

Introducción al control de versiones y trabajo colaborativo con GitHub

Oscar Perpiñán Lamigueiro

AIGORA

Lo expuesto en este documento se enmarca en el proyecto de innovación educativa AIGORA (Aprendizaje de Informática con Github Organizado en Repositorios Abiertos).



<https://innovacioneducativa.upm.es/proyectosIE/informacion?anyo=2018-2019&id=2840>

① Conceptos básicos

② Uso de `git` y GitHub

③ Trabajo en colaboración

④ GitHub Classroom

⑤ Publicación de páginas web en GitHub

- ① Conceptos básicos
 - ¿Qué es el control de versiones?
 - ¿Qué son Git y GitHub?
- ② Uso de git y GitHub
- ③ Trabajo en colaboración
- ④ GitHub Classroom
- ⑤ Publicación de páginas web en GitHub

"FINAL".doc



↖ FINAL.doc!



↖ FINAL_rev.2.doc

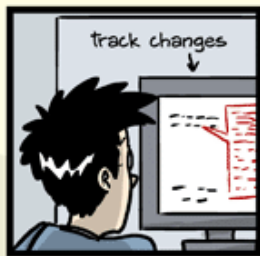


FINAL_rev.6.COMMENTS.doc



FINAL_rev.8.comments5.
CORRECTIONS.doc

<http://phdcomics.com/comics/archive.php?comid=1531>



FINAL_rev.18.comments7.
corrections9.MORE.30.doc



FINAL_rev.22.comments49.
corrections.10.#@\$%WHYDID
ICOMETOGRADSCHOOL?????.doc



¿Qué es el control de versiones y por qué debería importarte?

*El control de versiones es un sistema que **registra los cambios** realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que se puedan **recuperar** versiones específicas más adelante.¹*

Viajar en el tiempo

- ▶ Nada que haya sido sometido a un control de versiones se pierde jamás (*salvo que realmente quieras eliminarlo...*)
- ▶ **Todas** las versiones antiguas de un fichero se almacenan: un fichero se puede revertir a un estado anterior sin límites.

¹<https://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones>

¿Qué es el control de versiones y por qué debería importarte?

*El control de versiones es el **cuaderno de laboratorio** en el mundo digital. Es lo que los profesionales usan para realizar un **seguimiento** de lo que han hecho y para **colaborar** con otras personas.²*

¿Qué? ¿Cuándo? ¿Quién?

Un sistema de control de versiones registra:

- ▶ El detalle de los cambios realizados.
- ▶ La fecha y hora en la que fueron realizados.
- ▶ La persona que los realizó.

²<https://swcarpentry.github.io/git-novice/>

¿Qué es el control de versiones y por qué debería importarte?

*El control de versiones es el **cuaderno de laboratorio** en el mundo digital. Es lo que los profesionales usan para realizar un **seguimiento** de lo que han hecho y para **colaborar** con otras personas.²*

Trabajo Colaborativo

- ▶ Cuando un equipo de personas trabaja conjuntamente en un proyecto, es posible que se produzcan cambios incompatibles en un mismo fichero.
- ▶ El sistema de control de versiones **impide** cambios simultáneos en un fichero. A cambio, permite la **resolución de conflictos** y los documenta.

¿Qué es el control de versiones y por qué debería importarte?

***No sirve sólo para software:** libros, documentos, pequeños conjuntos de datos y cualquier cosa que cambie con el tiempo o que deba compartirse puede y debe almacenarse en un sistema de control de versiones.²*

① Conceptos básicos

¿Qué es el control de versiones?

¿Qué son Git y GitHub?

② Uso de git y GitHub

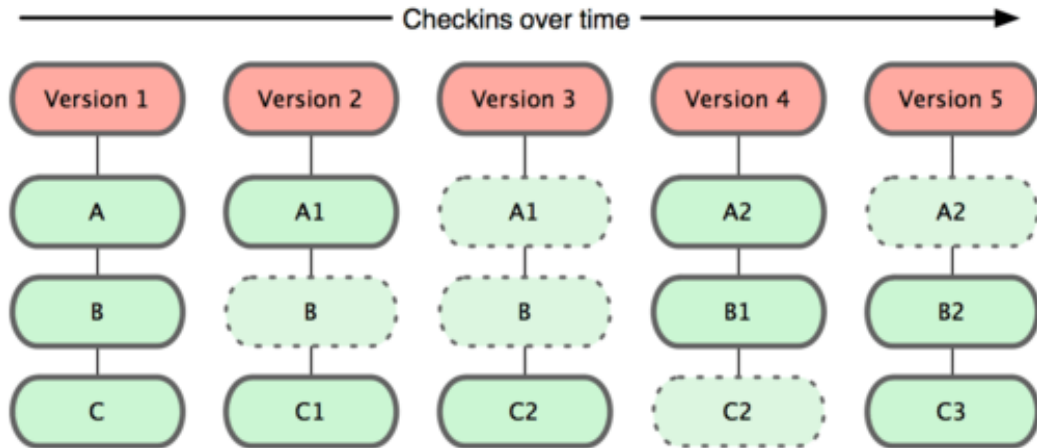
③ Trabajo en colaboración

④ GitHub Classroom

⑤ Publicación de páginas web en GitHub

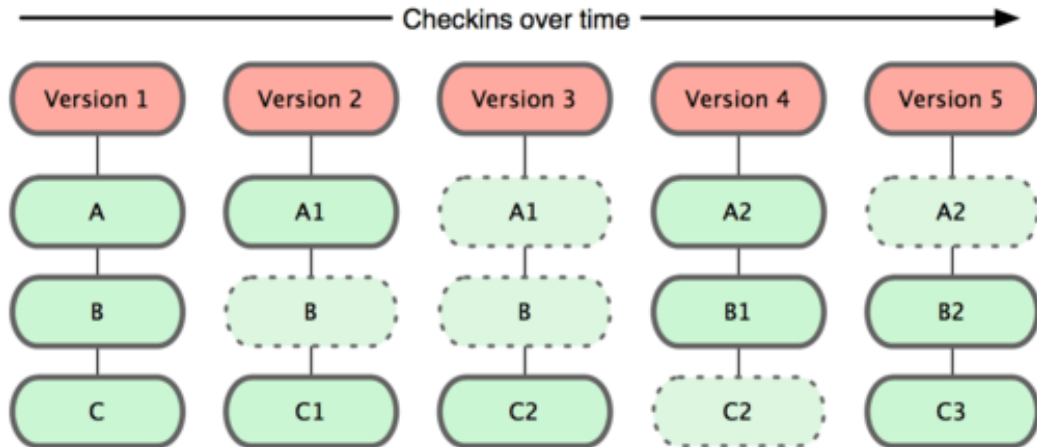
Git es un Sistema de Control de Versiones

Git es una herramienta software (accesible mediante línea de comandos con git) que implementa un Sistema de Control de Versiones.



