Índice

2. COMANDOS

- git add
- git branch
- git checkout
- git clean
- git clone
- git commit
- git commit -- amend
- git config
- git fetch
- git init
- git log
- git merge
- git pull
- git push
- git rebase
- git reflog
- git remote
- git reset

- git revert

- git status

3. HERRAMIENTAS

- git extras

- GitLab

- Zenhub.io

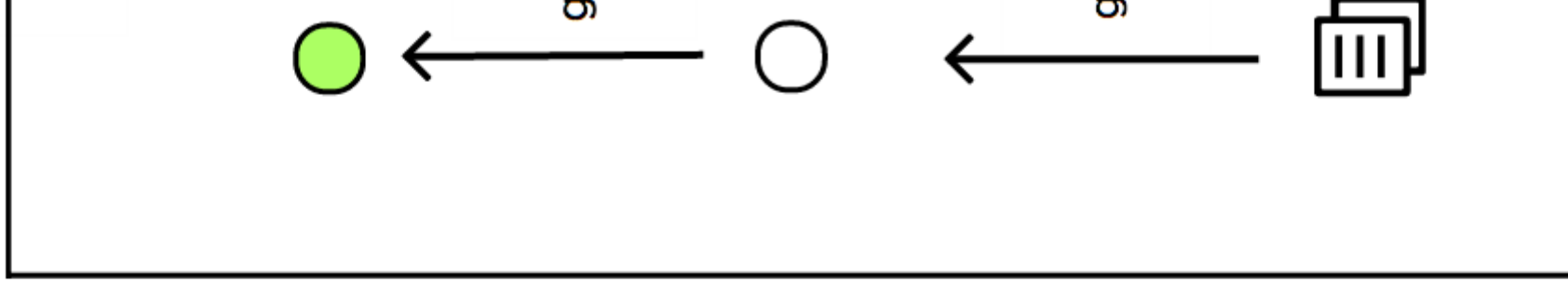
1. **CONCEPTOS**

Arquitectura de árbol

El concepto más importante de Git es la arquitectura de árbol.

Es la iteración básica para poder registrar cambios e ir construyendo un repositorio.

Está formado por 3 áreas y 2 acciones.



Ambientes (Local, Producción, Testing)

Los ambientes, que usualmente se utilizan para identificar “Deployment” ó arquitectura de despliegues, son entornos, donde se ejecuta software, con diferentes objetivos y reglas.

Dependiendo del tipo y tamaño del proyecto se pueden crear muchos ambientes, pero generalmente están enfocados en 3 tipos:

1. Development (Desarrollo)

Servidor de desarrollo en local. Regularmente es tu máquina, corriendo el proyecto, en el cual puedes ejecutarlo y desplegarlo en tu navegador.

2. - Production (Producción)

Se refiere al servidor donde corre el proyecto "en línea”, donde los usuarios pueden interactuar con él.

3. Testing

Son exámenes que se ejecutan sobre un proyecto con la finalidad de encontrar fallos en el código.
Puede suceder en un “Central Repository” como GitHub ó integrado con el servidor de producción, antes de entrar plenamente a esta última área.

1

Development

Su área local

(PC, Laptop)

2

Existe un cuarto ambiente general, que es un servidor de prueba con reglas de “Production”. Se le conoce como staging y se utiliza para verificar que no cuente con errores.
Puede mezclarse con Testing posteriormente para hacer una confirmación eficiente de que “nada se rompe” en los despliegues.

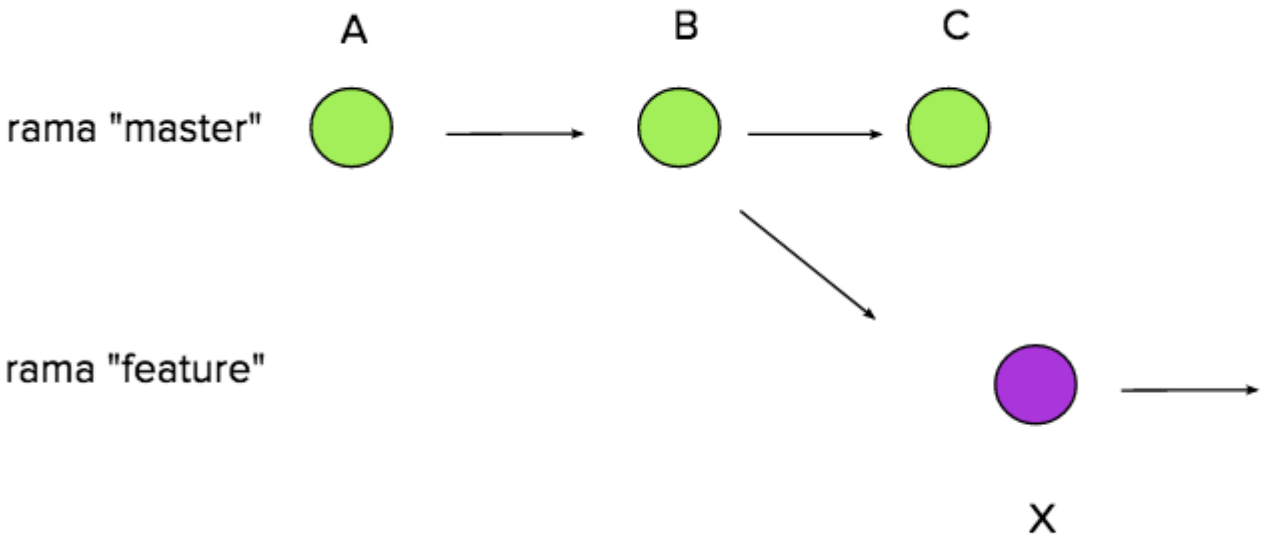
Branch (Rama)

Una rama es una línea del tiempo independiente al desarrollo principal.

Generan una abstracción para cada proceso de edición/stage/commit.

Piénsalo como una forma de generar una nueva área de working directory, staging area e historia del proyecto.

Los nuevos “commits” son grabados en la historia de la rama actual, que resultan en un “fork” en la historia del proyecto.



Deployment (Despliegue)

Son todas las actividades que hacen que un proyecto de software esté disponible para su uso, por parte de muchos usuarios.

Usualmente, también se le conoce a la ejecución del proyecto, del área de “Development” al área de “Production”, el cual se considera que el mismo ya está disponible “online” y cualquier persona puede acceder a él.

1

Development

Su área local

(PC, Laptop)

Fork

Fork es una acción que se utiliza en “GitHub” para hacer una copia exacta de un repositorio ajeno, al nuestro.

Regularmente, se encuentra en el borde derecho del repositorio, en GitHub.

Al momento de presionarlo, tendrás una copia exacta de ese repositorio, en tu cuenta de GitHub.

Al ser una copia, puedes hacer lo que quieras con él. Adaptarlo a tus necesidades ó mejorarlo.

Detalle importante, el repositorio nuevo, creado en nuestra cuenta (forked repository), NO SE ACTUALIZA automáticamente.

Al hacer la copia exacta, se queda con el último commit que el repositorio original tenía en ese momento.

Si quieres actualizarlo, deberás conectar ambos repositorios a tu área local y posteriormente hacer el “pull” del original y el “push” hacia el “forked repository” correspondiente.

Gitflow Workflow

Es un estricto modelo de diseño de ramas, para el desarrollo de un proyecto.

Es considerado un flujo de trabajo avanzado, basado en buenas prácticas.

Se dividen las áreas de ramas en 3 tipos:

