... |
| | 8dd87bb | Mike Minns... | remove extra , when a line is skipped due to empty server | Nov 23, 2015, 2:2... |
| | faa195e | Mike Minns... | Skip records with empty server/user id as gas rejects them | Nov 23, 2015, 2:1... |
| | 0cdf96 | Mike Minns... | corrected paths after merge | Nov 23, 2015, 2:0... |
| | 051ab1b | Mike Minns... | corrected column counting | Nov 23, 2015, 1:5... |
| | a723bc2 | Mike Minns... | Merge branch 'au2gex' | Nov 23, 2015, 1:5... |
| | 65fd580 | Mike Minns... | deal with invalid instanceids | Nov 23, 2015, 1:5... |
| | 500a892 | Michael Min... | Merged in au2gex (pull request #11) | Nov 23, 2015, 1:0... |

Introducción

- **Clientes**
 - Existen **múltiples aplicaciones** para trabajar con los repositorios locales (clientes)



Git en Linux / Mac



Git for Windows

Línea de comandos

Introducción

```

express — less — 72x21

mac express: git log --oneline --graph
* 11a77a3 Fix inner numeric indices incorrectly altering parent req.params
* ee90042 Fix infinite loop condition using mergeParams: true
* 97b2d70 4.13.2
* a4fcd91 deps: update example dependencies
* a559ca2 build: istanbul@0.3.17
* c398a99 deps: type-is@~1.6.6
* 1cea9ce deps: accepts@~1.2.12
* e33c503 deps: path-to-regexp@0.1.7
* 9848645 Merge tag '3.21.2'
| \
| * cb59086 3.21.2
| * ce087e5 build: marked@0.3.5
| * 93dd15c deps: connect@2.30.2
| * b53feaa deps: vary@~1.0.1
| * d51d1ea build: ejs@2.3.3
| * fc95112 build: should@7.0.2
* | 659c0b1 deps: update example dependencies
* | 09c80bf deps: array-flatten@1.1.1
* | de7ffca tests: add test for matching route after error

```


Introducción

- **Clientes**
 - Los **clientes gráficos** es posible que no ofrezcan todas las funcionalidades de Git
 - Es buena idea conocer cómo realizar las operaciones por **línea de comandos**
 - Se pueden **combinar** sin problemas



Introducción

- Servidores



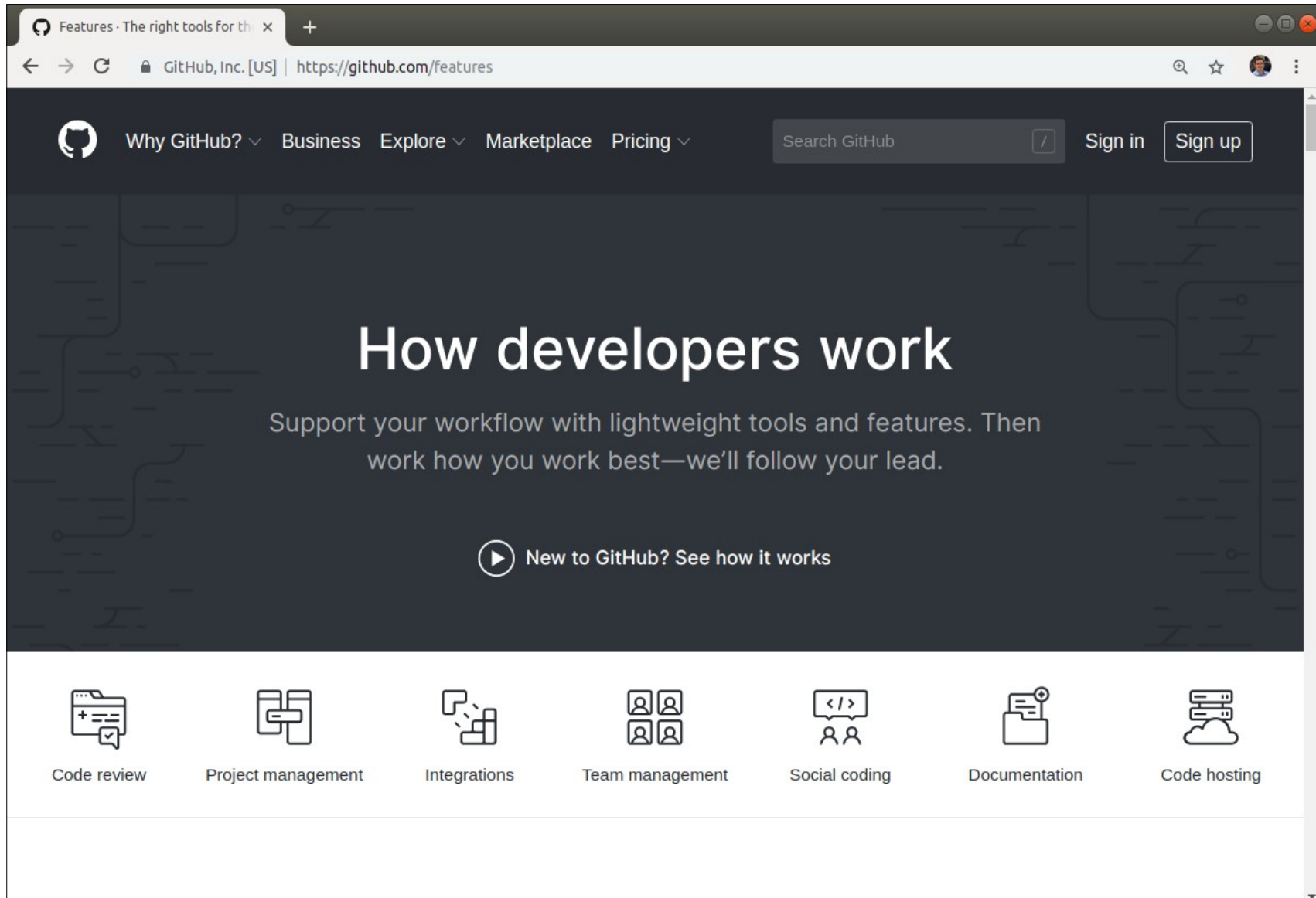
Introducción

- **GitHub** es un servicio para desarrolladores que proporciona repositorios git en sus servidores
 - **Gratuito** para repositorios públicos con software libre
 - **De pago** para repositorios privados
 - **Funcionalidades adicionales:**
 - Bug reports, wiki, releases, Pull Requests...



<https://github.com/>

Introducción



Conceptos básicos

- **¿Cómo se crea un repositorio?**
 - Se puede crear localmente y luego publicarlo en GitHub
 - Se puede crear en GitHub y descargarlo (clonar)
- **¿Qué es un repositorio?**
 - Es una carpeta en disco (con una subcarpeta llamada .git)
- **¿Qué se guarda un repositorio?**
 - Todas las versiones de todos los ficheros
 - Quién modificó cada línea de cada fichero (ficheros de texto)
 - Para ahorrar espacio se comprimen los ficheros del repositorio

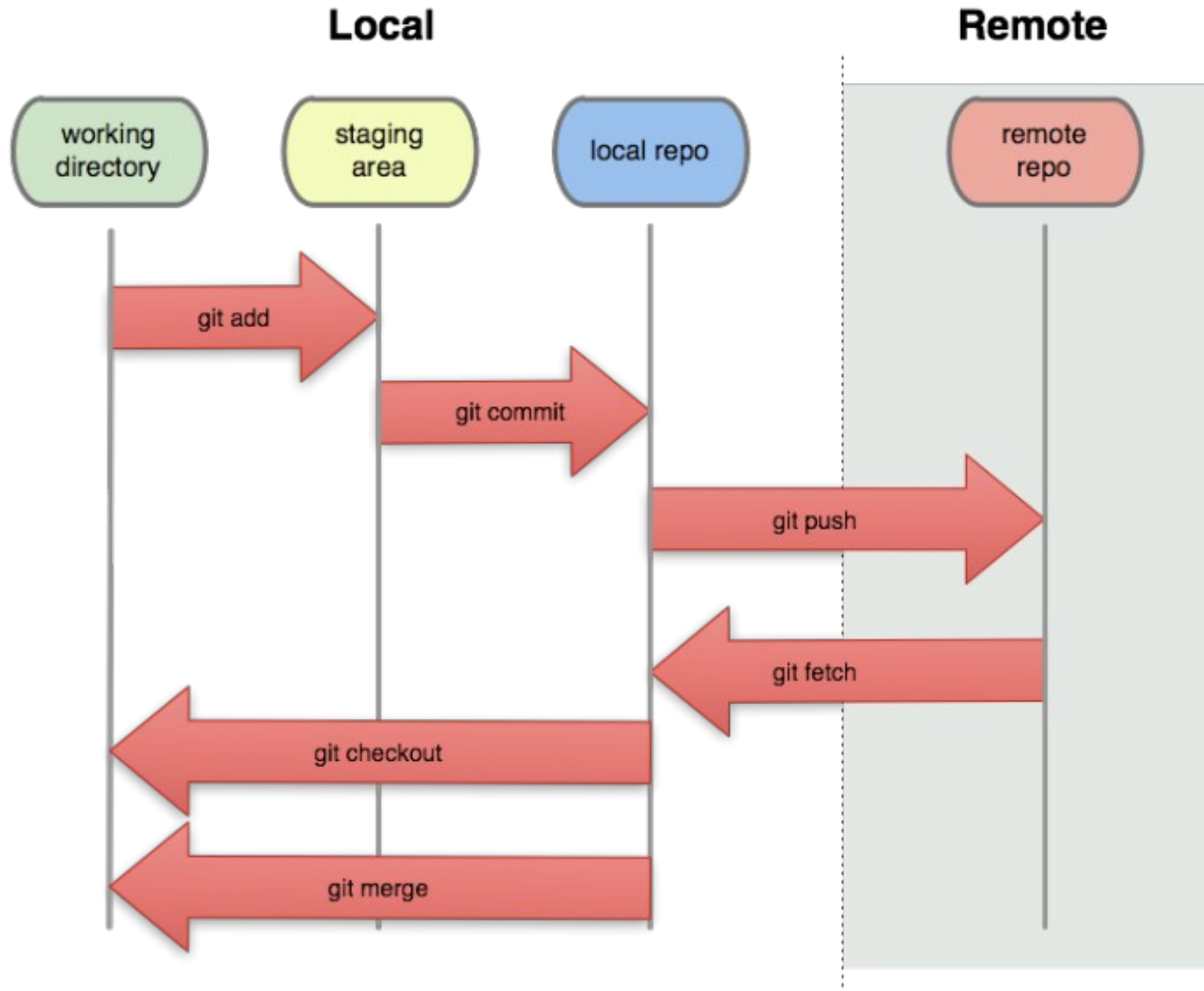
Conceptos básicos

- ¿Cómo se trabaja con un repositorio?
 - **Paso 1)** El desarrollador crea y edita los ficheros de forma habitual con editores o cualquier programa (son **ficheros normales** en disco)
 - **Paso 2)** Cuando se completa una funcionalidad, se indican los ficheros nuevos o los que han cambiado (**add**) y se crea la nueva mini-versión (**commit**)
 - **Paso 3)** De vez en cuando se suben las mini-versiones que tenemos en local al servidor compartido (**push**)
 - **Paso 4)** De vez en cuando se descargan los cambios del servidor compartido que han realizado otros programadores al repositorio local (**fetch** o **pull**)

Conceptos básicos

- Zonas en las que está un fichero
 - Carpeta raíz del proyecto (**working directory**)
 - Es el espacio donde el desarrollador crea y modifica los ficheros
 - Si se va a trabajar con un repositorio existente, los ficheros se obtienen de una mini-versión (**commit**) alojada en el directorio git (usando el comando **checkout**)
 - El directorio **.git** (**local repo**)
 - Contiene todas las versiones de los ficheros del repositorio y los metadatos
 - Es lo que se copia cuando se clona un repositorio
 - Es el contenido de la carpeta **.git**
 - **Staging** (no es una carpeta real, es una lista de ficheros)
 - Listado de ficheros que se añadirán en la próxima mini-versión (**commit**)

Conceptos básicos



Conceptos básicos

- Los 4 estados (de los ficheros)

Untracked

El fichero se acaba de crear y todavía no se ha añadido al repositorio

Staged

El fichero ha sido modificado y se ha marcado para formar parte de la próxima mini-versión (commit)

Modified

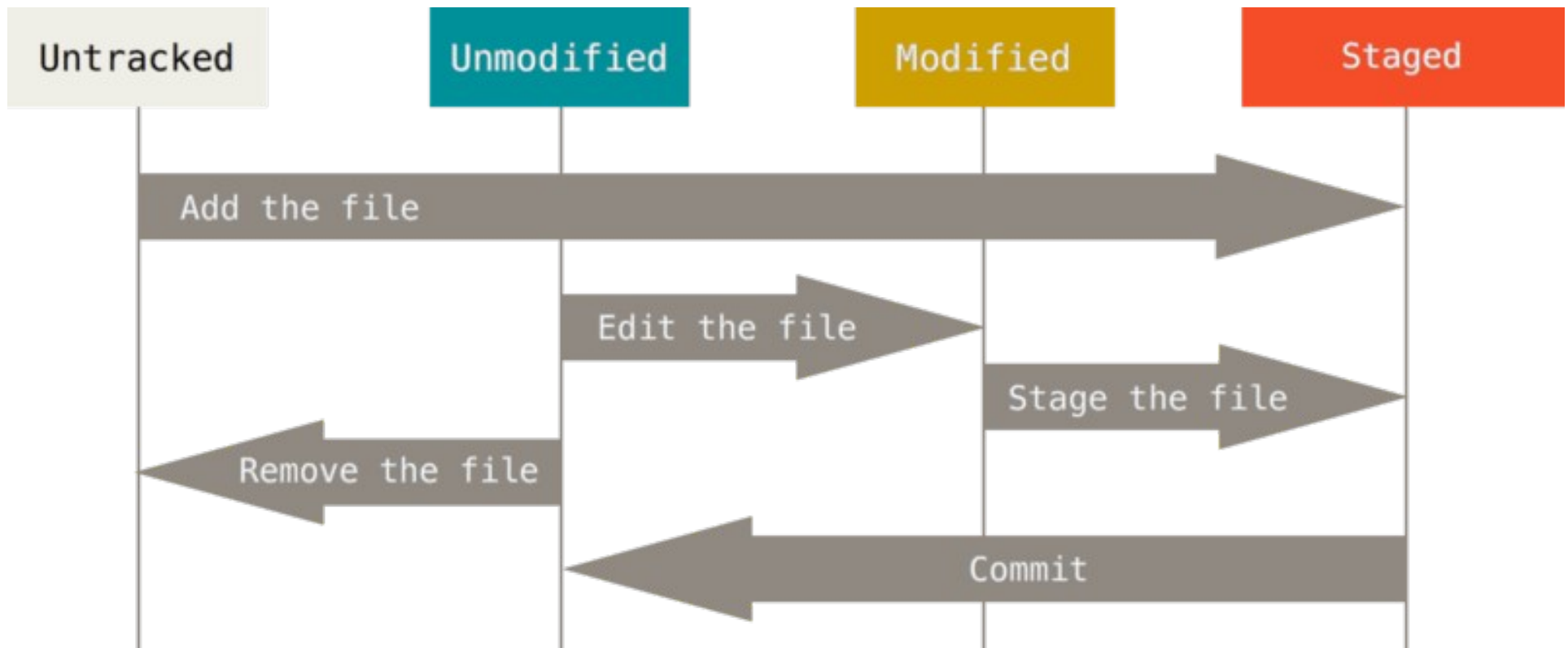
El fichero ha sido modificado desde la última vez que se añadió al repositorio

Unmodified

El fichero forma parte de una mini-versión (commit) del repositorio y no se ha modificado

Conceptos básicos

- Los 4 estados (de los ficheros)



Conceptos básicos

- **Commits (mini-versiones)**

- Cuando se quiere registrar una mini-versión del proyecto se hace un **commit**
- El commit tiene un **mensaje** indicando qué cambios se han introducido en el código en ese commit (ideal para analizar el histórico del proyecto)
- Cada commit se identifica con un **número muy largo**

```
commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    removed unnecessary test
```

- Como es tan largo, se suelen usar únicamente los primeros 7 caracteres (en este caso 085bb3b)

Conceptos básicos

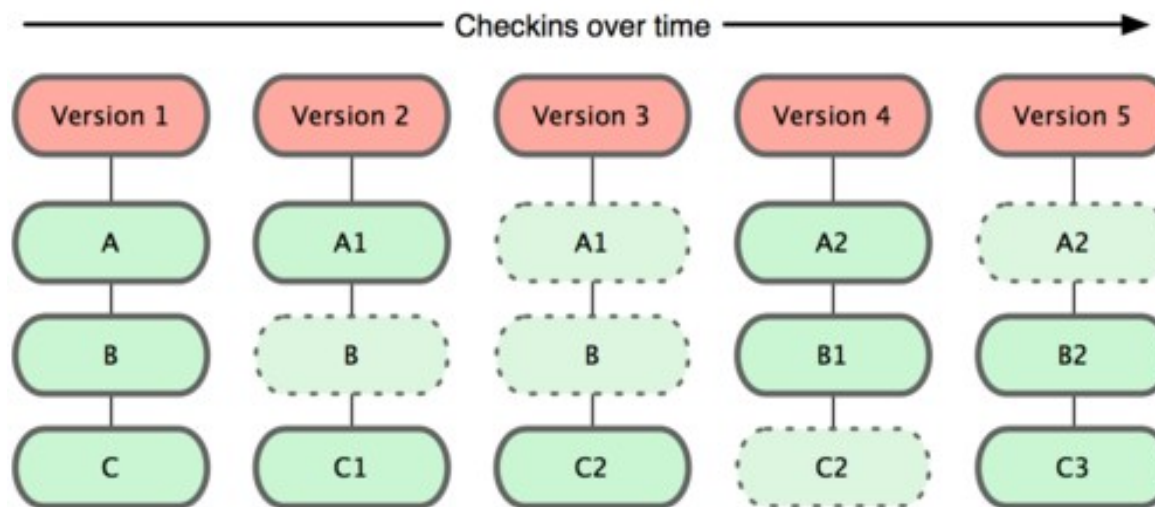
- **Commits (mini-versiones)**
 - ¿Por qué usar ese número “tan raro”?
 - En otros sistemas se usa un número incremental (1, 2, 3 ...) pero eso tiene problemas cuando dos desarrolladores cambian el mismo fichero en su propia máquina (antes de sincronizar con el servidor)
 - ¿Cómo se obtiene ese número?
 - Usando la función Hash SHA1 con la información del commit (autor, mensaje, commit anterior, etc.)
 - Se suele usar la expresión “el hash del commit” en vez de “número del commit” o “id del commit”