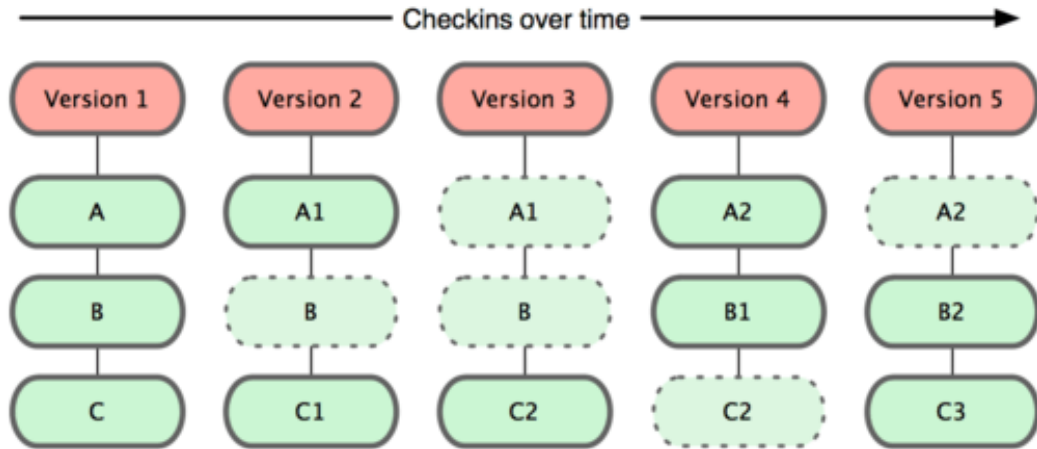
Git es un Sistema de Control de Versiones

Cada vez que se ejecuta un cambio en una estructura de ficheros controlada con Git, realiza una «foto» del estado de los archivos en ese momento, y guarda una referencia a esa instantánea.



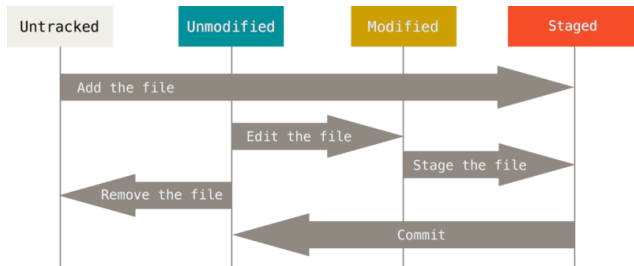
Git es un Sistema de Control de Versiones

Por eficiencia, Git no almacena los archivos sin modificaciones sino un enlace al archivo anterior idéntico que ya está almacenado



Los estados de Git

- ▶ El desarrollador incorpora uno o varios ficheros al control de versiones. (*tracked*)
- ▶ Realiza modificaciones en los ficheros (*modified*).
- ▶ Incorpora esos ficheros modificados al área de preparación (*staged*).
- ▶ Finalmente, confirma todos los cambios del área de preparación: se realiza la instantánea de los ficheros. (*committed*)



¿Qué es GitHub?

- ▶ GitHub es la plataforma de alojamiento de código más importante a nivel mundial.
- ▶ Emplea el sistema de control de versiones `git`
- ▶ Ofrece una amplia variedad de funcionalidades
 - ▶ Alojamiento de código
 - ▶ Revisión de código
 - ▶ Trabajo colaborativo
 - ▶ Publicación de páginas web

① Conceptos básicos

② Uso de git y GitHub

③ Trabajo en colaboración

④ GitHub Classroom

⑤ Publicación de páginas web en GitHub

① Conceptos básicos

② Uso de git y GitHub

Primeros Pasos

Creación y sincronización de repositorios

Flujo de Trabajo

Recuperación de cambios

③ Trabajo en colaboración

④ GitHub Classroom

⑤ Publicación de páginas web en GitHub

Itinerario

- ① Abriremos una cuenta en GitHub³.
- ② Instalaremos GitHub Desktop.
- ③ Configuraremos GitHub Desktop para conectarlo con la cuenta de GitHub.


³Es recomendable emplear el correo electrónico de la UPM (será útil más tarde, al usar GitHub Classroom).


Creación de una cuenta en GitHub


<https://github.com/join>

Join GitHub

The best way to design, build, and ship software.

 **Step 1:**
Set up your account

 **Step 2:**
Choose your plan

 **Step 3:**
Tailor your experience

Create your personal account

Username *

This will be your username. You can add the name of your organization later.

Email address *

We'll occasionally send updates about your account to this inbox. We'll never share your email address with anyone.

Password *

Make sure it's **more than 15 characters** OR **at least 8 characters including a number and a lowercase letter**. [Learn more](#).

Verify account

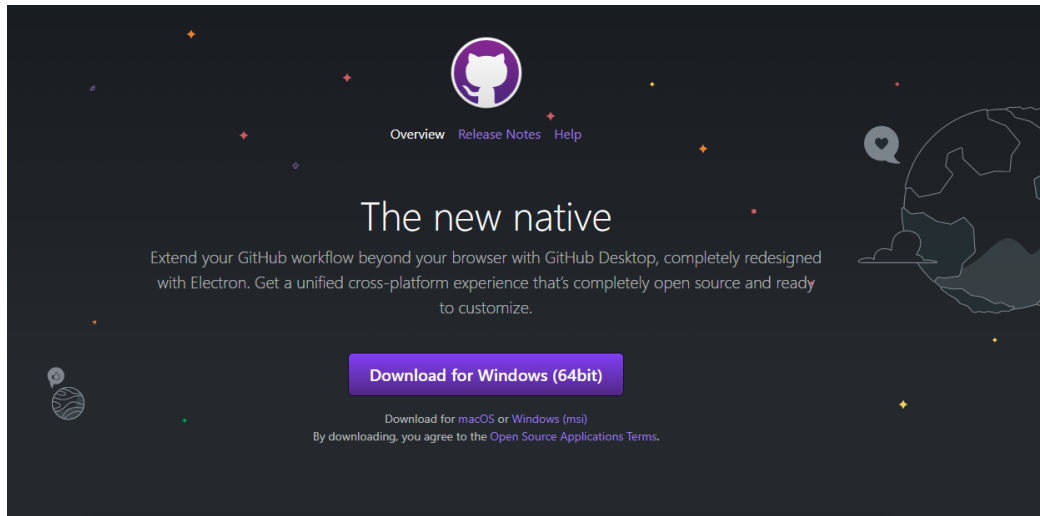
You'll love GitHub

- Unlimited** public repositories
- Unlimited** private repositories
- ✓ Limitless collaboration
- ✓ Frictionless development
- ✓ Open source community