a) Bugs

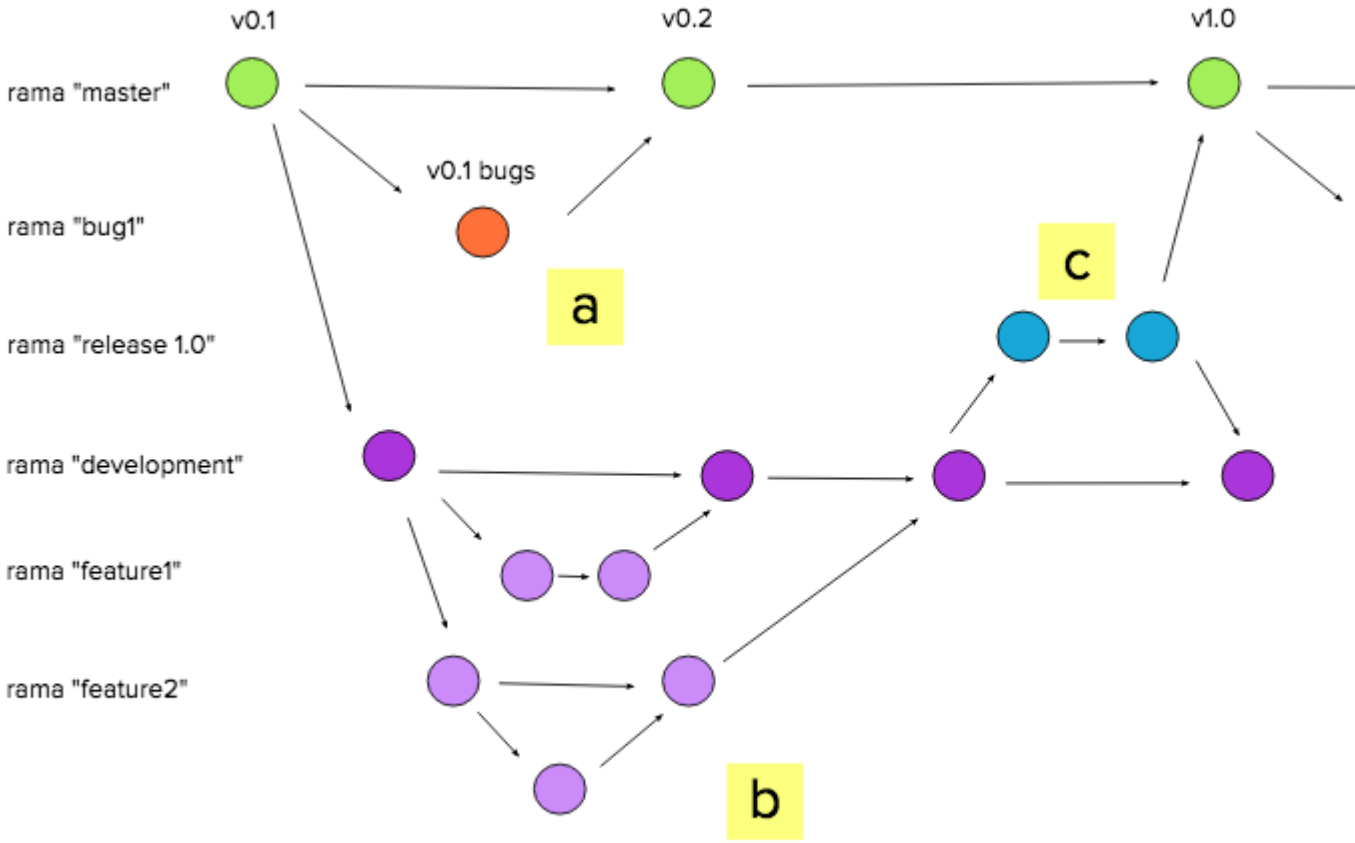
b) Features,

c) Releases.

Su estructura estricta de ramas permite trabajar proyectos masivamente grandes.

Puede funcionar bajo un líder de proyecto, ó con alta comunicación en el equipo.

El proceso es el siguiente.



a) Bugs. Ramas de mantenimiento.

Cuando un bug es encontrado por un usuario, se crea un Issue en el repositorio de GitHub y se crea una rama.

La rama puede llamarse con el nombre del issue creado.

Por ejemplo: "issue001".

Es la única rama que puede subir directamente a la rama master.

b) Features. Rama de Development.

Para esta sección, se dividen en 2 tipos, la rama principal llamada “development”, el cual va a recibir todas las conclusiones de las siguientes ramas derivadas, las cuales son el segundo tipo, llamadas “features”.

Las ramas “features” se derivan de la rama de “development” y las puedes nombrar de acuerdo al desarrollo que vas a realizar.

Una vez que estén listas, se fusionan directamente con la rama “development”, el cual va acumulando todos los features confirmados.

c) Releases. Rama de lanzamiento

Conforme vayas teniendo “features” confirmados en la rama de “development”, podrás crear una rama nueva llamada “releases”.

Esta rama funciona para prepara los lanzamientos. Se pueden revisar ante “testing” y por auditoría de otros miembros del equipo.

En caso de que haya modificaciones por parte de este equipo, se trabaja sobre la misma rama de "releases".

Una vez que tengan listo las modificaciones, ellos pueden subir a “master”. Regularmente ellos son los líderes del proyecto.

Como dato importante, no sólo hacen un lanzamiento hacia la rama “master”, si no también hacia la rama de “development”, esto con la intención de que los features restantes no tengan problemas con la actualización que hizo el equipo de “releases”.

Finalmente, en el área d, sólo denotamos que cuando se genera un release, se van generando “tags” con el número del release realizado, v1.0.

La denominación de cada tag depende del tipo de organización que tenga el equipo.

GUI’s (Graphical User Interfaces)

Referencia: https://git-scm.com/downloads/guis

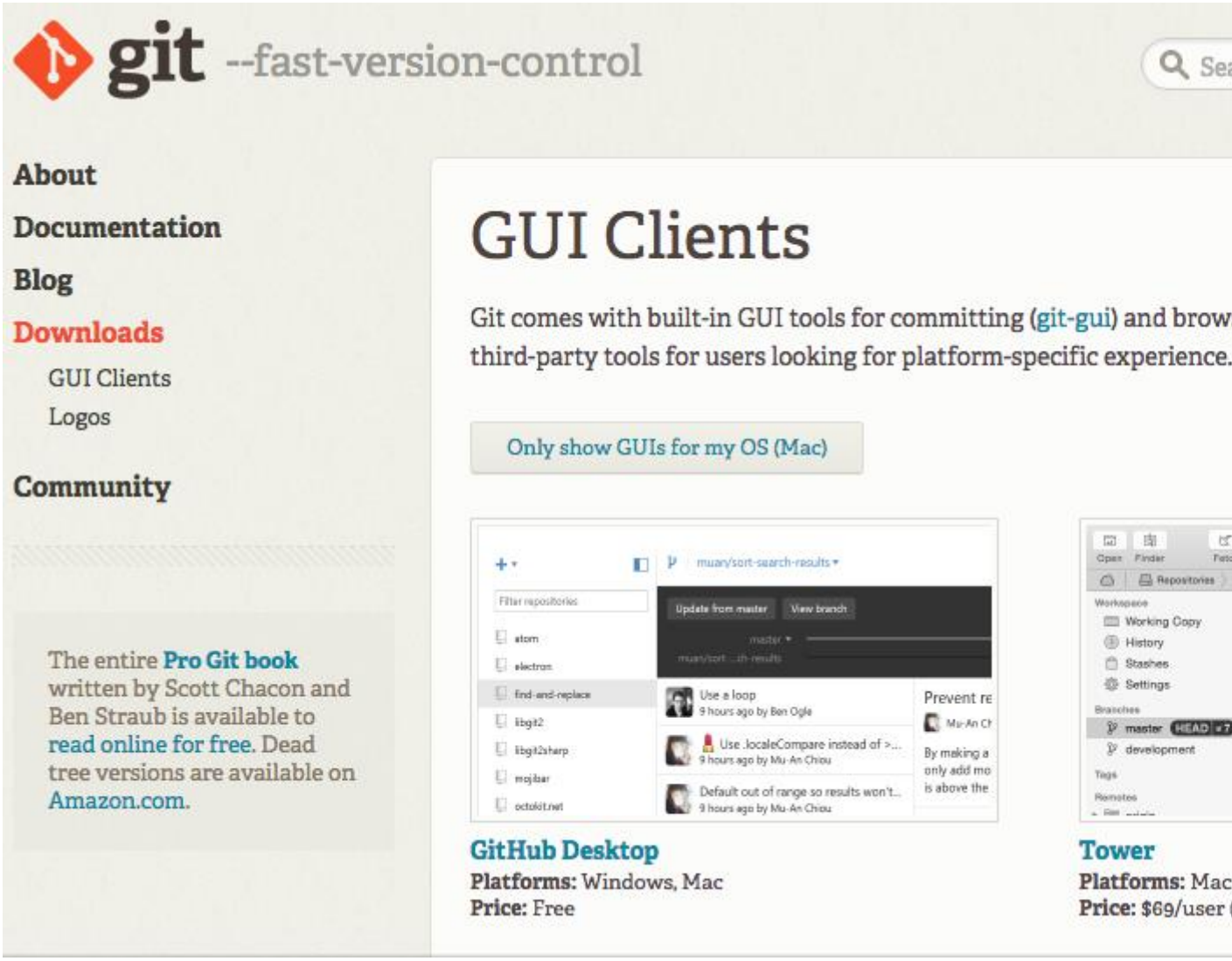
Los Graphical User Interfaces son clientes, programas, que puedes instalar en tu sistema operativo y sintetizan la manera en cómo puedes interactuar con GIT, de manera gráfica.

Tienen la ventaja de que son más cómodos para trabajar con GIT y ver cómo el proyecto está evolucionando, autores, commits, ramas, tiempos de desarrollo.