<https://gist.github.com/masak/2415865>

Conceptos básicos

- **Snapshots**

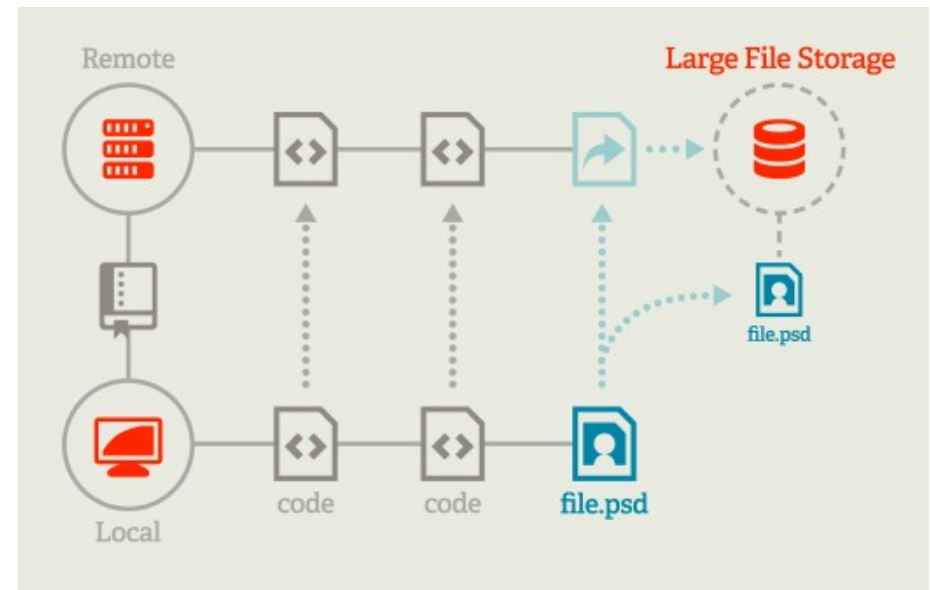
- En general, por cada commit no se guardan las diferencias de los ficheros que han cambiado. Se guardan los ficheros completos (para que sea rápido obtener un commit concreto)



Conceptos básicos

- **Snapshots**

- Un repositorio git no está diseñado para guardar grandes ficheros binarios que cambian frecuentemente porque se guardan todas las versiones
- Existen extensiones de git para estos casos



Conceptos básicos

- **La identidad (del desarrollador)**
 - Cada commit lleva asociado la identidad del desarrollador que lo hizo
 - La identidad está formada por nombre y correo electrónico
 - Es importante tener bien configurados estos datos
 - Dos commits realizados desde máquinas diferentes pueden “parecer” que pertenecen a usuarios diferentes
 - Es posible que haya problemas de acceso al repositorio compartido en el servidor

Conceptos básicos

- **La identidad (del desarrollador)**
 - Los datos de identidad se suelen configurar a nivel de usuario en el sistema operativo
 - **Fichero .gitconfig en la carpeta HOME del usuario**
 - /home/<user>/.gitconfig
 - C:\Users\<user>\.gitconfig
 - **El comando “config” permite configurar el fichero**
 - git config --global user.name “patxigortazar”
 - git config --global user.email “patxi.gortazar@gmail.com”
 - cat \$HOME/.gitconfig

Conceptos básicos

- **La identidad (del desarrollador)**
 - ¿Qué pasa si en un determinado repositorio se necesita otra identidad?
 - Omitiendo el parámetro `--global` los cambios se aplican exclusivamente al repositorio actual
 - Tenemos que estar posicionados en la carpeta de un repositorio git
 - `git config user.name "mi otra identidad"`
 - `git config user.email "alterego@acme.com"`
 - `git config [--global|--local] --list`

Conceptos básicos

- **Integridad**
 - Como el **hash** del commit se obtiene directamente de los datos del commit (no es un número aleatorio generado) se puede detectar si existe algún tipo de corrupción en los datos del repositorio
 - Para detectar posible corrupción de datos en los ficheros también se usa el **Hash SHA1**

Desarrollo colaborativo con Git y GitHub

- Introducción: Git y GitHub
- **Git: trabajando en local**
- Git y GitHub: trabajando en equipo

Instalación del cliente git

- Instalación rápida
 - Windows:
 - <https://git-scm.com/download/win>
 - Mantener valores por defecto
 - Linux (diferentes sabores)
 - <https://git-scm.com/download/linux>
 - MacOS
 - <https://git-scm.com/download/mac>

Instalación del cliente git

- Comprobación rápida
 - Windows:
 - Busca la aplicación Git bash y ábrela
 - Es una línea de comandos tipo unix
 - Linux y MacOS: abrir la línea de comandos
 - Comprobar que git está instalado y en el PATH:

```
$ git --version  
git version 2.17.1
```

Crear el repositorio en local

```
~$ mkdir repo
```

```
~$ cd repo
```

```
~/repo$ git init
```

```
Initialized empty Git repository in /home/mica/repo/.git/
```

```
~/repo$ git status
```

```
On branch master
```

```
No commits yet
```

```
nothing to commit (create/copy files and use "git add" to track)
```

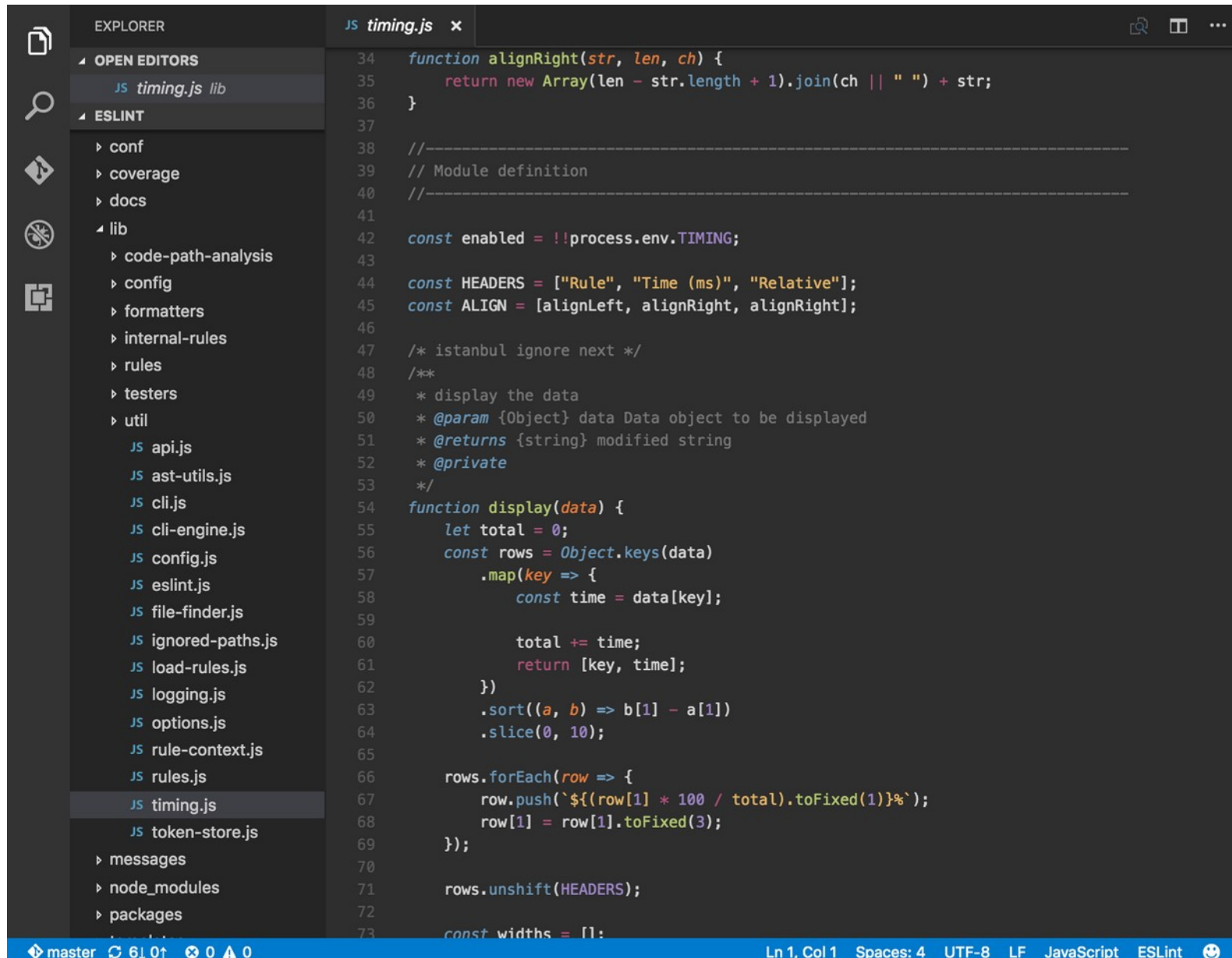
```
~/repo$ git log
```

```
fatal: your current branch 'master' does not have any commits yet
```

Trabajando en el repositorio

- **Instalar Visual Studio Code**
 - Editor ligero
 - Con plugins para multitud de lenguajes de programación
 - Tiene plugin de git básico instalado
 - Se pueden instalar más plugins adicionales
 - Nos permitirá modificar los ficheros sin tener que hacerlo desde la línea de comandos
 - Versiones para Win, Mac y Linux
 - <https://code.visualstudio.com/>

Trabajando en el repositorio



```

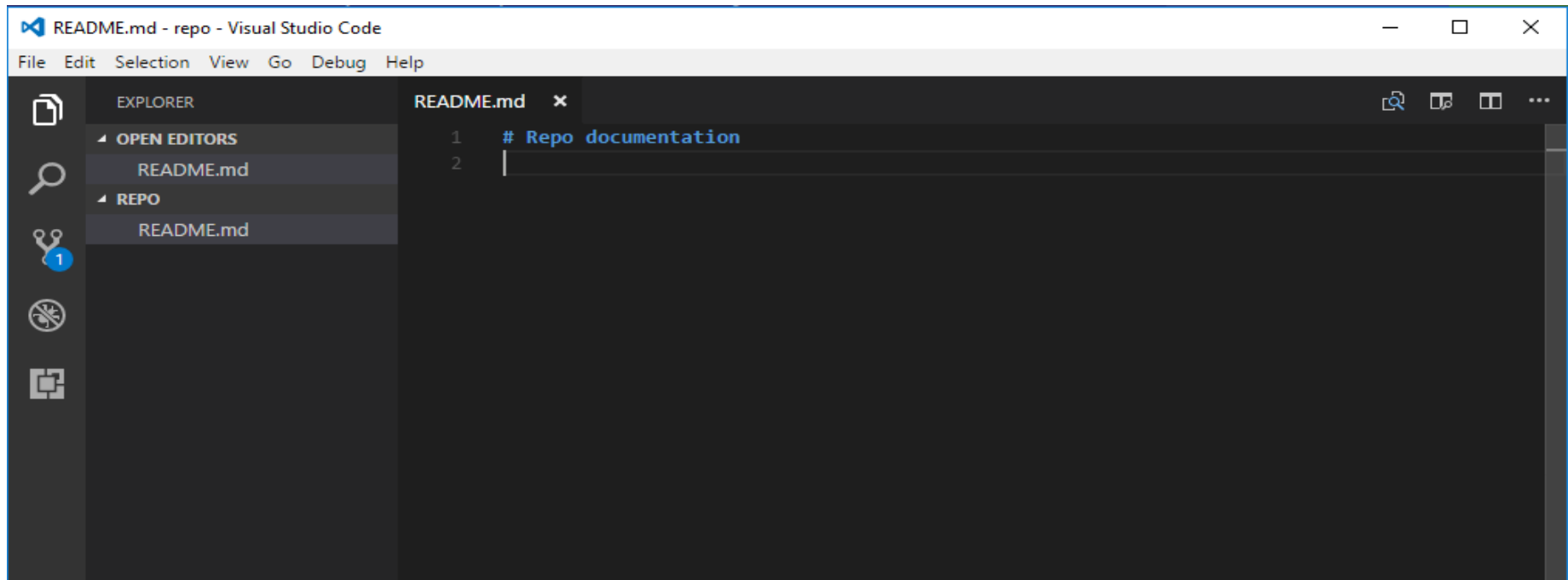
34 function alignRight(str, len, ch) {
35     return new Array(len - str.length + 1).join(ch || " ") + str;
36 }
37
38 //-----
39 // Module definition
40 //-----
41
42 const enabled = !!process.env.TIMING;
43
44 const HEADERS = ["Rule", "Time (ms)", "Relative"];
45 const ALIGN = [alignLeft, alignRight, alignRight];
46
47 /* istanbul ignore next */
48 /**
49  * display the data
50  * @param {Object} data Data object to be displayed
51  * @returns {string} modified string
52  * @private
53  */
54 function display(data) {
55     let total = 0;
56     const rows = Object.keys(data)
57         .map(key => {
58             const time = data[key];
59
60             total += time;
61             return [key, time];
62         })
63     .sort((a, b) => b[1] - a[1])
64     .slice(0, 10);
65
66     rows.forEach(row => {
67         row.push(`${(row[1] * 100 / total).toFixed(1)}%`);
68         row[1] = row[1].toFixed(3);
69     });
70
71     rows.unshift(HEADERS);
72
73     const widths = [];

```


repositorio

- Crear un fichero README.md en la raiz del repositorio

```
$ code .
```



Crear un fichero y añadirlo al repositorio

```
~/repo$ 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)
```

Instrucciones sobre
cómo proceder a
continuación



Crear un fichero y añadirlo al repositorio

```
~/repo$ git add README.md
```

```
~/repo$ git status
```

```
On branch master
```

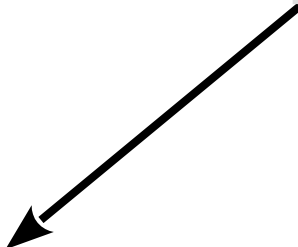
```
No commits yet
```

```
Changes to be committed:
```

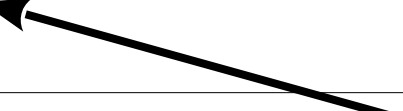
```
(use "git rm --cached <file>..." to unstage)
```

```
new file:   README.md
```

Instrucciones sobre
cómo revertir los
cambios



Qué se va a
añadir al repo



Crear un fichero y añadirlo al repositorio

```
~/repo$ git commit -m "Add README"
[master (root-commit) 85fe787] Add README
1 file changed, 1 insertion(+)
create mode 100644 README.md
```

Working area limpia,
todos los cambios se
incluyeron en el repo

```
~/repo$ git status
On branch master
nothing to commit, working tree clean
```

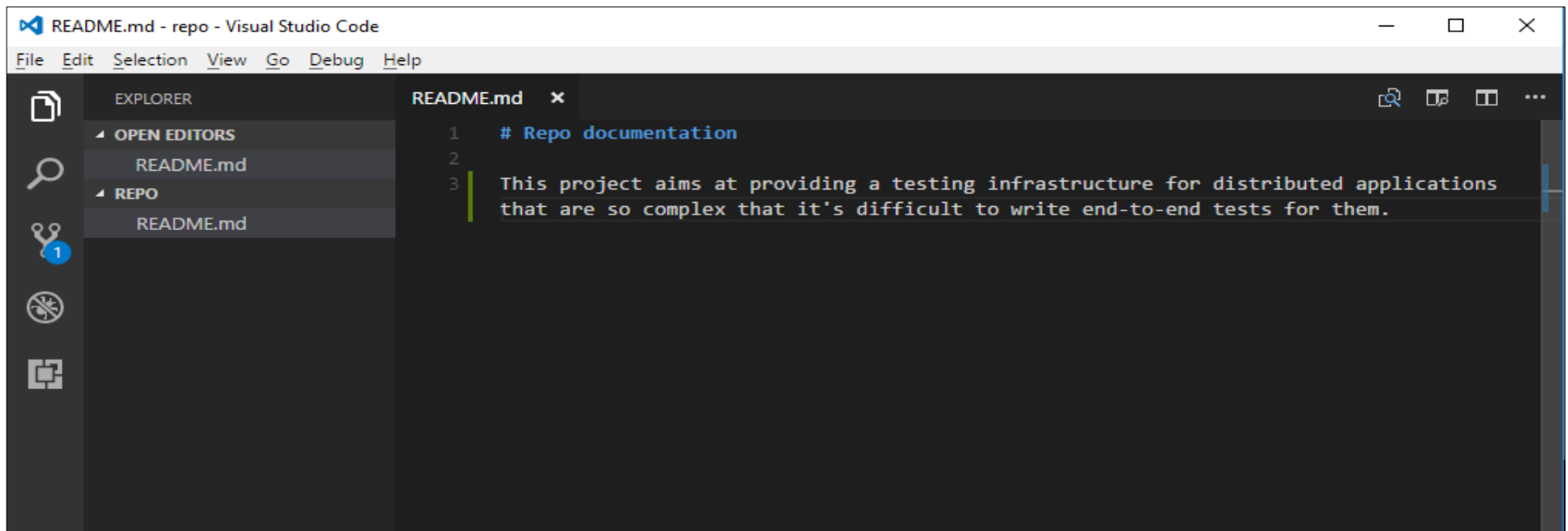
```
~/repo$ git log
commit 85fe787f225a1df8ea03803d8abf0cc581465372 (HEAD -> master)
Author: patxigortazar <patxi.gortazar@urjc.es>
Date: Mon May 15 01:00:34 2017 -0700
```

Lista de commits

Add README

Subir una actualización a un fichero

- Editar el fichero README.md añadiendo contenido



The screenshot shows the Visual Studio Code interface with the README.md file open in the editor. The Explorer sidebar on the left shows the file structure with 'README.md' listed under 'OPEN EDITORS' and 'REPO'. The editor window displays the following content:

```

1  # Repo documentation
2
3  This project aims at providing a testing infrastructure for distributed applications
   that are so complex that it's difficult to write end-to-end tests for them.
  
