1. GIT es el SCV (sistema de control de versiones) de código abierto [**más utilizado**](https://www.g2.com/categories/version-control-systems?utf8=%E2%9C%93&order=g2_score) que te permite rastrear los cambios realizados en los archivos. Las empresas y los programadores suelen utilizar el GIT para colaborar en el desarrollo de software y aplicaciones.
2. Un proyecto GIT consta de tres secciones principales: **el directorio de trabajo**, **el área de preparación** y **el directorio git**.
3. El directorio de trabajo es donde se agregan, borran y editan los archivos. Luego, los cambios son preparados (indexados) en el área de preparación. Después de que confirmes tus cambios, la instantánea de los cambios se guardará en el directorio git.
4. Todo el mundo puede usar GIT ya que está disponible para [**Linux**](https://git-scm.com/book/es/v2), [**Windows**](https://gitforwindows.org/), [**Mac**](https://git-scm.com/download/mac) y [**Solaris**](https://www.opencsw.org/packages/git/). El software puede tener una fuerte curva de aprendizaje, pero hay muchos [**tutoriales**](https://www.hostinger.co/tutoriales/instalar-git-en-distintos-sistemas-operativos) disponibles para ayudarte.
5. Aquí hay algunos comandos básicos de GIT que debes conocer:

git initcreará un nuevo repositorio local GIT en el directorio actual:

1. Como alternativa, puedes crear un repositorio dentro de un nuevo directorio especificando el nombre del proyecto:

git init [nombre del proyecto]

1. git clone se usa para copiar un repositorio. Si el repositorio está en un servidor remoto, usa:

git clone nombredeusuario@host:/path/to/repository

A la inversa, ejecuta el siguiente comando básico para copiar un repositorio local:

git clone /path/to/repository

1. git addse usa para agregar archivos al área de preparación. Por ejemplo, el siguiente comando de Git básico indexará el archivo temp.txt:

git add <temp.txt>

1. git commitcreará una instantánea de los cambios y la guardará en el directorio git.

git commit –m “El mensaje que acompaña al commit va aquí”

* Ten en cuenta que los cambios confirmados no llegarán al repositorio remoto.
* **git config** puede ser usado para establecer una configuración específica de usuario, como el email, nombre de usuario y tipo de formato, etc. Por ejemplo, el siguiente comando se usa para establecer un email:

git config --global user.email tuemail@ejemplo.com

* La opción -global le dice a GIT que vas a usar ese correo electrónico para todos los repositorios locales. Si quieres utilizar diferentes correos electrónicos para diferentes repositorios, usa el siguiente comando:

git config --local user.email tuemail@ejemplo.com

* **git status**muestra la lista de los archivos que se han cambiado junto con los archivos que están por ser preparados o confirmados.

git status

* **git push**se usa para enviar confirmaciones locales a la rama maestra del repositorio remoto. Aquí está la estructura básica del código:

git push  origin <master>

* Reemplaza <master> con la rama en la que quieres enviar los cambios cuando no quieras enviarlos a la rama maestra.
* **git checkout**crea ramas y te ayuda a navegar entre ellas. Por ejemplo, el siguiente comando crea una nueva y automáticamente se cambia a ella:

command git checkout -b <branch-name>

* Para cambiar de una rama a otra, sólo usa:

git checkout <branch-name>

* **git remote**te permite ver todos los repositorios remotos. El siguiente comando listará todas las conexiones junto con sus URLs:

git remote -v

* Para conectar el repositorio local a un servidor remoto, usa este comando:

git remote add origin <host-or-remoteURL>

* Por otro lado, el siguiente comando borrará una conexión a un repositorio remoto especificado:

git remote <nombre-del-repositorio>

* **git branch** se usa para listar, crear o borrar ramas. Por ejemplo, si quieres listar todas las ramas presentes en el repositorio, el comando debería verse así:

git branch

* Si quieres borrar una rama, usa:

git branch -d <branch-name>

* **git pull**fusiona todos los cambios que se han hecho en el repositorio remoto con el directorio de trabajo local.

git pull

* **git merge** se usa para fusionar una rama con otra rama activa:

git merge <branch-name>

* **git diff**se usa para hacer una lista de conflictos. Para poder ver conflictos con respecto al archivo base, usa:

git diff --base <file-name>

* El siguiente comando se usa para ver los conflictos que hay entre ramas antes de fusionarlas:

git diff <source-branch> <target-branch>

* Para ver una lista de todos los conflictos presentes usa:

git diff

* **git tag**marca commits específicos. Los desarrolladores lo usan para marcar puntos de lanzamiento como v1.0 y v2.0.

git tag 1.1.0 <instert-commitID-here>

* **git log**se usa para ver el historial del repositorio listando ciertos detalles de la confirmación. Al ejecutar el comando se obtiene una salida como ésta:

commit 15f4b6c44b3c8344caasdac9e4be13246e21sadw

Author: Alex Hunter <alexh@gmail.com>

Date:   Mon Oct 1 12:56:29 2016 -0600

* **git reset**sirve para resetear el index y el directorio de trabajo al último estado de confirmación.

