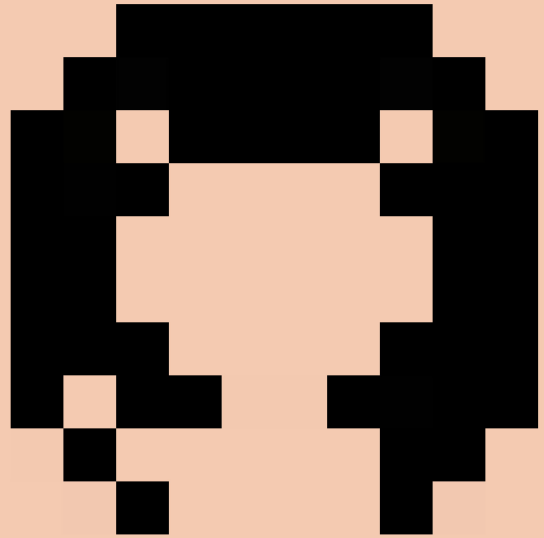
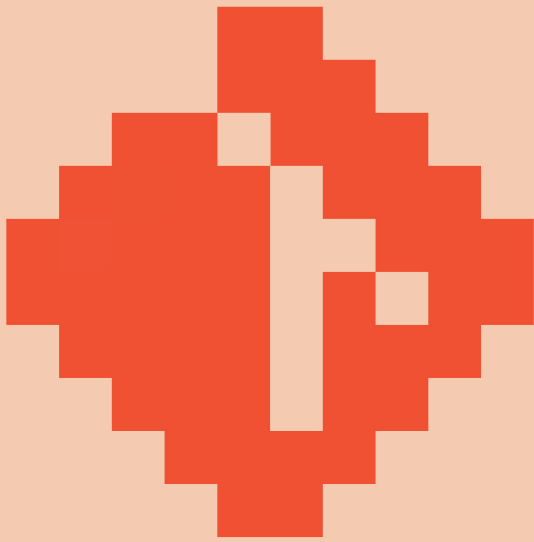


{ LUIS JOSÉ SÁNCHEZ }



# Git y GitHub

## Guía de Supervivencia

[Lo básico para mantener tu código bajo control]

# Git y GitHub. Guía de Supervivencia.

Luis José Sánchez González

Este libro está a la venta en <http://leanpub.com/gitygithub>

Esta versión se publicó en 2016-09-08



Leanpub

This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2016 Luis José Sánchez González

# Índice general

Sobre el autor . . . . .	i
Sobre este libro . . . . .	ii
El libro original . . . . .	iii
<b>1. Instalación y configuración de Git . . . . .</b>	<b>1</b>
1.1 ¿Qué es Git? . . . . .	1
1.2 Instalación de Git . . . . .	1
1.3 Configuración de Git . . . . .	2
1.4 Comandos utilizados en este capítulo . . . . .	3
<b>2. GitHub - Creación de una cuenta y configuración básica . . . . .</b>	<b>4</b>
2.1 ¿Qué es GitHub? . . . . .	4
2.2 Creación de una cuenta en GitHub . . . . .	4
2.3 Configuración de GitHub . . . . .	8
<b>3. Primeros pasos con Git y GitHub . . . . .</b>	<b>11</b>
3.1 Clonado de repositorios (de GitHub a nuestra máquina) . . . . .	11
3.2 Actualización local de repositorios de GitHub . . . . .	16
3.3 Creación de repositorios en GitHub . . . . .	18
3.4 Comandos utilizados en este capítulo . . . . .	22
<b>4. Flujo de trabajo con Git . . . . .</b>	<b>23</b>
4.1 Punto de partida . . . . .	23
4.2 Ciclo completo de actualización de un repositorio (add, commit, push) . . . . .	24
4.3 Aperitivo, comida y postre . . . . .	28

## ÍNDICE GENERAL

4.4	Comandos utilizados en este capítulo . . . . .	31
<b>5.</b>	<b>Ficheros README.md, .gitignore y HEAD . . . . .</b>	<b>33</b>
5.1	El fichero README.md . . . . .	33
5.2	El fichero .gitignore . . . . .	36
5.3	El fichero HEAD . . . . .	38
5.4	Comandos utilizados en este capítulo . . . . .	41
<b>6.</b>	<b>Versiones y ramas . . . . .</b>	<b>43</b>
6.1	Etiquetado de versiones . . . . .	43
6.2	Ramas . . . . .	48
6.3	Comandos utilizados en este capítulo . . . . .	55
<b>7.</b>	<b>Desarrollo colaborativo (fork y pull request) . . . . .</b>	<b>58</b>
7.1	Comandos utilizados en este capítulo . . . . .	77
	<b>Invitación . . . . .</b>	<b>79</b>
	<b>Referencias . . . . .</b>	<b>80</b>
	Git . . . . .	80
	GitHub . . . . .	80
	Fichero .gitignore . . . . .	80
	Markdown . . . . .	80
	Emojis . . . . .	80
	Java . . . . .	80

# Sobre el autor

**Luis José Sánchez González** es Ingeniero Técnico en Informática de Gestión por la **Universidad de Málaga (España)** y funcionario de carrera de la Junta de Andalucía desde 1998. En su trabajo de profesor de Informática combina sus dos pasiones: la enseñanza y la programación.

En el momento de publicar este libro, es profesor del **I.E.S. Campanillas (Málaga)** e imparte clases en el **Ciclo Superior de Desarrollo de Aplicaciones Web**.

Puedes ponerte en contacto con el autor mediante la dirección de correo electrónico [luisjoseprofe@gmail.com](mailto:luisjoseprofe@gmail.com) o mediante LinkedIn (<https://es.linkedin.com/pub/luis-josé-sánchez/86/b08/34>).

# Sobre este libro

“*Git y GitHub - Miniguía de Supervivencia*” es un manual que enseña lo imprescindible (y un poquito más) para manejar estas dos herramientas que son básicas en el quehacer diario de cualquier programador.

Esta miniguía ilustra cómo mantener el código mínimamente organizado mediante comandos de [Git](#) y cómo, además, hacer visible ese código de cara a la comunidad de programadores en el vasto universo de [GitHub](#).

Normalmente pido a mis alumnos que tengan disponible en [GitHub](#) su trabajo: ejemplos de prueba, ejercicios de clase, proyectos, etc. Además, les insto a que actualicen sus repositorios<sup>1</sup> con frecuencia para ver así cómo va evolucionando su trabajo en el tiempo. En este libro pretendo dar los conceptos básicos para llevar a cabo estas tareas.

Si nunca antes has oído hablar de [Git](#) ni de [GitHub](#) no te preocupes, empezamos de cero. Cuando adquieras soltura manejando estas herramientas ya no querrás vivir sin ellas.

---

<sup>1</sup>Veremos más adelante qué son los **repositorios** y cómo crearlos y mantenerlos. De momento, te los puedes imaginar como lugares en internet donde guardamos nuestro código o cualquier otro tipo de contenido. Normalmente un repositorio contiene un proyecto de cierta entidad o bien agrupa pequeños proyectos que tienen alguna relación entre sí. Un ejemplo de repositorio es <https://github.com/LuisJoseSanchez/aprende-java-con-ejercicios> que contiene todos los ejemplos y soluciones a los ejercicios de mi libro [Aprende Java con Ejercicios](#).

# El libro original

Este libro es completamente gratuito y lo puedes descargar desde la página <https://leanpub.com/gitygithub>.

Si has descargado o copiado el libro de otra fuente, puede que no tengas la última versión corregida y actualizada. Te aconsejo encarecidamente que descargues el libro desde la página indicada.

# 1. Instalación y configuración de Git

## 1.1 ¿Qué es Git?

Git es una herramienta de control de versiones. Permite controlar el proceso de creación de software<sup>1</sup> llevando un registro exhaustivo de todos los cambios realizados. Ofrece la posibilidad de crear versiones, de ramificar un proyecto en diferentes flujos e incluso de volver hacia atrás deshaciendo los últimos cambios realizados.

Git es un programa libre y gratuito que se distribuye mediante la licencia GNU (GPL 2.0)<sup>2</sup>

## 1.2 Instalación de Git

Para instalar Git en Ubuntu basta con teclear la siguiente línea en una ventana de terminal:

```
sudo apt install git
```

Para instalar Git en otras plataformas, puedes consultar los binarios disponibles en <https://git-scm.com/downloads>.

Una vez instalado Git, se utiliza mediante comandos en una ventana de terminal.

Podemos comprobar que el programa se ha instalado correctamente comprobando el número de versión.

---

<sup>1</sup>En realidad se puede extender el uso de Git a proyectos no solo de software sino también de documentos (hay muchos libros escritos con Git), presentaciones y, en general, a casi cualquier tipo de contenido.

<sup>2</sup>Los términos de la licencia están descritos en <https://opensource.org/licenses/GPL-2.0>

```
git --version  
git version 2.7.4
```



## ¡Atención!

Aunque existen interfaces gráficas para Git, no te dejes engatusar por sugerentes botones y colorines; eso es de blandengues. Los hombres y mujeres duros de pelar, los auténticos programadores, usan Git mediante comandos en una ventana de terminal.

## 1.3 Configuración de Git

Antes de usar Git es conveniente emplear unos minutos en configurar la herramienta.

Lo primero que hay que hacer es indicar el nombre y el correo electrónico.

```
git config --global user.name "Alan Brito Delgado"  
git config --global user.email "alan.brito.delgado.1972@gmail.com"
```

Para realizar ciertas acciones se nos pedirá una contraseña. Puede resultar tedioso tener que introducirla constantemente. Git permite “cachear” la contraseña de modo que solo hará falta introducirla una vez al principio de la sesión.

```
git config --global credential.helper 'cache --timeout=36000'
```

## 1.4 Comandos utilizados en este capítulo

### >\_ Instalación de Git

```
sudo apt install git
```

### >\_ ¿Cuál es la versión de Git instalada?

```
git --version
```

### >\_ Nombre y correo electrónico del usuario de Git

```
git config --global user.name "Alan Brito Delgado"  
git config --global user.email "alan.brito.delgado.1972@gmail.com"
```

### >\_ Cacheo de la contraseña durante la sesión

```
git config --global credential.helper 'cache --timeout=36000'
```

# 2. GitHub - Creación de una cuenta y configuración básica

## 2.1 ¿Qué es GitHub?

GitHub es una plataforma web que permite alojar proyectos controlados mediante Git. En GitHub tendremos una especie de “espejo” o duplicado de los proyectos que tenemos en nuestro propio ordenador y, además, podremos ver los proyectos de otra mucha gente.

GitHub funciona también como una red social de programadores en la que se pueden votar los proyectos y los usuarios. Cada usuario puede establecer vínculos con otros usuarios y puede seguir lo que hace.

Muchas empresas buscan programadores en esta plataforma hasta tal punto que hoy día GitHub es considerado el escaparate del programador.

El uso de GitHub es gratuito para los proyectos públicos, sin límite en cuanto a la cantidad de repositorios. Para tener proyectos privados necesitaremos una cuenta de pago.

## 2.2 Creación de una cuenta en GitHub

Entra en la web de GitHub (<https://github.com>) y haz clic en el botón **Sign up** (figura 2.2.1).

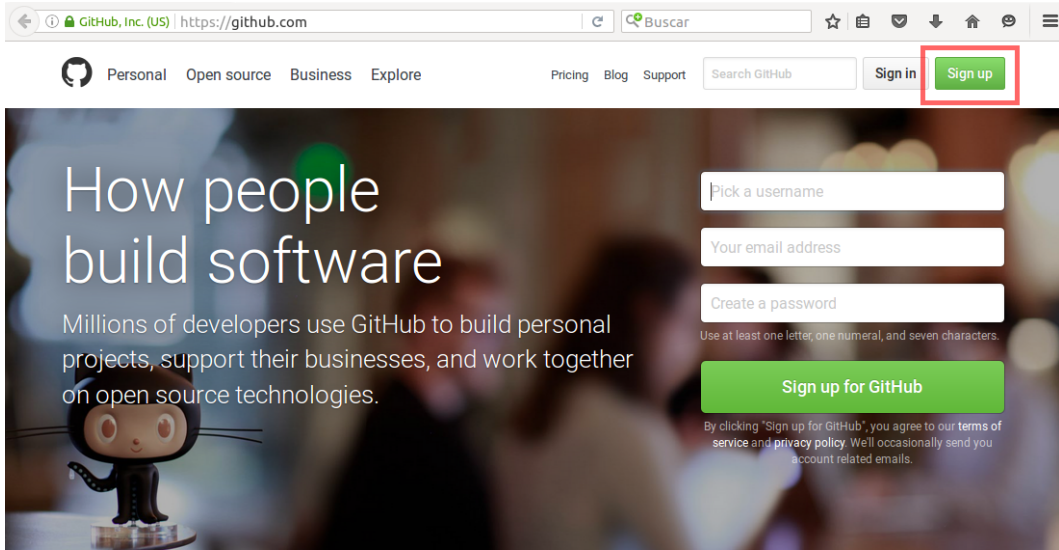


Figura 2.2.1: Botón Sign up en <https://github.com>.

En el primer paso se nos piden los datos personales para la nueva cuenta (figura 2.2.2).

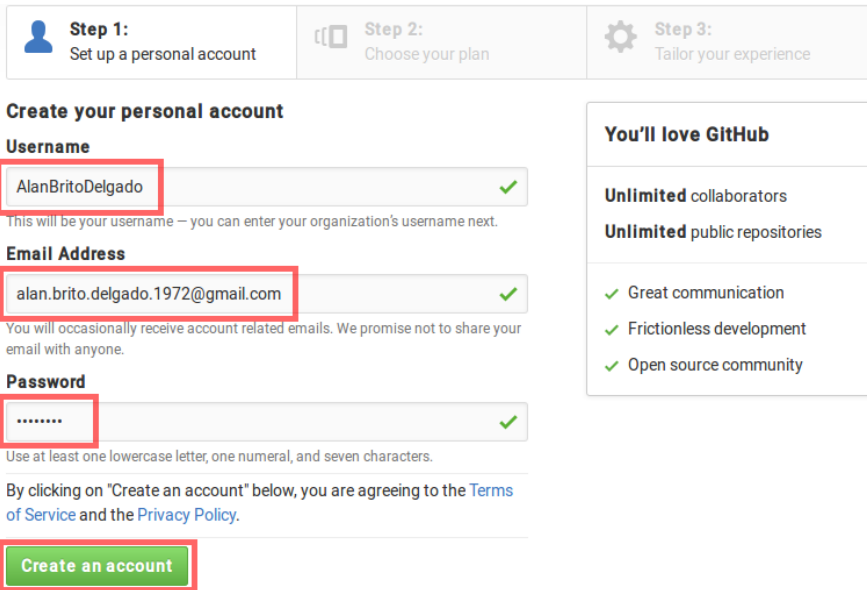


Figura 2.2.2: Creación de cuenta en GitHub. Paso 1: Datos personales.

El nombre de usuario solo puede contener letras y números. Están prohibidos los espacios, aunque se pueden usar los guiones como separadores. Una buena práctica consiste en usar el nombre completo sin espacios en formato capitalizado; por ejemplo, AlanBritoDelgado es un buen nombre de usuario.

La dirección de correo electrónico debe tener un formato correcto y la contraseña debe tener un mínimo de siete caracteres y contener al menos un número.

The screenshot shows the GitHub account creation interface. At the top, it says "Welcome to GitHub" and "You've taken your first step into a larger world, @AlanBritoDelgado." Below this is a progress bar with three steps: "Step 1: Completed Set up a personal account", "Step 2: Choose your plan", and "Step 3: Tailor your experience". The "Step 2" section is active and titled "Choose your personal plan". It offers two options: "Unlimited public repositories for free." (selected with a red radio button and highlighted by a red box) and "Unlimited private repositories for \$7/month. (view in EUR)". Below these options is a note: "Don't worry, you can cancel or upgrade at any time." There is also a checkbox for "Help me set up an organization next" with a brief description and a link to "Learn more about organizations." To the right, a box titled "Both plans include:" lists benefits: "Collaborative code review", "Issue tracking", "Open source community", "Unlimited public repositories", and "Join any organization". At the bottom left, there is a green "Continue" button highlighted with a red box.

Figura 2.2.3: Creación de cuenta en GitHub. Paso 2: Elección del plan (gratis o de pago).

En el segundo paso debes elegir el plan (figura 2.2.3). De forma gratuita se puede crear un número ilimitado de repositorios públicos.

Para tener la posibilidad de crear repositorios privados, es necesario contratar un plan de pago. Los repositorios privados son accesibles (y por lo tanto visibles) para el que los crea pero son invisibles para el resto de la comunidad. Este tipo de repositorio tiene sentido cuando el software que se va a almacenar en el repositorio tiene fines comerciales.

Una vez creada la cuenta es posible cambiar de plan.

# Welcome to GitHub

You'll find endless opportunities to learn, code, and create,  
**@AlanBritoDelgado**.

✓ Completed Set up a personal account	📄 Step 2: Choose your plan	⚙️ Step 3: Tailor your experience
--	-------------------------------	--------------------------------------

**How would you describe your level of programming experience?**

Very experienced       Somewhat experienced       Totally new to programming

**What do you plan to use GitHub for?** (check all that apply)

Research       Project Management       Design

School projects       Development       Other (please specify)

**Which is closest to how you would describe yourself?**

I'm a professional       I'm a hobbyist       I'm a student

Other (please specify)

**What are you interested in?**

e.g. tutorials, android, ruby, web-development, machine-learning, open-source

[skip this step](#)

Figura 2.2.4: Creación de cuenta en GitHub. Paso 3: Información adicional.

A continuación se pide cierta información adicional (figura 2.2.4) como el nivel de programación, el propósito por el que creas la cuenta, etc. Tómate tu tiempo y rellena todos los campos, solo tendrás que hacerlo una vez.

## Learn Git and GitHub without any code!

Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request.

Figura 2.2.5: Creación de cuenta en GitHub. Cuenta creada.

Por último, si todo el proceso de creación de la cuenta ha sido satisfactorio, aparecerá una pantalla en la que se te invita a leer una guía o a crear un proyecto. No te

impacientes, antes de nada te invito a que configures tu nueva cuenta.

