

Taller: Git _



Sección 1

Git Basics

Conocer qué es Git, sus estados y componentes básicos.

Desafío Branching I:
Fast-forward

Sección 2

Estrategia *recursive*

Entender el funcionamiento de la estrategia recursive

Demostración:
Recursive strategy

Sección 3

Resolución de Conflictos

Aprender a resolver conflictos en Git

Desafío Branching II:
Merge conflict

Sección 4

GitHub

Conocer GitHub como plataforma de desarrollo colaborativo y los elementos necesarios para su utilización.

Desafío GitHub:
Fork y pull request

Git Basics

¿Qué es Git?

Git es un software de control de versiones, gratuito y de código abierto diseñado para manejar proyectos de todos los tamaños con rapidez y eficiencia.



Control de Versiones

- Recuperar una versión anterior de nuestro proyecto.
- Mantener registro histórico de los cambios realizados al proyecto e información de quién los realizó.
- Ayudar a encontrar en qué momento se implementó un cambio que produjo errores.

¿Quién utiliza Git?

Companies & Projects Using Git

Google

facebook

Microsoft

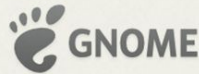
twitter

LinkedIn

NETFLIX



PostgreSQL



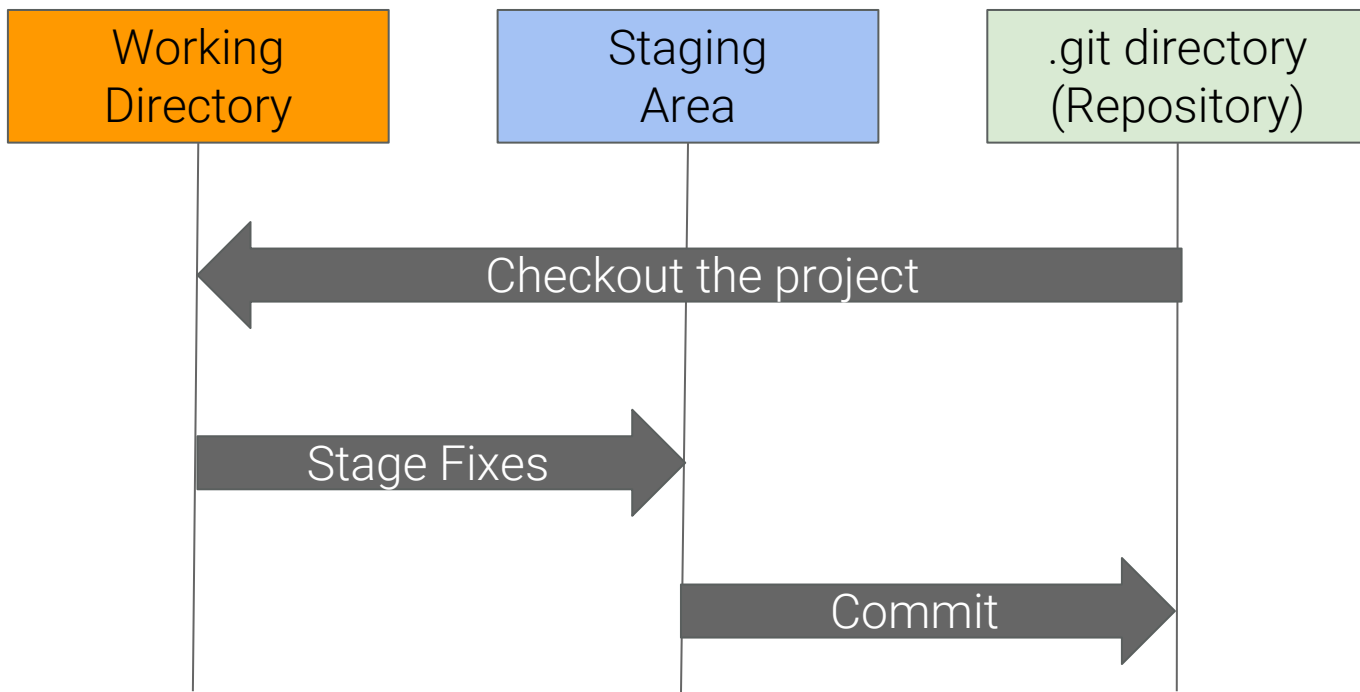
Estados

Luego de haber inicializado Git, nuestros archivos se pueden encontrar en uno de los siguientes estados.