Sólo hay un inconveniente, el cual es que debes dominar GIT en terminal primero, antes de usarlos, como una recomendación dura.

Esto debido a que habrá momentos donde el GUI elegido no sepa resolver ciertos problemas, como fusiones, y él mismo te obligue a resolverlo vía terminal.

Por ello, aprende GIT y luego ten la libertad de descargar el que más te guste.



HEAD

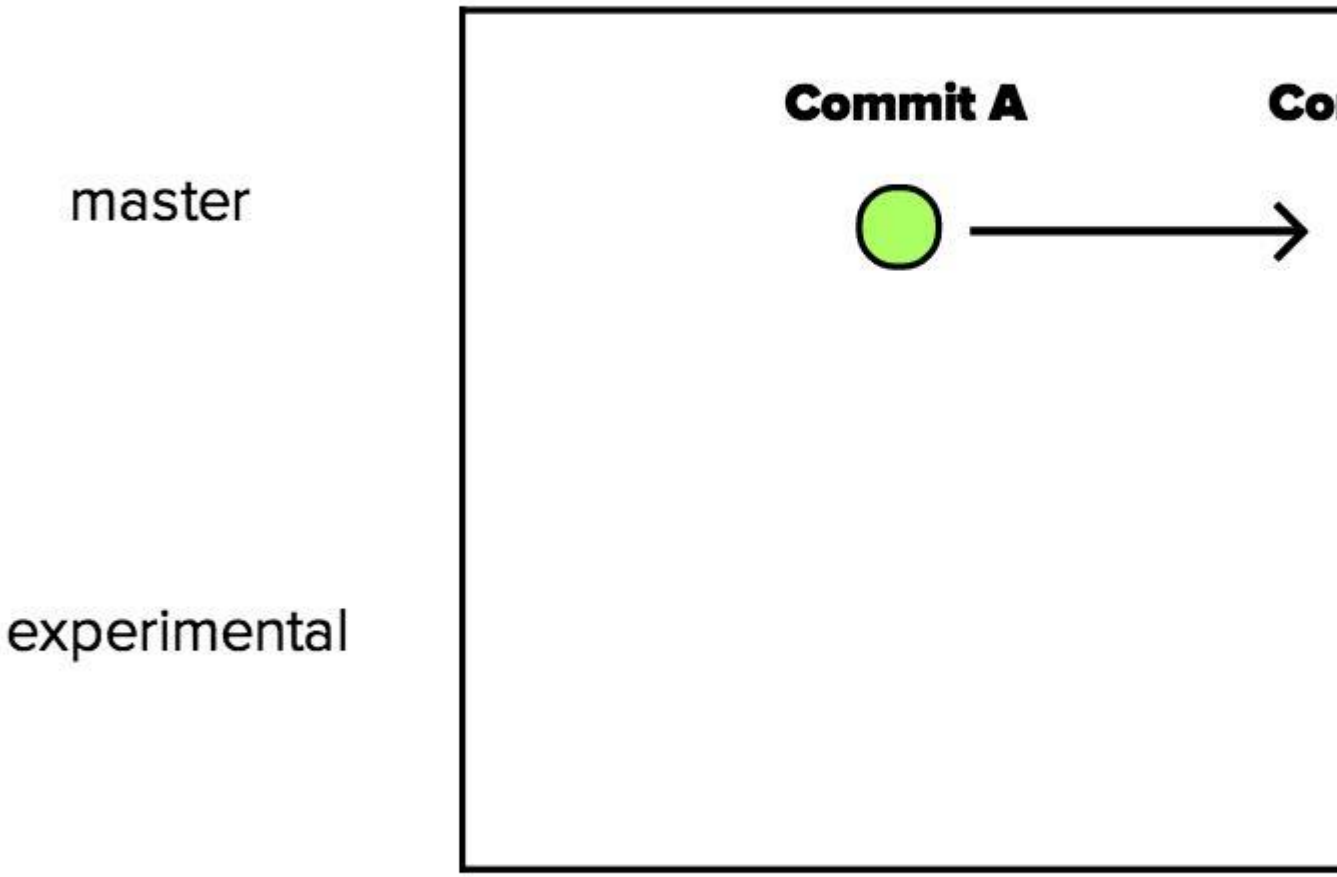
Es la referencia principal de GIT para identificar en qué commit se encuentra. Puedes pensarlo como si fuera una flecha apuntando, para situarte dentro del repositorio.

Internamente, git checkout se puede utilizar para actualizar el HEAD hacia un punto espefícifico (branch ó commit)

Por ejemplo, si ejecutamos:

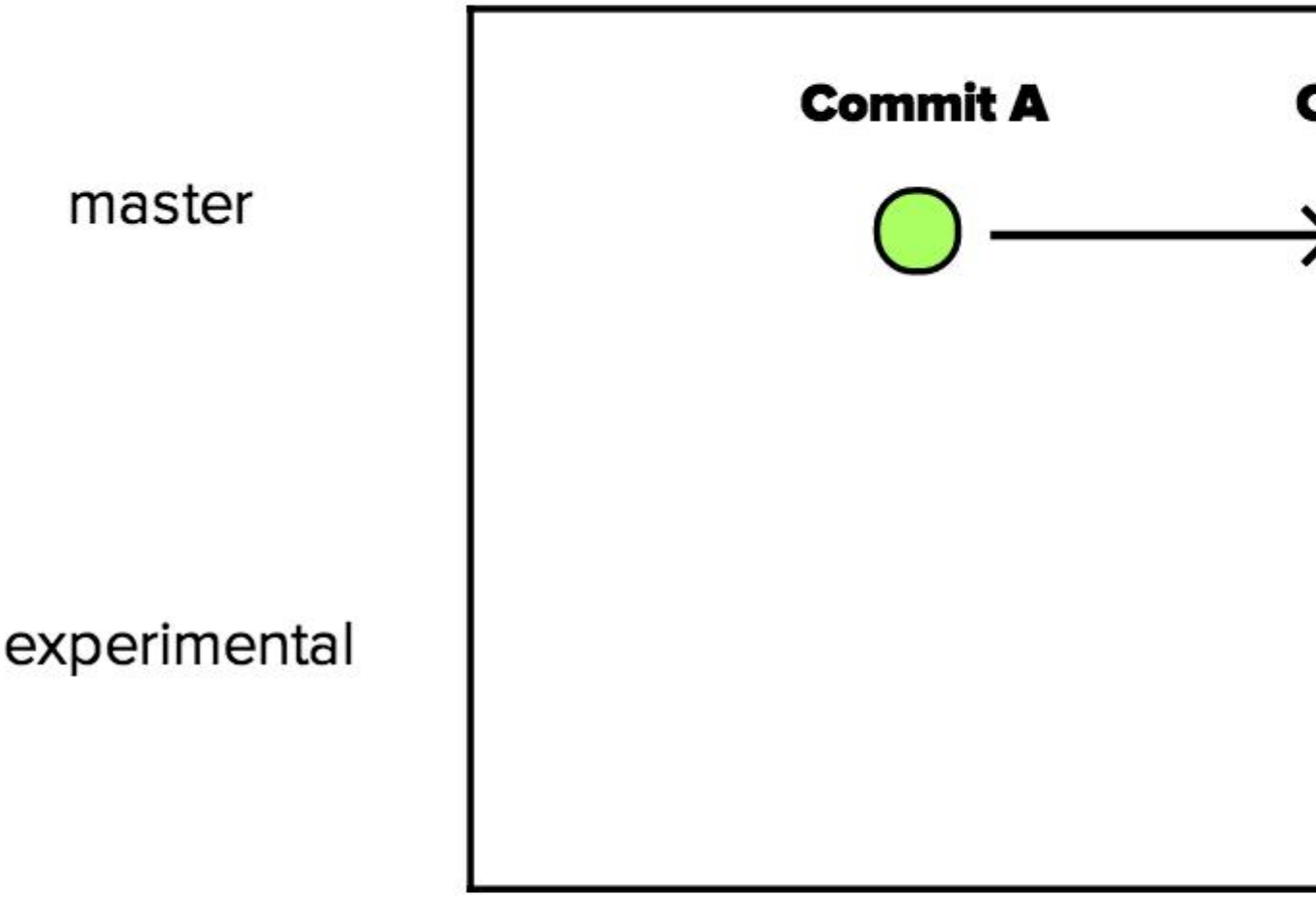
git checkout experimental

… HEAD se mueve hacia el último commit de experimental



Si ejecutamos:

git checkout master


Commit A
master
experimental

… HEAD se moverá al último commit de master.

Si quisieras moverte hacia el commit B, deberás ejecutar:

git checkout B

… el cual B representa el COMMIT ID, el número largo bajo el formato de SHA-1.