## 2.3 Configuración de GitHub

Antes de realizar cualquier acción dentro de GitHub es conveniente realizar unos ajustes previos.

Haz clic en el icono de tu avatar (inicialmente se asigna un avatar aleatorio por defecto). En el menú desplegable que aparece selecciona **Your profile**.

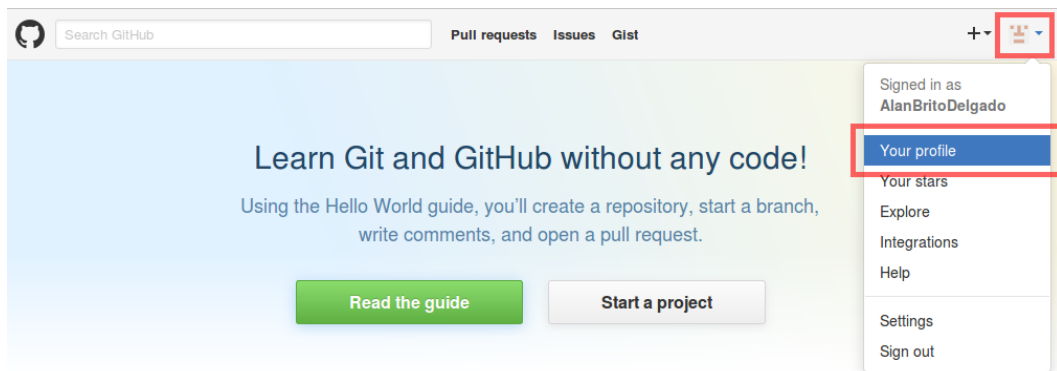


Figura 2.3.1: Entrar en el perfil de GitHub.

De esta manera accedes a tu página de perfil. Ahora haz clic en el botón **Edit profile**.

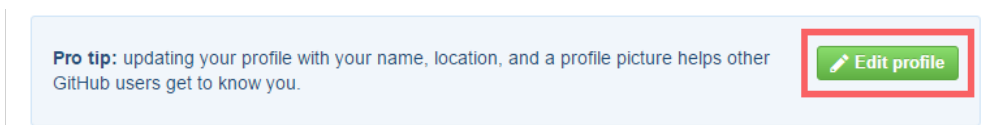


Figura 2.3.2: Acceso a la edición del perfil.

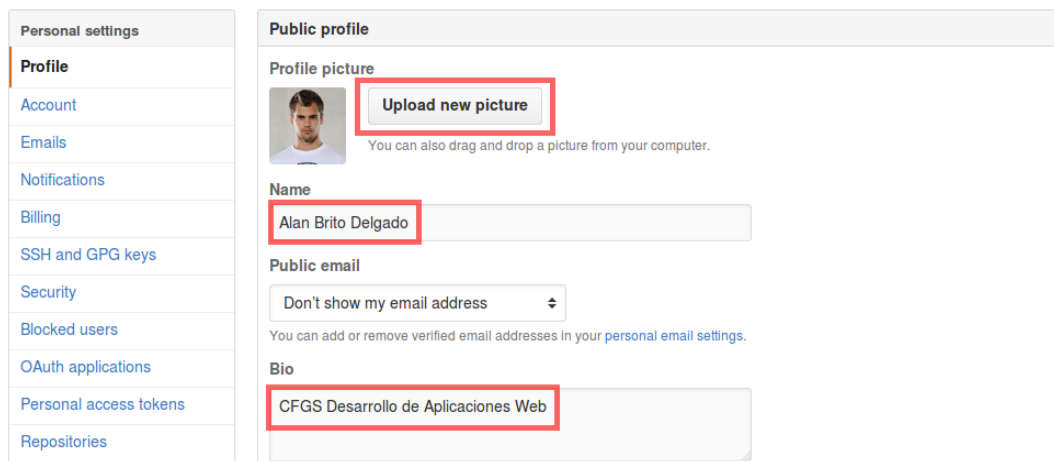
Cambia el horroroso avatar que te asigna GitHub por una foto en la que se te reconozca<sup>1</sup>. No pongas una foto de una orla ni una en la que aparezcas con traje y corbata. Recuerda que GitHub es una herramienta para programadores, no para agentes de bolsa ni corredores de seguros. Está bien poner una foto con camiseta (si

<sup>1</sup>La imagen usada para el perfil de prueba de "Alan Brito" es de dominio público y está obtenida de Pixabay (<https://pixabay.com>).

es una camiseta “friki” mejor todavía). Ni se te ocurra poner una foto de tu mascota, de tu novio/a o de tu coche. Hay muchos perfiles de buenos programadores en GitHub estropeados por una mala foto.

Escribe tu nombre completo (nombre y apellidos), con los correspondientes espacios, mayúsculas y tildes si procede.

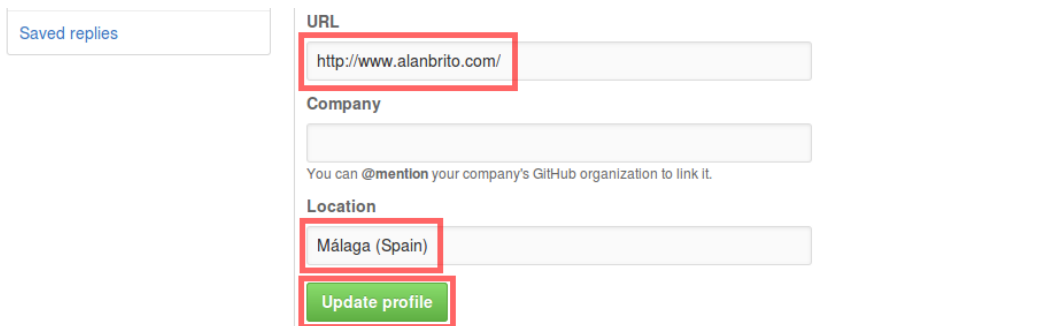
Anota en el campo **Bio** cualquier curso que hayas hecho o estés haciendo relacionado con la programación o con la informática en general, en orden de importancia. Por ejemplo, si estás estudiando un grado universitario o un ciclo formativo, éste es el lugar idóneo para escribirlo.



The image shows a screenshot of the GitHub 'Public profile' settings page. On the left is a sidebar with 'Personal settings' and a list of menu items: Profile, Account, Emails, Notifications, Billing, SSH and GPG keys, Security, Blocked users, OAuth applications, Personal access tokens, and Repositories. The main content area is titled 'Public profile' and contains several fields: 'Profile picture' with a placeholder image and an 'Upload new picture' button; 'Name' with the text 'Alan Brito Delgado'; 'Public email' with a dropdown menu set to 'Don't show my email address'; and 'Bio' with the text 'CFGS Desarrollo de Aplicaciones Web'. Red boxes highlight the 'Upload new picture' button, the 'Name' field, and the 'Bio' field.

Figura 2.3.3: Foto, nombre y biografía.

Si tienes una web personal o un blog con contenidos relacionados con la programación o con algo que tenga que ver con la informática, escribe la dirección en el campo **URL**. Si no tienes nada de eso, escribe en este campo la dirección de tu perfil de [LinkedIn](#). En caso de no tener cuenta en [LinkedIn](#), te recomiendo encarecidamente que te des de alta; es una red social profesional muy práctica para hacer contactos a nivel corporativo y también es muy útil en la búsqueda de empleo.

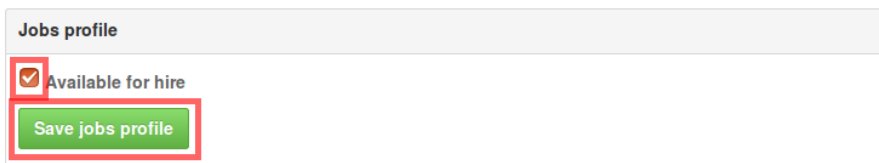


The screenshot shows a form for updating a GitHub profile. On the left, there is a box labeled "Saved replies". The main form has three input fields: "URL" containing "http://www.alanbrito.com/", "Company" (empty), and "Location" containing "Málaga (Spain)". Below the "Location" field is a green button labeled "Update profile". A small text note below the "Company" field says "You can @mention your company's GitHub organization to link it."

Figura 2.3.4: URL, empresa y localización.

Rellena el campo **Company** si trabajas para una empresa.

El campo **Location** es muy importante. Escribe el nombre de tu ciudad y, entre paréntesis, el nombre de tu país en inglés. Imagina que una compañía importante está buscando programadores en tu ciudad. Lo primero que haría la empresa es mirar el ranking de un determinado lenguaje (por ejemplo Java) en tu ciudad. Si no tienes relleno el campo **Location** simplemente no aparecerías en ese ranking (ni siquiera en los últimos puestos) y habrías perdido una buena oportunidad.



The screenshot shows the "Jobs profile" section. It has a header "Jobs profile" and a checkbox labeled "Available for hire" which is checked. Below the checkbox is a green button labeled "Save jobs profile".

Figura 2.3.5: Disponible para ser contratado.

Por último, si quieres decirle al mundo que estás disponible para que te contraten, marca la casilla **Available for hire**. Hay aplicaciones que sondean los perfiles de GitHub en busca de candidatos para puestos de programación y miran si esta casilla está marcada. Recuerda que tu cuenta de GitHub es tu mejor carta de presentación como programador.

# 3. Primeros pasos con Git y GitHub

## 3.1 Clonado de repositorios (de GitHub a nuestra máquina)

Clonar significa hacer una copia exacta; por tanto, clonar un repositorio es hacer una copia idéntica de un proyecto que existe en GitHub y llevártela a tu máquina.

Estarás pensando ¿qué diferencia hay entre clonar un repositorio y simplemente bajar los archivos? En principio no hay diferencia, pero si el repositorio de GitHub cambia porque se añaden, se borran o se modifican ficheros, entonces tendrás en tu máquina una versión desactualizada, ya no será un clon de lo que hay en GitHub. Sin embargo, cuando se ha clonado un repositorio con Git, la actualización es algo trivial como veremos en el siguiente apartado.

Para clonar un repositorio tan solo nos hace falta saber su dirección exacta en GitHub. Unas veces se nos proporcionará ese dato y, en otras ocasiones, deberemos buscar la dirección desde el mismo GitHub.

Veamos un caso práctico. Todos los ejemplos y soluciones a los ejercicios de mi libro [Aprende Java con Ejercicios](https://github.com/LuisJoseSanchez/aprende-java-con-ejercicios) están en GitHub, en un repositorio cuya dirección es <https://github.com/LuisJoseSanchez/aprende-java-con-ejercicios.git>. Para clonar este repositorio tan solo tendrás que hacer lo siguiente.

Nos situamos dentro de la carpeta donde queremos clonar el repositorio, por ejemplo en la carpeta Programacion que está dentro de Documentos:

```
cd Documentos/Programacion/
```

Clonamos el repositorio:

```
git clone https://github.com/LuisJoseSanchez/aprende-java-con-ejercicios.git
```

### Aparecerán los siguientes mensajes:

```
Clonar en «aprende-java-con-ejercicios»...
remote: Counting objects: 2852, done.
remote: Total 2852 (delta 0), reused 0 (delta 0), pack-reused 2852
Receiving objects: 100% (2852/2852), 3.92 MiB | 1.17 MiB/s, done.
Resolving deltas: 100% (1690/1690), done.
Comprobando la conectividad... hecho.
```

Lo que indica que el proceso ha terminado satisfactoriamente.

Ahora dentro de Programacion hay una carpeta con nombre aprende-java-con-ejercicios que contiene todos los archivos y carpetas del repositorio.

Puedes ver todo el contenido - carpetas, subcarpetas y archivos - con el comando `tree`<sup>1</sup>.

```
tree
```

Obteniendo la salida que se muestra a continuación.

```
.
├── aprende-java-con-ejercicios
│   ├── ejemplos
│   │   ├── 01_Hola_mundo_Salida_de_datos_por_pantalla
│   │   │   ├── Colores.java
│   │   │   └── HolaMundo.java
│   │   ├── 02_Variables
│   │   │   ├── Asignaciones.java
│   │   │   └── ...
│   │   └── ...
│   └── ...
└── ...
```

---

<sup>1</sup>Para instalar el comando `tree`, teclea `sudo apt install tree` en una ventana de terminal.

Veamos otro ejemplo práctico. Supongamos que desconocemos la dirección exacta de un repositorio. De hecho, vamos a cotillar un poco por GitHub y vamos a clonar algún proyecto que simplemente nos llame la atención, solo para hacer una prueba.

Entra en tu cuenta de GitHub (<https://github.com/>) y haz click sobre tu avatar.

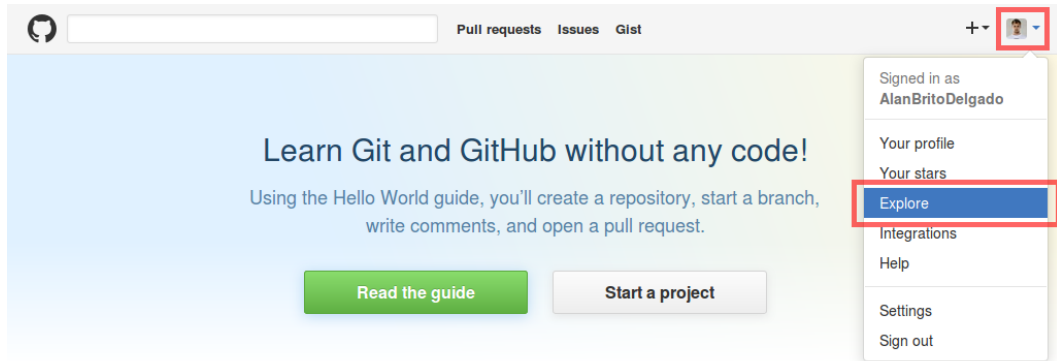


Figura 3.1.1: Explorar

A continuación, selecciona **Explore** en el menú desplegable.

GitHub elegirá para ti una serie de repositorios que considera interesantes, agrupados por categorías.

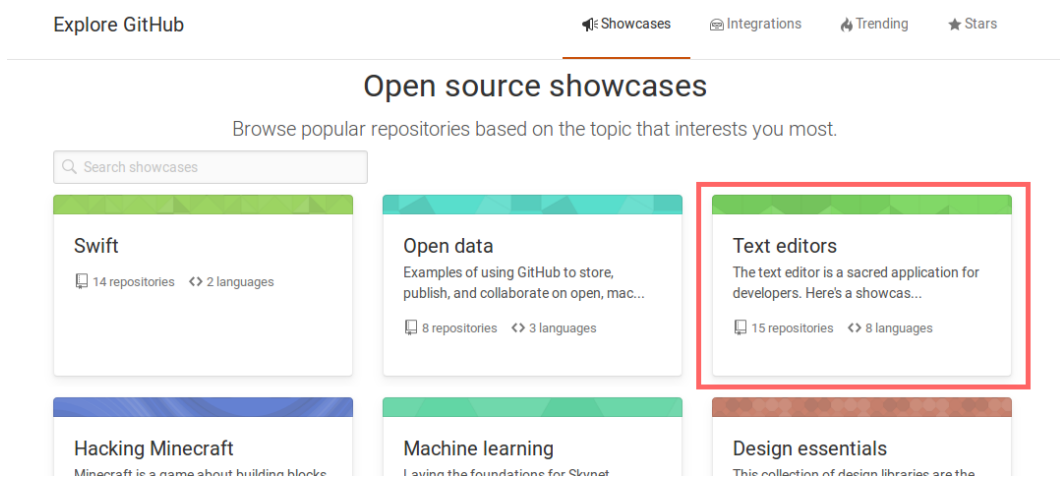


Figura 3.1.2: Seleccionar categoría

Lo que verás en tu ordenador seguramente será diferente a lo que se muestra en la captura de pantalla de este manual; no te preocupes, las sugerencias que muestra GitHub van cambiando.

Si encuentras la categoría **Text editors**, selecciónala. Si no la encuentras, puedes seleccionar **3D Modeling**, **Virtual Reality**, **Productivity tools** o cualquier otra de las que aparecen en el listado.

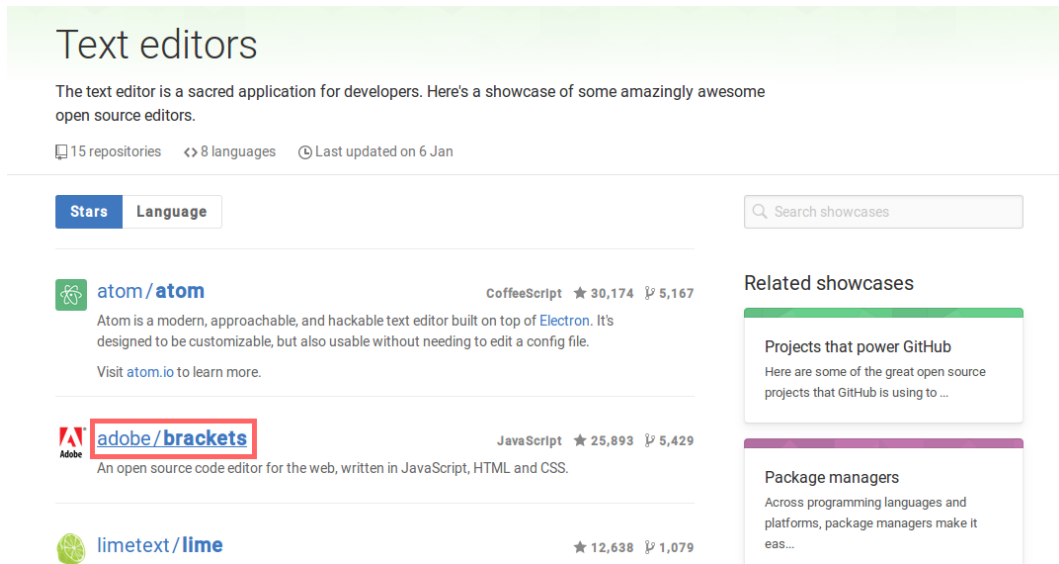


Figura 3.1.3: Seleccionar repositorio

Una vez seleccionada la categoría, aparecen los repositorios que contiene. Para ilustrar este caso práctico, he elegido el repositorio donde se almacena el código fuente del fantástico editor **Brackets**.