git reset - -hard HEAD

* **git rm**se puede usar para remover archivos del index y del directorio de trabajo.

git rm filename.txt

* **git stash** guardará momentáneamente los cambios que no están listos para ser confirmados. De esta manera, pudes volver al proyecto más tarde.

git stash

* **git show**se usa para mostrar información sobre cualquier objeto git.

git show

* **git fetch**le permite al usuario buscar todos los objetos de un repositorio remoto que actualmente no se encuentran en el directorio de trabajo local.

git fetch origin

* **git ls-tree**te permite ver un objeto de árbol junto con el nombre y modo de cada ítem, y el valor blob de SHA-1. Si quieres ver el HEAD, usa:

git ls-tree HEAD

* **git cat-file**se usa para ver la información de tipo y tamaño de un objeto del repositorio. Usa la opción -p junto con el valor SHA-1 del objeto para ver la información de un objeto específico, por ejemplo:

git cat-file –p d670460b4b4aece5915caf5c68d12f560a9fe3e4

* **git grep** le permite al usuario buscar frases y palabras específicas en los árboles de confirmación, el directorio de trabajo y en el área de preparación. Para buscar por www.hostinger.com en todos los archivos, usa:

git grep “www.hostinger.com”

* **gitk** muestra la interfaz gráfica para un repositorio local. Simplemente ejecuta:

gitk

* **git instaweb**te permite explorar tu repositorio local en la interfaz GitWeb. Por ejemplo:

git instaweb –http=webrick

* **git gc** limpiará archivos innecesarios y optimizará el repositorio local.

git gc

* **git archive**le permite al usuario crear archivos zip o tar que contengan los constituyentes de un solo árbol de repositorio. Por ejemplo:

git archive - -format=tar master

* **git prune**elimina los objetos que no tengan ningún apuntador entrante.

git prune

* **git fsck** realiza una comprobación de integridad del sistema de archivos git e identifica cualquier objeto corrupto

git fsck

* **git rebase** se usa para aplicar ciertos cambios de una rama en otra. Por ejemplo:

git rebase master

II -10 Comandos de Git Que Todo Desarrollador Debería Saber

[**Nora Gonzalo Ciordia**](https://www.freecodecamp.org/espanol/news/author/nora/)



Git es una parte importante de quien programa a diario (especialmente si estás trabajando con un equipo) y se usa extensamente en la industria de software.

Desde que existe una gran variedad de comandos que puedes utilizar, dominar Git requiere tiempo. Pero algunos comandos se utilizan más frecuentemente (algunos hasta a diario varias veces). Por tanto, en este artículo, les compartiré y explicaré los 10 comandos de Git más usados que todo desarrollador debería de conocer.

**Nota: Para entender este artículo, tienes que conocer lo básico acerca de Git.**

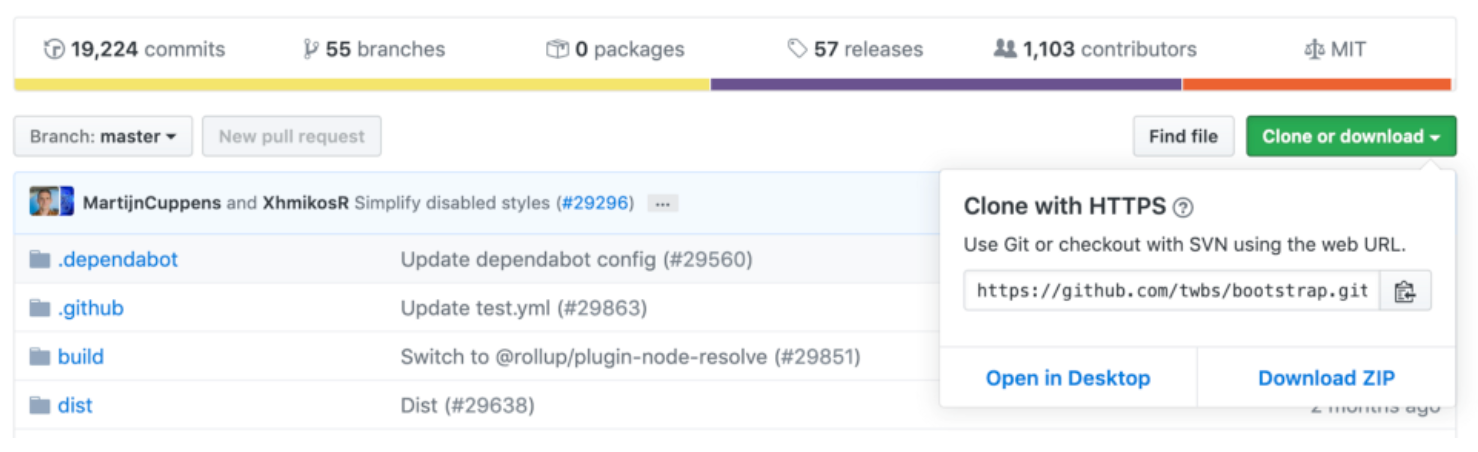
**1. Git clone**

Git clone es un comando para descargarte el código fuente existente desde un repositorio remoto (como Github, por ejemplo). En otras palabras, Git clone básicamente realiza una copia idéntica de la última versión de un proyecto en un repositorio y la guarda en tu ordenador.

