**Configurar Git**

Una vez GitHub Desktop (y Git) estén instalados, abre tu **terminal/consola**. Puedes verificar si está disponible escribiendo:

git --version

Esto retornará la versión de Git en tu computadora y se verá similar a lo siguiente:

git version 1.9.1

Lo siguiente, configurar Git de manera que pueda asociarte los cambios que hagas:

Establece tu nombre:

git config --global user.name "Tu nombre"

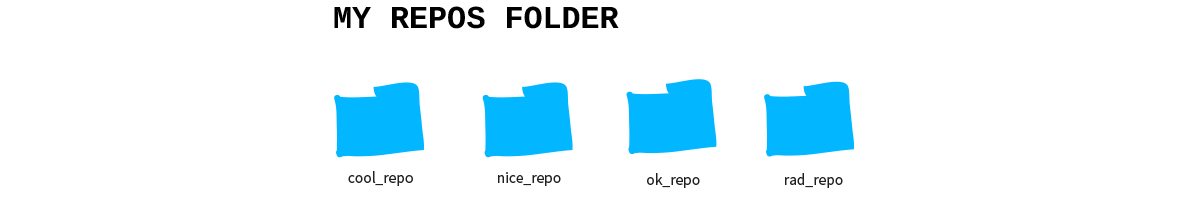
Establece tu email:

git config --global user.email "tuemail@example.com"

Has terminado tu primer reto! Haz click en el botón 'Verificar' para mostrarte el resultado.

## Repositorios

Un **repositorio** es una colección de elementos relacionados. En nuestro caso, cuándo estamos escribiendo software, es una colección de archivos relacionados a un proyecto de software. Puedes imaginarlo cómo la carpeta de un proyecto con todos los archivos relevantes en us interior. De hecho, es cómo se verá en tu computadora de todas maneras. Algunas veces son llamados "repos" cómo abreviación.



Le dices a Git lo que es tu proyect y Git empezará a hacerle seguimiento a todos los cambios en esa carpeta. Esto es lo que conforma repositorio de Git: una carpeta de elementos que son seguidos por Git. Git hace seguimiento cuándo se añaden archivos, cuándo se extraen incluso cuándo una simple letra en un simple archivo ha sido cambiada. A todo esto Git le añade quién y cuándo lo hizo. En software, al proceso de hacer seguimiento a los cambios se le llama **control de versiones**.

## Usando la Terminal(consola)

**Terminal** (o Bash, o consola o línea de comandos) es la manera de decirle a tu computadora que haga cosas escribiendo comandos en vez de dar click en las aplicaciones con el ratón (mouse). Puedes renombrar archivos, abrir archivos, crear nuevas carpetas, moverse entre directorios (carpetas) y mucho más sólo con escribir comandos. Puedes incluso usar un editor de texto para código (como [Vim](http://en.wikipedia.org/wiki/Vim_(text_editor)" \t "_blank)) en tu terminal y y nunca dejar de usarla!

Además de navegar en tu computadora, puedes usar programas en la Terminal que tienen **Interfaz de línea de comandos (command-line interface CLI)**, esto significa que pueden funcionar con comandos desde la terminal. Git es uno de estos. La primera parte del comando se le permiet a tu computadora que le estás hablando a Git. Las siguientes partes son comandos y diferentes opciones que quieres Git haga.

En Git-it aprenderás algunas acciones de línea de comandos básicas (además de los comandos de Git) para navegar tu computadora; ellos son descritos dentro de los pasos.

### Crear un Repositorio

Vas a crear un nuevo proyecto en una carpeta y entonces iniciarás un repositorio de Git. Llamaremos la carpeta 'hello-world' en este reto, pero puedes elegir el nombre que quieras si ya tienes una carpeta con este nombre (es un nombre común para crear proyectos de prueba).

Es tiempo de abrir tu terminal!

En la ventana de tu terminal, escribe estos comandos, uno a la vez, presionando enter después de cada uno.

Primero, crear una nueva carpeta:  
Ten en cuenta: mkdir hace referencia a make directory (crear directorio) del idioma inglés

mkdir hello-world

Ahora vamos a situarnos dentro de la carpeta:  
Ten en cuenta: cd hace referencia a change directory (cambiar directorio) del idioma inglés

cd hello-world

Finalmente, dile a Git que inicie (empiece el seguimiento) de la carpeta en la que estás situado:

git init

El último comando debe retornar algo cómo "Initialized empty Git repository". Los otros comandos no tienen retorno.

Lo lograste! Si quieres estar seguro que es repositorio de Git, escribe git status y si no retorna 'fatal: Not a git repository...', lo has logrado!

Tips

* **Crear nueva carpera (make directory)**

mkdir <NOMBREDELACARPETA>

* **Navegar o moverse hacia una carpeta existente (change directory)**

cd <NOMBREDELACARPETA>

* **Listar los contenidos de una carpeta**

ls

* **Iniciar Git en una carpeta**

git init

**COMMIT TO IT**

Crea un archivo en tu nuevo repositorio, añade algo al archivo y guarda (commit) el cambio en Git.

## Commits (guardar)

Los Commits son el centro de Git. Ellos son momentos en los cuales guardas y describes el trabajo que has hecho. Ellos son los puntos en la línea del tiempo del historial de tu proyecto.

### Crea un nuevo archivo

Ahora que tienes un repositorio iniciado agreguemosle un archivo.

Abre tu editor de texto y crea un nuevo archivo vacío. Ahora escribe algo de texto, por ejemplo "Hello!", y guarda el archivo como 'readme.txt' en la carpeta 'hello-world' que creaste en el anterior reto.