Más información en [New GitHub account](#)

Instalación de GitHub Desktop

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



Conectamos Git, GitHub y GitHub Desktop

- ▶ Una vez instalado comienza el proceso de autenticación, usando las credenciales del paso anterior⁴.

File > Options > Accounts > Sign In

- ▶ A continuación, conectamos la información de usuario con Git⁵.

File > Options > Git

⁴Más información en [Authenticating to GitHub](#).

⁵Más información en [Configuring Git](#).

git en línea de comandos

- ▶ `git` tiene un elevado número de funcionalidades. Sólo algunas están implementadas en GitHub Desktop.
- ▶ Para poder acceder al conjunto completo es necesario instalar `git` y utilizar línea de comandos:
 - ▶ [Git for Windows](#)
 - ▶ [Git for Linux](#)
 - ▶ [Git for Mac](#)

① Conceptos básicos

② Uso de git y GitHub

Primeros Pasos

Creación y sincronización de repositorios

Flujo de Trabajo

Recuperación de cambios

③ Trabajo en colaboración

④ GitHub Classroom

⑤ Publicación de páginas web en GitHub

Remoto y Local

- ▶ **Github.com** aloja los **repositorios remotos** (*nube*).
- ▶ En tu ordenador trabajas con una **copia local** del repositorio. Otros desarrolladores tendrán sus propias copias locales.
- ▶ La(s) copia(s) local(es) y el repositorio remoto deben estar **sincronizados** mediante diferentes comandos de `git`.

Decisiones al crear un repositorio

► Portada

Debes inicializar y cumplimentar el fichero [README.md](#). Para el formato veáse [Formatting syntax](#).

► Licencia

Elige una adecuada a tu proyecto y a tus intereses actuales y futuros. Veáse <https://choosealicense.com>.

► Ficheros ignorados

Elige un [.gitignore](#) adecuado al proyecto: Veáse <https://github.com/github/gitignore>.

Método 1: de remoto a local

Itinerario

- ① Crearemos un nuevo repositorio **remoto**.
- ② Clonaremos este repositorio para tener una copia local.

Método 1: de remoto a local


Abrimos un nuevo repositorio *remoto*: <https://github.com/new>

Create a new repository

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

Owner

Repository name *

 oscarperpinan ▾ /

Great repository names are short and memorable. Need inspiration? How about **refactored-spoon**.

Description (optional)



Public

Anyone can see this repository. You choose who can commit.



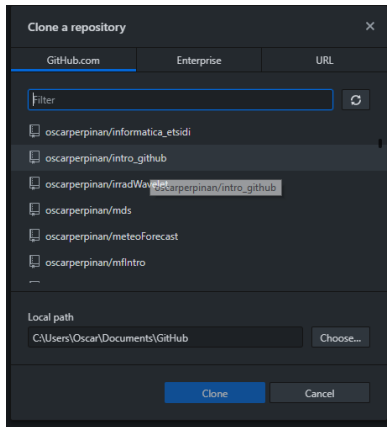
Private

You choose who can see and commit to this repository.

Método 1: de remoto a local

Clonamos el *repositorio remoto* para poder trabajar con él (*copia local*).

File > Clone Repository



Método 2: de local a remoto

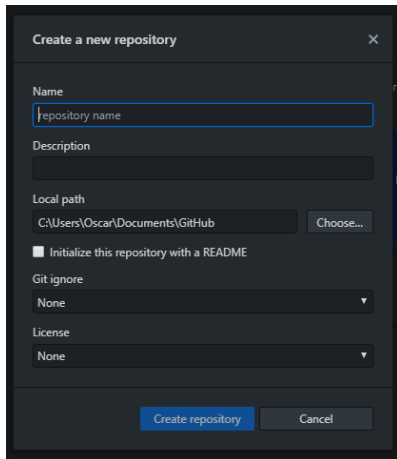
Itinerario

- ① Crearemos un nuevo repositorio **local**.
- ② Lo publicaremos en remoto para que sea accesible desde otros equipos.

Método 2: de local a remoto

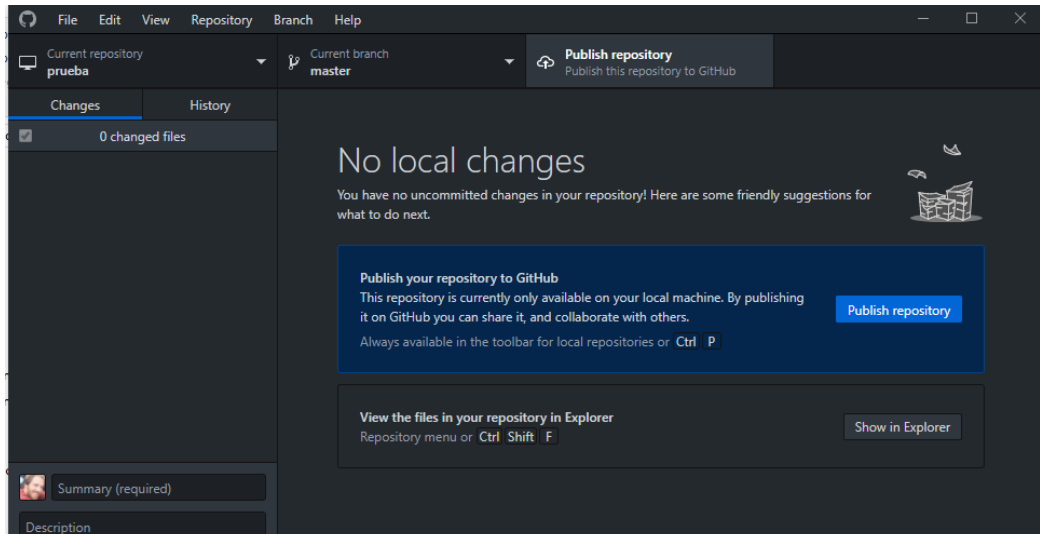
Abrimos un nuevo repositorio *local* desde GitHub Desktop

File > New Repository



Método 2: de local a remoto

Publicamos el *repositorio local* github.com (*repositorio remoto*) para que otros puedan verlo y trabajar con él.



