

Los primeros pasos

Miguel Equihua

Xalapa, Ver., 21 de febrero, 2025

Markdown, Quarto y la ciencia abierta

Tan simple como escribir en la página en blanco con pequeñas marcas de intención. Se ha inventado varias veces y el columpio ha ido y venido entre el interés en ver y controlar las marcas directamente sobre el texto y la preferencia de ver sólo el producto terminado **WYSIWYG** (*lo que ves es lo que tienes*, aunque el marcado existe, pero lo hace la máquina por ti).

Figura 1: Libro de Irene Vallejo



El trabajo del científico, el ingeniero, el estudiante o el creador de contenidos transita por el proceso de escribir, almacenar y dar formato presentable a los documentos. La escritura da origen a la historia y deja atrás la prehistoria. Un largo proceso de evolución cultural apasionante que tiene un hermoso recuento en el libro de Irene Vallejo *El infinito en un Junco* (Fig. 1). Hemos explorado las rocas, la arcilla, los pigmentos vegetales y minerales, las pieles de animales y los tejidos vegetales para escribir. Hoy lo hacemos sobre el éter y nos apoyamos en procesos electromagnéticos.

Hacer esto ha involucrado a incontables inventores y lo hacemos recurriendo a herramientas y formatos que tienen registro de propiedad a nombre de lucrativas empresas particulares. Lo hacemos así y el hecho no nos merece ni un suspiro reflexivo sobre sus implicaciones.

A veces nos incomodan detalles o grandes fallas que obstaculizan la expresividad que requiere la escritura académica creativa. En general, la práctica actual se opone a lograr un flujo ágil y transparente a lo largo del proceso completo que involucra organizar los datos, analizarlos,

sintetizarlos y publicar los hallazgos obtenidos. Por la fuerza del hábito, y a pesar de inconvenientes, la mayoría de las revistas aún insisten en recibir textos en formato **docx**.

En el movimiento social que nos invita a reflexionar sobre la *ciencia abierta*, hay quienes sostengamos que el conocimiento y el proceso creativo que lo impulsa debe ser lo más libre posible. El talento y la sabiduría su núcleo. Sobre todo en áreas como la salud y la calidad del entorno ecológico en el que vivimos.

Markdown fue desarrollado en 2004 por **John Gruber** (Fig. 2). Ideó una manera de poner marcas de formato en un texto común y corriente (lo llamaremos *texto plano*). También construyó un programa de cómputo (lo escribió en el lenguaje **Perl**), para convertir los archivos ya marcados en *Markdown* a algo conveniente para que las computadoras nos los pudieran presentar a través de la **Web**. Hacer esto implica reconstruir y transformar el documento original a un nuevo formato, el **HTML** (*HyperText Markup Language*). Encontrarás ayuda sobre como usar *Markdown* en el menú de ayuda de **RStudio**: *Help → Markdown Quick Reference*.

Figura 2: John Gruber



He aquí uno de los grandes valores que busca el movimiento en favor de una *ciencia abierta*: **romper las barreras que limitan el acceso a los textos y a los datos**. El uso de *texto plano* para escribir y organizar archivos de datos tienen muchas ventajas. Para empezar se pueden leer prácticamente en cualquier dispositivo, independientemente de *sistema operativo* e intereses comerciales de los fabricantes. Los archivos escritos así han superado la *dura prueba del paso del tiempo* mejor que otros tipos de archivos.

El día de hoy empezaremos a utilizar la idea del *Markdown*. Producirás tus primeros archivos que serán legibles como texto plano y que a la vez estarán listos para ser producidos en una variedad de presentaciones que usualmente requerimos para nuestro propio registro de actividades y para interactuar con colegas o maestros. Además, de lo que hizo en su momento *Gruber*, ahora existen herramientas como **Pandoc**, que pueden convertir archivos desde *Markdown* a una variedad de otros formatos que seguramente serán de tu interés en algún momento. Otro de los valores de la *ciencia abierta*: **favorecer el reuso de los productos de información y conocimiento**

