

# **GIT, GITHUB Y MARKDOWN**

**ADOLFO SANZ DE DIEGO**

**NOVIEMBRE 2017**

**1 ACERCA DE**

# 1.1 AUTOR

- **Adolfo Sanz De Diego**
  - Blog: [asanzdiego.blogspot.com.es](http://asanzdiego.blogspot.com.es)
  - Correo: [asanzdiego@gmail.com](mailto:asanzdiego@gmail.com)
  - GitHub: [github.com/asanzdiego](https://github.com/asanzdiego)
  - Twitter: [twitter.com/asanzdiego](https://twitter.com/asanzdiego)
  - LinkedIn: [in/asanzdiego](https://in/asanzdiego)
  - SlideShare: [slideshare.net/asanzdiego](https://slideshare.net/asanzdiego)

## 1.2 LICENCIA

- **Esta obra está bajo una licencia:**
  - Creative Commons Reconocimiento-CompartirIgual 3.0

## 1.3 FUENTE

- Las slides y sus fuentes las podéis encontrar en:
  - <https://github.com/asanzdiego/curso-git-github-markdown-2017>

# **2 INTRODUCCIÓN**

## 2.1 ENLACES IMPRESCINDIBLES

- Pro GIT (sobre todo temas 1, 2, 3 y 6):
  - <https://git-scm.com/book/es/v2>
- Página oficial de Git:
  - <https://git-scm.com/>
- Página oficial de GitHub:
  - <https://github.com/>
- Chuleta de la sintaxis de Markdown:
  - <http://warpedvisions.org/projects/markdown-cheat-sheet>

## 2.2 OTROS ENLACES DE INTERÉS

- Aprender GIT... y de camino GitHub:
  - <https://github.com/oslugr/curso-git>
- Minitutorial de GIT:
  - <https://try.github.io/>
- Tutorial de GIT de Codecademy:
  - <https://www.codecademy.com/learn/learn-git>
- Curso de DevCode "Fundamentos de Git y GitHub":
  - <https://devcode.la/cursos/git/>



# 3 USO BÁSICO DE GIT

## 3.1 SISTEMA CONTROL DE VERSIONES

*"Sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante."*

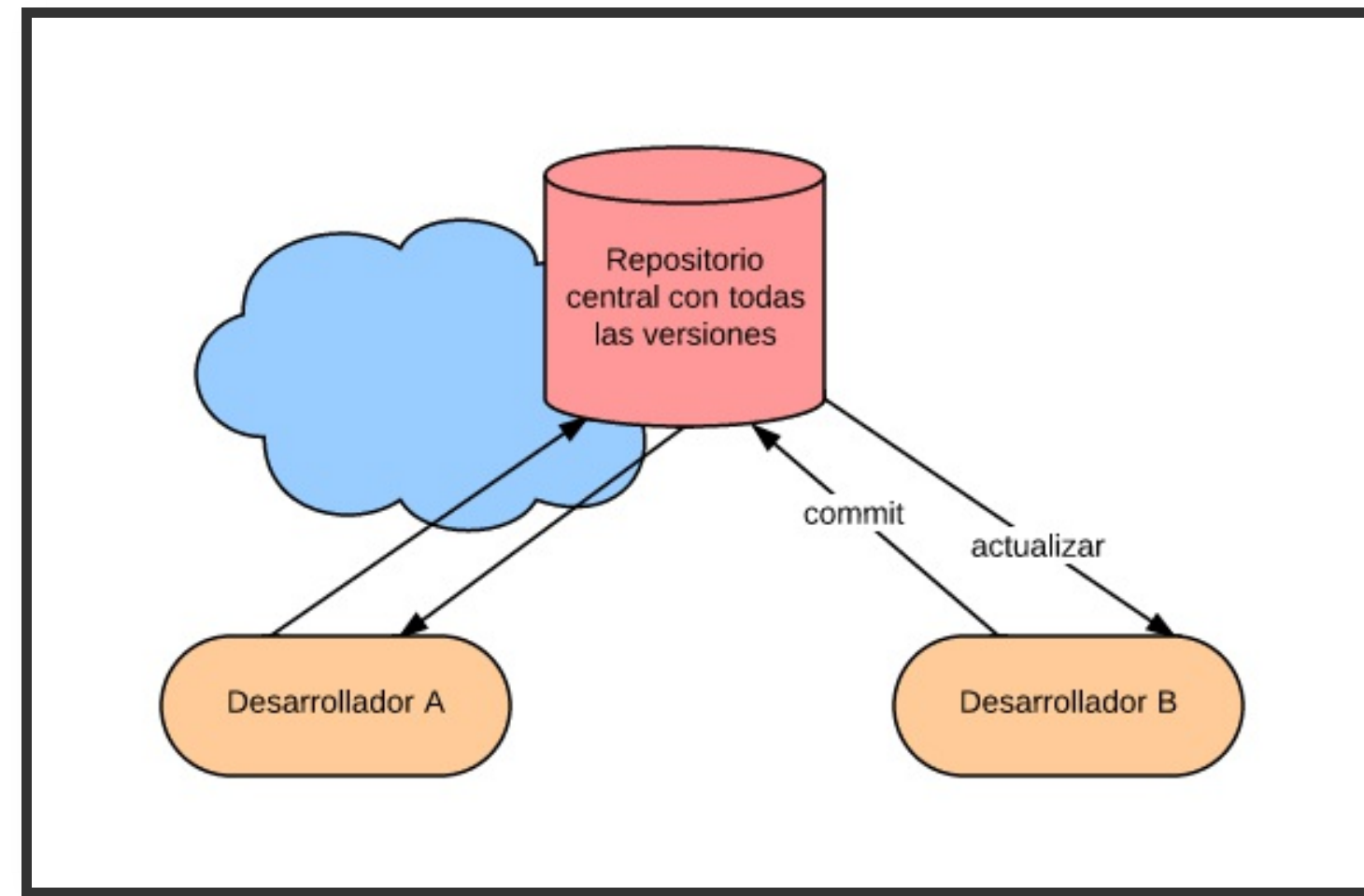
- <https://git-scm.com/book/es/v2/Empezando-Acerca-del-control-de-versiones>

## 3.2 VCS LOCALES

- **Lo más simple:** hacer copias de directorios.
- Aparecieron **BD en local** que guardan el registro de los cambios realizados a los archivos.

## 3.3 VCS CENTRALIZADOS

- Un **servidor central** que guarda los cambios.