Hook

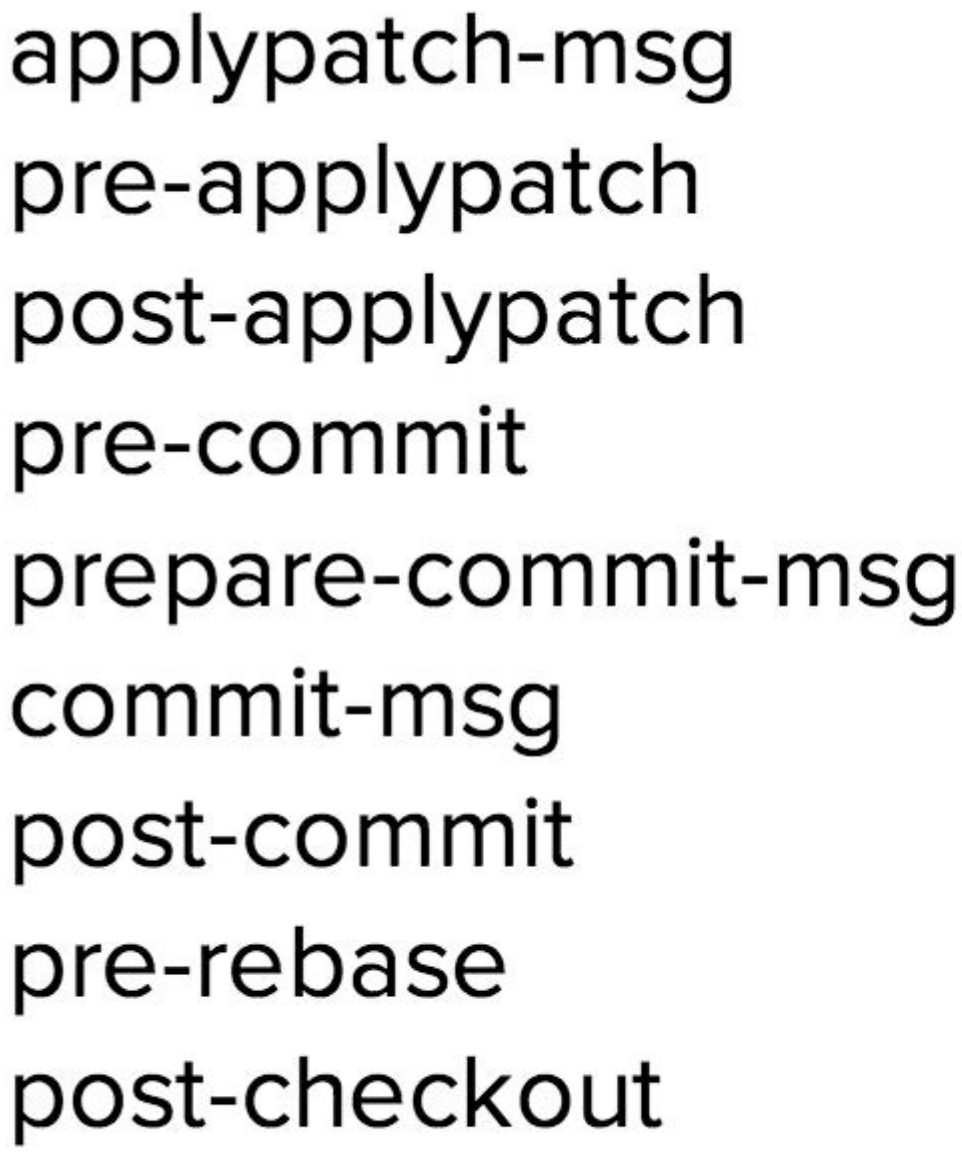
Es un script que corre automáticamente cada vez que sucede un evento particular en un repositorio.

Los “Hooks” te permiten personalizar el comportamiento interno de GIT y ejecutar acciones codificadas por ti en ciertos puntos del ciclo de vida del desarrollo.

Para entrar a ellos, los puedes localizar dentro de tu carpeta de trabajo, entrando en:

cd .git/hooks

Existen cerca de 17 hooks:



Cada uno se activa conforme al tipo de acción que realices en terminal.

Si activas y automatizas, por ejemplo, post-commit, y dentro de este archivo lo llenas con:

git status

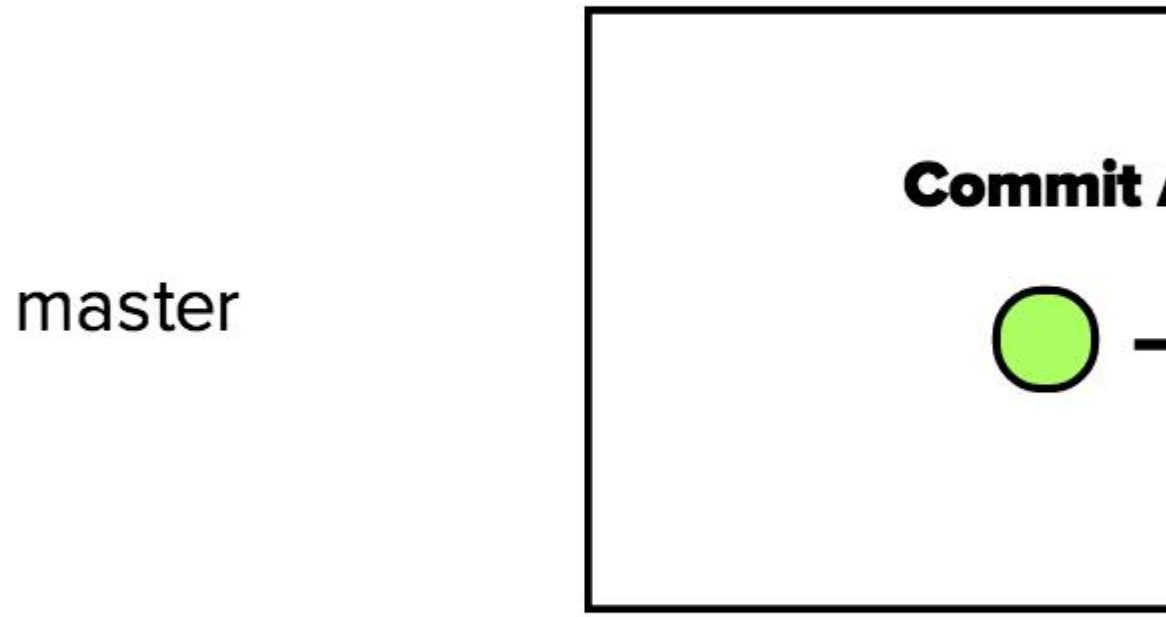
git log

Entonces, después de crear el commit, automáticamente correrá esos comandos que están dentro de ese archivo.

Puedes conocer cada una de las acciones en:
https://github.com/git/git/blob/master/Documentation/githooks.txt

Master

La rama principal de desarrollo. Cada vez que vayas a crear un repositorio de GIT, una rama llamada “master” es creada, y se vuelve la rama activa por defecto.



Pull Request

Es un “feature” que permite a los desarrolladores colaborar fácilmente en comunidades como GitHub ó Bitbucket.

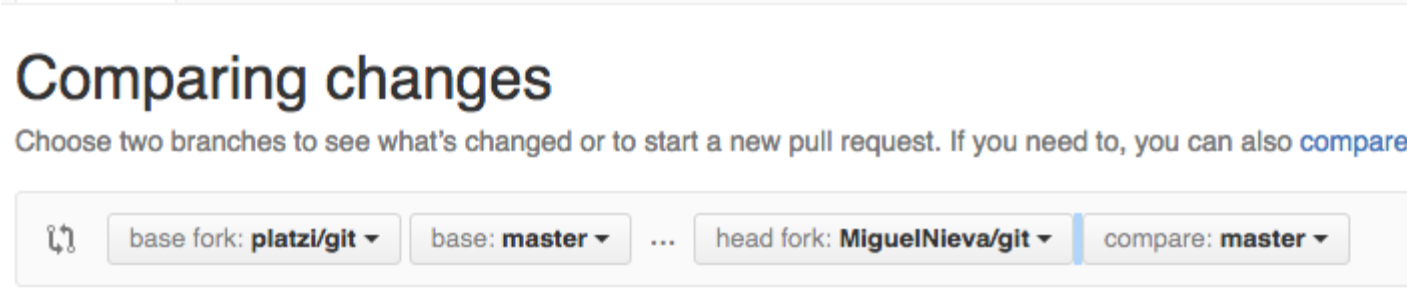
El ciclo completo de colaboración para un Pull Request funciona de esta manera:

1. Realizar un fork de un repositorio ajeno, a tu cuenta de GitHub.
2. Conectar tu área local al repositorio original (principal) bajo el nombre remoto “upstream” (puede ser cualquier nombre, pero es por buenas prácticas) y al repositorio “forked”bajo el nombre “origin”.
3. Realiza “fetch” constantes hacia tus ramas “espejo”. Éstas son ramas que se encuentran escondidas y separadas de tu área de trabajo con el objetivo de ser el

“reflejo” de los repositorios remotos, descargados a tu área local. En la mayor medida posible, no los alteres. Los puedes identificar bajo el nombre: “[nombre del remoto]/[nombre de la rama]”. Por ejemplo, "origin/master ó "upstream/master"

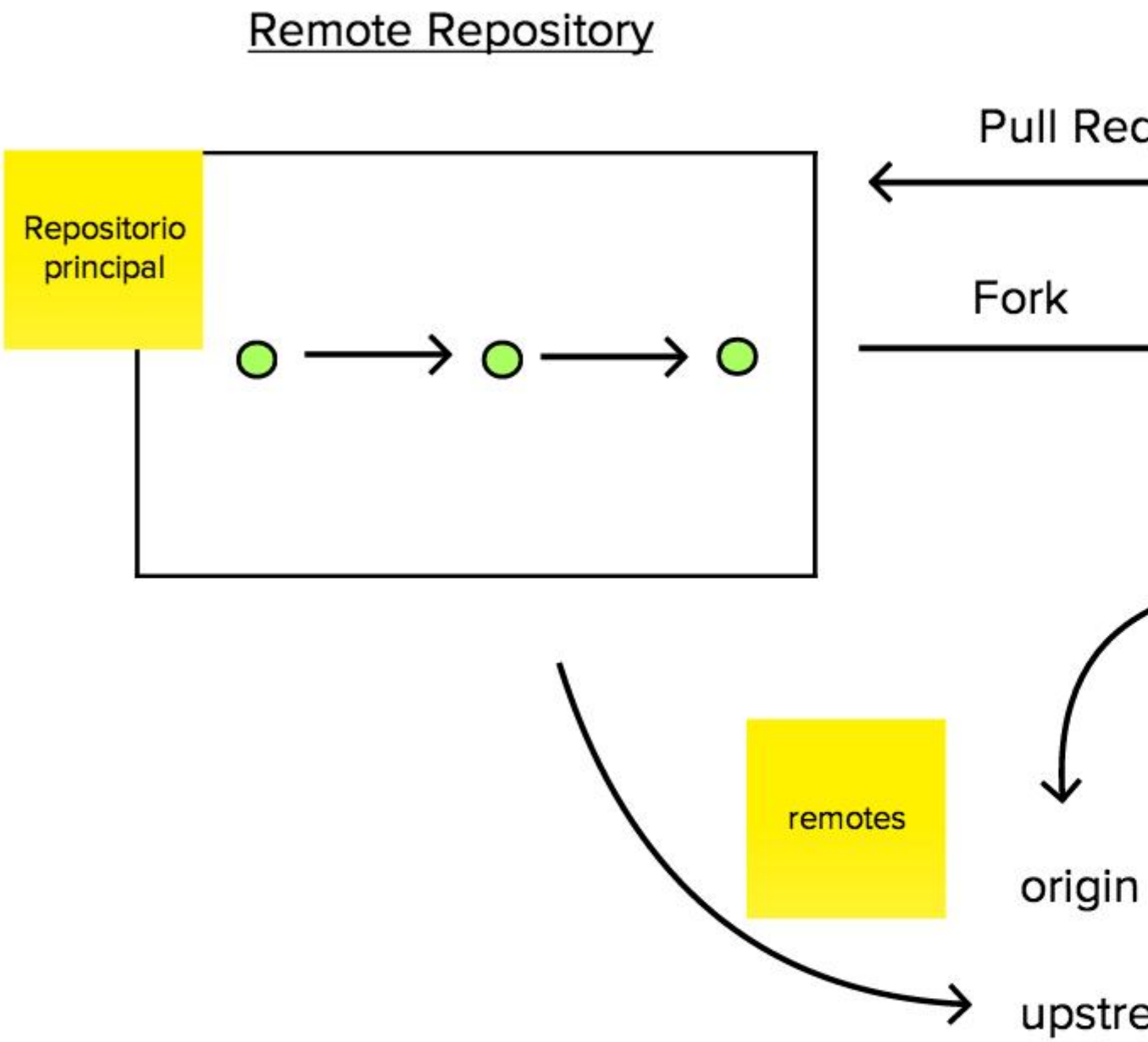
4. Haz un push al repositorio “forked”, el repositorio copia en tu cuenta de GitHub, a partir de los cambios que hagas.
5. Cuando estos cambios estén en el repositorio “forked”, puedes encontrar un botón en tu perfil de GitHub llamado “Pull Request”.

Finalmente, verás un área donde se comparan ambos repositorios (el original con el “forked”) para fusionarse en el original.



El dueño, project manager ó líder del repositorio original será el único que podrá aceptar los cambios ó desecharlos.

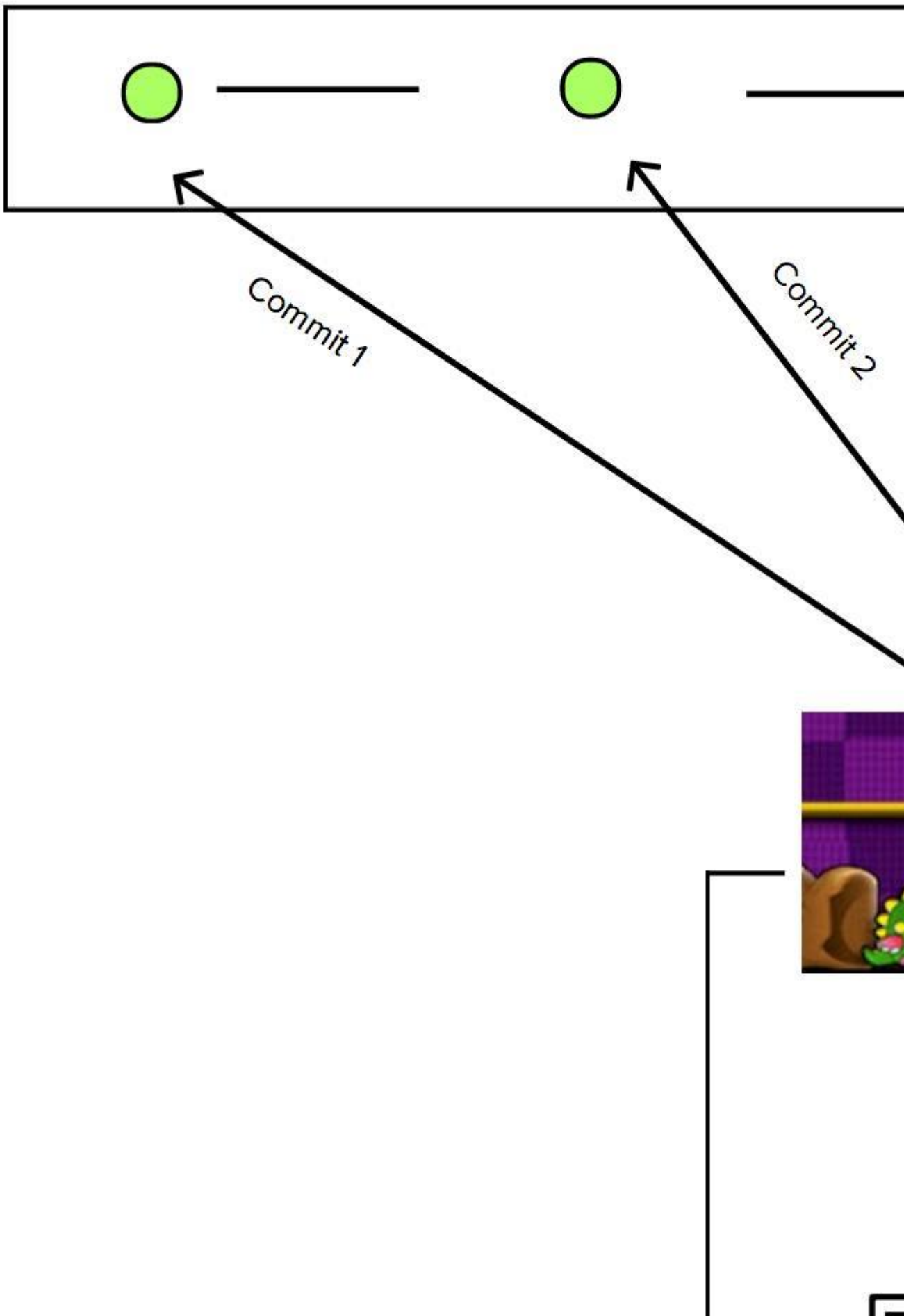
A este proceso se le denomina colaboración web. Lo puedes visualizar completamente en el diagrama inferior:


Remote Repository
Pull Rec
Repositorio principal
Fork
remotes
origin
upstre

Repository

Una colección de commits, “tags” para identificarlos y, ramas.

