

Academia Java Xideral

16 Diciembre 2022

Samuel Cornelio Santiago

Instructor: Miguel Rugerio

Examen semana 4

Git y Github

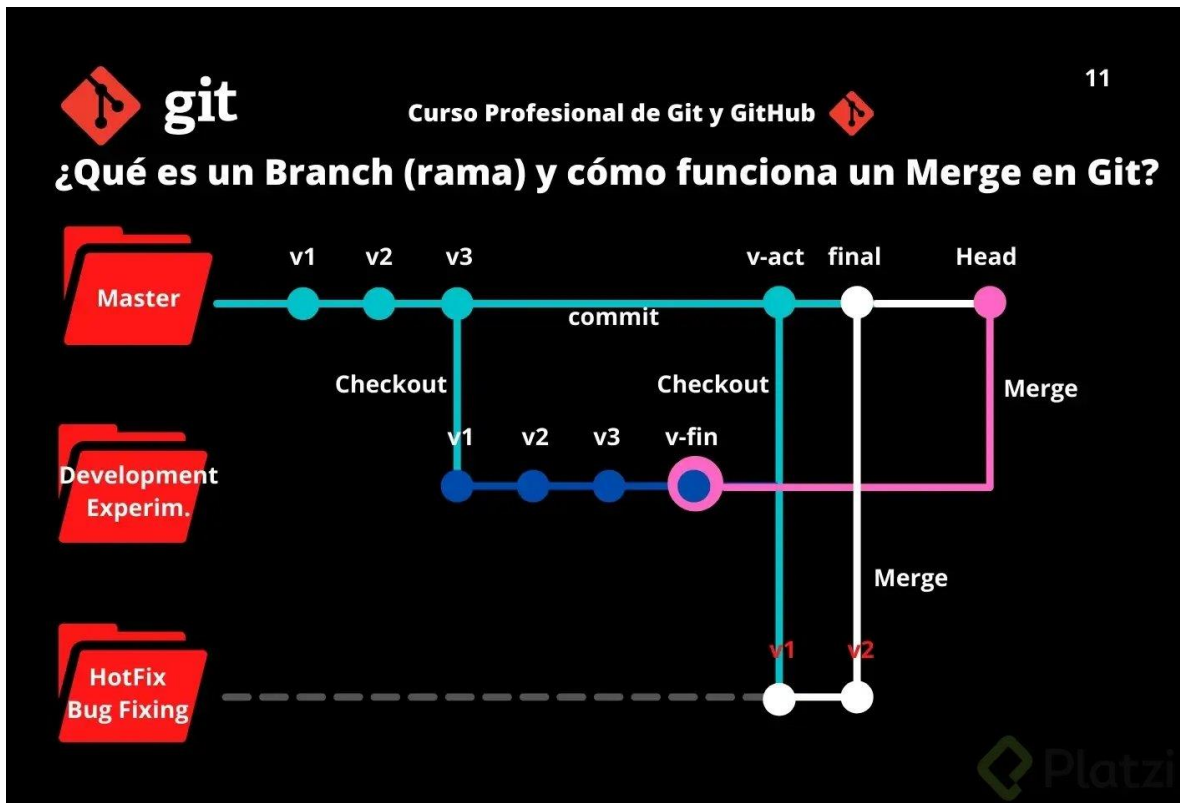
Rest Api



## Branches

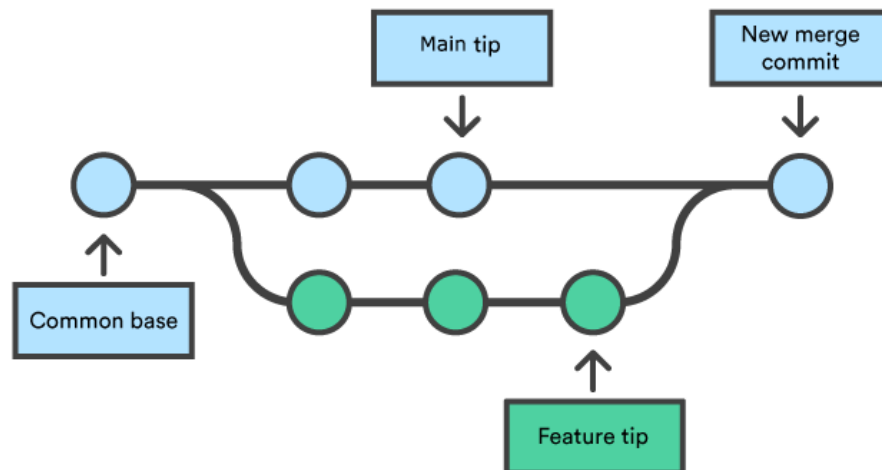
Una rama o branch es una versión del código del proyecto sobre el que estás trabajando. Estas ramas ayudan a mantener el orden en el control de versiones y manipular el código de forma segura.