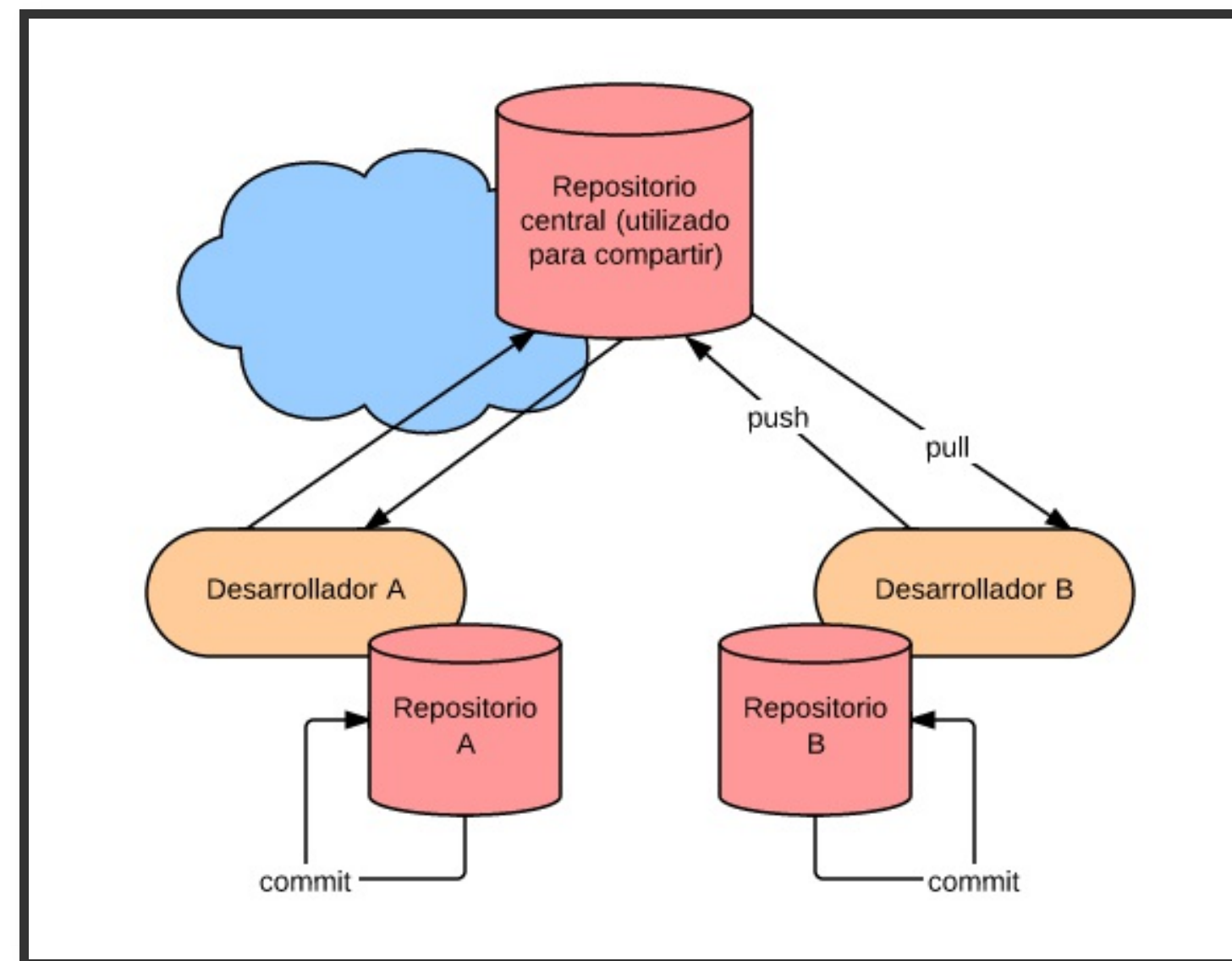
VCS Centralizado

## 3.4 PROS Y CONTRAS VCS CENTRALIZADOS

- **Pros:** más colaborativo que el local.
- **Contras:** dependes de un servidor central.

## 3.5 VCS DISTRIBUIDOS

- Cada cliente **no solo descarga la última copia, sino todo el repositorio.**



VCS Distribuido

## 3.6 VENTAJAS VCS DISTRIBUIDOS

- Puedes seguir trabajando aunque el repositorio remoto esté caído.
  - **más autonomía**
- La información está más replicada.
  - **menos vulnerable**
- Permite pruebas en local y subir solo lo relevante.
  - **más limpieza**

## 3.7 CARACTERÍSTICAS DE GIT

- Creado por **Linux Torvalds**, líder del equipo del kernel Linux.
- Objetivos cuando se creó:
  - **Rápido**
  - **Sencillo**
  - **Multi rama**
  - **Distribuido**
  - **Grandes proyectos**



## 3.8 INSTALACIÓN

- Windows: <https://git-scm.com/download/win>
- Mac: <https://git-scm.com/download/mac>
- Linux: <https://git-scm.com/download/linux>

## 3.9 CONFIGURACIÓN INICIAL

```
git config --global user.name "Nombre que quieras mostrar"
```

```
git config --global user.email "correo@electronico.es"
```

## 3.10 GUIs

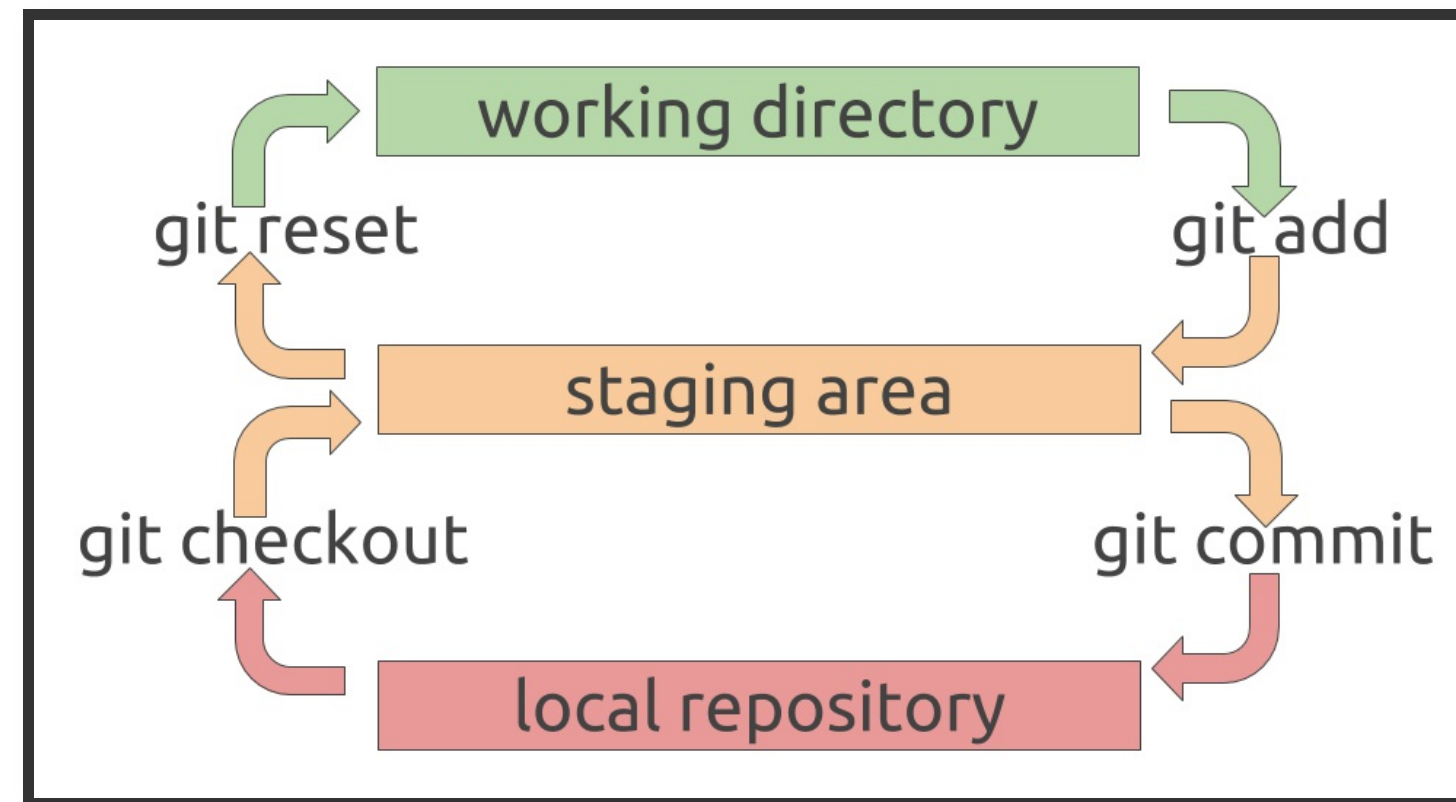
- <https://git-scm.com/downloads/guis>

## 3.11 INICIALIZAR UN REPOSITORIO

- Crea el **subdirectorio .git** con archivos de git para gestionar el repositorio.

```
git init
```