① Conceptos básicos

② Uso de git y GitHub

Primeros Pasos

Creación y sincronización de repositorios

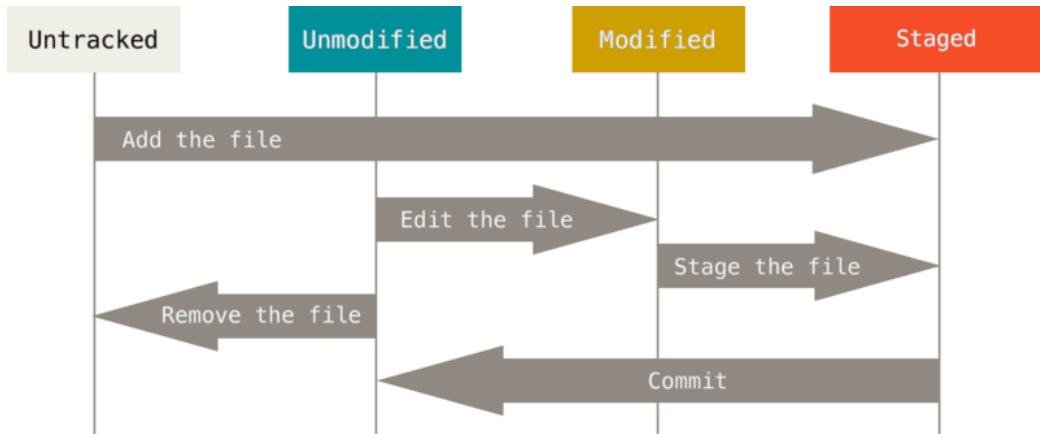
Flujo de Trabajo

Recuperación de cambios

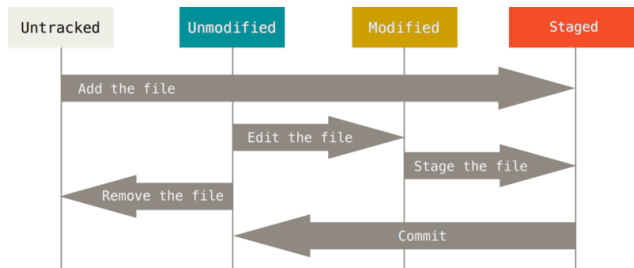
③ Trabajo en colaboración

④ GitHub Classroom

⑤ Publicación de páginas web en GitHub

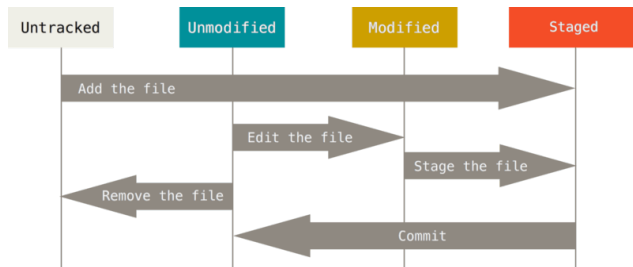


Cambios en la copia local



En la carpeta que contiene la copia local, haz **modificaciones** en los ficheros (empleando tu editor de código/texto preferido).

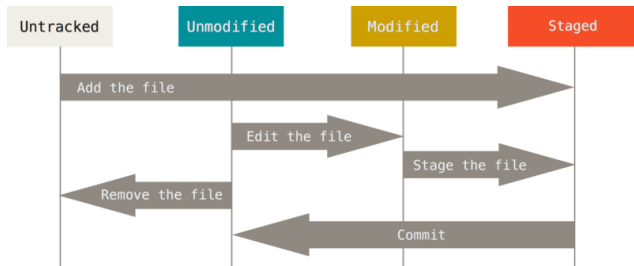
Cambios en la copia local



Añade los cambios realizados a la siguiente «instantánea» del repositorio (`git add`)

Changes ●		History	README.md	
✓	1 changed file			@@ -1 +1,3 @@
✓	README.md	□	1	-# prueba1819
			1	+# Esta es una prueba
			2	+
			3	+Aquí debe ir la descripción del proyecto.

Cambios en la copia local



Confirma los cambios, escribiendo un resumen de lo realizado (`git commit`)

Interfaz de confirmación de cambios (commit) de GitHub:

- Nombre del commit: `Añade texto a README`
- Descripción de los cambios:

```
Modifico el título de README.
Añado algo de contenido en el cuerpo.
```
- Botón: **Commit to master**
- Estado: Committed 24 minutes ago, Initial commit
- Botón: **Undo**

Consejos para commit

- ▶ *Commit early and often*: cada commit debe incluir cambios pequeños y coherentes.
- ▶ Escribir un mensaje de calidad al ejecutar cambios facilita tanto el trabajo personal como la colaboración en equipo con Git.⁶
- ▶ El **título** debe ser **conciso** (50 caracteres) y escrito en imperativo (*Do something...*)
- ▶ El **cuerpo** debe explicar **el qué y el por qué del cambio**, no el cómo, comparando con el comportamiento anterior al cambio.
- ▶ Se pueden incluir referencias a *issues* (ver a continuación) con #XX siendo XX el número de la *issue*.

⁶<https://chris.beams.io/posts/git-commit/>

Histórico de cambios

Los cambios confirmados con `commit` se anotan en la historia (`git log`)

View > History

Changes	History
No branches to compare	
Añade texto a README Oscar Perpiñán Lamigueiro committed 7 ...	Añade texto a README Oscar Perpiñán Lamigueiro committed eb5a029 1 changed file Modifico el título de README. Añado algo de contenido en el cuerpo.
Initial commit Oscar Perpiñán Lamigueiro committed 3...	README.md @@ -1 +1,3 @@ 1 -# prueba1819 2 + 3 +Aquí debe ir la descripción del proyecto.