Desde dentro del repositorio tenemos acceso a gran cantidad de información. Lo más importante es, lógicamente, el código; podemos ver todos y cada uno de los ficheros que componen la aplicación y meternos dentro de ellos para ver cómo está programada. También se ven otros datos como la cantidad de personas que colabora en el proyecto, el número de versiones que se han liberado, las incidencias pendientes de resolver y un largo etcétera.

Lo que nos interesa a nosotros es la dirección exacta del repositorio para poder clonarlo. Haz click en **Clone or download**.

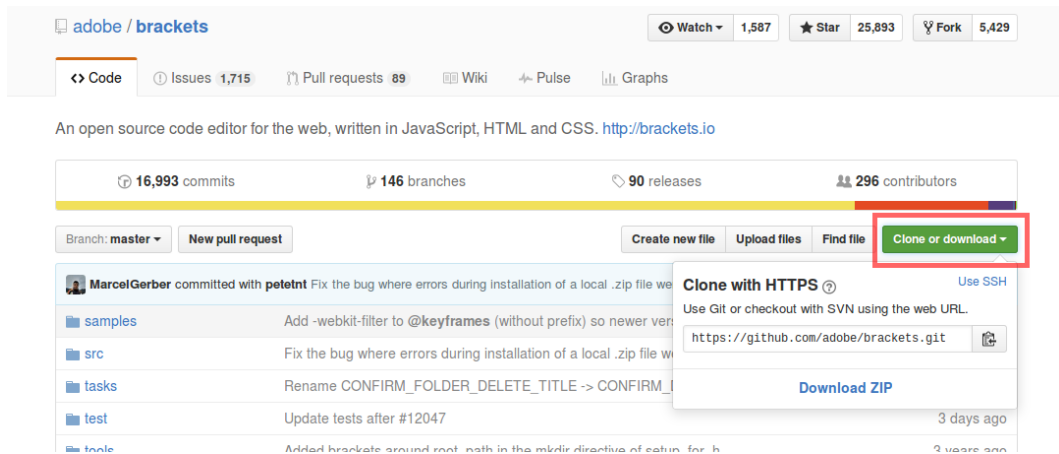


Figura 3.1.4: Botón “Clone or download”

Cuando aparezca la ventana emergente, haz click en el icono del portapapeles; de esta manera, la dirección del repositorio se copia a la memoria. Para pegarla en una ventana de terminal se utiliza el botón derecho del ratón y la opción **Pegar** o, más rápido, la combinación de teclas **Control + Mayúscula + V**.

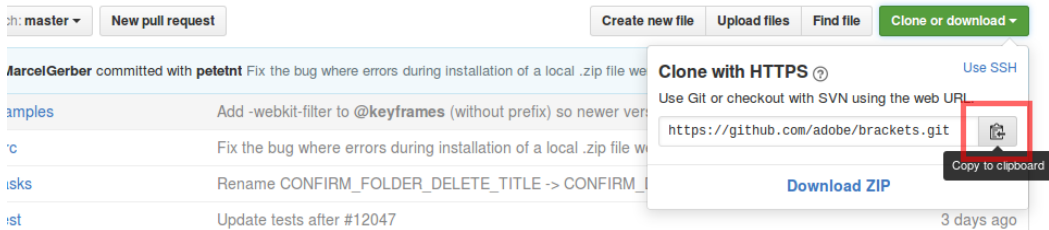


Figura 3.1.5: Copiar dirección en el portapapeles

Teclea `git clone` y pega la dirección del repositorio que tienes en el portapapeles.

```
git clone https://github.com/adobe/brackets.git
```

Deberías obtener una salida parecida a la siguiente.

```
Clonar en «brackets»...
remote: Counting objects: 119156, done.
remote: Compressing objects: 100% (99/99), done.
remote: Total 119156 (delta 55), reused 0 (delta 0), pack-reused 119046
Receiving objects: 100% (119156/119156), 83.78 MiB | 3.87 MiB/s, done.
Resolving deltas: 100% (80989/80989), done.
Comprobando la conectividad... hecho.
```

Ya tienes una copia exacta del proyecto en tu máquina. Puedes trastear y cambiar lo que quieras sin miedo a que afecte al repositorio remoto que se encuentra en GitHub.

## 3.2 Actualización local de repositorios de GitHub

Vimos en el apartado anterior cómo clonar repositorios. Como práctica, hicimos una copia exacta en nuestro ordenador del proyecto **Brackets** que estaba en Github.

Lo habitual es que el contenido de un repositorio vaya cambiando con el tiempo: se arreglan errores, se amplía la funcionalidad con nuevas características, se cambia el aspecto de la aplicación, etc.

Vamos a actualizar los repositorios que clonamos anteriormente; para ello utilizaremos el comando `git pull`.

Es muy importante situarse justo dentro del directorio del proyecto.

```
cd Documentos/Programacion/brackets
```

Procedemos con la actualización.

```
git pull
```

Se muestra la siguiente salida (se ha editado para que no ocupe demasiado):

```

remote: Counting objects: 133, done.
remote: Compressing objects: 100% (73/73), done.
remote: Total 133 (delta 82), reused 57 (delta 57), pack-reused 2
Receiving objects: 100% (133/133), 26.50 KiB | 0 bytes/s, done.
Resolving deltas: 100% (82/82), completed with 8 local objects.
De https://github.com/adobe/brackets
   4be34b6..be62881  master    -> origin/master
Updating 4be34b6..be62881
Fast-forward
 Gruntfile.js                | 1 +
 npm-shrinkwrap.json         | 281 ++++++
 package.json                | 4 +
 src/config.json             | 4 +
 src/filesystem/FileSystem.js | 19 +-
 src/filesystem/FileSystemStats.js | 8 +-
 src/filesystem/WatchedRoot.js | 9 +-
 .../impls/appshell/AppshellFileSystem.js | 72 ++--
 (...)
43 files changed, 836 insertions(+), 2050 deletions(-)
create mode 100644 npm-shrinkwrap.json
create mode 100644 src/filesystem/impls/appshell/node/CSharpWatcher.js
create mode 100644 src/filesystem/impls/appshell/node/ChokidarWatcher.js
create mode 100644 src/filesystem/impls/appshell/node/FileWatcherManager.js
(...)
create mode 100644 src/filesystem/impls/appshell/node/win32/CodeHelper.exe
create mode 100644 src/filesystem/impls/appshell/node/win32/CodeHelper.md
create mode 100644 src/filesystem/impls/appshell/node/win32/LICENSE
create mode 100644 tasks/npm-install.js

```

Como podemos ver, se han modificado 43 ficheros, tenemos la información en la siguiente línea:

```
43 files changed, 836 insertions(+), 2050 deletions(-)
```

Ahora los ficheros y carpetas que hay en el repositorio que está en nuestro ordenador son exactamente iguales que los que hay en el repositorio de GitHub. Recuerda que la actualización hay que realizarla siempre con `git pull` de forma manual. Los archivos

no se sincronizan automáticamente al estilo de otras aplicaciones como Dropbox o Google Drive.



Los repositorios clonados no se actualizan automáticamente. La actualización se debe hacer de forma manual con `git pull`.

Vamos a actualizar el otro repositorio que clonamos anteriormente.

Salimos del directorio del proyecto **Brackets** y entramos en el repositorio que contiene los ejemplos y soluciones del libro [Aprende Java con Ejercicios](#).

```
cd ..  
cd aprende-java-con-ejercicios
```

Procedemos con la actualización.

```
git pull
```

Se muestra la siguiente salida:

```
Already up-to-date.
```

Esto significa que el repositorio está actualizado, es decir, no se ha producido ningún cambio desde que fue clonado.

## 3.3 Creación de repositorios en GitHub

Ya sabemos clonar y actualizar repositorios. Veamos cómo podemos crear nuestros propios proyectos.



Figura 3.3.1: Nuevo repositorio

Haz click en el signo + que hay junto a tu avatar. Selecciona **New repository** en el menú desplegable.

The image shows the 'Create a new repository' form. The title is 'Create a new repository' with a subtitle 'A repository contains all the files for your project, including the revision history.' The form has two columns: 'Owner' and 'Repository name'. The owner is 'Alan Brito Delgado' and the repository name is 'hola-mundo-en-java'. Below this is a 'Description (optional)' field with the text 'Programa "hola mundo" en Java'. There are two radio buttons for visibility: 'Public' (selected) and 'Private'. At the bottom, there is a checked checkbox for 'Initialize this repository with a README' and two dropdown menus for 'Add .gitignore: None' and 'Add a license: None'. A green 'Create repository' button is at the bottom.

Figura 3.3.2: Datos del repositorio

Escribe el nombre del repositorio en el campo **Repository name**. Debe ser un nombre lo más claro y conciso posible. No se permiten espacios en blanco ni caracteres especiales en este campo. Las palabras pueden estar separadas por guiones. Como ejemplo práctico, el repositorio de prueba que vamos a crear contendrá un programa en lenguaje Java que muestra “Hola mundo” por pantalla, así que un nombre perfecto

para este proyecto es `hola-mundo-en-java`.

El campo **Description** es opcional, no obstante es muy recomendable rellenarlo. Esta casilla debe contener la descripción del repositorio, con una línea es suficiente.

Marca la casilla **Initialize this repository with README**. Este paso es muy importante. Al marcar esta opción, se crea el fichero `README.md` que contiene por defecto el nombre y la descripción del repositorio; de esta forma ya hay algo dentro del proyecto, no está vacío y, por tanto, ya lo podemos clonar a nuestro ordenador.

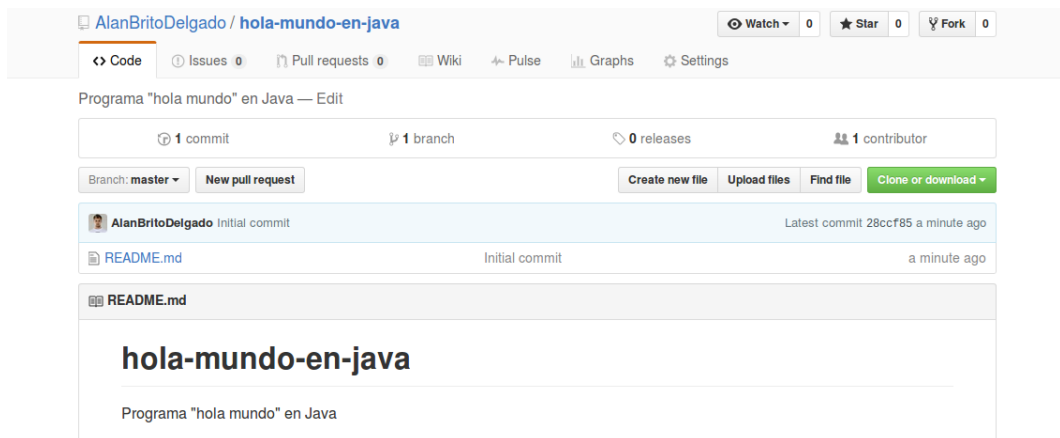


Figura 3.3.3: Vista general del repositorio

Una vez creado el repositorio, aparece una vista general desde la que se puede ver el contenido del mismo y mucha otra información que iremos analizando más adelante.

El repositorio recién creado está en GitHub, para trabajar sobre él lo vamos a clonar en nuestro ordenador. Lo haremos exactamente igual que cuando clonamos repositorios que no eran nuestros. Haz click en el botón **Clone or download** y, a continuación, haz click sobre el icono del portapapeles.

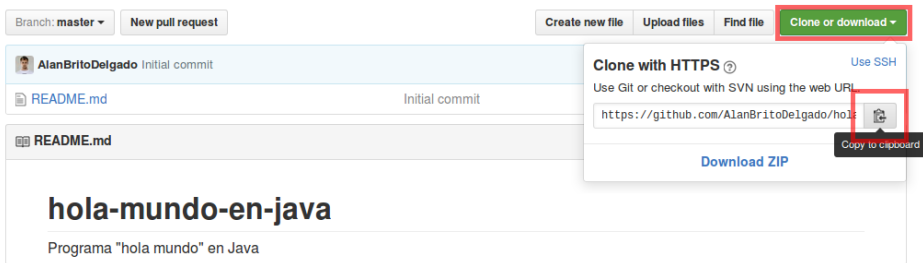


Figura 3.3.4: Copia de la dirección del repositorio

Sitúate dentro del directorio Programacion, donde estamos haciendo las pruebas.

```
cd
cd Documentos/Programacion
```

Clona el repositorio `hola-mundo-en-java`. Recuerda puedes pegar del portapapeles a la ventana de terminal la combinación de teclas **Control + Mayúscula + V**.

```
git clone https://github.com/AlanBritoDelgado/hola-mundo-en-java.git
```

Se debería obtener una salida como la siguiente:

```
Clonar en «hola-mundo-en-java»...
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
Comprobando la conectividad... hecho.
```

¡Enhorabuena! Si has llegado hasta aquí siguiendo todos los pasos, has creado tu primer repositorio en GitHub y lo has clonado en tu ordenador.

## 3.4 Comandos utilizados en este capítulo



### Cambio de directorio

```
cd
cd Documentos/Programacion/
cd Documentos/Programacion/brackets
cd aprende-java-con-ejercicios
```



### Clonación de repositorios

```
git clone https://github.com/LuisJoseSanchez/aprende-java-con-ejercicios.git
git clone https://github.com/adobe/brackets.git
```



### Vista de archivos y directorios en forma de árbol

```
tree
```



### Actualización de un repositorio en local

```
git pull
```

# 4. Flujo de trabajo con Git

## 4.1 Punto de partida

Si has seguido todos los pasos indicados en los capítulos anteriores, has debido crear en GitHub un repositorio con el nombre `hola-mundo-en-java` y lo has clonado en tu máquina. De momento solo contiene el fichero `README.md`.

Sitúate dentro del repositorio `hola-mundo-en-java`<sup>1</sup>.

```
cd ~/Documentos/Programacion/hola-mundo-en-java
```

Mira lo que hay dentro del repositorio.

```
ls
README.md
```

Mira el contenido del fichero `README.md`.

```
cat README.md
# hola-mundo-en-java
Programa "hola mundo" en Java
```

En realidad, dentro del repositorio hay algo más que un fichero. En todos los repositorios existe una carpeta oculta que contiene muchas otras carpetas y ficheros con información sobre el proyecto. Para ver tanto los archivos y directorios visibles como los ocultos se utiliza el comando `ls -a`.

---

<sup>1</sup>El símbolo de la virgulilla (~) se obtiene mediante la combinación de teclas **Alt derecho** + **ñ**.

```
ls -a
.  ..  .git  README.md
```

Para ver todo el contenido de la carpeta `.git` en forma de árbol, puedes utilizar `tree` como hemos visto anteriormente.

```
tree .git
.git
├── branches
├── config
├── description
├── HEAD
├── hooks
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   └── pre-commit.sample
(...)
└── tags
19 directories, 23 files
```

No es necesario modificar manualmente ninguno de estos ficheros, Git lo hace de forma automática.

## 4.2 Ciclo completo de actualización de un repositorio (add, commit, push)

### add

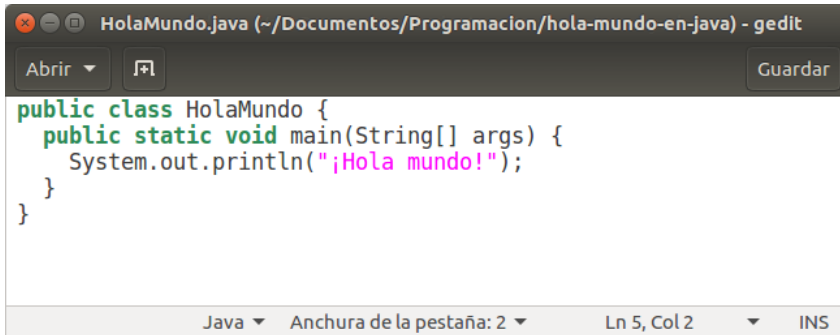
Crea el archivo `HolaMundo.java` dentro del repositorio `hola-mundo-en-java`.

```
touch HolaMundo.java
```

Edita el fichero, por ejemplo con GEdit.

```
gedit HolaMundo.java
```

Escribe un programa en Java que muestre por pantalla la frase “Hola mundo”, como se muestra en la figura y guarda los cambios.



```
public class HolaMundo {
    public static void main(String[] args) {
        System.out.println("¡Hola mundo!");
    }
}
```

Figura 4.2.1: Edición del fichero `HolaMundo.java`

No te preocupes si ahora mismo no conoces el lenguaje de programación Java<sup>2</sup>, de lo que se trata es de añadir archivos al repositorio.

Ahora tenemos un archivo en el ordenador que no está todavía en nuestro repositorio Github, veamos cómo actualizarlo.

Hay un comando muy útil que, en ocasiones, nos puede ayudar a solventar posibles errores y nos da pistas sobre el estado actual del repositorio y sobre cuál es el siguiente paso que hay que dar, se trata de `git status`.

```
git status
En la rama master
Su rama está actualizada con «origin/master».
Archivos sin seguimiento:
  (use «git add <archivo>...» para incluir en lo que se ha de confirmar)
```

```
HolaMundo.java
```

```
no se ha agregado nada al commit pero existen archivos sin seguimiento (use \
«git add» para darle seguimiento)
```

---

<sup>2</sup>Para aprender a programar en Java, te recomiendo el libro [Aprende Java con Ejercicios](#).

Básicamente, lo que nos está diciendo `git status` es que hemos añadido el archivo `HolaMundo.java` a nuestro repositorio pero no lo estamos teniendo en cuenta de cara a futuras actualizaciones. Además nos está dando una pista muy importante sobre el siguiente paso: utilizar el comando `git add` seguido del nombre del archivo.

Para que el fichero `HolaMundo.java` sea añadido al índice de archivos a tener en cuenta, podemos teclear `git add HolaMundo.java`. Ahora bien, lo normal es añadir, modificar y borrar con frecuencia muchos ficheros de un repositorio; hacer `git add` para cada uno de ellos puede resultar tedioso y, lo peor, se nos puede olvidar alguno. En la práctica, lo más cómodo es utilizar `git add . --all` (fíjate que hay un punto detrás de `add`), de esta manera Git chequea todos los archivos que se han añadido al directorio y todas las modificaciones que se han realizado.

```
git add . --all
```

Si después de teclear `git add . --all` no se muestra ningún mensaje, todo va bien.