### Verificar Status (estado) + añadir cambios al Commit

Ahore revisemos el **status (estado)** de tu repositorio para verificar si han habido cambios. ¿Tú sabes que has cambiado algo, pero lo sabe Git?

Asegúrate de que aún estás situado en el directorio 'hello-world' cuándo ejecutes estos comandos. Usa Git para ver qué cambió en tu repositorio:

Primero, verifica el status (estado):

git status

Git debería decirte que un archivo ha sido añadido.

Entonces **añade (add)** el archivo que acabaste de crear que empiece a ser parte de los cambios que vas a **guardar (commit)** con Git:

git add readme.txt

Finalmente, **commit (guarda)** aquellos cambios en el historial del repositorio con un mensaje (m) corto que describe la actualización realizada.

git commit -m "readme creado"

### Siguiente paso: Crea más cambios

Ahora añade otra línea a 'readme.txt' y guarda el archivo nuevamente.

En la terminal, puedes ver las diferencias **diff** que hay entre el archivo ahora mismo y el archivo en el momento que creaste el último commit.

Dile a Git que te muestre las diferencias **diff**:

git diff

Ahora con lo que aprendiste previamente, crea un commit (guarda) el cambio más reciente.

* **Para verificar el estado de los cambios en un repositorio**
* git status
* **Ver diferencias entre las versiones (commits) de los archivos**
* git diff
* **Añadir un archivo para ser guardado (commit)**
* git add <NOMBREDELARCHIVO>
* **Para añadir todos los cambios**
* git add .
* **Para crear un commit (guardar) los cambios que añadiste con un mensaje corto que los describe**
* git commit -m "tu mensaje commit"

**GITHUBBIN**

Crea una cuenta de GitHub y agrega tu nombre de usuario a la configuración de Git.

## Trabaja mejor, Juntos

El repositorio que has creado solamente existe en tu computadora, que es útil, pero hace que sea bastante difícil compartir y trabajar con otros. No te preocupes, para eso está GitHub!

GitHub es un sitio web que permite a personas de cualquier lugar subir el código que tienen en Git para que cualquier otra persona pueda fácilmente trabajar en conjunto en el mismo proyecto. GitHub actúa cómo el repositorio central para que tú y otras personas compartan. Tú empujas (push) tus cambios a Github y puedes halar (pull) los cambios que han hecho otras personas.

### Crea una cuenta de GitHub

Visita [github.com](http://github.com) y registra una cuenta gratuita. **Choca esos cinco, bienvenido!**

### Agrega el nombre de usuario de GitHub a Git

Agraga tu nombre de usuario de GitHub a tú configuración local de Git. Haremos esto solo por el bien de Git-it; ya que hace más fácil verificar los próximos retos. Guarda exactamente el nombre de usuario que creaste en GitHub y **pon las mayúsculas donde las haya**. Nota, no necesitas ingresar los símbolos "<" y ">".

Añade tu nombre de usuario de GitHub configuración de Git:

git config --global user.username <USerNamE>

Puedes verificar lo que acabas de ingresar en tu configuración de Git escribiendo:

git config --global user.username

Asgurate de escribir "**user.username**" arriba y no "user.name", que puede sobre escribir lo que hiciste en el primer reto y dejarte sin configurar correctamente tu nombre de usuario (username)! Si te das cuenta que lo hiciste, está bien, sólo repite el paso del primer reto para agregar tu nombre y completar el paso anterior.

#### Si los nombres de usuario de GitHub y Git no coindiden en la configuración

Un error común es que el usuario de GitHub no coincida con las mayúsculas del que configuraste en git config. por ejemplo, 'JLord' no es lo mismo que 'jlord'

Para cambiar el nombre de usuario en Git, sólo ejecuta el comando que pusiste a correr antes, pero con las letras mayúsculas en el lugar correcto:

git config --global user.username <USerNamE>

Cuándo hayas hecho las correcciones, verifica nuevamente!