Publicar cambios al repositorio remoto

- ▶ Para sincronizar los cambios realizados **desde la copia local hasta el repositorio remoto** hay que publicar mediante `git push`.

Repository > Push

- ▶ A partir de este punto, la copia local y el repositorio remoto están sincronizados.

Recibir cambios de un repositorio remoto

Para obtener en la copia local los cambios recientes que existan en el repositorio hay que emplear `git pull`, que es la combinación de la secuencia:

- 1 `git fetch`, para obtener los cambios recientes del repositorio remoto.
- 2 `git merge`, para combinarlos con la copia local.

Repository > Pull

Importante

En el caso de **repositorios compartidos**, antes de un `git push` es imprescindible actualizar la copia local incorporando los cambios del repositorio con `git pull`.

Resumen

- ① Realiza modificaciones en los ficheros de la copia local.
- ② **COMMIT** :: Confirma los cambios con un mensaje informativo.
- ③ **PULL** :: Incorpora los cambios del repositorio remoto a la copia local.
- ④ **PUSH** :: Publica los cambios de la copia local al repositorio remoto.

① Conceptos básicos

② Uso de git y GitHub

Primeros Pasos

Creación y sincronización de repositorios

Flujo de Trabajo

Recuperación de cambios

③ Trabajo en colaboración

④ GitHub Classroom

⑤ Publicación de páginas web en GitHub

Recordatorio

*El control de versiones es un sistema que **registra los cambios** realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que se puedan **recuperar** versiones específicas más adelante.*

Viajar en el tiempo

- ▶ Nada que haya sido sometido a un control de versiones se pierde jamás (*salvo que realmente quieras eliminarlo...*)
- ▶ **Todas** las versiones antiguas de un fichero se almacenan: un fichero se puede revertir a un estado anterior sin límites.

git revert

- ▶ Con `git revert` podemos revertir **todos** los cambios realizados en un determinado `commit`.
- ▶ Al hacer uso de este comando no se elimina el `commit` de la historia.
- ▶ Es más, se genera un **nuevo commit**, que se añade a la historia de cambios, y que debe ser sincronizado con el repositorio.
- ▶ **GitHub Desktop**
 - ▶ Se elige el `commit` que se quiere revertir.
 - ▶ Al pulsar botón derecho, en el menú contextual se elige *Revert changes in this commit*.

git checkout

- ▶ Con `git checkout` también se pueden deshacer cambios, y se puede elegir restringirlos a un fichero determinado.
- ▶ Esta funcionalidad de `git` no está disponible en GitHub Desktop.
- ▶ Con línea de comandos se consigue a través de `git checkout`⁷.

⁷<https://git-scm.com/docs/git-checkout>

Ejemplos

- ▶ Recuperar el fichero filename del último commit (es decir, descartar cambios que aún no han sido confirmados):

```
git checkout HEAD filename
```

- ▶ Recuperar el fichero filename de un commit más antiguo:

```
#Visitar la historia de cambios
git log --oneline
#Recuperar el commit a través del hash
git checkout 55df4c2 -- filename
#O el cambio anterior a este commit
git checkout 55df4c2~1 -- filename
```

- ▶ Después de checkout hay que aplicar cambios y sincronizar

```
git commit -m "Mensaje..."
git push
```


① Conceptos básicos

② Uso de git y GitHub

③ Trabajo en colaboración

④ GitHub Classroom

⑤ Publicación de páginas web en GitHub

① Conceptos básicos

② Uso de git y GitHub

③ Trabajo en colaboración

Ramas

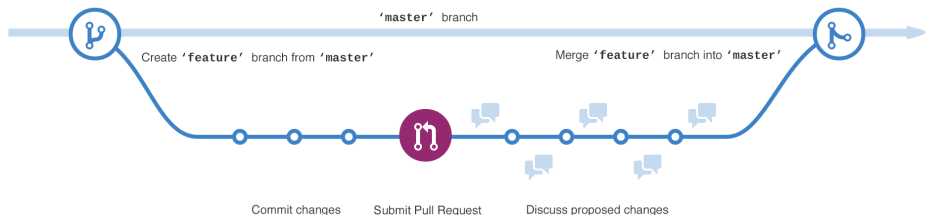
Persiguiendo a los bichos

Herramientas gráficas para el análisis de un repositorio

④ GitHub Classroom

⑤ Publicación de páginas web en GitHub

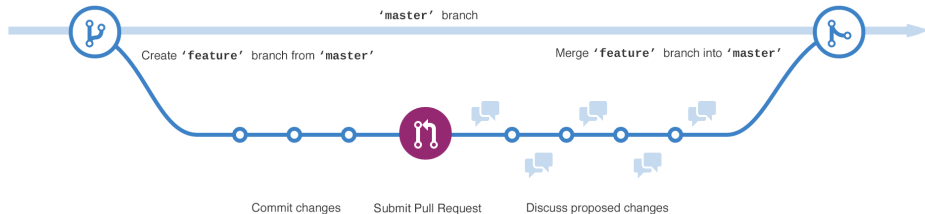
Rama master



En un repositorio de GitHub existe una rama (*branch*) que se usa por defecto: **master**⁸.

⁸Understanding the GitHub Flow

Ramas para facilitar la colaboración

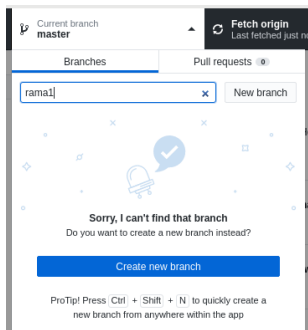


Cuando hay varias personas trabajando sobre un mismo repositorio, es necesario crear nuevas ramas para evitar conflictos.

De esta forma, cada persona implementa **cambios** en una **rama determinada** de forma paralela al resto del equipo.

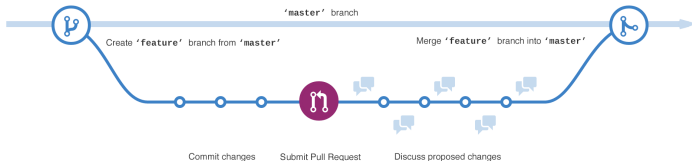
Crear una nueva rama

- ▶ En menú: *Branch > New branch...*
- ▶ O en pantalla principal: *Current branch > Branches > New branch*

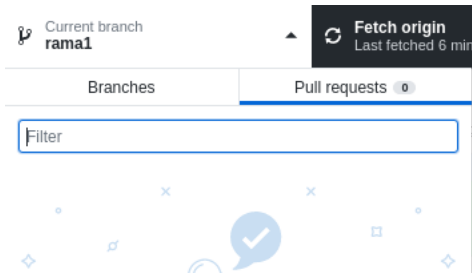


La nueva rama puede tener como origen la rama master u otra rama existente.

