



Examen Semana 4

Luis Fernando Pedraza Estañol

Instructor: Miguel Rugerio

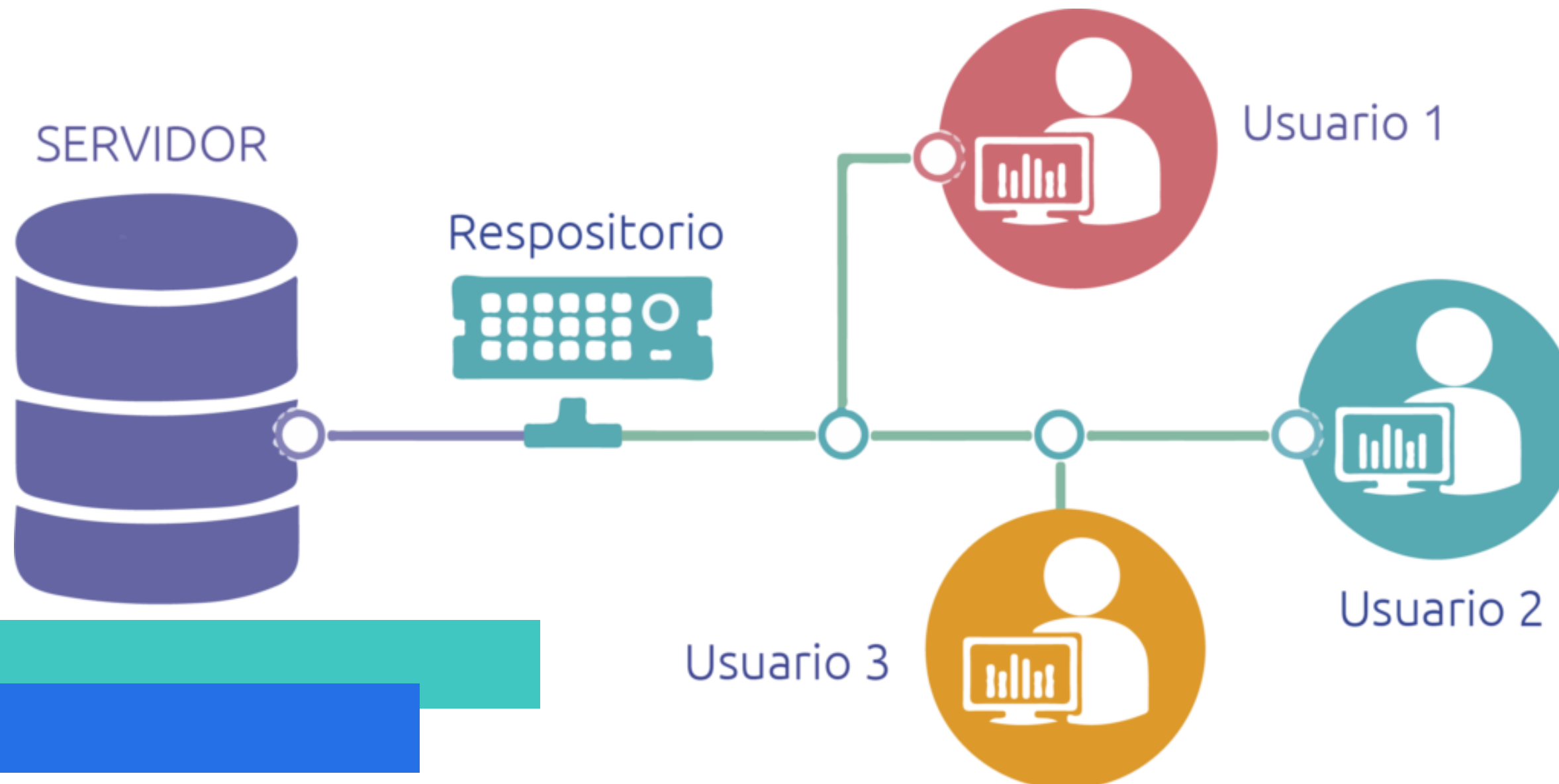


Métodos de GIT

The background is a solid blue color. In the top-left corner, there is a teal-colored triangle. In the bottom-right corner, there is another teal-colored triangle. Two white decorative lines are present: one in the top-right area and one in the bottom-left area. Each line consists of a horizontal segment and a diagonal segment, with a small white dot at the end of the horizontal segment.

GIT

- ▶ Es un sistema de control de versiones de código abierto y actualmente es el sistema de control de versiones más extendido.
- ▶ Cuenta con una arquitectura distribuida, por lo que cada usuario contiene una copia del repositorio con el historial de cambios completo del proyecto, aumentando su rendimiento.



The background is a solid teal color. In the top-left corner, there is a blue triangle. In the bottom-right corner, there is a blue triangle. A white line starts from the left edge, goes right, then diagonally up-right, then right again, ending with a white dot. Another white line starts from a white dot, goes left, then diagonally down-left, then left again, ending with a white dot.

Pull Request

Pull Request

Es una petición para integrar propuestas o cambios de código a un proyecto. Permiten llevar las tareas más ordenadas en la etapa de desarrollo, y crear propuestas o cambios que puedan ser integrados posteriormente en un proyecto.

Pasos a seguir:

1. Crear un repositorio Git.
2. Una vez clonado el repositorio de forma local, se procede a crear una nueva rama de trabajo, denominada «index-page».

```
$ git checkout -b index-page
```

Dentro de la rama, se crea un nuevo archivo llamado index.html con el código mostrado en la imagen.

```
1.  <!DOCTYPE html>
2.  <html lang="en">
3.    <head>
4.      <meta charset="utf-8">
5.      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6.      <meta name="viewport" content="width=device-width, initial-sca
7.      <title>Bootstrap 101 Template</title>
8.      <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/b
9.    </head>
10.   <body>
11.     <h1>Hello, world!</h1>
12.
13.     <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11
14.     <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/j
15.   </body>
16. </html>
```

Pull Request

4. La nueva rama se envía al repositorio para crear el pull request: para ello deben agregarse los cambios, hacer commit, y por último, push de esta rama.

```
1. //para agregar los nuevos cambios
2. $ git add --all
3.
4. //hacer commit de estos cambios
5. $ git commit -m "página de inicio de la aplicación"
6.
7. //hacer push de esta rama al repositorio remoto
8. $ git push -u origin index-page
```


Hasta ahora sólo se ha iniciado la petición de integración: en lugar de realizar los cambios sobre la rama «master», se trabaja en ramas separadas.