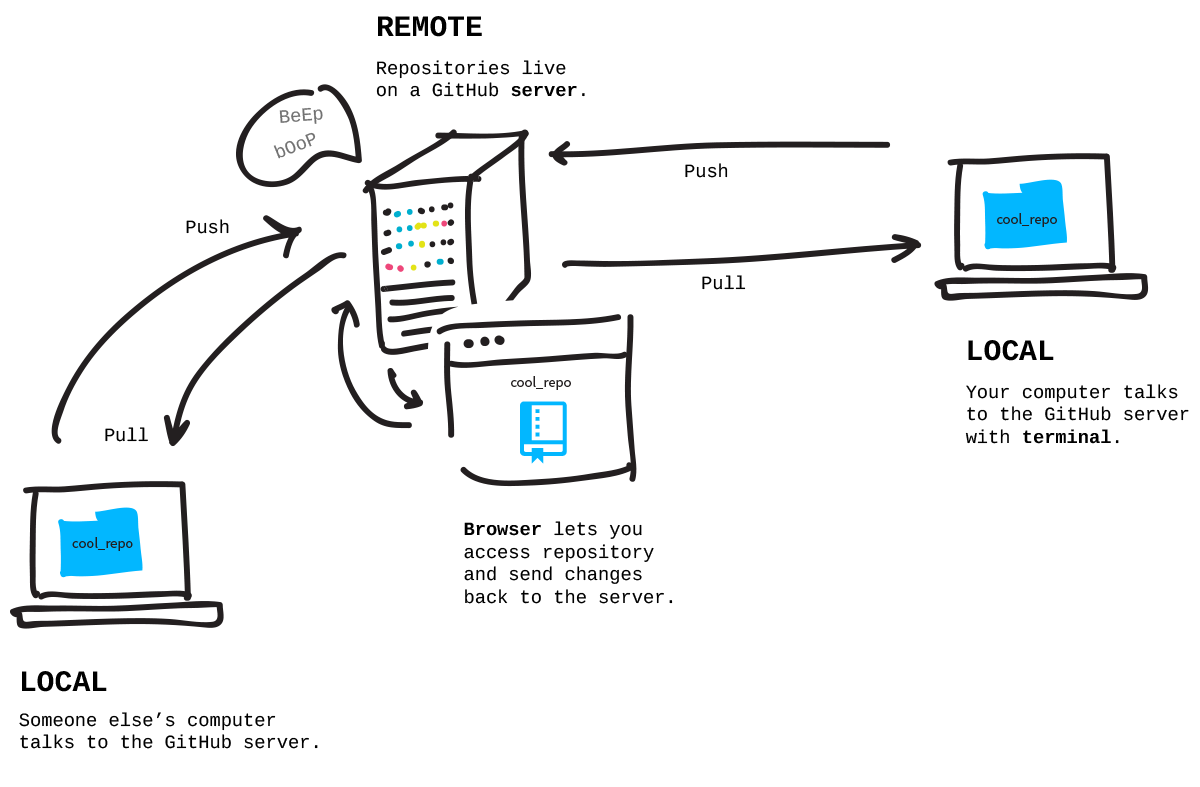
REMOTO

Conecta tu repositorio local a uno remoto y empuja (push) los cambios a este último.

## Remotos (remotes)

Cuándo pones algo en GitHub esa copia vive en uno de los servidores de GitHub. Esto lo convierte en un repositorio **remoto (remote)** porque no está en tu computadora, está en un servidor, "remoto" en algún otro lugar. **Empujando (pushing)** tus cambios **locales** (en tú computadora) a este, lo mantienes al día.

Otras personas siempre pueden obtener lo más reciente de tu proyecto **halando (pulling)** tus cambios desde el remoto (a otra computadora). Así es cómo varias personas pueden trabajar juntas en un proyecto sin necesidad de tener acceso a tu computadora dónde tu copia local es almacenada.



### Crear un repositorio remoto

Si quieres sincronizar tu versión **local** con la que está guardada GitHub.com. Primero crea un nuevo repositorio **remoto (remote)** en GitHub.com.

* Ve a github.com, logueate, y haz click en el '+' en la parte derecha arriba y haz click 'Nuevo repositorio' ('New repository').
* Dale un nombre que coincida con el nombre de tu repositorio local, 'hello-world', y una descripción corta.
* Hazlo público. Esto significa que será listado en tu perfil público.
* No inicialices con una archivo README porque ya tienes un archivo, localmente, llamado 'readme.txt'. Esta opción es una ayuda de GitHub si todavía no has creado uno.
* Las opciones '.gitignore' and 'license' deben quedar en 'none' (ninguna). No las usaremos en este tutorial.
* Haz Click en crear repositorio (create repository)!

## Readmes, .gitignores y Licenses

Estos son archivos comunes en proyectos de código abierto así que para iniciar un proyecto más fácilmente, GitHub provee una opción para crearlos por ti.

Un **readme** lo que es el proyecto, cómo usarlo, y regularmente, cómo contribuir al mismo (Algunas veces hay un archivo extra, 'CONTRIBUTING.md', para esos detalles).

Un **.gitignore** es una lista de archivos a los que Git NO debería hacer seguimiento, por ejemplo, archivos con contraseñas!

Un archivo **license** describe el tipo de licencia que pusiste en tú proyecto. Esto permite a otros saber cómo pueden usarlo. Puedes encontrar más información de los tipos de licencia en: [choosealicense.com](http://www.choosealicense.com).

### Conecta tu Local a tu Remoto

Ahora que has iniciado un repositorio remoto en GitHub.com. En la parte de arriba puedes ver una 'configuración rápida' ('Quick Setup'), asegurate de que el botón 'HTTPS' está seleccionado y copia la dirección, es la dirección de tu repositorio en los servidores de GitHub.

Vuelve a la terminal, y situado dentro de la carpeta 'hello-world' que inicializaste cómo repositorio de Git repository en el reto anterior, vas a decirle a Git que recuerde la dirección de la versión remota en los servidores GitHub. Tú puedes tener varios remotos configurados en tu local por lo que el remoto necesita un nombre. El remoto principal es típicamente llamado origin.

Para agragar un remoto llamado 'origin' a tú repositorio:

git remote add origin <URLFROMGITHUB>

Tu repositorio **local** ahora sabe dónde está tu repositorio **remoto**, llamado 'origin', que está en los servidores de GitHub. Piensalo cómo añadiendo un nombre y un número telefónico en los contactos de tu teléfono móvil para llamarlos rápidamente cada vez que necesites.

**Usuarios de Windows:**

Si tienen **GitHub Desktop** en sus computadoras, un remoto llamado 'origin' es automáticamente creado en su repositorio local. En ese caso, sólo necesitarás decirle cuál URL asociar con 'origin'. Para eso en vez de usar el comando 'add' mencionado anteriormente usen el siguiente:

git remote set-url origin <URLFROMGITHUB>

### Empuja (Push) tu trabajo al remoto

Lo siguiente es hacer **push** (empujar) todo lo que has hecho localmente a tu repositorio remoto en GitHub. Esto es algo que harás con frecuencia para que tu versión remota esté al día y coincida con el estado tu versión local.