Es el último paso de la arquitectura de árbol, de GIT.


Commit 1
Commit 2

Sistema de Control de Versiones

Un sistema que graba cambios a un archivo ó a un conjunto de archivos sobre el tiempo, en la cual puedes revisar específicas versiones más tarde.

Se utiliza también como herramienta para la colaboración entre diferentes profesionales web.

Existen varios, como:

- Subversion
- Mercurial
- Git

El que se utiliza en este curso es el último de ellos. El más popular y usado con respecto a tendencias de búsqueda.

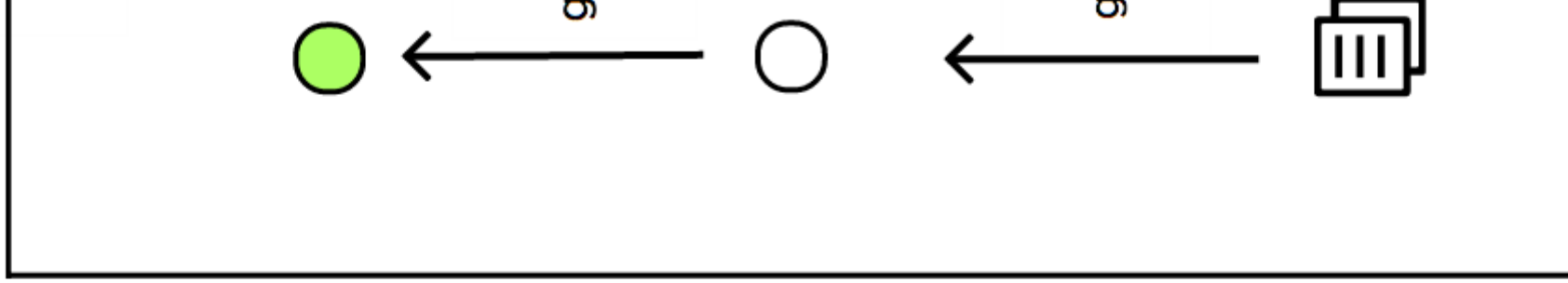
Staging Area

Es el área de preparación, antes de que el conjunto de cambios suban al repositorio y se vuelva “commit"

Regularmente, en comparación con otros sistemas de control de versiones, el área existe para poder agrupar correctamente los cambios.

Imagina que quieres realizar un commit, e hiciste varios cambios en el proyecto, pero sólo quieres hacer un commit de ciertos archivos. Es aquí donde el staging area tiene sentido.

Elige los que quieras que suban al repositorio y prepáralos en esta área.



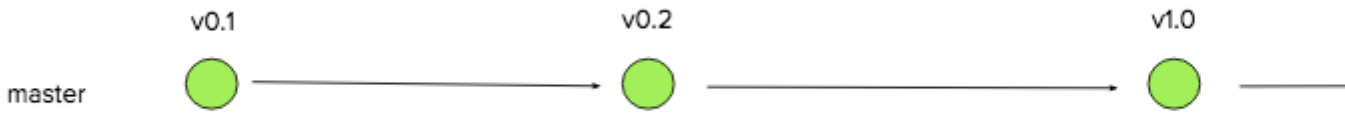
Tag

Una referencia típicamente usada para marcar un punto particular en una cadena de commits.

En contraste con HEAD, un tag no es actualizado por el comando de commit. Se debe de agregar con el comando de tag.

git tag -a v1.4 -m `"my version 1.4"`

El "-a” coloca la versión del tag.El “-m" te permite ponerle una descripción sobre lo que trata ese tag.



Si colocas:

git tag

Te muestra la lista de los tags que tienes.

Si colocas:

git show [la versión del tag]git show v1.0

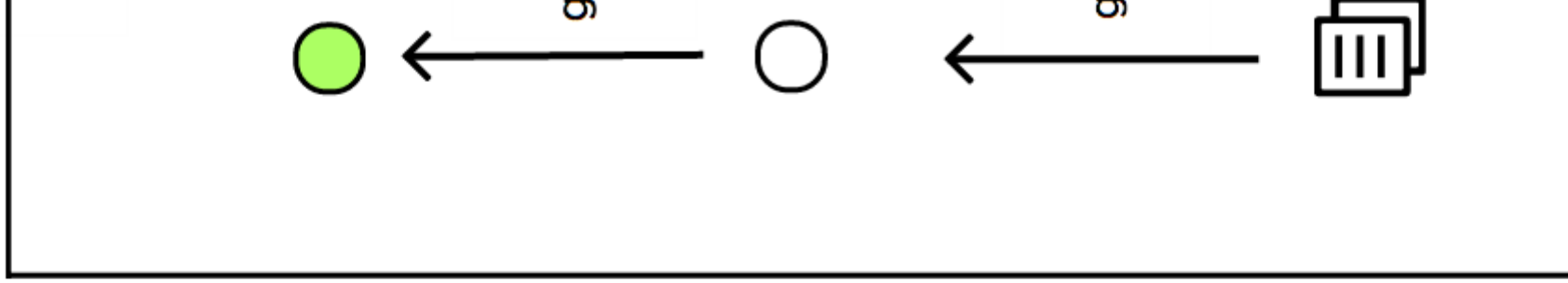
Entonces obtendrás una descripción del tag:

```
tag v1.0
Tagger: Miguel Nieva <m@platzi.com>
Date:   Sat May 18 21:12:13 2016 -0700
my version 1.4
commit ca82a6dff817ec66f44342007202690a93763949
Author: Miguel Nieva <m@platzi.com>Date:   Mon Mar 10 14:20:12
2016 -0700
    The nav tab is added.
```

Working Tree ó Working Directory

Un área en el cual contiene los cambios en “local” pero que no se ha realizado ningún tipo de “guardado” en el área de staging “preparación” ni en el repositorio.

Git tiene identificado los cambios que haces, pero no hace nada con ellos, hasta que tú se lo indiques.



COMANDOS

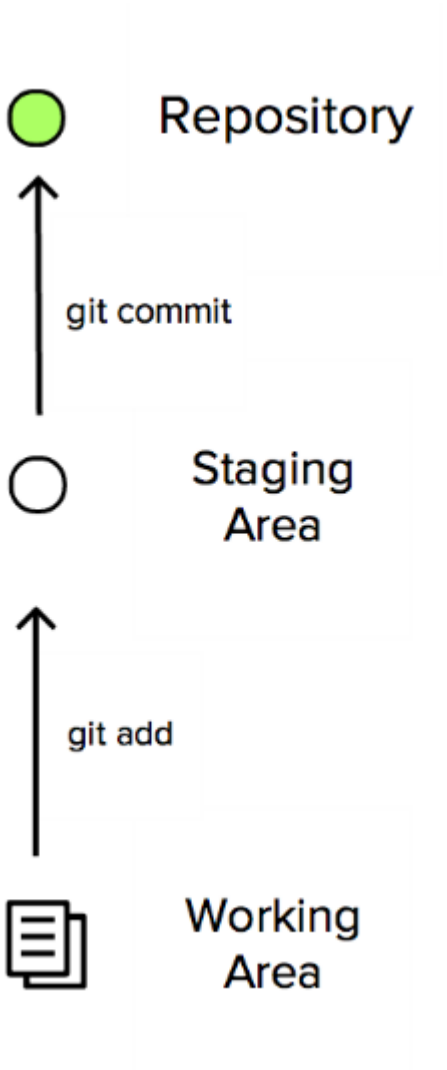
git add

Mueve todos los cambios del “Working Directory” al “Staging Area”.

Esto da la oportunidad de preparar los archivos antes de realizar la confirmación con el “commit” a la historia principal del proyecto.

El comando es:

git add [nombre del archivo] -> Agrega el cambio ó creación del archivo de manera individual.
git add . -> Agrega los cambios de todos los archivos, pero no los nuevos creados ó nuevos eliminados.git add -A -> Agrega los cambios los archivos, incluidos nuevos creados ó nuevos eliminados.



git alias (junto a config)

Git alias es un comando que te permite crear tus propios comandos de GIT, a partir de otros, con sus parámetros.