```

Subir una actualización a un fichero

```
~/repo$ git status
```

```
On branch master
```

```
Changes not staged for commit:
```

```
(use "git add <file>..." to update what will be committed)
```

```
(use "git checkout -- <file>..." to discard changes in working directory)
```

```
    modified:   README.md
```

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

Subir una actualización a un fichero

- **Ejercicio 1**
 - Comitear el fichero README.md
 - Verificar que tenemos dos commits en el repo

Trabajo con el repositorio

- **Ejercicio 2**
 - Añadir un fichero LICENSE y editar el fichero README.md
 - Comitear el fichero LICENSE primero
 - Comitear el fichero README.md después

Trabajo con el repositorio

- **Consulta de la historia del proyecto**
 - El comando “log” muestra por defecto bastante información de los commits
 - Existen opciones que permiten un histórico más compacto

```
$ git log --pretty=format:"%h - %an, %ar : %s"
ca82a6d - Scott Chacon, 6 years ago : changed the version number
085bb3b - Scott Chacon, 6 years ago : removed unnecessary test
a11bef0 - Scott Chacon, 6 years ago : first commit
```

Buenas prácticas en los commits

- **Commits pequeños**
 - Cada cambio debería tener su propio commit
 - Así es más fácil identificar qué ha cambiado por el mensaje del commit
 - Es más sencillo de gestionar el commit para resolver conflictos, revisar su contenido por otros desarrolladores, etc.
 - En cada commit el proyecto tiene que compilar y pasar los tests (si tiene)

Buenas prácticas en los commits

- Buenas prácticas en mensajes de commits

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

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

Buenas prácticas en los commits

- **Buenas prácticas en mensajes de commits**
 1. Separa el asunto del cuerpo con una línea en blanco
 2. Limita el asunto a 50 caracteres
 3. Inicia el asunto con mayúsculas
 4. No finalices el asunto con un punto
 5. Usa el verbo imperativo (no el pasado) en el asunto
 6. No uses líneas mayores de 72 caracteres
 7. Usa el cuerpo para explicar qué y por qué, no cómo (está en el código)

<https://chris.beams.io/posts/git-commit/>

<https://github.com/erlang/otp/wiki/writing-good-commit-messages>

Buenas prácticas en los commits

- Buenas prácticas en mensajes de commits

```
commit eb0b56b19017ab5c16c745e6da39c53126924ed6
Author: Pieter Wuille <pieter.wuille@gmail.com>
Date:   Fri Aug 1 22:57:55 2014 +0200
```

Simplify serialize.h's exception handling

Remove the 'state' and 'exceptmask' from serialize.h's stream implementations, as well as related methods.

As exceptmask always included 'failbit', and setstate was always called with bits = failbit, all it did was immediately raise an exception. Get rid of those variables, and replace the setstate with direct exception throwing (which also removes some dead code).

As a result, good() is never reached after a failure (there are only 2 calls, one of which is in tests), and can just be replaced by !eof().

fail(), clear(n) and exceptions() are just never called. Delete them.

Buenas prácticas en los commits

- **Buenas prácticas en mensajes de commits**
 - El cuerpo del mensaje debería responder a:
 - ¿Por qué el cambio es necesario?
 - ¿Cómo resuelve el problema?
 - ¿Qué efectos colaterales (*side effects*) tiene el cambio?
 - Si el proyecto usa un sistema para gestión de issues (features o bugs reports) incluye la URL del issue que motiva este cambio

<https://robots.thoughtbot.com/5-useful-tips-for-a-better-commit-message>

Buenas prácticas en los commits

- Buenas prácticas en mensajes de commits
 - Algunos proyectos define un prefijo para identificar el tipo de commit

Formato

```
<type>(<scope>): <subject>
<BLANK LINE>
<body>
<BLANK LINE>
<footer>
```

Type

- **feat:** A new feature
- **fix:** A bug fix
- **docs:** Documentation only changes
- **style:** Changes that do not affect the meaning of the code (white-space, formatting..)
- **refactor:** A code change that neither fixes a bug or adds a feature
- **perf:** A code change that improves performance
- **test:** Adding missing tests
- **chore:** Changes to the build process or auxiliary tools and libraries

Scope

- The scope could be anything specifying place or module of the commit change

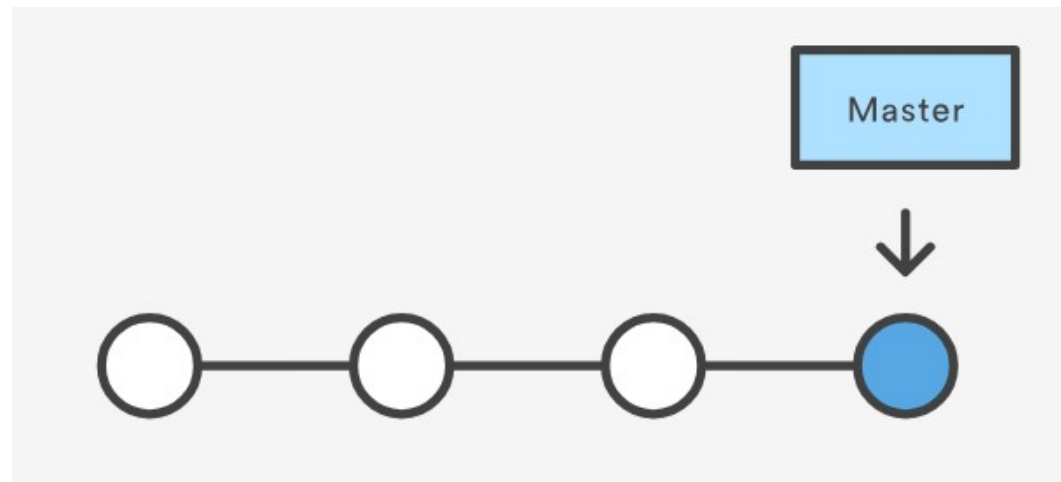
Footer

- The footer should contain any information about Breaking Changes and is also the place to reference GitHub issues that this commit Closes.

<http://karma-runner.github.io/3.0/dev/git-commit-msg.html>
<https://gist.github.com/brianclements/841ea7bffdbo1346392c>

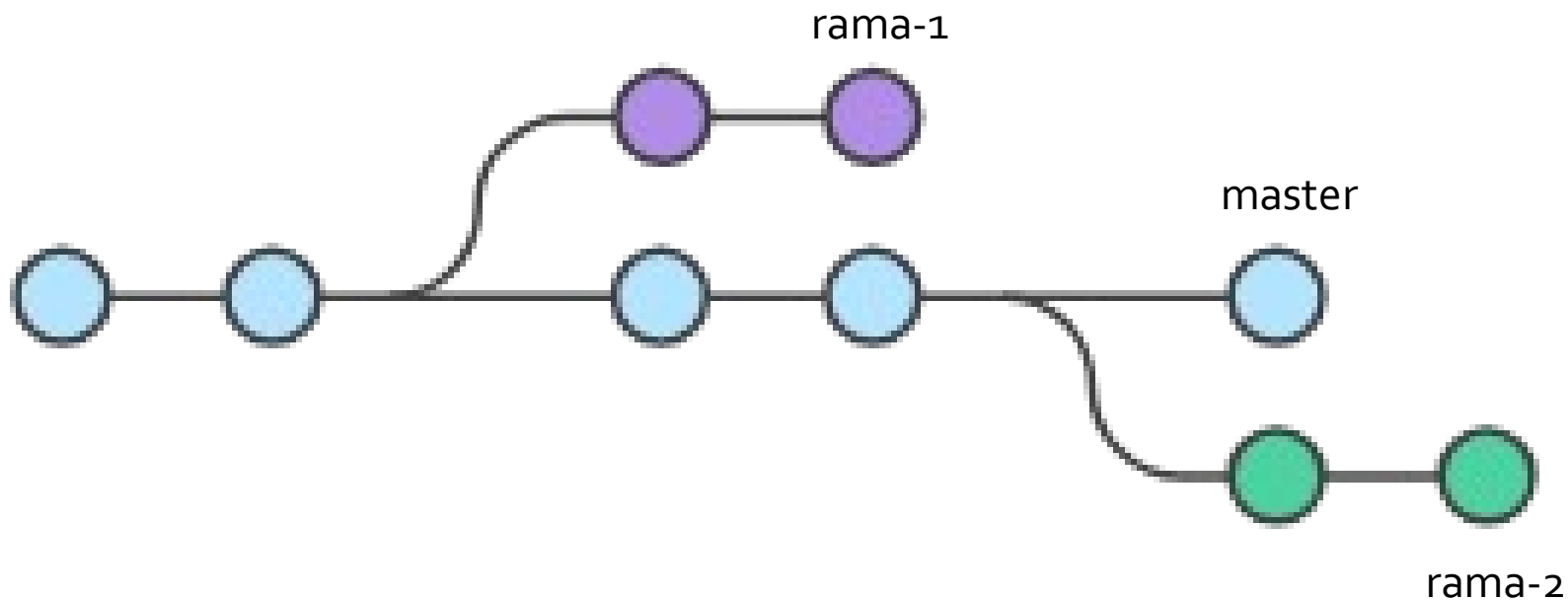
Ramas (*branches*)

- Hemos visto como los commits se añadían al repositorio
- En realidad se añaden a una zona llamada **rama master**
- Cuando el código de la rama master está estable, se publicará una **nueva release**



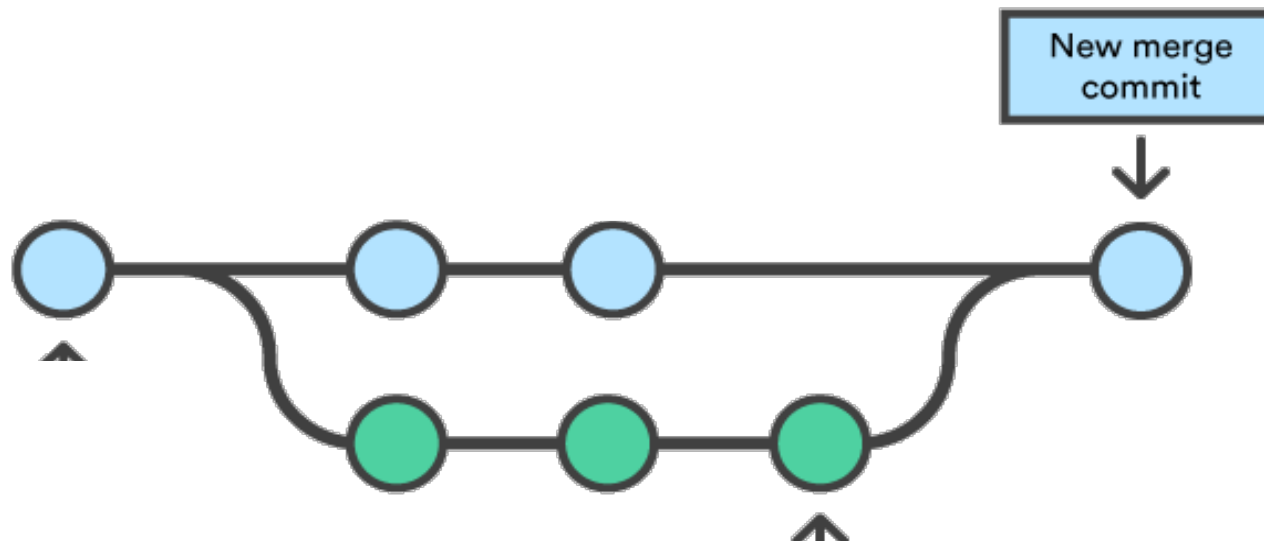
Ramas (*branches*)

- Creación de una rama
 - Una rama se puede crear partiendo de cualquier commit del repositorio
 - Es como hacer una copia de un documento para poder trabajar en dos “líneas de trabajo” en paralelo



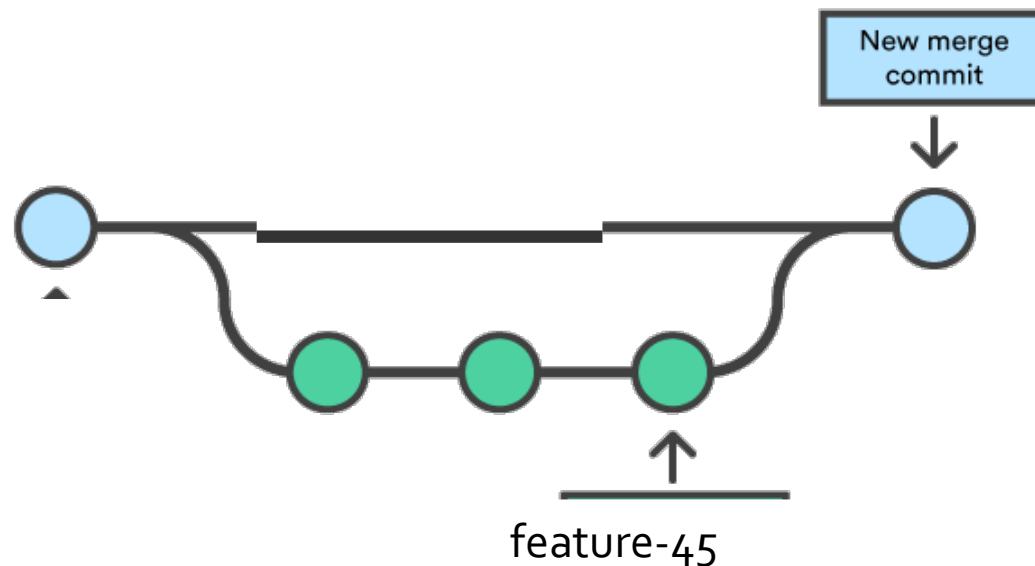
Ramas (*branches*)

- Mezcla de una rama
 - En cualquier momento se puede mezclar el código de dos ramas diferentes
 - Se suele utilizar para integrar el trabajo en una rama en la rama master



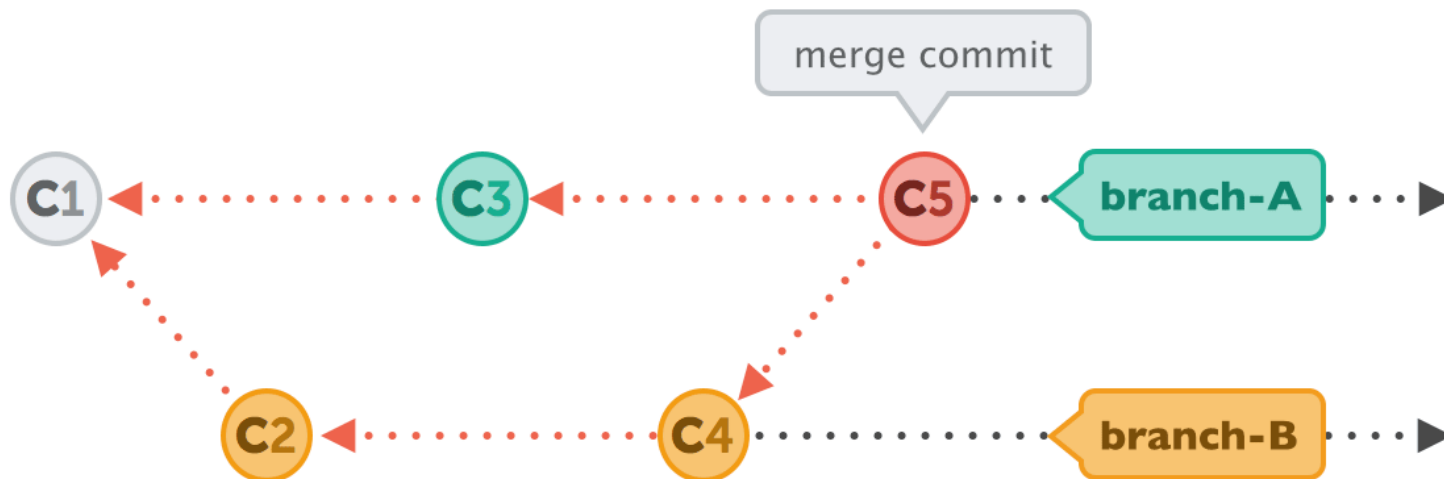
Ramas (*branches*)

- Cada funcionalidad en una rama (*branch-per-feature*)
 - Existe una estrategia de desarrollo en la que cada nueva funcionalidad se implementa en una nueva rama
 - Cuando la funcionalidad se termina, se mezcla con master
 - En master no se hacen commits, sólo mezclas (merge)



Ramas (*branches*)

- Ramas y commits
 - Los commits de mezcla (merge) se basan en dos commits anteriores
 - El resto de commits, únicamente se basan en un commit anterior que se conoce como *parent* commit



Ramas (*branches*)

- Comandos para trabajar con ramas

- Crear una rama

```
$ git branch feature1
```

- Ver las ramas de un repositorio

```
$ git branch --list
```

- Borrar una rama

```
$ git branch -d <rama>
```

- Trabajar en una rama

```
$ git checkout <rama>
```

- Crear una nueva rama y trabajar en ella

```
$ git checkout -b <rama>
```

Ramas (*branches*)

- **Ejercicio 3**
 - Añadir una rama llamada “documentation”
 - En dicha rama:
 - Crear una carpeta docs
 - Movernos a la carpeta docs
 - Añadir los siguientes ficheros (no importa el contenido):
 - index.html
 - styles.css
 - Comitear

Ramas (*branches*)

- Ejercicio 4
 - Cambiar a master (hacer un checkout sin -b)
 - `git checkout master`
 - En master:
 - Modificar el fichero README.md
 - Comitear

Ramas (*branches*)

- Estado actual del repositorio

```
$ git log --graph --all --oneline
* 1450e35 (HEAD -> master) Add new section to README.md
| * 09d9e74 (documentation) Add documentation
|/
* 52046cc Improve readme again
* 95cb647 Add LICENSE
* e815802 Improve readme
* 85fe787 Add README
```


Ramas (*branches*)

- **Referencias**

- HEAD, master, documentation son referencias a commits
- Existe una referencia por cada rama
- HEAD es una referencia que indica el commit del repositorio que ha cargado el directorio de trabajo
- HEAD apunta al commit que será parent commit del nuevo commit que hagamos
- Es posible que HEAD apunte a un commit del pasado (para consulta), pero si hacemos cambios debemos crear una rama nueva en ese commit

```
$ git checkout <commit>
```

Ramas (*branches*)

- Volvamos a la rama documentación y añadamos un par de commits más

```
$ git checkout documentation
$ echo "A new file" > file1.txt
$ git status
$ git add file1.txt
$ git commit -m "Add file1"
$ echo "Another file" > file2.txt
$ git add file2.txt
$ git commit -m "Add file2"
```

Ramas (*branches*)

- Veamos nuestro árbol

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

```
* 39aec4f (HEAD -> documentation) Add file2
* 4605118 Add file1
* 09d9e74 Add documentation
```

```
* 1450e35 (master) Add new section to README.md
```

```
|/
```

```
* 52046cc Improve readme again
* 95cb647 Add LICENSE
* e815802 Improve readme
* 85fe787 Add README
```

Ramas (*branches*)

- Vamos a incorporar la documentación a la rama master
- Hay tres estrategias diferentes
 - Merge
 - Merge con squash
 - Fast-forward merge

Ramas (*branches*)

- **Merge**
 - Se crea un nuevo commit basado en el último commit de master y en el último commit de documentation
 - La rama master ahora incluye los cambios de la rama documentation
 - Nos posicionamos en la rama en la que queremos continuar trabajando (master) y solicitamos el merge de la otra rama (documentation)

Ramas (*branches*)