Git tiene un sistema de ramas (branches) para que puedas trabajar en diferentes partes del proyecto en tiempos diferentes. Aprenderemos más acerca de ello después, pero por defecto la primera branch es nombrada 'master'. Cuándo empujas (push) (y más adelante halas) desde un proyecto, le dices a Git el **nombre de la rama (branch name)** que quieres y el nombre del **remoto** que en el que debe residir.

En este caso, vamos a enviar una rama llamada 'master' a nuestro remoto en GitHub llamado 'origin'.

git push origin master

Ahora ve a la página de tu repositorio remoto en GitHub.com y refrescala. **Wow!**Todo es lo mismo ahora tanto local cómo remotamente. Felicitaciones, has creado tu primer repositorio público!

* **Agregar conexiones remotas**

git remote add <REMOTENAME> <URL>

* **Modificar la URL de un remoto**

git remote set-url <REMOTENAME> <URL>

* **Halar cambios**

git pull <REMOTENAME> <BRANCHNAME>

* **Ver las direcciones remotas**

git remote -v

* **Empujar cambios**

git push <REMOTENAME> <BRANCH>

FORKS

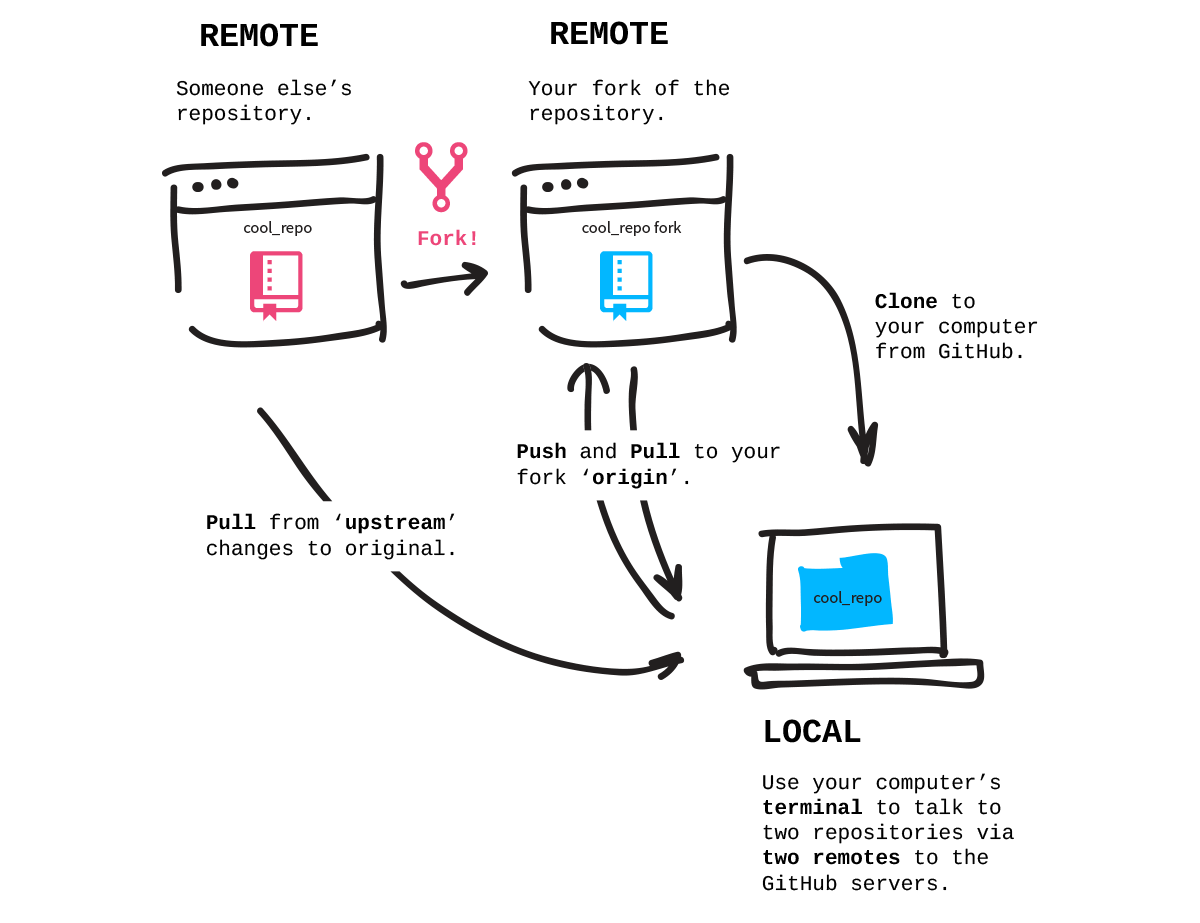
Bifurca (Fork) un proyecto desde GitHub.com y clonalo localmente.

## Bifurcaciones (Forks)

Hasta ahora has creado un proyecto local y lo has subido a GitHub, pero es sólo la mitad de la diversión! La otra mitad está con otras personas y otros proyectos.

Cuándo **bifurcas (fork)** un repositorio, estás creando una copia del mismo en tu cuenta de GitHub. Tu copia bifurcada empieza su vida cómo un repositorio **remoto** que sólo existe en tu cuente de GitHub, no en tu computadora. Las bifucaciones son usadas para crear tu propia versión de un proyecto (Esta división es cómo tomar una bifucación cuándo vas por una carretera) o contribuir con tus cambios (cómo solución de error o nuevas funcionalidades) al proyecto original.