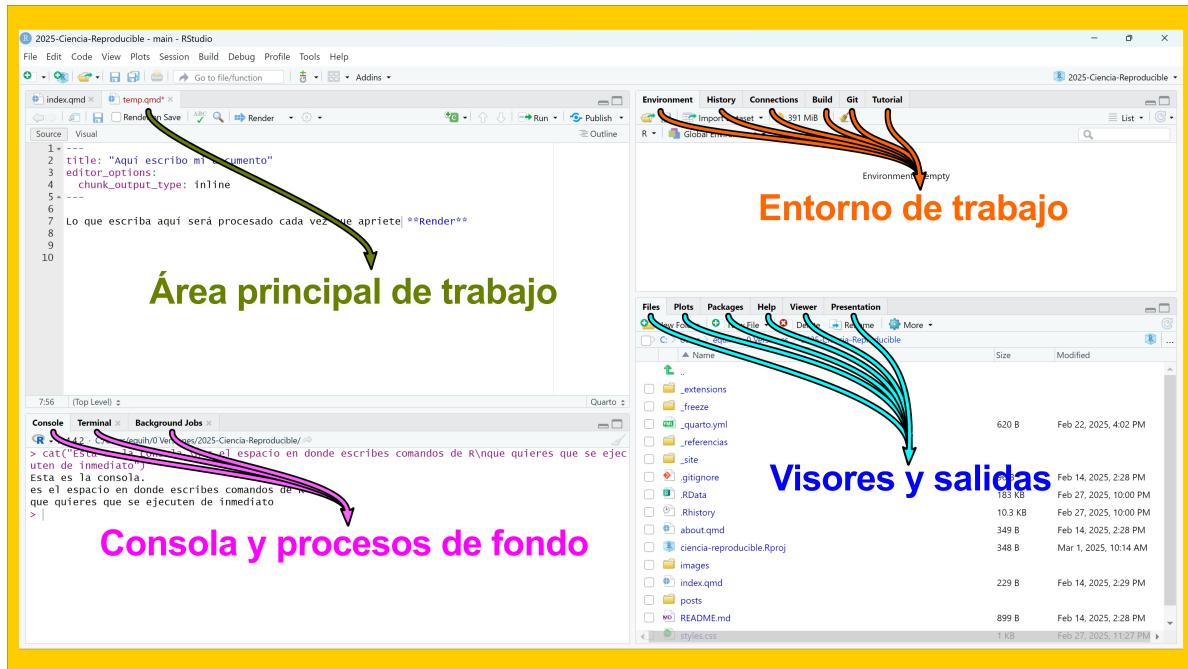
Quarto

Lo que haremos es:

1. Arrancar *RStudio*
2. Crear un nuevo proyecto
3. !!!Empezar a escribir!!!

RStudio

Es un *entorno integrado de desarrollo (IDE)*. Como tal, ofrece una estructura que permite interactuar con distintos aspectos del proceso creativo. En la figura puedes ver lo que usualmente tienes a tu alcance en una instalación común. Esta configuración la puedes modificar según tus preferencias desde el menú *View panes*. Pero probablemente al inicio te bastará con la configuración que ya tienes.



La gente de **Posit**, desarrolladores de *RStudio* está trabajando en la producción de versiones del programa en otros idiomas, aparte del inglés. En este momento existe una prueba en francés, pero no está listo nada en español. Mientras tanto, una ayuda parcial puede ser tener acceso a los reportes de error que dan los comandos en español, para eso deberás usar el comando siguiente en la ventana de *consola*

```
# Idioma español en sistema de codificación digital UTF-8
Sys.setLanguage(lang = "es.UTF-8")
```

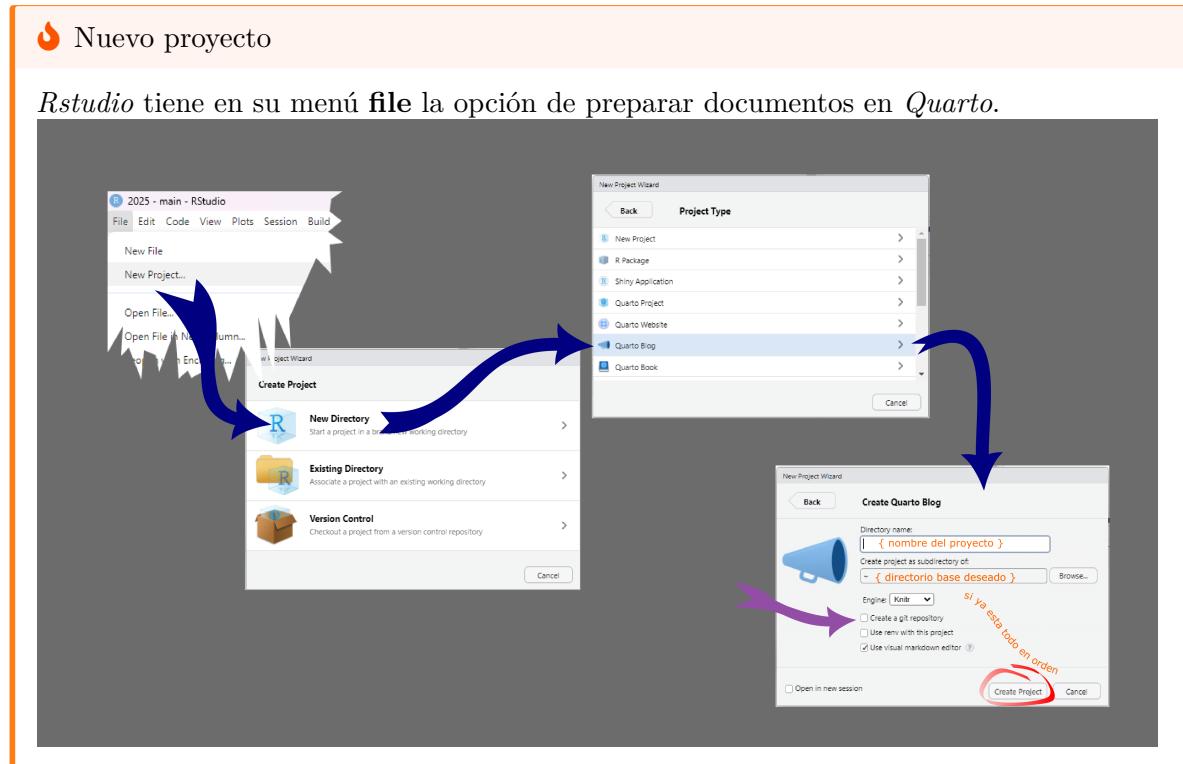
Esto puede ayudar, pero puede ser un problema si necesitas buscar ayuda en Internet, que las más de las veces está en inglés. Entonces, si quieres regresar a éste idioma, harás algo semejante

a lo de arriba, pero usarás el código apropiado, por ejemplo para inglés norteamericano: “en_us.UTF-8”

```
# Idioma inglés norteamericano en sistema de codificación digital UTF-8  
Sys.setLanguage(lang = "en_us.UTF-8")
```