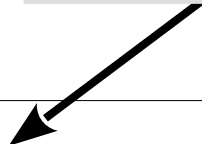
- Merge

```
$ git checkout master
$ git merge documentation
Merge made by the 'recursive' strategy.
 docs/index.html | 4 ++++
 docs/styles.css | 1 +
 file1.txt       | 1 +
 file2.yxy       | 1 +
4 files changed, 7 insertions(+)
create mode 100644 docs/index.html
create mode 100644 docs/styles.css
create mode 100644 file1.txt
create mode 100644 file2.yxy
```

Ramas (*branches*)

- Merge

Commit con dos padres
72c38ef y 52046cc



```
$ git log --graph --all --oneline
* 72c38ef (HEAD -> master) Merge branch 'documentation'
|\
| * 39aec4f (documentation) Add file2
| * 4605118 Add file1
| * 09d9e74 Add documentation
* | 1450e35 Add new section to README.md
|/
* 52046cc Improve readme again
* 95cb647 Add LICENSE
* e815802 Improve readme
* 85fe787 Add README
```

Ramas (*branches*)

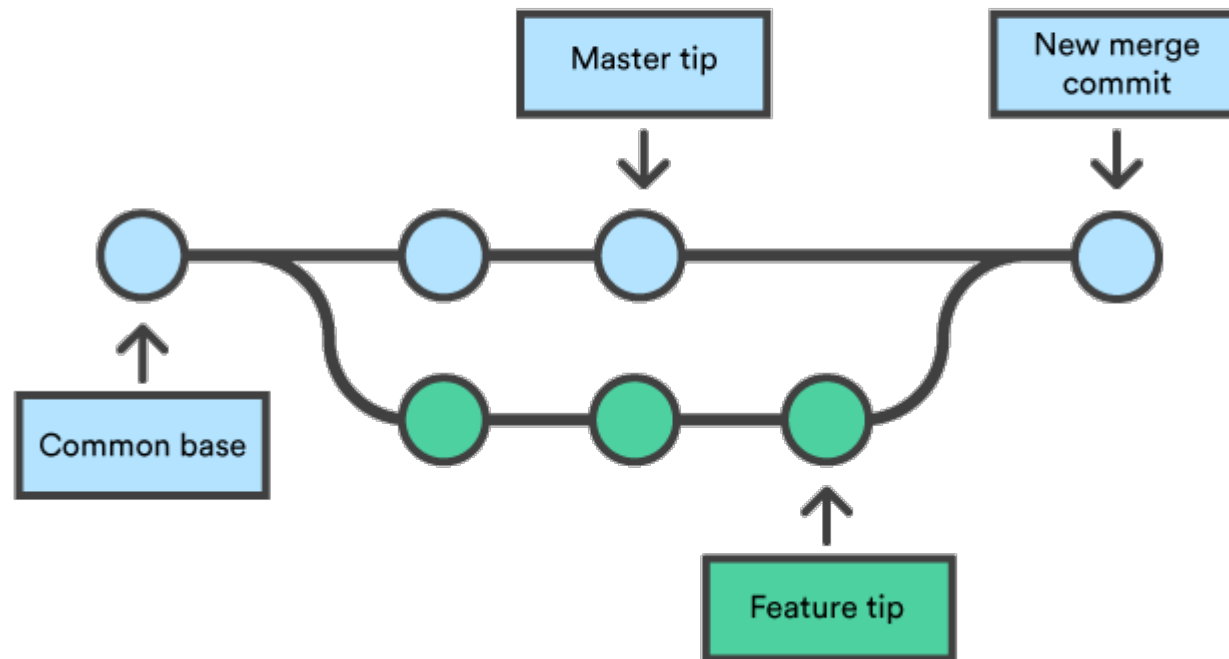
- Merge

Todos los commits de ambas ramas forman parte ahora de la historia de master

```
$ git log --oneline
72c38ef (HEAD -> master) Merge branch 'documentation'
39aec4f (documentation) Add file2
4605118 Add file1
1450e35 Add new section to README.md
09d9e74 Add documentation
52046cc Improve readme again
95cb647 Add LICENSE
e815802 Improve readme
85fe787 Add README
```


Ramas (*branches*)

- Merge



Ramas (*branches*)

- **Merge**

- Cuando existen commits de merge no hay una evolución “lineal” del código
- Hay bifurcaciones (en los commits de merge) que en un punto del pasado nacen de un commit común
- Hay desarrolladores que prefieren mantener una historia lineal, para ello existen dos estrategias:
 - Merge con squash
 - Fast-Forward merge (con o sin rebase)

Ramas (*branches*)

- Merge con squash

- Todos los commits de una rama se “fusionan” en un único commit sobre la rama destino

```
$ git merge <rama> --squash
```

- El comando deja en el directorio de trabajo la fusión de los commits
- No hace el commit para que el usuario pueda revisar y editar y haga él mismo el commit (con el mensaje que quiera)
- La historia es lineal, pero se pierde trazabilidad
- El repositorio no vincula las ramas master y documentation

Ramas (*branches*)

- Merge con squash
 - Creamos una nueva rama (ramasquash) y hacemos dos commits en el fichero README.md
 - Add paragraph1
 - Add paragraph2
 - Vamos a master y creamos el fichero README2.md
 - Add README2.md

Ramas (*branches*)

- Merge con squash

```
~/repo$ git log --graph --all --oneline
* 811983f (HEAD -> master) Add README2.md
| * eef095b (ramasquash) Add paragraph2
| * 3894584 Add paragraph1
|/
* c0eacee (addfile) Add nuevo2.txt
* 2c0d8e2 Add nuevo.txt
* b15f345 Merge branch 'documentation'
|\
| * 4c7fab3 (documentation) Add file2
| * 2da7fc5 Add file1
| * 6dd54cc New docs
* | e5d8f72 Add new paragraph to README.md
|/
* c6b5451 Improve documentation
* a8b186e Add license
* b7a1a04 Mi segundo commit
```

Ramas (*branches*)

- Merge con squash
 - Ejecutamos el merge con squash

```
~/repo$ git merge ramasquash --squash  
Squash commit -- not updating HEAD  
Automatic merge went well; stopped before committing as requested
```

Ramas (*branches*)

- Merge con squash
 - Los cambios de la rama ramasquash se aplican al directorio de trabajo

```
~/repo$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to
  unstage)

        modified:   README.md
```

- Los comiteamos a master

```
~/repo$ git add README.md
~/repo$ git commit -m "Update README.md"
```

Ramas (*branches*)

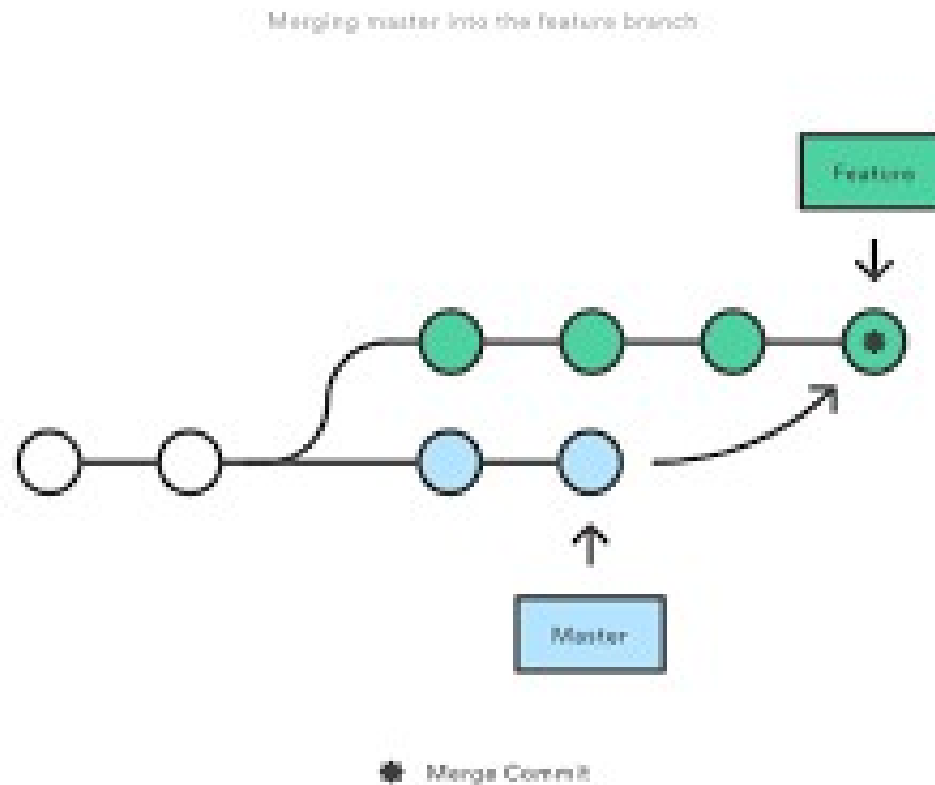
- Merge con squash

```
~/repo$ git log --graph --all --oneline
* 566ec18 (HEAD -> master) Update README.md
* 811983f Add README2.md
| * eef095b (ramasquash) Add paragraph2
| * 3894584 Add paragraph1
|/
* c0eacee (addfile) Add nuevo2.txt
* 2c0d8e2 Add nuevo.txt
* b15f345 Merge branch 'documentation'
|\
| * 4c7fab3 (documentation) Add file2
| * 2da7fc5 Add file1
| * 6dd54cc New docs
* | e5d8f72 Add new paragraph to README.md
|/
* c6b5451 Improve documentation
* a8b186e Add license
```

No hay relación entre la rama master y la rama ramasquash

Ramas (*branches*)

- Merge con squash



Ramas (*branches*)

- **Fast-forward merge**
 - Cuando no ha habido commits en master desde que se creó la otra rama, no es necesario crear un commit de merge
 - Basta con hacer que la rama master apunte al último commit de la rama que se quiere mezclar
 - Cuando se puede evitar el commit de merge se llama **fast-forward**
 - Se puede forzar la creación del commit merge con la opción **--no-ff**

<https://git-scm.com/docs/git-merge>

Ramas (*branches*)

- **Fast-forward merge**
 - Creamos una rama addfile y añadimos dos ficheros

```
$ git checkout addfile
$ echo "New file" > new.txt
$ git add new.txt
$ git commit -m "Add new.txt"
$ echo "New file2" > new2.txt
$ git add new2.txt
$ git commit -m "Add new2.txt"
```

Ramas (*branches*)

- Fast-forward merge


```
$ git log --graph --all --oneline
* c0eacee (HEAD -> addfile) Add new2.txt
* 2c0d8e2 Add new.txt
* b15f345 (master) Merge branch 'documentation'
| \
| * 4c7fab3 (documentation) Add file2
| * 2da7fc5 Add file1
| * 6dd54cc New docs
* | e5d8f72 Add new paragraph to README.md
| /
* c6b5451 Improve documentation
* a8b186e Add license
* b7a1a04 Mi segundo commit
* ed11fcf Add README
```

Ramas (*branches*)

- Fast-forward merge

Se ha detectado que el merge puede hacerse con fast-forward (sin crear commit de merge)

```
$ git checkout master
$ git merge addfile
Updating b15f345..e0eacee
Fast-forward
 new.txt    | 1 +
 new2.txt   | 1 +
 2 files changed, 2 insertions(+)
 create mode 100644 new.txt
 create mode 100644 new2.txt
```



Ramas (*branches*)

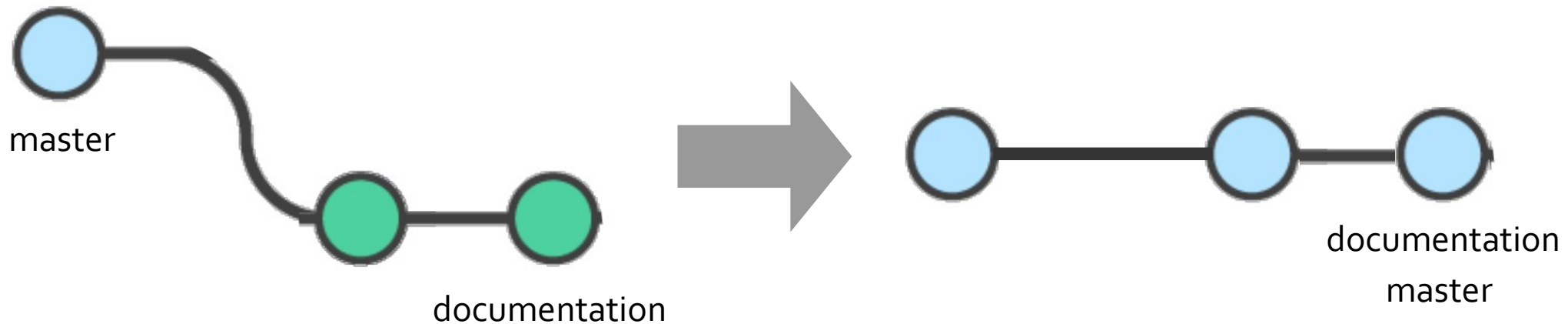
- Fast-forward merge

La rama master apunta al último commit de la rama addfile.
No hay commit de merge

```
$ git log --graph --all --oneline
* c0eacee (HEAD -> master, addfile) Add new2.txt
* 2c0d8e2 Add new.txt
* b15f345 Merge branch 'documentation'
|\
| * 4c7fab3 (documentation) Add file2
| * 2da7fc5 Add file1
| * 6dd54cc New docs
* | e5d8f72 Add new paragraph to README.md
|/
* c6b5451 Improve documentation
* a8b186e Add license
* b7a1a04 Mi segundo commit
* ed11fcf Add README
```

Ramas (*branches*)

- Fast-forward merge



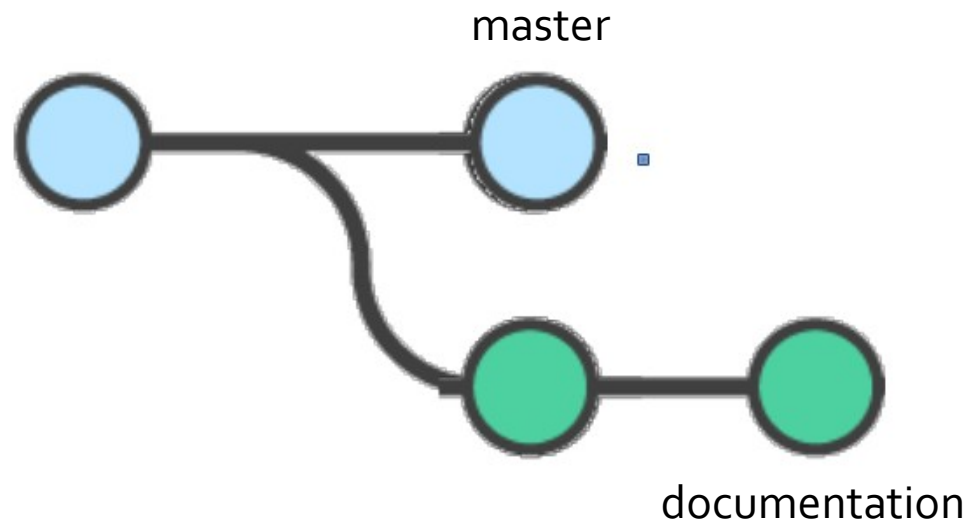
Sólo se cambia la referencia de la rama master

Ramas (*branches*)

- **Fast-forward merge**
 - El resultado obtenido con el **merge con fast-forward** hace que el repositorio siga una evolución lineal sin perder ningún commit (como con squash)
 - Pero no siempre se cumplen las condiciones para que se pueda aplicar
 - Se pueden reorganizar los commits para que se pueda aplicar haciendo un “**rebase**”

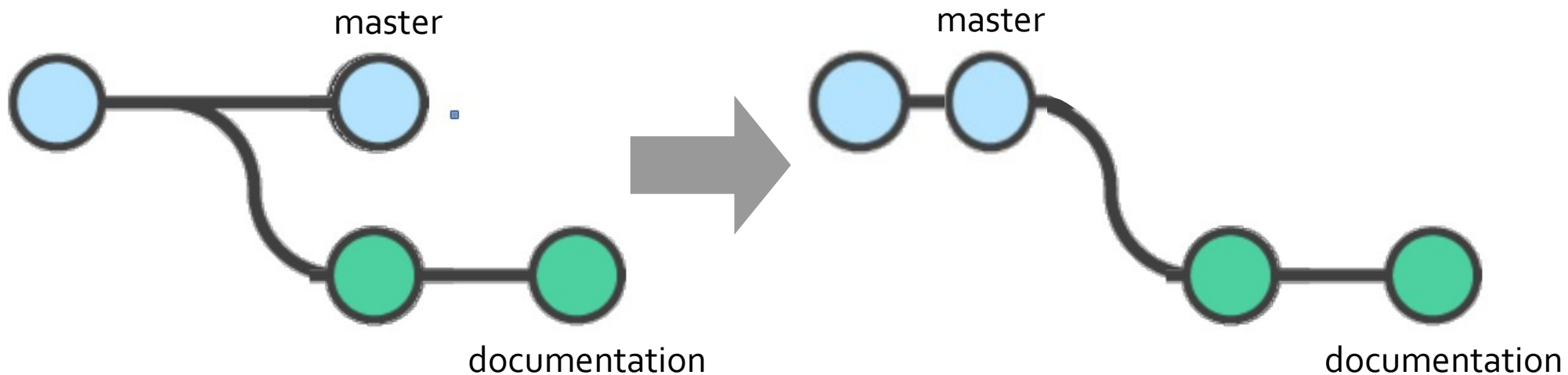
Ramas (*branches*)

- **Fast-forward merge con rebase**
 - ¿Qué ocurre si se han hecho commits en master después de haber creado la rama que se quiere mezclar?
 - No se permite el fast-forward merge



Ramas (*branches*)

- **Fast-forward merge con rebase**
 - El rebase desplaza la nueva rama para que se base en el último commit de la rama master
 - De esa forma se puede aplicar el fast-forward



Ramas (*branches*)

- **Fast-forward merge con rebase**
 - Vamos a crear la rama newdoc con un commit en README.md
 - Vamos a volver a master y hacer un commit en README2.md
 - Vamos a intentar hacer un merge forzando fast-forward de newdoc a master para que nos detecte que no se puede
 - Rebasamos newdoc sobre master
 - Volvemos a intentar el merge con fast-forward

Ramas (*branches*)

- Fast-forward merge con rebase

```
~/repo$ git checkout -b newdoc
Switched to a new branch 'newdoc'
~/repo$ echo "New content" >> README.md
~/repo$ git add README.md
~/repo$ git commit -m "Add new line in README.md"
[newdoc 4b75181] Add new line in README.md
1 file changed, 3 insertions(+), 1 deletion(-)
~/repo$ git checkout master
Switched to branch 'master'
~/repo$ echo "New content" >> README2.md
~/repo$ git add README2.md
~/repo$ git commit -m "Add content to README2.md"
[master 1687778] Add content to README2.md
1 file changed, 1 insertion(+)
```