## 3.12 EL ÁREA DE STAGING



Staging Area

## 3.13 VER EL ESTADO DE LOS ARCHIVOS

- Importante saber el **estado** de los archivos.

```
git status
```

## 3.14 VER LAS DIFERENCIAS

- Podemos ver las **diferencias** entre el área de staging y el área de trabajo.

```
git diff
```

## 3.15 AÑADIR ARCHIVOS

- Podemos **añadir** los cambios de un fichero (o varios) al área de staging (desde el área de trabajo).

```
git add nombre-del-fichero
```

```
git add *.extension
```

```
git add -A
```



## 3.16 BORRAR ARCHIVOS

- Podemos **borrar archivos** del área de staging (también lo borrará del área de trabajo)

```
git rm nombre-del-fichero
```

## 3.17 MOVER/RENOMBRAR ARCHIVOS

- Podemos **mover/renombrar archivos** en el área de staging (también lo hará en el área de trabajo)

```
git mv antiguo-nombre-del-fichero nuevo-nombre-del-fichero
```

## 3.18 RESETEAR ARCHIVOS

- Para **resetear** los cambios de un fichero (o varios) al área de trabajo (desde el área de staging).

```
git reset nombre-del-fichero
```

## 3.19 GRABAR LOS CAMBIOS

- Para **grabar** los cambios realizados al repositorio (desde el área de staging).

```
git commit -m "mensaje corto descriptivo con los cambios"
```

## 3.20 DESHACER LOS CAMBIOS

- Para **deshacer** los cambios de un fichero (o varios) al área de staging (desde el repositorio).

```
git checkout nombre-del-fichero
```

## 3.21 LISTADO DE CAMBIOS

- Para ver el **listado de cambios** realizados en el repositorio.

```
git log
```

## 3.22 ALIAS

- Podemos crear **alias**.

```
git config --global alias.list 'log --oneline --decorate --graph --all'
```

## 3.23 IGNORAR ARCHIVOS

- Podemos ignorar archivos añadiéndolos al fichero **.gitignore**.



## 3.24 CREANDO ETIQUETAS

- Existen etiquetas **ligeras**, y etiquetas **anotadas** (iguales pero éstas con más información)

```
git tag nombre-etiqueta-ligera
```

```
git tag -a nombre-etiqueta-anotada -m "mensaje que acompaña a la etiqueta"
```

## 3.25 ETIQUETAS TARDÍAS

- Se puede crear una etiqueta **conociendo el hash del commit** (verlo con git log).

```
git tag -a nombre-etiqueta-annotada -m "mensaje que acompaña a la etiqueta" h
```

## 3.26 VER UNA ETIQUETA

- Podemos **ver información concreta de una etiqueta**.

```
git show nombre-etiqueta
```

## 3.27 SACAR UNA ETIQUETA

- No podemos sacar una etiqueta, pero podemos **colocar en nuestro directorio de trabajo una versión que coincida con alguna etiqueta, creando una rama nueva:**

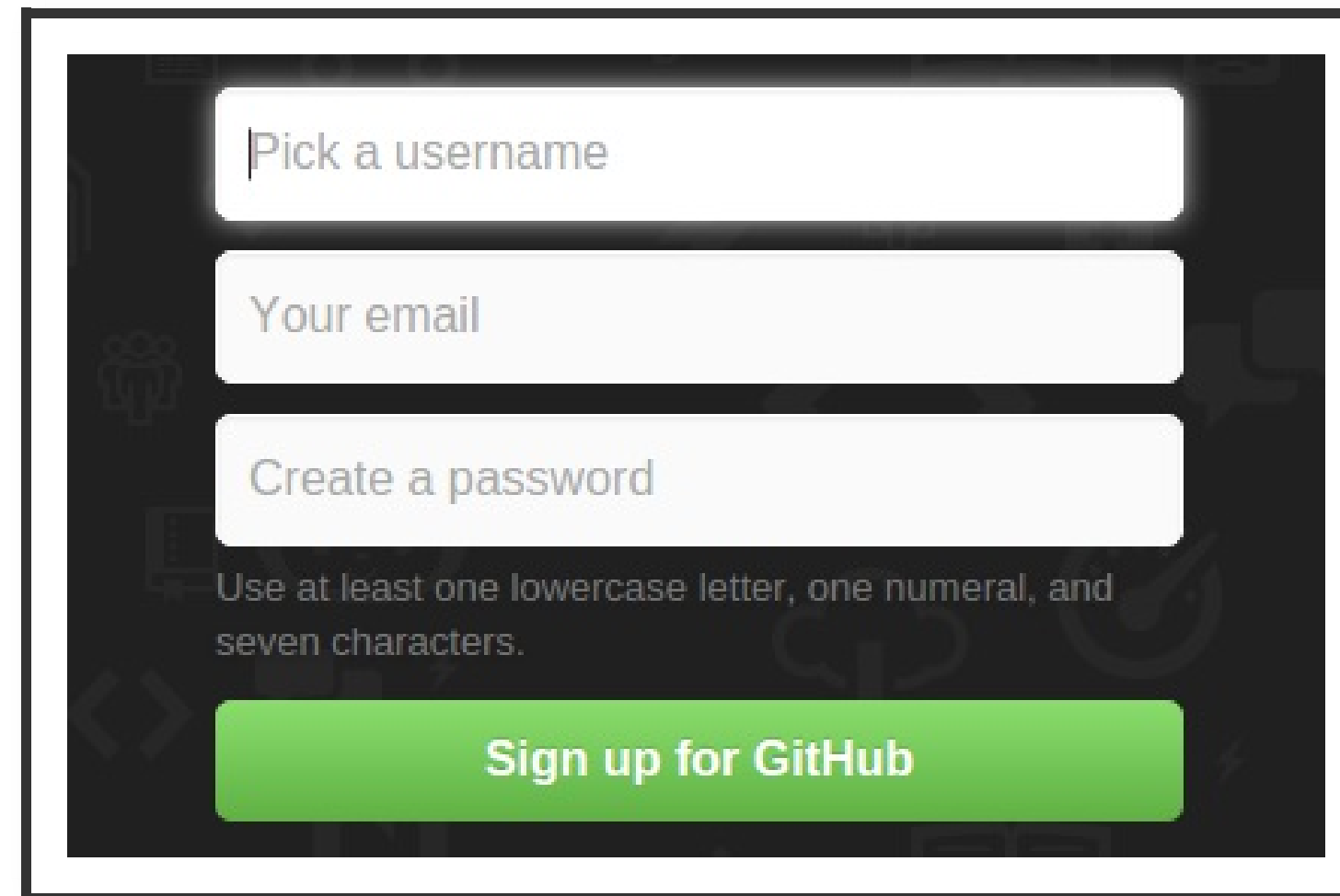
```
git checkout -b nombre-rama nombre-etiqueta
```

# 4 USO BÁSICO DE GITHUB

## 4.1 CARACTERÍSTICAS DE GITHUB

- **Plataforma de desarrollo colaborativo**, que utiliza Git.
- Los **repositorios son públicos**, salvo con cuenta de pago.
- Tiene facetas de **red social** (perfil público, seguidores, estrellas, etc.)
- Nos permite **gestionar organizaciones y equipos**.
- **Gestión de proyectos** (wiki, releases, incidencias, gráficos, etc.)
- **Servidor web**.