Hay un par de formas de descargar el código fuente, pero principalmente yo prefiero **clonar de la forma con https**:

git clone <https://link-con-nombre-del-repositorio>

Por ejemplo, si queremos descargar un proyecto desde Github, todo lo que necesitamos es hacer clic sobre el botón verde (clonar o descargar), copiar la URL de la caja y pegarla después del comando git clone que he mostrado más arriba.

Código fuente de Bootstrap en Github

Esto hará una copia del proyecto en tu espacio de trabajo local y así podrás empezar a trabajar con él.

**2. Git branch**

Las ramas (branch) son altamente importantes en el mundo de Git. Usando ramas, varios desarrolladores pueden trabajar en paralelo en el mismo proyecto simultáneamente. Podemos usar el comando git branch para crearlas, listarlas y eliminarlas.

**Creando una nueva rama:**

git branch <nombre-de-la-rama>

Este comando creará una rama en local. Para enviar (push) la nueva rama al repositorio remoto, necesitarás usar el siguiente comando:

git push <nombre-remoto> <nombre-rama>

**Visualización de ramas:**

git branch

git branch --list

**Borrar una rama:**

git branch -d <nombre-de-la-rama>

**3. Git checkout**

Este es también uno de los comandos más utilizados en Git. Para trabajar en una rama, primero tienes que cambiarte a ella. Usaremos **git checkout** principalmente para cambiarte de una rama a otra. También lo podemos usar para chequear archivos y commits.

git checkout <nombre-de-la-rama>

Hay algunos pasos que debes seguir para cambiarte exitosamente entre ramas:

* Los cambios en tu rama actual tienen que ser confirmados o almacenados en el guardado rápido (stash) antes de que cambies de rama.
* La rama a la que te quieras cambiar debe existir en local.

**Hay también un comando de acceso directo que te permite crear y cambiarte a esa rama al mismo tiempo:**

git checkout -b <nombre-de-tu-rama>

Este comando crea una nueva rama en local (-b viene de rama (branch)) y te cambia a la rama que acabas de crear.

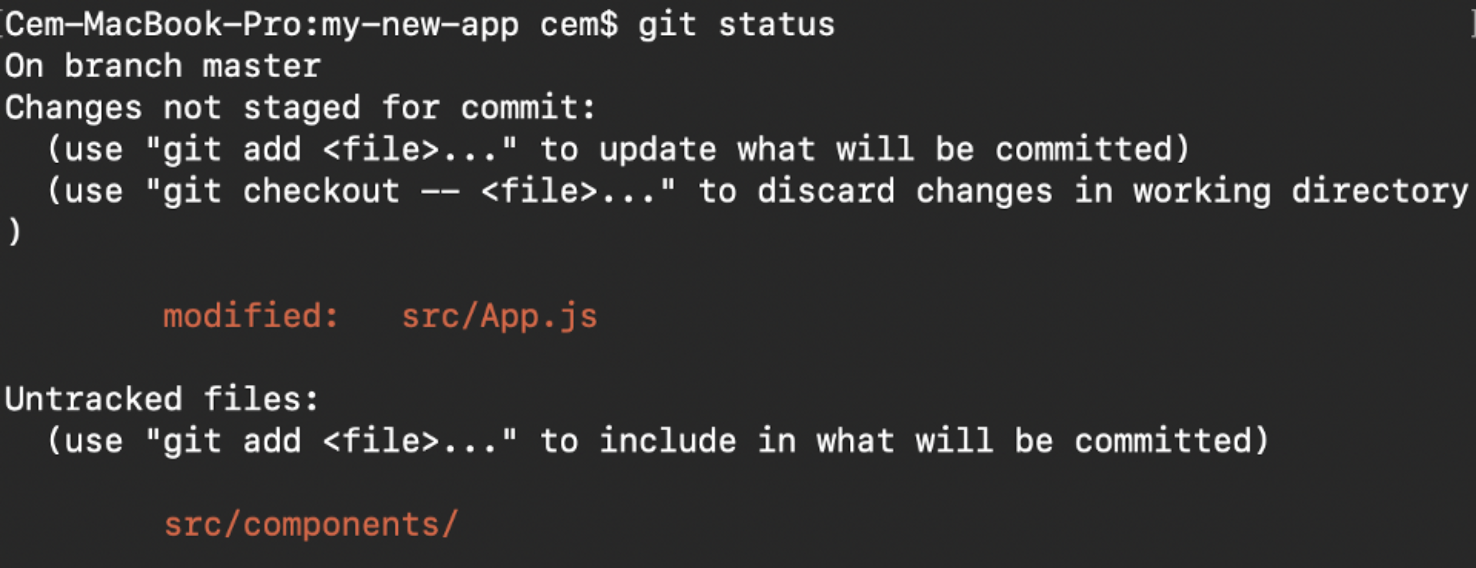
**4. Git status**

El comando de git status nos da toda la información necesaria sobre la rama actual.

git status

Podemos encontrar información como:

* Si la rama actual está actualizada
* Si hay algo para confirmar, enviar o recibir (pull).
* Si hay archivos en preparación (staged), sin preparación(unstaged) o que no están recibiendo seguimiento (untracked)
* Si hay archivos creados, modificados o eliminados

git status nos da información acerca del archivo y las ramas

**5. Git add**

Cuando creamos, modificamos o eliminamos un archivo, estos cambios suceden en local y no se incluirán en el siguiente commit (a menos que cambiemos la configuración).

