# # Control de versiones y colaboración con Git y Github

{#version-control-and-collaboration-with-git-and-github}

Este capítulo presenta una visión general del uso de Git para colaborar con otros. Puedes encontrar tutoriales más extensos al final, en la sección de Recursos.

## ¿Qué es Git?

Git es un software de **control de versiones** que permite seguir los cambios realizados en una carpeta. Se puede utilizar como la opción "control de cambios" de Word, LibreOffice o Google docs, pero para todo tipo de archivos. Es una de las opciones más potentes y más utilizadas para el control de versiones.

**¿Por qué nunca he oído hablar de Git? -** Mientras que las personas con formación como desarrollador aprenden habitualmente a utilizar un software de control de versiones (Git, Mercurial, Subversion u otros), a pocas personas de las disciplinas cuantitativas se nos enseñan estas habilidades. En consecuencia, la mayoría de profesionales de la epidemiología nunca hemos oído hablar sobre esto en sus estudios, y tenemos que aprenderlo sobre la marcha.

**Espera, he oído hablar de Github, ¿es lo mismo?** - No exactamente, pero a menudo se utilizan juntos, y veremos aquí cómo hacerlo. En resumen:

* **Git** es el sistema de control de versiones, una pieza de software. Se puede utilizar localmente en el ordenador o para sincronizar una carpeta con un **sitio web** anfitrión. Por defecto, se utiliza una ventana de terminal para escribir las instrucciones de Git en la línea de comandos.
* Se puede utilizar un **cliente/interfaz Git** para evitar la línea de comandos y realizar las mismas acciones (al menos para las más sencillas y supercomunes).
* Si se quiere almacenar una carpeta en un **sitio web** para colaborar con otros, se puede utilizar una cuenta en Github, Gitlab, Bitbucket u otros.

Se puede utilizar el cliente/interfaz **Github Desktop**, que utiliza **Git** en segundo plano para gestionar los archivos, tanto localmente en el ordenador, como remotamente en un servidor de **Github**.

## ¿Por qué utilizar la combinación de Git y Github?

El uso de **Git** facilita:

1. Almacenar versiones de archivos con cambios incrementales de forma que permite volver fácilmente a cualquier estado anterior
2. Mantener ramas paralelas, es decir, versiones de desarrollo/"trabajo" que más adelante pueden integrar los cambios después de su revisión

Esto también se puede hacer localmente en tu ordenador, incluso si no colaboras con otras personas. Alguna vez ….:

* ¿te has arrepentido de haber eliminado una sección de código, para darte cuenta dos meses después de que realmente la necesitabas?
* ¿has vuelto a un proyecto que había estado en pausa e intentado recordar si habías hecho esa complicada modificación en uno de los modelos?
* ¿tenías un archivo modelo\_1.R y otro archivo modelo\_1\_prueba.R y un archivo modelo\_1\_no\_funciona.R para probar las cosas?
* ¿tenías un archivo report.Rmd, un archivo report\_full.Rmd, un archivo report\_true\_final.Rmd, un archivo report\_final\_20210304.Rmd, un archivo report\_final\_20210402.Rmd y maldecías tus habilidades de almacenamiento?

Git puede ayudar con todo eso, y vale la pena aprenderlo sólo por eso.

Sin embargo, se vuelve aún más potente cuando se utiliza con un repositorio en línea como Github para apoyar **proyectos de colaboración**. Esto facilita:

* Colaboración: otros pueden revisar, comentar y aceptar o rechazar los cambios
* Compartir el código, los datos y los resultados, e invitar a hacer comentarios al público (o en privado, con tu equipo)

y evitar:

* "Uy, me olvidé de enviar la última versión y ahora tienes que rehacer el trabajo de dos días en este nuevo archivo"
* Mina, Henry y Oumar trabajaron al mismo tiempo en un script y necesitan fusionar manualmente sus cambios
* Dos personas intentan modificar el mismo archivo en Dropbox y Sharepoint y esto crea un error de sincronización.

### Esto suena complicado, yo no soy un programador

Puede ser. Los ejemplos de usos avanzados pueden ser bastante aterradores. Sin embargo, al igual que ocurre con R, o incluso con Excel, no es necesario convertirse en un experto para aprovechar las ventajas de la herramienta. El aprendizaje de un pequeño número de funciones y nociones te permite seguir sus cambios, sincronizar los archivos en un repositorio en línea y colaborar con los colegas en muy poco tiempo.

Debido a la curva de aprendizaje, el contexto de emergencia puede no ser el mejor momento para aprender estas herramientas. Pero el aprendizaje puede hacerse por pasos. Una vez que adquieras un par de nociones, tu flujo de trabajo puede ser bastante eficiente y rápido. Si no estás trabajando en un proyecto en el que la colaboración con personas a través de Git sea una necesidad, ... **en realidad es un buen momento para adquirir confianza en su uso** en solitario antes de sumergirte en ello en un proyecto colaborativo.

## Configuración

### Instalar Git

Git es el motor que está de este control de cambios la computadora; rastrea los cambios, las ramas (versiones), las fusiones y las reversiones. **Primero debes instalar Git desde https://git-scm.com/downloads.**

### Instalar una interfaz gráfica (opcional pero recomendable)

Git tiene su propio lenguaje de comandos, que se pueden escribir en la línea de comandos de un terminal. Sin embargo, hay muchos clientes/interfaces que proporcionan una buena visualización de las modificaciones de archivos o ramas. Esto es recomendable ya que personas que no son desarrolladoras, en su uso diario, rara vez necesitarán interactuar directamente con Git.