Ramas (*branches*)

- Fast-forward merge con rebase

```
~/repo$ git log --graph --all --oneline
* 1687778 (HEAD -> master) Add content to README2.md
| * 4b75181 (newdoc) Add new line in README.md
|/
* 566ec18 Update README.md
* 811983f Add README2.md
```

Ramas (*branches*)

- **Fast-forward merge con rebase**
 - Intentamos el merge forzando fast-forward

```
~/repo$ git merge newdoc --ff-only
fatal: Not possible to fast-forward, aborting.
```

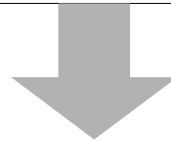
- Rebasamos newdoc sobre master

```
~/repo$ git checkout newdoc
Switched to branch 'newdoc'
~/repo$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: Add new line in README.md
```

Ramas (*branches*)

- **Fast-forward merge con rebase**
- Ahora se cumplen las condiciones para el fast-forward merge

```
~/repo$ git log --graph --all --oneline
* 1687778 (HEAD -> master) Add content to README2.md
| * 4b75181 (newdoc) Add new line in README.md
|/
* 566ec18 Update README.md
* 811983f Add README2.md
```

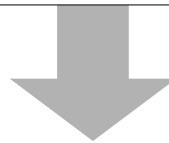


```
~/repo$ git log --graph --all --oneline
* 531b278 (HEAD -> newdoc) Add new line in README.md
* 1687778 (master) Add content to README2.md
* 566ec18 Update README.md
* 811983f Add README2.md
```

Ramas (*branches*)

- Fast-forward merge con rebase
- Volvemos a intentar el merge

```
~/repo$ git merge newdoc --ff-only
fatal: Not possible to fast-forward, aborting.
```



```
~/repo$ git merge newdoc --ff-only
Updating 1687778..531b278
Fast-forward
 README.md | 4 +++-
1 file changed, 3 insertions(+), 1 deletion(-)
```


Ramas (*branches*)

- Fast-forward merge con rebase
 - La historia es lineal

```
~/repo$ git log --graph --all --oneline
* 531b278 (HEAD -> master, newdoc) Add new line in README.md
* 1687778 Add content to README2.md
* 566ec18 Update README.md
* 811983f Add README2.md
```

Borrar commits

- Podemos **borrar los últimos commits** de la rama
- Al borrarlos posicionamos **HEAD** y la **rama** en un commit del pasado
- Todavía sería posible **recuperar esos commits** si estuvieran referenciados por **otra rama**

```
$ git reset
```

Borrar commits

- Hay que tener cuidado de **no borrar más de la cuenta** y perder información
- Ante la duda, haz una **copia del repositorio "a mano"**
- Si has **subido al servidor los commits** que vas a borrar, es posible que otros compañeros se basen en ellos y es peligroso borrarlos



Borrar commits

- **\$ git reset --hard <commit>**
 - Borra el contenido actual del area de trabajo y del area de staging (salvo los ficheros *untracked*)
 - Retrocede hasta el commit indicado
 - Saca el contenido del mismo al directorio de trabajo
 - No habrá posibilidad de recuperar los commits posteriores a menos que estén apuntados por otra rama o tag

Borrar commits

- **\$ git reset --hard HEAD**
 - Borra el contenido actual del area de trabajo y del area de staging (salvo los ficheros *untracked*)
 - Restaura en el directorio de trabajo el contenido del último commit de la rama

Borrar commits

- Ejercicio 5

- Crea un fichero nuevo en master
- Modifica el contenido del README.md
- Ejecuta: **\$ git reset --hard HEAD**
- Comprueba que el contenido del README.md se restaura al contenido original pero que el nuevo fichero (*untracked*) sigue intacto

Borrar commits

- **\$ git reset --soft <commit>**
 - Deja el area de trabajo y el area de staging sin cambios
 - Retrocede hasta el commit que le digamos
 - Como el area de trabajo (y staging) se mantienen, no se pierde contenido de los ficheros si se hace commit
 - El histórico de commits si se pierde con el reset

Borrar commits

- **Ejercicio 6**
 - Modifica el contenido de README.md y README2.md
 - Retrocede master dos commits anteriores
 - Revisa que los ficheros README.md y README2.md están intactos (no han perdido información)
 - Haz un nuevo commit

Borrar commits

- **\$ git reset --mixed <commit>**
 - Deja el area de trabajo sin cambios
 - Borra el area de staging
 - Retrocede hasta el commit que le digamos
 - Como el area de trabajo se mantiene, no se pierde contenido de los ficheros si se hace commit
 - El histórico de commits si se pierde con el reset
 - Es el modo por defecto

Borrar commits

- Ejercicio 7
 - Agrupar los dos últimos commits en un único commit

Resolviendo conflictos

- En los ejemplos hemos visto que dos ramas se **mezclaban de forma automática**
- Cuando en dos ramas se editan **ficheros distintos** o el mismo fichero pero **diferentes partes**, no existen conflictos y la mezcla se hace de forma automática

Que no haya conflictos (y git no se queje) no implica que el código resultante de la mezcla **funcione sin problemas** (o incluso compile)



Resolviendo conflictos

- Cuando en dos ramas se ha modificado **la misma línea del mismo fichero**, se produce un **conflicto al mezclar** y el merge (con commit, squash o rebase) no puede hacerse de forma automática
- Es necesario que el desarrollador decida cómo debe ser la línea de código final cuando en cada commit padre se **ha modificado la misma línea del mismo fichero**

Resolviendo conflictos

- Crear y posicionarnos en una nueva rama fix-readme
 - Editar README.md
 - Añadir contenido en una línea ya existente (marcado en azul aquí), hacer un commit

```
# Repo documentation
```

```
This project aims at providing a testing infrastructure for  
distributed applications that are so complex that it's  
difficult to write end-to-end tests for them.
```

```
New content.
```

```
## A new section
```

```
With its own content. And some typos fixed.
```

Resolviendo conflictos

- Volver a master
 - Editar README.md
 - Añadir contenido en la misma línea (marcado en rojo aquí), hacer un commit

```
# Repo documentation
```

```
This project aims at providing a testing infrastructure for  
distributed applications that are so complex that it's  
difficult to write end-to-end tests for them.
```

```
New content.
```

```
## A new section
```

```
This should conflict. With its own content.
```

Resolviendo conflictos

- Volvemos a fix-readme
- Editar README.md
- Añadir una nueva sección al final, hacer un commit

```
# Repo documentation
```

```
This project aims at providing a testing infrastructure for  
distributed applications that are so complex that it's  
difficult to write end-to-end tests for them.
```

```
New content.
```

```
## A new section
```

```
With its own content. And some typos fixed.
```

```
## A third section
```

```
Where we explain very important topics.
```

Resolviendo conflictos

- Merge con conflictos
 - Queremos integrar los cambios de la rama fix-readme en master
 - Vamos a seguir un merge fast-forward
 - Tenemos que rebasar fix-readme sobre master (porque se han hecho commits en master desde que se creó la rama fix-readme)

Resolviendo conflictos

- Rebasamos master...

```
~/repo$ git rebase master
```

```
First, rewinding head to replay your work on top of it...
```

```
Applying: add "more info" to README.md
```

```
Using index info to reconstruct a base tree...
```

```
M      README.md
```

```
Falling back to patching base and 3-way merge...
```

```
Auto-merging README.md
```

```
CONFLICT (content): Merge conflict in README.md
```

```
error: Failed to merge in the changes.
```

```
Patch failed at 0001 add "more info" to README.md
```

```
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".

- Error: failed to merge

Resolviendo conflictos

- Rebase mueve HEAD hasta el primer ancestro común que encuentra, entonces empieza a aplicar los commits de master, luego los de fix-readme
- El último commit de master cambia una línea de readme que también cambia fix-readme en su primer commit
- El rebase no puede mezclar ambos cambios, ¿qué línea deja/quita?
- El rebase para, y nos deja la opción de solucionar el problema
- Una vez solucionado, ejecutamos

```
$ git add <fichero>
$ git rebase --continue
```

Resolviendo conflictos

- Veamos el fichero README.md
 - <<<<<< HEAD: marca el inicio del contenido de HEAD (situado en ese momento en el último commit de master)
 - =====: separa ambas partes, lo que viene después es el contenido de fix-readme
 - >>>>>> fix typos: marca el final del conflicto y el commit

```
## A new section
<<<<<< HEAD
This should conflict. With its own content.
=====
With its own content. And some typos fixed.
>>>>>> Fix typos
```

Resolviendo conflictos

- Visual Studio Code sabe que es un conflicto de git y nos ayuda a solucionarlo

Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes

```
<<<<<< HEAD (Current Change)
```

```
(more info) New paragraph
```

```
=====
```

```
New paragraph (more info)
```

```
>>>>>> add "more info" to README.md (Incoming Change)
```

Resolviendo conflictos

- Mezclamos ambas líneas manualmente como queramos

```
## A new section
```

```
This should conflict. With its own content. And some  
typos fixed.
```

- Añadimos el fichero con git add
 - Esto le indica a git que ya lo hemos solucionado
- Continuamos el rebase con git rebase --continue

Resolviendo conflictos

- **Ejercicio 8**
 - Crea una nueva rama, modifica un fichero y comitea
 - Vuelve a master, modifica el mismo fichero en la misma línea y comitea
 - Haz merge resolviendo los conflictos

Desarrollo colaborativo con Git y GitHub

- Introducción: Git y GitHub
- Git: trabajando en local
- **Git y GitHub: trabajando en equipo**

Git y GitHub

- La forma recomendada de trabajar con repositorios remotos en git es usar una **clave SSH**
- Técnicamente se deben generar **dos claves**, una **pública** y una **privada**
- La **pública** se configura en el **servidor git** y la **privada** queda siempre en la **máquina del usuario**
- Permite una forma de **seguridad avanzada** porque la clave privada nunca se transmite

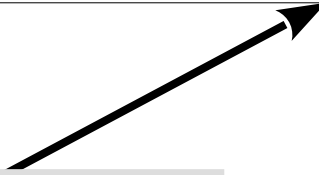
<https://help.github.com/enterprise/2.15/user/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/>

Git y GitHub

- Generación en linux

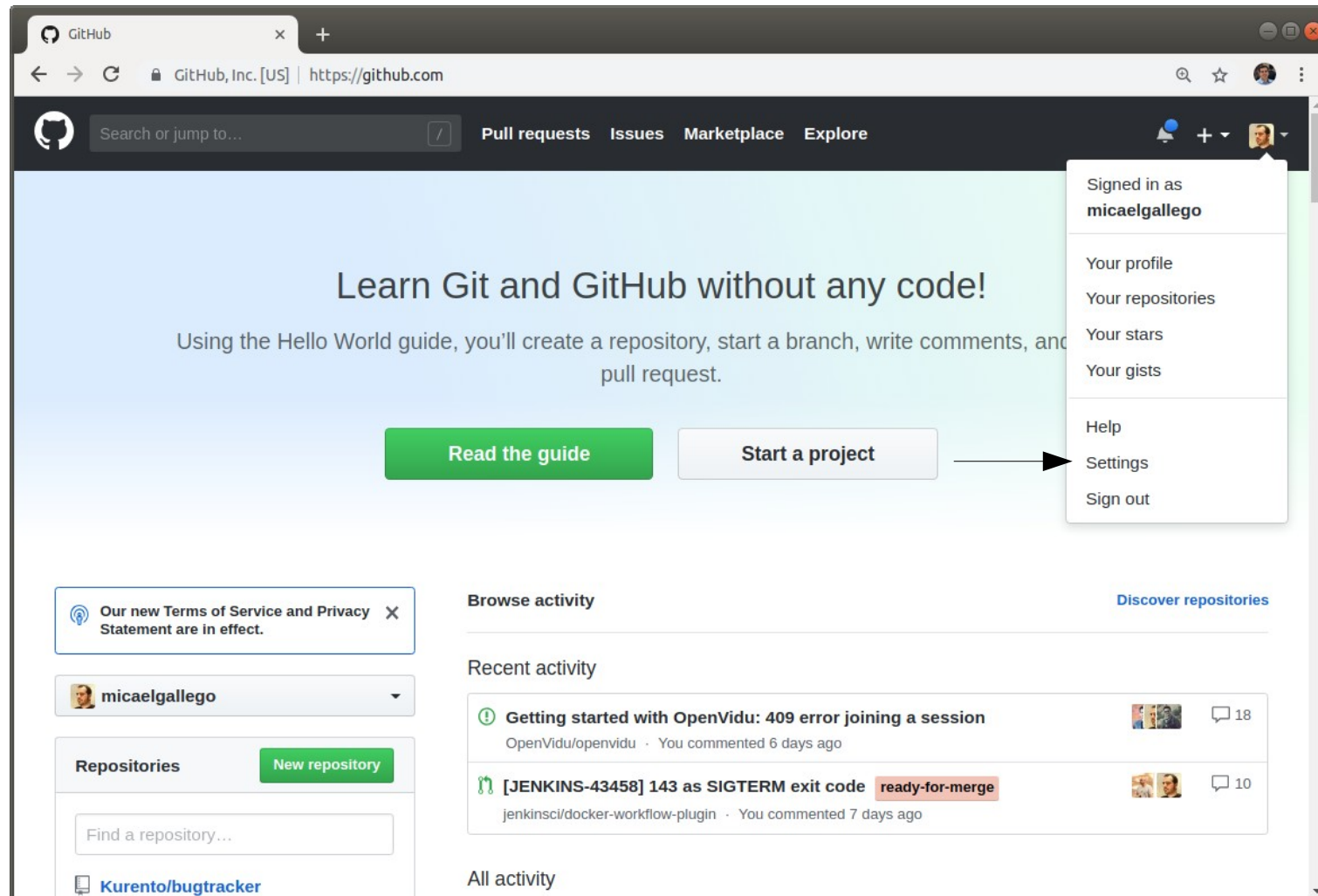
```
$ ssh-keygen -t rsa -b 4096 -C "michael.gallego@gmail.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/mica/.ssh/id_rsa):
Created directory '/c/Users/patxi/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/mica/.ssh/id_rsa.
Your public key has been saved in /home/mica/.ssh/id_rsa.pub.
```

Tenemos que añadir la clave pública a GitHub



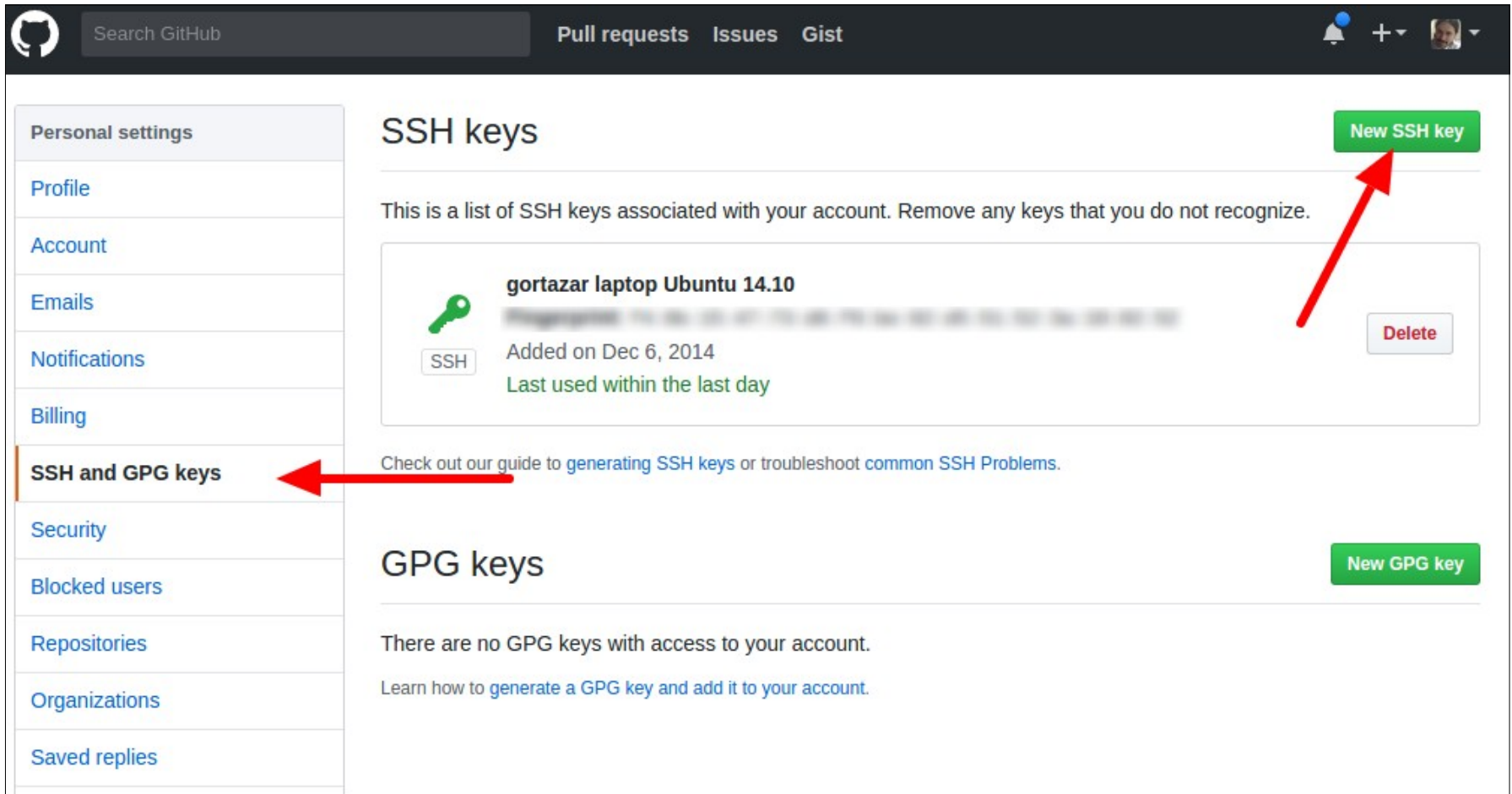
Git y GitHub

- Añadimos la clave pública a GitHub



Git y GitHub

- Añadimos la clave pública a GitHub



The screenshot shows the GitHub 'SSH keys' settings page. On the left sidebar, 'SSH and GPG keys' is highlighted. The main content area has a 'New SSH key' button in the top right. Below it, a list of SSH keys is shown, with one key named 'gortazar laptop Ubuntu 14.10'. A red arrow points to the 'New SSH key' button, and another red arrow points to the 'SSH and GPG keys' link in the sidebar.

SSH keys

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

| Key Name | Added | Last Used | Actions |
|------------------------------|----------------------|-------------------------------|---------|
| gortazar laptop Ubuntu 14.10 | Added on Dec 6, 2014 | Last used within the last day | Delete |

[Check out our guide to generating SSH keys](#) or [troubleshoot common SSH Problems](#).

GPG keys

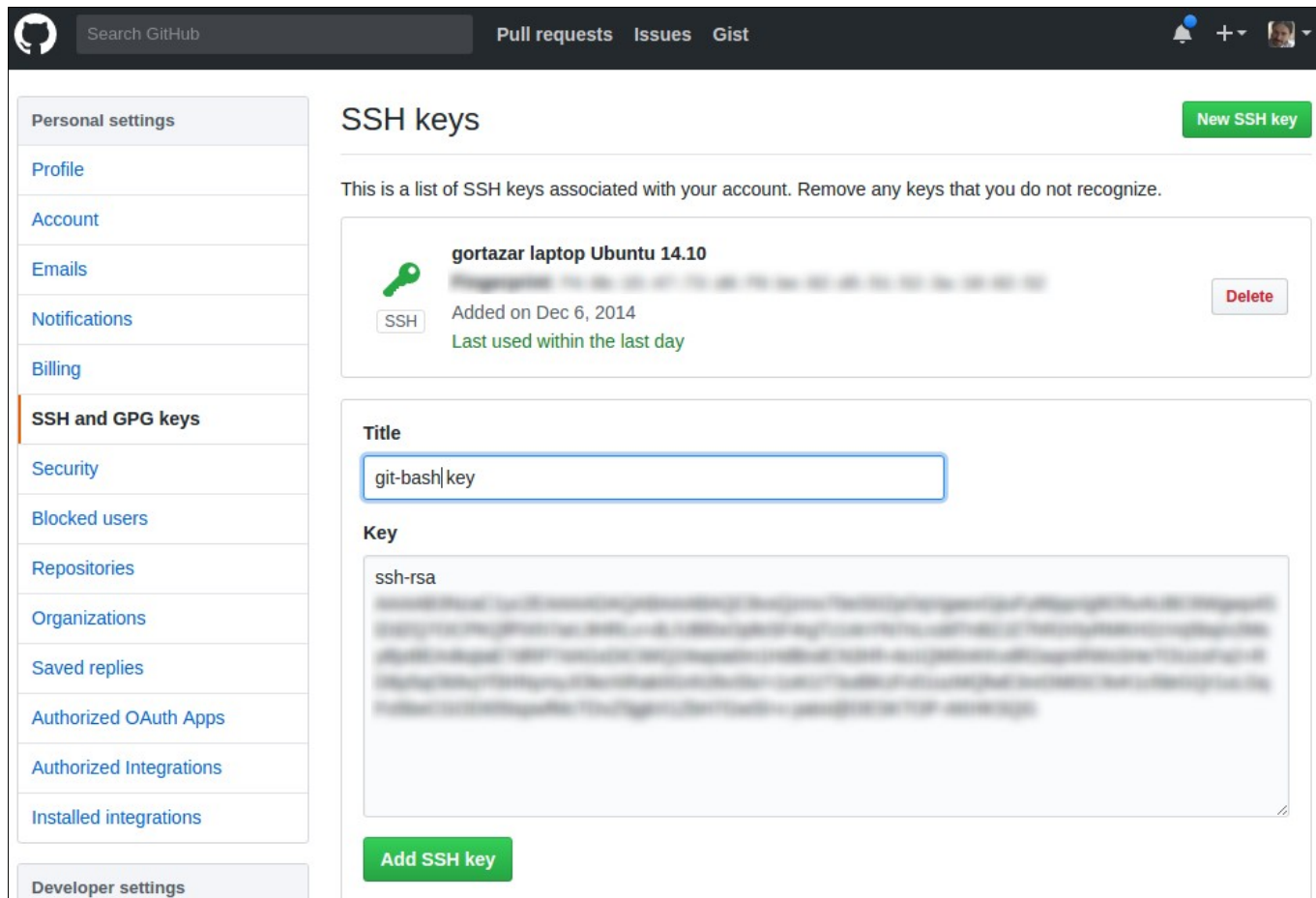
There are no GPG keys with access to your account.

Learn how to [generate a GPG key and add it to your account](#).

Git y GitHub

- Añadimos la clave pública a GitHub