Un branch o rama en Git es una rama que proviene de otra. Imagina un árbol, que tiene una rama gruesa, y otra más fina, en la rama más gruesa tenemos los commits principales y en la rama fina tenemos otros commits menos relevantes.



## Merge

git merge combinará varias secuencias de confirmaciones en un historial unificado. En los casos de uso más frecuentes, git merge se utiliza para combinar dos ramas.



## conflictos

Algunas veces la unión de dos ramas no resulta tan bien, sino que ocurre un conflicto, esto cuando los commits de la rama a fusionar y la rama actual modifican la misma parte en un archivo en particular y git no puede decidir cuál versión elegir, y te avisa que tu debes resolverlo. Por ejemplo:

Supongamos que un directorio tenemos un archivo `index.html` con el siguiente contenido:

```
1.  <!DOCTYPE HTML>
2.  <html>
3.    <head>
4.      <title>Titulo</title>
5.    </head>
6.    <body>
7.      <p>Contenido de la web</p>
8.    </body>
9.  </html>
```

Inicializaremos en el un repositorio en el mismo haciendo git init y luego haremos nuestro primer commit con git add index.html y luego git commit -m "commit inicial"

Vamos a crear una nueva rama para añadir algo de contenido:

```
1. git checkout -b contenido
```

Con ello ya estamos en la nueva rama y ahora vamos a cambiar el título.

Guardamos los cambios y hacemos commit en esa rama

```
1. git commit -a -m "cambios en el titulo"
```

Nos movemos de nuevo a la rama master

```
1. git checkout master
```

Hacemos otros cambios en el archivo incluyendo el título y luego commit de los cambios

```
1. git commit -a -m "se añade más contenido"
```

Ahora intentamos hacer merge con la rama creada anteriormente.

```
1. git merge contenido
```

En este caso no podrá hacer el merge y nos mostrará que hay un conflicto que no nos permitirá continuar hasta que se resuelva:

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

Git no proporciona una ayuda diciéndonos que archivo tiene el conflicto, el cual al abrirlo nos muestra cuáles son los cambios tanto de una rama como de la otra:

```
<!DOCTYPE HTML>
<html>
  <head>
<<<<<<< HEAD
    <title>Nuevo Titulo</title>
=====
    <title>Nuevo Titulo para la web</title>
>>>>>>> contenido
  </head>
  <body>
    <p>Contenido de la web</p>
    <p>Nuevo párrafo de la página</p>
  </body>
</html>
```

donde tenemos que elegir entre lo que está entre <<<<<< HEAD y ===== que es contenido que tenemos en la rama donde estamos haciendo el merge (master) o entre ===== y >>>>>> contenido donde están los cambios hechos en la rama que queremos unir (contenido).

Para ello arreglamos el archivo con los cambios elegidos, guardamos, agregamos y hacemos commit de los cambios

```
1. git commit -a
```

y de esta manera logramos hacer merge con éxito.

Otra manera para resolver los conflictos es que podemos indicarle de antemano a git que estrategia tomar cuando tiene que decidir un conflicto, esto con las opciones ours y theirs, de esta manera:

```
1. git merge -s recursive -X theirs rama-a-fusionar
```

Deshaciendo merges

Si hemos realizado un merge con una rama con la que no queríamos, puedes hacer:

```
1. git reset --merge ORIG_HEAD
```

Unir automáticamente múltiples commits en uno durante el merge

Se puede unir todos los commits de una rama y fusionarlos en la rama actual especificando la opción --squash, es decir,

```
1. git merge --squash rama-a-fusionar
```

Merge es una de las alternativas que nos da git para unir las ramas de nuestro repositorio pero no es la única; en la próxima entrega conoceremos a rebase y su diferencia con merge.