Mi primer documento Quarto

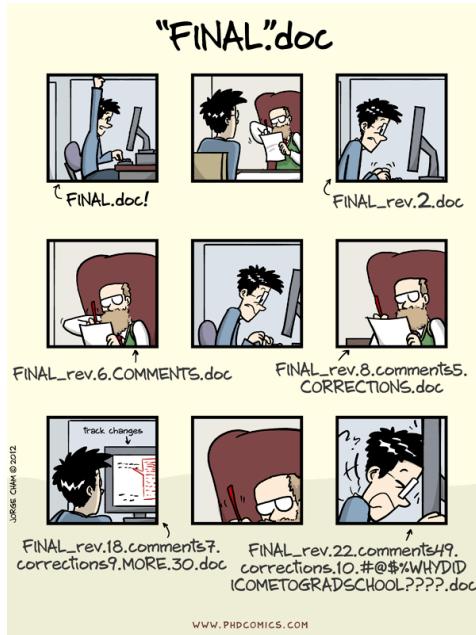
Todo listo. Anímate a escribir tus primeras líneas en un documento *Quarto*.



Guardar el trabajo con la intención de colaborar

Ahora ya tenemos el texto en nuestras máquinas, almacenado en casa. ¿Podemos hacer algo más para asegurar esos materiales y facilitar compartirlos con quienes queramos? Te sugiero considerar **git** y **github** para eso. Podemos imaginar que el espacio de almacenamiento en tu máquina es como una parcela de siembra, cada dato tiene coordenadas de localización y así los recuperas cuando los quieres. Lo que hace *git* es agregar una *ventana de tiempo* que te permite asomarte a la historia de lo que pasó en esas ubicaciones que te interesan.

Figura 3: Fuente: *Final.doc* en **Piled Higher and Deeper** por Jorge Cham,
<http://www.phdcomics.com>



¿Qué es Git?

Es una aplicación diseñada por el iniciador del desarrollo de Linux, **Linus Torvalds** (Fig. 4). **Git** es un sistema eficiente confiable y distribuido de control de versiones. El control de versiones es simplemente el seguimiento y registro de los cambios que va teniendo un documento a lo largo del tiempo. El concepto *distribuido* se refiere a que el registro local que tengas en tu máquina o para el caso en cualquier número de máquinas, es un registro completo, **clonado** del proyecto. Estos repositorios locales plenamente funcionales permiten trabajar aún cuando no tengas acceso a Internet. Los autores realizan y registran su trabajo localmente y, cuando lo encuentren conveniente, sincronizan la copia local del repositorio con la del servidor. En la actualidad *Git* se ha convertido en el estándar mundial *de facto* para el control de versiones.

Figura 4: Linus Torvalds



i Activar git

Para activar **git** en tu proyecto tienes dos opciones:

1. Hacerlo desde el principio marcando la casilla respectiva al momento de crear el proyecto.
2. Utilizar la biblioteca de herramientas auxiliares `usethis`.

Con este comando creas lo necesario para usar `git` en tu proyecto.

```
usethis::use_git()
```

En cualquier caso, ahora conviene verificar como está configurado el espacio de trabajo. En la ventana de **consola** puedes escribir los siguientes comandos para averiguar detalles de tu configuración.

Esto te dirá como se llama la *ventana de tiempo* que has elegido definir como base de trabajo, puedes tener tantas ramas distintas como consideres, pero conviene que una sea la principal. Se solía llamar a esta rama **master**, pero ahora se ha considerado que la !esclavitud ya ha sido abolida!, así que hay una tendencia a mejor llamarle **main**. En realidad puedes llamarla como quieras.

```
usethis::git_default_branch()
```

Si quieres configurar tu instalación de **RStudio** para que siempre defina la rama base como *main*, puedes usar el siguiente comando. Aunque esto sólo actuará para futuros proyectos, no cambiará nada en los que tienes ya creados hasta este momento.

```
usethis::git_default_branch_configure(name = "main")
```

Si lo que quieres es modificar la rama principal del proyecto con el que estas trabajando y que ya tienes abierto, este es el comando que te ayudará. En este ejemplo uso lo que es ya práctica común, migrar de *master* a *main*, pero puedes tomar tus propias preferencias sin ningún problema, aunque obviamente la parte **from** debe ser la existente que deseas modificar.

```
usethis::git_default_branch_rename(from = "master", to = "main")
```

No todos los archivos que están en el espacio de trabajo son realmente de interés como para seguir su historia en el tiempo y podría haber también cosas que nunca deberían estar registradas en un sistema que te expone al acceso público generalizado: claves personales, tokens, identificadores de archivos privados, etc. Aunque ante esto no hay mejor cosa que ser prudente y estar atentos, existe la función **vacunar** que busca ayudarte a evitar estos problemas. Para activar esta ayuda en tu proyecto puedes usar este comando.

```
usethis::git_vaccinate()
```

Esto pone ya en operación las capacidades de **git** en tu máquina. Para usarlas debes dirigirte a la pestaña respectiva. Con la función **Commit** generas el registro del estado de los archivos del proyecto al momento de activar el comando. Para operar esto debes decidir que archivos enviar al registro histórico, marcados como *staged*. Al apretar el botón **Commit** aparecerá una ventana en donde se reportan los detalles de lo que estas registrando. Cada **Commit** requiere anotar un mensaje descriptivo breve de lo que contiene el “corte”. Una vez que está todo resuelto, hay que apretar el botón **Commit** en esa pantalla y esperar algunos segundos a que termine el proceso de registro en la base de datos respectiva.