Pull Request

5. La nueva rama «index-page» aparece en la lista de «branches», y puede visualizarse un nuevo enlace a «Compare & pull request».

Open a pull request

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

 base: **master** — compare: **index-page** ✓ Able to merge. These branches can be automatically merged.



Index page

Write

Preview

Markdown supported

Edit in fullscreen

Se ha creado la página index para el proyecto

Attach images by dragging & dropping, [selecting them](#), or pasting from the clipboard.

Create pull request

Labels

None yet

Milestone

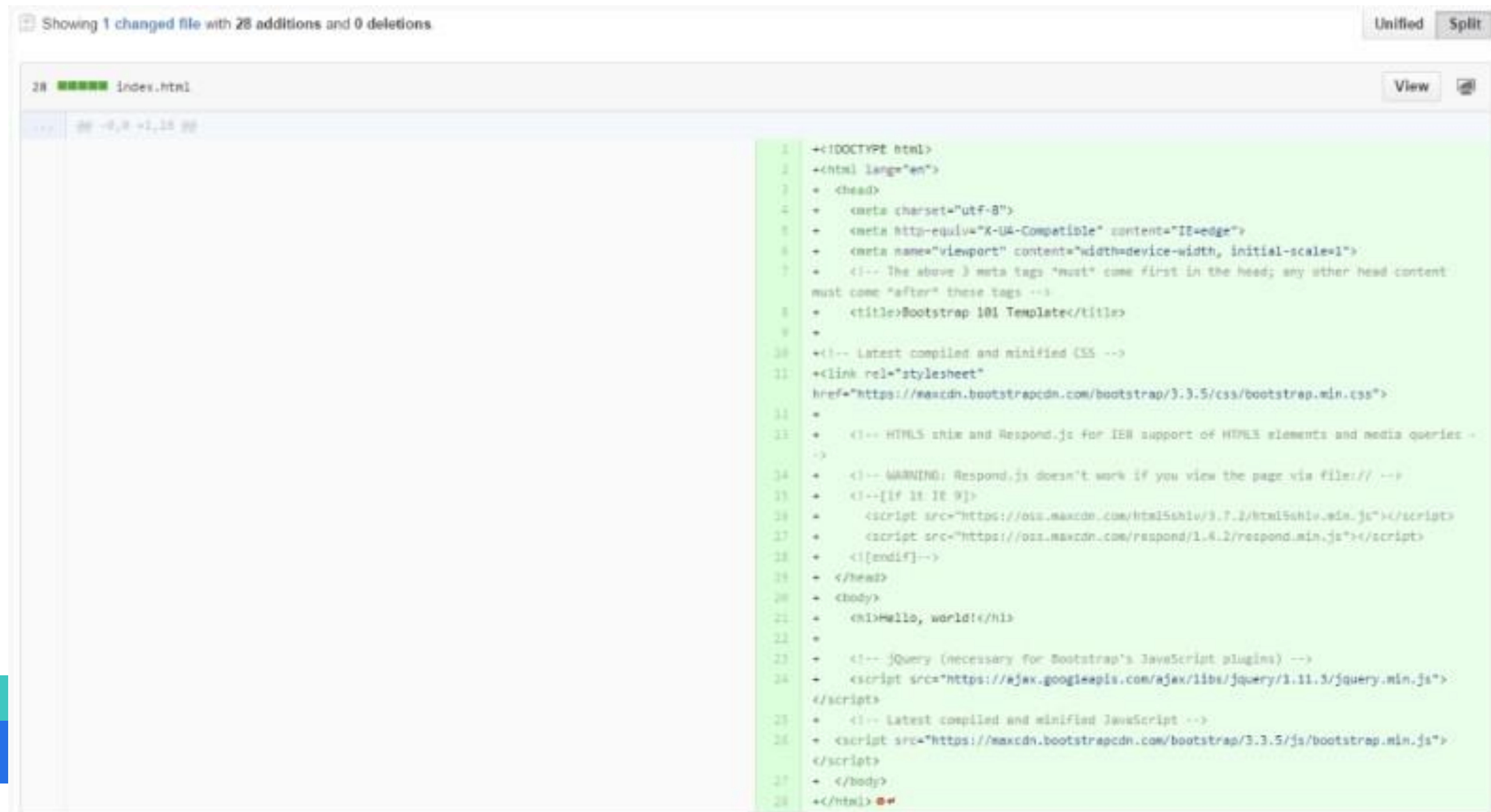
No milestone

Assignee

No one—assign yourself

Pull Request

6. Puede escribirse una descripción en detalle de lo que se ha incluido en este cambio, algunas instrucciones para la puesta en marcha, y puede verse una comparación del código de la rama máster con la rama «index-page». De lado izquierdo aparece la versión del código almacenada en la rama máster, y de lado derecho los cambios actuales.



The screenshot shows a Pull Request interface with a diff view for the file `index.html`. The top status bar indicates "Showing 1 changed file with 28 additions and 0 deletions". The interface is split into two panes: the left pane shows the original code from the master branch, and the right pane shows the proposed changes. The code is an HTML template for Bootstrap 101, including meta tags, CSS links, and JavaScript links. The diff highlights the changes with green lines for additions and red lines for deletions. The right pane shows the full code with line numbers from 1 to 28.