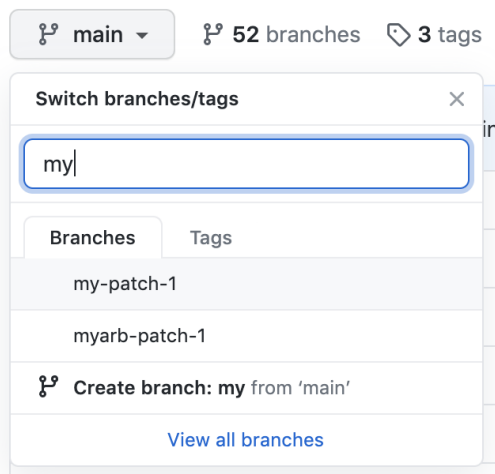
## PULL REQUEST

Crear una solicitud de incorporación de cambios

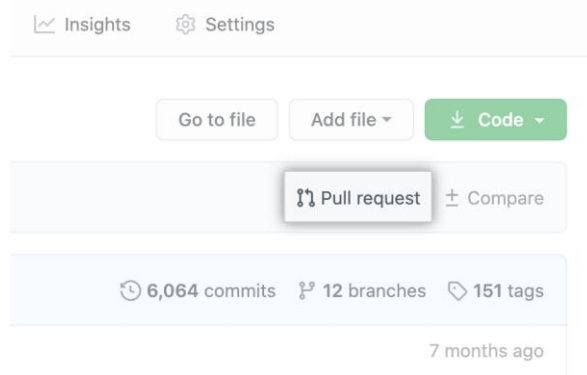
Crea una solicitud de incorporación de cambios para proponer cambios en un repositorio y colaborar con ellos. Estos cambios se proponen en una rama, lo cual garantiza que la rama predeterminada contenga únicamente trabajo finalizado y aprobado.

Crear una solicitud de extracción

1. En, vaya a la página principal del repositorio.
2. En el menú "Branch" (Rama), elige la rama que contiene tus confirmaciones.



1. Encima de la lista de archivos, haga clic en Solicitud de incorporación de cambios.



3. Use el menú desplegable de la rama base para seleccionar la rama en la que quiera combinar los cambios y, después, use el menú desplegable de la rama de comparación para elegir la rama de tema en la que ha realizado los cambios.


## 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: main ▼ ← compare: my-patch-1 ▼ ✓ Able to merge. These branches can be automatically merged.

1. Escribe un título y una descripción para tu solicitud de extracción.

Please review the [guidelines for contributing](#) to this repository.



Add CONTRIBUTING.md

Write

Preview

AA ▼ B i “ < > ↺ ⋮ ≡ ≡ ↶ @ 📌

Let's add a contributing file, so we can work better, together!

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

Labels

None yet

Milesto

No mile

Assign

No one

1. Para crear una solicitud de incorporación de cambios que esté lista para revisión, haga clic en Crear solicitud de incorporación de cambios. Para crear un borrador de una solicitud de incorporación de cambios, use el menú desplegable y seleccione Crear solicitud de incorporación de cambios de borrador, y después haga clic en Solicitud de incorporación de cambios de borrador. Para más información sobre el borrador de solicitudes de incorporación de cambios, vea "Acerca de las solicitudes de incorporación de cambios".

pasting from the clipboard.

Create Pull Request ▼

✓ Create Pull Request

Open a pull request that is ready for review

Create Draft Pull Request

Cannot be merged until marked ready for review

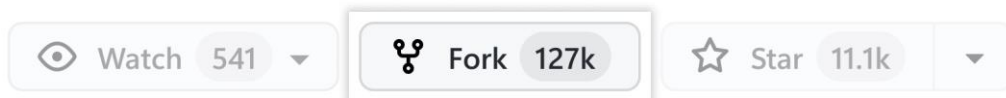
## FORK

### Bifurcar un repositorio

Una bifurcación es un nuevo repositorio que comparte la configuración de visibilidad y código con el repositorio “acendente” original.