Combinación de código

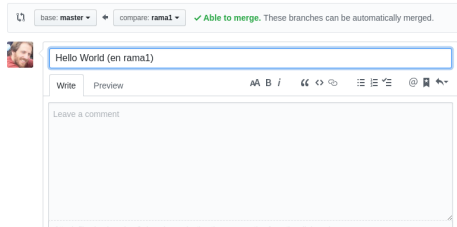


Cuando los cambios están listos y confirmados (*commit* + *push* en la rama específica), se realiza una petición (*pull request*) para combinar estos cambios en la rama **master**.

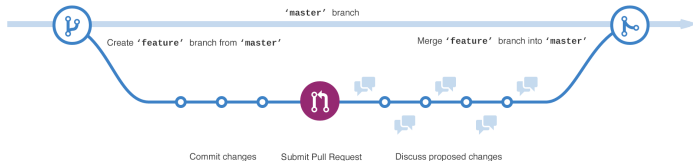


Open a pull request

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



Combinación de código



El coordinador del proyecto es el encargado de revisar cada petición y, si todo está correcto, incluir los cambios (*merge*) en la rama **master**.



Continuous integration has not been set up

[Several apps are available](#) to automatically catch bugs and enforce style.



This branch has no conflicts with the base branch

Merging can be performed automatically.

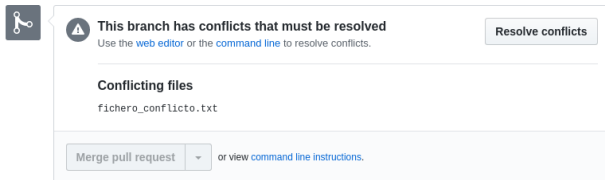
Merge pull request



or view [command line instructions](#).

Resolución de conflictos

Si no se pueden combinar los cambios automáticamente se produce un conflicto (ejemplo: dos usuarios modifican un mismo fichero).



Un conflicto se debe resolver manualmente.



Lectura recomendada: [Merge Conflicts in the Classroom](#)

Consejos

- ▶ **No olvides hacer *pull* antes de iniciar una nueva interacción con el repositorio.**
- ▶ Recuerda las recomendaciones sobre un buen mensaje de `git commit`.
- ▶ **Organización previa:** las **tareas** asignadas a un rama deben ser **independientes** de las otras ramas para evitar conflictos.
- ▶ Cuando el trabajo en una rama ha concluido, hay que **combinar cambios con master lo antes posible** para reducir la posibilidad de conflicto. Las ramas accesorias utilizadas se pueden eliminar una vez finalizado el proceso.
- ▶ Este proceso se debe repetir tantas veces como sea necesario para realizar cambios de forma colaborativa.

Ejercicio 1

Crea una nueva rama en tu repositorio. En esta rama crea un nuevo fichero de texto y añade contenido en él. Sincroniza con el repositorio. Vuelve a la rama master y comprueba que este fichero nuevo no está presente. Combina ambas ramas.

Ejercicio 2

Vuelve a la rama nueva y modifica un fichero. Vuelve a la rama master y modifica el mismo fichero. Combina ambas ramas y resuelve los conflictos.

① Conceptos básicos

② Uso de git y GitHub

③ Trabajo en colaboración

Ramas

Persiguiendo a los bichos

Herramientas gráficas para el análisis de un repositorio

④ GitHub Classroom

⑤ Publicación de páginas web en GitHub

Issues

Todos los repositorios de GitHub tienen una sección denominada «Issues»⁹ a modo de *bug tracker*.

Pueden usarse para seguimiento de fallos, mejoras, tareas, etc.

Filters ▾

is:open label:cluster

Labels

Milestones

✕ Clear current search query, filters, and sorts

🔔 13 Open ✓ 105 Closed

Author ▾ Projects ▾ Labels ▾ Milestones ▾ Assignee ▾ Sort ▾

🔔 net::Server.unref() failed on cluster mode

S-confirmed-bug

cluster

iojs-backport

net

2

#25782 opened on Jul 30, 2015 by kyriosli

🔔 ENOTSUP errnoException in NodeJS using mssql in cluster worker

cluster

1

#14382 opened on Apr 1, 2015 by ghost

🔔 listen() doesn't work with cluster on "node -e"

P-3

S-confirmed-bug

cluster

v0.10

v0.12

4

#14168 opened on Mar 26, 2015 by FCO

🔔 cluster round robin could know about ipc send back pressure

cluster

defer-to-converged

feature-request

#8746 opened on Nov 19, 2014 by tjfontaine

🔔 cluster: v0.11.13, cluster crashed in windows 2008r2 running on VMWare.

asynclistener

cluster

windows

#7667 opened on May 23, 2014 by missing1984

🔔 Cluster worker kill(signal) does not pass the signal parameter

cluster

10

Estructura de una issue

Una issue es un tablero de discusión en el que pueden participar los responsables del repositorio y cualquier usuario de GitHub.

Debe contener un título y una descripción.

Puede contener etiquetas, metas, y responsables.

Running cluster "hello world" on Win7 x64 returns -1073741819 #4707

 **Open** turanuk opened this issue on Feb 3, 2013 · 3 comments



turanuk commented on Feb 3, 2013

...

Running code listed here: http://nodejs.org/api/cluster.html#cluster_how_it_works seems to cause the worker to die with an error code of -1073741819 referring to an access violation. I assume this is a port conflict because if I change the code to only spawn a single worker the problem goes away and I'm able to access the web server. I tried port 3000 and ran into the same thing. Thoughts/debugging tips?

Assignees

No one assigned

Labels

cluster

windows



bnoordhuis commented on Feb 10, 2013

Member

...

Ah, good old 0xc0000005. If you're using the official binary with no native add-ons, that obviously shouldn't happen. Try hooking up the processes to a debugger to see where the access violation happens.