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1">
7     <!-- The above 3 meta tags *must* come first in the head; any other head content
8          must come "after" these tags -->
9     <title>Bootstrap 101 Template</title>
10  *
11  <!-- Latest compiled and minified CSS -->
12  <link rel="stylesheet"
13        href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
14  *
15  <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media queries -->
16  <!-- WARNING: Respond.js doesn't work if you view the page via file:// -->
17  <!--[if lt IE 9]>
18    <script src="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.min.js"></script>
19    <script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>
20  <![endif]-->
21  </head>
22  <body>
23    <h1>Hello, world!</h1>
24  *
25  <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
26  <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js">
27  </script>
28  <!-- Latest compiled and minified JavaScript -->
29  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js">
30  </script>
31  </body>
32 </html>
```


Pull Request


Luego de incluir la información necesaria y hacer click en el botón «Create pull request», se visualizará el siguiente mensaje:

[Open](#) jeffer8a wants to merge 2 commits into `master` from `index-page`


Conversation 0

Commits 2


Files changed 1




jeffer8a commented 15 seconds ago


Owner 

Se ha creado la página index para el proyecto





jeffer8a added some commits 11 minutes ago


 página de inicio de la aplicación 66f26c5

 Merge branch 'master' of https://github.com/jeffer8a/StydeNet-pull-re... b33637d


Add more commits by pushing to the `index-page` branch on jeffer8a/StydeNet-pull-request.





 **This branch is up-to-date with the base branch**
Merging can be performed automatically.

 Merge pull request

You can also open this in [GitHub for Windows](#) or view [command line instructions](#).



Write Preview

 Markdown supported  Edit in fullscreen

Leave a comment

Attach images by dragging & dropping, [selecting them](#), or pasting from the clipboard.

Close pull request

Comment

Pull Request

7. Luego de revisar el código, se prueba de forma local y se comprueba que todo funciona correctamente, para incluir los cambios en la rama master del proyecto haciendo click en «Merge pull request»

Index page #1


Merged

jeffer8a merged 2 commits into master from index-page 15 seconds ago

Conversation 0

Commits 2

Files changed 1




jeffer8a commented 5 minutes ago


Owner


✎


Se ha creado la página index para el proyecto



jeffer8a added some commits 14 minutes ago


 página de inicio de la aplicación 66f26c5

 Merge branch 'master' of https://github.com/jeffer8a/StydeNet-pull-re... b33637d



jeffer8a merged commit 96e2423 into master 14 seconds ago


Revert



Pull request successfully merged and closed

You're all set—the index-page branch can be safely deleted.

Delete branch



Write Preview

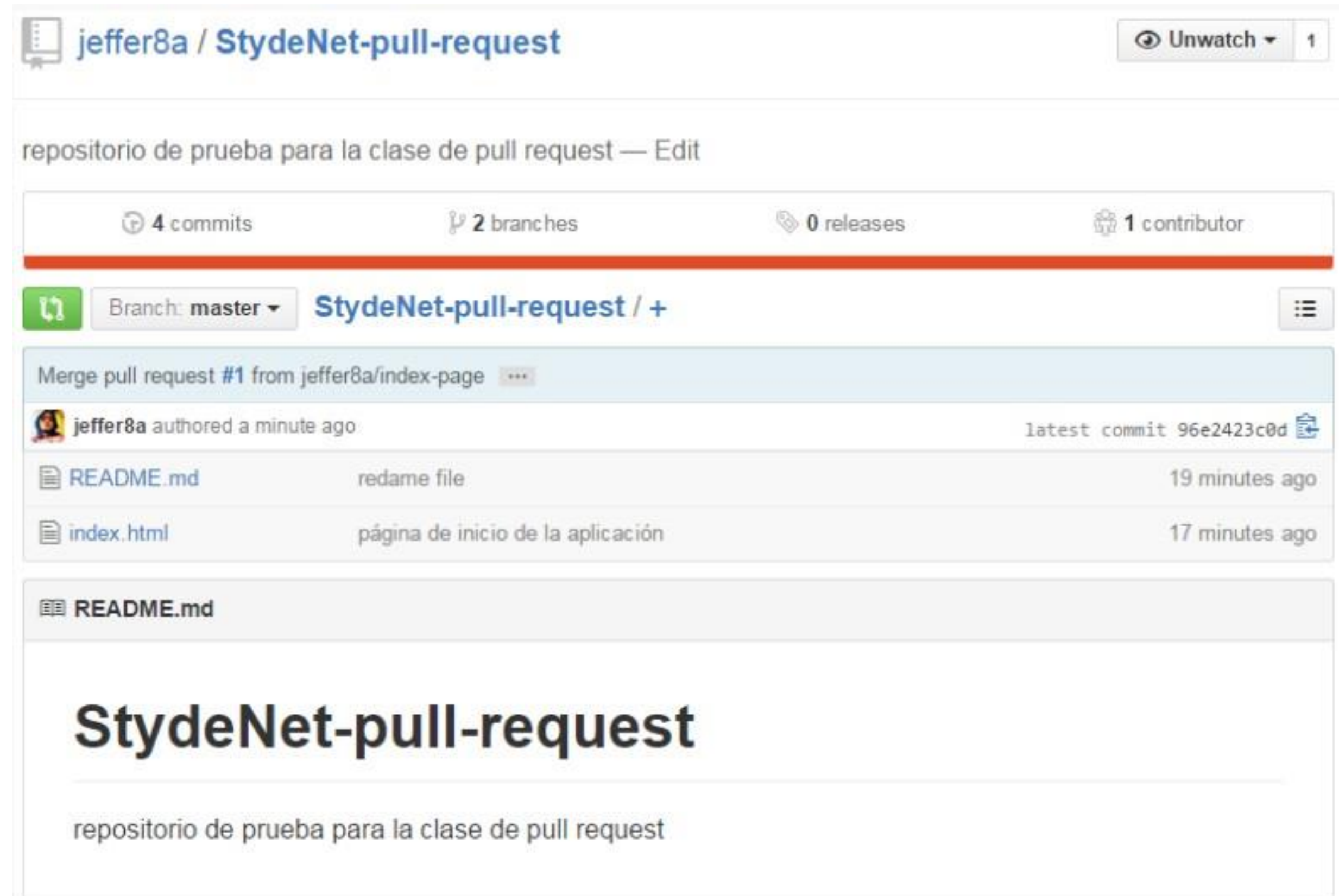
Markdown supported Edit in fullscreen

Leave a comment

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