1. Modified
2. Staged
3. Committed

Esto da lugar a las tres secciones más importantes de un proyecto Git.

Las tres secciones más importantes



Activación de Conceptos

Commit

Checksum

Merge

Head

Branch

Master

- En Git, la mayoría del trabajo se realizar de manera **local**.
- Es decir, **no necesito** conexión a internet para utilizar Git.
- **No** es lo mismo que GitHub.
- Toda la información de versionamiento se encuentra en la carpeta **.git**

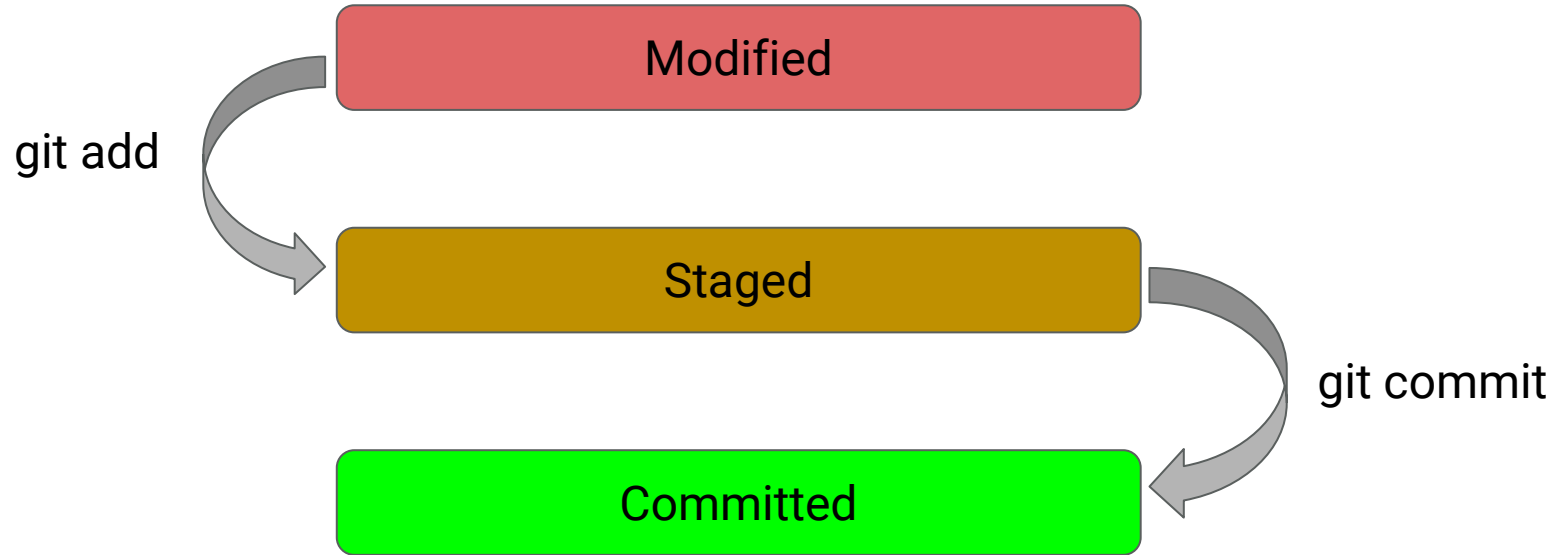
- Puedo subir mi proyecto a un servidor de repositorios a través de **remotes**.
- Un proyecto puede tener más de un **remote**.
- Si así lo requerimos, cada remote puede apuntar a un servidor distinto (GitHub, GitLab, Bitbucket, etc.)
- Al inicializar Git obtendremos -por defecto- un branch llamado **master**.

Git Basic Workflow

El flujo básico de Git es descrito al pasar por las tres secciones principales:

- Modificamos uno o varios archivos y guardamos los cambios.
- Agregamos los cambios a stage para ser commiteados.
- Generamos un nuevo commit.