Necesitamos usar el comando git add para incluir los cambios del o de los archivos en tu siguiente commit.

**Añadir un único archivo:**

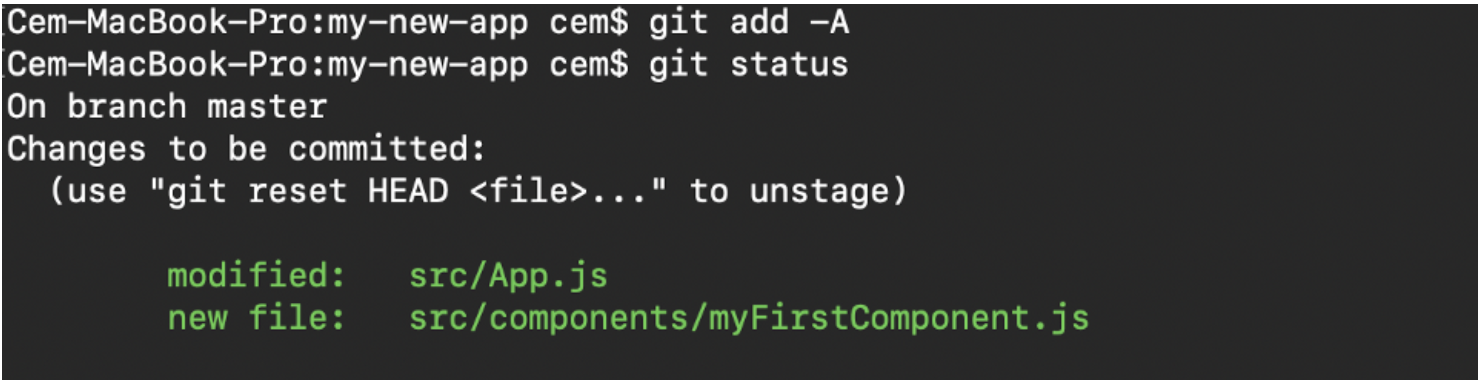
git add <archivo>

**Añadir todo de una vez:**

git add -A

Si revisas la captura de pantalla que he dejado en la sección 4, verás que hay nombres de archivos en rojo - esto significa que los archivos sin preparación. Estos archivos no serán incluidos en tus commits hasta que no los añadas.

**Para añadirlos, necesitas usar el git add:**

Los archivos en verde han sido añadidos a la preparación gracias al git add

**Importante: El comando git add no cambia el repositorio y los cambios que no han sido guardados hasta que no utilicemos el comando de confirmación git commit.**

**6. Git commit**

Este sea quizás el comando más utilizado de Git. Una vez que se llega a cierto punto en el desarrollo, queremos guardar nuestros cambios (quizás después de una tarea o asunto específico).

Git commit es como establecer un punto de control en el proceso de desarrollo al cual puedes volver más tarde si es necesario.

También necesitamos escribir un mensaje corto para explicar qué hemos desarrollado o modificado en el código fuente.

git commit -m "mensaje de confirmación"

**Importante: Git commit guarda tus cambios únicamente en local.**

**7. Git push**

Después de haber confirmado tus cambios, el siguiente paso que quieres dar es enviar tus cambios al servidor remoto. Git push envía tus commits al repositorio remoto.

git push <nombre-remoto> <nombre-de-tu-rama>

De todas formas, si tu rama ha sido creada recientemente, puede que tengas que cargar y subir tu rama con el siguiente comando:

git push --set-upstream <nombre-remoto> <nombre-de-tu-rama>

or

git push -u origin <nombre-de-tu-rama>

**Importante: Git push solamente carga los cambios que han sido confirmados.**

**8. Git pull**

El comando **git pull**se utiliza para recibir actualizaciones del repositorio remoto. Este comando es una combinación del **git fetch** y del **git merge** lo cual significa que cundo usemos el git pull recogeremos actualizaciones del repositorio remoto (git fetch) e inmediatamente aplicamos estos últimos cambios en local (git merge).

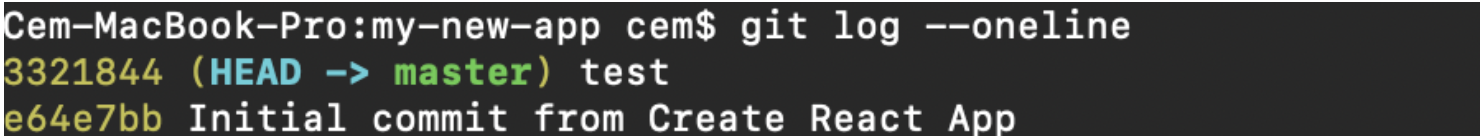
git pull <nombre-remoto>

**Esta operación puede generar conflictos que tengamos que resolver manualmente.**

**9. Git revert**

A veces, necesitaremos deshacer los cambios que hemos hecho. Hay varias maneras para deshacer nuestros cambios en local y/o en remoto (dependiendo de lo que necesitemos), pero necesitaremos utilizar cuidadosamente estos comandos para evitar borrados no deseados.