Para conseguir una copia de tu bifurcación en GitHub a tu computadora tu **clonas (clone)** proyecto. La acción de clonar copia el repositorio remoto en tu computadora para que puedas trabajar localmente.



### Bifurca (fork) el repositorio Patchwork

El proyecto con el que vamos a trabajar está en [github.com/jlord/patchwork](http://github.com/jlord/patchwork). Ve a esa paǵina y haz click en el botón 'Fork' arriba a la derecha. Una vez finalizada la animación de bifurcación en la página, tienes una copia en tu cuenta. Copia la dirección HTTP de tu fork desde la barra de direcciones del navegador, esta es la dirección de tu fork en los servidores de GitHub.

### Clona tu fork localmente

Volvemos a la terminal, vas a clonar el repositorio en tu computadora. Automaticamente creará una nueva carpeta para el repositorio así que no tienes necesidad de crear una tú mismo. Asegurate de que no estas clonandolo dentro de otra carpeta con otro repositorio de Git!

Entonces asegurate que no estás dentro del directorio de 'hello-world' de los anteriores retos. Para abandonar esa carpeta para que puedas clonar el nuevo repositorio, sigue los pasos a continuación:

Para salir de la carpeta 'hello-world':  
Consejo: los dos puntos significan un nivel afuera del directorio actual

cd ..

Ahora que ya no estás en un repositorio de Git, clona tu fork:

git clone <URLFROMGITHUB>

Ve a la carpeta creada para el fork (en este caso se debería llamar 'patchwork').

cd patchwork

Ahora que tienes una copia del repositorio en tu computadora y está automaticamente conectada a el repositorio remoto (tu copia bifurcada) en tu cuenta de GitHub. Escribe git remote -v para ver las direcciones que el fork ya tiene configuradas.

### Conectar con el repositorio original

Qué pasa si el repositorio original tiene cambios nuevos? Vas a querer tenerlos y **halar (pull)** aquellos cambios. Para eso agregaremos otra conexión remota, esta vez la original, [github.com/jlord/patchwork](http://github.com/jlord/patchwork), con su URL.

Puedes ponerle el nombre que quieras a esta conexión remota, pero normalmente la gente usa 'upstream' cómo nombre; usemos ese nombre para esto.

git remote add upstream https://github.com/jlord/patchwork.git

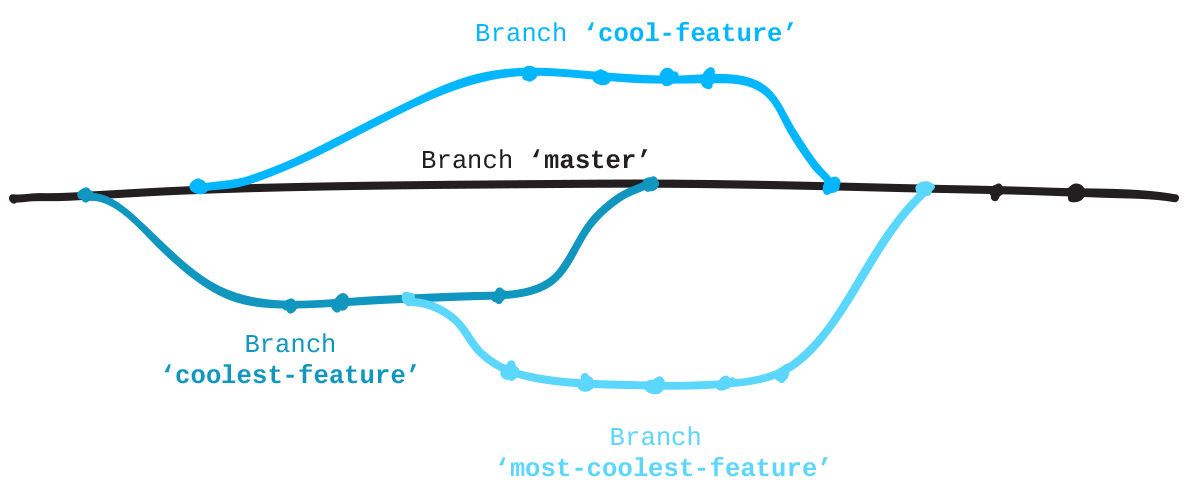
Para estar seguros de que tienes correctamente configuradas las conexiones remotas, escribe git remote -v para listar las direcciones que tienes guardadas. **Derías tener un remoto con nombre 'origin' con la dirección remota de tu fork y otro con el nombre 'upstream' con la dirección del remoto original, osea la URL que está encima de este párrafo.**

* **Agregar un remoto**
* git remote add <REMOTENAME> <URL>
* **Cambiarl la url a un remoto**
* git remote set-url <REMOTENAME> <URL>
* **Ver las conexiones remotas**
* git remote -v

## Branches (ramas)

Los repositorios de Git usan ramas para separar el trabajo cuándo es necesario. Es una práctica común que cuándo se trabaja en un proyecto con otras personas crear una **branch** mantener tus cambios. De esta manera puedes hacer tu trabajo mientras que la rama principal, comunmente llamada 'master', mantiene estable. Cuándo el trabajo en tu rama es finalizado fusionas tu rama con la rama 'master'.

El siguiente diagrama muestra cómo puedes crear una rama a partir de 'master', hacer el trabajo y luego fusionar esos cambios de vuelta a 'master'. Incluso puedes crear una rama desde tu rama, la rama 'master' no necesariamente tiene que ser la base.