Git Basic Workflow



Git Branches



Desafío Branching I: Fast-Forward

Desafío Branching I: Fast-Forward

1. Descargar el proyecto base.
2. Inicializar Git dentro del proyecto.
3. Agregar los archivos a stage y luego generar nuestro primer commit.

Desafío Branching I: Fast-Forward

Implementar: Al hacer click en la imagen debemos ser redirigidos a GitHub

4. Para eso crearemos una nueva rama de desarrollo e implementaremos la funcionalidad ahí.

```
git checkout -b development
```

Desafío Branching I: Fast-Forward

5. Ahora, posicionados en nuestra rama development, implementaremos la funcionalidad:

```
<a href="https://github.com/" target="blank">  
    
</a>
```

Desafío Branching I: Fast-Forward

6. Guardamos los cambios.
7. Ejecutamos `git status` para saber en qué estado se encuentra proyecto.
8. Agregamos los cambios a stage para ser commiteados.
9. Generamos un nuevo commit.
10. Volvemos a la rama master.

Desafío Branching I: Fast-Forward

¿Existe la imagen en la rama master?

11. Vamos a integrar los cambios realizados en **development** a la rama **master**.

¿Existe el link a la rama master?

Análisis del Desafío

Al momento de realizar el commit en development y luego volver a master lo más importante es darnos cuenta que nuestra rama **development** se encuentra un commit delante de master.

Es decir, los cambios implementados no se encuentran en master.

Análisis del Desafío

Como mencionamos con anterioridad, en este punto tenemos dos caminos:

1. Descartar los cambios de la rama development

```
git branch -D development
```

2. Integrar los cambios de la rama development a master

```
git merge master
```

El camino escogido fue el de implementar los cambios a través de merge.

¿Qué información obtendremos del merge realizado?

```
> git merge development
```

```
Updating 1acb730..62d980e
```

```
Fast-forward
```

```
index.html | 4 +++-
```

```
1 file changed, 3 insertions(+), 1 deletion(-)
```

Fast-forward

Corresponde al tipo de merge que se ejecuta cuando se cumple que:

La rama que queremos integrar corresponde a un descendiente directo de nuestra rama principal.

Dicho de otra forma:

El último commit de la rama master es ancestro directo del commit en la rama development.


```
git log --graph
```

```
* commit 62d980eef9a6826128825bf62e85243d35f48355 (development)
```

```
|   Author: Juan Pablo Cuevas <jp.cuevaslavin@gmail.com>
```

```
|   Date:    Wed Oct 24 12:35:09 2018 -0300
```

```
|
```

```
|       add link to index image
```

```
* commit 1acb730ae71e2a2c71089999cd4a7eb40b830db2
```

```
   Author: Juan Pablo Cuevas <jp.cuevaslavin@gmail.com>
```

```
   Date:    Web Oct 24 12:32:59 2018 -0300
```

```
       initial commit
```

¿Siempre un merge se implementa a través de fast-forward?

No.

Fast-forward es la estrategia de merge -por defecto- cuando nuestra **rama secundaria es descendiente directo de la rama principal** y no hay inconsistencias ni objeciones con respecto al merge.

En otras situaciones, obtendremos un comportamiento diferente.

¿Existen otras estrategias además de fast-forward?

Sí.

Dentro de ellas podemos destacar la estrategia **recursive**.

Estrategia *recursive*

Demostración

Al crear la rama development estamos generando un descendiente directo de master.

Todos los commits que generemos en development seguirán siendo generados a partir de una descendencia directa.

Al volver a master y generar un nuevo commit, perdemos la descendencia directa.

Cuando queremos generar un merge, y no existe descendencia directa, Git generará un nuevo commit que posee 2 ancestros:

1. El commit de la rama principal
2. El commit de la rama secundaria

Este commit recibe el nombre de merge commit.

```
Merge branch 'style'
```

```
# Please enter a commit message to explain why this merge is necessary,  
# especially if it merges an updated upstream into a topic branch.
```

```
#
```

```
# Lines starting with '#' will be ignored, and an empty message aborts  
# the commit.
```

```
~
```

```
~
```

```
~
```

Podemos modificar el comentario o utilizar el que existe por defecto.

```
> git merge style
```

Merge made by the 'recursive' strategy.

index.css | 2 +-

1 file changed, 1 insertion(+), 1 deletion(-)

