Guía completa de entornos virtuales en R con RStudio (renv)

Edison Achalma

Escuela Profesional de Economía, Universidad Nacional de San Cristóbal de Huamanga

Nota del Autor

Edison Achalma https://orcid.org/0000-0001-6996-3364

El autor no tiene conflictos de interés que revelar. Los roles de autor se clasificaron utilizando la taxonomía de roles de colaborador (CRediT; https://credit.niso.org/) de la siguiente manera: Edison Achalma: conceptualización, metodología, análisis formal, investigación, recursos, curación de datos, redacción, visualización, supervisión, administración del proyecto

La correspondencia relativa a este artículo debe dirigirse a Edison Achalma, Escuela Profesional de Economía, Universidad Nacional de San Cristóbal de Huamanga, Portal Independencia N° 57, Ayacucho, AYA 5001, Perú, Email: elmer.achalma.09@unsch.edu.pe

Resumen

Este abstract será actualizado una vez que se complete el contenido final del artículo.

Palabras Claves: keyword1, keyword2

Guía completa de entornos virtuales en R con RStudio (renv)

Tabla de contenidos

Guía completa de entornos virtuales en R con RStudio (renv)

1 Guía completa de entornos virtuales en R con RStudio

Referencia técnica independiente sobre **entornos virtuales en R**, centrada en `renv`, el paquete oficial recomendado por Posit/RStudio para crear bibliotecas de paquetes aisladas por proyecto. Basada en la documentación oficial: rstudio.github.io/renv, el repositorio GitHub rstudio/renv, y la documentación de RStudio IDE sobre Projects y entornos.

1.1 Tabla de contenidos

1.2 1. ¿Qué es un entorno virtual en R?

En R, todos los paquetes se instalan por defecto en una **biblioteca de sistema** (system library) compartida por todas las sesiones y proyectos. Esto significa que si un proyecto necesita `dplyr` versión 1.0 y otro necesita la versión 1.1, ambos se pisan entre sí: actualizar un paquete para un proyecto puede romper silenciosamente otro proyecto que dependía de una versión anterior.

Un **entorno virtual** en R resuelve esto dando a cada proyecto su propia biblioteca de paquetes, completamente independiente de la biblioteca de sistema y de la de cualquier otro proyecto. La idea es análoga a `venv` en Python o a un `node_modules` por proyecto en Node.js, pero adaptada a las particularidades de R.

El paquete oficial de Posit/RStudio para esto es **`renv`** (abreviatura de *reproducible environment*), sucesor de un paquete anterior llamado `packrat`. `renv` es, en la práctica, **el estándar actual** para este propósito en el ecosistema R.

1.3 2. renv: el estándar oficial

Según la documentación oficial, `renv` ayuda a que los proyectos de R sean:

- **Aislados (isolated)**: instalar o actualizar un paquete en un proyecto no afecta a otros proyectos, porque cada uno tiene su propia biblioteca privada.
- **Portables (portable)**: es fácil transportar un proyecto de una máquina a otra, incluso entre distintas plataformas (Linux, Windows, etc.).
- **Reproducibles (reproducible)**: `renv` registra las versiones exactas de los paquetes usados y garantiza que esas mismas versiones se instalen en cualquier lugar donde se restaure el proyecto.

Estos tres pilares —aislamiento, portabilidad, reproducibilidad— son el motivo por el cual `renv` es relevante en prácticamente cualquier flujo de trabajo serio en R: desde un script personal hasta un pipeline de producción.

1.4 3. Conceptos clave: biblioteca, repositorio, caché, lockfile

Antes de usar `renv` conviene tener claros cuatro términos que la documentación oficial distingue cuidadosamente:

*1.4.1 3.1. Biblioteca (*library*)*

Una **biblioteca** es un directorio que contiene paquetes instalados. Es una fuente común de confusión, porque al escribir `library(dplyr)` parece que se está "cargando la biblioteca dplyr", cuando en realidad se está cargando el *paquete* dplyr desde una biblioteca (carpeta). Por defecto, todos los paquetes van a una única biblioteca de sistema compartida; con `renv`, cada proyecto obtiene su propia **biblioteca de proyecto** (`renv/library`).

Puedes ver tus bibliotecas activas con:

```
.libPaths()
```

*1.4.2 3.2. Repositorio (*repository*)*

Un **repositorio** es una fuente de paquetes (por ejemplo CRAN, Bioconductor, GitHub, o el Posit Public Package Manager). `install.packages()` descarga un paquete desde un repositorio y lo coloca en una biblioteca.

1.4.3 3.3. Caché global de renv

Para no tener que descargar o compilar el mismo paquete una y otra vez para cada proyecto, `renv` mantiene una **caché global** compartida en el sistema. Cuando instalas un paquete por primera vez, se descarga y compila una sola vez; para cada proyecto adicional que lo necesite, `renv` simplemente crea un enlace desde la biblioteca del proyecto hacia esa copia en caché, ahorrando tiempo y espacio en disco.