Enviar el *repositorio git* a la nube

Ahora estas preparada o preparado para enviar tu trabajo a *la nube*, lo haremos con el servicio de **Github**, aunque hay varias opciones (como **gitlab** por ejemplo).

Nuevamente nos ayudará **usethis** para hacer esto. Lo primero es que para comunicar **RStudio** con **Github** necesitas registrar un **token** de ese servicio en tu equipo. El comando para esto es:

```
usethis::create_github_token()
```

Esto te lleva a la página de **Github** en la que hay que generar el **token**. Hay que responder las preguntas que te haga la página, aunque todo estará *llenado* con lo normalmente necesario. Cuando esté a tu gusto, aprieta el botón respectivo. Aparecerá una nueva pantalla con el **token**. Este **token** que aparece, es la única vez que lo verás, por lo que conviene copiarlo al *portapapeles* de tu máquina (**ctrl-c** en windows) y tenerlo a buen resguardo por lo pronto. En seguida hay que ejecutar este otro comando en la consola de *RStudio*

```
gitcreds::gitcreds_set()
```

Si es la primera vez que registras un **token** te pedirá que lo registres, dale *paste* (**ctrl-v** en Windows). Si ya tienes un registro dado de alta, te informará sobre lo que tiene anotado y te dará oportunidad de decidir qué quieras hacer en seguida.

Todo está ya preparado, sólo falta poner en uso el vínculo que acabamos de crear. Para eso bastará con decir:

```
usethis::use_github()
```

Por cierto, este es el comando que necesitarán en lo sucesivo para vincular cualquier nuevo proyecto a tu cuenta de **Github**, siempre y cuando tu **token** este vigente.

Una vez terminadas estas tareas puedes ir a la pestaña `git` cuando lo consideres conveniente y ordenar a **RStudio** que envíe todos los **commits** que están pendientes hasta el momento a **Github**. Para hacerlo deberás apretar el botón **push**. Antes de hacerlo siempre es conveniente pedirle a `git` que se ponga al día con lo que ya está registrado en la *nube*: esto lo logras con el botón **pull**. Esto nos lleva a una rutina de operación con `git` que se resume en la figura siguiente.