La estrategia utilizada recibe el nombre de recursive.

git log --graph

```
>> git log --graph
*   commit e200b630a57de4d2262ef7835c36c1442ffc832c (HEAD -> master)
| \ Merge: 0f65e33 6293373
|  | Author: Juan Pablo Cuevas <jp.cuevaslavin@gmail.com>
|  | Date:   Wed Oct 24 14:46:54 2018 -0300
|  |
|  | Merge branch 'style'
|  |
|  | *   commit 62933736cc5f8f19ccdbdd71fa51b80862d32ae7 (style)
|  | | Author: Juan Pablo Cuevas <jp.cuevaslavin@gmail.com>
|  | | Date:   Wed Oct 24 14:44:00 2018 -0300
|  | |
|  | | update body background color
|  | |
|  | *   commit 0f65e333583785b3b9550c2172f152650d6a39dd
|  | / Author: Juan Pablo Cuevas <jp.cuevaslavin@gmail.com>
|  | | Date:   Wed Oct 24 14:45:04 2018 -0300
|  | |
|  | | add button
|  | |
|  | *   commit 62d980eef9a6826128825bf62e85243d35f48355 (development)
|  | | Author: Juan Pablo Cuevas <jp.cuevaslavin@gmail.com>
|  | | Date:   Wed Oct 24 12:35:09 2018 -0300
|  | |
|  | | add link to index image
|  | |
|  | *   commit 1acb730ae71e2a2c71089999cb4a7eb40b830db2
|  | | Author: Juan Pablo Cuevas <jp.cuevaslavin@gmail.com>
|  | | Date:   Wed Oct 24 12:32:59 2018 -0300
|  | |
|  | | initial commit
```

Resolución de Conflictos

Existe un tercer escenario donde obtendremos un comportamiento diferente:

Manipular un mismo archivo en dos ramas distintas y luego realizar un merge.

**¿Qué sucede si modificamos
el mismo archivo en dos ramas distintas?**

Desafío Branching II: Merge conflict

Desafío Branching II: Merge conflict

1. En la rama **master** crear y moverse a un branch nuevo llamado **feature_one**.
2. En el branch **feature_one** modificar el texto del botón en index.html.

```
<button type="button" class="btn">  
  <a href="https://github.com/" target="_blank">Click aquí para  
  ir a GitHub!</a>  
  
</button>
```

Desafío Branching II: Merge conflict

3. Guardar los cambios, agregar el archivo a stage y generar un commit.
4. Volver a la rama master.
5. Modificar el texto del botón en index.html.

```
<button type="button" class="btn">  
  <a href="https://github.com/" target="_blank">Haz click  
aquí</a>  
  
</button>
```

Desafío Branching II: Merge conflict

6. Guardar los cambios, agregar el archivo a stage y generar un commit.
7. Realizar un merge de **feature_one** a master.

Análisis y resolución del desafío

```
[>> git merge feature_one  
Auto-merging index.html  
CONFLICT (content): Merge conflict in index.html  
Automatic merge failed; fix conflicts and then commit the result.
```

El mensaje obtenido es evidente:

“Merge automático falló; soluciona los conflictos y commitea el resultado”

git status

```
>> git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   index.html

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

Ambos commits modificaron index.html

A partir de aquí tenemos dos caminos:

1. Abortar el merge.

```
git merge --abort
```

2. Solucionar el conflicto y realizar un nuevo commit.

Anatomía de un conflicto

Siempre un conflicto tendrá la siguiente estructura:

```
<<<<<< HEAD
< Contenido de la rama principal >
=====
< Contenido de la rama secundaria >
>>>>>> nombre_rama_secundaria
```

Solucionando el conflicto

Vamos a abrir el archivo en Visual Studio Code.

Este editor nos entrega, por defecto, una herramienta para solucionar conflictos de manera amigable.