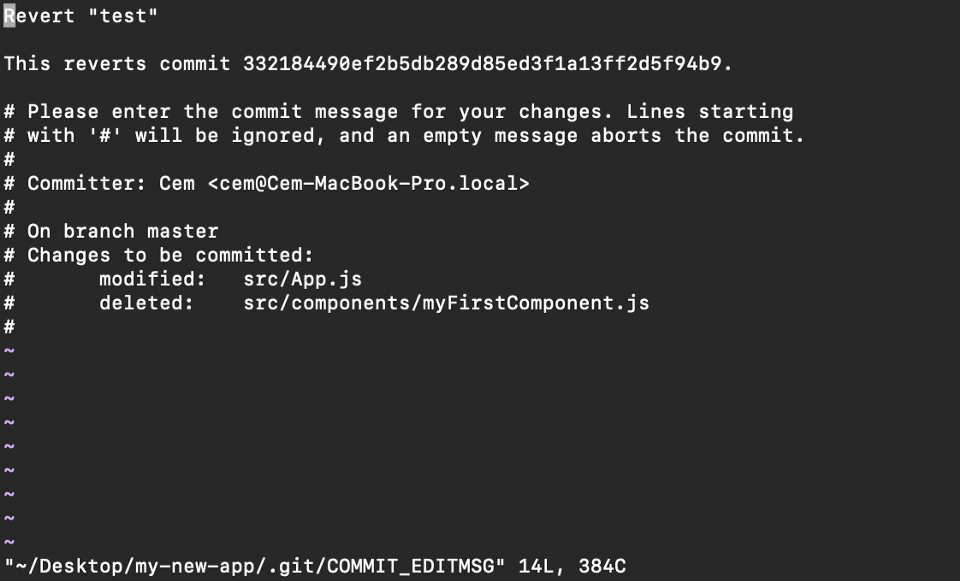
Una manera segura para deshacer nuestras commits es utilizar **git revert**. Para ver nuestro historial de commits, primero necesitamos utilizar el  **git log -- oneline:**

histórico de git en mi rama master

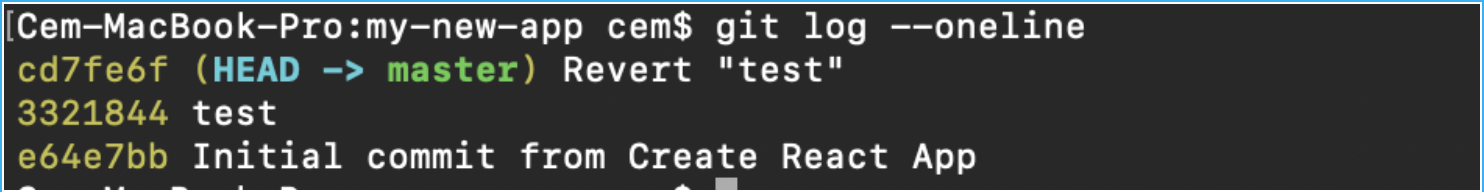
Entonces, solo necesitamos especificar el código de comprobación que encontrarás junto al commit que queremos deshacer:

git revert 3321844

Después de esto, verás una pantalla como la de abajo -tan solo presiona **shift + q** para salir:



El comando git revert deshará el commit que le hemos indicado, pero creará un nuevo commit deshaciendo la anterior:

commit generado con el git revert

La ventaja de utilizar **git revert** es que no afecta al commit histórico. Esto significa que puedes seguir viendo todos los commits en tu histórico, incluso los revertidos.

Otra medida de seguridad es que todo sucede en local a no ser que los enviemos al repositorio remoto. Por esto es que git revert es más seguro de usar y es la manera preferida para deshacer los commits.

**10. Git merge**

Cuando ya hayas completado el desarrollo de tu proyecto en tu rama y todo funcione correctamente, el último paso es fusionar la rama con su rama padre (dev o master). Esto se hace con el comando git merge.

Git merge básicamente integra las características de tu rama con todos los commits realizados a las ramas dev (o master).  Es importante que recuerdes que tienes que estar en esa rama específica que quieres fusionar  con tu rama de características.