## **commit**

Veamos cuál es el estado del repositorio.

```
git status
```

```
En la rama master
```

```
Su rama está actualizada con «origin/master».
```

```
Cambios para hacer commit:
```

```
(use «git reset HEAD <archivo>...» para sacar del stage)
```

```
    nuevo archivo: HolaMundo.java
```

Fijémonos en las siguientes líneas:

```
Cambios para hacer commit:
```

```
    nuevo archivo: HolaMundo.java
```

Esto nos está diciendo que en el siguiente *commit* habrá un nuevo archivo: `HolaMundo.java`. ¿Qué quiere decir eso de *commit*? En inglés, *to commit* significa cometer, por tanto, el sustantivo *commit* significa “cometida” o “acometida”, es decir, algo que se ha cometido/hecho/realizado. En un contexto informático, la palabra *commit* no se suele traducir y hace referencia simplemente a los cambios que se han realizado. Llevemos a cabo nuestro primer *commit*.

```
git commit -m "Añadido el fichero HolaMundo.java"
[master 107985c] Añadido el fichero HolaMundo.java
 1 file changed, 5 insertions(+)
 create mode 100644 HolaMundo.java
```

Fíjate que debemos escribir tras la opción `-m` un mensaje explicativo indicando en qué consisten los cambios realizados.

## push

El último paso consiste en “empujar” todos los cambios realizados en local y llevarlos al repositorio que está en GitHub. Para ello se utiliza el comando `git push`.

```
git push
Username for 'https://github.com': AlanBritoDelgado
Password for 'https://AlanBritoDelgado@github.com':
Counting objects: 3, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 423 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/AlanBritoDelgado/hola-mundo-en-java.git
 28ccf85..107985c master -> master
```

Veamos nuestro repositorio en GitHub. Si todo ha ido bien, veremos el fichero `HolaMundo.java` junto a nuestro viejo conocido `README.md`.

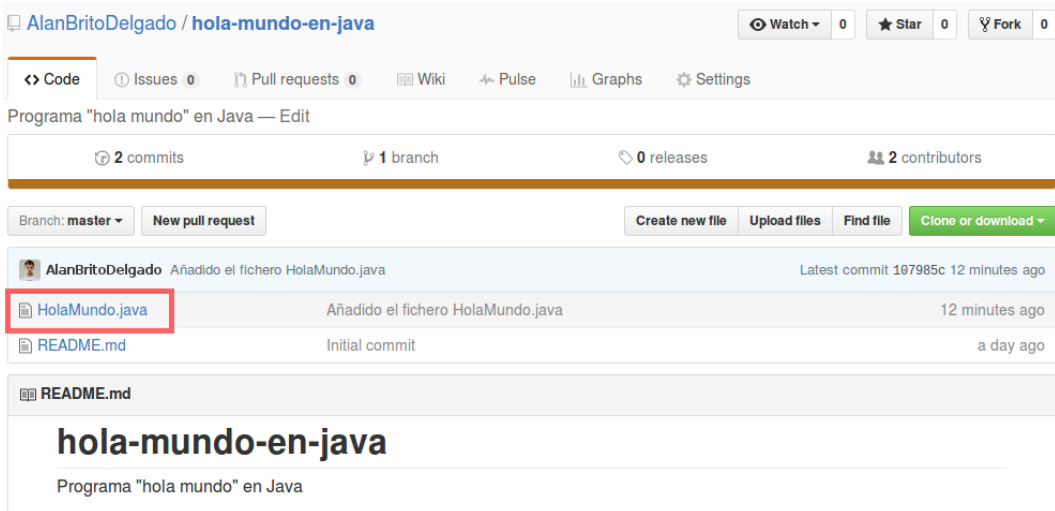


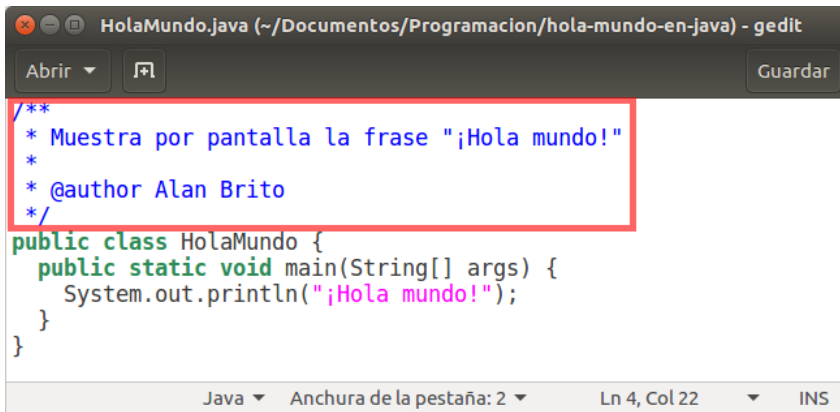
Figura 4.2.2: Repositorio actualizado

## 4.3 Aperitivo, comida y postre

Las reglas mnemotécnicas son muy útiles para recordar casi cualquier cosa. Si eres de buen comer, igual que yo, seguro que puedes recordar perfectamente las palabras **aperitivo**, **comida** y **postre**, que se corresponden con `add`, `commit` y `push` respectivamente. Valiéndote de este pequeño truco, ejecutarás siempre los comandos en el orden adecuado.

Realicemos de nuevo un ciclo completo de actualización con Git.

Modifica tu programa de prueba añadiendo algunos comentarios en el código. Asegúrate de estar dentro del repositorio `hola-mundo-en-java` y edita el fichero `HolaMundo.java`. Añade algunos comentarios al comienzo del archivo tal como se indica en la figura.

A screenshot of a gedit editor window titled "HolaMundo.java (~/Documentos/Programacion/hola-mundo-en-java) - gedit". The window contains Java code with several lines of comments highlighted in blue. The comments are: `/**`, `* Muestra por pantalla la frase ";Hola mundo!"`, `*`, `* @author Alan Brito`, and `*/`. Below the comments is the Java class definition: `public class HolaMundo {`, `public static void main(String[] args) {`, `System.out.println(";Hola mundo!");`, `}`, and `}`. The status bar at the bottom shows "Java", "Anchura de la pestaña: 2", "Ln 4, Col 22", and "INS".

```
/**
 * Muestra por pantalla la frase ";Hola mundo!"
 *
 * @author Alan Brito
 */
public class HolaMundo {
    public static void main(String[] args) {
        System.out.println(";Hola mundo!");
    }
}
```

Figura 4.3.1: Comentarios añadidos a `HolaMundo.java`

Ahora viene el aperitivo, la comida y el postre (add, commit y push).

```
git add . --all
```

```
git commit -m "comentarios añadidos al código"
[master c439564] comentarios añadidos al código
1 file changed, 5 insertions(+)
```

```
git push
Username for 'https://github.com': AlanBritoDelgado
Password for 'https://AlanBritoDelgado@github.com':
Counting objects: 3, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 469 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/AlanBritoDelgado/hola-mundo-en-java.git
107985c..c439564 master -> master
```

Comprueba ahora que en GitHub que el fichero `HolaMundo.java` está actualizado.



The screenshot shows the GitHub interface for the repository 'AlanBritoDelgado / hola-mundo-en-java'. The file 'HolaMundo.java' is selected and highlighted with a red box. The code is displayed in a monospaced font with syntax highlighting. The code includes a multi-line comment and a Java class definition for 'HolaMundo' with a 'main' method that prints '¡Hola mundo!' to the console.

```
1  /**
2   * Muestra por pantalla la frase "¡Hola mundo!"
3   *
4   * @author Alan Brito
5   */
6  public class HolaMundo {
7      public static void main(String[] args) {
8          System.out.println("¡Hola mundo!");
9      }
10 }
```

Figura 4.3.2: Fichero HolaMundo.java actualizado

## 4.4 Comandos utilizados en este capítulo

>\_

### Cambio de directorio

```
cd ~/Documentos/Programacion/hola-mundo-en-java
```

>\_

### Contenido de un directorio

```
ls  
ls -a
```

>\_

### Contenido de un fichero

```
cat README.md
```

>\_

### Vista de archivos y directorios en forma de árbol

```
tree
```

>\_

### Creación de un fichero vacío

```
touch HolaMundo.java
```

**>\_ Edición de un fichero**

```
gedit HolaMundo.java
```

**>\_ Estado de un repositorio**

```
git status
```

**>\_ Adición de ficheros y directorios al índice de elementos a tener en cuenta en el próximo commit**

```
git add . --all
```

**>\_ Commit (confirmación de cambios realizados)**

```
git commit -m "Añadido el fichero HolaMundo.java"
```

**>\_ Actualización en GitHub de los cambios realizados en local**

```
git push
```

# 5. Ficheros README.md, .gitignore y HEAD

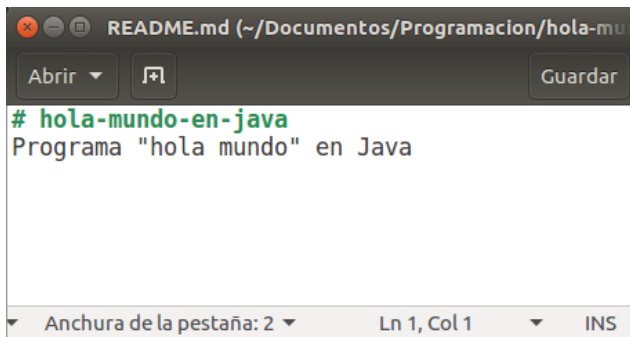
## 5.1 El fichero README.md

El objeto de este fichero es el de proporcionar información sobre el repositorio. Como mínimo debería incluir el título (no tiene por qué coincidir exactamente con el nombre que aparece en la URL) y una descripción.

En muchas ocasiones, el fichero README .md contiene información sobre configuración, instalación y uso de la aplicación contenida en el repositorio.

Como ejemplo práctico utilizaremos el archivo README.md del repositorio hola-mundo-en-java.

Inicialmente, este fichero tiene el contenido que se muestra a continuación.

A screenshot of a text editor window titled "README.md (~/Documentos/Programacion/hola-mu...". The window has a dark theme and contains the following text: "# hola-mundo-en-java" followed by "Programa \"hola mundo\" en Java" on the next line. The status bar at the bottom shows "Anchura de la pestaña: 2", "Ln 1, Col 1", and "INS".

```
README.md (~/Documentos/Programacion/hola-mu
Abrir  +  Guardar
# hola-mundo-en-java
Programa "hola mundo" en Java
Anchura de la pestaña: 2  Ln 1, Col 1  INS
```

Figura 5.1.1: Contenido inicial del fichero README .md

Y luce en GitHub de la manera en que ilustra la figura [figura 5.1.2](#).

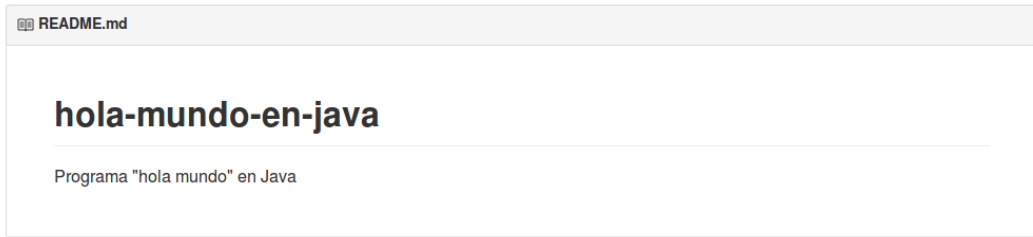
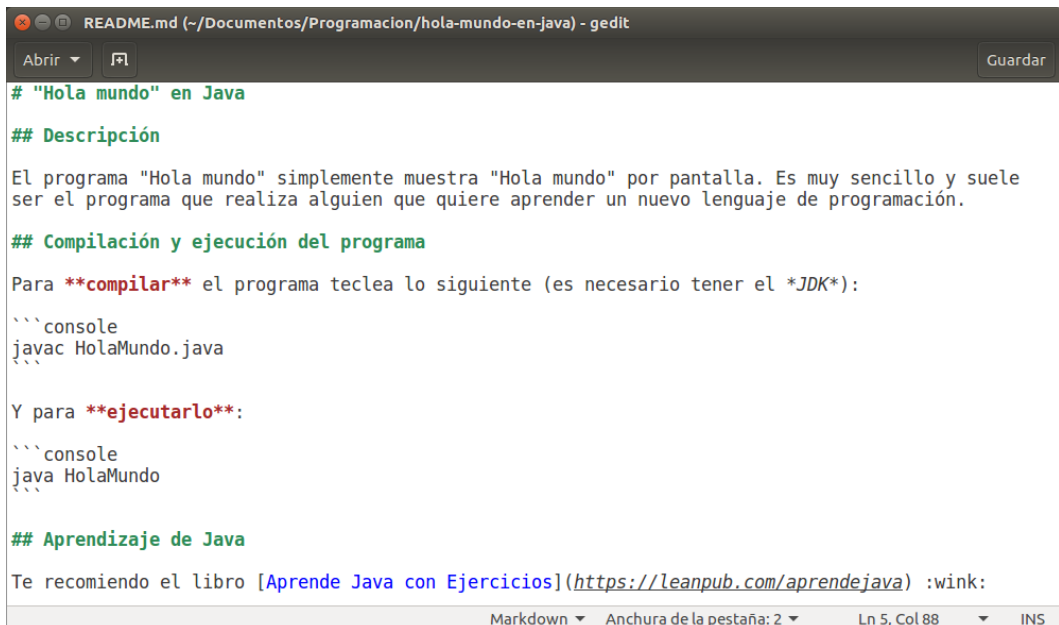


Figura 5.1.2: Visualización del fichero README .md en GitHub

Demasiado escueto ¿no te parece? Es bueno hacer las cosas sencillas, pero hasta cierto límite. Estaría bien ofrecer alguna información adicional. Ten en cuenta que el fichero README .md es lo primero en lo que se fija un usuario una vez que “atteriza” en el repositorio.

Edita el fichero README .md que está dentro del directorio hola-mundo-en-java de tu ordenador y escribe el texto que se muestra en la figura [figura 5.1.3](#).

A screenshot of a gedit editor window showing the updated content of the README.md file. The window title is "README.md (~/Documentos/Programacion/hola-mundo-en-java) - gedit". The editor contains the following markdown text:

```
# "Hola mundo" en Java
## Descripción
El programa "Hola mundo" simplemente muestra "Hola mundo" por pantalla. Es muy sencillo y suele ser el programa que realiza alguien que quiere aprender un nuevo lenguaje de programación.
## Compilación y ejecución del programa
Para compilar el programa teclea lo siguiente (es necesario tener el *JDK*):
```console
javac HolaMundo.java
```
Y para ejecutarlo:
```console
java HolaMundo
```
## Aprendizaje de Java
Te recomiendo el libro [Aprende Java con Ejercicios](https://leanpub.com/aprendejava) :wink:
```

The editor interface includes a menu bar with "Abrir" and "Guardar" buttons, and a status bar at the bottom showing "Markdown", "Anchura de la pestaña: 2", "Ln 5, Col 88", and "INS".

Figura 5.1.3: Fichero README .md actualizado

El archivo README.md está escrito en formato markdown (fíjate en que tiene la extensión .md). Se trata de un lenguaje de marcado al estilo de HTML pero mucho

más sencillo.

Observa que la primera línea está precedida por un carácter de almohadilla (#). Eso significa que esta línea - “Hola mundo” en Java - es una cabecera principal que indica el título del documento completo, o bien, el título de un capítulo. No existe una cabecera más grande, sería el equivalente a la etiqueta h1 de HTML. En el caso que nos ocupa, indica claramente el título del repositorio.

Además del título principal tenemos tres apartados que son “Descripción”, “Compilación y ejecución del programa” y “Aprendizaje de Java”. Se indica que son apartados mediante la doble almohadilla (##). Sería equivalente a la etiqueta h2 de HTML.

Se podrían especificar subapartados con tres almohadillas (###) e incluso subapartados dentro de ellos con cuatro almohadillas (####) que serían equivalentes a las etiquetas h3 y h4 respectivamente.

Todo lo que se coloque entre asteriscos (un asterisco delante y otro detrás) se mostrará en cursiva. La palabra “JDK” se muestra en cursiva.

Cualquier trozo de texto que vaya entre dobles asteriscos (dos asteriscos delante y otros dos detrás) se muestra en negrita. En este caso, “compilar” y “ejecutarlo” aparecen en negrita.

Tres comillas invertidas colocadas antes y después de un texto indican que se trata de código fuente o comandos. Detrás de las primeras comillas se indica el lenguaje de programación: java, javascript, php, etc. En caso de comandos de consola, se escribe la palabra “console”.

En markdown también se pueden insertar hiperenlaces. El texto que debe aparecer se indica entre corchetes y la URL se especifica justo detrás y entre paréntesis.

Una manera de añadirle un poco de sal y pimienta a un fichero README .md consiste en insertar algún que otro emoticono. Igual que con el condimento, conviene no abusar.

[Emoji Cheat Sheet](#) es una página que contiene los emoticonos disponibles que puedes insertar en tus documentos en formato markdown.

Actualiza el archivo README .md en GitHub como en el capítulo anterior.

```
git add . --all
git commit -m "Información interesante en README.md"
(...)
git push
(...)
```



Figura 5.1.4: Visualización del fichero README.md actualizado en GitHub

Si quieres saber más sobre el formato markdown, la página [Mastering Markdown](#) contiene muchos ejemplos.

## 5.2 El fichero .gitignore

Los ficheros que se especifican en `.gitignore` se ignoran por completo y no se incluyen en el seguimiento. Resulta útil para que no se suban al repositorio de GitHub archivos irrelevantes como copias de seguridad, binarios, logs, etc. Lo que interesa tener en GitHub es básicamente el código fuente.

Mediante el carácter almohadilla (#) colocado en la primera posición de la línea se puede indicar un comentario.

Vamos a probar el funcionamiento de `.gitignore`. Colócate dentro del repositorio `hola-mundo-en-java` y lanza el editor.

```
gedit .gitignore
```

Vamos a hacer que Git deje fuera del seguimiento las copias de seguridad (archivos con la extensión `.bak`) y las clases compiladas de Java (archivos con la extensión `.class`).