```
<button type="button" class="btn">
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<<< HEAD (Current Change)
  <a href="https://github.com/" target="_blank">Haz click aquí!</a>
=====
  <a href="https://github.com/" target="_blank">Click aquí para ir a GitHub!</a>
>>>>>>> feature_one (Incoming Change)
</button>
```

Utilizando esta herramienta, podemos escoger entre:

- **Accept Current Change:** Mantener el contenido de la rama actual.
- **Accept Incoming Change:** Mantener el contenido de la rama secundaria.
- **Accept Both Changes:** Mantener ambos contenidos.
- **Compare Changes:** Comparar los contenidos en ventas nuevas.

Vamos a aceptar los cambios de la rama secundaria

Damos clic en **Accept Incoming Change**

```
<button type="button" class="btn">  
  <a href="https://github.com/" target="_blank">Click aquí para ir a GitHub!</a>  
</button>
```

Hemos solucionado el conflicto, ahora sólo debemos:

- Guardar los cambios.
- Agregar el archivo a stage.
- Generar un nuevo commit.

git log --graph

```
>> git log --graph
*   commit df4de6db5708f896bbae871c88949b4a8e82ce8c (HEAD -> master)
| \ Merge: 4363c81 1583cda
| | Author: Juan Pablo Cuevas <jp.cuevaslavin@gmail.com>
| | Date:   Wed Oct 24 17:43:24 2018 -0300
| |
| |     fix conflict
| |
| *   commit 1583cda199456ae4c0b63b1d95031dfabcc80212 (feature_one)
| | Author: Juan Pablo Cuevas <jp.cuevaslavin@gmail.com>
| | Date:   Wed Oct 24 16:53:41 2018 -0300
| |
| |     update button text
| |
| *   commit 4363c810c4e1532c283bc1660d94244be84bc654
| / Author: Juan Pablo Cuevas <jp.cuevaslavin@gmail.com>
| | Date:   Wed Oct 24 16:57:37 2018 -0300
| |
| |     update button text
| |
| *   commit e200b630a57de4d2262ef7835c36c1442ffc832c
| \ Merge: 0f65e33 6293373
| | Author: Juan Pablo Cuevas <jp.cuevaslavin@gmail.com>
