

Ciencia reproducible para la política pública

Miguel Equihua Zamora Octavio Pérez-Maqueo
Elio Lagunes Díaz

2025-05-07

Tabla de contenidos

Prefacio	3
Herramientas que usaremos	3
Ciencia reproducible y abierta	3
1 Introducción	5
1.1 RStudio	5
1.2 Flujo de trabajo	6
1.2.1 Archivo de configuración <code>_quarto.yml</code>	7
1.2.2 Previsualizar el libro	8
1.3 Markdown	8
2 Git: lo básico	10
2.1 Huellas de tu trabajo: control de versiones	10
2.1.1 ¿Qué es Git?	11
3 Resumen	16
Referencias	17

Prefacio

Herramientas que usaremos

En este libro desarrollamos el contenido propuesto para un *curso/taller* de ciencia reproducible, basado en el uso de [Quarto](#) y el *lenguaje R*.

La intención es desarrollar una experiencia inmersiva sobre la incorporación de datos y el apoyo de evidencia en la construcción de instrumentos relevantes para la política pública en materia ambiental. Buscamos ofrecer al lector ejercicios con los que poner a prueba el uso de herramientas que propician el desarrollo de un enfoque científico reproducible.

Los ejercicios prácticos permitirán al lector hacer un recorrido a lo largo de la siguiente ruta:

1. Uso de *Quarto*. Lo proponemos como base de encuentro de la producción de contenido en texto, análisis computacional y gráficos de datos, en un sólo espacio.
2. Uso de *Git*, [Github](#) y [Netlify](#). Se trata de recursos convenientes para el control del proceso de desarrollo de contenidos y para la publicación en línea.
3. Introducción al uso del *lenguaje R* como base de construcción de algoritmos computacionales para análisis de datos.
4. Uso de *R* para el procesamiento de datos geoespaciales, análisis de cartografía y producción de mapas.

Con base en estos elementos desarrollaremos, paso a paso, los componentes necesarios para integrar, a manera de ejemplo, un documento tipo *Manifiesto de Impacto Ambiental (MIA)*. Lo haremos con un apego genérico a lo que se suele hacer en [México en el orden federal](#), pero la experiencia es fácilmente extrapolable a la práctica de cualquier otro nivel administrativo e incluso cualquier país del mundo.

Ciencia reproducible y abierta

En el movimiento social que nos invita a reflexionar sobre la *ciencia abierta*, hay quienes sostenemos 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. La ciencia abierta también nos plantea la necesidad de cuidar la confiabilidad de los datos, pues una *buen ciencia exige buenos datos*. Otro aspecto

importante, además de los datos, es la forma como los analizamos. Lo usual es confiar en sus propiedades estadísticas para llegar a conclusiones, así que los algoritmos y procedimientos estadísticos empleados son otro componente que debe ser transparente.

La meticulosidad y apertura con la que abordemos estos asuntos tiene un impacto definitivo sobre la posibilidad de que otras personas puedan llegar a similares conclusiones que las nuestras, lo que abona a la noción de *objetividad científica*, y es una de las razones que nos lleva a confiar en la ciencia al tomar decisiones.