## 4.2 CREAR CUENTA

A screenshot of the GitHub sign-up form. The form is set against a dark background with a white border. It contains three white input fields stacked vertically. The first field has the placeholder text 'Pick a username'. The second field has the placeholder text 'Your email'. The third field has the placeholder text 'Create a password'. Below the third field, there is a line of text: 'Use at least one lowercase letter, one numeral, and seven characters.' At the bottom of the form is a green button with the text 'Sign up for GitHub' in white.

Pick a username

Your email

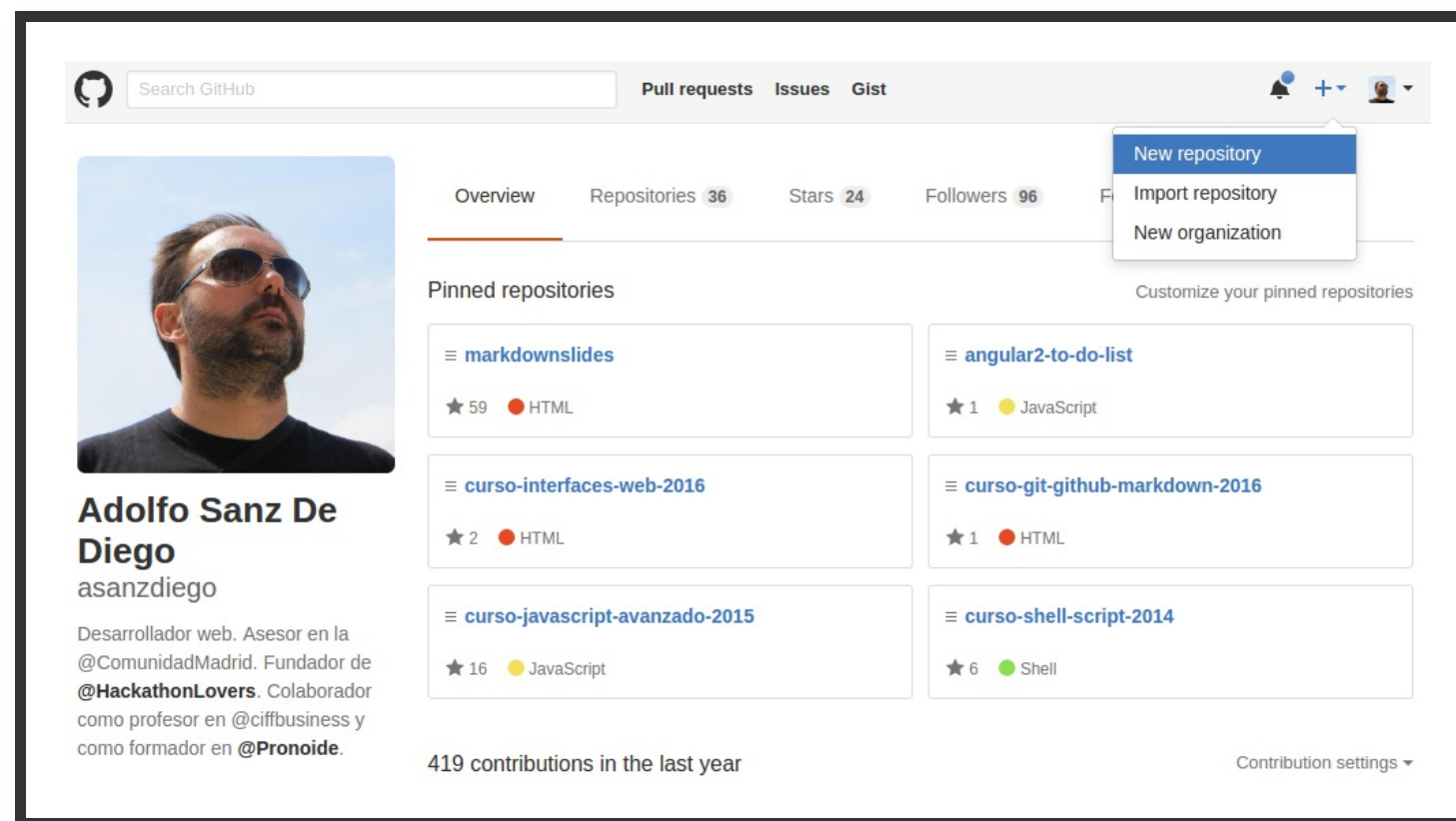
Create a password

Use at least one lowercase letter, one numeral, and seven characters.

Sign up for GitHub

Crear cuenta en GitHub

# 4.3 CREAR REPOSITORIO



## Crear un repositorio



## 4.4 CONFIGURAR CLAVES (I)

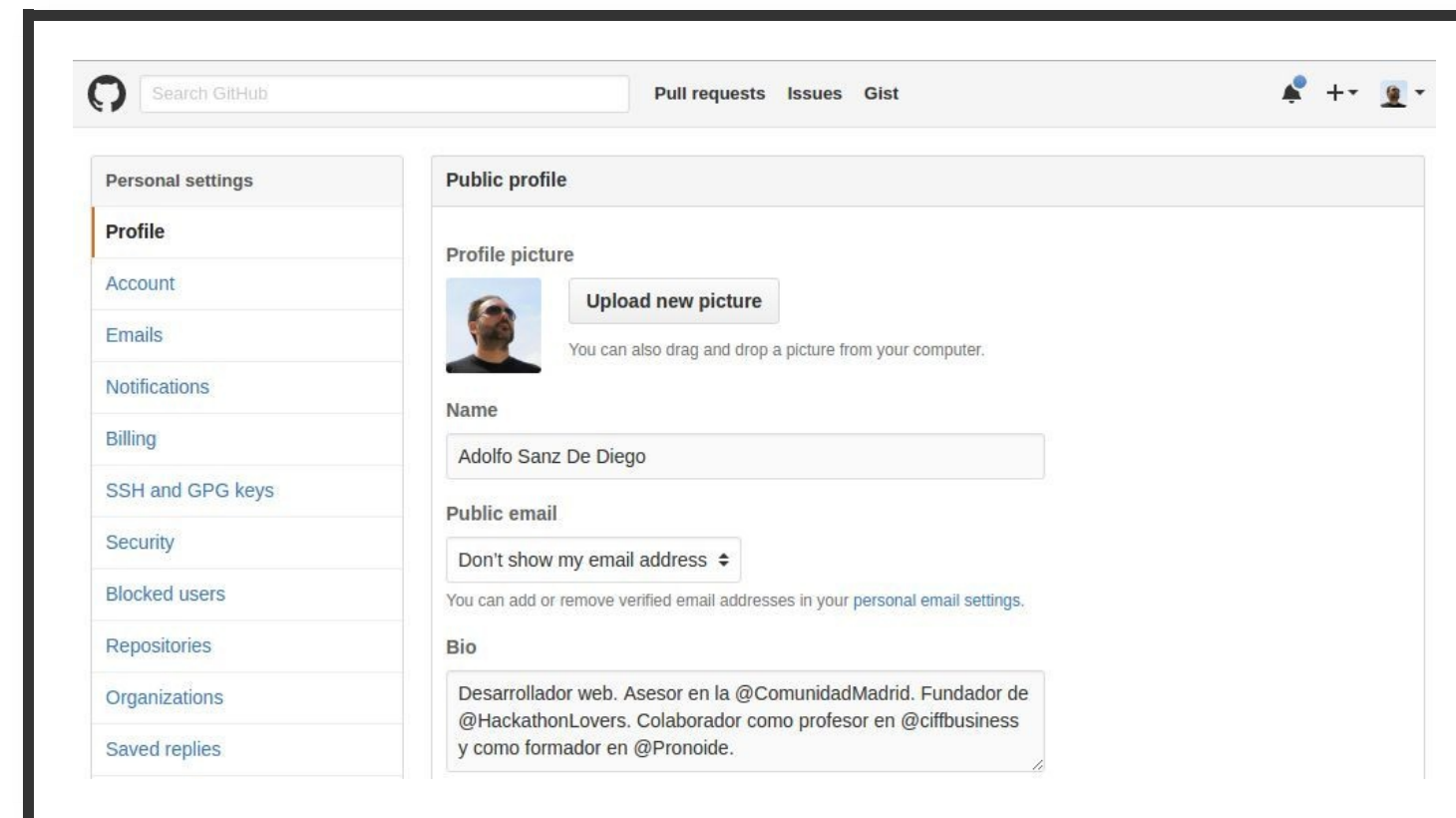
- Nos permite gestionar repositorios **mediante SSH** sin tener que estar poniendo siempre nuestra contraseña.
- Generamos en nuestro ordenador una clave SSH, que es un par de archivos, donde
  - uno es la **clave privada** (normalmente `~/.ssh/id_rsa`)
  - el otro es la **clave pública** (normalmente `~/.ssh/id_rsa.pub`)
- La **clave privada** la dejamos en nuestro ordenador.
- La **clave pública** la tenemos que registrar en nuestra cuenta de GitHub.