| | Date:   Wed Oct 24 14:46:54 2018 -0300
| |
| |     Merge branch 'style'
```

GitHub

¿Qué es GitHub?

Plataforma de desarrollo colaborativo que nos permite **almacenar y gestionar** proyectos Git.

Existen diversas plataformas para este propósito, entre ellas podemos destacar:

- GitHub
- GitLab
- Bitbucket

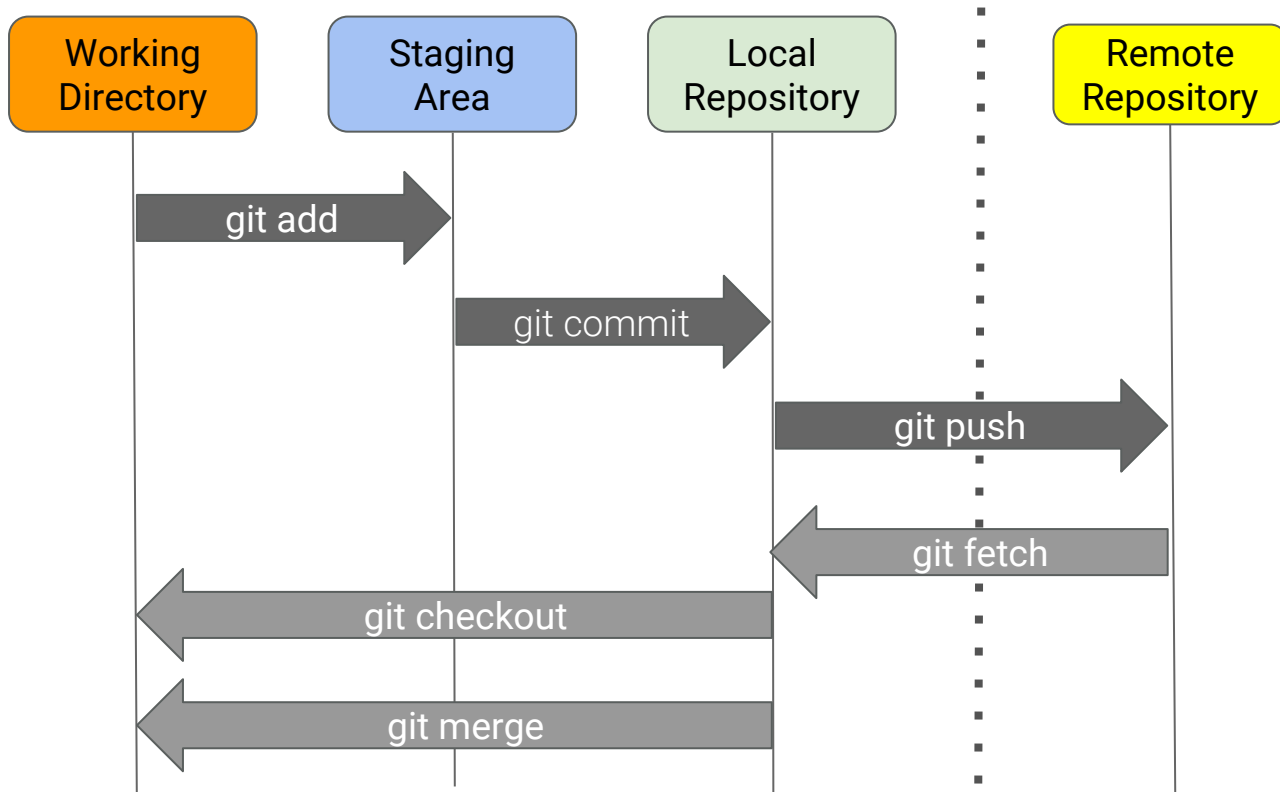
¿Control de versiones?

Nos permite mantener registro de los cambios realizados a un archivo, o a un conjunto de archivos, en el tiempo.

Git vs GitHub



Git	GitHub
Software de versionamiento que registra cambios en el tiempo	Plataforma de desarrollo colaborativo que nos permite almacenar y gestionar proyectos Git
Se instala y funciona de manera local en nuestro computador	Debemos crear una cuenta en la plataforma web
No necesitamos conexión a internet	Necesitamos conexión a internet



La comunicación entre nuestro repositorio local y el servidor se realiza a través de la implementación de remotes.

Remotes

Corresponden a direcciones que apuntan a servidores donde podremos, entre otras funcionalidades:

- Almacenar y gestionar nuestro proyecto.
- Trabajar de manera colaborativa.
- Compartir el código de nuestro proyecto.

Agregando un remote

Para agregar un remote lo primero que debemos hacer es crear el repositorio en la plataforma que decidamos utilizar.

Para este ejemplo utilizaremos GitHub.

Create a new repository

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

Owner

 DesafíoLatam ▾

Repository name

/ taller-git-landing ✓

Great repository names are short and memorable. Need inspiration? How about **bookish-octo-train**.

Description (optional)

Landing para taller de Git



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** ▾



Create repository

Ahora podemos agregar -en nuestro repositorio local- el remote que apunte a este servidor.