Bifurcar un repositorio


1. En GitHub.com, ve al repositorio
2. En la esquina superior derecha de la página, haga clic en Fork (Bifurcar).



3. Selecciona un propietario para el repositorio bifurcado.

### Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. [View existing forks.](#)

**Owner \***  
 octocat▼


**Repository name \***  
Spoon-Knife ✓

By default, forks are named the same as their parent repository. You can customize the name to distinguish it further.

4. De forma predeterminada, las bifurcaciones tienen el mismo nombre que sus repositorios ascendentes. Puedes cambiar el nombre de la bifurcación para distinguirlo aún más.

### Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. [View existing forks.](#)

**Owner \***  
 octocat▼

**Repository name \***  
Spoon-Knife ✓

By default, forks are named the same as their parent repository. You can customize the name to distinguish it further.

5. Opcionalmente, puedes agregar una descripción de la bifurcación.



## Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. [View existing forks.](#)

Owner \*

 octocat▼

Repository name \*

Spoon-Knife ✓

By default, forks are named the same as their parent repository. You can customize the name to distinguish it further.

Description (optional)

This repo is for demonstration purposes only.

6. Elige si quieres copiar solo la rama predeterminada, o bien todas las ramas en la nueva bifurcación. En muchos escenarios de bifurcación, como los de contribución a proyectos de código abierto, solo tienes que copiar la rama predeterminada. De forma predeterminada, solo se copia la rama predeterminada.

Owner \*

Select an owner ▼

Repository name \*

Spoon-Knife

By default, forks are named the same as their parent repository. You can customize the name to distinguish it further.

Description (optional)

This repo is for demonstration purposes only.

☒ Copy the **main** branch only

Contribute back to octocat/Spoon-Knife by adding your own branch. [Learn more.](#)

7. Haz clic en Crear bifurcación.

☒ Copy the **main** branch only

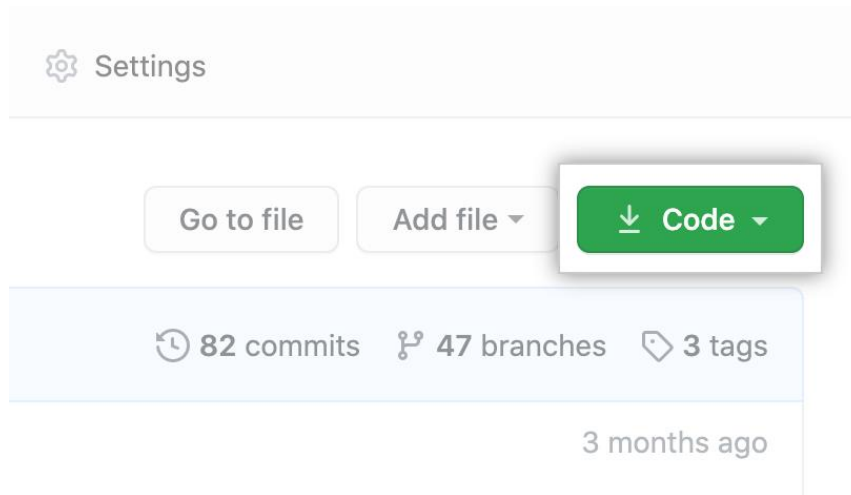
Contribute back to octocat/Spoon-Knife by adding your own branch. [Learn more.](#)




 You are creating a fork in your personal account.

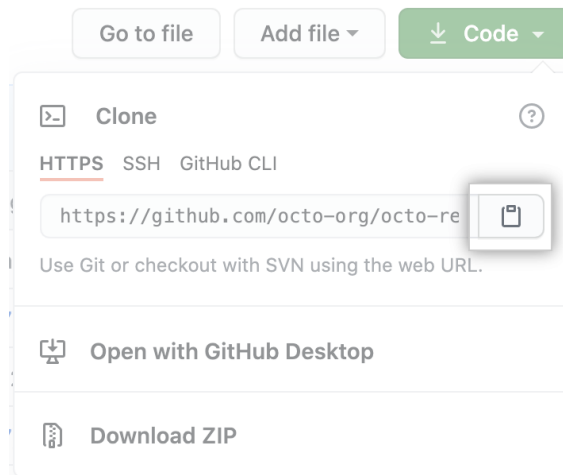
Create fork

## Clonar tu repositorio bifurcado

1. En GitHub.com, ve a tu bifurcación en el repositorio Spoon-Knife.
2. Encima de la lista de archivos, haga click en code.