## 4.5 CONFIGURAR CLAVES (II)

- Podemos usar la clave SSH desde otros ordenadores (copiando la **clave privada**), aunque lo normal es usarla **solo con un ordenador**.
- Instrucciones:
  - <https://help.github.com/articles/generating-ssh-keys/>

# 4.6 CAMBIAR AVATAR

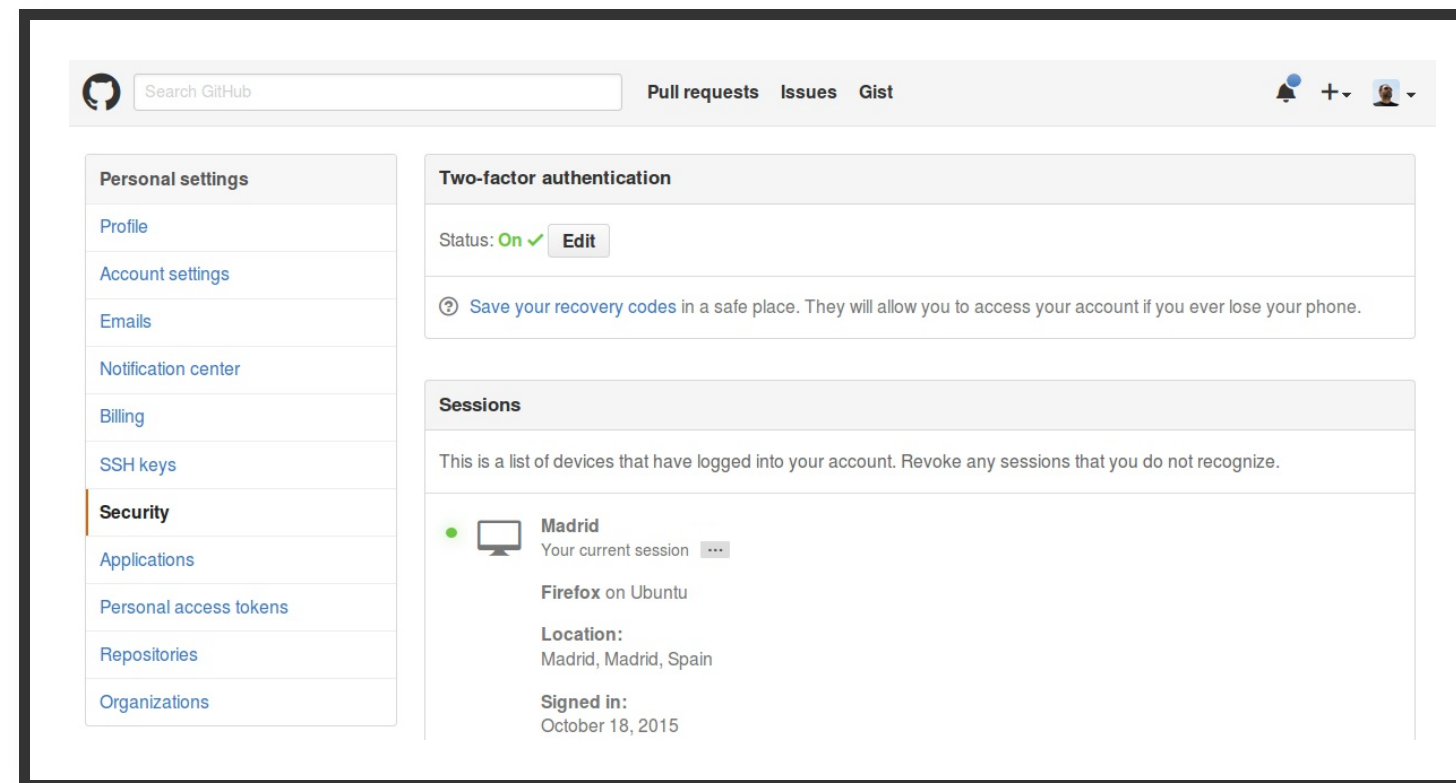
- View profile and more > Settings > Profile



Cambiar avatar en GitHub

# 4.7 DOBLE FACTOR DE AUTENTIFICACIÓN

- View profile and more > Settings > Security



Activar el doble factor de autenticación en GitHub

## 4.8 USO SOCIAL

- Características sociales:
  - Seguir a gente.
  - Seguir proyectos (watch).
  - Premiar proyectos (start).
  - *Forkear* proyectos (fork).
  - Crear organizaciones.

# **5 USO AVANZADO DE GIT**

## 5.1 CONECTAR UN REPOSITORIO REMOTO

- Podemos **conectar uno o varios repositorios remotos** a nuestro repositorio.

```
git remote add alias-repositorio-remoto url-repositorio-remoto
```

## 5.2 RENOMBRAR UN REPOSITORIO REMOTO

- Podemos **renombrar el alias de un repositorio remoto.**

```
git remote rename antiguo-alias nuevo-alias
```



## 5.3 DESCONECTAR UN REPOSITORIO REMOTO

- Podemos **desconectar un repositorio remoto**.

```
git remote remove alias-repositorio-remoto
```

## 5.4 VER LOS REPOSITORIOS REMOTOS

- Podemos **ver los repositorios remotos conectados y los permisos que tenemos.**

```
git remote -v
```

## 5.5 DESCARGAR CAMBIOS REMOTOS

- Podemos **descargar los cambios remotos sin modificar nuestro repositorio local.**

```
git fetch alias-repositorio-remoto
```

## 5.6 DESCARGAR Y COMBINAR

- Podemos **descargar y combinar los cambios remotos** con los de tu repositorio local.

```
git pull alias-repositorio-remoto nombre-rama-repositorio-remoto
```

## 5.7 ENVIAR DATOS

- Podemos **enviar datos al repositorio remoto** (solo si está up-to-date).

```
git push alias-repositorio-remoto nombre-rama-repositorio-remoto
```

## 5.8 REPOS Y RAMAS

- Normalmente:

```
git pull/push origin master
```

## 5.9 ENVIAR DATOS (III)

- Si queremos subir los tags:

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

## 5.10 CLONAR REPOSITARIOS

- Clonar es como:
  - hacer un init
  - luego un remote add
  - luego un fetch con alias=origin
  - dejando las ramas remota y local en master

```
git clone url-repositorio-remoto
```