```
git remote add nombre_del_remoto url_del_remoto
```

Por convención un remote único y principal suele llamarse origin.

Podemos mantener o modificar este nombre, nosotros lo llamaremos GitHub.

```
git remote add github git@github.com:DesafioLatam/taller-git-landing.git
```

¡Listo!

Nuestro repositorio local tiene un remote que apunta a GitHub

¿Puedo tener más de un remote?

Sí.

Puedes agregar, renombrar o eliminar tantos remotes como desees.

Cada remote puede apuntar a una URL distinta, sin restricción alguna.

Remotes

Cuando necesitemos enviar el contenido de nuestro repositorio local a uno remoto, lo haremos utilizando la instrucción push

```
git push -u nombre_del_remoto nombre_de_la_rama
```


- Nuestro remoto se llama **github**
- Nuestra rama se llama **master**

```
git push -u github master
```




El parámetro -u es opcional, sin embargo, se recomienda su uso.
Lo que hace es generar una referencia para tracking.

De tal manera, podemos comparar local vs remote y entregarnos información como por ejemplo:

"Tu repositorio local está 2 commits delante de tu remoto".



[Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)

[DesafioLatam](#) / [taller-git-landing](#)

Unwatch 6

Star 0

Fork 0

<> Code

Issues 0

Pull requests 0

Projects 0

Wiki

Insights

Settings

Landing para taller de Git

Edit

[Manage topics](#)

1 commit

1 branch

0 releases

1 contributor

Branch: master


New pull request




Create new file

Upload files

Find file

Clone or download


 **jpcuevaslavin** initial commit Latest commit d621754 2 hours ago

 assets	initial commit	2 hours ago
 .gitignore	initial commit	2 hours ago
 index.html	initial commit	2 hours ago

Help people interested in this repository understand your project by adding a README.

Add a README

© 2018 GitHub, Inc. [Terms](#) [Privacy](#) [Security](#) [Status](#) [Help](#)



[Contact GitHub](#) [Pricing](#) [API](#) [Training](#) [Blog](#) [About](#)

Remotes

Corresponden a direcciones que apuntan a servidores donde podremos, entre otras funcionalidades:

- Almacenar y gestionar nuestro proyecto.
- Trabajar de manera colaborativa.
- Compartir el código de nuestro proyecto.

Trabajando con remotes

- En GitHub los repositorios son -por defecto- públicos.
- Cualquier usuario puede acceder y descargar su contenido.

¿Podemos modificar el contenido de un repositorio público?

No.

A menos que nuestro usuario de GitHub se encuentre en la lista de colaboradores del repositorio.

Sin embargo, existen mecanismos que nos permiten:

- Generar una copia de un repositorio público.
- Asociar esta copia a nuestra propia cuenta.
- Poder descargar este repositorio, generar cambios y luego subirlos.
- Informar al autor del repositorio original de estos cambios, transparentarlos y darle la opción de integrarlo.

**Estos mecanismos se conocen
como fork y pull request**

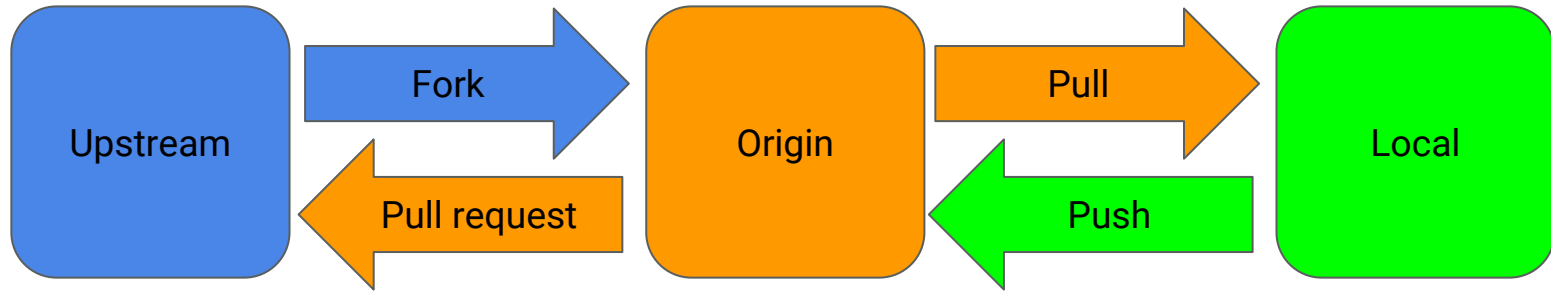
Fork

Corresponde a la acción de generar una copia exacta de un repositorio y asociar esta copia a nuestra cuenta (manteniendo la referencia al autor).

Pull Request

Corresponde a la acción de solicitar la implementación de nuestros cambios en un repositorio.

Big Picture



Desafío GitHub: Fork y Pull request

Desafío GitHub: Fork y Pull request

Imaginemos que el proyecto a utilizar corresponde a un repositorio público desarrollado por el profesor y ampliamente utilizado por la comunidad de desarrolladores.

Los alumnos -al ver el proyecto- consideran que hay varios cambios que podrían mejorar el contenido del repositorio y deciden colaborar con el proyecto.

Desafío GitHub: Fork y Pull request

1. Crear un fork del repositorio.
2. Clonar el repositorio forkeado a nuestro computador.
3. Generar cambios simples en el proyecto.
4. Commitear los cambios.
5. Hacer push a nuestro repositorio forkeado.
6. Crear un pull request describiendo los cambios implementados.

El profesor analizará los pull requests y decidirá qué cambios implementar.

**¡Así funciona la colaboración en proyectos
de código abierto mantenidos por la comunidad!**

{desafío}
latam_

*Academia de
talentos digitales*

www.desafiolatam.com