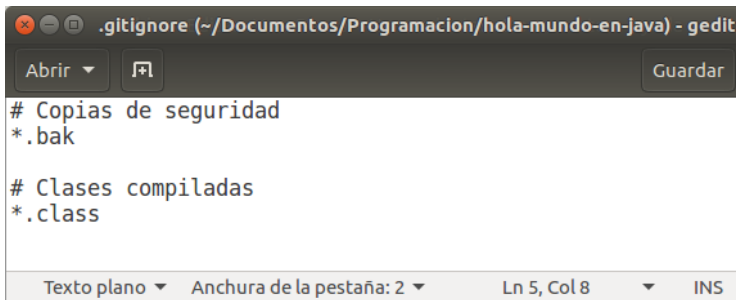


Figura 5.2.1: Fichero `.gitignore`

Crea una copia de seguridad del fichero `HolaMundo.java` y llámala `HolaMundo.java.bak`

```
cp HolaMundo.java HolaMundo.java.bak
```

Compila el fichero `HolaMundo.java` para generar `HolaMundo.class`<sup>1</sup>

```
javac HolaMundo.java
```

Ahora tienes varios ficheros en tu repositorio que no tenías antes.

```
ls -a
.  ..  .git  .gitignore  HolaMundo.class  HolaMundo.java  HolaMundo.java.bak\
  README.md
```

Actualiza tu repositorio en GitHub. Recuerda la regla mnemotécnica “aperitivo, comida y postre”.

---

<sup>1</sup>Para poder compilar archivos Java es necesario tener instalado el JDK (`sudo apt install openjdk-8-jdk`).

```
git add . --all
git commit -m "Información interesante en README.md"
(...)
git push
(...)
```

Comprueba que se ha subido el nuevo fichero `.gitignore` a GitHub y que los archivos `HolaMundo.java.bak` y `HolaMundo.class` no se han subido.

The screenshot shows the GitHub interface for the repository 'hola-mundo-en-java' by user 'AlanBritoDelgado'. The repository has 5 commits, 1 branch, 0 releases, and 2 contributors. The commit history table is as follows:

| Commit                      | Message   | Time        |
|-----------------------------|---|-------------|
| <code>.gitignore</code>     | Añadido el fichero <code>.gitignore</code>        | an hour ago |
| <code>HolaMundo.java</code> | comentarios añadidos al código                    | 2 days ago  |
| <code>README.md</code>      | Información interesante en <code>README.md</code> | 2 days ago  |

Figura 5.2.2: GitHub actualizado con el archivo `.gitignore`

Hay un ejemplo muy completo de fichero `.gitignore` en el repositorio <https://gist.github.com/octocat/9257657>.

## 5.3 El fichero HEAD

Este fichero se encuentra dentro de `.git/logs/` y contiene un registro de los cambios realizados. Veamos su contenido.

```

cat .git/logs/HEAD
00000000000000000000000000000000000000000000000000000000000000000000 28ccf8597660c46c1501b833db62f2b5520\
4ff7c Alan Brito Delgado <alan.brito.delgado.1972@gmail.com> 1472139870 +020\
0      clone: from https://github.com/AlanBritoDelgado/hola-mundo-en-java.git
28ccf8597660c46c1501b833db62f2b55204ff7c 107985caafbce0891ef61678767ed2b8cd9\
528ec Alan Brito Delgado <alan.brito.delgado.1972@gmail.com> 1472200764 +020\
0      commit: Añadido el fichero HolaMundo.java
107985caafbce0891ef61678767ed2b8cd9528ec c439564ff162572573d29dfdfd6feac6ede\
39f56 Alan Brito Delgado <alan.brito.delgado.1972@gmail.com> 1472293710 +020\
0      commit: comentarios añadidos al código
c439564ff162572573d29dfdfd6feac6ede39f56 095a9a4837907e4e08302d56b9bddb2f020\
a1a09 Alan Brito Delgado <alan.brito.delgado.1972@gmail.com> 1472309400 +020\
0      commit: Información interesante en README.md
095a9a4837907e4e08302d56b9bddb2f020a1a09 62757c25174a06e538a6d7f055b72d541c8\
a9fec Alan Brito Delgado <alan.brito.delgado.1972@gmail.com> 1472447469 +020\
0      commit: Añadido el fichero .gitignore

```

Como puedes apreciar, todo lo que has ido haciendo desde el momento en que clonaste el repositorio ha sido registrado.

Mediante `git log` se puede obtener también la información almacenada en este fichero.

```

git log
commit 62757c25174a06e538a6d7f055b72d541c8a9fec
Author: Alan Brito Delgado <alan.brito.delgado.1972@gmail.com>
Date:   Mon Aug 29 07:11:09 2016 +0200

```

Añadido el fichero .gitignore

```

commit 095a9a4837907e4e08302d56b9bddb2f020a1a09
Author: Alan Brito Delgado <alan.brito.delgado.1972@gmail.com>
Date:   Sat Aug 27 16:50:00 2016 +0200

```

Información interesante en README.md

```

commit c439564ff162572573d29dfdfd6feac6ede39f56
Author: Alan Brito Delgado <alan.brito.delgado.1972@gmail.com>

```

Date: Sat Aug 27 12:28:30 2016 +0200

comentarios añadidos al código

commit 107985caafbce0891ef61678767ed2b8cd9528ec

Author: Alan Brito Delgado <alan.brito.delgado.1972@gmail.com>

Date: Fri Aug 26 10:39:24 2016 +0200

Añadido el fichero HolaMundo.java

commit 28ccf8597660c46c1501b833db62f2b55204ff7c

Author: Alan Brito Delgado <alan.brito.delgado.1972@gmail.com>

Date: Thu Aug 25 09:14:25 2016 +0200

Initial commit

El registro de cambios se puede mostrar de muy diversas maneras, ordenado y filtrado según muchos criterios. En este libro no vamos a ver más detalles ya que pretende ser una guía de supervivencia. Si quieres profundizar en el tema, puedes ver todas las opciones que permite `git log` invocando la ayuda `git help log`.

## 5.4 Comandos utilizados en este capítulo

- >— Adición de ficheros y directorios al índice de elementos a tener en cuenta en el próximo commit

```
git add . --all
```

- >— Commit (confirmación de cambios realizados)

```
git commit -m "Añadido el fichero HolaMundo.java"
```

- >— Actualización en GitHub de los cambios realizados en local

```
git push
```

- >— Edición de un fichero

```
gedit .gitignore
```

- >— Copia de un fichero

```
cp HolaMundo.java HolaMundo.java.bak
```

## >\_ **Compilación de una clase de Java**

```
javac HolaMundo.java
```

## >\_ **Cambio de directorio**

```
ls -a
```

## >\_ **Registro de cambios**

```
git log
```

## >\_ **Ayuda sobre un comando**

```
git help log
```

# 6. Versiones y ramas

## 6.1 Etiquetado de versiones

A medida que se avanza en un proyecto y se van completando hitos o se añaden nuevas funcionalidades, es recomendable asignar etiquetas con un nombre y/o un número de versión.

Vamos a etiquetar nuestro repositorio en su estado actual como “v1.0”.

```
git tag -a v1.0 -m 'Versión 1.0 - "Hola mundo" clásico'
```

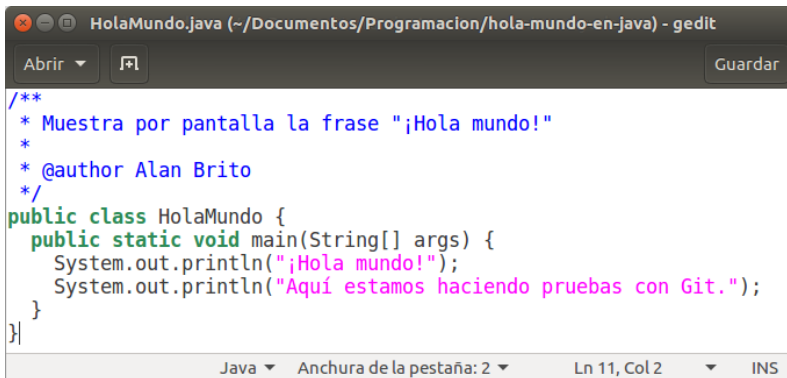
La opción `-a` indica que la etiqueta que estamos creando es de tipo “anotación” y hace que se guarde más información que si creamos una etiqueta “ligera” (sin la opción `-a`)<sup>1</sup>.

La opción `-m` permite dar una descripción de la versión. Es análogo al `-m` que se utiliza en los *commits*. Observa que hemos utilizado comillas simples para esta descripción porque dentro de ella hay, a su vez, comillas dobles.

Edita el fichero `HolaMundo.java` tal como se muestra en la figura [figura 6.1.1](#) con el fin de crear una versión algo más avanzada. En lugar de mostrar una línea por pantalla, mostrará dos líneas.

---

<sup>1</sup>Para más información sobre las diferencias entre etiquetas ligeras y etiquetas anotadas consultar la fuente <https://git-scm.com/book/en/v2/Git-Basics-Tagging>



```
HolaMundo.java (~/Documentos/Programacion/hola-mundo-en-java) - gedit
Abrir Guardar

/**
 * Muestra por pantalla la frase ";Hola mundo!"
 *
 * @author Alan Brito
 */
public class HolaMundo {
    public static void main(String[] args) {
        System.out.println(";Hola mundo!");
        System.out.println("Aquí estamos haciendo pruebas con Git.");
    }
}
```

Figura 6.1.1: Fichero `HolaMundo.java` ampliado.

Nuestro programa ya no es un “Hola mundo” al uso, es algo más avanzado ya que muestra dos líneas por pantalla. Etiqueta el repositorio indicando que se trata de la versión “v1.1”.

```
git tag -a v1.1 -m 'Versión 1.1 - Saludo ampliado'
```

Muestra todas las versiones que tienes hasta el momento (con la opción `-n` se muestran también las descripciones).

```
git tag -n
v1.0          Versión 1.0 - "Hola mundo" clásico
v1.1          Versión 1.1 - Saludo ampliado
```

Como era de esperar, comprobamos que tenemos dos versiones de nuestro programa. La primera versión (v1.0) muestra una línea por pantalla y la segunda (v1.1) muestra dos líneas.

Actualiza el repositorio en GitHub.

```
git add . --a
git commit - m "Saludo ampliado"
(...)
git push
(...)
```

Comprueba que el fichero `HolaMundo.java` se ha actualizado en tu GitHub.



The screenshot shows the GitHub interface for a repository named 'hola-mundo-en-java'. The current branch is 'master'. A commit by 'Alan Brito Delgado' with the message 'Saludo ampliado' is displayed, showing 1 contributor. The file 'HolaMundo.java' is selected, showing 12 lines (11 sloc) and 264 Bytes. The code content is as follows:

```
1  /**
2   * Muestra por pantalla la frase ";Hola mundo!"
3   *
4   * @author Alan Brito
5   */
6  public class HolaMundo {
7      public static void main(String[] args) {
8          System.out.println(";Hola mundo!");
9          System.out.println("Aquí estamos haciendo pruebas con Git.");
10     }
11 }
```

Figura 6.1.2: Fichero `HolaMundo.java` actualizado en GitHub.

Efectivamente, en GitHub se encuentra la versión más reciente del fichero `HolaMundo.java`.

Busquemos ahora las etiquetas. Haz click en el botón **Branch: master** (figura 6.1.3) y, a continuación haz click en la pestaña **Tags** ¿Qué nos encontramos? Un descorazonador **Nothing to show**, a pesar de que el repositorio está correctamente actualizado no hay ni rastro de las etiquetas ¿Que no cunda el pánico! vamos a ver lo que ha sucedido.

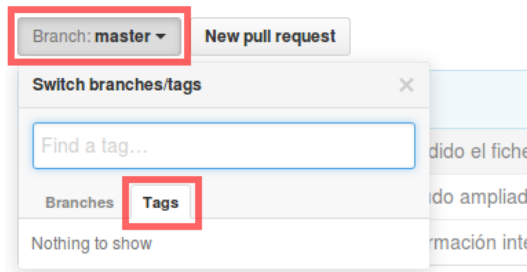


Figura 6.1.3: No aparecen etiquetas

Por defecto, el comando `git push` no actualiza las etiquetas en el repositorio remoto, es necesario indicarlo de forma explícita con la opción `--tags`.

```
git push --tags
Counting objects: 2, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 269 bytes | 0 bytes/s, done.
Total 2 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/AlanBritoDelgado/hola-mundo-en-java.git
 * [new tag]          v1.0 -> v1.0
 * [new tag]          v1.1 -> v1.1
```

Busquemos de nuevo las etiquetas. Haz click en el botón **Branch: master** (figura 6.1.4) y luego en **Tags**. Ahora sí podemos ver nuestras etiquetas `v1.0` y `v1.1` y, lo que es todavía mejor, podemos cambiar de una versión a otra. Haz clic en **v1.0**.

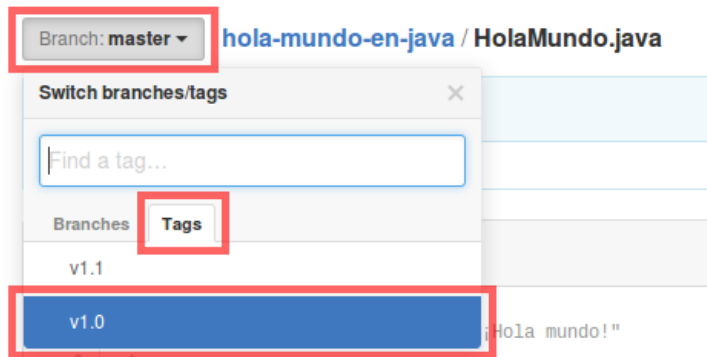


Figura 6.1.4: Etiquetas v1.0 y v1.1

Comprueba ahora cuál es el contenido de `HolaMundo.java`.

Figura 6.1.5: Fichero `HolaMundo.java` en la versión v1.0

Como vemos en la figura [figura 6.1.5](#), al volver a la versión v1.0, tenemos el `HolaMundo.java` que muestra una sola línea por pantalla. Puedes saltar en cualquier momento de una versión a otra. Para volver a mostrar **Branch: master** en lugar del nombre de la etiqueta, simplemente haz click en el botón **Tag: v1.0** (o **Tag: v1.1** si estás en la versión v1.1) y, en la pestaña **Branches** haz click en **master**.

## 6.2 Ramas

Una rama es un flujo en el cual se van haciendo cambios en el código (modificando, borrando o añadiendo archivos). Por defecto, cuando se crea un repositorio, el flujo principal se llama rama `master` o rama principal.

Cuando se crea una rama adicional, ese flujo se bifurca, es decir, se puede optar por seguir por la rama principal o seguir por la nueva rama. Es posible saltar de una rama a otra y continuar avanzando por cualquiera de ellas.

Dado el caso, la rama creada se puede fusionar con la rama `master` y volveríamos a tener una única rama como al principio.

Vamos a realizar un ejemplo práctico. Crearemos una rama experimental en la que haremos pruebas con el fichero `HolaMundo.java`. Intentaremos mostrar los mensajes por pantalla en colores. Si lo conseguimos, fusionaremos la rama experimental con la rama `master`.

Para crear una rama se utiliza el comando `git branch` y para saltar de una rama a otra `git checkout`. Se pueden realizar las dos acciones de una vez con `git checkout -b`.

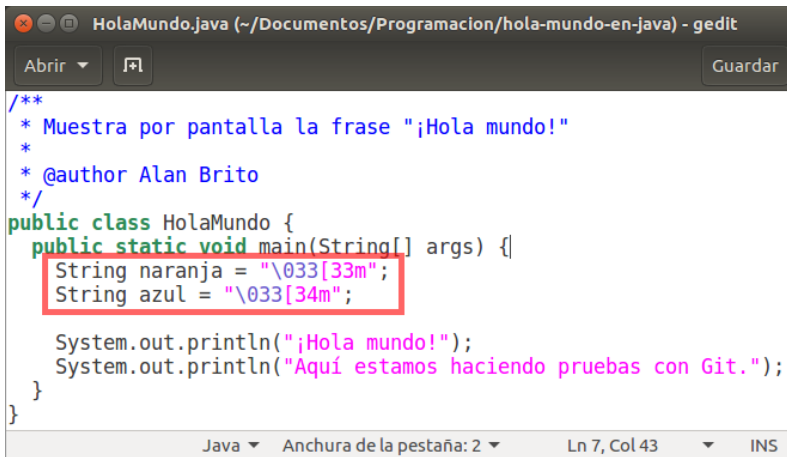
Crea la rama `experimentocolores` y, a la vez, salta a esa rama.

```
git checkout -b experimentocolores
Switched to a new branch 'experimentocolores'
```

Asegúrate de estar en la rama nueva.

```
git status
En la rama experimentocolores
nothing to commit, working directory clean
```

Edita el archivo `HolaMundo.java` y define los colores como se muestra en la [figura 6.2.1](#).

A screenshot of a code editor window titled 'HolaMundo.java (~/Documentos/Programacion/hola-mundo-en-java) - gedit'. The editor shows Java code with various colors: blue for comments, green for class and method declarations, and pink for string literals. Two lines of string literals are highlighted with a red box: 'String naranja = "\033[33m";' and 'String azul = "\033[34m";'. The status bar at the bottom indicates 'Java', 'Anchura de la pestaña: 2', 'Ln 7, Col 43', and 'INS'.

```
/**
 * Muestra por pantalla la frase "¡Hola mundo!"
 *
 * @author Alan Brito
 */
public class HolaMundo {
    public static void main(String[] args) {
        String naranja = "\033[33m";
        String azul = "\033[34m";

        System.out.println("¡Hola mundo!");
        System.out.println("Aquí estamos haciendo pruebas con Git.");
    }
}
```

Figura 6.2.1: Colores definidos en `HolaMundo.java`.