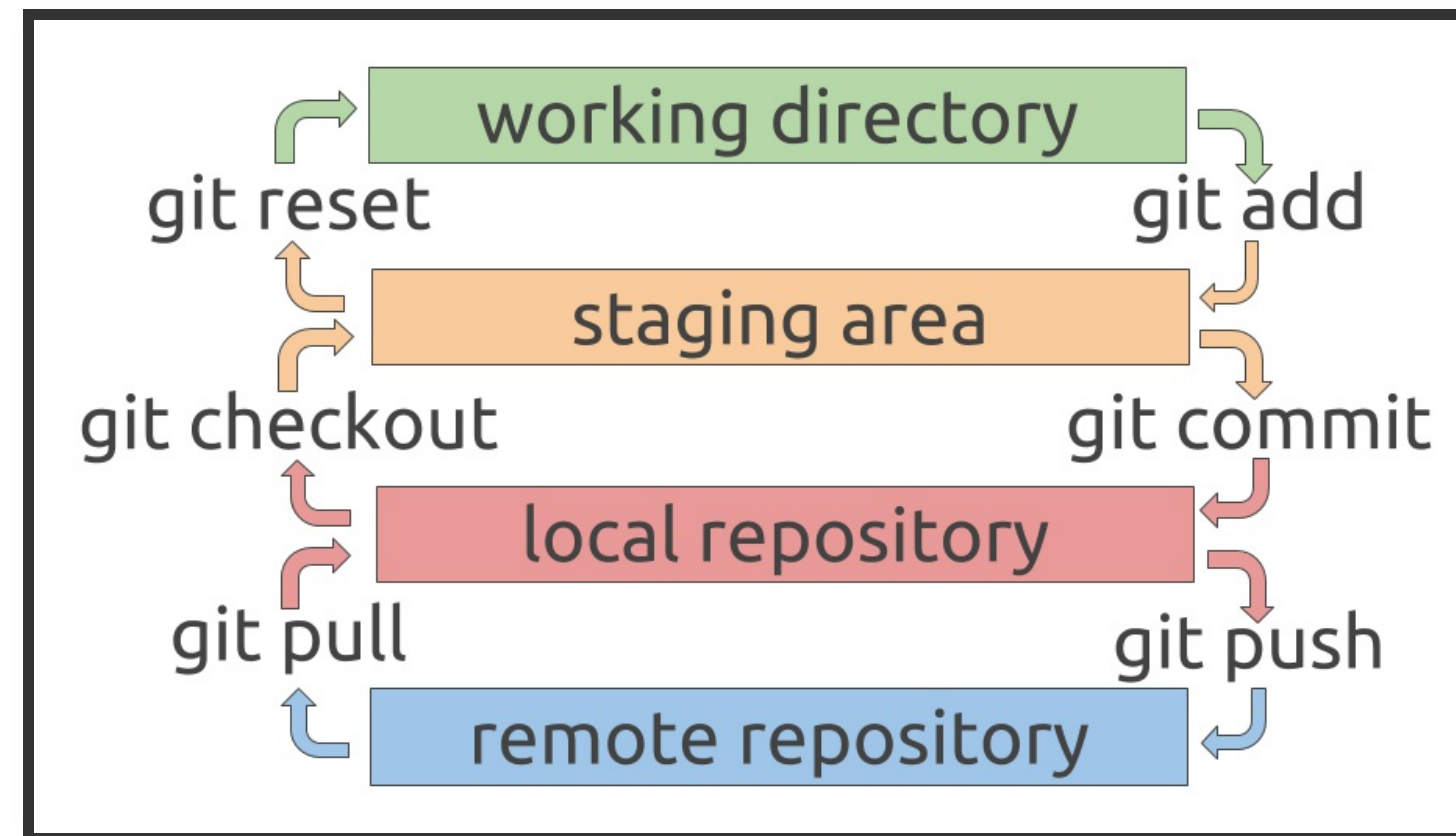


## 5.11 INSPECCIONAR REPOSITORIO REMOTO

- Podemos ver **información de un remoto particular, y como están configurados pull y push.**

```
git remote show alias-repositorio-remoto
```

## 5.12 RESUMEN ÁREAS



Resumen áreas GIT

## 5.13 CREAR UNA RAMA

- Podemos crear ramas que son **apuntadores que podemos mover por los distintos snapshots.**
- Solo la creamos, no nos situamos en ella.

```
git branch nombre-rama
```

## 5.14 CAMBIAR DE RAMA

- El HEAD es el apuntador que usa GIT para saber en qué rama estás.
- Cuando cambiamos de rama GIT **cambia el HEAD y los ficheros de tu área de trabajo.**

```
git checkout nombre-rama
```

## 5.15 CREAR Y CAMBIAR DE RAMA

- Podemos **crear y cambiar de rama** con un mismo comando.

```
git checkout -b nombre-rama
```

## 5.16 VER LAS RAMAS Y EL HEAD

- Podemos **ver las ramas y donde apunta el HEAD**.

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

```
git branch -v
```

## 5.17 FUSIONAR RAMAS

- GIT es **muy potente** con la fusión de ramas.

```
git merge nombre-rama
```

## 5.18 SOLUCIONAR CONFLICTOS

- Si al hacer un merge existan conflictos **GIT** los **apunta en los propios ficheros.**

```
<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=====
<div id="footer">please contact us at support@github.com</div>
>>>>>> issue:index.html
```



## 5.19 BORRAR RAMAS

- Una vez fusionado la rama en el master, **conviene borrarla** (solo nos deja si está fusionada).

```
git branch -d nombre-rama
```

## 5.20 LISTADO DE RAMAS POR ESTADO

- Podemos saber **qué ramas están fusionadas y cuáles no.**

```
git branch --merged
```

```
git branch --no-merged
```

## 5.21 SINCRONIZAR RAMA REMOTA

- Igual que sincronizamos la rama master remota, podemos **sincronizar otras ramas remotas**.

```
git checkout -b nombre-rama-local alias-repositorio-remoto/nombre-rama-remota
```

```
git checkout --track alias-repositorio-remoto/nombre-rama-remota
```

## 5.22 ASIGNAR RAMA REMOTA

- Podemos **asignar el área de trabajo a una rama remota.**

```
git checkout -u alias-repositorio-remoto/nombre-rama-remota
```

## 5.23 LISTADO DE TODAS LAS RAMAS

- Podemos listar no solo las ramas locales, sino **también las remotas.**

```
git branch -vv
```