Pull Request

Ahora el archivo index.html se encuentra disponible en la rama master del proyecto, y cada persona que clone el repositorio contará con los nuevos cambios



The screenshot shows a GitHub pull request interface. At the top, the repository name 'jeffer8a / StydeNet-pull-request' is displayed, along with an 'Unwatch' button and a notification count of '1'. Below this, the repository is described as 'repositorio de prueba para la clase de pull request' with an 'Edit' link. A summary bar indicates '4 commits', '2 branches', '0 releases', and '1 contributor'. The main section shows the pull request details: 'Merge pull request #1 from jeffer8a/index-page'. It lists the author 'jeffer8a' and the time 'a minute ago', along with the 'latest commit 96e2423c0d'. A table of files shows 'README.md' (redame file) and 'index.html' (página de inicio de la aplicación) with their respective commit times. The bottom section displays the 'README.md' content, including the repository title 'StydeNet-pull-request' and the description 'repositorio de prueba para la clase de pull request'.

jeffer8a / StydeNet-pull-request

repositorio de prueba para la clase de pull request — Edit

4 commits 2 branches 0 releases 1 contributor

Branch: master StydeNet-pull-request / +

Merge pull request #1 from jeffer8a/index-page

jeffer8a authored a minute ago latest commit 96e2423c0d

README.md	redame file	19 minutes ago
index.html	página de inicio de la aplicación	17 minutes ago

README.md

StydeNet-pull-request

repositorio de prueba para la clase de pull request



Fork

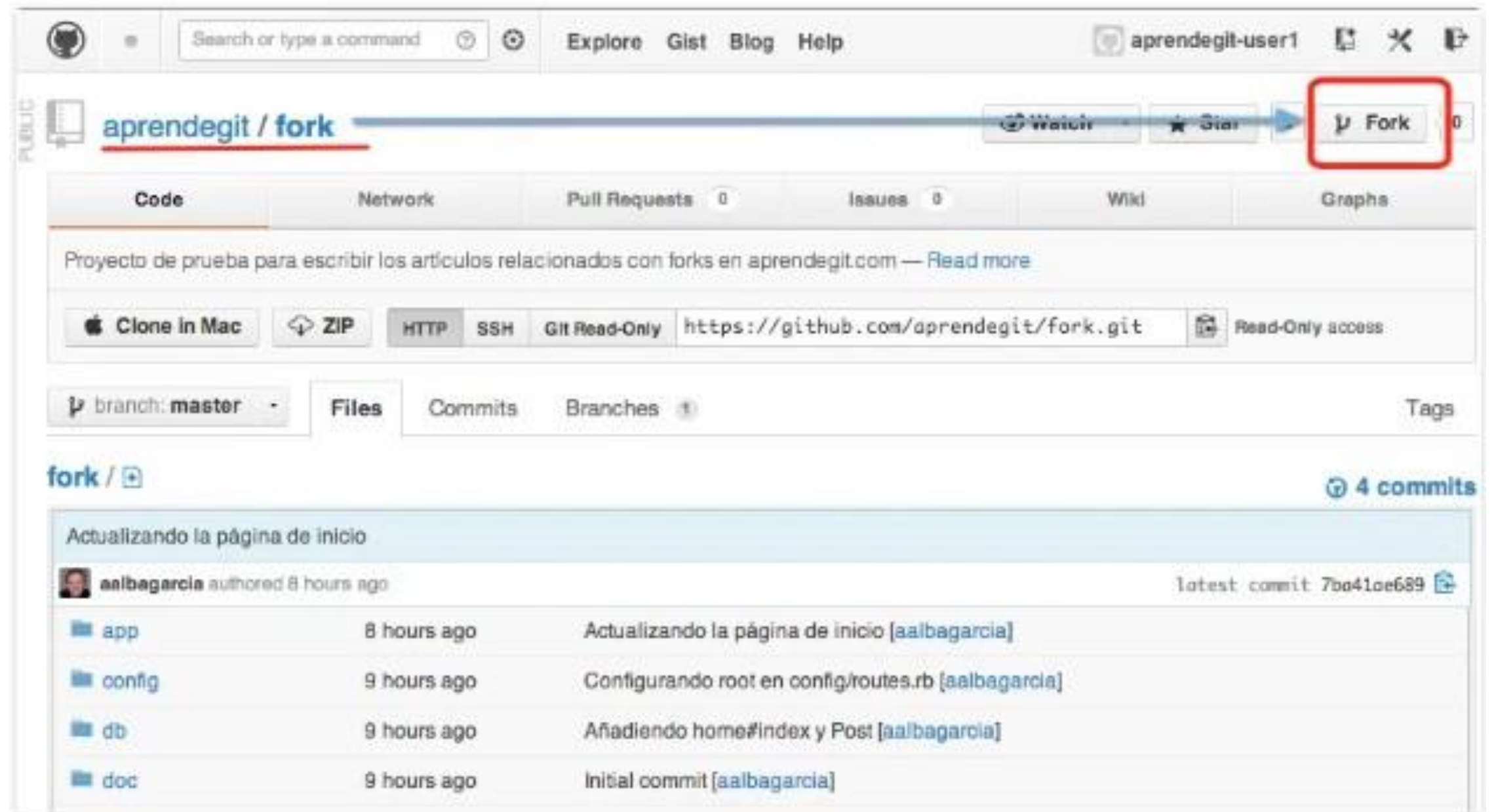
Fork

Es una operación usual en el sistema de Git que se encarga de la creación de una copia de un repositorio en la cuenta de usuario. El repositorio copiado será igual al repositorio desde el que se realiza el fork en Git.

Cuando se cree la copia, cada repositorio se ubicará en espacios diferentes y tendrán la posibilidad de evolucionar de forma diferente.

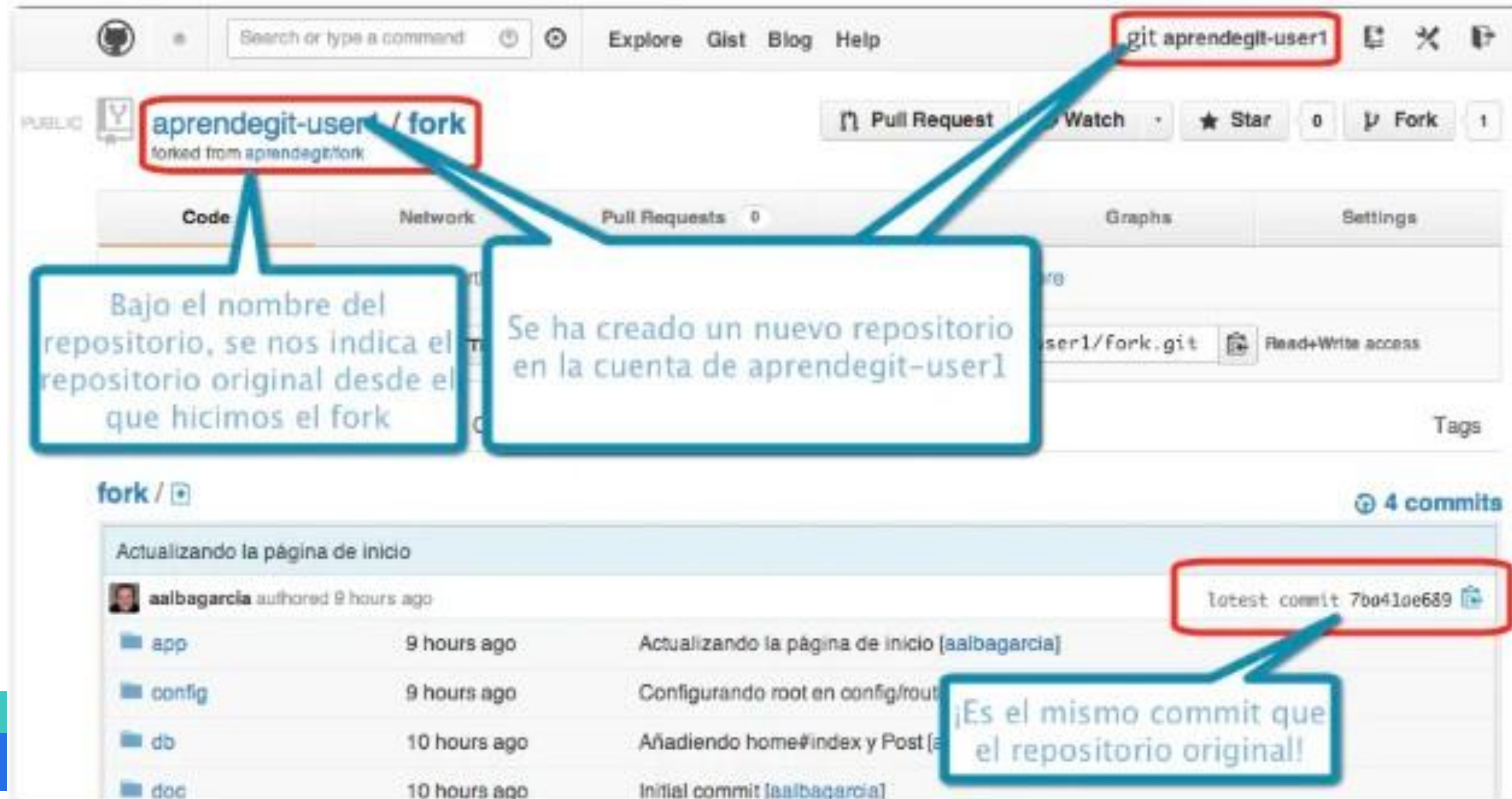
Pasos a seguir:

En la página del repositorio, hacer click sobre el botón Fork, Github va a crear un nuevo repositorio en copia.



Fork

Cuando el proceso de forking termina, se muestra la siguiente pantalla. Se obtendrá como resultado 2 repositorios iguales con URLs diferentes.



Rebase

The background is a solid teal color. In the top-left corner, there is a blue triangle with a light blue diagonal stripe. In the bottom-right corner, there is a blue triangle with a light blue diagonal stripe. Two white geometric lines are present: one starts from the bottom-left, goes right, then up-right, then right, ending with a dot; the other starts from the top-right, goes left, then up-left, then left, ending with a dot.

Rebase

Comando enfocado en integrar modificaciones de una rama a otra, a través de reorganizar o cambiar la base de una rama de commit a otra.

El comando rebase permite utilizarse con otros subcomandos, como pick, reword, edit, squash, fixup, exec.

Pasos a seguir:

1. Una rama que se separó de la rama master se puede mover así al utilizar rebase:

```
      /o-----o---o---o-----o----- branch
--o-o--A--o---o---o---o---o---o-o-o--- master
```

```
                        /o-----o---o---o-----o----- branch
--o-o--A--o---o---o---o---o---o-o-o master
```

2. Se deben tener todos los commits deseados en el rebase en la rama master. Revisar la rama en la que se desea hacer rebase y escribir git rebase master

```
                        /---o-o branch
      /---o-o-o-o---o---o----- feature
----o-o-o-A---o---o---o-o-o-o--- master
```

Rebase

3. Después de `git rebase master branch` o `git rebase master`, si se encuentra en branch, se obtiene:

```
      /---o-o-o-o---o-o----- feature
-----o-o-o-A-----o---o-o-o-o---o---o- master
                        \---o-o branch
```

Git Rebase interactivo

- Introduce `git rebase -i HEAD~5` con el último número que sea cualquier número de commits del más reciente hacia atrás que se quiera revisar.
- En vim, se presiona `esc`, luego `i` para empezar a editar la prueba.
- A la izquierda se puede sobrescribir pick con uno de los comandos de abajo. Si se quiere aplastar (squash) un commit a uno anterior y descartar el mensaje de commit, se introduce `f` en lugar de pick en el commit.
- Guardar y salir del editor de texto.
- Cuando se detiene a rebase, se hacen los ajustes necesarios, y luego se usa `git rebase --continue` hasta que el rebase sea exitoso.

Rebase

- Si el rebase es exitoso, entonces se necesita forzar el push de los cambios con `git push -f` para agregar la versión con rebase a un repositorio remoto.
- Si hay un "merge conflict" (conflicto al querer fusionar las ramas), hay varias maneras de arreglarlo. Una forma es abrir los archivos en un editor de texto y eliminar las partes del código que no se quieran. Luego se usa `git add <file name>` seguido de `git rebase --continue`. Puede saltarse el commit de conflicto ejecutando `git rebase --skip`, para el rebase ejecutando `git rebase --abort` en la consola.

```
pick 452b159 <message for this commit>
pick 7fd4192 <message for this commit>
pick c1af3e5 <message for this commit>
pick 5f5e8d3 <message for this commit>
pick 5186a9f <message for this commit>

# Rebase 0617e63..5186a9f onto 0617e63 (30 commands)
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but stop to edit the commit message.
# e, edit = use commit, but stop to amend or add commit.
# s, squash = use commit, meld into previous commit and stop to edit the commit message.
# f, fixup = like "squash", but discard this commit's log message thus doesn't stop.
# x, exec = run command (the rest of the line) using shell
# d, drop = remove commit
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

The word "Stash" is centered in a white, bold, sans-serif font. The background is a solid teal color. In the top-left and bottom-right corners, there are diagonal stripes of blue and light blue. Two white lines with circular endpoints are positioned diagonally: one in the top-right area and one in the bottom-left area, mirroring each other.