```
$ cat /home/mica/.ssh/id_rsa.pub
```



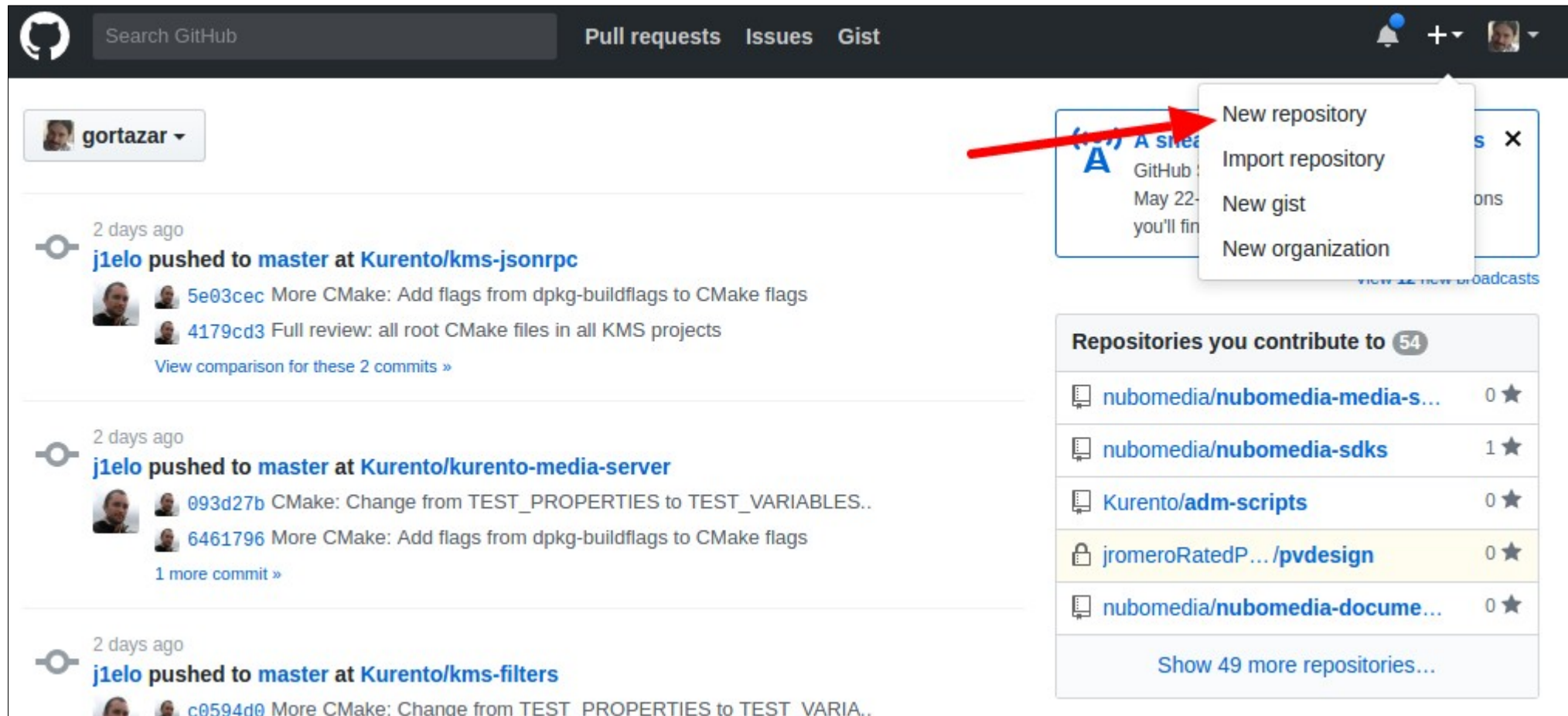
The screenshot shows the GitHub 'SSH keys' page. On the left is a sidebar with navigation links: Personal settings, Profile, Account, Emails, Notifications, Billing, SSH and GPG keys (highlighted), Security, Blocked users, Repositories, Organizations, Saved replies, Authorized OAuth Apps, Authorized Integrations, Installed integrations, and Developer settings. The main content area is titled 'SSH keys' and includes a 'New SSH key' button. Below the title is a message: 'This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.' There is one key listed: 'gortazar laptop Ubuntu 14.10', which was added on Dec 6, 2014, and last used within the last day. It has a 'Delete' button. Below the list is a form to add a new key, with fields for 'Title' (containing 'git-bash|key') and 'Key' (containing a public key). An 'Add SSH key' button is at the bottom of the form.

Crear un repositorio en GitHub

- Hay dos formas de comenzar a trabajar en GitHub
 - Creamos un repositorio vacío
 - Esto nos permite subir nuestros cambios si ya tenemos contenido
 - Es un poco más engorroso, pero GitHub nos indica cómo hacerlo
 - Crear un repositorio con un commit inicial
 - Normalmente se incluye un README.md y el fichero de licencia (LICENSE)
 - Este repositorio se puede clonar directamente
 - Al clonar el repositorio, se crea la copia local y se configura para que esté conectado al repositorio remoto

Crear un repositorio en GitHub

- Crear un repositorio con un commit inicial



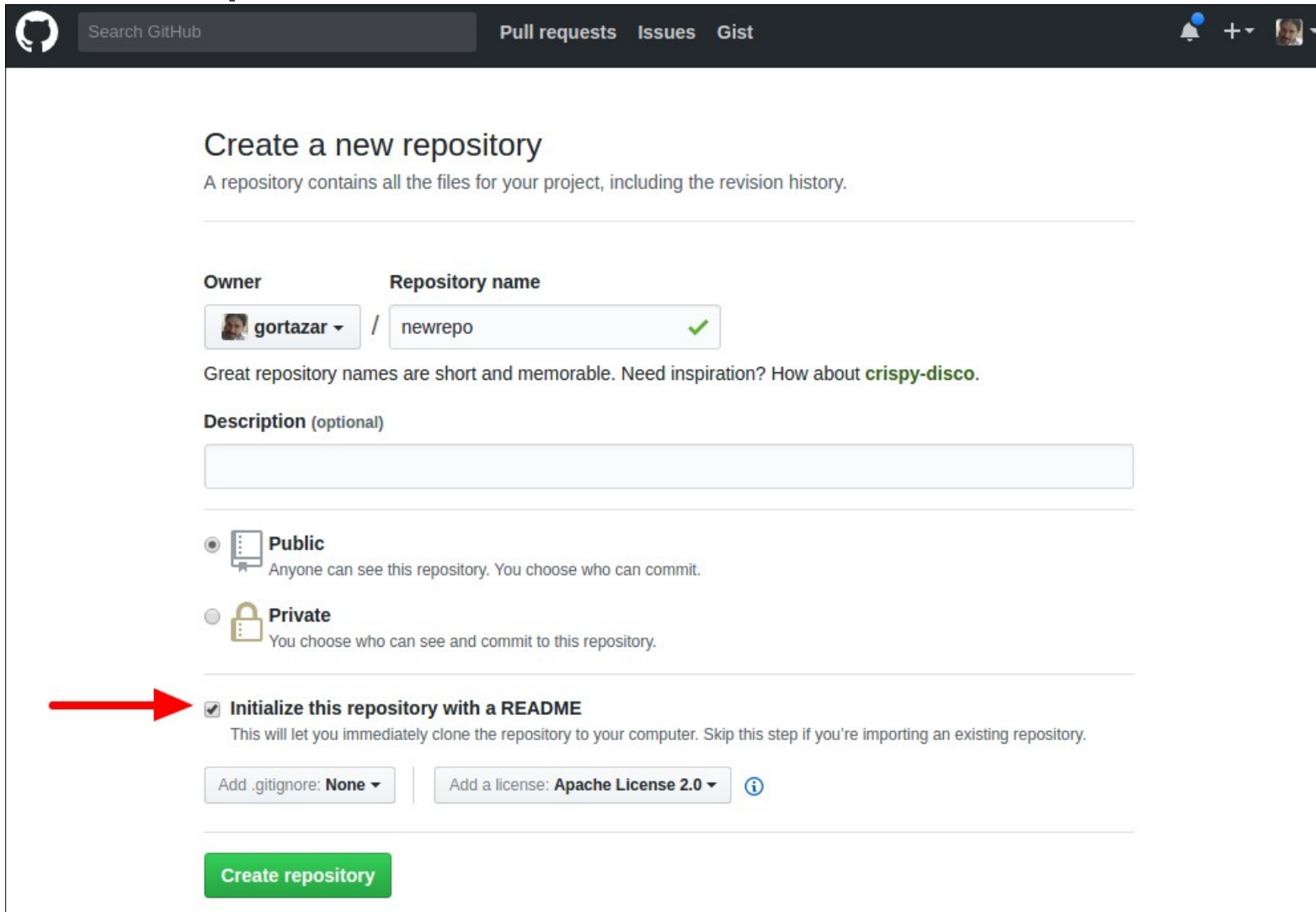
The screenshot shows the GitHub homepage for user 'gortazar'. The top navigation bar includes 'Pull requests', 'Issues', and 'Gist'. The user's profile picture is in the top right corner. A red arrow points to the user menu, which is open, showing options: 'New repository', 'Import repository', 'New gist', and 'New organization'. The main content area shows a list of recent commits from 'j1elo' to repositories like 'Kurento/kms-jsonrpc', 'Kurento/kurento-media-server', and 'Kurento/kms-filters'. On the right, there is a section titled 'Repositories you contribute to' with a list of repositories and their star counts.

| Repositories you contribute to 54 | |
|-----------------------------------|-----|
| nubomedia/nubomedia-media-s... | 0 ★ |
| nubomedia/nubomedia-sdks | 1 ★ |
| Kurento/adm-scripts | 0 ★ |
| jromeroRatedP.../pvdesign | 0 ★ |
| nubomedia/nubomedia-docume... | 0 ★ |

Show 49 more repositories...

Crear un repositorio en GitHub

- Crear un repositorio con un commit inicial



Search GitHub Pull requests Issues Gist

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: gortazar / Repository name: newrepo ✓

Great repository names are short and memorable. Need inspiration? How about **crispy-disco**.

Description (optional)

☒ **Public**
Anyone can see this repository. You choose who can commit.

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

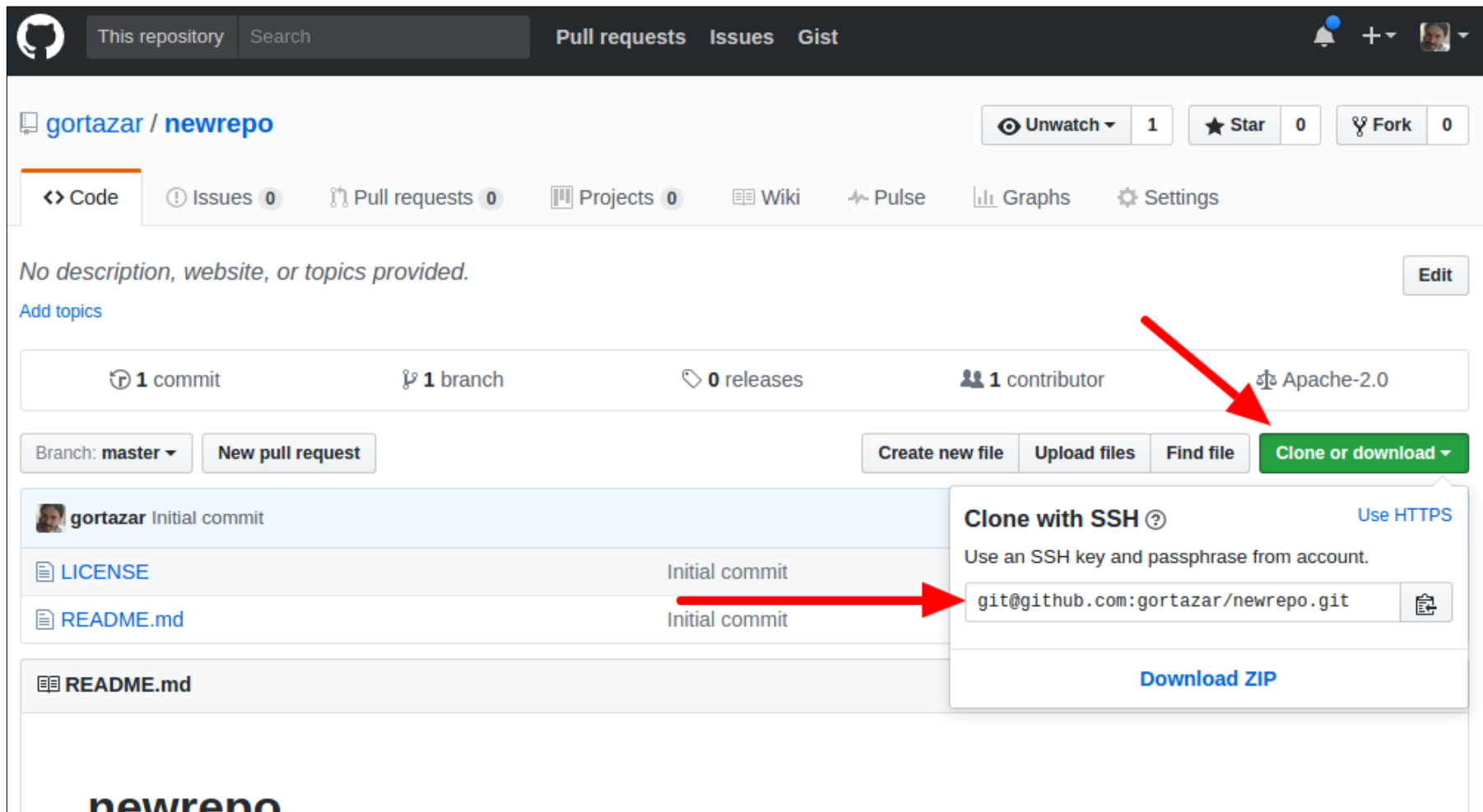
☒ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: Apache License 2.0 ⓘ

Create repository

Trabajo con un repositorio remoto

- Clonar el repositorio



The screenshot shows the GitHub interface for a repository named 'newrepo' by user 'gortazar'. The repository has 1 commit, 1 branch, 0 releases, and 1 contributor. The 'Clone or download' button is highlighted with a red arrow. A dropdown menu is open, showing the 'Clone with SSH' option, which is also highlighted with a red arrow. The SSH clone command is displayed as 'git@github.com:gortazar/newrepo.git'.

Repository: gortazar / newrepo

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs Settings

No description, website, or topics provided. [Add topics](#) [Edit](#)

1 commit 1 branch 0 releases 1 contributor Apache-2.0

Branch: master New pull request Create new file Upload files Find file Clone or download

gortazar Initial commit

LICENSE Initial commit

README.md Initial commit

README.md

newrepo

Clone with SSH ? Use HTTPS

Use an SSH key and passphrase from account.

git@github.com:gortazar/newrepo.git

Download ZIP

Trabajo con un repositorio remoto

- Clonar el repositorio