## 5.24 ELIMINAR RAMA REMOTA

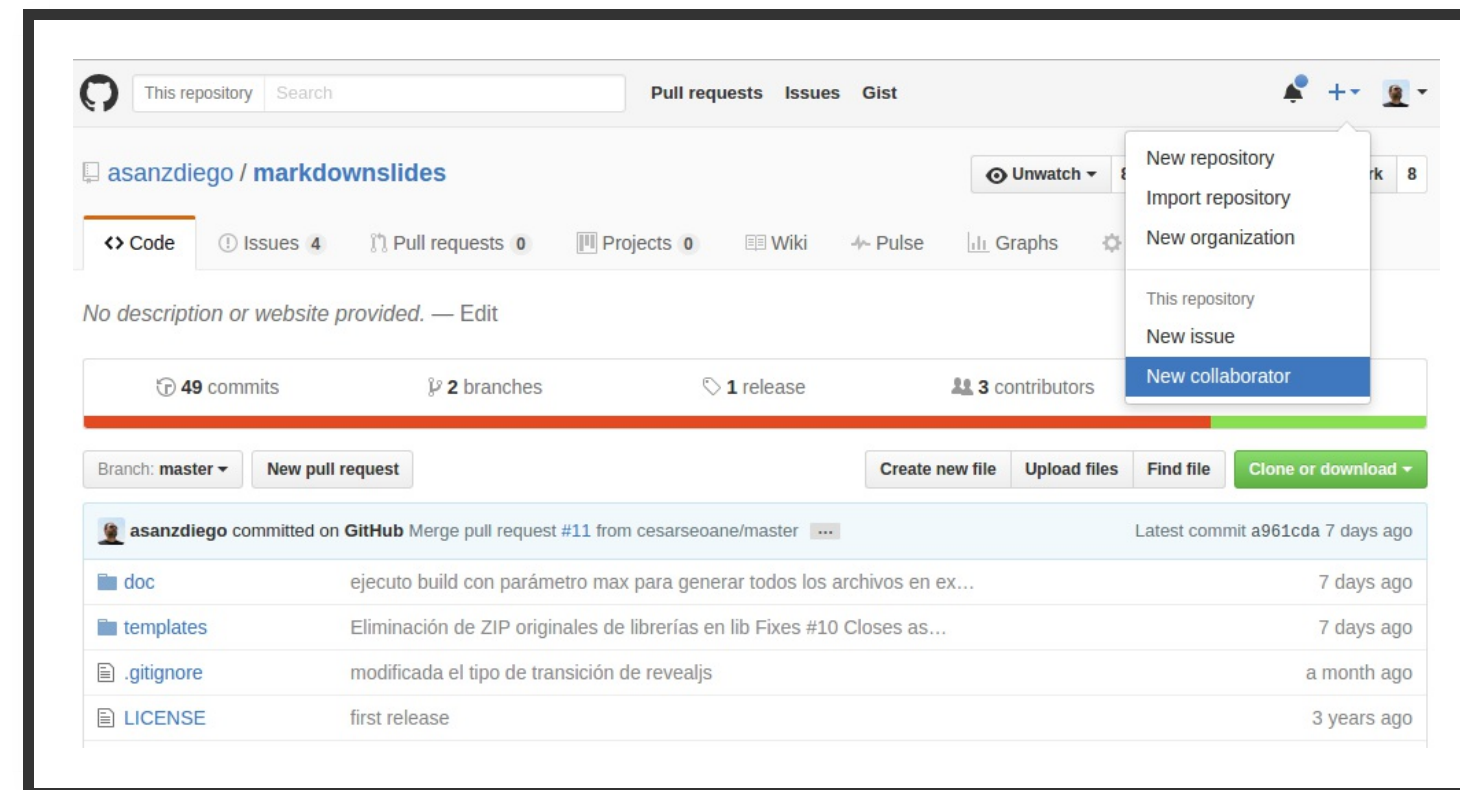
- Podemos **eliminar las ramas remotas**.

```
git push alias-repositorio-remoto --delete nombre-rama-remota
```

# **6 USO AVANZADO DE GITHUB**

# 6.1 AÑADIR COLABORADORES

- Podemos **dar permisos de push** a quien queramos.

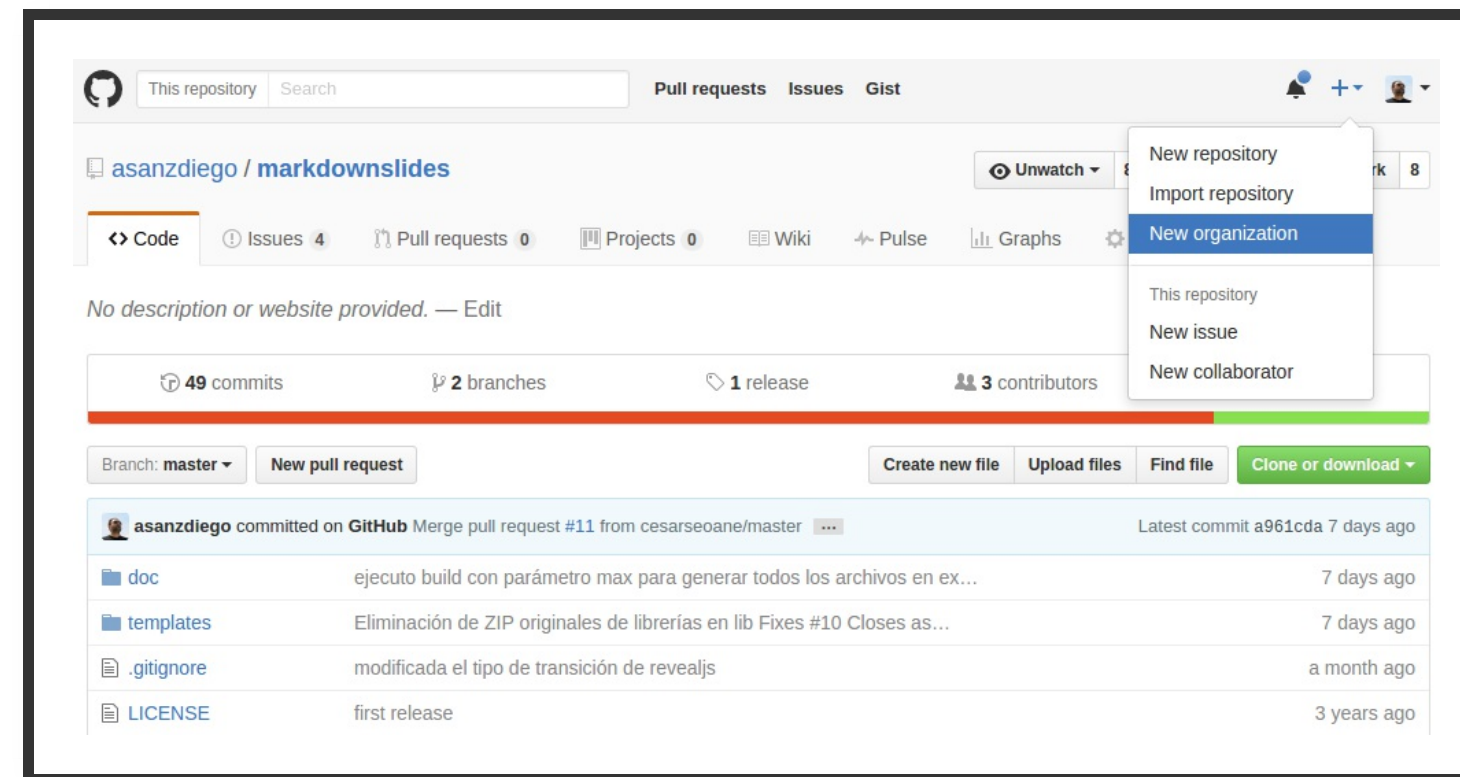


GitHub New Collaborator



# 6.2 CREAR ORGANIZACIONES

- Podemos **crear organizaciones**.



GitHub New Organization

## 6.3 GESTIONAR ORGANIZACIONES

- Dentro de las organizaciones podemos **crear equipos** y/o trabajar con colaboradores externos.
- El **nivel de permisos se gestiona a nivel de equipo**.
- Las personas tendrán los permisos de los equipos a los que pertenezca.
- Los permisos se otorgan a cada repositorio.

## 6.4 FORKEAR PROYECTOS

- Para **participar en un proyecto sin permisos de escritura**, puedes *forkearlo*.
- Consiste en crear una copia completa del repositorio bajo tu control: se encontrará **en tu cuenta** y podrás escribir en él sin limitaciones.

## 6.5 PULL-REQUESTS (I)

- Para **enviar propuestas de mejora**.
- Se usa mucho para proyectos que no son tuyos donde te gustaría colaborar.
- También se usa dentro de equipos para gestionar proyectos grandes.

## 6.6 PULL-REQUESTS (II)

1. Crear un fork de proyecto.
2. Clonar nuestro fork en nuestro equipo.
3. Crear una rama que sea descriptiva.
4. Realizar nuestros cambios.
5. Comprobar los cambios.
6. Enviar nuestra nueva rama de vuelta a nuestro fork.