1.4.4 3.4. Lockfile (`renv.lock`)

El **lockfile** es un archivo JSON, siempre llamado `renv.lock`, que registra toda la información necesaria para recrear el proyecto en el futuro: la versión de R usada, los repositorios de origen, y un registro por cada paquete con su versión exacta, fuente (CRAN,

GitHub, etc.) y hash de verificación. Es el "contrato" que permite reproducir el entorno en otra máquina.

1.5 4. Instalación de renv

`renv` se instala como cualquier paquete de R, desde la consola (en RStudio o en R puro, sobre cualquier sistema operativo donde ya tengas R y RStudio instalados):

```
install.packages("renv")
```

Verifica la instalación:

```
library(renv)
packageVersion("renv")
```

No requiere ninguna instalación adicional a nivel de sistema operativo: a diferencia de `venv` en Python, `renv` es un paquete de R puro y funciona igual en Kubuntu, Arch Linux o Windows, siempre que R y (opcionalmente) RStudio ya estén instalados.

1.6 5. Flujo de trabajo básico

Este es el ciclo de trabajo central documentado oficialmente, y se repite (con variaciones) en todos los casos de uso de las secciones siguientes.

1.6.1 Paso 1 — Inicializar renv en el proyecto

Desde la consola de R, **con el directorio de trabajo ubicado en la carpeta del proyecto**:

```
renv::init()
```

Esto hace lo siguiente automáticamente:

1. Crea la biblioteca de proyecto en `renv/library`.

2. Detecta los paquetes que ya estás usando en el proyecto (rastreando los archivos `.R`/`.Rmd`/`.qmd` en busca de llamadas a `library()`, `require()`, etc., mediante `renv::dependencies()`).
3. Instala esos paquetes detectados en la biblioteca del proyecto (mediante `renv::hydrate()`, que copia desde tu biblioteca de usuario existente cuando es posible, en vez de reinstalar todo desde cero).
4. Crea el lockfile `renv.lock` con el estado inicial.
5. Crea (o modifica) un `.Rprofile` local al proyecto, que se ejecuta automáticamente cada vez que abres R en esa carpeta, activando la biblioteca del proyecto sin que tengas que hacer nada manualmente.
6. Reinicia la sesión de R (si estás dentro de RStudio).

1.6.2 Paso 2 — Trabajar con normalidad

Sigues usando `install.packages()`, `update.packages()`, etc. como siempre. También puedes usar las variantes propias de renv:

```
renv::install("dplyr")          # instalar un paquete
renv::install("usuario/repo")   # instalar desde GitHub
```

1.6.3 Paso 3 — Tomar una "fotografía" del entorno (snapshot)

Cuando confirmes que tu código funciona con el conjunto de paquetes actual:

```
renv::snapshot()
```

Esto actualiza `renv.lock` con las versiones exactas de los paquetes actualmente instalados en la biblioteca del proyecto.

1.6.4 Paso 4 — Restaurar el entorno en otra máquina (o más adelante)

Si compartes el proyecto, o lo abres en otra computadora, o simplemente quieres volver al estado exacto registrado en el lockfile:

```
renv::restore()
```

Esto reinstala exactamente las versiones de paquetes anotadas en `renv.lock`, sin importar qué versiones tengas instaladas en ese momento en tu biblioteca de usuario.

1.6.5 Resumen visual del ciclo

```
renv::init()        →  crea infraestructura del proyecto
      ↓
trabajas, instalas/actualizas paquetes
      ↓
renv::snapshot()    →  registra el estado actual en renv.lock
      ↓
(compartes el proyecto / cambias de máquina)
      ↓
renv::restore()     →  reinstala exactamente lo que dice renv.lock
```

1.7 6. Integración con RStudio Projects

RStudio (el IDE) tiene soporte nativo para `renv` integrado en el flujo de **RStudio Projects**.

Al crear un nuevo proyecto en RStudio (`File` → `New Project...`), el asistente incluye una casilla **"Use renv with this project"**. Si la marcas, RStudio crea automáticamente toda la infraestructura de `renv` (equivalente a haber corrido `renv::init()` manualmente) en el momento mismo de crear el proyecto, antes incluso de escribir una sola línea de código.

Esto es la forma más cómoda de empezar un proyecto nuevo con entorno virtual desde cero: no hace falta recordar ejecutar ningún comando, el propio flujo de creación de proyecto en RStudio ya lo deja activado.

Para proyectos ya existentes (creados sin esa casilla marcada), simplemente ejecuta `renv::init()` manualmente una vez, como se describió en la sección 5.

1.8 7. Caso de uso 1 — Tesis o trabajo de investigación individual

Escenario: estás escribiendo tu tesis o un artículo y usas R/RStudio para el análisis estadístico, con un conjunto de paquetes (por ejemplo `tidyverse`, `lme4`, `apaTables`) que pueden actualizarse con el tiempo mientras dura el trabajo (meses o años).

Por qué importa: si una actualización de un paquete cambia el comportamiento de una función estadística a mitad de tu tesis, tus resultados podrían dejar de ser reproducibles exactamente, incluso en tu propia máquina, varios meses después.