Por ejemplo, cuando quieres fusionar tu rama de características en la rama dev:

**Primero, debes cambiarte a la rama dev:**

git checkout dev

**Antes de fusionar, debes actualizar tu rama dev local:**

git fetch

**Por último, puedes fusionar tu rama de características en la rama dev:**

git merge <nombre-de-la-rama>

**Pista: Asegúrate de que tu rama dev tiene la última versión antes de fusionar otras ramas, si no, te enfrentarás a conflictos u otros problemas no deseados.**

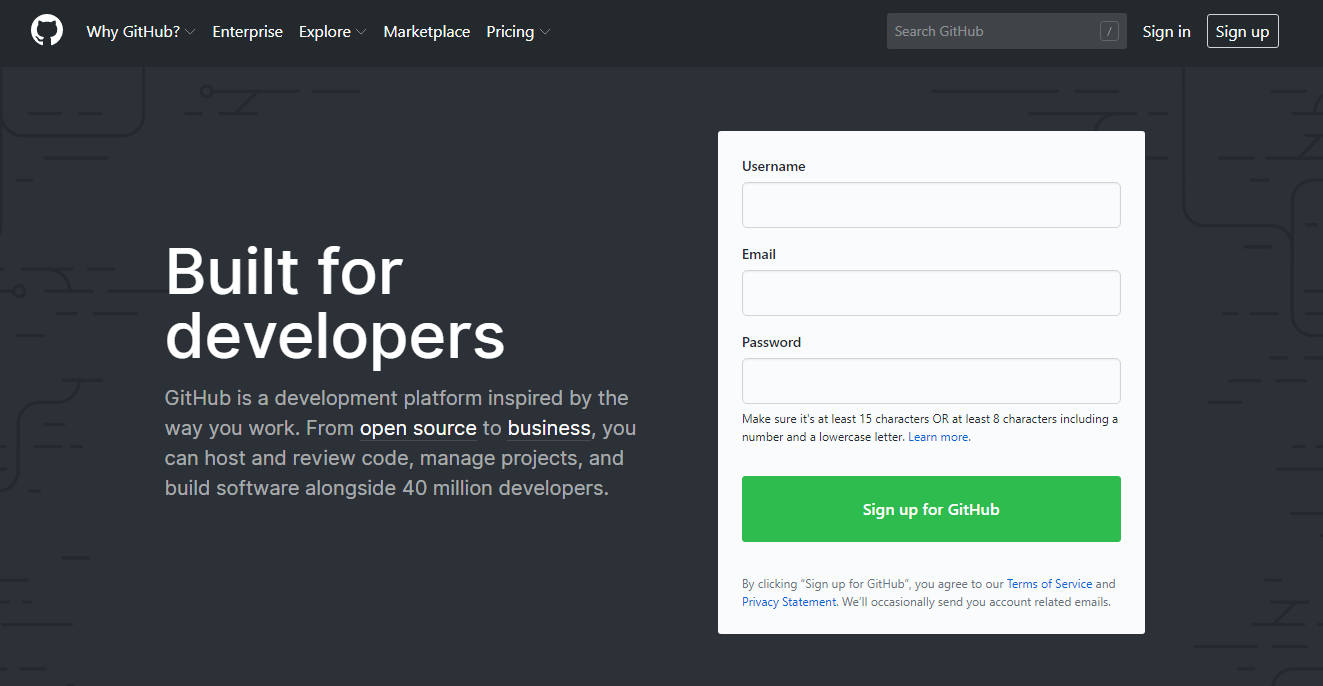
Aquí están mis 10 comandos de git más usados cuando me enfrento a la programación en mi día a día. Hay muchas más cosas que aprender sobre Git y las explicaré más adelante en otros artículos.

# III -Qué es GitHub y comandos esenciales de Git

* [22 AGOSTO 2019](https://www.wnpower.com/blog/que-es-github-comandos-de-git/)
* por: [WNPOWER](https://www.wnpower.com/blog/author/staff-wnpower/)
* en: [HOSTING](https://www.wnpower.com/blog/hosting/)

Además de ser un gran repositorio de código, **[GitHub](https://github.com/" \t "_blank)** es un sistema control de versiones y seguimiento de proyectos, que por medio de sus funciones “social” se convierte en una red colaborativa para para desarrolladores. Cuenta con una interfaz web amigable, está hecho en Ruby y ha estado ganando popularidad constantemente desde su aparición el año 2010.

Es normal que si te relacionas con desarrolladores o con personas que se dediquen a algo relacionado con el desarrollo de sitios web y apps, hayas escuchado hablar acerca de GitHub. Lo más probable es que cuando presencies una conversación acerca del tema, se trate de elogios, de lo fácil que es de utilizar, y oirás cosas como: “Oye, te comparto mi repo”, “copia el código y pégalo” o “ayer lo dejé documentado”. ¡Y no es para menos! A partir de su lanzamiento, la vida de los desarrolladores ha cambiado sustancialmente, haciendo que su trabajo de coding sea más fácil, comprensible y organizado.



## ****Antes de ir a GitHub, ¿Qué es Git?****

Antes de posar nuestra mirada sobre GitHub, veamos de qué se trata Git. Git es un sistema de control de versiones de código abierto que fue creado por Linus Trovalds, el creador de Linux. Por si no sabías lo que significa un control de versiones, los desarrolladores aplican cambios en sus códigos todo el tiempo. Primero para las versiones beta, luego vienen las versiones estables o definitivas, y también luego del lanzamiento de la aplicación o sitio web que están codeando.