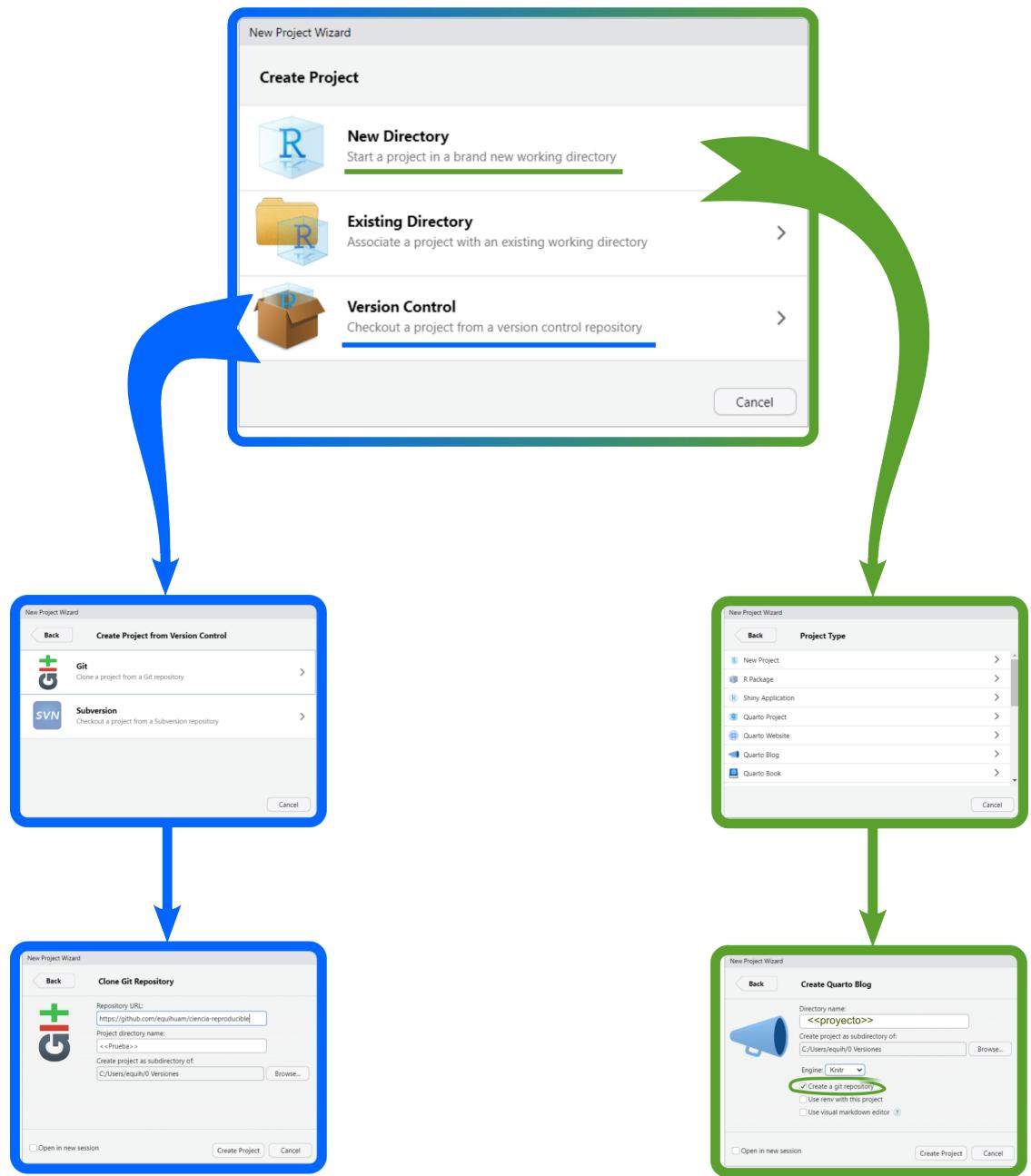
En caso de Incendio

- 1. `git commit`
- ↓ 2. `git pull`
- ↑ 3. `git push`
- ✖ 4. `git out`

💡 Resumen rutinario para usar `git`

Claro está que configurar todo la primera vez es un poco complicado, pero si todo está listo: `git` instalado, cuenta de Github, token activado, etc. la operación cotidiana es mucho más sencilla.

En la figura se ilustran las dos rutas para hacerlo en *RStudio*.



Evidentemente, si seguiste la ruta azul, tu repositorio ya existe en *Github*, una vez que hayas **clonado** el repositorio en tu máquina todo queda listo para concentrarte en escribir. Si optaste por la ruta verde, entonces deberás crear un nuevo repositorio en *Github*. Para hacerlo Utiliza **use this** en la pestaña de consola.

```
usethis::use_github()
```

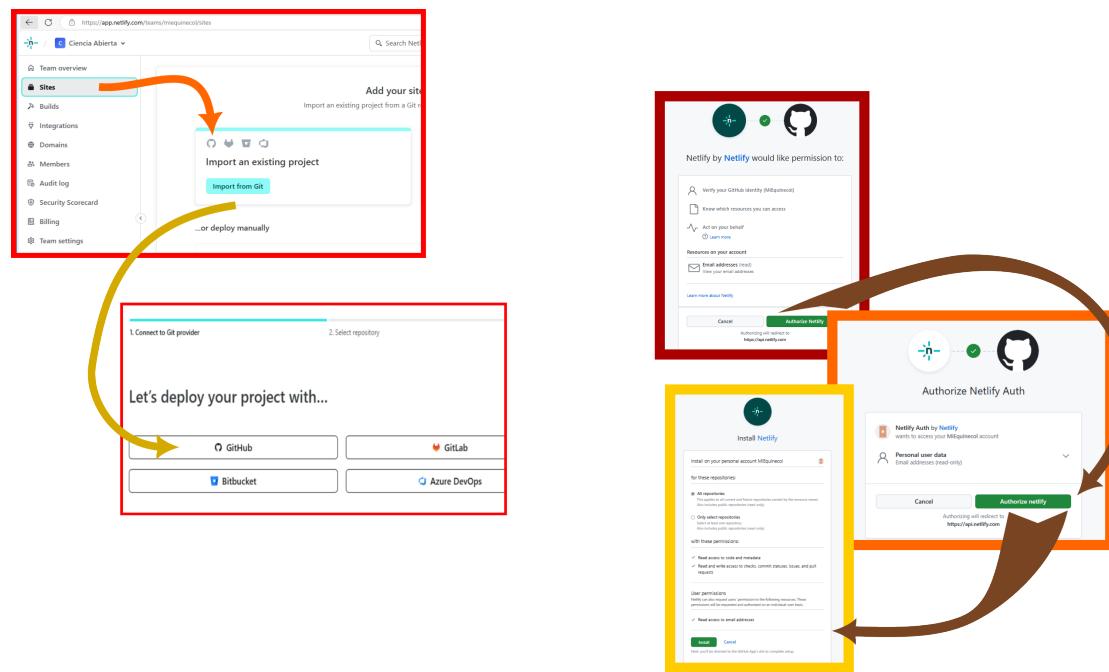
Eso es todo.