## 6.7 PULL-REQUESTS (III)

1. Abrir un Pull Request en GitHub.
2. Participa en la discusión asociada.
3. Opcionalmente, se realizan nuevos commits.
4. El propietario del proyecto original cierra el Pull Request
  - bien fusionando la rama con tus cambios
  - o bien rechazándolos.

## 6.8 ISSUES Y WIKIS

- Todos los repositorios de GitHub tienen asociados:
  - un gestor de incidencias (issues)
  - una wiki para documentar

## 6.9 GITHUB PAGES

- Podemos tener **servidor web en los repositorios simplemente configurándolo:**
- Ver : <https://pages.github.com/>



## 6.10 FICHERO README.MD

- Nos **lo muestra renderizado** en la página del repositorio.

## 6.11 WEBHOOKS & SERVICES

- Para que GitHub pueda **interactuar con sistemas externos**.
- Los servicios están ya medio configurados.
- Si necesitas algo más específico lo tienes que hacer con webhooks, que lo que hace GitHub es hacer un POST a la URL que indiques cuando se lance algún evento (push, pull request, fork, etc.)

# 7 MARKDOWN

## 7.1 ¿QUÉ ES MARKDOWN?

*"Es un lenguaje de marcado ligero que trata de conseguir la máxima legibilidad y 'publicabilidad' usando texto plano."*

- <https://es.wikipedia.org/wiki/Markdown>

## 7.2 CARACTERÍSTICAS PRINCIPALES

- Texto plano
- Sintaxis sencilla
- Legibilidad
- Publicabilidad
- Exportabilidad

## 7.3 MARKDOWNSLIDES

- <https://github.com/asanzdiego/markdownslides>

## 7.4 CHULETA DE MARKDOWN

- <http://warpedvisions.org/projects/markdown-cheat-sheet>

## 7.5 EDITOR ONLINE

- <https://jbt.github.io/markdown-editor/>



## 7.6 ENCABEZADOS (I)

- <h1>, <h2>, <h3>

```
# Encabezado de primer nivel
```

```
## Encabezado de segundo nivel
```

```
## Encabezado de tercer nivel
```

## 7.7 ENCABEZADOS (II)

- Equivalente a lo anterior.

```
Encabezado de primer nivel  
=====
```

```
Encabezado de segundo nivel  
-----
```

```
## Encabezado de tercer nivel ##
```

## 7.8 LISTAS NO NUMERADAS

- No enumeradas:
  - se puede usar el menos
  - se puede usar el asterisco
  - se puede usar el más

```
- se puede usar el menos  
* se puede usar el asterisco  
+ se puede usar el más
```

## 7.9 LISTAS NUMERADAS

- Enumeradas:
  1. Primer elemento
  2. Segundo elemento
  3. Tercer elemento

```
1. Primer elemento  
1. Segundo elemento  
1. Tercer elemento
```

## 7.10 FORMATO (NEGRITA, CURSIVA, TACHADO)

- Texto en cursiva con *un asterisco* o con *un guion bajo*.
- Texto en negrita con **dos asteriscos** o con **dos guiones bajos**.
- Texto tachado con ~~dos virgulillas~~.

- Texto negrita con `**dos asteriscos**` o con `__dos guiones bajos__`.
- Texto cursiva con `*un asterisco*` o con `_un guion bajo_`.
- Texto tachado con `~~dos virgulillas~~`.

# 7.11 TABLAS

Header	Header	Right
Cell	Cell	\$10
Cell	Cell	\$20

Header	Header	Right
Cell	Cell	\$10
Cell	Cell	\$20

## 7.12 CITAS

*"No hay camino hacia el Software Libre, el Software Libre es el camino"*

```
> "No hay camino hacia el Software Libre,  
el Software Libre es el camino"
```

## 7.13 CÓDIGO

```
require(maps) # activación de librería  
require(mapproj) # se usará para projection="polyconic"  
# Cargar los datos  
# unemp incluye datos para condados de los Estados Unidos continentales.  
data(unemp) # Datos de desempleo  
data(county.fips) # mapa de los condados
```

```
require(maps) # activación de librería  
require(mapproj) # se usará para projection="polyconic"  
# Cargar los datos  
# unemp incluye datos para condados de los Estados Unidos continentales.  
data(unemp) # Datos de desempleo  
data(county.fips) # mapa de los condados
```



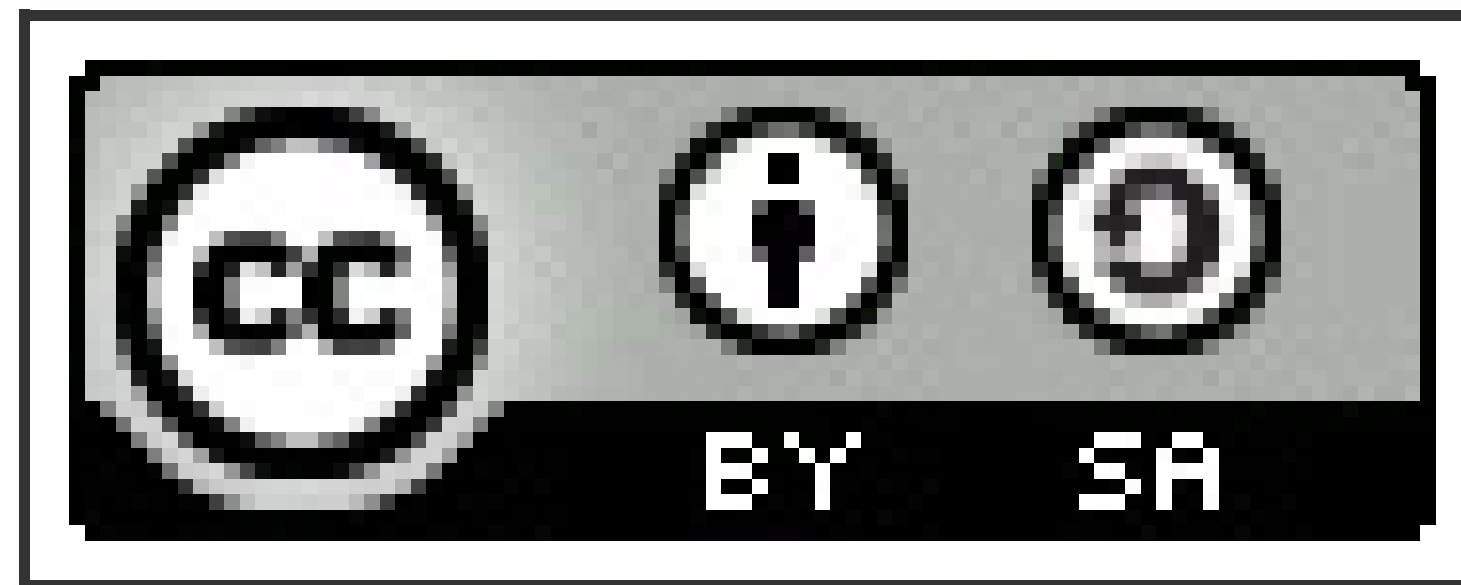
## 7.14 ENLACES

- Enlace con texto
- Enlace sencillo: -<https://github.com/asanzdiego/curso-git-github-markdown-2015>

```
- [Enlace con texto](https://github.com/asanzdiego/curso-git-github-markdown-2015)
- Enlace sencillo:
  -<https://github.com/asanzdiego/curso-git-github-markdown-2015>
```

## 7.15 IMÁGENES

- Esta obra está bajo una licencia:



Creative Commons BY SA

- Esta obra está bajo una licencia:

```
![[Creative Commons BY SA](../img/git/cc-by-sa.png){ width=50% text-align=cent
```