Está vinculado con el comando “config” y la fórmula es:

git config --global alias.[nombre del alias que quieres crear] ‘[comandos con parámetros que quieres que se ejecuten con el alias]

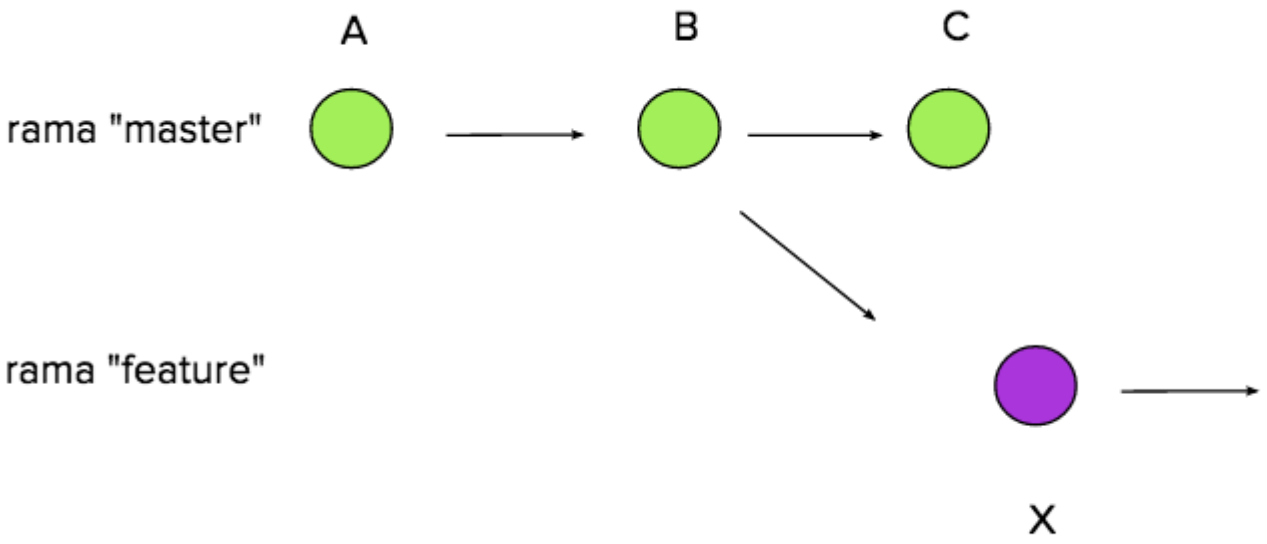
Un ejemplo, para la síntesis de un “git log” con ciertos parámetros que necesitas y no lo quieres escribir a cada momento, es:

git config --global alias.nicelog 'log --oneline --graph --all'

git branch

Este comando es tu administrador general de ramas.

Te permite crear diferentes ramas de desarrollo, dentro de un repositorio particular.
El comando a utilizar es:
1. git branch [nombre de la rama a crear]2. * git checkout [nombre de la rama a crear].
* Recuerda que una vez creada, debes cambiarte hacia ella.
Si quieres crear la rama y cambiarte automáticamente, puedes usar:
git checkout -b [nombre de la rama a crear]

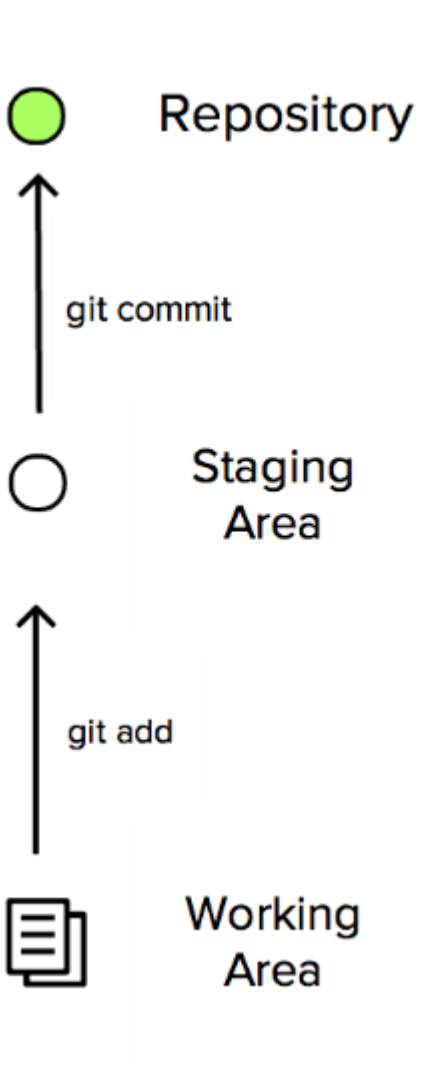


git checkout
En adición a poder moverte entre commits y viejos archivos para revisión, git checkout también te permite navegar entre las diferentes ramas existentes.
Combinado con los otros comandos básicos de GIT, es una forma de trabajar una particular línea de desarrollo.
git checkout [rama]git checkout [Commit ID]

git clone
Crea una copia de un repositorio existente de GIT.
Clonar es el camino más común para que los desarrolladores obtengan una copia

del proyecto, del repositorio central. Regularmente va hacia local
git clone [dirección del repositorio, puede ser https;//…git, ó, ssh:...]

git commit
Inserta el conjunto de archivos que se encuentra en el "Staging Area” y los colocan en el repositorio.
Cada “set” de archivos insertados en la historia del proyecto se la llama “commit”.
Combinado con git add, ese define el proceso básico de GIT.
git commit -m [nombre del título del commit]



git commit --amend

Incluir --amend corrige el más reciente commit.
Este es muy útil cuando olvidas preparar un archivo en el Staging Area y es importante tenerlo en el commit.

git commit --amend

git config

Configura todas las opciones que puedes hacer con GIT.

git fetch

Te permite descargar una rama de otro repositorio, con todos sus commits y archivos. Pero, no busca integrar nada en tu repositorio local.

Esto da la oportunidad de inspeccionar cambios antes de fusionarlos con tu proyecto.

git init

Inicia un nuevo repositorio de Git. Esto permite que Git empiece a rastrear al repositorio con todos los cambios que hagas dentro.

Es lo primero que tienes que hacer antes de empezar a trabajar.

Te sitúas en la carpeta de trabajo que vas a utilizar Git y ejecutas:

git init

git log

Te ayuda a explorar las previas revisiones de un proyecto.

Provee diferentes opciones de formato para mostrar commits.

git log [parámetros]

git merge

Fusión. Integra cambios de diferentes ramas independientes, en una sola.

git checkout [la rama base que quieres que sea “el que absorbe"]git merge [la otra rama que quieres "que sea absorbida"]

git pull

Es la versión automática de git fetch. Descarga la rama desde un repositorio remoto y luego, inmediatamente lo fusiona con la rama actual.

git fetch + git merge = git pull

git push

“Pushing” es el opuesto a “fetching”.

Te permite mover una rama local a otro repositorio, que usualmente es la forma de publicar contribuciones, en un servidor remoto.

Se pueden enviar muchos commits al mismo tiempo, no sólo uno.

git push [nombre del remoto] [nombre de la rama]

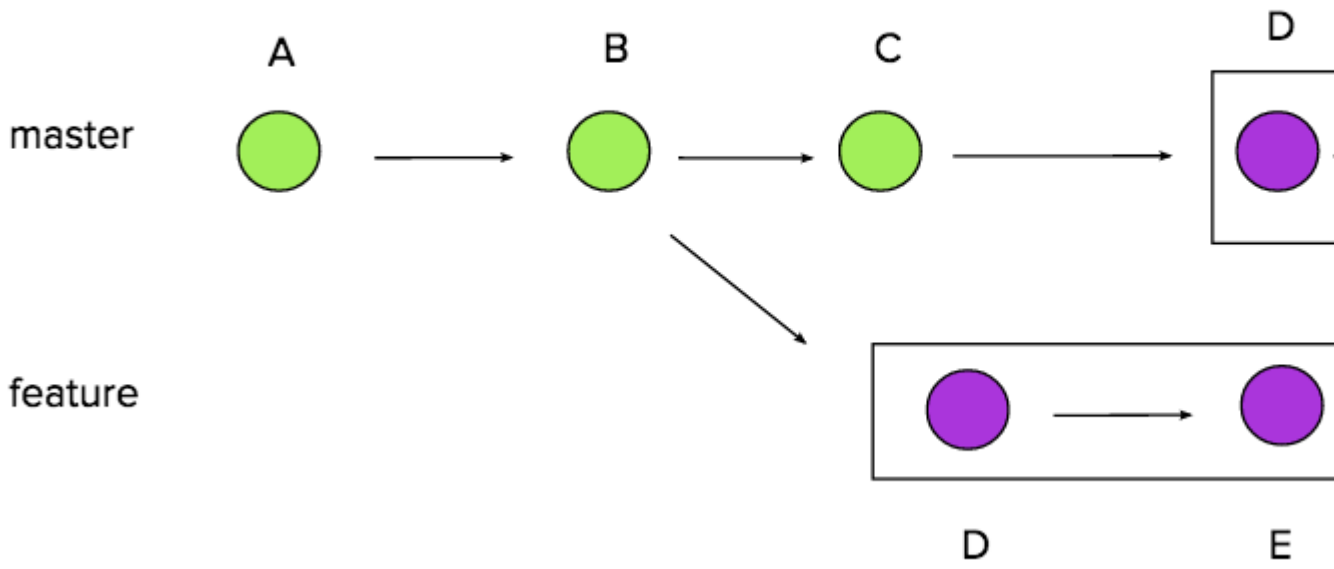
git rebase

Te permite mover tus ramas hacia adelante, en lugar de fusionarlas. Esto ayuda a que no hagas fusiones innecesarias.