Stash

Stash

Almacena temporalmente los cambios que se hayan efectuado en el código en el que se trabaja para retomarlo y volver a aplicar los cambios más tarde

Pasos a seguir:

Se toman los cambios sin confirmar, se guardan aparte y se deshace en el código en que se trabaja.

Al hacer git stash pop, se eliminan los cambios y se vuelve a aplicar el código en proceso.

```
$ git status
On branch main
Changes to be committed:

  new file:   style.css

Changes not staged for commit:

  modified:   index.html

$ git stash
Saved working directory and index state WIP on main: 5002d47 our new homepage
HEAD is now at 5002d47 our new homepage

$ git status
On branch main
nothing to commit, working tree clean
```

```
$ git status
On branch main
nothing to commit, working tree clean
$ git stash pop
On branch main
Changes to be committed:

  new file:   style.css

Changes not staged for commit:

  modified:   index.html

Dropped refs/stash@{0} (32b3aa1d185dfe6d57b3c3cc3b32cbf)
```




Clean

Clean

Comando útil para eliminar los archivos sin seguimiento en un directorio de trabajo del repositorio.

Pasos a seguir:

Se crea un nuevo repositorio en el directorio git clean test, se procede a crear un tracked file que se añade al índice. Además se crea un untracked file y un untracked dir.

```
$ mkdir git_clean_test
$ cd git_clean_test/
$ git init .
Initialized empty Git repository in /Users/kev/code/git_clean_test/
$ echo "tracked" > ./tracked_file
$ git add ./tracked_file
$ echo "untracked" > ./untracked_file
$ mkdir ./untracked_dir && touch ./untracked_dir/file
$ git status
On branch master

Initial commit

Changes to be committed: (use "git rm --cached <file>..." to unstage)
    new file:   tracked_file

Untracked files: (use "git add <file>..." to include in commit)
    ./untracked_dir/
```



Cherry-pick

Cherry-pick

Comando que se encarga de elegir uno o más commit o confirmaciones de cambios de una rama específica para luego aplicarla en otra rama.

Pasos a seguir:

1. Se tiene un repositorio con el siguiente estado de rama:

```
a - b - c - d Main
      \
      e - f - g Feature
```

3. Se ejecuta cherry-pick con el siguiente comando:

```
git cherry-pick f
```

2. Commit Sha es una referencia de confirmación. Se debe asegurar trabajar con la rama principal

```
git cherry-pick commitSha
```

4. Una vez ejecutado, el historial de Git se verá así:

```
a - b - c - d - f Main
      \
      e - f - g Feature
```



Branch

Branch

Comando que permite crear, enumerar y eliminar ramas, así como cambiar su nombre. No permite cambiar entre ramas, por lo que está ligado a checkout y merge.

Opciones:

1.Crea una nueva rama. Este comando no extrae la nueva rama.

```
git branch <branch>
```

3.Enumera las ramas remotas.

```
git branch -a
```

2.Elimina la rama especificada. Esta es una operación segura, ya que Git evita que se elimine la rama si tiene cambios que aún no se han fusionado.

```
git branch -D <branch>
```

4.Fuerza la eliminación de la rama especificada, incluso si se tiene cambios sin fusionar. Este comando se puede usar si se quiere eliminar de forma permanente todas las confirmaciones asociadas con una línea concreta de desarrollo.

```
git branch -m <branch>
```

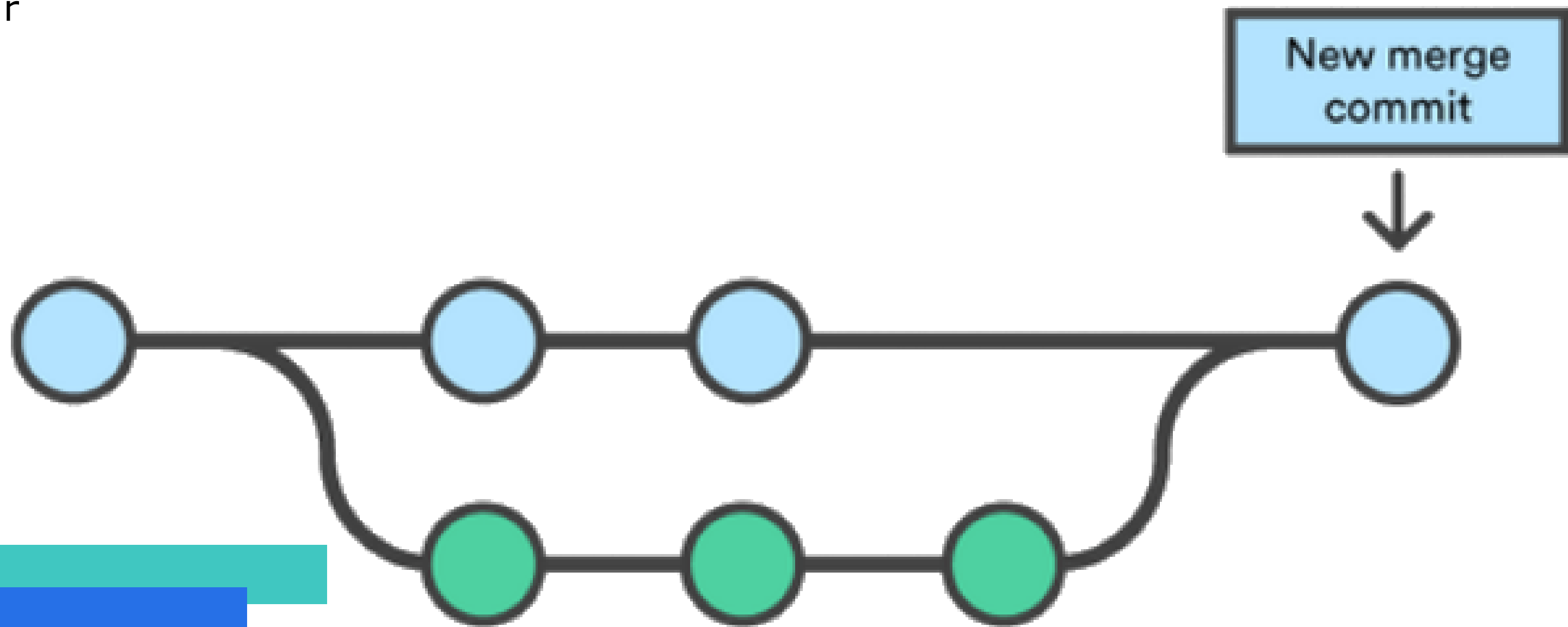



Merge

Merge

Se utiliza con el objetivo de fusionar una o más ramas en el interior de la rama que se encuentre activa.