Existen muchas opciones, en todos los sistemas operativos, desde las amigables para los principiantes hasta las más complejas. Unas buenas opciones para principiantes son el panel Git de RStudio y [Github Desktop](https://desktop.github.com/), que mostraremos en este capítulo. Las opciones intermedias (más potentes, pero más complejas) incluyen Source Tree, Gitkracken, Smart Git y otras.

Explicación rápida sobre [los clientes Git](https:/happygitwithr.com/git-client.html#git-client).

Nota: dado que todas las interfaces utilizan Git internamente, puedes probar varias de ellas, cambiar de una a otra en un proyecto determinado, utilizar la consola puntualmente para una acción que tu interfaz no soporta, o incluso realizar una serie de acciones online en Github.

Como se indica más adelante, es posible que ocasionalmente tengas que escribir comandos Git en un terminal como en la pestaña “terminal” de RStudio (una pestaña adyacente a la consola de R) o la aplicación de terminal Git Bash.

### Cuenta de Github

Regístrate para obtener una cuenta gratuita en [github.com](../../../../../C:/Users/Neale/OneDrive%20-%20Neale%20Batra/Documents/Analytic%20Software/R/Projects/R%20handbook/Epi_R_handbook/github.com).

Es posible que se te ofrezca configurar la autenticación de dos pasos con una aplicación en tu teléfono. Lee más en [los documentos de ayuda](https://docs.github.com/es/github/authenticating-to-github/securing-your-account-with-two-factor-authentication-2fa) de Github.

Si usas Github Desktop, puedes introducir tus credenciales de Github después de la instalación siguiendo estos [pasos](https://docs.github.com/es/desktop/installing-and-configuring-github-desktop/authenticating-to-github). Si no lo haces, las credenciales se te pedirán más tarde cuando intentes clonar un proyecto desde Github.

## Vocabulario, conceptos y funciones básicas

Al igual que cuando se aprende R, hay que recordar un poco de vocabulario para entender Git. Aquí están los [conceptos básicos para empezar](https://www.freecodecamp.org/news/an-introduction-to-git-for-absolute-beginners-86fa1d32ff71/) / [tutorial interactivo](https:/learngitbranching.js.org). En las próximas secciones, mostraremos cómo usar las interfaces, pero es bueno tener el vocabulario y los conceptos en mente, para construir tu modelo mental, ya que lo necesitarás cuando más tarde, aunque uses las interfaces de los programas.

### Repositorio

Un repositorio Git ("repo") es una carpeta que contiene todas las subcarpetas y archivos de tu proyecto (datos, código, imágenes, etc.) y sus historiales de revisión. Cuando empieces a seguir los cambios en el repositorio con él, Git creará una carpeta oculta que contiene toda la información de seguimiento. Un repositorio típico de Git es la carpeta de tu proyecto R (ver la página del manual sobre [proyectos R](#r-projects)).

Mostraremos cómo crear (inicializar) un repositorio Git desde Github, Github Desktop o Rstudio en las siguientes secciones.

### Commits (Consolidaci**ones**)

Cuando realices un cambio en el proyecto, hay que ejecutar commit para consolidar estos cambios (el delta) realizados en tus archivos. Por ejemplo, quizás hayas editado algunas líneas de código y actualizado unos datos relacionados. Una vez guardados los cambios, puedes agrupar y confirmar estos cambios en un solo "commit".

Cada consolidación (commit) tiene un ID único (un hash). Para el control de versiones, puedes revertir tu proyecto hacia atrás en base a estas Consolidaciones, así que es mejor mantenerlas relativamente pequeñas y coherentes. También realizarás una breve descripción de los cambios llamada "commit message (mensaje de consolidación)". En cierto modo, cada commit es una **instantánea** del proyecto en un momento dado.

¿Cambios por etapas (staged)? Poner etapas en los cambios es añadirlos a la zona de preparación para la siguiente consolidación. La idea es que puedas decidir con precisión qué cambios incluir en un determinado commit. Por ejemplo, si trabajas en la especificación del modelo en un script, y más tarde en una figura en otro script, tendría sentido tener dos commits diferentes (sería más fácil en caso de que quisieras revertir los cambios en la figura pero no en el modelo).

### Ramas (Branchs)

Una rama representa una línea independiente de cambios en su repo, una versión paralela y alternativa de los archivos del proyecto.

Las ramas son útiles para probar los cambios antes de incorporarlos a la rama principal (main, master), que suele ser la versión primaria/final/"viva" de tu proyecto. Cuando termines de experimentar en una rama, puedes incorporar los cambios a tu rama principal, fusionándola, o eliminarla, si los cambios no fueron tan exitosos.

Nota: no es necesario colaborar con otras personas para utilizar las ramas, ni es necesario tener un repositorio remoto en línea.

### Repositorios locales y remotos

Clonar es crear una copia de un repositorio Git en otro lugar.

Por ejemplo, puedes clonar un repositorio online de Github localmente en tu ordenador, o empezar con un repositorio local y clonarlo online en Github.

Cuando se ha clonado un repositorio, los archivos del proyecto existen en dos lugares:

* el repositorio LOCAL en el ordenador físico. Aquí es donde se hacen los cambios reales a los archivos/código.
* el repositorio REMOTO, en línea: las versiones de los archivos del proyecto en el repositorio Github (o en cualquier otro alojamiento web).