3. Copia la dirección URL del repositorio.
  - Para clonar el repositorio con HTTPS, en «HTTPS» haz clic en .
  - Para clonar el repositorio mediante una clave SSH, incluido un certificado emitido por la entidad de certificación SSH de la organización, haga clic en Usar SSH y luego en .
  - Para clonar un repositorio mediante GitHub CLI, haz clic en GitHub CLI y, después, haz clic en .



Abra Git Bash. Cambia el directorio de trabajo actual a la ubicación en donde quieres clonar el directorio.

4. Escriba git clone y pegue la dirección URL que ha copiado antes. Tendrá este aspecto, con su nombre de usuario de GitHub en lugar de YOUR-USERNAME:

## Rebase

Acerca de la fusión mediante cambio de base de Git

El comando git rebase te permite cambiar fácilmente una serie de confirmaciones, modificando el historial de tu repositorio. Puedes reordenar, editar o combinar confirmaciones.

### Cambiar de base las confirmaciones con una rama

Para cambiar de base todas las confirmaciones entre otra rama y el estado de rama actual, puedes ingresar el siguiente comando:

```
$ git rebase --interactive other_branch_name
```

### Cambiar de base las confirmaciones en un momento específico

Para cambiar de base las últimas confirmaciones en tu rama actual, puedes ingresar el siguiente comando:

```
$ git rebase --interactive HEAD~7
```

### Comandos disponibles mientras se cambia de base

Hay seis comandos disponibles mientras se cambia la base:

pick

pick simplemente significa que se incluye la confirmación. Reordenar los comandos pick cambia el orden de las confirmaciones cuando la fusión mediante cambio de base está en curso. Si eliges no incluir una confirmación, debes eliminar la línea completa.

reword

El comando reword es similar a pick, pero después de usarlo, la fusión mediante cambio de base se pausará y le dará la oportunidad de modificar el mensaje de confirmación. Cualquier cambio hecho por la confirmación no se ve afectado.

edit

Si elige realizar edit en una confirmación, se le dará la oportunidad de modificar la confirmación, lo que significa que puede agregar o cambiar la confirmación por completo. También puedes realizar más confirmaciones antes de continuar con el cambio de base. Esto te permite dividir una confirmación grande en otras más pequeñas o eliminar cambios erróneos hechos en una confirmación.

squash

Este comando te permite combinar dos o más confirmaciones en una única confirmación. Una confirmación se combina en la confirmación de arriba. Git te da la oportunidad de escribir un mensaje de confirmación nuevo describiendo ambos cambios.

fixup

Esto es similar a squash, pero la confirmación que se va a combinar tiene su mensaje descartado. La confirmación simplemente se fusiona en la confirmación de arriba y el mensaje de la confirmación anterior se usa para describir ambos cambios.