Para una mejor entendimiento de cómo funcionan las ramas en un proyecto, revisa la guía de GitHub: [guides.github.com/overviews/flow](http://guides.github.com/overviews/flow/) (en inglés)

## Páginas de GitHub

GitHub automáticamente servirá y alojará sitio web de archivos estáticos en las ramas branches nombradas cómo 'gh-pages'. Este servicio gratuito es llamado [GitHub Pages](http://pages.github.com). Cómo el proyecto bifurcado crea un sitio web, su rama principal se llama 'gh-pages' en vez de 'master'. Todos los repositorios que tienen una rama 'gh-pages' con sitio web pueden ser encontradas, en línea, usando el siguiente patrón en la URL:

http://githubusername.github.io/repositoryname

### Crea una rama

Cuándo creas una rama, Git copia todo lo que hay en la rama que estás actualmente y lo pone en la rama que acabas de crear.

Estando dentro de tu repositorio local 'patchwork', escribe git status para ver en cuál rama estás actualmente. Git podría decirte que estás en la rama 'gh-pages'.

Ahora crea una nueva rama y nombrala "add-<username>", donde 'username' es tu nombre de usuario de GitHub. Por ejemplo, "add-jlord". **Las ramas son sensibles a mayúsculas así que nombra tu rama exactamente de la misma forma que aparece tu cuenta de usuario de GitHub**.

git branch <BRANCHNAME>

Ahora tienes una rama con un nombre que es idéntica a la rama 'gh-pages'.

Para pasarte a esa rama debes hacer **checkout** a ella. Ve a la nueva rama:

git checkout <BRANCHNAME>

### Paso: Crear un nuevo archivo

Vuelve a tu editor de texto:

* Crea un archivo llamado "add-<USERNAME>.txt", dónde 'username' es tu nombre de usuario. Por ejemplo, "add-jlord.txt".
* Entonces, sólo escribe tu nombre de usuario GitHub en él, eso es todo. Por ejemplo, Yo escribiría 'jlord'.
* Guarda este archivo en la carpeta 'contributors' dentro de Patchwork: **Patchwork/contributors/add-yourusername.txt**
* Siguiente paso, Verifica los cambios (mira abajo).

### Check-in

Haz los siguientes pasos para guardar tus cambios:

git status

git add <contributors/FILENAME>

git commit -m "mensaje de commit"

Ahora haz push de tus cambios **al fork**, 'origin', en GitHub:

git push origin <BRANCHNAME>

#### Permission denied...error: 403

Estas empujando cambios a un repositorio en el cual no tienes permisos de escritura. En este caso, vas a queerer enviarlos a el original 'jlord/patchwork'. Asegurate que estás apuntando a 'origin' y que origin apunta a tu fork en GitHub. To check and see what your remotes are and where they point run git remote -v. Deberías tener a 'upstream' apuntando a 'jlord/patchwork' y 'origin' apuntando a 'yourusername/patchwork'.

Para arreglar un remoto que apunta a la dirección equivocada puedes cambiar su url: git remote set-url origin .

#### Authentication failed...error: 401

Tu identidad no pudo ser verificada. Pueden que tengas (Autenticación a dos pasos) activada en este caso tienes que usar un toke de acceso personal cómo tu contraseña. Puedes generar uno en [siguiendo las instrucciones](https://help.github.com/articles/creating-an-access-token-for-command-line-use). Necesitarás mantenelo y usarlo en lugar de tu contraseña cuándo sea requerido. También puedes [guardar este token de acceso](https://help.github.com/articles/caching-your-github-password-in-git/) en tu computadora para re usarlo.

#### El archivo no está en la carpeta contributors

El archivo creado debería estar en el interior de la carpeta 'contributors' en el repositorio Patchwork repository. Si lo pusiste en otro lugar, simplemente usa el Finder el explorador de windows para mover el archivo a la carpeta. Puedes verificar con git status otra vez y verás tus cambios. Añade y haz commit "all" (-A) de estos cambios (adiciones y borrados) con los siguientes comandos.

git add -A

git commit -m "mover archivo a la carpeta contributors"

#### Nombre de la rama esperado: \_\_\_\_\_

El nombre de la rama debería coincidir con tu nombre de usuario exactament. Para cambiar el nombre de la rama:

git branch -m <NEWBRANCHNAME>

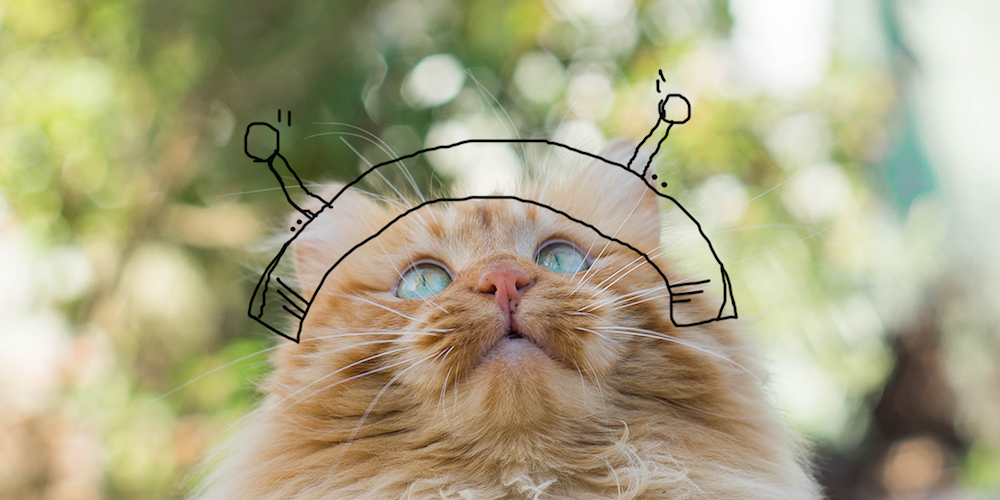
Una vez hagas estos cambios, verifica nuevamente!