Para sincronizar estos repositorios, utilizaremos más funciones. En efecto, a diferencia de Sharepoint, Dropbox u otro software de sincronización, Git no actualiza automáticamente el repositorio local en base a lo que está en línea, o viceversa. Tú eliges cuándo y cómo sincronizarlo.

* git fetch descarga los cambios realizados en el repositorio remoto pero no cambia el repositorio local. Piensa en ello para una comprobación del estado del repositorio remoto.
* git pull descarga archivos cambiados en los repositorios remotos y actualiza el repositorio local.
* Cuando hayas hecho uno o varios commits localmente, puedes hacer git push de los commits al repositorio remoto. Esto envía tus cambios a Github para actualizar el repositorio y que otras personas puedan verlos y extraerlos si lo desean.

## Empezar: crear un nuevo repositorio

Hay muchas formas de crear nuevos repositorios. Puedes hacerlo desde la consola/terminal, desde Github, desde una interfaz gráfica, como Github Desktop o Rstudio->Git.

Hay dos enfoques generales para la puesta en marcha:

* Crear un nuevo proyecto R a partir de un repositorio de Github existente o nuevo (preferible para los principiantes), o
* Crear un repositorio Github para un proyecto R existente

### Archivos de inicio

Cuando se crea un nuevo repositorio, se pueden crear todos los archivos de más abajo, o bien, pueden añadirse al repositorio en una etapa posterior. Normalmente estarán en la carpeta "raíz" del repositorio.

* Un archivo README es un archivo que alguien puede leer para entender por qué existe tu proyecto y qué más deben saber para usarlo. Al principio estará vacío, pero deberías completarlo más adelante.
* Un archivo .gitignore es un archivo de texto donde cada línea contendría carpetas o archivos que Git debería ignorar (no rastrear los cambios). Lee más sobre esto y mira ejemplos [aquí](https://www.freecodecamp.org/news/gitignore-what-is-it-and-how-to-add-to-repo/).
* Puedes elegir un tipo de licencia para el trabajo, de modo que otras personas sepan en qué condiciones pueden utilizar o reproducir tu obra. Para más información, consulta las [licencias Creative Commons](https://creativecommons.org/licenses/).

### Crear un nuevo repositorio en Github

Para crear un nuevo repositorio, entra en Github y busca el botón verde para crear un nuevo repositorio. Este repositorio, ahora vacío, puede ser clonado localmente en tu ordenador (ver la siguiente sección).

Debes elegir si quieres que tu repositorio sea **público** (visible para todo el mundo en Internet) o **privado** (sólo visible para aquellos con permiso). Esto tiene importantes implicaciones si tus datos son sensibles. Si tu repositorio es privado te encontrarás con algunos límites en circunstancias especiales avanzadas, como por ejemplo si estás usando actions de Github para ejecutar automáticamente tu código en la nube.

### Clonar desde un repositorio de Github

Puedes clonar un repositorio de Github existente para crear un nuevo proyecto R local en tu ordenador.

El repositorio de Github puede ser uno que ya existe y tiene contenido, o puede ser un repositorio vacío que acabas de crear. En este último caso, básicamente estás creando el repositorio de Github y el proyecto local de R al mismo tiempo (ver las instrucciones anteriores).

Nota: si no tienes derechos de contribución en un repositorio de Github, es posible primero bifurcar (fork) el repositorio hacia tu perfil, y luego proceder con las otras acciones. La bifurcación se explica al final de este capítulo, pero recomendamos que leas primero las otras secciones.

Paso 1: Navega en Github hasta el repositorio, clica en el botón verde "**Code**" y copia la **URL del clon HTTPS** (ver imagen inferior)

El siguiente paso se puede realizar en cualquier interfaz. Lo ilustraremos con Rstudio y Github desktop.

#### En Rstudio

En RStudio, inicia un nuevo proyecto R clicando en File>New project > Version control > Git) (Archivo > Nuevo proyecto > Control de versiones > Git)

* Cuando te pida la "URL del repositorio", pega la URL HTTPS de Github
* Asigna al proyecto R un nombre corto e informativo
* Elige dónde se guardará el nuevo proyecto R localmente
* Marca "Abrir en una nueva sesión" y clica en "Crear proyecto".

Ahora estás en un nuevo proyecto local de RStudio que es un clon del repositorio de Github. Este proyecto local y el repositorio de Github están ahora vinculados.

#### En Github Desktop

* Clica en File>Clone repository (Archivo > Clonar un repositorio)
* Selecciona la pestaña URL
* Pegua la URL HTTPS de Github en la primera casilla
* Selecciona la carpeta en la que deseas tener tu repositorio local
* Clica en "CLONE"

### Nuevo repositorio de Github a partir de un proyecto R existente

Un escenario alternativo de configuración es que ya tengas un proyecto R con contenido, y quieras crear un repositorio Github para él.

1. Crear un nuevo repositorio de Github vacío para el proyecto (ver instrucciones anteriores)
2. Clona este repositorio localmente (ver las instrucciones de HTTPS más arriba)
3. Copia todo el contenido de tu proyecto R preexistente (códigos, datos, etc.) en este nuevo repositorio local vacío (por ejemplo, utiliza copiar y pegar).
4. Abre tu nuevo proyecto en RStudio, y ve al panel Git. Los nuevos archivos deberían registrarse como cambios de archivo, ahora rastreados por Git. Por lo tanto, puedes agrupar estos cambios bajo un commit y empujarlos a Github. Una vez empujado, el repositorio en Github reflejará todos los archivos.