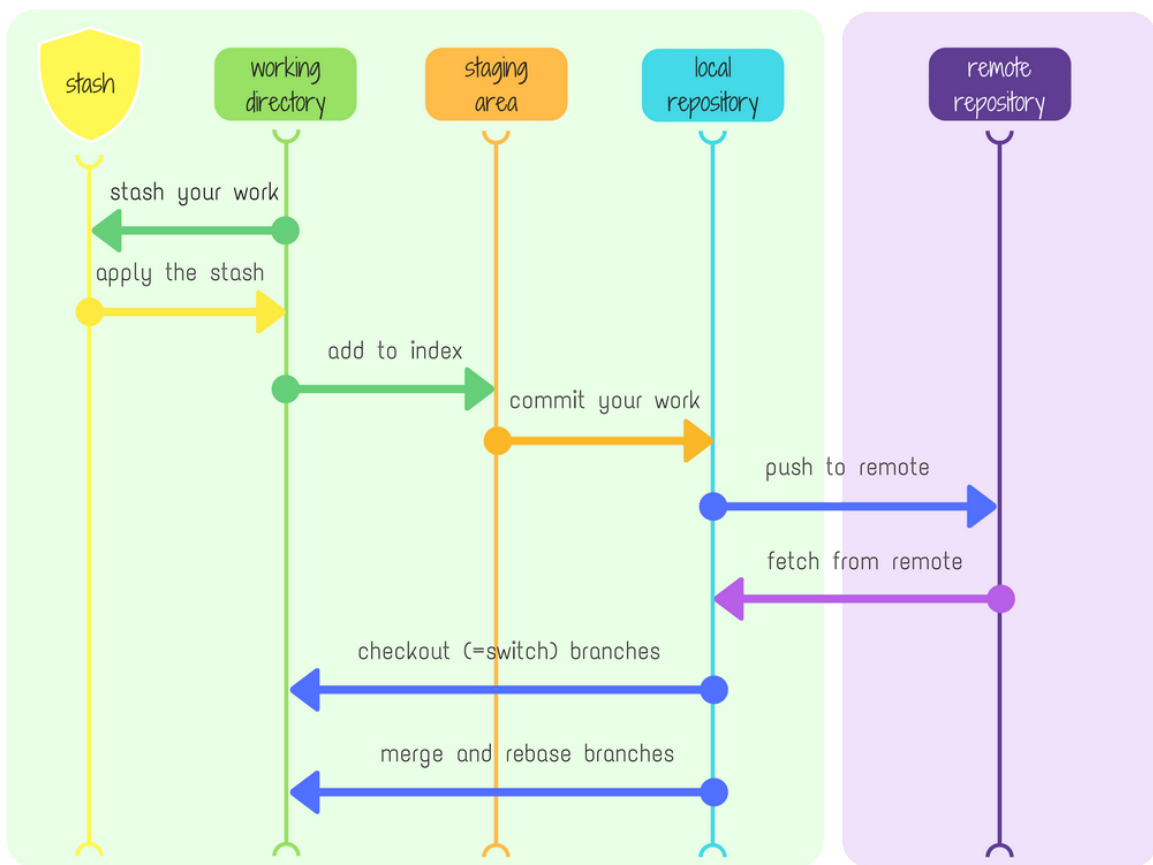
exec

Esto te permite ejecutar comandos shell de forma arbitraria con una confirmación.

## Stash

Guardado provisional de cambios

Puedes guardar tus cambios temporalmente sin confirmarlos en una rama si los acumulas.



### Cómo guardar el trabajo en un stash

El comando `git stash` coge los cambios sin confirmar (tanto los que están preparados como los que no), los guarda aparte para usarlos más adelante y, acto seguido, los deshace en el código en el que estás trabajando.

Llegados a este punto, tienes libertad para hacer cambios, crear confirmaciones, cambiar de rama y efectuar cualesquiera otras operaciones de Git; y, luego, regresar y volver a aplicar el stash cuando lo tengas todo listo.

Ten en cuenta que el stash es local para tu repositorio de Git y que los stashes no se transfieren al servidor cuando subes los cambios al repositorio remoto.

Cómo volver a aplicar los cambios de un stash

Puedes volver a aplicar los cambios de un stash mediante el comando `git stash pop`.

## CLEAN

Mientras estamos trabajando en un repositorio podemos añadir archivos a él, que realmente no forma parte de nuestro directorio de trabajo, archivos que no se deberían de agregar al repositorio remoto.

El comando `clean` actúa en archivos sin seguimiento, este tipo de archivos son aquellos que se encuentran en el directorio de trabajo, pero que aún no se han añadido al índice de seguimiento de repositorio con el comando `add`.

```
$ git clean
```

La ejecución del comando predeterminado puede producir un error. La configuración global de Git obliga a usar la opción `force` con el comando para que sea efectivo. Se trata de un importante mecanismo de seguridad ya que este comando no se puede deshacer.

Revisar que archivos no tienen seguimiento.