Projects

None yet

Milestone

No milestone

Contenido de una issue

- ▶ En la descripción de una issue se debe suministrar toda la información posible para el responsable del repositorio, **incluyendo un ejemplo mínimo, completo y verificable**¹⁰.
- ▶ El contenido será formateado como Markdown (incluye un *preview*)¹¹.
- ▶ Se pueden incluir referencias al código y a otras issues¹².

¹⁰<https://stackoverflow.com/help/mcve>

¹¹Veáse la guía [Basic Writing and Formatting syntax](#).

¹²Veáse la guía [Autolinked references and urls](#).

Ejercicio

- ▶ Abre nuevas issues en tu propio repositorio, o en repositorios ajenos (por ejemplo, https://github.com/oscarperpinan/prueba_github).
- ▶ Responde y cierra issues de otros usuarios en tu propio repositorio.

① Conceptos básicos

② Uso de git y GitHub

③ Trabajo en colaboración

Ramas

Persiguiendo a los bichos

Herramientas gráficas para el análisis de un repositorio

④ GitHub Classroom

⑤ Publicación de páginas web en GitHub

Toda la actividad realizada en un repositorio puede verse de manera gráfica a través del botón *Insights* en la web del repositorio en GitHub¹³. Por ejemplo,

- ▶ Contribución de los integrantes del equipo
- ▶ Estructuras de ramas de un repositorio
- ▶ Histórico de cambios en un repositorio

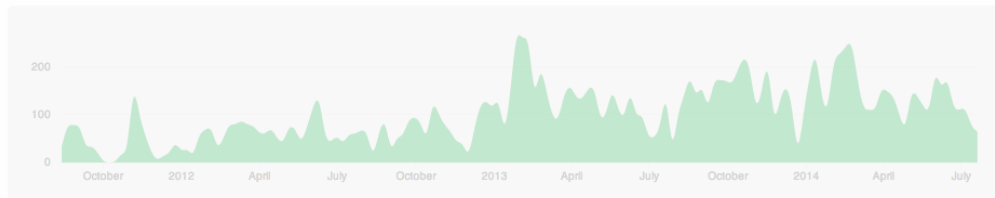
¹³Más detalles en [Ver información del repositorio de forma gráfica.](#)

Contribución de los integrantes del equipo

Aug 14, 2011 - Jul 23, 2014

Contributions to master, excluding merge commits

Contributions: **Commits** ▾



kevinsawicki

#1

5,913 commits / 122,808 ++ / 220,877 --



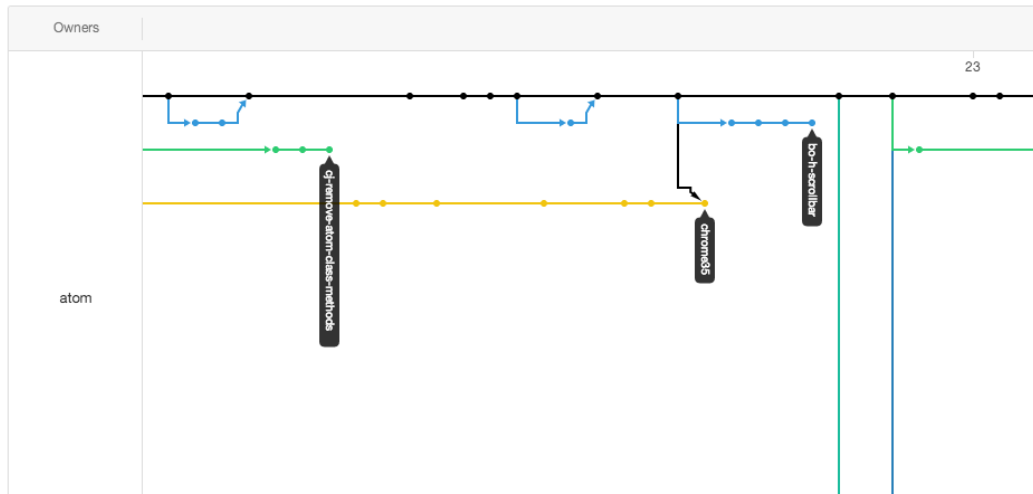
nathansobo

#2

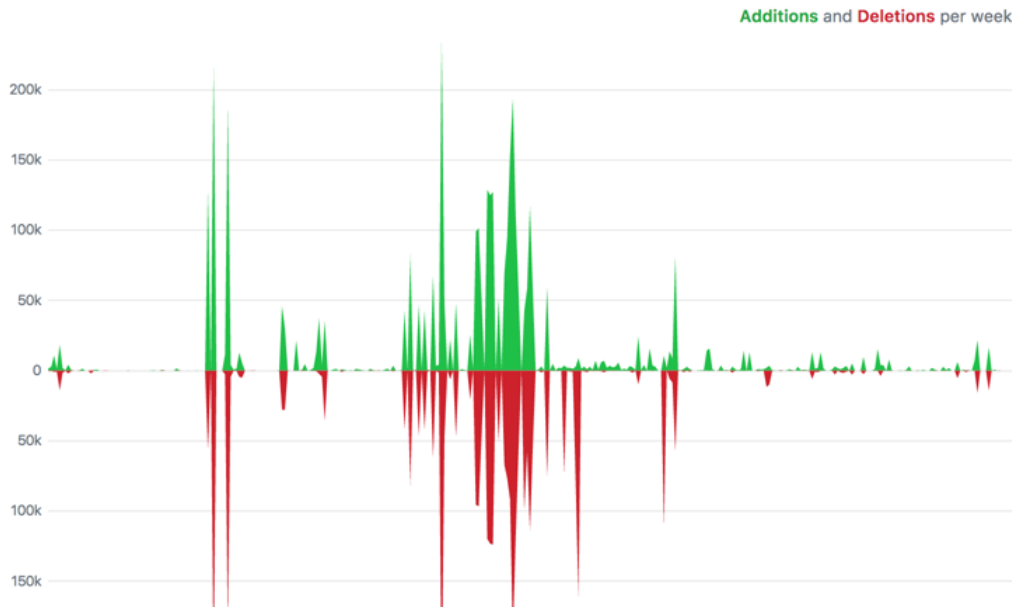
2,850 commits / 441,964 ++ / 340,415 --



Estructura de ramas de un repositorio



Cambios en un repositorio



Ejercicio

Visita la sección Insights de los repositorios del proyecto AIGORA (<https://github.com/aigora>).