Consulta la sección de flujo de trabajo de Github para obtener más detalles sobre este proceso.

### ¿Qué aspecto tiene ahora?

#### En RStudio

Una vez que hayas clonado un repositorio de Github a un nuevo proyecto R, ahora verás en RStudio una pestaña "Git". Esta pestaña aparece en el mismo panel de RStudio que Environment:

Ten en cuenta los botones marcados con un círculo en la imagen anterior, ya que se hará referencia a ellos más adelante (de izquierda a derecha):

* Botón commit para consolidar los cambios del archivo guardado en local (se abrirá una nueva ventana para añadir la descripción y confirmarlo)
* Flecha azul para tirar (actualiza tu versión local de la rama con los cambios realizados en la versión remota/Github de esa rama)
* Flecha verde para empujar (enviar cualquier commits/cambio de tu versión local de la rama a la versión remota/Github de esa rama)
* La pestaña Git en RStudio
* Botón para crear una rama NUEVA dependiente de la rama que se muestra a la derecha como base. Casi siempre querrá bifurcarse desde la rama principal (después de haber tirado primero para actualizar la rama principal)
* La Rama en la que trabaja actualmente
* A continuación aparecerán los cambios que haya realizado en el código o en otros archivos

#### En Github Desktop

Github Desktop es una aplicación independiente que te permite gestionar todos tus repositorios. Cuando la abres, la interfaz te permite elegir el repositorio en el que quieres trabajar, y luego realizar acciones básicas de Git desde allí.

## Flujo de trabajo Git + Github

### Resumen del proceso

Una vez que hayas completado la configuración (descrita anteriormente), tendrás un repo de Github que está conectado (clonado) a un proyecto local de R. La rama principal (main por defecto) es la llamada versión "viva" de todos los archivos. Cuando quieras hacer modificaciones, es una buena práctica crear una nueva rama a partir de la rama principal (como "Make a Copy"). Este es un flujo de trabajo típico en Git porque crear una rama es fácil y rápido.

Un flujo de trabajo típico es el siguiente:

1. Asegúrate de que tu repositorio local está actualizado, actualízalo si no es así
2. Ve a la rama en la que estabas trabajando anteriormente, o crea una nueva rama para probar algunas cosas
3. Trabaja en los archivos localmente en tu ordenador, haz uno o varios commits en esta rama
4. Actualiza la versión remota de la rama con tus cambios (push)
5. Cuando estés satisfecho con tu rama, puedes fusionar la versión en línea de la rama de trabajo con la rama "principal" en línea para transferir los cambios

Otros miembros del equipo pueden estar haciendo lo mismo con sus propias ramas, o quizás contribuyendo con commits en su rama de trabajo también.

A continuación, repasamos el proceso anterior paso a paso con más detalle. Es un esquema que hemos desarrollado - está en el formato de una tabla de dos x dos, por lo que debería ayudarnos a entenderlo.

Aquí hay [otro diagrama](https://build5nines.com/introduction-to-git-version-control-workflow/).

Nota: hasta hace poco, se utilizaba el término rama "master" (maestra), pero ahora se denomina rama "main" (principal).

[Fuente](https://build5nines.com/introduction-to-git-version-control-workflow/) de la imagen

## Crear una nueva rama

Cuando seleccionas una rama para trabajar, **Git restablece tu directorio de trabajo tal y como estaba la última vez que estuviste en esta rama**.

### En el panel Git de Rstudio

Asegúrate que te encuentras en la rama “main” (master, principal) y, a continuación, clica en el icono morado para crear una nueva rama (véase la imagen anterior).

* Pedirá un nombre descriptivo para esa rama, de una palabra (se pueden usar barras bajas si es necesario).
* Verás que localmente, sigues en el mismo proyecto R, pero ya no estás trabajando en la rama "principal".
* Una vez creada, la nueva rama también aparecerá en el sitio web de Github como una rama.

Puedes visualizar las ramas en el panel Git de Rstudio tras clicar en "Historial"

### En Github Desktop

El proceso es muy similar, se pide que dé un nombre a su rama. Después, pedirá que "publique su rama en Github" para que la nueva rama aparezca también en el repositorio remoto.

### En la consola

Lo que realmente ocurre entre bastidores es que creas una nueva rama con git branch, y luego vas a la rama con git checkout (es decir, le dices a Git que tus próximos commits se producirán allí). Desde tu repositorio git:

Para más información sobre el uso de la consola, consulta la sección sobre comandos Git al final.

## Consolidar los cambios (Commit)

Ahora puedes editar el código, añadir nuevos archivos, actualizar conjuntos de datos, etc.

Cada uno de tus cambios es rastreado, una vez que el archivo respectivo es guardado. Los archivos modificados aparecerán en la pestaña Git de RStudio, en Github Desktop, o utilizando el comando git status en el terminal (ver más abajo).

Siempre que hagas cambios sustanciales (por ejemplo, añadir o actualizar una sección de código), haz una pausa y consolida esos cambios (Commit). Piensa en una Consolidación como un "lote" de cambios relacionados con un propósito común. Siempre puedes seguir revisando un archivo después de haber confirmado los cambios en él.