```
$ git clone git@github.com:gortazar/newrepo.git
Cloning into 'newrepo'...
The authenticity of host 'github.com (192.30.253.112)' can't be established.
RSA key fingerprint is SHA256:nThbg6kXUpJWGl7E1IGOCspRomTxdCARLviKw6E5SY8.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'github.com,192.30.253.112' (RSA) to the list of
known hosts.
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), 4.13 KiB | 0 bytes/s, done.
```

Trabajo con un repositorio remoto

- El repositorio local tiene automáticamente configurado el repositorio de GitHub como un repositorio remoto
- Lo habitual es tener un único repositorio remoto (el del servidor)
- Pero podríamos tener varios repositorios remotos para casos de uso avanzados (p.e. sincronizar repositorios)
- Cada uno tiene un nombre, el repositorio de GitHub se llama **origin**
- Dentro de cada repositorio remoto, podemos configurar una o varias ramas

Trabajo con un repositorio remoto

```
$ git remote show origin
* remote origin
  Fetch URL: git@github.com:gortazar/newrepo.git
  Push URL: git@github.com:gortazar/newrepo.git
  HEAD branch: master
  Remote branch:
    master tracked
  Local branch configured for 'git pull':
    master merges with remote master
  Local ref configured for 'git push':
    master pushes to master (up to date)
```

Ahora mismo la rama master está sincronizada con la rama master del remoto (up to date)

Trabajo con un repositorio remoto

- Vamos a hacer un cambio

```
$ echo "Some update" >> README.md
```

```
$ git status
```

```
On branch master
```

```
Your branch is up-to-date with 'origin/master'.
```

```
Changes not staged for commit:
```


```
  (use "git add <file>..." to update what will be committed)
```

```
  (use "git checkout -- <file>..." to discard changes in working directory)
```

```
    modified:   README.md
```

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

git status ahora nos informa de si estamos sincronizados con la versión remota de esta rama



Trabajo con un repositorio remoto

- Comiteamos

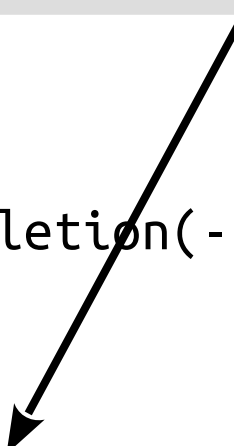
```
$ git add README.md

$ git commit -m "Updated README"
[master 444bf6a] Updated README
1 file changed, 1 insertion(+), 1 deletion(-)

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

nothing to commit, working tree clean
```

git status nos indica que el repositorio local está 1 commit más avanzado que el remoto



Trabajo con un repositorio remoto

- Subamos los cambios al repositorio remoto

```
$ git push origin master
Counting objects: 3, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 305 bytes | 0 bytes/s,
done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:gortazar/newrepo.git
    a53fc64..444bf6a  master -> master

$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

El repositorio remoto al que queremos subir los cambios y la rama dentro del repositorio

Trabajo colaborativo

- Normalmente se utiliza **GitHub** como repositorio centralizado de confianza
- Cada desarrollador **tendrá una copia** de este repositorio en local
- Los cambios, además de comitearlos a nuestro repositorio local tenemos que “empujarlos” (**push**) al repositorio remoto
- Periódicamente tendremos que traernos los cambios de otros desarrolladores “tirando” (**pull/fetch**) de ellos

Trabajo colaborativo

- Podemos dar permisos a otros usuarios a nuestro repositorio GitHub
- Esos usuarios se pueden clonar el repositorio y hacer commits al mismo
- Para colaborar, podemos descargar los commits de otros usuarios al repositorio

Trabajo colaborativo

The screenshot shows the GitHub web interface for the 'Collaborators' settings of a repository named 'newrepo' by user 'michaelgallego'. The browser address bar shows the URL 'https://github.com/michaelgallego/newrepo/settings/collaboration'. The top navigation bar includes links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. Below this, the repository name and statistics (1 Unwatch, 0 Stars, 0 Forks) are displayed. A horizontal menu contains links for 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Insights', and 'Settings'. The 'Settings' link is highlighted with an orange underline and a black arrow pointing to it from the right. On the left side, a sidebar menu lists various settings categories: 'Options', 'Collaborators' (highlighted with an orange bar and a black arrow pointing to it from the left), 'Branches', 'Webhooks', 'Integrations & services', 'Deploy keys', 'Secrets', 'Moderation', and 'Interaction limits'. The main content area is titled 'Collaborators' with a subtitle 'Push access to the repository'. It contains the text: 'This repository doesn't have any collaborators yet. Use the form below to add a collaborator.' Below this is a search instruction: 'Search by username, full name or email address' followed by a note: 'You'll only be able to find a GitHub user by their email address if they've chosen to list it publicly. Otherwise, use their username instead.' A text input field is provided for the search, with a black arrow pointing to it from the bottom-left. To the right of the input field is a button labeled 'Add collaborator'. The footer of the page includes copyright information for 2018 GitHub, Inc. and links to 'Terms', 'Privacy', 'Security', 'Status', 'Help', 'Contact GitHub', 'Pricing', 'API', 'Training', 'Blog', and 'About'.

Trabajo colaborativo

- **Simulación de dos usuarios**
 - Vamos a ver cómo descargar commits del repositorio duplicando el repositorio local en otra carpeta (newrepo2)
 - Así podemos crear un commit en un repositorio y descargarlo en otro

Trabajo colaborativo

- Creamos README2.md en newrepo2

```
~/newrepo2$ echo "Other README" >> "README2.md"
~/newrepo2$ git add .
~/newrepo2$ git commit -m "Add README2 file"
[master 8b7fb4d] Add README2 file
1 file changed, 1 insertion(+)
create mode 100644 README2.md

$ git push origin master
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 288 bytes | 288.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:micaelgallego/newrepo.git
dc3d426..8b7fb4d master -> master
```

Trabajo colaborativo

- Volvemos a newrepo y descargamos el repo remoto
- **git fetch** descarga el servidor remoto, pero no integra de ninguna forma su contenido en la rama master que tenemos en local

```
~/newrepo$ git fetch
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From github.com:micaelgallego/newrepo
    dc3d426..8b7fb4d  master    -> origin/master
```

Trabajo colaborativo

- Vemos en qué estado se encuentra el repo local

```
~/newrepo$ git status
On branch master
Your branch is behind 'origin/master' by 1 commit, and can be fast-
forwarded.
    (use "git pull" to update your local branch)

nothing to commit, working tree clean
```

- Estamos 1 commit por detrás de origin/master
- El master local se puede sincronizar con origin/master con **fast-forward**, lo que quiere decir que se puede integrar simplemente integrando los commits y haciendo que master apunte al último commit

Trabajo colaborativo

- **git merge** sirve para integrar los cambios de remoto a local

```
~/newrepo$ git merge
Updating dc3d426..8b7fb4d
Fast-forward
 README2.md | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 README2.md

~/newrepo$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

- **git pull** es equivalente a un **git fetch** seguido de **git merge**

Trabajo colaborativo

- ¿Qué ocurre si dos usuarios hacen commits a master?
- El primero que los suba lo hace sin problemas

```
~/newrepo$ echo "More content" >> README.md
~/newrepo$ git add .
~/newrepo$ git commit -m "Add more content in README"
[master 87fa529] Add more content in README
1 file changed, 1 insertion(+)

~/newrepo$ git push
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 318 bytes | 318.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:micaelgallego/newrepo.git
8b7fb4d..87fa529 master -> master
```

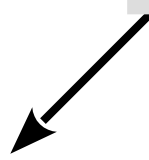
Trabajo colaborativo

- ¿Qué ocurre si dos usuarios hacen commits a master?
- El segundo tiene problemas al subir

```
~/newrepo2$ echo "More content" >> README2.md
~/newrepo2$ git add .
~/newrepo2$ git commit -m "Add more content in README2"
[master 09f7403] Add more content in README2
1 file changed, 1 insertion(+)
```

```
~/newrepo2$ git push
To github.com:micaelgallego/newrepo.git
! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'git@github.com:micaelgallego/newrepo.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Nos dice los problemas que tenemos y cómo solucionarlo



Trabajo colaborativo

- ¿Qué ocurre si dos usuarios hacen commits a master?
- El comando **git push** intenta hacer una mezcla de las ramas con **fast-forward**
- Si **no puede hacerlo**, da **error** y recomienda primero integrar los cambios remotos en la rama local antes de hacer el push
- Esa integración se puede hacer:
 - Rebasando los commits locales sobre los commits remotos (**git rebase**)
 - Creando un nuevo commit de mezcla (merge) que integra los cambios locales y los remotos (**git merge**)
- Cuando la integración se ha hecho, se puede hacer push

Trabajo colaborativo

- ¿Qué ocurre si dos usuarios hacen commits a master?
- Hacemos un **git fetch** para descargar el repositorio remoto

```
~/newrepo2$ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From github.com:micaelgallego/newrepo
   8b7fb4d..87fa529  master       -> origin/master

~/newrepo2$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commits each, respectively.
   (use "git pull" to merge the remote branch into yours)

nothing to commit, working tree clean
```

Trabajo colaborativo

- ¿Qué ocurre si dos usuarios hacen commits a master?
- Vemos en el histórico que efectivamente la rama master ha divergido en local y en remoto

```
~/newrepo2$ git log --graph --all --oneline
* 09f7403 (HEAD -> master) Add more content in README2
| * 87fa529 (origin/master, origin/HEAD) Add more content in README
|/
* 8b7fb4d Add README2 file
* dc3d426 Update README
* 9c431a9 Initial commit
```

Trabajo colaborativo

- ¿Qué ocurre si dos usuarios hacen commits a master?
- Hacemos un **git merge** (o **git pull**) para mezclar ambas ramas

```
~/newrepo2$ git merge
Merge made by the 'recursive' strategy.
 README.md | 1 +
 1 file changed, 1 insertion(+)

~/newrepo2$ git log --graph --all --oneline
*   da4c96b (HEAD -> master) Merge remote-tracking branch
'refs/remotes/origin/master'
|\
| * 87fa529 (origin/master, origin/HEAD) Add more content in README
* | 09f7403 Add more content in README2
|/
* 8b7fb4d Add README2 file
* dc3d426 Update README
* 9c431a9 Initial commit
```

Trabajo colaborativo

- ¿Qué ocurre si dos usuarios hacen commits a master?
- Ahora ya podemos subir nuestro cambio al servidor (y además también subimos el commit de merge)

```
~/newrepo2$ git push
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 601 bytes | 300.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0)
To github.com:micaelgallego/newrepo.git
 87fa529..da4c96b  master -> master
```

```
~/newrepo2$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

Trabajo colaborativo

- ¿Qué ocurre si dos usuarios hacen commits a master?
- Podemos hacer lo mismo rebasando los cambios locales sobre los cambios remotos
 - Vamos a hacer que el usuario en newrepo haga un commit nuevo
 - Intentará hacer push y obtenga un error
 - Se rebasará sobre la rama remota
 - Finalmente podrá hacer el push sin problemas

Trabajo colaborativo

- ¿Qué ocurre si dos usuarios hacen commits a master?
- Nuevo commit en newrepo e intento de push

```
~/newrepo$ echo "More Content" >> README.md
~/newrepo$ git add .
~/newrepo$ git commit -m "Add more content (again) in README"
[master 4b2540f] Add more content (again) in README
1 file changed, 1 insertion(+)

~/newrepo$ git push
To github.com:micaelgallego/newrepo.git
! [rejected]          master -> master (fetch first)
error: failed to push some refs to
'git@github.com:micaelgallego/newrepo.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Trabajo colaborativo

- ¿Qué ocurre si dos usuarios hacen commits a master?
- Descarga

```
~/newrepo$ git fetch
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 5 (delta 0), pack-reused 0
Unpacking objects: 100% (5/5), done.
From github.com:micaelgallego/newrepo
   87fa529..da4c96b  master    -> origin/master

~/newrepo$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 2 different commits each, respectively.
   (use "git pull" to merge the remote branch into yours)

nothing to commit, working tree clean
```


Trabajo colaborativo

- ¿Qué ocurre si dos usuarios hacen commits a master?
- Rebase (`git rebase` o `git pull --rebase`)

```
~/newrepo$ git rebase
```

```
First, rewinding head to replay your work on top of it...
```

```
Applying: Add more content (again) in README
```

```
~/newrepo$ git log --graph --all --oneline
```

```
* 7b86c28 (HEAD -> master) Add more content (again) in README
```

```
*   da4c96b (origin/master, origin/HEAD) Merge remote-tracking branch  
'refs/remotes/origin/master'
```

```
|\
```

```
| * 87fa529 Add more content in README
```

```
* | 09f7403 Add more content in README2
```

```
||
```

```
* 8b7fb4d Add README2 file
```

```
* dc3d426 Update README
```

```
* 9c431a9 Initial commit
```

Trabajo colaborativo

- ¿Qué ocurre si dos usuarios hacen commits a master?
- Push

```
~/newrepo$ git push
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 335 bytes | 335.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:micaelgallego/newrepo.git
da4c96b..7b86c28 master -> master
```

Trabajo colaborativo

- ¿Qué ocurre si dos usuarios hacen commits a master?
- Push

```
~/newrepo$ git log --graph --all --oneline
* 7b86c28 (HEAD -> master, origin/master, origin/HEAD) Add
more content (again) in README
*   da4c96b Merge remote-tracking branch
'refs/remotes/origin/master'
|\
| * 87fa529 Add more content in README
* | 09f7403 Add more content in README2
|/
* 8b7fb4d Add README2 file
* dc3d426 Update README
* 9c431a9 Initial commit
```

Trabajo colaborativo

- Cada usuario añade commits a su rama master local
- Cuando se van a subir los cambios al repositorio
 - El primero puede subir sin problemas (**git push**)
 - El segundo tiene que mezclar en local y luego subir
 - **Merge:** crea commit de merge entre los dos trabajos

git pull

git fetch + git merge

- **Rebase:** Rebasa sus cambios locales encima de los cambios remotos

git pull --rebase

git fetch + git rebase

Trabajo colaborativo con Pull Requests

- Cuando se trabaja sobre **master**, los miembros del equipo sólo pueden **conocer** lo que hace un compañero cuando ya **está integrado**
- Sería interesante que otros miembros del equipo pudieran **revisar** los cambios **antes** de que finalmente se **integren** y para, si es necesario, solicitar **mejoras**
- Es especialmente útil con:
 - **Desarrolladores con poca experiencia**
 - **Proyectos de software libre** en los que desarrolladores ajenos al proyecto hacen contribuciones al mismo
 - **Búsqueda de vulnerabilidades**

Trabajo colaborativo con Pull Requests

- Desarrollo basado en Pull Request
 - Convertimos la rama en un Pull Request: una petición de integrar ciertos cambios en la rama master
 - Asociado al Pull Request (PR) tenemos un hilo de comentarios
 - Otros desarrolladores/as pueden darnos feedback
 - Es posible que tengamos que actualizar/cambiar cosas en nuestra rama
 - Automáticamente se reflejarán en el PR cuando hagamos push de la rama de nuevo
 - Es un mecanismo habitual para trabajar en equipo con git
 - Equivalente a los parches de Gerrit
 - Disponible también en los Merge Requests de GitLab

Trabajo colaborativo con Pull Requests

- Los servidores de git han incorporado un mecanismo para que los **desarrolladores puedan solicitar revisión** antes de que los cambios se integren en master
 - Pull Requests (GitHub y Bitbucket)
 - Merge Request (GitLab)
 - Change (Gerrit)



Trabajo colaborativo con Pull Requests

- **Desarrollo basado en Pull Request de GitHub**
 - Convertimos la rama en un Pull Request: una petición de integrar ciertos cambios en la rama master
 - Asociado al Pull Request (PR) tenemos un hilo de comentarios
 - Otros desarrolladores/as pueden darnos feedback
 - Es posible que tengamos que actualizar/cambiar cosas en nuestra rama
 - Automáticamente se reflejarán en el PR cuando hagamos push de la rama de nuevo

Trabajo colaborativo con Pull Requests

- **Funcionamiento (GitHub)**

- El desarrollador crea una rama en local (generalmente asociada a una funcionalidad, *branch-per-feature*)
- Hace cambios en la rama (commits) y sube la rama a GitHub
- Desde la interfaz web, solicita integrar la rama en master (Pull Request)
- El Pull Request tiene un hilo de comentarios asociado para dar feedback sobre sus commits
- Se pueden hacer más commits sobre la rama en respuesta al feedback
- Cuando el código está listo, se "Acepta el Pull Request" y su contenido se integra en master (con merge o fast-forward)

Trabajo colaborativo con Pull Requests

- Implementamos una nueva funcionalidad
 - Creamos una rama feature-12
 - Hacemos commits en ella

```
~/newrepo$ git checkout -b feature-12
Switched to a new branch 'feature-12'
~/newrepo$ echo "code" >> code.src
~/newrepo$ git add .
~/newrepo$ git commit -m "Create first code file"
[feature-12 3f03103] Create first code file
1 file changed, 1 insertion(+)
create mode 100644 code.src
```