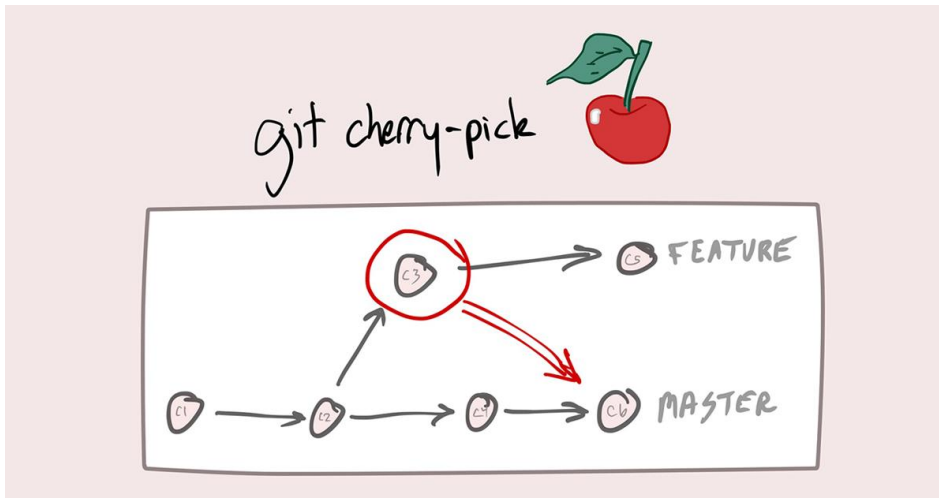
```
$ git clean --dry-run
```

Eliminar los archivos listados de no seguimiento.

```
$ git clean
```

## Cherry Pick

git cherry-pick es un comando poderoso que permite seleccionar confirmaciones de Git arbitrarias por referencia y agregarlas al HEAD de trabajo actual. La selección de Cherry-pick es el acto de seleccionar una confirmación de una rama y aplicarla a otra. git cherry-pick puede ser útil para deshacer cambios. Por ejemplo, supongamos que una confirmación se realiza accidentalmente en la rama incorrecta. Puede cambiar a la rama correcta y seleccionar la confirmación donde debería pertenecer.



git cherry-pick es una herramienta útil para algunos escenarios...

### Colaboración en equipo.

A menudo, un equipo encontrará miembros individuales trabajando en o alrededor del mismo código. Tal vez una nueva característica del producto tenga un componente de backend y frontend. Puede haber algún código compartido entre dos sectores de productos. Tal vez el desarrollador del backend crea una estructura de datos que el frontend también necesitará utilizar. El desarrollador frontend podría usar git cherry-pick para elegir la confirmación en la que se creó esta estructura de datos hipotética. Esta elección permitiría al desarrollador frontend continuar el progreso en su lado del proyecto.

### Correcciones de errores

Cuando se descubre un error, es importante entregar una solución a los usuarios finales lo más rápido posible. Para un escenario de ejemplo, digamos que un desarrollador ha comenzado a trabajar en una nueva característica. Durante el desarrollo de esa nueva función, identifican un error preexistente. El desarrollador crea una confirmación explícita para corregir este error. Esta nueva confirmación de parche se puede seleccionar directamente en la rama principal para corregir el error antes de que afecte a más usuarios.

## Deshacer cambios y restaurar confirmaciones perdidas

A veces, una rama de características puede volverse obsoleta y no fusionarse con la principal. A veces, una solicitud de extracción puede cerrarse sin fusionarse. Git nunca pierde esas confirmaciones y, a través de comandos como `git log` y `git reflog`, se pueden encontrar y recuperar.

## Buenas Prácticas de Api Rest

Muchas API utilizan la transferencia de estado representacional (REST) que devuelve datos en formato JSON como una forma de hacer que esos datos estén disponibles. Los desarrolladores que esperan crear una API REST robusta y flexible suelen seguir un conjunto de prácticas recomendadas.

### Desarrolle sus API para producir datos JSON

Al diseñar una API REST que usarán las aplicaciones móviles creadas con Dropsource, es importante recordar que las aplicaciones Dropsource están diseñadas para conectarse a una API REST que devuelve datos JSON. Por lo tanto, la API REST debe producir y devolver datos JSON. La mayoría de los lenguajes de programación modernos permiten una configuración ambiental para especificar qué producirá el servicio. Si el back-end se crea con algo como Spring Boot, lo más probable es que el valor predeterminado sea JSON para todos los controladores REST marcados con una anotación `@RestController`.