Por ejemplo: https://github.com/aigora/twIE_2021-chkr_s-1/pulse

- ① Conceptos básicos
- ② Uso de git y GitHub
- ③ Trabajo en colaboración
- ④ **GitHub Classroom**
- ⑤ Publicación de páginas web en GitHub

¿Qué es GitHub Classroom?

- ▶ Es un asistente para configurar grupos y asignar tareas en GitHub.
- ▶ Accesible en: <https://classroom.github.com/>
- ▶ Es recomendable [solicitar descuento por cuenta académica](#) (repositorios públicos y privados ilimitados, colaboradores ilimitados).

GitHub Classroom necesita una Organización

- ▶ Para poder trabajar con GitHub Classroom es necesario crear una Organización en Github (grupo de cuentas que tienen acceso compartido a un grupo de repositorios):
<https://github.com/organizations/new>
- ▶ Ejemplos
<https://github.com/swcarpentry>
<https://github.com/aigora>
- ▶ Las cuentas tipo organization también pueden solicitar el [descuento por cuenta académica](#).

Dentro de una organización se pueden definir equipos

- ▶ Dentro de una `organization` pueden coexistir diferentes equipos (Team)
- ▶ Se pueden asignar diferentes permisos de acceso y escritura a cada equipo, o a conjuntos de equipos.
- ▶ Ejemplo:
<https://github.com/orgs/aigora/teams>
- ▶ Más información en el [blog de GitHub](#).

Configuración de aulas

- ▶ Una única cuenta `organization` puede dar cabida a múltiples aulas (`classrooms`):
<https://classroom.github.com/classrooms/>
- ▶ Cada `classroom` contiene estudiantes (identificados por su usuario de GitHub y su correo electrónico) y grupos (`teams`).
- ▶ Esta correspondencia se puede facilitar mediante el `classroom roster`¹⁴.

¹⁴[Creating a roster for your classroom \(GitHub Docs\)](#)

Asignación de tareas

- ▶ Dentro de cada classroom se pueden incluir múltiples tareas (assignments).
- ▶ Un assignment puede ser individual o grupal.
- ▶ Los grupos se pueden definir antes por el profesor, o por los estudiantes en el momento de aceptar la tarea.

Cómo hemos usado GitHub Classroom en AIGORA

Configuración General

- ▶ Usamos una `organization` común para todos los grupos de matriculación de una misma asignatura.
- ▶ Definimos un `classroom` para cada grupo de matriculación.
- ▶ En cada `classroom` subimos la lista de correos UPM de los estudiantes de ese grupo de matriculación mediante el `roster management`.
- ▶ Dentro de cada `classroom` definimos diferentes equipos de trabajo (Teams) para trabajos grupales.

Cómo hemos usado GitHub Classroom en AIGORA

Registro de estudiantes

- ① En cada classroom creamos una Group Assignment denominada «registro estudiantes». Como identificador del «Group» utilizamos el grupo de matriculación. Este identificador queda almacenado en GitHub Classroom para futuros usos, pero **no** crea un Team en GitHub.
- ② Abrimos el enlace del assignment con nuestro usuario (podemos hacer skip cuando nos pide que enlacemos a algún miembro del roster), y **creamos un nuevo Team**, que se debe llamar igual que el grupo de matriculación. Este paso **sí** crea un nuevo Team en la organization de Github (útil para asignar permisos grupales).
- ③ A continuación enviamos el enlace del assignment a los alumnos para que enlacen su usuario de GitHub a su correo incluido en el roster. **Deben elegir el Team que hemos definido nosotros, sin crear uno nuevo**. A partir de este punto forman parte del Team con los permisos y repositorios definidos para el mismo.

Cómo hemos usado GitHub Classroom en AIGORA

Grupos de Trabajo

- ❶ Para el trabajo en equipo creamos igualmente una Group Assignment, pero dejamos libertad a la hora de definir los Teams.
- ❷ El primer estudiante de un grupo de trabajo que acepta este assignment define el nombre del Team. Los siguientes estudiantes de ese mismo grupo deben seleccionar ese Team.
- ❸ Se pueden definir repositorios plantilla para incluir contenido inicial en todos los repositorios creados para cada Team. Ejemplo:
<https://github.com/aigora/starter-code>

- ① Conceptos básicos
- ② Uso de git y GitHub
- ③ Trabajo en colaboración
- ④ GitHub Classroom
- ⑤ Publicación de páginas web en GitHub

Página web de **proyecto**

<https://<username>.github.io/<repository>>

Si no sabes HTML

- ▶ En la página del repositorio:

Settings > GitHub Pages > Source > master branch

Settings > GitHub Pages > Theme Chooser

- ▶ Modifica el fichero `README.md`^a (commit + push).

^aMás información sobre formato Markdown

<https://guides.github.com/features/mastering-markdown/>.

Página web de **proyecto**

`https://<username>.github.io/<repository>`

Si sabes HTML

- ▶ Crea una carpeta docs en la rama master del repositorio.
- ▶ En esta carpeta docs crea/modifica un fichero index.html (commit + push).

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    Hello World!
  </body>
</html>
```

- ▶ En la página del repositorio: *Settings > GitHub Pages > Source > docs folder*

Ejemplo de web de **proyecto**

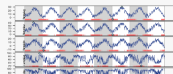
- ▶ Página web: <https://oscarperpinan.github.io/bookvis/>
- ▶ Repositorio: <https://github.com/oscarperpinan/bookvis/tree/master/docs>

Displaying time series, spatial and space-time data with R

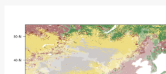
This is the accompanying website of the **second edition** of the book “*Displaying time series, spatial and space-time data with R*”, published with [Chapman&Hall/CRC](#).

Code, data, and figures are available at this [GitHub repository](#).

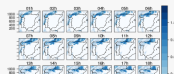
Time Series



Spatial Data



Space-time Data



Página web de **usuario u organización**

- ❶ Crea un repositorio nuevo con el nombre `<username>.github.io`¹⁵.
- ❷ Sube (commit + push) un fichero `index.html` a la rama `master` con código HTML.
- ❸ Con un navegador ve a la dirección `https://<username>.github.io`

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    Hello World!
  </body>
</html>
```

¹⁵Siendo `<username>` tu nombre de usuario en GitHub.

Ejemplo de web de **organización**

- ▶ Página web: <https://aigora.github.io/>
- ▶ Repositorio: <https://github.com/aigora/aigora.github.io>