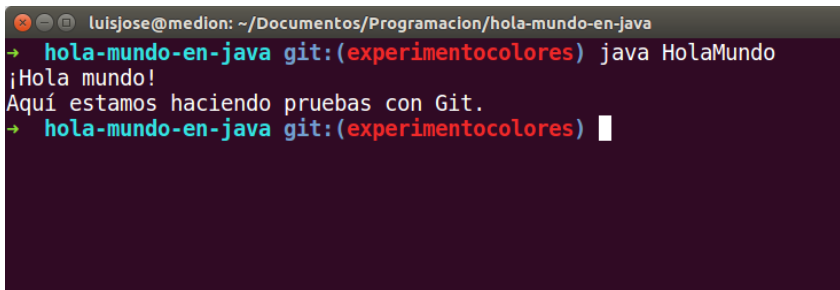
Actualiza el repositorio remoto. Como no te encuentras en la rama `master`, debes indicar que estás actualizando desde la rama `experimentocolores` mediante el comando `git push --set-upstream origin experimentocolores`. Para la siguiente actualización, Git recordará la rama y no hará falta especificarla otra vez.

```
git add . --a
git commit -m "Colores definidos"
(...)
git push --set-upstream origin experimentocolores
(...)
```

Compila el programa.

```
javac HolaMundo.java
```

Veamos si el programa hace lo que queremos.

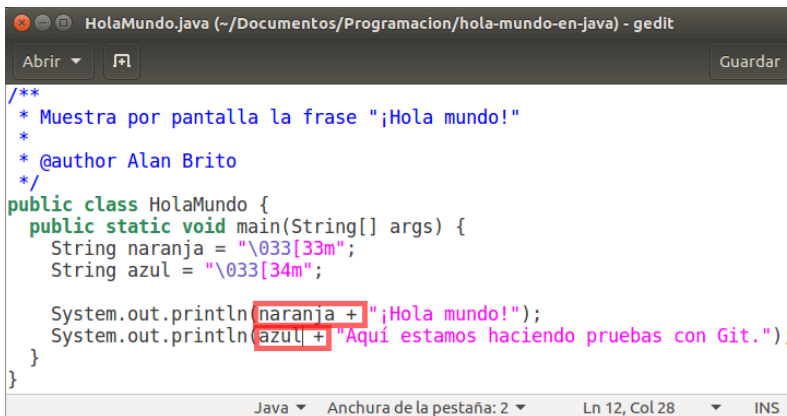


```
luisjose@medion: ~/Documentos/Programacion/hola-mundo-en-java
→ hola-mundo-en-java git:(experimentocolores) java HolaMundo
¡Hola mundo!
Aquí estamos haciendo pruebas con Git.
→ hola-mundo-en-java git:(experimentocolores) █
```

Figura 6.2.2: Ejecución de Ho1aMundo en consola.

Las dos líneas siguen saliendo en blanco ¿por qué? muy fácil, hemos definido los colores pero no los hemos aplicado al pintar las líneas.

Edita el archivo `HolaMundo.java` y aplica los colores a los mensajes que queremos mostrar.



```
HolaMundo.java (~/Documentos/Programacion/hola-mundo-en-java) - gedit
Abrir Guardar

/**
 * Muestra por pantalla la frase "¡Hola mundo!"
 *
 * @author Alan Brito
 */
public class HolaMundo {
    public static void main(String[] args) {
        String naranja = "\033[33m";
        String azul = "\033[34m";

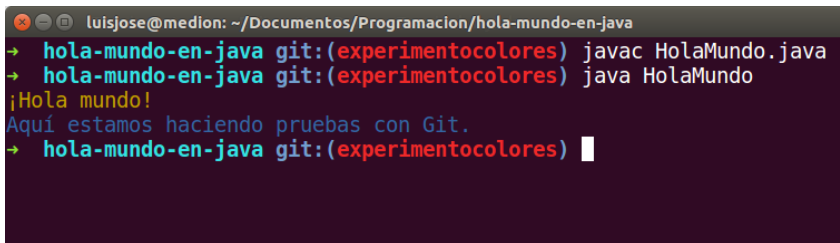
        System.out.println(naranja + "¡Hola mundo!");
        System.out.println(azul + "Aquí estamos haciendo pruebas con Git.");
    }
}
```

Figura 6.2.3: Fichero Ho1aMundo.java con colores.

Actualiza el repositorio remoto. Como en la última actualización especificamos que estábamos en la rama `experimentocolores` no es necesario indicarlo de nuevo.

```
git add . --a
git commit -m "Saludo en colores"
(...)
git push
(...)
```

Compila y ejecuta el programa.



```
luisjose@medion: ~/Documentos/Programacion/hola-mundo-en-java
→ hola-mundo-en-java git:(experimentocolores) javac HolaMundo.java
→ hola-mundo-en-java git:(experimentocolores) java HolaMundo
¡Hola mundo!
Aquí estamos haciendo pruebas con Git.
→ hola-mundo-en-java git:(experimentocolores) █
```

Figura 6.2.4: Ejecución de HolaMundo con colores en consola.

¡Bravo! Ahora sí ha salido lo que queríamos. La primera línea aparece escrita en color naranja y la segunda en azul.

Veamos cómo se muestran las ramas en GitHub.

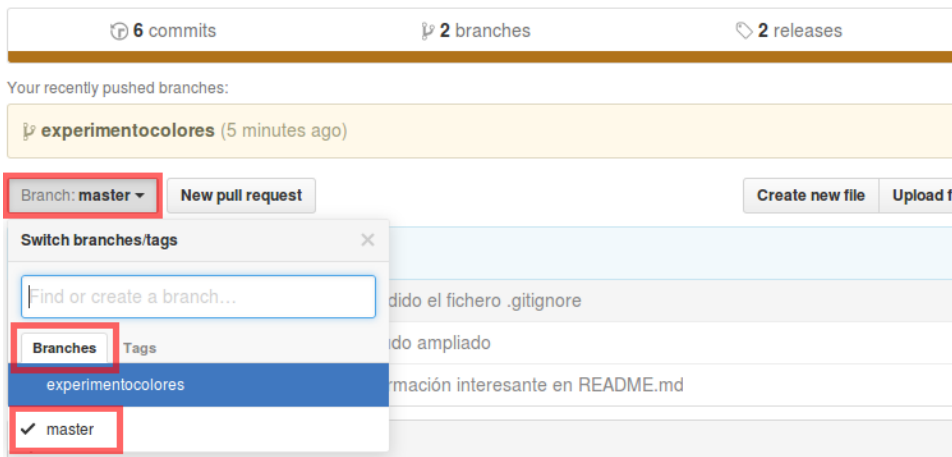


Figura 6.2.5: Ramas en GitHub.

Hay un mensaje destacado avisando que se ha hecho una actualización en una rama.

Si haces click en **Branch: master** y luego en la pestaña **Branches** verás que aparecen las dos ramas: master y experimentocolores.

Seleccionando master se puede ver cómo está HolaMundo.java en la rama principal.



```
Branch: master hola-mundo-en-java / HolaMundo.java
Alan Brito Delgado Saludo ampliado
1 contributor
12 lines (11 sloc) | 264 Bytes
1 /**
2  * Muestra por pantalla la frase "¡Hola mundo!"
3  *
4  * @author Alan Brito
5  */
6 public class HolaMundo {
7     public static void main(String[] args) {
8         System.out.println("¡Hola mundo!");
9         System.out.println("¡Aquí estamos haciendo pruebas con Git.");
10    }
11 }
```

Figura 6.2.6: HolaMundo.java en la rama master.

Así mismo, seleccionando la rama experimentocolores se puede ver el HolaMundo.java que muestra los mensajes en colores.



Branch: **experimentocol...** [hola-mundo-en-java / HolaMundo.java](#)

**Alan Brito Delgado** Saludo en colores

1 contributor

15 lines (13 sloc) | 345 Bytes Raw

```

1  /**
2  * Muestra por pantalla la frase ";Hola mundo!"
3  *
4  * @author Alan Brito
5  */
6  public class HolaMundo {
7      public static void main(String[] args) {
8          String naranja = "\033[33m";
9          String azul = "\033[34m";
10
11         System.out.println(naranja + ";Hola mundo!");
12         System.out.println(azul + "Aquí estamos haciendo pruebas con Git.");
13     }
14 }

```

Figura 6.2.7: `HolaMundo.java` en la rama `experimentocolores`.

Nuestro experimento ha sido todo un éxito. Ahora, vamos a fusionar la rama `experimentocolores` con la rama principal.

Primero salta a la rama `master` con `git checkout`.

```

git checkout master
Switched to branch 'master'
Su rama está actualizada con «origin/master».

```

Y ahora fusiona la rama `master` con `experimentocolores` mediante `git merge`.

```

git merge experimentocolores
Updating 87a2d1a..9e85b66
Fast-forward
 HolaMundo.java | 7 +++++--
 1 file changed, 5 insertions(+), 2 deletions(-)

```

Actualiza la fusión de las dos ramas en el repositorio remoto.

```
git push
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/AlanBritoDelgado/hola-mundo-en-java.git
 87a2d1a..9e85b66  master -> master
```

Veamos cómo ha quedado el repositorio en GitHub. Observa que sigues teniendo dos ramas.



Branch: **master** | [hola-mundo-en-java](#) / **HolaMundo.java**

**AlanBritoDelgado** Saludo en colores

1 contributor

15 lines (13 sloc) | 345 Bytes

```
1  /**
2   * Muestra por pantalla la frase ";Hola mundo!"
3   *
4   * @author Alan Brito
5   */
6  public class HolaMundo {
7      public static void main(String[] args) {
8          String naranja = "\033[33m";
9          String azul = "\033[34m";
10
11         System.out.println(naranja + ";Hola mundo!");
12         System.out.println(azul + "Aquí estamos haciendo pruebas con Git.");
13     }
14 }
```

Figura 6.2.8: Estado de las ramas en GitHub.

Pero ahora, si miras el fichero `HolaMundo.java` de la rama `master` verás que es el mismo que había en la rama `experimentocolores`. Ya podemos seguir trabajando en la rama principal.

## 6.3 Comandos utilizados en este capítulo

### >\_ Etiquetado de versiones

```
git tag -a v1.0 -m 'Versión 1.0 - "Hola mundo" clásico'  
git tag -a v1.1 -m 'Versión 1.1 - Saludo ampliado'
```

### >\_ Listado de todas las etiquetas existentes con sus correspondientes descripciones

```
git tag -n
```

### >\_ Adición de ficheros y directorios al índice de elementos a tener en cuenta en el próximo commit

```
git add . --all
```

### >\_ Commit (confirmación de cambios realizados)

```
git commit -m "Añadido el fichero HolaMundo.java"
```

### >\_ Actualización en GitHub de los cambios realizados en local

```
git push
```

### >\_ Actualización de las etiquetas en el repositorio remoto

```
git push --tags
```

### >\_ Creación de la rama `experimentocolores` y cambio de contexto hacia esa rama

```
git checkout -b experimentocolores
```

### >\_ Estado del repositorio

```
git status
```

### >\_ Actualización del repositorio remoto indicando que los cambios se hacen desde una rama distinta a la principal

```
git push --set-upstream origin experimentocolores
```

### >\_ Compilación de una clase de Java

```
javac HolaMundo.java
```



### **Cambio de contexto hacia la rama `master`**

```
git checkout master
```



### **Fusión de ramas**

```
git merge experimentocolores
```

# 7. Desarrollo colaborativo (fork y pull request)

Una de las características más importantes, si no la que más, que hace atractivo a GitHub es la posibilidad de desarrollo colaborativo.

Cuatro ojos ven más que dos, y seis ven más que cuatro, y ocho ven más que seis... Exponer el código de un proyecto en un repositorio público de GitHub significa que habrá miles de miradas potenciales sobre ese código. Por tanto, si el proyecto interesa a la comunidad, enseguida se formará un grupo de programadores en torno al proyecto que detecten fallos y que sugieran mejoras.

Supón que detectas un fallo en un repositorio de GitHub o simplemente quieres mejorar algún aspecto del proyecto. No tiene que ser algo excepcional, puede ser tan simple como corregir una falta de ortografía o añadir un comentario. En definitiva, quieres contribuir en el progreso del proyecto. Los pasos a seguir en el desarrollo colaborativo serían los siguientes:

- Hacer un *fork* del repositorio en el que se quiere contribuir. En la propia cuenta de GitHub aparecerá una rama (una copia) de ese repositorio.
- Clonar ese repositorio en local.
- Realizar los cambios o añadidos pertinentes en el código.
- Actualizar el repositorio remoto.
- Hacer un *pull request*, es decir, pedir permiso al dueño del repositorio original para que incorpore los cambios.
- El dueño del repositorio original revisa los cambios que ha hecho el voluntario y, si lo cree oportuno, acepta el *pull request* para que las modificaciones sean efectivas.

Con un ejemplo práctico se verá todo mucho más claro. Utilizaremos como base una historia. Alan escribirá el principio de un cuento. A Elena le parece muy interesante

y hace su aportación trabajando sobre una copia. Finalmente, Elena le pide permiso a Alan para que su interesante contribución sea incorporada a la historia original. Veamos todo el proceso paso a paso.

Alan crea el repositorio `las-aventuras-de-paco-el-pulpogato` haciendo click en el botón **New repository**.

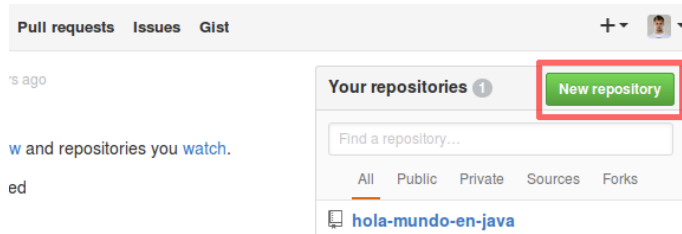


Figura 7.1: Creación del repositorio `las-aventuras-de-paco-el-pulpogato`.

A continuación, Alan le da un nombre y una descripción al repositorio (figura 7.2). Observa que es importante marcar la casilla **Initialize this repository with a README**.

### Create a new repository

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

**Owner**  / **Repository name**

Great repository names are short and memorable. Need inspiration? How about [literate-octo-carnival](#).

**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: **None** ⓘ

Figura 7.2: Creación del repositorio `las-aventuras-de-paco-el-pulpogato`.

Una vez creado el repositorio, Alan lo clona en local. Para ello, primero hace click en el botón **Clone or download** y luego en el botón con el icono del portapapeles para copiar la dirección del repositorio al portapapeles.

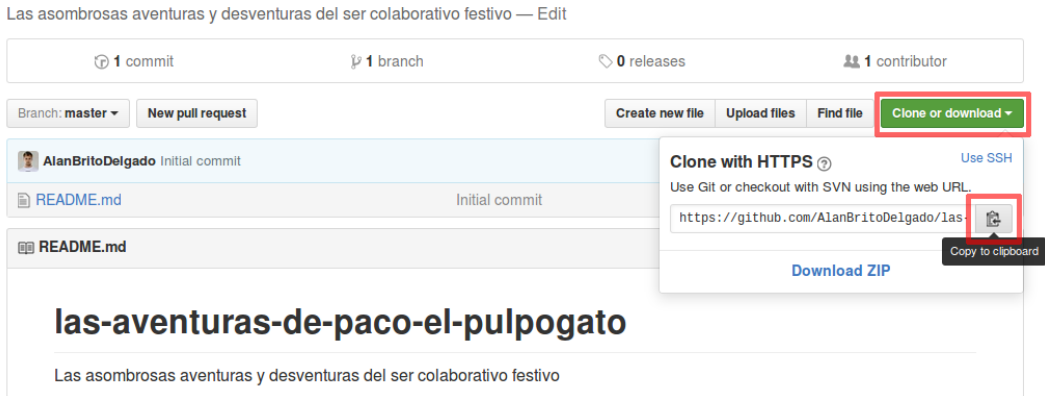


Figura 7.3: Copia de la dirección del repositorio `las-aventuras-de-paco-el-pulpogato` al portapapeles.

Para clonar el repositorio en local, recuerda que se utiliza el comando `git clone` seguido de la dirección del repositorio que se quiere copiar. En este caso, la dirección está en el portapapeles.

```
git clone https://github.com/AlanBritoDelgado/las-aventuras-de-paco-el-pulpo\
gato.git
(...)
```

Ahora toca editar el archivo `README.md`

```
cd las-aventuras-de-paco-el-pulpogato/
gedit README.md
```

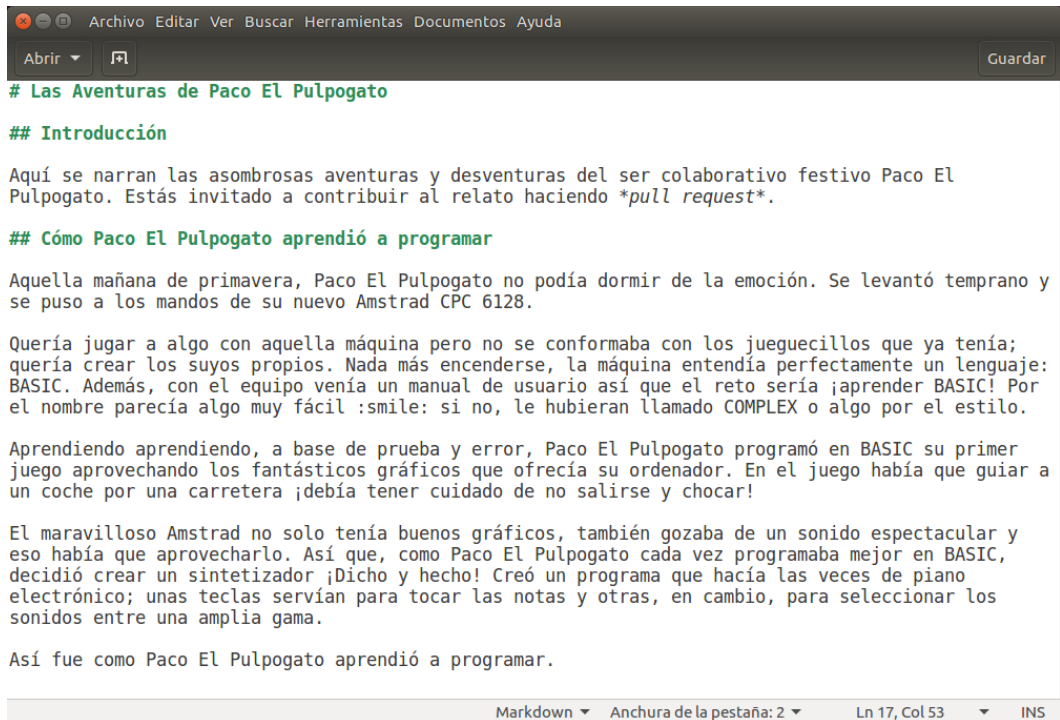


Figura 7.4: Edición del archivo README.md

Como ves, nuestro personaje Alan ya ha escrito el inicio de la historia. Para actualizarla en remoto, tan solo hay que usar los comandos `git add`, `git commit` y `git push` - recuerda la regla mnemotécnica de “aperitivo, comida y postre”.

```
git add . --all
git commit -m "Inicio de la historia"
(...)
git push
(...)
```

Veamos cómo luce la historia sobre Paco El Pulpogato en GitHub.