Cuando necesitas una rama más completa y detallada, entonces se utiliza rebase para lograrlo.

Por ejemplo, en este caso, la rama “feature”, en lugar de fusionar, colocaremos toda la rama completa enfrente de master.

Con esto, en lugar de haber hecho una fusión en un solo commit, en caso de que se necesite un mayor análisis de la rama, dejamos todo lo que se hizo en la rama feature y la colocamos adelante de nuestra rama master (principal).



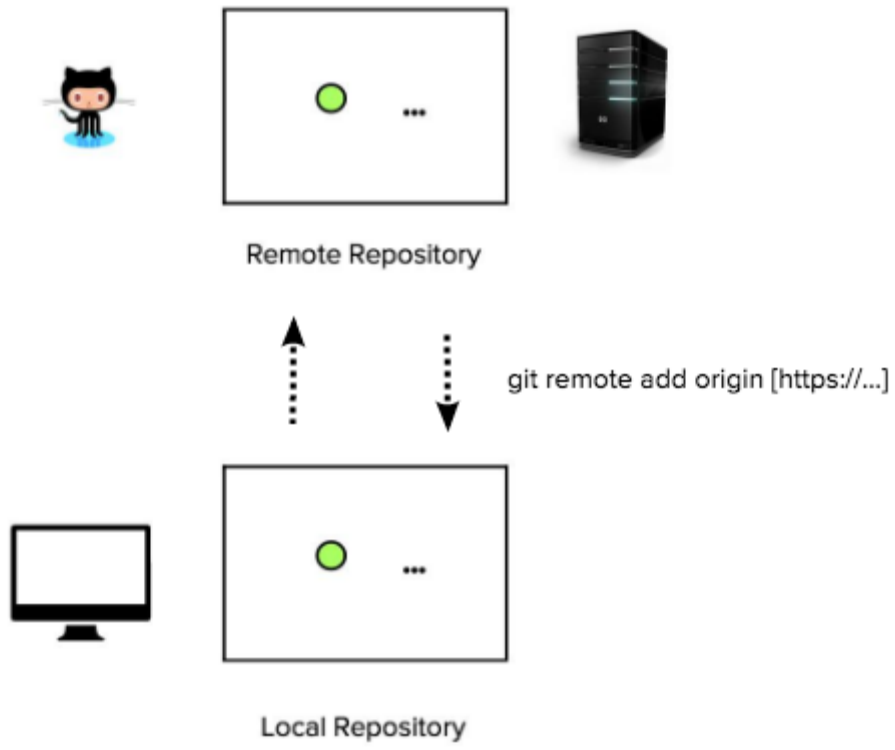
git remote

Es un comando que te permite conectar un repositorio locales y remotos.

Un administrador de conexiones, en definición sencilla.

En lugar de poner constantemente la URL para realizar los comandos de “fetch”, “pull”, y “push”, sólo le asignamos un nombre y podemos llamar esa conexión de manera rápida.

git remote add [“nombre del remoto"]* Por defecto, se llama “origin”.



---

git reset

Deshace cambios a archivos en el “Working Directory”.

Hacer un reset te permite limpiar ó completamente eliminar cambios que no han sido publicados al repositorio público.

git revert

Deshace un commit, colocando uno extra adelante de la rama, quitando los cambios del commit elegido.

El proceso es:

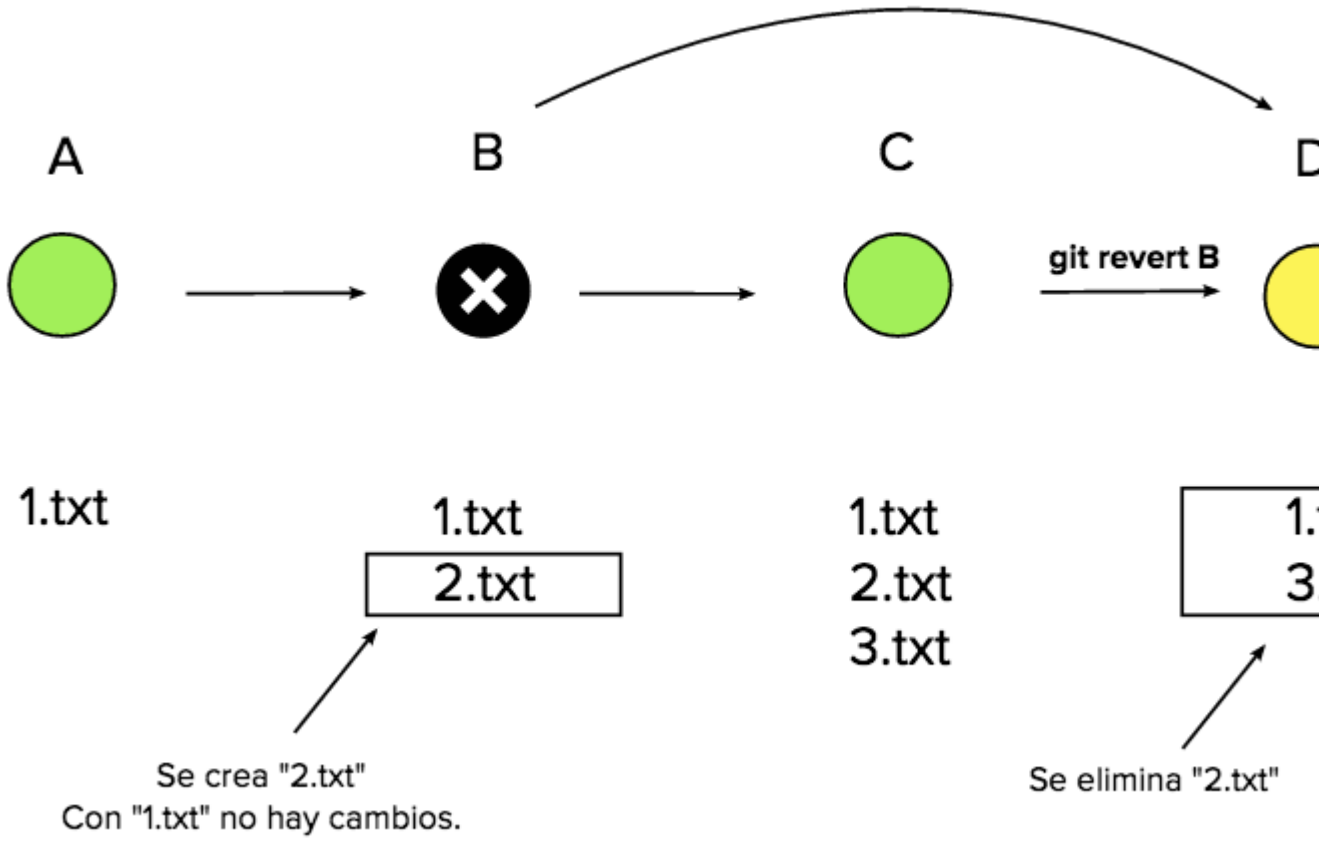
1. Toma el commit ID que quieres eliminar,
2. Colócalo como:

```
git revert [commit ID]
```

Git lo identificará pero no lo borrará, si no, más bien, verá qué cambios se ejecutaron en ese commit puntualmente y creará uno nuevo, adelante del último, revirtiendo todas las acciones de ese commit.

¿Por qué no borra el commit que elegimos?

Porque alteraría la historia del proyecto. Y, aunque haya un comió que esté mal creado, si se borra, alteraría toda la historia. La solución es conocer qué tiene ese commit y revertirlo con un commit nuevo.



git status

Muestra el estado del “Working Directory” y cada commit subido al repositorio.

```
git status
```

HERRAMIENTAS

git extras

Extensión, hecha por la comunidad, de los comandos de Git.

Recomendado para automatizar y agilizar procesos.

https://github.com/tj/git-extras

GitLab
Es una plataforma privada, con una interfaz similar a GitHub, que se instala dentro de tu servidor.

Zenhub.io

Es una herramienta que aprovecha el project management de GitHub para expandir funcionalidades.

Incluye trabajo con SCRUM, sprints y una mejor organización con respecto a Issues y un concepto llamado “Epics”.