* **Crear y moverse a la rama en una sola línea**
* git checkout -b <BRANCHNAME>
* **Crear una rama nueva**
* git branch <BRANCHNAME>
* **Moverse a una rama**
* git checkout <BRANCHNAME>
* **Listado de las ramas**
* git branch
* **Renombrar la rama en la que estás situado**
* git branch -m <NEWBRANCHNAME>
* **Vericar la rama en la que estás y su estado**
* git status

## Social Coding

Trabajar con otros es una de las mejores cosas de GitHub porque es más fácil trabajar desde cualquier lugar o zona horaria con otras personas.

**Los colaboradores** otros usuarios de GitHub que tienen permisos para hacer cambios a un repositorio que pertenece a otra persona. Puedes escuchar también el término **contribuidor (contributor)**, esto describe a un usuario de GitHub que ha hecho contribuciones, por medio de Forks y Pull requests, a un proyecto.

[@reporobot](https://github.com/reporobot) es un robot del espacio exterior que ama colaborar en repos.

### Hola, Repo Robot!

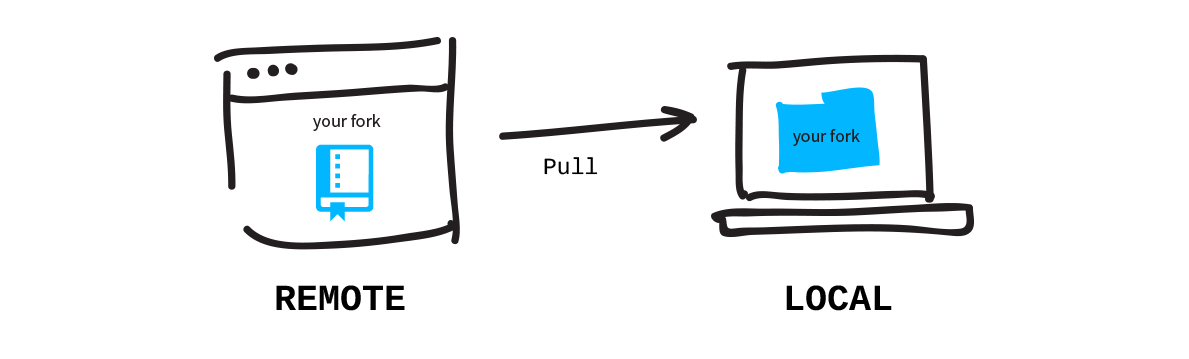
Para añadir colaboradores a uno de nuestros proyectos, visita la página de tu repositorio en tu cuenta de GitHub y haz click en la pestaña 'Settings' (configuración) en el menú en la parte superior de la página. Selecciona la opción 'Collaborators' (colaboradores). Escribe el nombre de usuario dentro del campo de texto y haz click en 'Add'.

Ve a la página de tu repositorio bifurcado en GitHub y añade 'reporobot' cómo colaborador. La URL debería cómo la siguiente, pero cómo nombre de usuario.

github.com/YOURUSERNAME/patchwork/settings/collaboration

**Halando (pulling) de un remoto**

Si estás trabajando en algo con alguien más necesitas estar al día con los últimos cambios. Allí vas a querer **halar (pull)** cualquiera de los cambios que pueden haber sido hechos al repositorio central en GitHub.



**¿Qué ha estado haciendo Reporobot?**

Revisa si Reporobot ha hecho algún cambio en tu branch halando cualquier cambio hecho en el remoto con nombre 'origin' en GitHub:

git pull <REMOTENAME> <BRANCHNAME>

Si nada ha cambiado, Git te mostrará un mensaje similar a 'Already up-to-date'. Si hay cambios, Git fusionará esos cambios en versión local.

¿Hizo Reporobot cambios? Git te dice dónde fueron los cambios. Puedes abrir ese archivo con los cambios que Reporobot hizo. ¡Sorpresa, Reporobot es un artista!

* **Git status**

git status

* **Halar (pull) los cambios desde una branch remota**

git pull <REMOTENAME> <REMOTEBRANCH>

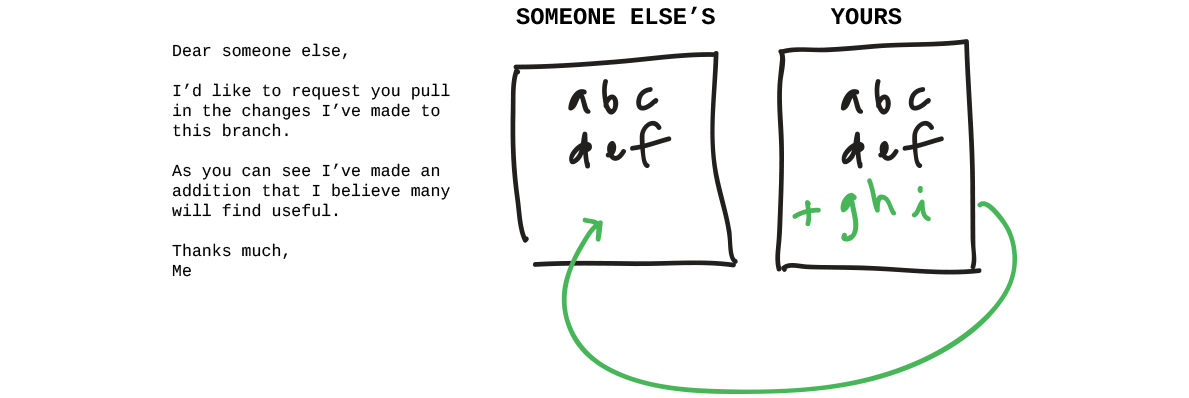
* **Ver los cambios de una rama remota antes de halarlos**