The screenshot shows a GitHub repository page for 'Las Aventuras de Paco El Pulpogato'. At the top, there are navigation buttons: 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. Below this, a commit message from 'Alan Brito Delgado' is visible, dated 'Update README.md' with the commit hash '8a7a148' and the time 'a minute ago'. The main content area displays the README file, which is titled 'Las Aventuras de Paco El Pulpogato' and includes sections for 'Introducción' and 'Cómo Paco El Pulpogato aprendió a programar'. The text in the README describes the adventures of Paco El Pulpogato, a collaborative and festive character, and details how he learned to program on an Amstrad CPC 6128 computer using BASIC.

Figura 7.5: Repositorio `las-aventuras-de-paco-el-pulpogato` en GitHub.

Alan ha encendido la chispa. A partir de ahora puede que se sume a su proyecto un buen puñado de usuarios, o miles o tal vez millones...

De momento, a Elena le gusta la iniciativa. A ella también le gusta escribir historias, así que se pone manos a la obra.

Desde su cuenta, Elena encuentra el repositorio `las-aventuras-de-paco-el-pulpogato` de Alan. Fíjate bien en la [figura 7.6](#). La usuaria logueada es Elena y el repositorio pertenece a Alan.

El primer paso que tiene que dar Elean es hacer un *fork*, es decir, crear una rama (una copia) en su propia cuenta.

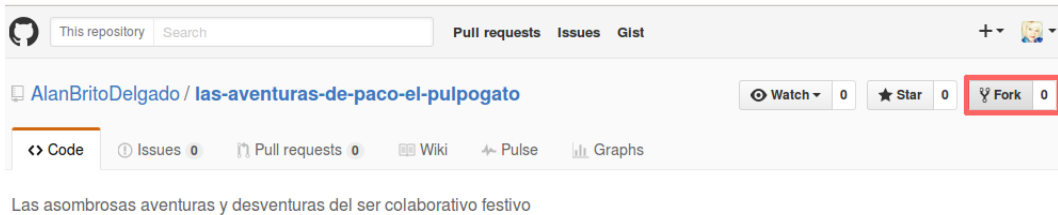


Figura 7.6: Fork de `las-aventuras-de-paco-el-pulpogato`.

Elena hace click en el botón **Fork**<sup>1</sup> y comienza el proceso de copia, que puede durar unos segundos.



Figura 7.7: Copia de `AlanBritoDelgado/las-aventuras-de-paco-el-pulpogato` en `ElenaKozina/las-aventuras-de-paco-el-pulpogato`.

Una vez que Elena tiene una copia de la historia, tiene que clonarla en su ordenador para trabajar con ella. Ya hemos visto varias veces en este manual cómo se hace esto. Elena hace click en el botón **Clone or download** y luego en el botón con el icono del portapapeles para copiar la dirección del repositorio al portapapeles.

<sup>1</sup>Al pasar el ratón por encima del botón **Fork** aparece el mensaje *Fork your own copy of AlanBritoDelgado/las-aventuras-de-paco-el-pulpogato to your account* que quiere decir en español "Bifurca/ramifica tu propia copia de AlanBritoDelgado/las-aventuras-de-paco-el-pulpogato en tu propia cuenta".

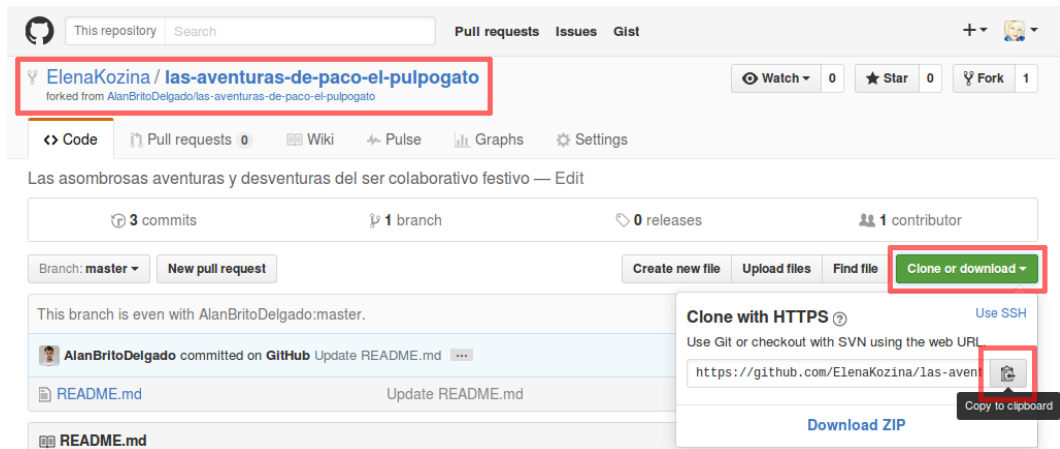


Figura 7.8: Copia de al portapapeles de la dirección del repositorio.

Observa en la [figura 7.8](#) que el repositorio es ElenaKozina/las-aventuras-de-paco-el-pulpogato y justo a la izquierda de este nombre aparece un icono que indica que se trata de un repositorio *forkeado* (copiado de otro). Debajo del nombre del repositorio se puede leer forked from AlanBritoDelgado/las-aventuras-de-paco-el-pulpogato que es el repositorio original, el que se ha copiado.

A continuación, Elena ejecuta el comando `git clone` desde una ventana de terminal.

```
git clone https://github.com/ElenaKozina/las-aventuras-de-paco-el-pulpogato.\
git
Cloning into 'las-aventuras-de-paco-el-pulpogato'...
remote: Counting objects: 9, done.
remote: Compressing objects: 100% (5/5), done.
Unpacking objects: 100% (9/9), done.
remote: Total 9 (delta 1), reused 9 (delta 1), pack-reused 0
Checking connectivity... done.
```

Observa que Elena ha clonado su propia copia, es decir, el repositorio ElenaKozina/las-aventuras-de-paco-el-pulpogato.

Ahora Elena edita el archivo `README.md` para añadir unas líneas a la historia original.

```
cd las-aventuras-de-paco-el-pulpogato/
gedit README.md
```

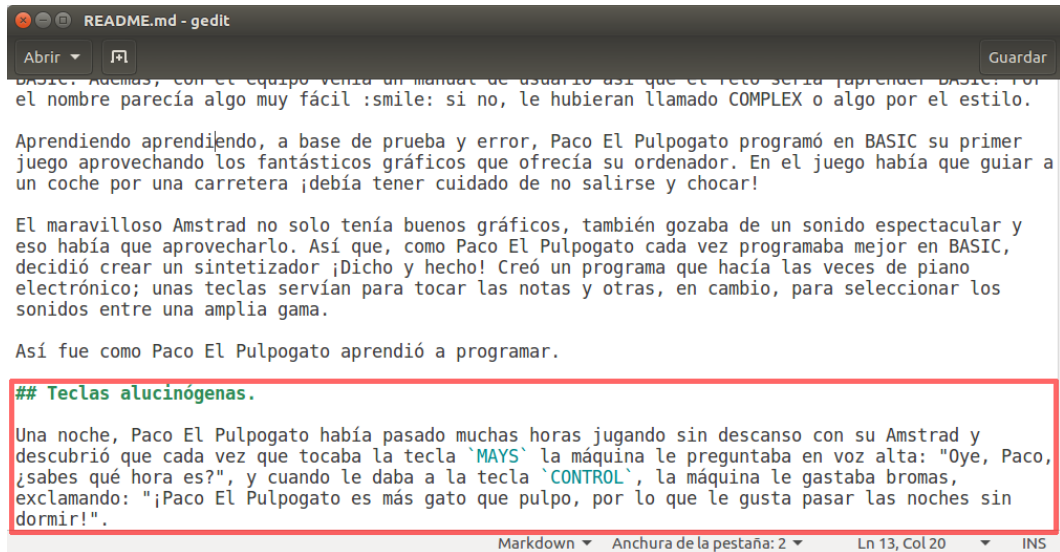


Figura 7.9: Edición del archivo README .md.

Elena ha añadido un nuevo capítulo titulado “Teclas alucinógenas”.

Una vez editado y guardado el archivo README.md en local, hay que actualizar el repositorio remoto.

```
git add . --all
git commit -m "Capítulo nuevo"
(...)
git push
(...)
```

Ahora el repositorio ElenaKozina/las-aventuras-de-paco-el-pulpogato está actualizado y contiene las líneas que ha añadido Elena como se puede comprobar en la [figura 7.10](#).



Figura 7.10: Repositorio `las-aventuras-de-paco-el-pulpogato` en GitHub (las franjas grises indican por dónde se ha recortado la imagen).

A Elena se le ocurren nuevas ideas y quiere seguir añadiendo algunas líneas al capítulo que ha escrito. Podría repetir el proceso que hemos visto, es decir, editar en local el archivo `README.md` y luego ejecutar los comandos `git add`, `git commit` y `git push` para actualizar el repositorio remoto. Sin embargo decide utilizar una función muy útil que tiene GitHub, esto es, editar el archivo `README.md` *on-line*, o sea, desde su cuenta de GitHub.

Elena hace click sobre el enlace al fichero `README.md` en color azul (ver figura 7.10).

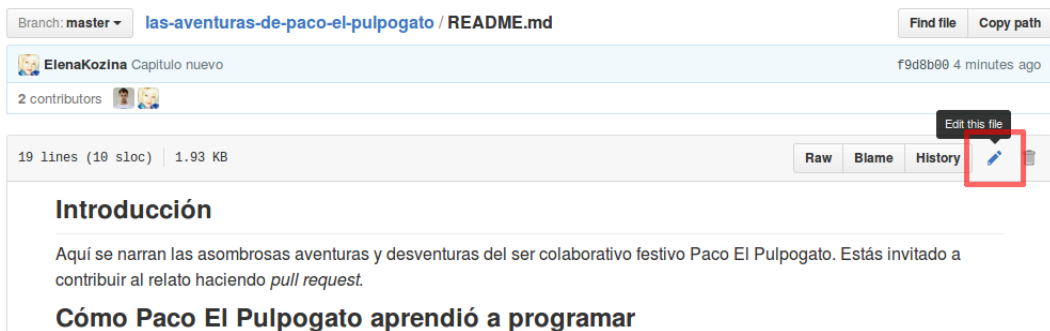
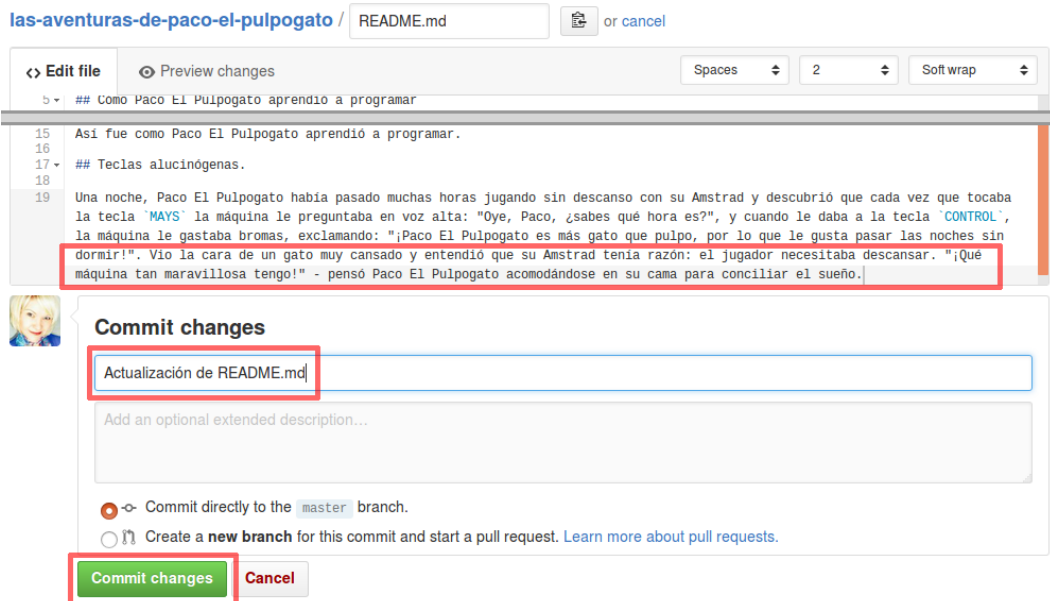


Figura 7.11: Archivo README.md en GitHub.

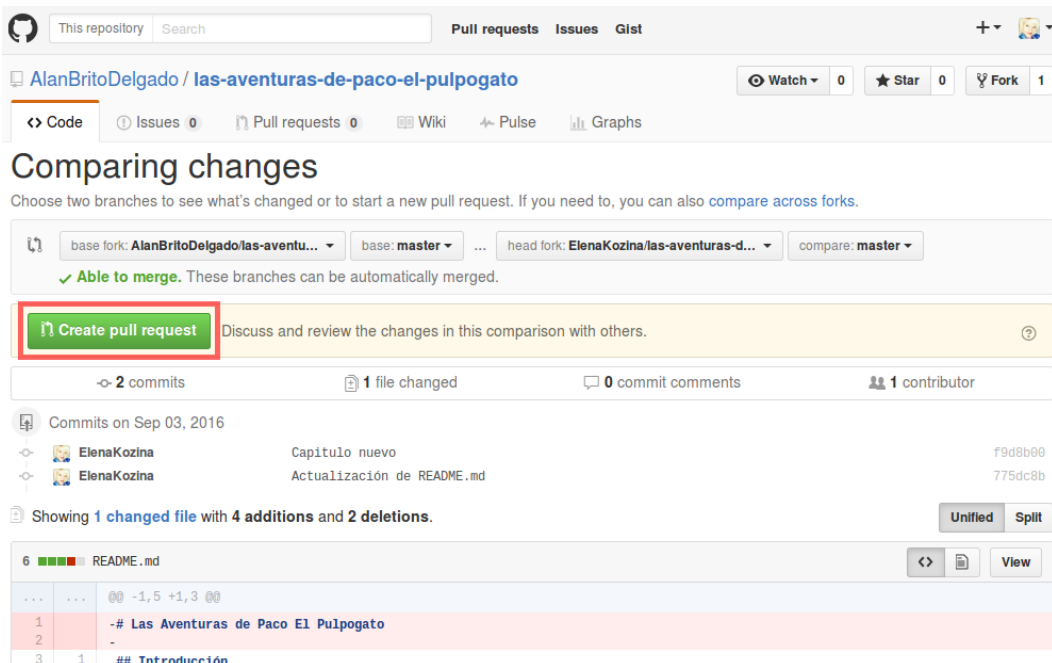
Como se puede ver en la [figura 7.11](#), aparece el contenido del fichero README.md. Hay dos iconos, uno de un lápiz que permite acceder al área de edición, y otro de una papelerera que da la posibilidad de borrar el archivo. Cuando se pasa el ratón por encima del icono del lápiz aparece el mensaje *Edit this file* que significa en español “Editar este fichero”. Elena hace click en el icono de edición.

Figura 7.12: Edición *on-line* del fichero README.md (la franja gris indica por dónde se ha recortado la imagen).

Como se puede ver en la [figura 7.12](#) Elena añade algo de texto hasta completar su

capítulo “Teclas alucinógenas”, escribe un comentario indicando lo que ha hecho (Actualización de README.md) y hace click en el botón **Commit changes**.

Observa en la [figura 7.13](#) que Elena ha realizado dos *commits*. El primero de ellos lo hizo en local y el segundo lo realizó *on-line*. Todo está listo para hacer el *pull request*, o sea, para incorporar las últimas actualizaciones al repositorio original con el permiso de Alan.



The screenshot shows the GitHub interface for a pull request. At the top, the repository name is 'AlanBritoDelgado / las-aventuras-de-paco-el-pulpogato'. Below that, there are navigation tabs for 'Code', 'Issues', 'Pull requests', 'Wiki', 'Pulse', and 'Graphs'. The main heading is 'Comparing changes', followed by a sub-heading 'Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#).' Below this, there are dropdown menus for 'base fork: AlanBritoDelgado/las-aventu...', 'base: master', 'head fork: ElenaKozina/las-aventuras-d...', and 'compare: master'. A green checkmark indicates 'Able to merge. These branches can be automatically merged.' A prominent green button labeled 'Create pull request' is highlighted with a red box. Below the button, there is a summary: '2 commits', '1 file changed', '0 commit comments', and '1 contributor'. The commit history shows two commits by ElenaKozina on Sep 03, 2016: 'Capitulo nuevo' (commit f9d8b00) and 'Actualización de README.md' (commit 775dc8b). At the bottom, the diff view for README.md is shown, with line 1 containing '-# Las Aventuras de Paco El Pulpogato' and line 3 containing '## Introducción'.

Figura 7.13: Todo listo para crear el pull request.