Consejo sobre los commits: en general, es mejor hacer Consolidaciones pequeñas, que puedan revertirse fácilmente si surge un problema, para Consolidar juntas modificaciones relacionadas con un propósito común. Para lograr esto, verás que debes hacer commits a menudo. Al principio, es probable que te olvides de hacer commits a menudo, pero luego el hábito se impone.

### En Rstudio

El ejemplo siguiente muestra que, desde la última consolidación, el script de R Markdown "collaboration.Rmd" ha cambiado, y se han añadido varias imágenes PNG.

Puede que te preguntes qué representan los cuadrados amarillo, azul, verde y rojo que aparecen junto a los nombres de los archivos. Aquí hay una captura de la [hoja de trucos de RStudio](https://www.rstudio.com/wp-content/uploads/2016/01/rstudio-IDE-cheatsheet.pdf) que explica su significado. Ten en cuenta que los cambios con el amarillo "?" aún pueden ser puestos en escena, confirmados y enviados.

* Clica el botón "Commit" en la pestaña Git, que abrirá una nueva ventana (mostrada a continuación)
* Clica en un nombre de archivo en el cuadro superior izquierdo
* Revisa los cambios que ha realizado en ese archivo (resaltados en verde o rojo)
* "Poner en etapas" el archivo (stage), lo que incluirá esos cambios en la consolidación. Para ello, marca la casilla situada junto al nombre del archivo. También puedes marcar varios nombres de archivo y clicar en "Stage".
* Escribe un mensaje de consolidación breve pero descriptivo (obligatorio)
* Clica el botón "Confirmar". Aparecerá un cuadro emergente mostrando el éxito o un mensaje de error.

Ahora puedes hacer más cambios y más commits, tantas veces como quieras

### En Github Desktop

Puedes ver la lista de los archivos que se han modificado a la izquierda. Si seleccionas un archivo de texto, verás en el panel derecho un resumen de las modificaciones que se han hecho (la vista no funcionará en archivos más complejos como .docs o .xlsx).

Para añadir los cambios, basta con marcar la pequeña casilla situada junto a los nombres de los archivos. Cuando hayas seleccionado los archivos que quieres añadir a esta consolidación, dale un nombre a la consolidación, opcionalmente una descripción y luego clica en el botón de **commit**.

### En la consola

Las dos funciones que se utilizan entre bastidores son git add para seleccionar/poner en escena los archivos y git commit para hacer realmente el commit.

### Modificar una consolidación anterior

¿Qué sucede si confirmas algunos cambios, sigues trabajando y te das cuenta de que hiciste cambios que deberían "pertenecer" a la consolidación anterior (en tu opinión)? No temas! Puedes añadir estos cambios a tu consolidación anterior.

En Rstudio, debería ser bastante obvio, ya que hay una casilla "Amend previous commit" (modificar una consolidación anterior) en la misma línea que el botón COMMIT.

Por alguna razón poco clara, la funcionalidad no se ha implementado como tal en Github Desktop, pero hay una forma (conceptualmente incómoda pero fácil) de hacerlo. Si has confirmado **pero** aún **no has enviado** tus cambios, aparece un botón "UNDO" justo debajo del botón COMMIT. Clica en él y revertirá tu consolidación (pero mantendrá sus archivos en etapa y tu mensaje de consolidación). Guarda los cambios, añade nuevos archivos a la consolidación si es necesario y vuelva a confirmar.

En la consola:

Nota: piensa antes de modificar los commits que ya son públicos y compartidos con tus colaboradores.

## Sube y empuja los cambios a Github

"Primero tirar, luego empujar"

Es una buena práctica obtener y tirar antes de empezar a trabajar en tu proyecto, para actualizar la versión de la rama en tu equipo local con los cambios que se han hecho en la versión remota/Github.

Tira a menudo. No dudes. Tira siempre antes de empujar.

Cuando los cambios estén hechos y confirmados y estés contento con el estado de tu proyecto, puedes enviar tus consolidaciones a la versión remota/Github de tu rama.

Repite y repite mientras trabajas en el repositorio.

**Nota:** es mucho más fácil revertir los cambios que fueron confirmados pero no empujados (es decir, siguen siendo locales) que revertir los cambios que fueron empujados al repositorio remoto (y tal vez ya sacados por otra persona), por lo que es mejor empujar cuando haya terminado de introducir cambios en la tarea en la que estaba trabajando.

#### En Rstudio

TIRAR - En primer lugar, clica en el icono "Tirar" (flecha hacia abajo) que busca y tira al mismo tiempo.

PUSH – Clicando en el icono verde "Push" (flecha hacia arriba). Es posible que pida que introduzcas tu nombre de usuario y contraseña de Github. La primera vez que la pida, es posible que tenga que introducir dos líneas de comando Git en el Terminal:

* **git config -global user.email "you@example.com"** (tu dirección de correo electrónico de Github), y
* **git config -global user.name "Tu nombre de usuario de Github"**

Para saber más sobre cómo introducir estos comandos, consulta la sección siguiente sobre comandos Git.

**SUGERENCIA:** ¿Te piden la contraseña muy a menudo? Consulta los capítulos 10 y 11 de este [tutorial](https://happygitwithr.com/credential-caching.html" \l "credential-caching) para conectarse a un repositorio usando una clave SSH (más complicado)

#### En Github Desktop

Clica en el botón "Fetch origin" para comprobar si hay nuevos commits en el repositorio remoto.