El comando git merge fue introducido por primera vez en Procedimientos Básicos de Ramificación. A pesar de que se utiliza en diversos lugares en el libro, hay muy pocas variaciones del comando merge — en general, sólo git merge <branch> con el nombre de la rama individual que se desea combinar





Conflicts

Conflicts

Normalmente los conflictos surgen cuando dos personas han cambiado las mismas líneas de un archivo o si un desarrollador ha eliminado un archivo mientras otro lo estaba modificando. En estos casos, Git no puede determinar automáticamente qué es correcto. Los conflictos solo afectan al desarrollador que realiza la fusión, el resto del equipo no se entera del conflicto. Git marcará el archivo como que tiene un conflicto y detendrá el proceso de fusión.

Tipos de conflicto:

Git no inicia la fusión

Una fusión no se iniciará si Git detecta que hay cambios en el directorio de trabajo o el entorno de ensayo del proyecto actual. Git no inicia la fusión porque las confirmaciones que se están fusionando podrían sobrescribir estos cambios pendientes. Cuando esto sucede, no se debe a conflictos con otros desarrolladores, sino con cambios locales pendientes. El estado local tendrá que estabilizarse mediante `git stash`, `git checkout`, `git commit` o `git reset`. Un fallo de fusión durante el inicio provocará que aparezca el siguiente mensaje de error:

```
error: Entry '<fileName>' not uptodate. Cannot merge. (C
```

Conflicts

Git falla durante la fusión

Un fallo DURANTE una fusión indica un conflicto entre la rama local actual y la rama que se está fusionando. Esto indica un conflicto con el código de otro desarrollador. Git hará lo posible para fusionar los archivos, pero te dejará cosas para que las resuelvas manualmente en los archivos con conflictos. Un fallo a la mitad de la fusión provocará que aparezca el siguiente mensaje de error:

```
error: Entry '<fileName>' would be overwritten by merge.
```



BUENAS PRÁCTICAS DE REST API



API PARA PRODUCIR DATOS JSON

Al diseñar una API REST que usarán las aplicaciones móviles creadas con Dropsource, están diseñadas para conectarse a una API REST que devuelve datos JSON.

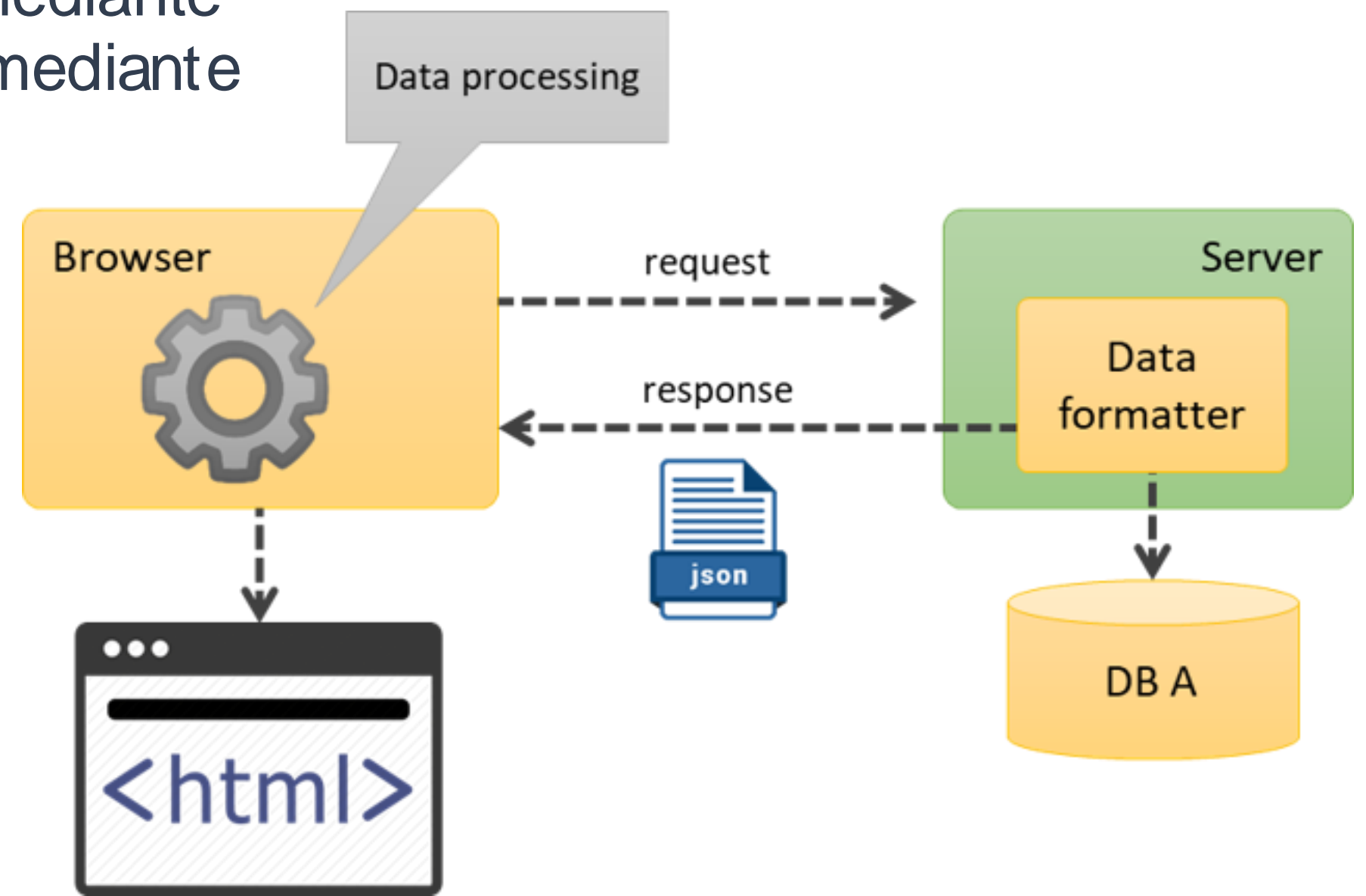
```
<script>
    Persona = {
        "ID": "1",
        "Nombre": "Jhon",
        "Apellido": "Doe",
        "Edad": 64,
        "Tel" : null,
        "check": true,
        "SO": ["Windows 10", "Ubuntu", "Android"]
    };
    console.log(Persona)
</script>
```