Los sistemas de control de versiones mantienen estas revisiones documentadas, almacenando las modificaciones usando un repositorio central. Esto permite a los desarrolladores hablar un mismo idioma y entenderse fácilmente, ya que pueden descargar una nueva versión del soft que estén programando, realizar cambios y cargar la última revisión. Cada desarrollador puede ver estos nuevos cambios, descargarlos y contribuir.

## ****¿Qué es Github y qué lo hace tan especial?****

Al inicio de esta nota, te contamos superficialmente de qué trata: GitHub es un aplicación web para control de cambios y documentación, que funciona a la vez como hosting de código y red social entre desarrolladores. ¿Qué es lo que lo hace tan especial? Bien, estas son algunas de las principales razones por las que a los dev les gusta usar GitHub:

### **Funciona a modo de repositorio**

En la jerga es conocido como “repo”, y es una ubicación en donde se almacenan todos los archivos y códigos de un proyecto. Cada proyecto tiene un espacio dedicado y es accesible desde una dirección web. Los repositorios pueden ser públicos o privados. También permite bifurcar (o hacer “branch”) nuevos proyectos a partir de los existentes de manera de tener correlación entre ellos o añadir nuevas funciones.

### **El aspecto social de GitHub**

Su similitud con las redes sociales es tal vez el aspecto más poderoso de GitHub. En GitHub cada desarrollador cuenta con un usuario privado que actúa como un perfil, mostrando sus trabajos anteriores, contribuciones a los proyectos en los que participó y más. Tanto en los proyectos públicos como privados, se pueden discutir diferentes aspectos de tal forma que permite aportar conocimientos y mejoras para poder avanzar mejor en el desarrollo. ¿Imaginas qué tedioso podría ser llevar un proyecto si tienes que contactarte con otros desarrolladores, por ejemplo vía correo electrónico? Pues bien, hasta no hace mucho tiempo esto sucedía.

### **Control de cambios**

Cuando varios desarrolladores colaboran en un proyecto es difícil hacer un seguimiento de las rev o versiones, quién cambió qué, cuándo y dónde se almacenan esos archivos. Afortunadamente GitHub se ocupa de este problema haciendo un seguimiento de todos los cambios que se han guardado en el repositorio. Si algo va mal, podrás volver a la versión anterior.

### **Documentación de proyectos**

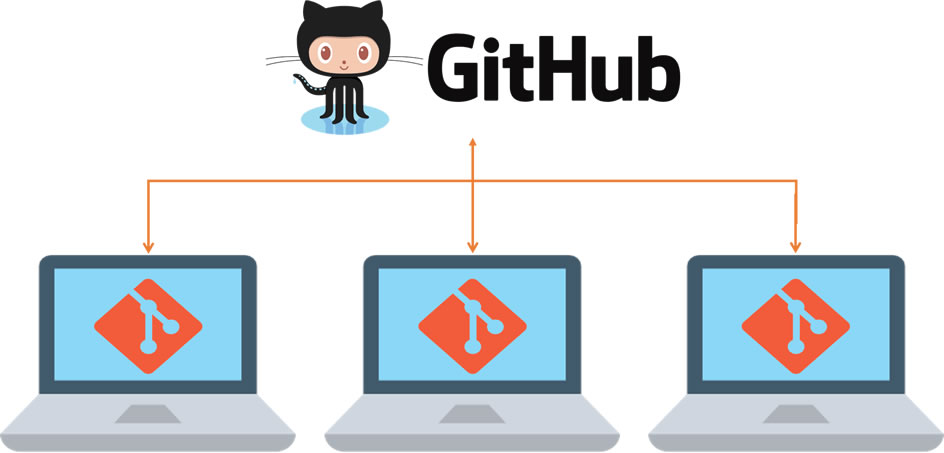
GitHub permite documentar y dar seguimiento a todo lo que ocurre en la cocina de una aplicación. Desde agregar comentarios a los códigos, hasta proporcionar guías con procedimientos específicos para implementación, fix de problemas e información que sea considerada relevante.

### **No es solo para desarrolladores**

Pese a que en mayoría lo utilizan los desarrolladores y la industria del software en general, GitHub permite almacenar cualquier tipo de archivo y a veces es más cómodo para compartir una aplicación o documento público o privado. Un ejemplo es una compañía que comercializa un software de gestión, y pone a disposición cada versión para que sus clientes la descarguen, incluyendo documentación visible, pasos para instalar, y demás.

### **Es seguro**

Tus códigos privados estarán protegidos al máximo de manera que solo tú seas quien decida a quien dar acceso para poder ver o editar sin que se filtre nada accidentalmente. GitHub mantiene los más altos estándares de seguridad.



## ****Comandos esenciales de Git****

Ahora que conocemos más acerca de GitHub, si eres de los valientes, veamos cuáles son los comandos más esenciales para trabajar con él. Git trabaja a modo de repositorio local, y al finalizar cada cambio podrás guardar en el repositorio remoto de GitHub. Si te gusta más trabajar con entornos gráficos, más abajo te mostraremos tres opciones para que puedas trabajar con Git más fácil.