🔥 Publicar tu Blog con ayuda de Netlify

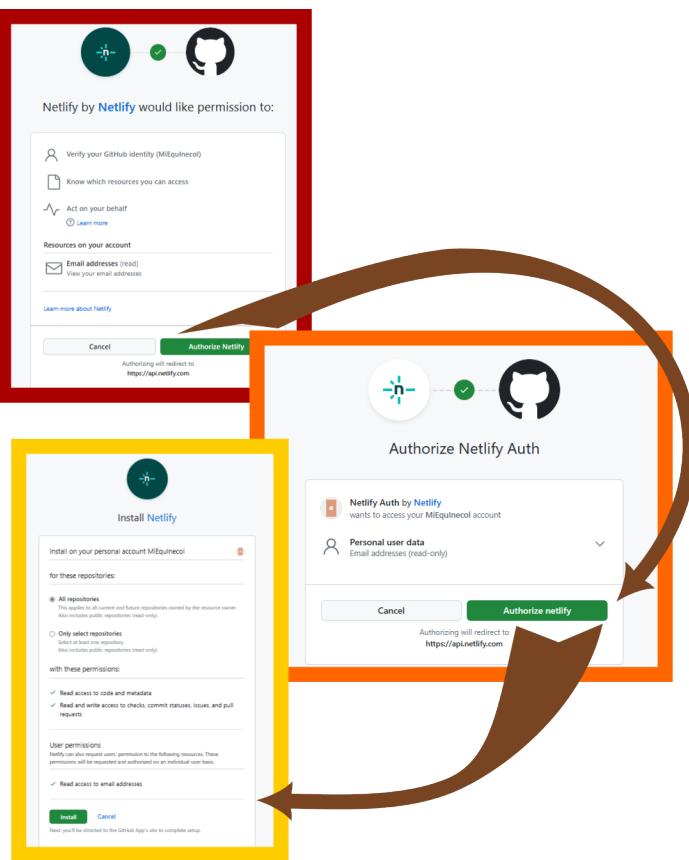
Esta operación inicia en el sitio de *Netlify*, así que por favor dirígete allá e ingresa con tus datos de cuenta si ya los tienes, o registrate para tener acceso al servicio. En un principio no necesitas pagar nada y para la mayoría de los propósitos es muy probable que no lo requieras. En todo caso hay opciones para acceder a especificaciones más *profesionales* que pueden requerir pago, pero siempre podrás elegir lo que más te convenga. En fin, ya debes estar en tu cuenta de *Netlify* ahora.

Ojalá las figuras siguientes te ayuden a seguir los pasos necesarios. La tarea es primero, autorizar a *Netlify* para que interactue con *Github*, así que debes autorizar este enlazamiento de servicios. Una vez que lo hayas hecho, podrás, especificar lo que deberá estar vigilando *Netlify* para que tome *continuamente* lo necesario, lo que hayas cambiado en *Github*. La meta es que construya un sitio Web con tu contenido y lo publique en Internet, y que lo actualice cada vez que des **push** a tu *blog* en *RStudio*. Después de autorizar el vínculo entre los servicios sigue crear un nuevo sitio con tu *blog*, que debe tener en este momento un directorio ****_site**** como resultado de haber hecho *Render* en *RStudio*, si no ves ese directorio ve a la pestaña **Build** en *RStudio* y pídele que construya el sitio de tu *blog* completo.

Los pasos que hay que seguir para esta primera interacción son los siguiente. Iniciar la vinculación con *Github* seleccionando la opción que ofrece importar los documentos desde un *repo* Git. Esto te llevará a una pantalla en donde podrás elegir la opción de utilizar *Github* como origen de datos, entre otras posibilidades.



El Siguiente paso es autorizar a Netlify a acceder a Github a través de tu cuenta, así como los específicos del repositorio que te interesa vincular. *Esto sólo necesitarás hacerlo la primera vez en que habrás de enlazar tus cuenta*. Esto también implica que se instalará una **aplicación** de vínculo de *Netlify* dentro de tu cuenta de *Github*.

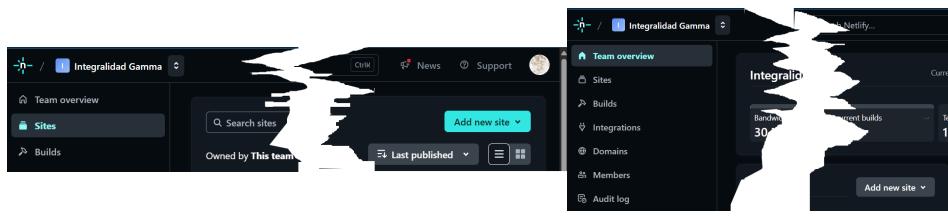


Si todo ocurrió sin problemas, tendrás ahora en *Github*, en el menú de aplicaciones (Avatar→ Settings→ Applications), un botón que te permitirá configurar el vínculo con *Netlify* según tus requerimientos. También podrás ver los *repos* que hayas autorizado desde *Netlify*. Si le dijiste a *Netlify* que autorice todos los repositorios, no tendrás que visitar esta opción nunca.

Creación de un nuevo sitio a publicar

1. En Netlify:

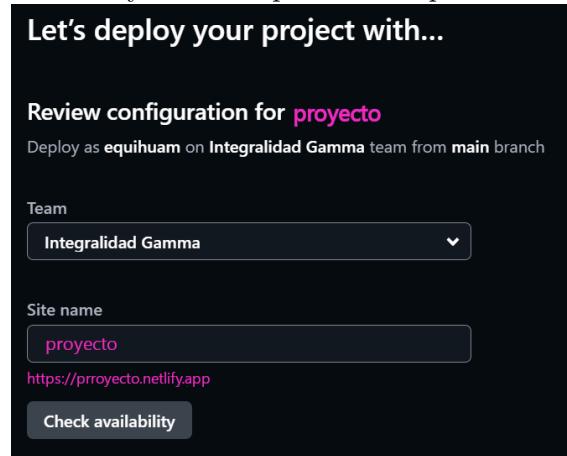
- Desde la opción *team* o *site* puedes generar un nuevo sitio.



Cuando aprietas el botón de *añadir sitio*, aparecerá una nueva pantalla que tiene tres secciones. Se trata de los atributos que tendrás que proporcionar para darle presencia en Internet a tu proyecto y algunos otros atributos que definen como se producirá y actualizará continuamente. Estas operaciones es poco probable que las vuelvas a ver, una vez que tu proyecto esté en producción, aunque desde luego estarán siempre ahí por si deseas hacer ajustes.