git fetch --dry-run

## Pull Requests

Frecuentemente, cuándo haces cambios y mejoras a un proyecto que has bifurcado, vas a querer enviar esos cambios a los encargados de mantener el repositorio original y **solicitarles (request)** que **halen (pull)** esos cambios al original para que aquellos que lo usan se puedan beneficiar de las actualizaciones, eso es un **pull request**.

Queremos añadirte a la lista de personas que finalizaron este taller, así que crea un **pull request** para añadir tu archivo con tu usuario al repositorio original: [github.com/jlord/patchwork](https://github.com/jlord/patchwork).



### Crear un pull request

Visita el repositorio original que bifurcaste en GitHub, en este caso [http://github.com/jlord/patchwork](https://github.com/jlord/patchwork).

GitHub detectará con frecuencia si has enviado (push) una branch a un fork y lo mostrará en la parte superior de la paǵina del repositorio original. Si puedes ver que con tu rama 'add-username', puedes hacer click en 'Crear pull request' (create a Pull Request) desde allí. Si no:

* Haz Click en el botón verde que dice 'New pull request'.
* Selecciona la rama que tiene los cambios que quieres enviar. Debería ser la que se llama 'add-yourusername'.

Ahora verás una página con los detalles que están en el proceso de ser enviados. Esta página muestra los commits los cambios, en el formulario de diff, asociando el pull request y comparandolo con la rama 'gh-pages' de el original.

Si el repositorio original tiene **documentación de contribución**, GitHub lo enlazará. Esta es la documentación que los propietarios del repositorio consideran cómo es la mejor manera de hacer contribuciones a ese proyecto —Es de gran ayuda leer si quisieras saber si tus cambios serán adoptados!

Si todo en la página se ve bien y es lo que esperabas:

* Click en 'Create pull request'
* Añade un título y una descripción a los cambios que han sugerido al autor del repositorio original para ser incorporados.
* Haz Click en 'Send pull request'!

**Bingo! Enviaste un pull request — tomate unos segundos para disfrutar el momento.**

Si todo está bien con tu pull request, será fusionado en unos momentos. Si no está automaticamente fusionado después de unos poco minutos, entonces tendrás algunos comentarios que te hizo @reporobot explicando por qué no los pudo fusionar (merge). Si este es el caso, cierra tu pull request en GitHub, haz los cambios necesarios en tu rama, vuelve a enviar los cambios a tu fork y reabre (esto le avisará @reporobot para que revise nuevamente) tu pull request

## Limpieza

Una vez que ya no necesitas las ramas puedes eliminarlas local y remotamente. Esto ayuda a que no termines con una pila de ramas viejas.

### Fusionar (merge) localmente

Tu pull request ha sido fusionado! Tu rama fue fusionada en la rama 'gh-pages' de el repositorio original en GitHub. Puedes fusionar la rama localmente, también.

Primero, muevete a la rama a la cual quieres fusionarle los cambios en este caso, la rama 'gh-pages'.

git checkout gh-pages

Ahora dile a Git desde cuál rama quieres traer los cambios, en este caso, tu rama "feature" que empieza con "add-".

git merge <BRANCHNAME>

Ordenar el repositorio eliminando la "feature branch". Ahora que has fusionado no la necesitarás.

git branch -d <BRANCHNAME>

También puedes eliminar la rama de tu remoto en GitHub:

git push <REMOTENAME> --delete <BRANCHNAME>

### Hala (Pull) desde Upstream

Y por último pero no menos importante, el repositorio original ha cambiado desde que tu pull request fue fusionado, añadió tu nombre a el sitio web! Si halas esos cambios desde upstream estarás al día y tendrás esa versión también. Puedes ver el sitio online en: tunombredeusuario.github.io/patchwork.

Para halar desde el original upstream:

git pull upstream gh-pages

## Felicitaciones!

Has creado repositorios, repositorios remotos, trabajado con un colaborador, empujado (push), halado (pulled) y te has unido millones de personas desarrollando y enriqueciendo el código abierto!

Visita [jlord.github.io/patchwork](https://jlord.github.io/patchwork) para ver tu nombre incorporado! Quieres continuar? Revisa la sección [Qué sigue](file:///C:\Users\exgciud\Downloads\Git-it-Win-ia32\Git-it-Win-ia32\resources\app\built\es-CO\pages\resources.html#what-next) section en la página de recursos de esta aplicación.

* **Fusiona una rama a tu rama actual**

git merge <BRANCHNAME>

* **Cambia la branch en la que estás trabajando**

git checkout <BRANCHNAME>

* **Borra una rama local**

git branch -d <BRANCHNAME>

* **Borra una rama remota**

git push <REMOTENAME> --delete <BRANCHNAME>

* **Hala (Pull) desde una rama remota**

git pull <REMOTENAME> <BRANCHNAME>