### **Git init**

Este comando crea el repositorio Git en local para que puedas empezar a trabajar

git init

### **Git add**

Utiliza git add para agregar archivos al índice del proyecto. Este comando se utiliza agregando al final el nombre del archivo con un espacio al final del comando como lo ves en el ejemplo debajo.

git add archivo.php

### **Git commit**

Guarda los cambios realizardos utilizando git commit. Al guardar podrás especificar un mensaje para identificar el cambio que realizaste o agregar un comentario.

git commit -m "mensaje de referencia"

### **Git branch**

Este comando se usa para listar, crear o borrar ramas. Para listar todas las ramas utiliza:

git branch

### **Git stash**

Utiliza git stash para guardar cambios que son temporales.

git stash

### **Git log**

Con git log podrás ver una lista de los commits realizados.

git log

### **Git tag**

Se utiliza para etiquetar commits específicos, por ejemplo:

git tag 1.0.1 <id de commit>

### **Git diff**

Usa git diff para ver la lista de conflictos existentes.

git diff

### **Git merge**

Este comando se usa para fusionar un branch activo con otro.

git merge <nombre del branch>

### **git remote add**

Agrega un repositorio remoto.

git remote add origin https://github.com/user/repo.git

### **Git clone**

Este comando se usa con el propósito de revisar repositorios.

git clone

### **Git push**

Envía los cambios que se han realizado.

git push origin master

### **Git pull**

Fusionar todos los cambios que se han hecho en el repositorio local.

git pull

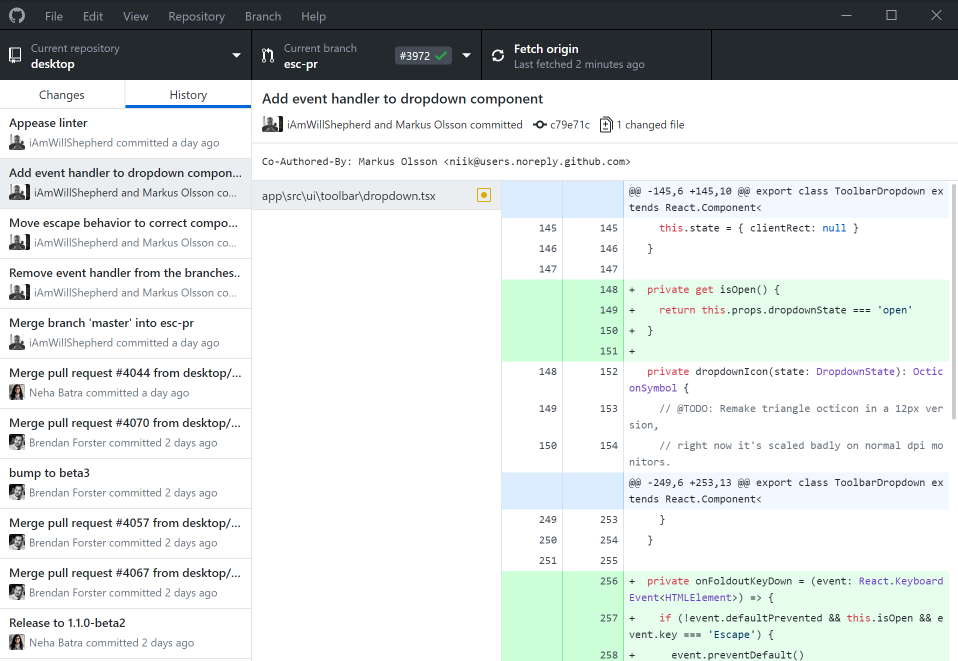
### **Git fetch**

Busca todos los recursos de un repositorio remoto que actualmente no están en el directorio local.

git fetch origin

## ****Entornos gráficos para Git****

Habrás notado que trabajar Git por consola, más si estás iniciándote, puede ser engorroso. Es por eso que te propondremos estas 3 aplicaciones para simplificar tu trabajo y conectar con tus repos en GitHub.



### **Github Desktop**

[**Github Dekstop**](https://desktop.github.com/) es la aplicación de escritorio más elegida para trabajar online con tus repositorios online, extendiendo el flujo de trabajo más allá de la consola y el navegador web. Lo mejor de todo es que es nativa de GitHub y es compatible con PC y Mac.

### **Visual Studio Code**

[**Visual Studio Code**](https://code.visualstudio.com/) es la aplicación de Microsoft que te permitirá codear tus aplicaciones, con comandos Git integrados, listos para usar desde el editor.

### **Sourcetree**

[**Sourcetree**](https://www.sourcetreeapp.com/) pertenece al fabricante de software Atlassian, es gratis y limpia a la vista. Definida por el desarrollador como simple para el principiante, y potente para usuarios de nivel experto.

## ****Conclusión****

Si perteneces a un equipo de desarrollo, o estás por desarrollar una aplicación en grupo, Github aliviará tus tareas de control, edición y documentación entre los miembros de tu team. Existen otras herramientas similares como GitLab o BitBucket. ¿Estás utilizando alguna de ellas? ¡Cuéntanos!