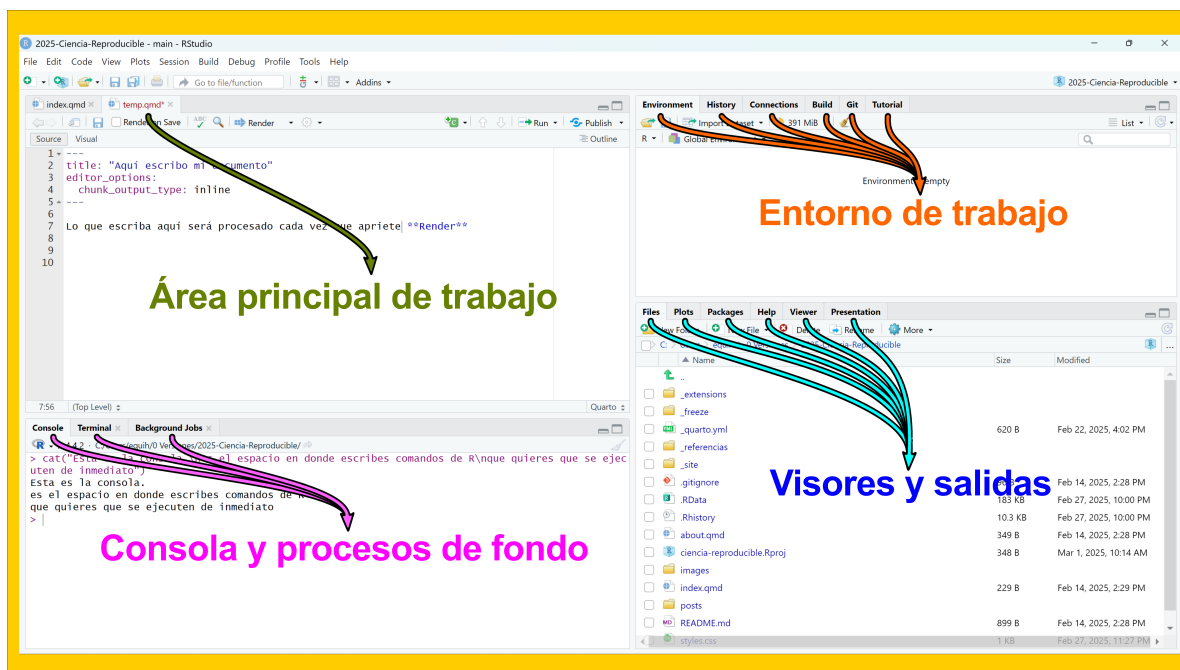
1 Introducción

En términos generales el contenido de una MIA de orden federal debe apegarse a lo especificado en el Reglamento de la LEGEPA en materia de Evaluación del Impacto Ambiental (SEMARNAT, 2014). Ahí se especifica que estos instrumentos pueden ser sólo de dos tipos: regionales o particulares. La que haremos como ensayo seguirá en términos generales el marco de un estudio *MIA* particular. Si se requiriera un poco más de especificidad recurriremos a la guía de una MIA Particular - Sector Turístico (SEMARNAT, 2022). Con base en este marco normativo los capítulos que habrá que desarrollar a lo largo de los ejercicios propuestos son los siguientes.

1. Datos generales del proyecto, del promovente y del responsable del estudio de impacto ambiental.
2. Descripción del proyecto;.
3. Vinculación con los ordenamientos jurídicos aplicables en materia ambiental y, en su caso, con la regulación sobre uso del suelo.
4. Descripción del sistema ambiental y señalamiento de la problemática ambiental detectada en el área de influencia del proyecto.
5. Identificación, descripción y evaluación de los impactos ambientales.
6. Medidas preventivas y de mitigación de los impactos ambientales.
7. Pronósticos ambientales y, en su caso, evaluación de alternativas.
8. Identificación de los instrumentos metodológicos y elementos técnicos que sustentan la información señalada en las fracciones anteriores.

1.1 RStudio

Es un *entorno integrado de desarrollo (IDE)*. El proceso de trabajo que utilizaremos se basa en el programa *RStudio*. Se trata de un *Entorno de Desarrollo Integrado (IDE)* que permite la interacción entre el *lenguaje R*, *pandoc*, *git* y *Github* de una manera sencilla y eficiente. Además, 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 también un problema si necesitas buscar ayuda en Internet, que las más de las veces está en inglés. Entonces, si quieres regresar al mundo anglosajón 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")
```

1.2 Flujo de trabajo

El proceso de trabajo que seguiremos en este libro inicia con la creación de un proyecto *Quarto book* en *Rstudio*, lo que genera un conjunto de archivos y carpetas dentro de la carpeta raíz

del proyecto. Los principales componentes de esta etapa inicial son el archivo `_quarto.yml`, y la carpeta `_book`.

El primero es un *documento de control y metadatos*. Define los atributos generales del libro y datos como autores, fechas, idioma, etc. La carpeta contendrá el libro producido en los formato de publicación que especifiques. Los nombres de estos dos elementos no los debes modificar nunca. *RStudio* crea además algunos archivos de ejemplo para ilustrar lo que puede ser el contenido del libro: `intro.qmd`, `summary.qmd` y una colección de datos de referencias documentales: `references.bib`, los que también se anotan en el archivo de control. Los nombres de estos archivos pueden ser los que tu prefieras, pero te sugiero mantenerlos cortos y no incluir caracteres latinos. Piensa que deben ser fáciles de procesar por una computadora, no tanto inteligibles por seres humanos.

1.2.1 Archivo de configuración `_quarto.yml` .

Este archivo contiene la configuración inicial de tu libro. En un principio puede tener un contenido como el siguiente:

```
project:
  type: book

book:
  title: "mybook"
  author: "Juanita Staku"
  date: "8/18/2021"
  chapters:
    - index.qmd
    - intro.qmd
    - summary.qmd
    - references.qmd

bibliography: references.bib

format: html:
  tema: cosmo

pdf:
  documentclass: scrreprt

epub:
  cover-image: cover.png
```

1.2.2 Previsualizar el libro

En cualquier momento puedes producir una *vista previa* de tu documento. Lo único que hay que hacer es activar el **botón render** en *RStudio*. Como la ista previa la realiza *quarto* con ayuda de *pandoc*, podemos también hacerlo directamente desde una ventana de terminal (la integrada en *RStudio* o la que puedas abrir por separado en tu equipo). Ahí sólo tendrás que asegurarte de estar ubicada en la carpeta de tu proyecto y proporcionar el siguiente comando: `quarto preview`. ¡Eso es todo!, por lo pronto.

1.3 Markdown

Markdown fue desarrollado en 2004 por **John Gruber** (Fig. 1.1). Ideo 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 1.1: John Gruber

He aquí uno de los grande 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**

2 Git: lo básico

2.1 Huellas de tu trabajo: control de versiones

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.

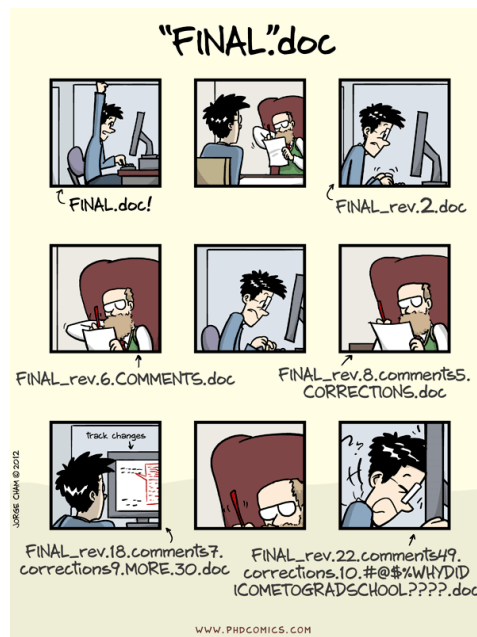


Figura 2.1: Fuente: *Final.doc* en **Piled Higher and Deeper** por Jorge Cham, <http://www.phdcomics.com>

Es una aplicación diseñada por el iniciador del desarrollo de Linux, **Linus Torvalds** (Fig. 2.2). **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 2.2: Linus Torvalds

2.1.1 ¿Qué es Git?

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 desees 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 *prellenado* 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 quieres 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 tarea 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 `usethis` en la pestaña de consola.

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

Eso es todo.

3 Resumen

Por lo pronto no tenemos nada que decir

Referencias

- SEMARNAT. (2014). *Reglamento de la Ley General del Equilibrio Ecológico y la Protección al Ambiente en materia de Evaluación del Impacto Ambiental*. https://www.diputados.gob.mx/LeyesBiblio/regley/Reg_LGEEPA_MEIA_311014.pdf
- SEMARNAT. (2022). *Guía MIA Particular - Sector Turístico*. https://www.gob.mx/cms/uploads/attachment/file/121010/Guia_MIA-Particular_Turistico.pdf