Antes de hacer el *pull request* no vendría mal actualizar el repositorio local. Elena ha realizado los últimos cambios *on-line*, directamente desde su cuenta de GitHub; por tanto el repositorio de su máquina no está actualizado. Para actualizar en local, simplemente hay que ejecutar el comando `git pull`.

```
git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/ElenaKozina/las-aventuras-de-paco-el-pulpogato
   f9d8b00..775dc8b  master    -> origin/master
Updating f9d8b00..775dc8b
Fast-forward
 README.md | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```



## ¡Atención!

Cuando se realizan cambios en los archivos de un repositorio directamente desde la cuenta de GitHub (*on-line*), es necesario ejecutar el comando `git pull` para actualizar el repositorio local.

Elena hace click en el botón **Create pull request** (ver [figura 7.13](#)). A continuación aparece una página ([figura 7.14](#)) en la que pueden introducir comentarios sobre los cambios que ha realizado en el proyecto. Elena escribe el título “Nuevo capítulo” y la descripción “He escrito un nuevo capítulo de Las Aventuras de Paco El Pulpogato”.

## 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 fork: **AlanBritoDelgado/las-aventu...** base: **master** ... head fork: **ElenaKozina/las-aventuras-d...** compare: **master**

✓ **Able to merge.** These branches can be automatically merged.

**Nuevo capítulo**

Write Preview AA B i “ < > @

He escrito un nuevo capítulo de "Las Aventuras de Paco el Pulpogato"

Attach files by dragging & dropping or [selecting them](#).

Styling with Markdown is supported

**Create pull request**

2 commits 1 file changed 0 commit comments 1 contributor

Commits on Sep 03, 2016

|             |                            |         |
|-------------|----------------------------|---------|
| ElenaKozina | Capitulo nuevo             | f9d8b90 |
| ElenaKozina | Actualización de README.md | 775dc8b |

Figura 7.14: Comentario del *pull request*.

Elena hace click en el botón **Create pull request** y (por fin) se envía el *pull request* a Alan.

Vemos en la [figura 7.15](#) que después de hacer el *pull request*, se marca como *Open* (abierto) y permanecerá así mientras no sea autorizado por Alan.

This repository Search Pull requests Issues Gist

AlanBritoDelgado / **las-aventuras-de-paco-el-pulpogato** Watch 0 Star 0 Fork 1

<> Code Issues 0 Pull requests 1 Wiki Pulse Graphs

### Nuevo capítulo #1

**Open** ElenaKozina wants to merge 2 commits into AlanBritoDelgado:master from ElenaKozina:master

Conversation 0 Commits 2 Files changed 1 +4 -2

ElenaKozina commented just now

He escrito un nuevo capítulo de "Las Aventuras de Paco el Pulpogato"

Labels: None yet

Figura 7.15: El *pull request* está abierto.

Veamos ahora las cosas desde el punto de vista de Alan ([figura 7.16](#)).

Cuando Alan entra en su cuenta de GitHub ¡pega un salto de alegría!<sup>2</sup> Hay una pestaña que reza **Pull requests 1**. Eso significa que alguien ha realizado cambios en una copia de su historia y quiere que esos cambios sean aceptados por el dueño del repositorio, que es él mismo.

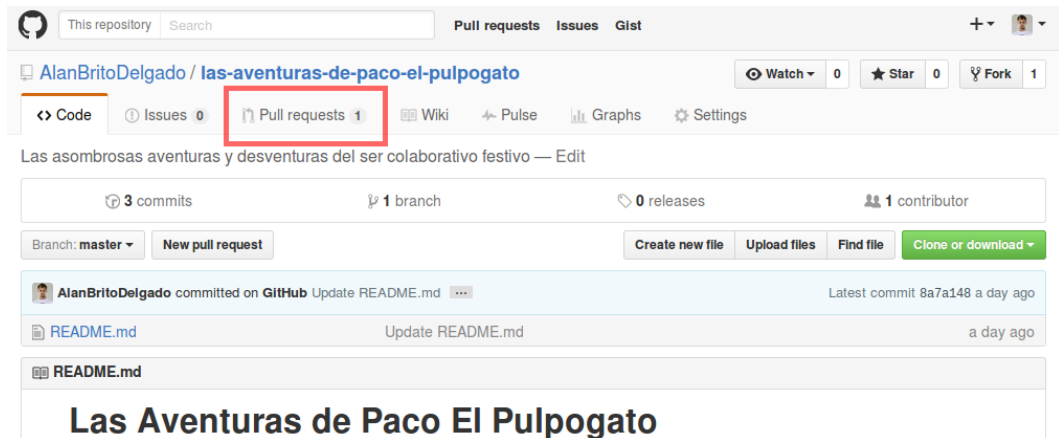


Figura 7.16: Hay un *pull request* pendiente de aceptar.

Alan hace click en la pestaña **Pull requests** y observa (figura 7.17) que la usuaria ElenaKozina ha hecho un *fork* con el título “Nuevo capítulo”.

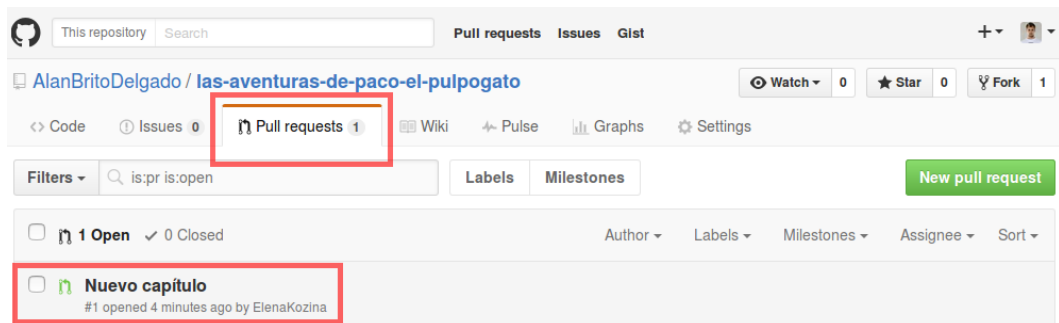


Figura 7.17: *Fork* “Nuevo capítulo”.

Alan hace click en **Nuevo capítulo**.

<sup>2</sup>Cuando te hagan el primer *pull request* ya verás como tú también saltas de alegría.

The screenshot shows a GitHub pull request interface. At the top, the repository name is 'AlanBritoDelgado / las-aventuras-de-paco-el-pulpogato'. The pull request title is 'Nuevo capítulo #1'. A green 'Open' button is visible. The pull request is from 'ElenaKozina:master' to 'AlanBritoDelgado:master', merging 2 commits. The 'Commits' tab is selected and highlighted with a red box, showing two commits: 'Capitulo nuevo' (f9d8b00) and 'Actualización de README.md' (775dc8b). A green checkmark indicates 'This branch has no conflicts with the base branch'. A 'Merge pull request' button is present. The right sidebar shows 'Labels' (None yet), 'Milestone' (No milestone), 'Assignees' (No one—assign yourself), '1 participant' (ElenaKozina), and 'Notifications' (Subscribe button). The bottom section shows a comment input area with a 'Comment' button.

Figura 7.18: Información detallada sobre el *fork* “Nuevo capítulo”.

Aparece una nueva página (figura 7.18) con toda la información sobre el *fork* realizado por Elena: comentarios sobre el *fork*, *commits* realizados, ficheros que han cambiado (en este caso únicamente el archivo `README.md`), etc.

Alan hace click en la pestaña **Commits**.

The screenshot shows the GitHub interface for the repository 'AlanBritoDelgado / las-aventuras-de-paco-el-pulpogato'. At the top, there are navigation links for 'Pull requests', 'Issues', and 'Gist'. Below that, the repository name and statistics (Watch: 0, Star: 0, Fork: 1) are displayed. The main heading is 'Nuevo capítulo #1'. A green 'Open' button indicates the pull request is ready for review. Below this, the 'Commits' tab is selected, showing two commits by ElenaKozina: 'Capitulo nuevo' (19 minutes ago) and 'Actualización de README.md' (11 minutes ago). The second commit is highlighted with a red box.

Figura 7.19: Repositorio *las-aventuras-de-paco-el-pulpogato* en GitHub.

En la pestaña **Commits** aparecen los dos commits que ha hecho Elena, para echar un vistazo, puede hacer click en el último, que es “Actualización de README.md”.

The screenshot shows the detailed view of the commit 'Actualización de README.md' by ElenaKozina. The commit hash is '775dc8b4da5ca7ff4c59a4ac4d58457cc6dd5da'. Below the commit information, the file 'README.md' is listed with a diff view. The diff shows changes to the file, with a red box highlighting the 'View' button in the top right corner of the diff area.

Figura 7.20: Fichero *README.md* modificado por Elena.

Alan puede comprobar cómo quedaría finalmente la historia con los cambios que ha realizado Elena pulsando en el botón **View** (saltaría al repositorio de Elena). Ya vimos anteriormente cómo quedaba el texto (figura 7.10).

Alan regresa otra vez a su repositorio y se vuelve a la pestaña **Pull requests**.

Una vez dentro del *fork* de Elena, Alan acepta los cambios haciendo click en el botón

## Merge pull request(figura 7.21).

The screenshot shows a GitHub pull request page for the repository 'AlanBritoDelgado / las-aventuras-de-paco-el-pulpogato'. The pull request title is 'Nuevo capítulo #1'. It shows that ElenaKozina wants to merge 2 commits into 'AlanBritoDelgado:master' from 'ElenaKozina:master'. The pull request has 2 commits and 1 file changed. A comment from ElenaKozina states: 'He escrito un nuevo capítulo de "Las Aventuras de Paco el Pulpogato"'. The commit history shows 'Capitulo nuevo' (f9d8b00) and 'Actualización de README.md' (775dc8b). A green box highlights the 'Merge pull request' button, with a dropdown arrow and the text 'or view command line instructions.' to its right. The interface also shows a 'Labels' section with 'None yet', a 'Milestone' section with 'No milestone', and an 'Assignees' section with 'No one—assign yourself'. There is 1 participant in the pull request.

Figura 7.21: Fusiona la rama (Merge pull request).

Al fusionar la rama, se pide confirmación. Alan hace click en **Confirm merge**. Es conveniente escribir un comentario cuando se fusiona una rama (figura 7.22).

**Nuevo capítulo #1** Edit

**Merged** AlanBritoDelgado merged 2 commits into AlanBritoDelgado:master from ElenaKozina:master 2 minutes ago

Conversation 0 Commits 2 Files changed 1 +4 -2

**ElenaKozina** commented 15 minutos ago

He escrito un nuevo capítulo de "Las Aventuras de Paco el Pulpogato"

**ElenaKozina** added some commits 28 minutos ago

- Capitulo nuevo f9d8b00
- Actualización de README.md 775dc8b

**AlanBritoDelgado** merged commit 73cd8ac into AlanBritoDelgado:master 2 minutes ago Revert

**Write** **Preview** AA B i “ <> Attach files by dragging & dropping or selecting them. Styling with Markdown is supported Comment

**Labels** None yet

**Milestone** No milestone

**Assignees** No one—assign yourself

**2 participants**

**Notifications** Unsubscribe

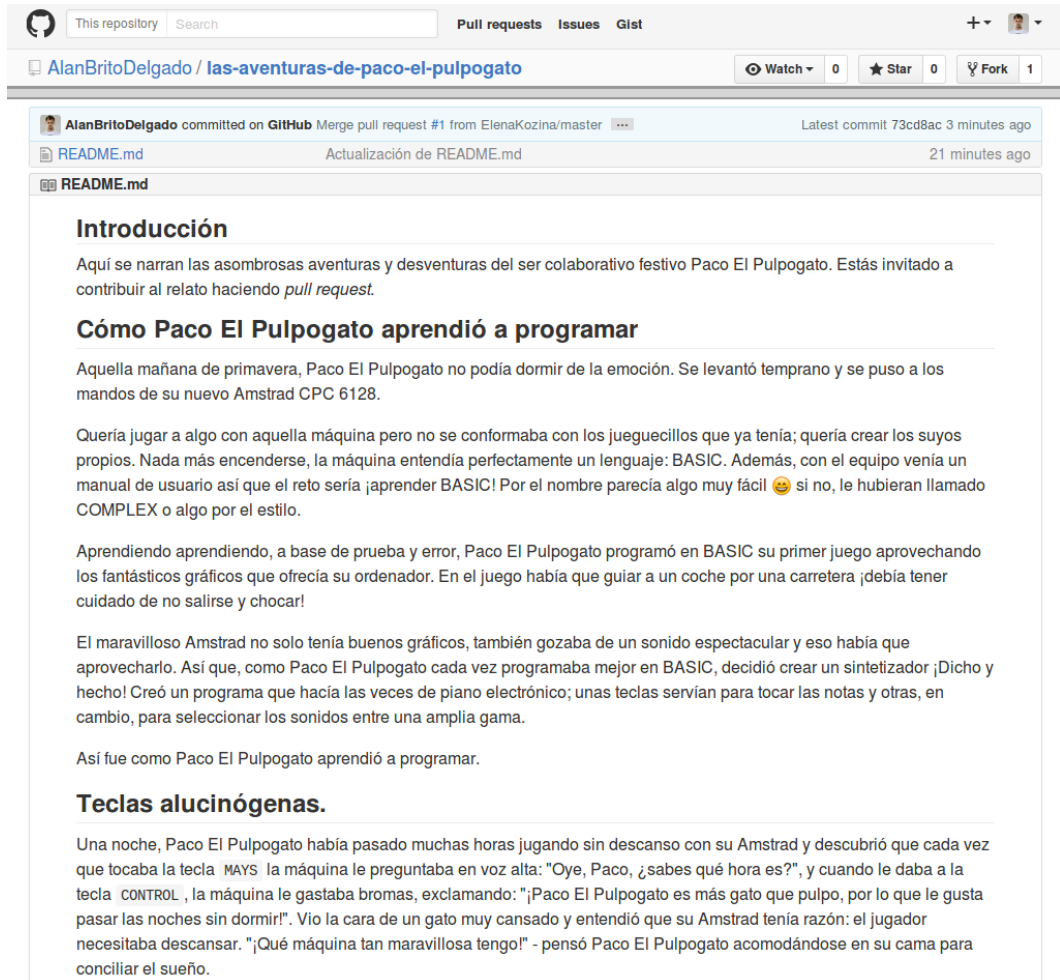
You're receiving notifications because you modified the open/close state.

Lock conversation

Figura 7.22: Comentario sobre los cambios.

Observa que ahora aparece la etiqueta **Merged** (fusionado) en lugar de **Open** (abierto). El **pull request** ha concluido satisfactoriamente.

En la [figura 7.23](#) se muestra la versión final de la historia.



This repository Search Pull requests Issues Gist

Alan Brito Delgado / las-aventuras-de-paco-el-pulpogato Watch 0 Star 0 Fork 1

Alan Brito Delgado committed on GitHub Merge pull request #1 from ElenaKozina/master Latest commit 73cd8ac 3 minutes ago

README.md Actualización de README.md 21 minutes ago

README.md

## Introducción

Aquí se narran las asombrosas aventuras y desventuras del ser colaborativo festivo Paco El Pulpogato. Estás invitado a contribuir al relato haciendo *pull request*.

## Cómo Paco El Pulpogato aprendió a programar

Aquella mañana de primavera, Paco El Pulpogato no podía dormir de la emoción. Se levantó temprano y se puso a los mandos de su nuevo Amstrad CPC 6128.

Quería jugar a algo con aquella máquina pero no se conformaba con los jueguecillos que ya tenía; quería crear los suyos propios. Nada más encenderse, la máquina entendía perfectamente un lenguaje: BASIC. Además, con el equipo venía un manual de usuario así que el reto sería ¡aprender BASIC! Por el nombre parecía algo muy fácil 😊 si no, le hubieran llamado COMPLEX o algo por el estilo.

Aprendiendo aprendiendo, a base de prueba y error, Paco El Pulpogato programó en BASIC su primer juego aprovechando los fantásticos gráficos que ofrecía su ordenador. En el juego había que guiar a un coche por una carretera ¡debía tener cuidado de no salirse y chocar!

El maravilloso Amstrad no solo tenía buenos gráficos, también gozaba de un sonido espectacular y eso había que aprovecharlo. Así que, como Paco El Pulpogato cada vez programaba mejor en BASIC, decidió crear un sintetizador ¡Dicho y hecho! Creó un programa que hacía las veces de piano electrónico; unas teclas servían para tocar las notas y otras, en cambio, para seleccionar los sonidos entre una amplia gama.

Así fue como Paco El Pulpogato aprendió a programar.

## Teclas alucinógenas.

Una noche, Paco El Pulpogato había pasado muchas horas jugando sin descanso con su Amstrad y descubrió que cada vez que tocaba la tecla `MAYS` la máquina le preguntaba en voz alta: "Oye, Paco, ¿sabes qué hora es?", y cuando le daba a la tecla `CONTROL`, la máquina le gastaba bromas, exclamando: "¡Paco El Pulpogato es más gato que pulpo, por lo que le gusta pasar las noches sin dormir!". Vio la cara de un gato muy cansado y entendió que su Amstrad tenía razón: el jugador necesitaba descansar. "¡Qué máquina tan maravillosa tengo!" - pensó Paco El Pulpogato acomodándose en su cama para conciliar el sueño.

Figura 7.23: Texto final (la línea gris indica por dónde se ha cortado la imagen).

## 7.1 Comandos utilizados en este capítulo



### Clonación de repositorios

```
git clone https://github.com/AlanBritoDelgado/las-aventuras-de-paco-el-pulpo\
gato.git
git clone https://github.com/ElenaKozina/las-aventuras-de-paco-el-pulpogato.\
git
```



### Cambio de directorio

```
cd las-aventuras-de-paco-el-pulpogato/
```



### Edición de un fichero

```
gedit README.md
```



### Adición de ficheros y directorios al índice de elementos a tener en cuenta en el próximo commit

```
git add . --all
```



### Commit (confirmación de cambios realizados)

```
git commit -m "Añadido el fichero HolaMundo.java"
```



### **Actualización en GitHub de los cambios realizados en local**

```
git push
```



### **Actualización de un repositorio en local**

```
git pull
```

# Invitación

Si te gusta escribir y tienes imaginación, te invito a que participes en la elaboración de la historia “Las Aventuras de Paco El Pulpogato”.

El repositorio en cuestión está en <https://github.com/LuisJoseSanchez/las-aventuras-de-paco-el-pulpogato>

En el [Capítulo 7](#) se explica en detalle el desarrollo colaborativo con Git y GitHub.

¡Espero tus *pull requests*!

# Referencias

## Git

- [Git Pro](#)
- [Git Inmersion](#)

## GitHub

- [GitHub Help](#)

## Fichero .gitignore

- [Ejemplo de .gitignore](#)

## Markdown

- [Mastering Markdown](#)

## Emojis

- [Emoji Cheat Sheet](#)

## Java

- [Aprende Java con Ejercicios](#)
- [Ejemplos y soluciones a los ejercicios de “Aprende Java con Ejercicios”](#)