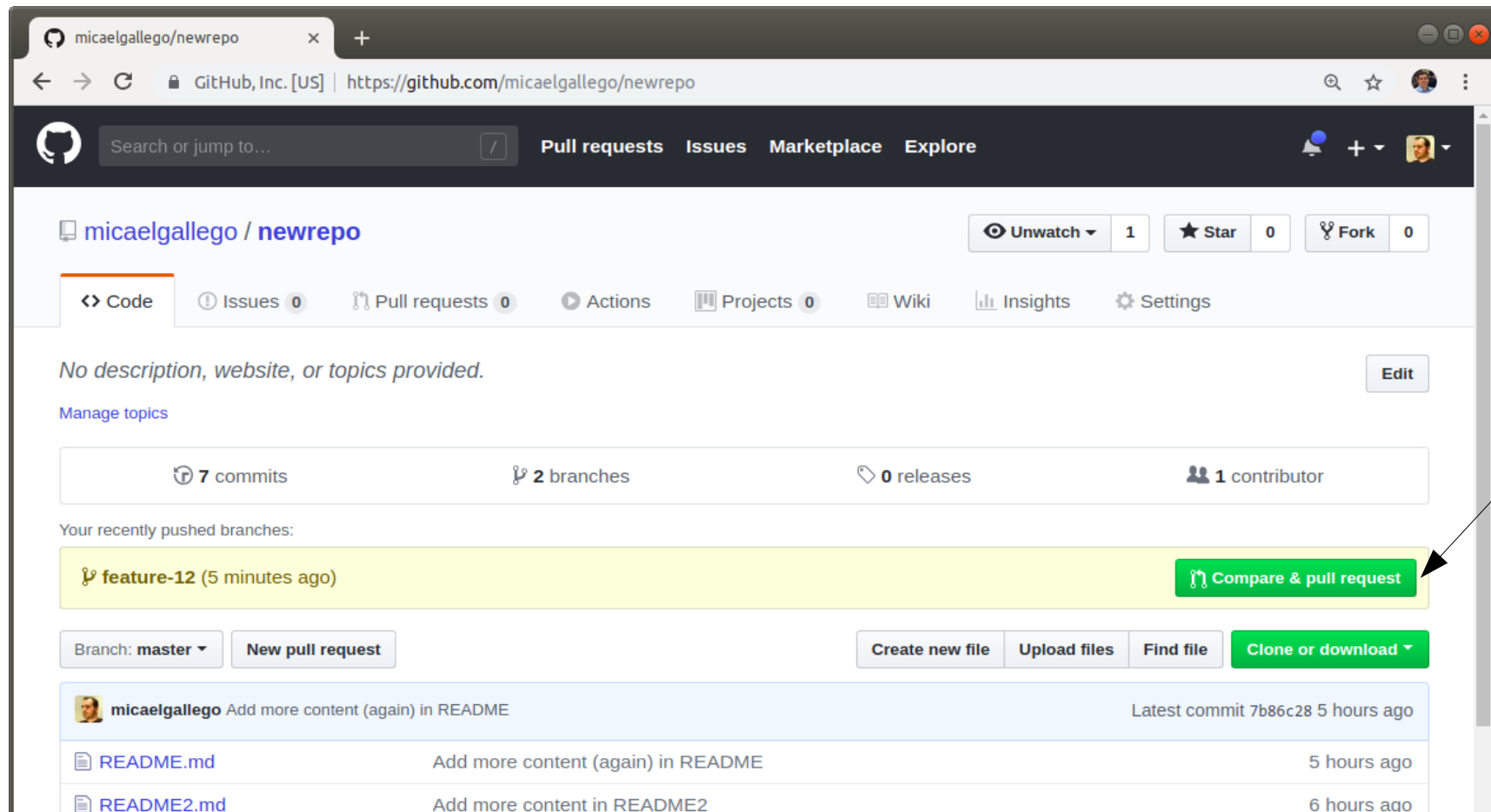
Trabajo colaborativo con Pull Requests

- Subimos la rama a GitHub

```
~/newrepo$ git push origin feature-12
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 313 bytes | 313.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'feature-12' on GitHub by visiting:
remote:   https://github.com/micaelgallego/newrepo/pull/new/feature-12
remote:
To github.com:micaelgallego/newrepo.git
* [new branch]      feature-12 -> feature-12
```

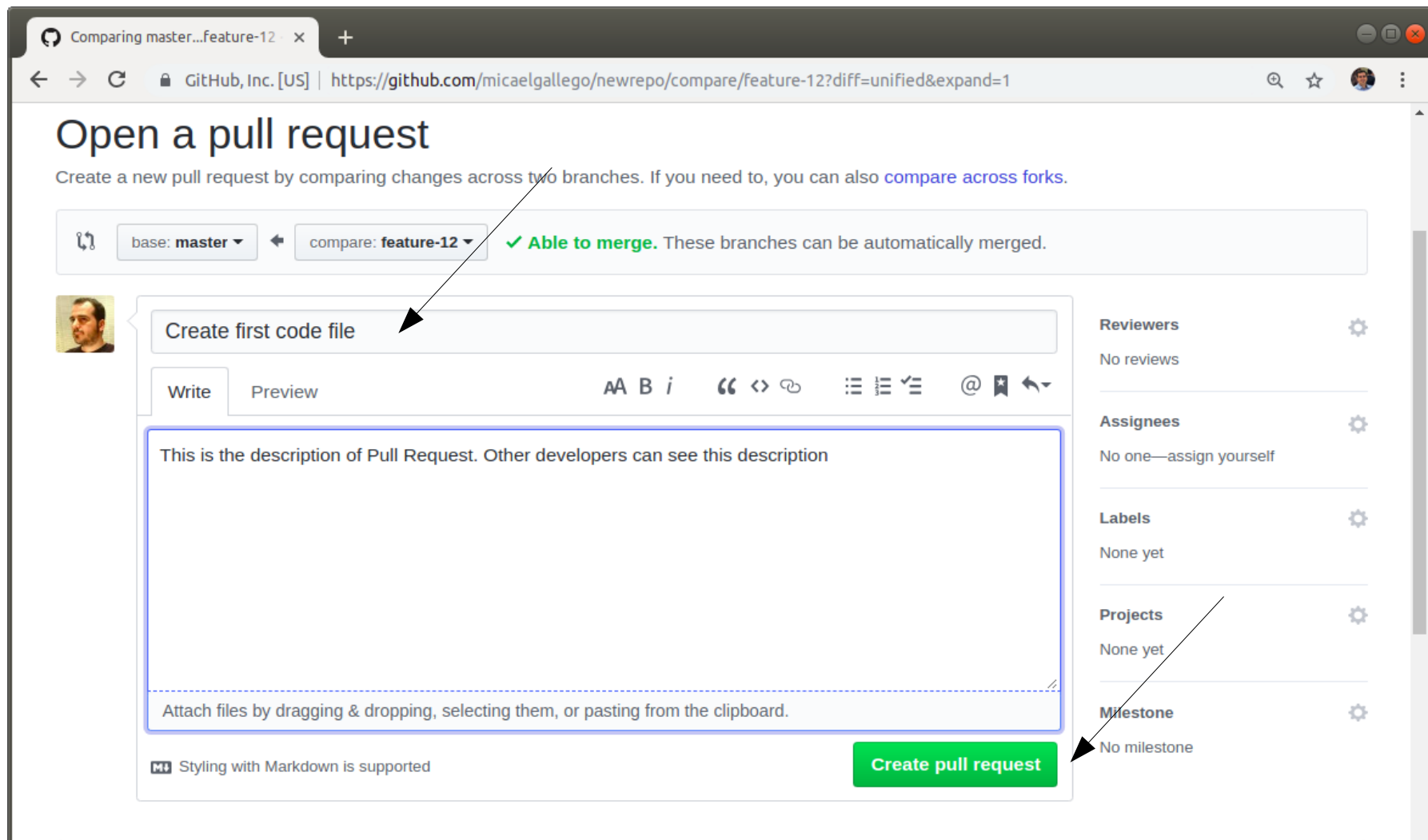
Trabajo colaborativo con Pull Requests

- Cuando consideramos que está terminada, la convertimos en Pull Request



Trabajo colaborativo con Pull Requests

- Podemos crear un título y una descripción



Comparing master...feature-12

GitHub, Inc. [US] | <https://github.com/micahelgallego/newrepo/compare/feature-12?diff=unified&expand=1>

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: master ← compare: feature-12 ✓ Able to merge. These branches can be automatically merged.

Create first code file

Write Preview

This is the description of Pull Request. Other developers can see this description

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

Styling with Markdown is supported

Create pull request

Reviewers
No reviews

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone

Trabajo colaborativo con Pull Requests

- Subimos otro commit a la rama

```

mica@mica-PR0214:~/newrepo$ echo "More code" >> code.src
mica@mica-PR0214:~/newrepo$ git add .
mica@mica-PR0214:~/newrepo$ git commit -m "More code"
[feature-12 e74fece] More code
1 file changed, 1 insertion(+)

mica@mica-PR0214:~/newrepo$ git push origin feature-12
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 261 bytes | 261.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:micaelgallego/newrepo.git
3f03103..e74fece feature-12 -> feature-12

```

Trabajo colaborativo con Pull Requests

The screenshot shows a GitHub Pull Request interface. At the top, the title is "Create first code file #1" with an "Edit" button. Below the title, a green "Open" button is followed by the text "michaelgallego wants to merge 2 commits into master from feature-12". A navigation bar shows "Conversation 0", "Commits 2", "Checks 0", and "Files changed 1". A status bar indicates "+2 -0" with a green progress bar.

A comment by "michaelgallego" (Owner) states: "This is the description of Pull Request. Other developers can see this description". Below the comment is a commit history section showing two commits by "michaelgallego": "Create first code file" (3f03103) and "More code" (e74fece). An arrow points to the first commit.

Below the commit history, a message says: "Add more commits by pushing to the feature-12 branch on michaelgallego/newrepo.".

At the bottom, a green box contains two status messages: "Continuous integration has not been set up" (with a link to "Several apps are available") and "This branch has no conflicts with the base branch" (with a note that "Merging can be performed automatically."). A green "Merge pull request" button is at the bottom left, followed by a link to "or view command line instructions.".

On the right side, there are sections for "Reviewers" (No reviews), "Assignees" (No one—assign yourself), "Labels" (None yet), "Projects" (None yet), "Milestone" (No milestone), and "Notifications" (Unsubscribe button). A note at the bottom right says: "You're receiving notifications because you authored the thread."

Trabajo colaborativo con Pull Requests

- Otro usuario puede integrar los cambios en master
 - Usando la web:
 - Creando commit de merge
 - Fusionando todos los commits (squash)
 - Rebasando (si no hay conflictos) y mezclando
 - Descargando la rama remota y usando comandos git

```
$ git fetch origin
$ git checkout -b feature-12 origin/feature-12
$ git merge master

$ git checkout master
$ git merge --no-ff feature-12
$ git push origin master
```


Trabajo colaborativo con Pull Requests

The screenshot displays a GitHub web interface for a pull request titled "Create first code file #1". The browser address bar shows the URL <https://github.com/micaelgallego/newrepo/pull/1>. The pull request status is "Merged", with a message indicating that micaelgallego merged 2 commits into master from feature-12 36 seconds ago. Below the title, there are tabs for Conversation (0), Commits (2), Checks (0), and Files changed (1), along with a diff summary of +2 -0. The main content area shows a timeline of activity: micaelgallego commented 16 minutes ago with the text "This is the description of Pull Request. Other developers can see this description"; micaelgallego added two commits 23 minutes ago, "Create first code file" (3f03103) and "More code" (e74fece); and micaelgallego merged commit e74fece into master 26 seconds ago. On the right side, there are sections for Reviewers (No reviews), Assignees (No one—assign yourself), Labels (None yet), Projects (None yet), and Milestone (No milestone). At the bottom, there is a "Pull request successfully merged and closed" message with a "Delete branch" button, and a comment input area with a "Write" tab and a "Preview" tab. The bottom right corner shows a "Notifications" section with an "Unsubscribe" button and a message stating "You're receiving notifications because you modified the open/close state."

Trabajo colaborativo con Pull Requests

- Para que los cambios en el servidor se reflejen en local, es necesario hacer un pull (fetch y merge o rebase)

Trabajo colaborativo con Pull Requests

```
$ git pull
```

```
Updating 444bf6a..0412ad4
```

```
Fast-forward
```

```
code.src | 1 +
```

```
1 file changed, 1 insertion(+)
```

```
create mode 100644 code.src
```

```
$ git status
```

```
On branch master
```

```
Your branch is up-to-date with 'origin/master'.
```

```
nothing to commit, working tree clean
```

Trabajo colaborativo con Pull Requests

- **Ejercicio 9**
 - Implementa una nueva feature en una nueva rama
 - Acepta esa rama en master con un commit que fusione los cambios

Trabajo colaborativo con Pull Requests

- Recordatorio sobre **git pull**
 - git pull intentará hacer un fast-forward
 - Si eso no es posible, intentará hacer un merge
 - Aparece un commit resultado del merge
 - Este commit habría que subirlo a la rama remota en algún momento
 - Si el merge tiene conflictos, nos dejará los conflictos en el working area para que los arreglemos y hagamos nosotros el commit
 - Si se usa git pull --rebase en vez de un merge se intentará hacer un rebase de los commits locales sobre los remotos, parando si hay conflictos

Trabajo colaborativo con Pull Requests

- En general es preferible rebasar los cambios de la rama remota
- No introduce commits adicionales
 - No es que sean malos, pero es posible que no queramos un commit adicional cada vez que actualizamos nuestra rama local con la remota
- Podemos hacer rebases a una rama remota con referencias del tipo <remoto>/<rama>:
 - git rebase origin/master
 - Busca el primer ancestro común entre origin/master y nuestra rama
 - Aplica los cambios de origin/master
 - Aplica nuestros cambios encima

Trabajo colaborativo con Pull Requests

- **Ejercicio 10**

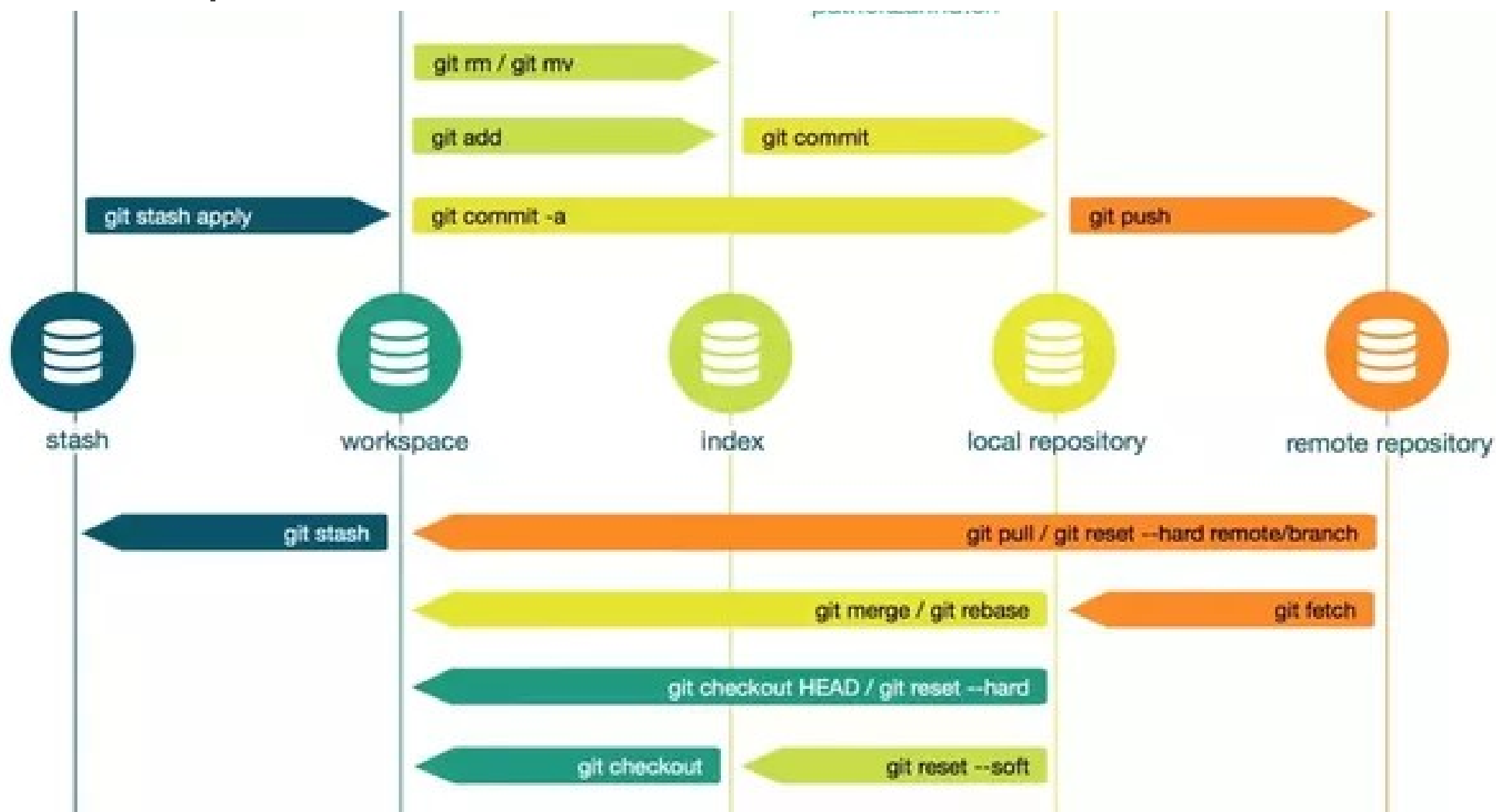
- Por parejas (Alice y Bob)
- Crear un repositorio con README.md en una de las cuentas
- Dar permiso al compañero/a
- Clonar el repositorio ambos
- Alice arrancará una rama donde modificará el README.md
- Bob hará lo propio en otra rama
- Ambos comitearán, subirán la rama y abrirán un PR
- Uno de los dos aceptará el PR primero
- El otro no podrá
- Debe conseguir subir sus cambios a master a través del PR

Trabajo colaborativo con Pull Requests

- **Ejercicio 11**
 - Por parejas (Alice y Bob)
 - Continuando con el ejercicio 10
 - Bob creará una nueva rama, la empujará a GitHub y creará un PR
 - Alice debe sacar la rama de Bob e incorporar nuevos commits empujándolos después a GitHub
 - Bob debe rebasar los cambios de Alice, incluir nuevos commits y empujarlos a GitHub
 - Bob debe aceptar el PR

Git y GitHub

- Recapitulemos...



Git y GitHub

- **git stash**
 - Stash es un área especial
 - Imaginemos un escenario donde, estando trabajando en una feature, nos llega un informe de un bug en la última release
 - Nos toca arreglarlo, pero no hemos terminado la feature
 - Tampoco está el código como para hacer un commit
 - Podemos guardar los cambios desde el último commit en el área de stash
 - Nuestra área de trabajo queda limpia
 - Podemos cambiar de rama, arreglar el bug y volver a la feature
 - Aplicamos git stash apply y recuperamos el working area en el punto en que la guardamos

Git y GitHub

- **git revert**
 - git reset nos permite volver a un commit anterior en la historia, generando commits que revierten el proyecto a un estado previo
 - git revert nos permite eliminar un commit de la historia que no sea el último
- **git rm <file>**
 - Marca un fichero en el índice para eliminar
 - El commit incluirá la eliminación de dicho fichero

Git y GitHub

- **git commit --amend**
 - Nos permite modificar el último commit
 - Permite arreglar algún olvido/despiste cuando hicimos commit por última vez
 - Incluir un nuevo fichero
 - Modificar uno existente
 - ...
 - Sólo tiene sentido si no hemos hecho push al repositorio remoto

Git y GitHub

- **git tag <tag name>**
 - Nos permite anotar un commit
 - Se utiliza para marcar releases, hitos, etc
 - Para subir el tag al repositorio remoto:
 - `git push --tags`
- **git tag**
 - Lista los tags del repositorio

Git y GitHub

- **Ejercicio 12**
 - Bob anota el último PR como v1.0
 - Alice hace un checkout de este tag v1.0