¿Qué nombre le das?

Deberás elegir un nombre que se convertirá en una URL para acceder a tu proyecto. Puede ser cualquier cosa que desees, pero debe ser único. En esta sección puedes escribir nombres y verificar que estén disponibles



¿Qué hará Netlify para operar tu sitio?

Es una colección de atributos para indicarle a *Netlify* dónde conseguir los documentos y como manejarlos. En nuestro caso, muy simple, básicamente hay que decirle en donde están los documentos que *Quarto*, con ayuda de *pandoc*, ha *renderizado*. Si no has cambiado nada en `_quarto.yml` la rama que estamos usando aquí para que *Git* los registre es **main** y, en ella el directorio de producción se llama **site**. Por favor verifica el contenido de esto para ayudarte a comprender mejor lo que estás haciendo.

Build settings

Specify how Netlify will build your site.
[Learn more in the docs](#)

Branch to deploy
main

Base directory

The directory where Netlify installs dependencies and runs your build command.

Build command

Examples: `jekyll build`, `gulp build`, `make all`

Publish directory
_site

Examples: `_site`, `dist`, `public`

Functions directory
netlify/functions

Example: `my_functions`

¿Todo listo? ¡a producción!

En nuestro caso no hay más que hacer, *Netlify* tiene información suficiente para encargarse de publicar tu proyecto continuamente. Incorporará los cambios que hagas en *RStudio* en la rama principal. Lo hará automáticamente cada vez que envíes tus cambios a *Github*.

Environment variables

Define environment variables for more control and flexibility over your build.

Add environment variables

Deploy proyecto

Si todo salió bien, en este momento ya debe estar tu proyecto publicado y accesible para cualquier lector del mundo que lo localice y se interese en su contenido.

i Finalmente ¿Cómo quedó todo organizado?

1. Tienes un proyecto en tu máquina
2. Está vinculado con tu cuenta en **Github**
3. Están vinculados **Github** y **Netlify**

Ahora, sólo queda crear el contenido del *Blog*. Recuerda usar un directorio para cada nueva contribución dentro de la carpeta *posts*. Te sugiero usar un esquema *fecha-tema* para llamar esos archivos. Evita usar espacios y caracteres latinos en los temas. Para trabajar hay que crear un archivo *index.qmd*. Puedes hacerlo desde el menú: **File Quarto document...**, o simplemente copiando algún *index.qmd* que tengas por ahí y quieras aprovechar como base para iniciar un nuevo documento.

Configura el encabezado de control con algo así como:

```
---
```

```
title: "Descriptivo del contenido"
author: "Tu Nombre y el de los autores involucrados"
lang: es
date: {{la fecha de publicación}}
categories: [colección de frases clave separadas por comas]
image: "archivo de la imagen que deseas incluir como portada"
code-fold: true
code-summary: "muestra el script:"
fig-cap-location: top
---
```

Si es de tu interés, aquí encontrarás [muchos detalles interesantes sobre YAML](#).

! Arruiné mi **Blog** ¡¡¡Ayuda!!!!

Descompuse archivos clave

Como ya nos pasó, puede ser que eches a perder tu archivo *index.qmd* en la raíz de tu proyecto. Ese documento se encarga de construir la estructura general de tu *blog*. Ojalá hayas echo un **commit** inicial justo acabando de crear tu proyecto. Si fuiste así de sabia o sabio, todo lo que tienes que hacer es buscar el archivo en cuestión y recuperarlo. **Github desktop** será tu amigo para eso.

Por si ocurriera un accidente de este tipo y no hubiera nada más que hacer pongo aquí las líneas generales que suele tener este archivo, cópialas en caso de emergencia y substituye todo lo que tengas en el archivo dañado (si pusiste cosas muy importantes que quiere conservar, haz una copia y haz la reparación en una de ellas. ¡Suerte!

```

---
title: "Título de inicio en tu blog"
listing:
  contents: posts
  sort: "date desc"
  type: default
  categories: cloud
  sort-ui: false
  filter-ui: false
page-layout: full
title-block-banner: true
---

```

Recuerda sólo hacer lo esencial en los documentos en la raíz de tu proyecto: el nombre de tu blog, título inicial y cosas así. No hagas más de lo indispensable en los archivos **index.qmd** y **_quarto.yml** que están en esta ubicación. Toda tu creatividad debes expresarla dentro de subcarpetas contenidas en la carpeta **posts**.

!Eché a perder mi blog, pero tengo una versión buena en git!

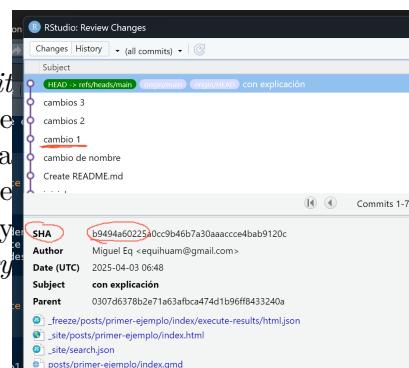
Lo primero es lserenidad y paciencia! Toda la idea de usar *git* es para estos casos. Claro, podrías volverte experta o experto en su uso más profesional mediante comandos, **ojalá te anime**s. Pero por lo pronto hagámoslo con lo que ya tenemos a la mano. Veamos algunos escenarios posibles:

1. Un escenario muy sencillo es cuando estas actualizando un documento que ya pusiste en el registro de *git*. Intentas *renderizarlo* y para tu sorpresa, ¡falla de manera inexplicable! Para colmo, hiciste un montón de cambios que ya ni te acuerdas. Eso sí, estas seguro o segura de que tu última versión, a la que le hiciste un *commit*, sí funcionaba, no a tu gusto completo, pero no se trababa como el que ahora te tiene de malas. En este caso, busca en la pestaña de *git* en *RStudio* el archivo que estás trabajando, imagino un **index.qmd** en un folder dentro de la carpeta **posts**. Seleccionalo y aprieta el botón derecho del mouse. En las ventanita emergente encontrarás la palabra *revert....* ¡Eso es!, dale click y recuperará el archivo de la última versión que resguardaste con un *commit*. ¡Listo!, estás de regreso y ahora puedes volver a empezar, con más cuidado y mejores ideas.
2. Otro caso es cuando has sido muy diligente y has echo todo bien. Haces cambios a tu gusto y cada vez que lo sientes apropiado has echo tus *commits*. Te vas por un tesito, cambias de humor, trabajas en otra parte de tu **blog** y haces commit a tus nuevos cambios. De pronto, un rayo de inspiración, te hace comprender que lo que hiciste al principio del día no era buena ideas, te arrepientes y quieres regresar a una versión que está seis o diez *commits* atrás, sabes a donde te gustaría regresar pues

anotaste un mensaje alusivo que te recuerda claramente el estado del documento que quieras retomar como inicio. ¿Cómo puedes viajar hacia atrás en el tiempo a ese preciso momento? Bueno, hacer esto es relativamente sencillo con el comando `git restore` que puede rescatarte desde la *terminal* de *RStudio*. Las indicaciones `--source` y `main~x` van de cajón (la `x` toma el valor de cuantos *commits* atras hay que ir en la rama `main`), y terminas con la ruta al archivo de tu interés, en este ejemplo `posts/primer-ejemplo/index.qmd`.

```
git restore --source main~5 posts/primer-ejemplo/index.qmd
```

Otra opción para identificar el *commit* preciso de mi interés es poner enseguida de `--source` una parte reconocible en forma única de la firma hash *SHA* de 40 caracteres, que identifica en forma exacta a cada *commit*, y que puedes encontrar con la opción *History* en la pestaña *git*.

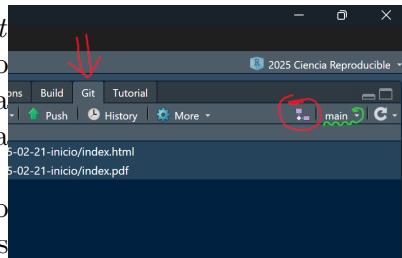


```
git restore --source cc12d23841a posts/primer-ejemplo/index.qmd
```

Como por arte de magia, estás de regreso en el documento solicitado. Quizás *RStudio* te advierta que hubo cambios y te pida le des indicaciones, lee con cuidado lo que te diga y responde adecuadamente. ¡Listo!

¿Cómo Evitar arriesgar mi blog a cada commit?

Una manera de trabajar con más tranquilidad en tu **blog** es otra vez recurriendo a la ayuda que ofrece *git*. Te sugiero evitar trabajar el desarrollo de tus ideas directamente sobre la rama `main` o la que sea tu rama **principal/publicación/producción**. La idea es que cuando estes planeando algo nuevo crees una rama nueva o si ya tienes una y está en la misma cosa, seleccionala como la rama activa.



Asegurate de que la rama del nuevo desarrollo sea la activa. Como recordarás, la publicación de tu **blog** ocurre a través de *Netlify exclusivamente con lo que pongas en la rama main..* Así que todo lo que hagas en la nueva rama no pasará a publicación en tu **blog**, sólo lo podrás ver localmente cuando hagas *render*, será una especie de borrador. Cuando todo esté a tu gusto y ya lo quieras publicar tienes que combinar el contenido de la rama en la que has estado trabajando con la rama *main*. Para hacerlo hay que seguir estos pasos:

1. Cambia el espacio de trabajo en *git* a la rama *main*.
2. En la pestaña de **terminal** de *RStudio*. Indicale a *git* que jale para combinar *la_rama_alterna_con_mi_trabajo*. Escribe para esto los siguientes comandos de *git* en la pestaña de **terminal**:

```
git merge la_rama_alterna_con_mi_trabajo
```

Como podrás imaginar, [el proceso requiere verificar muchas cosas](#), lo cual hace *git* por ti. Normalmente y si te portaste bien, la fusión de las ramas debe ocurrir sin tropiezos. Si hubiera dificultades, *git* te lo hará saber. Basicamente lo que hará es indicarte que hay cambios contradictorios. Es decir, hiciste cambios tú, al mismo archivo en las dos ramas, así que no se puede decidir cuál es el cambio que hay que conservar sin preguntarte a ti que es lo que prefieres. En tal caso, veras los archivos inconsistentes marcados con una etiqueta naranja y tendrás que revisarlos para que tú resuelvas directamente lo conducente. Puede ser que haya archivos que en realidad no cambiaste tú directamente, como serían los que se producen automáticamente en *_site* y que sólo arrastrán las consecuencias de tus cambios. De esos no te preocupes, se renovarán cuando hayas resuelto los importantes, que seguramente son los que tienes en **posts**, o algún detallito que arreglaste en los archivos general en la raíz de tu proyecto.