Si Git encuentra nuevos commits en el repositorio remoto, el botón cambiará a un botón "Pull". Dado que el mismo botón se utiliza para empujar y tirar, no puedes empujar tus cambios si no tiras antes.

Puedes ir a la pestaña "Historial" (cerca de la pestaña "Changes") para ver todos los commits (los tuyos y los de los demás). Esta es una buena manera de conocer lo que hicieron tus colaboradores. Puedes leer el mensaje de consolidación, la descripción si la hay, y comparar el código de los dos archivos usando el panel de diferencias.

Una vez que se han extraído todos los cambios remotos y se ha consignado al menos un cambio local, se puede empujar clicando en el mismo botón.

#### Consola

Sin sorpresa, las órdenes son fetch, pull y push.

### Quiero tirar pero tengo trabajo local

Esto puede ocurrir a veces: has hecho algunos cambios en tu repositorio local, pero el repositorio remoto tiene consolidaciones que no has descargado.

Git se negará a hacer un pull porque podría sobrescribir tus cambios. Hay varias estrategias para guardar tus cambios, bien descritas en [Happy Git with R](https://happygitwithr.com/pull-tricky.html), entre las cuales las dos principales son - Confirmar tus cambios, obtener los cambios remotos, extraerlos, resolver los conflictos si es necesario (ver la sección más abajo), y empujar todo en línea - stash tus cambios, lo que en cierto modo los guarda a un lado, tirar, desalmacenar (restaurar), y luego confirmar, resolver cualquier conflicto, y empujar.

Si los archivos afectados por los cambios remotos y los archivos afectados por tus cambios locales no se solapan, Git puede resolver los conflictos automáticamente.

En Github Desktop, esto se puede hacer con botones. Para almacenar, ve a Branch > Stash all changes.

## Combinar la rama con la principal

Si has terminado de hacer cambios, puedes comenzar el proceso de fusión de esos cambios en la rama principal. Dependiendo de su situación, esto puede ser rápido, o puede tener pasos deliberados de revisión y aprobación que involucren a compañeros de equipo.

### Localmente en Github Desktop

Se pueden fusionar ramas localmente usando Github Desktop. Primero, ve a (checkout) la rama que será la destinataria de los commits, es decir, la rama que quieres actualizar. A continuación, clica en el menú RBranch > Merge into current branch actual. Un cuadro te permitirá seleccionar la rama desde la que quieres importar.

### En la consola

Primero, vuelve a la rama que será la destinataria de los cambios. Normalmente es la rama maestra, pero puede ser otra rama. Luego fusiona tu rama de trabajo con la maestra.

[Esta página](https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging) muestra un ejemplo más avanzado de bifurcación y explica un poco lo que ocurre entre bastidores.

### En Github: envío de pull requests

Aunque es totalmente posible fusionar dos ramas localmente, o sin informar a nadie, una fusión puede ser discutida o investigada por varias personas antes de ser integrada en la rama maestra. Para ayudar en el proceso, Github ofrece algunas funciones de discusión en torno a la fusión: el **pull request**.

Un pull request (un "PR") es una solicitud para fusionar una rama con otra (en otras palabras, una solicitud para que tu rama de trabajo se incorpore a la rama "principal"). Una solicitud de extracción suele incluir varias consolidaciones. Un pull request suele iniciar un proceso de conversación y revisión antes de que sea aceptado y la rama sea fusionada. Por ejemplo, puedes leer las discusiones sobre pull requests en [el github de dplyr](https://github.com/tidyverse/dplyr/pulls).

Puedes enviar una solicitud de extracción (PR) directamente desde el sitio web (como se ilustra a continuación) o desde Github Desktop.

* Ir al repositorio Github (en línea)
* Ve a la pestaña "Pull Requests" y clica en el botón "New pull request".
* Selecciona en el menú desplegable para fusionar su rama en la principal
* Escribe un comentario detallado sobre la solicitud de extracción y clica en "Crear solicitud de extracción".

En la imagen siguiente, se ha seleccionado la rama "bosques" para fusionarla con la "principal":

Ahora se debería poder ver el pull request (imagen de ejemplo abajo):

* Revisa la pestaña "Archivos cambiados" para ver cómo cambiaría la rama "principal" si se fusionara la rama.
* A la derecha, puedes solicitar una revisión a los miembros de tu equipo etiquetando su ID de Github. Si quieres, puedes configurar el repositorio para que se requiera una revisión de aprobación para poder fusionarlo con el principal.
* Una vez aprobada la solicitud de extracción, se activará un botón para "fusionar la solicitud de extracción". Clica en él.
* Una vez completado, elimina tu rama como se explica a continuación.

### Resolución de conflictos

Cuando dos personas modifican la(s) misma(s) línea(s) al mismo tiempo, surge un conflicto de fusión. De hecho, Git se niega a tomar una decisión sobre qué versión mantener, pero te ayuda a encontrar dónde está el conflicto. **NO TE ASUSTES**. La mayoría de las veces, es bastante sencillo de resolver.

Por ejemplo, en Github:

Después de que la fusión haya planteado un conflicto, abre el archivo en tu editor favorito. El conflicto se indicará con una serie de caracteres:

El texto entre <<<<<<< HEAD y ======= proviene de tu repositorio local, y el que está entre ======= y >>>>>>> de la otra rama (que puede ser origin, master o cualquier rama de tu elección).