### Diseñe API usando principios REST clave

Puede ser bastante difícil realizar cambios significativos en una API REST una vez que se ha publicado, por lo que es esencial diseñarla correctamente por adelantado. Uno de los principios clave de Fielding 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 URL de cada punto final debe usar sustantivos para indicar claramente qué recurso se representa. Recuerde ser consistente con los nombres de los recursos y esforzarse por usar sustantivos en plural. Por ejemplo, favorezca "personas" sobre "persona" y "libros" sobre "libro". No mezcle 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

Para las API REST, los verbos HTTP comunes nos brindan la contrapartida de acción para nuestros recursos basados en sustantivos.

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.

PUT y PATCH pueden parecer casi lo mismo, ya que ambos están diseñados para realizar actualizaciones en un recurso existente. PATCH no se usa con tanta frecuencia como PUT, pero es



importante entender la diferencia. La diferencia es que PUT se usa para reemplazar completamente un recurso, mientras que PATCH se usa solo para reemplazar parcialmente un recurso.

### Ejemplo de **ruta de API REST**

Considere el siguiente ejemplo para un sistema de administración de usuarios. Al administrar usuarios, un sistema debe poder agregar nuevos usuarios a la base de datos, modificar los atributos del usuario, eliminar un usuario y asignar roles. El siguiente ejemplo de Swagger muestra una serie de puntos finales de API simples que definen un recurso de nivel superior llamado "personas". Cada punto final establece claramente el sustantivo (personas) que representa cada punto final y la acción (verbo HTTP) que se está realizando:

<b>persons</b> Operations about users of the system	
<b>POST</b>	<b>/persons</b> Create a new user in the database. (Note: JSON representation of a new user's data should be contained in the body.)
<b>GET</b>	<b>/persons/</b> Return a list of all users in the database.
<b>GET</b>	<b>/persons/{username}</b> Return a user by supplying a unique user name.
<b>PUT</b>	<b>/persons/{username}</b> 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).
<b>DELETE</b>	<b>/persons/{username}</b> Delete an existing user whos unique username is supplied.

### Ejemplo de implementación de Swagger de API REST de persona.

Los recursos no existen de forma aislada, pero la mayoría de las veces tienen relaciones con otros recursos. Una mejor práctica a seguir al decidir cómo modelar las relaciones en su API REST es considerar qué forma sería la más clara para el usuario de la API. Una buena regla es usar una relación "pertenece a". Por ejemplo, en el sistema de gestión de usuarios anterior, una persona podía tener un conjunto de roles asociados a su perfil. Para representar un recurso que devuelve los roles de una persona dado un nombre de usuario, se usaría la siguiente URL:

<b>GET</b>	<b>/persons/{username}/roles</b> Return a list of roles for a person given a unique username
------------	--

En este ejemplo, se puede considerar que los roles "pertenecen a" una persona. Un usuario potencial de la API podría mirar esta URL y, sin saber nada acerca de la estructura general de la API, podría descubrir qué hace este recurso.

Aquí hay algunos ejemplos más de URL que modelan una relación entre un recurso de nivel superior y subrecursos. Estos ejemplos muestran cómo funcionan los verbos HTTP para manipular las relaciones de recursos.

GET	/persons/{username}/roles/{rolename}	Get a single role for a single user given the user name and role name.
POST	/persons/{username}/roles	Add a new Role to a user given the user name.
PATCH	/persons/{username}	Partially update the information for a given user name.
PATCH	/persons/{username}/roles/{rolename}	Partially update the information for a single role given a role name for a user given a user name.
DELETE	/persons/{username}/roles/{rolename}	Delete a single role given a role name for a user given a user name.

## Documente su API utilizando la especificación OpenAPI

Se podría desarrollar la mejor API REST del mundo, pero ¿qué tan útil es si ningún desarrollador puede descubrirla o descubrir qué ofrece? Para que los usuarios de la API obtengan visibilidad de su API REST y la hagan lo más fácil de usar posible, documente la API utilizando la especificación OpenAPI. Al usar Dropsource para desarrollar aplicaciones móviles, Dropsource requiere que cualquier API REST que se use tenga todos los puntos finales documentados usando una especificación OpenAPI.

OpenAPI permite que su documentación crezca de forma interactiva a medida que escribe código. Esto es crucial. Para cualquiera que alguna vez haya sido responsable de escribir documentación, queda muy claro que generalmente va a la zaga del código. Cuando la documentación de la API está en línea, alienta a los desarrolladores a visualizar y consumir sus puntos finales.