DISEÑAR API CON PRINCIPIOS CLAVE DE REST

Los principios de transferencia de estado representacional (REST) son diseños arquitectónicos clave que los desarrolladores deben seguir al diseñar API REST.

Uno de los principios clave REST es separar una API en recursos lógicos.

Estos recursos lógicos deben representarse mediante sustantivos y los recursos deben manipularse mediante verbos HTTP comunes.






SUSTANTIVOS

La dirección URL de cada punto de enlace debe usar sustantivos para indicar claramente qué recurso se representa.

No se deben mezclar sustantivos singulares y plurales en su API. De esta manera, cuando los usuarios estén desarrollando contra la API, no tendrán que adivinar qué camino tomar.



VERBOS HTTP

Para las API REST, los verbos http brindan una contrapartida de acción para los recursos basados en sustantivos. Los más comunes son:

- POST: guarda una nueva representación de un recurso.
- GET: solicita el estado de un recurso específico. Get no se utiliza para guardar o actualizar datos.
- PUT: actualiza el estado de un recurso existente.
- PATCH: actualiza parcialmente el estado de un recurso existente. Esto es útil cuando se trata de datos complejos.
- ELIMINAR: elimina un recurso específico.

POST /persons Create a new user in the database. (Note: JSON representation of a new user's data should be contained in the body.)

GET /persons/ Return a list of all users in the database.

GET /persons/{username} Return a user by supplying a unique user name.

PUT /persons/{username} Update an existing user in the database. (Note: the JSON representation of the user should be supplied in the body along with the user's user id).

DELETE /persons/{username} Delete an existing user whos unique username is supplied.

DOCUMENTE SU API UTILIZANDO LA ESPECIFICACIÓN OPENAPI

Para que los usuarios de la API obtengan visibilidad de su API REST y la hagan lo más fácil de usar posible, se documenta la API utilizando la especificación OpenAPI.



OpenAPI

Sin una documentación concisa, es posible que los usuarios de la API no sepan necesariamente lo que representa cada campo.



DEJAR QUE EL SERVIDOR HAGA EL TRABAJO

Al desarrollar una API REST, se espera que una amplia variedad de clientes accedan a ella.

Todo el procesamiento intensivo, el procesamiento numérico y las reglas comerciales deben realizarse en un servidor remoto potente.

La descarga de este procesamiento es crítica debido a las restricciones de hardware impuestas por los dispositivos móviles.





PAGINACIÓN, FILTRADO Y ORDENACIÓN MEDIANTE PARÁMETROS DE CONSULTA

Esto le da flexibilidad a la API de REST mientras mantiene el recuento de puntos de conexión de API pequeño y manejable.

Especialmente importante para que los clientes móviles no tengan que procesar grandes cantidades de datos.



CONTROL DE VERSIONES DE URL

El control de versiones de URL se implementa agregando el número de versión directamente en la URL.

El siguiente punto de enlace para obtener una lista de todos los usuarios de un sistema estaría disponible:

GET `/v1/persons` API Version 1 - Return a list of all users in the database.


algunos campos obsoletos deben eliminarse de la carga útil de retorno, por lo que se lanza la versión 2. El punto de conexión para esta versión sería el siguiente:

GET `/v2/persons` API Version 2 - Return a list of all users in the database.

AUTENTICACIÓN Y AUTORIZACIÓN

También se considera una buena práctica incluir al menos un nivel básico de autenticación para que al menos entienda quién está usando su API y pueda limitar las solicitudes en caso de un ataque de seguridad como DDOS.

- **Autenticación:** indicar quién es usted y proporcionar credenciales al servidor para verificar que usted es quien dice ser.
- **Autorización:** se produce después de la autenticación. Implica la comprobación de que se permite la conexión a un recurso.



Para crear una API REST robusta, es necesario comunicar cualquier mensaje de error y entradas no válidas al usuario de la API. Los mensajes de error que se deben usar dependen del error encontrado y de la situación involucrada. Al comunicar condiciones de error al cliente, utilice códigos de estado HTTP.

CÓDIGOS DE ESTADO HTTP



- **200 (OK):** envíe este código de estado HTTP al cliente cuando se recibió la solicitud, todo se procesó normalmente y los datos devueltos se devuelven correctamente al cliente. No comunique ningún error al cliente en un código de estado 200.
- **400 (Bad Request):** envíe este código de estado HTTP junto con un mensaje de error detallado en el cuerpo de la respuesta cuando un cliente envía una solicitud no válida.
- **500 (Internal Server Error):** este código de estado HTTP generalmente indica un problema dentro del código API que genera una excepción.

Referencias

- <https://styde.net/pull-request-en-github/#:~:text=Los%20pull%20request%20permiten%20no,de%20código%20a%20un%20proyecto.>
- <https://aprendegit.com/fork-de-repositorios-para-que-sirve/>
- <https://www.freecodecamp.org/espanol/news/la-guia-definitiva-para-git-merge-y-git-rebase/>
- <https://aprendeconalf.es/docencia/git/manual/manual-git.pdf>
- <https://www.atlassian.com/es/git/tutorials/saving-changes/git-stash#stashing-your-work>
- <https://www.atlassian.com/es/git/tutorials/undoing-changes/git-clean#:~:text=quit%206%3A%20help-,Resumen,del%20repositorio%20con%20git%20add%20.>
- <https://www.atlassian.com/es/git/tutorials/cherry-pick>
- <https://www.atlassian.com/es/git/tutorials/using-branches/merge-conflicts>