Tienes que decidir qué versión del código prefieres (o incluso escribir una tercera, incluyendo los cambios de ambas partes si es pertinente), borrar el resto y eliminar todas las marcas que Git ha añadido (<<<<<<< HEAD, =======, >>>>>>> origin/master/your\_branch\_name).

A continuación, guarda el archivo, estadíalo y haz un commit: este es el commit que hace que la versión fusionada sea "oficial". No te olvides de hacer push después.

Cuanto más a menudo tiren y empujen tú y tus colaboradores, menores serán los conflictos.

Nota: Si te sientes cómodo con la consola, existen [*opciones de fusión*](https://git-scm.com/book/en/v2/Git-Tools-Advanced-Merging) más [*avanzadas*](https://git-scm.com/book/en/v2/Git-Tools-Advanced-Merging) (por ejemplo, ignorar los espacios en blanco, dar prioridad a un colaborador, etc.).

### Borrar una rama

Una vez que una rama se ha fusionado con la maestra y ya no es necesaria, puedes eliminarla.

#### Github + Rstudio

Ve al repositorio en Github y clica en el botón para ver todas las ramas (junto al desplegable para seleccionar ramas). Ahora busca tu rama y clica en el icono de la papelera junto a ella. Lee más detalles sobre cómo eliminar una rama [aquí](https://docs.github.com/en/free-pro-team@latest/github/collaborating-with-issues-and-pull-requests/creating-and-deleting-branches-within-your-repository" \l "deleting-a-branch).

Asegúrate de eliminar también la rama localmente en tu ordenador. Esto no ocurrirá automáticamente.

* Desde RStudio, asegúrese de estar en la rama principal
* Cambie a escribir los comandos Git en la "Terminal" de RStudio (la pestaña adyacente a la consola de R), y escribe: **git branch -d nombre\_de\_rama**, donde "nombre\_de\_rama" es el nombre de la rama a eliminar
* Actualiza tu pestaña Git y la rama debería desaparecer

#### En Github Desktop

Sólo tienes que comprobar la rama que quieres eliminar, e ir al menú Rama > Eliminar.

### Bifurcación

Puedes hacer un fork de un proyecto si quieres contribuir a él pero no tienes los derechos para hacerlo, o si sólo quieres modificarlo para tu uso personal. Puedes encontrar una breve descripción de la bifurcación [aquí](https://guides.github.com/activities/forking/).

En Github, clica en el botón "Fork":

Esto clonará el repositorio original, pero en tu propio perfil. Así que ahora hay dos versiones del repositorio **en Github**: la original, que no puedes modificar, y la versión clonada en tu perfil.

Entonces, puedes clonar tu versión del repositorio en línea localmente en tu ordenador, utilizando cualquiera de los métodos descritos en las secciones anteriores. Luego, puede crear una nueva rama, hacer cambios, confirmarlos y empujarlos a tu repositorio remoto.

Una vez que estés contento con el resultado, puedes crear un Pull Request desde Github o Github Desktop para iniciar la conversación con los propietarios/mantenedores del repositorio original.

**¿Y si necesitas algunos commits más recientes del repositorio oficial?**

Imagina que alguien hace una modificación crítica en el repositorio oficial, que quieres incluir en tu versión clonada. Es posible sincronizar tu fork con el repositorio oficial. Implica usar el terminal, pero no es demasiado complicado. Principalmente necesitas recordar que - upstream = el repositorio oficial, el que no has podido modificar - origin = tu versión del repositorio en tu perfil de Github

Puedes leer [este tutorial](https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/syncing-a-fork) o seguirlo a continuación:

Primero, escribe en tu terminal Git (dentro de tu repo):

Si aún no ha configurado el repositorio upstream debería ver dos líneas, que comienzan por origen. Muestran el repositorio remoto al que apuntan fetch y push. Recuerda que origen es el apodo convencional para tu propia versión del repositorio en Github. Por ejemplo:

Ahora, añada un nuevo repositorio remoto:

Aquí la dirección es la que genera Github cuando clonas un repositorio (ver sección de clonación). Ahora tendrás cuatro punteros remotos:

Ahora que la configuración está hecha, siempre que quieras obtener los cambios del repositorio original (upstream), sólo tienes que ir (checkout) a la rama que quieres actualizar y teclear:

Si hay conflictos, tendrá que resolverlos, tal y como se explica en la sección Resolución de conflictos.

**Resumen**: forking es clonación, pero en el lado del servidor de Github. El resto de las acciones son las típicas del flujo de trabajo de colaboración (clonar, empujar, tirar, confirmar, fusionar, enviar solicitudes de extracción...).

Nota: aunque la bifurcación es un concepto, no un comando de Git, también existe en otros hosts web, como [*Bitbucket*](https://www.atlassian.com/git/tutorials/comparing-workflows/forking-workflow).

## Lo que hemos aprendido

Has aprendido a hacerlo:

* Configurar Git para rastrear las modificaciones en tus carpetas,
* Conectar tu repositorio local a un repositorio remoto en línea,
* Confirmar los cambios,
* Sincronizar tus repositorios locales y remotos.

Todo esto debería ayudar a ponerte en marcha y ser suficiente para la mayoría de tus necesidades de análisis epidemiológico. Normalmente no tenemos un uso tan avanzado como los desarrolladores.

Sin embargo, debes saber que si quieres (o necesitas) ir más allá, Git ofrece más potencia para simplificar los historiales de commit, revertir uno o varios commits, hacer cherry-pick de commits, etc. Algunas cosas pueden parecer pura magia, pero ahora que tienes los fundamentos, es más fácil construir sobre ellos.

Ten en cuenta que mientras el panel Git en Rstudio y Github Desktop son buenos para los principiantes / uso diario en nuestra línea de trabajo, no ofrecen una interfaz para algunas de las funciones intermedias / avanzadas de Git. Algunas interfaces más completas permiten hacer más cosas con apuntar y clicar (normalmente a costa de un diseño más complejo).

Recuerda que, dado que puedes utilizar cualquier herramienta en cualquier momento para realizar el seguimiento de tu repositorio, puedes instalar muy fácilmente una interfaz para probarla a veces, o para realizar alguna tarea compleja menos común ocasionalmente, mientras prefieres una interfaz simplificada para el resto del tiempo (por ejemplo, utilizando Github Desktop la mayor parte del tiempo, y cambiando a SourceTree o Gitbash para algunas tareas específicas).

## Comandos Git

### Aprendizaje recomendado

Para aprender los comandos de Git en un tutorial interactivo, consulta [este sitio web](https://learngitbranching.js.org/).

### Dónde introducir los comandos

Se introducen comandos en un shell Git.

Opción 1 Puedes abrir una nueva Terminal en RStudio. Esta pestaña está al lado de la Consola R. Si no puede escribir ningún texto en ella, clica en el menú desplegable debajo de "Terminal" y selecciona "Nueva terminal". Escriba los comandos en el espacio parpadeante delante del signo de dólar "$".

Opción 2 También puede abrir un shell (un terminal para introducir comandos) clicando en el icono azul de "engranajes" en la pestaña Git (cerca del entorno de RStudio). Selecciona "Shell" en el menú desplegable. Se abrirá una nueva ventana en la que puedes escribir los comandos después del signo de dólar "$".

Opción 3 Clica con el botón derecho para abrir "Git Bash aquí" que abrirá el mismo tipo de terminal, o abra Git Bash desde tu lista de aplicaciones. [Más información para principiantes sobre Git Bash](https://happygitwithr.com/shell.html), cómo encontrarlo y algunos comandos bash que necesitarás.

### Ejemplos de comandos

A continuación presentamos algunos comandos git comunes. Cuando los uses, ten en cuenta qué rama está activa (check-out), ¡ya que eso cambiará la acción!

En los comandos de abajo, representa un nombre de rama. representa el hash ID de un commit específico. representa un número. No escriba los símbolos < o >.

| **Comando Git** | **Acción** |
| --- | --- |
| git branch <nombre> | Crear una nueva rama con el nombre |
| git checkout <nombre> | Cambiar la rama actual a |
| git checkout -b <nombre> | Acceso directo para crear una nueva rama y cambiar a ella |
| estado de git | Ver los cambios no rastreados |
| git add <fichero> | Poner en escena un archivo |
| git commit -m <mensaje> | Confirmar los cambios actualmente en fase a la rama actual con el mensaje |
| git fetch | Obtener los commits del repositorio remoto |
| git pull | Tirar de los commits del repositorio remoto en la rama actual |
| git push | Empujar los commits locales al directorio remoto |
| interruptor git | Una alternativa a git checkout que se está introduciendo en Git |
| git merge <nombre> | Fusionar rama en la rama actual |
| git rebase <nombre> | Añadir los commits de la rama actual a la rama |

## Recursos

Gran parte de esta página fue informada por [este sitio web "Happy Git with R"](https://happygitwithr.com/) de Jenny Bryan. Hay una sección muy útil de este sitio web que te ayuda a solucionar errores comunes relacionados con Git y R.

La [documentación y la guía de inicio de Github.com](https://docs.github.com/en/github).

La [hoja de trucos de](https://www.rstudio.com/wp-content/uploads/2016/01/rstudio-IDE-cheatsheet.pdf) RStudio ["IDE"](https://www.rstudio.com/wp-content/uploads/2016/01/rstudio-IDE-cheatsheet.pdf) que incluye consejos sobre Git con RStudio.

<https://ohi-science.org/news/github-going-back-in-time>

**Comandos Git para principiantes**

Un [tutorial interactivo](../../../../../C:/Users/Neale/OneDrive%20-%20Neale%20Batra/Documents/Analytic%20Software/R/Projects/R%20handbook/Epi_R_handbook/learngitbranching.js.org) para aprender los comandos de Git.

https://www.freecodecamp.org/news/an-introduction-to-git-for-absolute-beginners-86fa1d32ff71/: bueno para aprender lo más básico para rastrear los cambios en una carpeta en tu propio ordenador.

Buenos esquemas para entender las ramas: <https://speakerdeck.com/alicebartlett/git-for-humans>

**Tutoriales que cubren tanto los temas básicos como los más avanzados**

<https://tutorialzine.com/2016/06/learn-git-in-30-minutes>

[https://dzone.com/articles/git-tutorial-commands-and-operations-in-git](https://rsjakob.gitbooks.io/git/content/chapter1.html) <https://swcarpentry.github.io/git-novice/>(curso corto) <https://rsjakob.gitbooks.io/git/content/chapter1.html>

El [libro Pro Git](https://git-scm.com/book/en/v2) se considera una referencia oficial. Aunque algunos capítulos están bien, suele ser un poco técnico. Probablemente sea un buen recurso una vez que hayas usado un poco Git y quieras aprender un poco más precisamente lo que sucede y cómo ir más allá.