Flujo recomendado:

```
# Al iniciar el proyecto de tesis
renv::init()

# Trabajas normalmente, instalando lo que necesites
install.packages(c("tidyverse", "lme4", "apaTables"))

# Cada vez que termines una etapa importante (p. ej. antes de
# enviar un capítulo a tu asesor, o antes de la sustentación)
renv::snapshot()
```

Beneficio concreto: el día de la sustentación o defensa, si un jurado o evaluador quiere reproducir exactamente tus resultados en otra computadora, basta con que reciba la carpeta del proyecto (incluyendo `renv.lock`) y ejecute `renv::restore()`. Obtendrá las mismas versiones de paquete que usaste tú, eliminando una fuente común de "en mi máquina sí funciona".

1.9 8. Caso de uso 2 — Curso o asignatura con múltiples prácticas

Escenario: dictas o llevas un curso de econometría, estadística o programación en R con varias prácticas o controles a lo largo del semestre, cada una con requisitos de paquetes ligeramente distintos (una práctica usa `forecast`, otra usa `plm`, otra usa `caret`).

Problema sin entornos virtuales: instalar `caret` para la práctica 5 puede arrastrar actualizaciones de dependencias compartidas que rompan silenciosamente el código de la práctica 2, que ya habías entregado y calificado.

Flujo recomendado: crear **un proyecto de RStudio por práctica o por unidad**, cada uno con su propio `renv`:

```
curso-econometria/
   practica-01-regresion-lineal/   (renv propio)
   practica-02-series-tiempo/      (renv propio)
   practica-03-panel-data/         (renv propio)
   practica-04-machine-learning/   (renv propio)
```

En cada subcarpeta, al crearla como proyecto de RStudio, marcas "Use renv with this project", o ejecutas `renv::init()` manualmente. Cada práctica queda completamente aislada: actualizar `caret` en la práctica 4 no afecta en absoluto a la práctica 1.

Beneficio adicional para docentes: puedes distribuir cada práctica a tus estudiantes incluyendo el `renv.lock` correspondiente, garantizando que todos en el curso usen exactamente las mismas versiones de paquetes que tú usaste para diseñar y probar los ejercicios, evitando errores de "a mí me sale un resultado distinto" por diferencias de versión.

1.10 9. Caso de uso 3 — Colaboración en equipo / proyecto compartido en Git

Escenario: trabajas con coautores o colegas en un repositorio Git compartido (por ejemplo en GitHub, bajo tu cuenta `@achalmed`), con varios contribuyentes ejecutando el mismo código de análisis en distintas máquinas.

Flujo recomendado:

1. Quien crea el proyecto ejecuta `renv::init()` y hace el primer `renv::snapshot()`.
2. Se hace commit a Git de los archivos generados por renv:

```
git add renv.lock .Rprofile renv/settings.json renv/activate.R
git commit -m "Configurar entorno renv del proyecto"
git push
```

`renv` genera automáticamente un `.gitignore` apropiado dentro de la carpeta `renv/`, de modo que la biblioteca de paquetes en sí (`renv/library`, que puede pesar cientos de MB) **no** se sube al repositorio; solo se sube el lockfile y la infraestructura ligera necesaria para reconstruirla.

3. Cuando un colaborador clona el repositorio y abre el proyecto en RStudio, `renv` se autoarranca (*bootstrap*): detecta que el proyecto usa renv, descarga e instala automáticamente la versión apropiada del propio paquete `renv`, y le pregunta al colaborador si desea instalar todos los paquetes necesarios ejecutando `renv::restore()`.

4. Cuando alguien instala o actualiza un paquete nuevo para el proyecto, debe correr `renv::snapshot()` y avisar al resto del equipo (vía commit/push) que el lockfile cambió, para que los demás corran `renv::restore()` y se sincronicen.

Regla práctica para el equipo: "si tocas paquetes, haces snapshot y avisas; si te avisan que el lockfile cambió, haces restore antes de seguir trabajando".

1.11 10. Caso de uso 4 — Múltiples proyectos en la misma máquina (aislamiento)

Escenario: en tu propia computadora mantienes en paralelo varios proyectos activos: tus sitios Quarto académicos, scripts de tutoría en economía, un proyecto de análisis de datos para una consultoría, y experimentos personales con paquetes nuevos de ciencia de datos.

Problema sin entornos virtuales: probar la última versión de un paquete experimental para un proyecto personal puede romper, sin que te des cuenta, un script de tutoría que dependía de un comportamiento anterior de ese mismo paquete.

Flujo recomendado: activar `renv` en **cada proyecto independiente** que mantengas activo simultáneamente. Gracias a la caché global de renv (sección 3.3), esto no implica

descargar ni compilar cada paquete una vez por proyecto: si `ggplot2` ya está en caché por haberlo instalado en un proyecto, los demás proyectos simplemente enlazan a esa misma copia, sin gasto adicional relevante de espacio en disco ni tiempo de instalación, mientras cada proyecto conserva su propio registro independiente de qué versión está usando.

Resultado: puedes actualizar libremente los paquetes de tu proyecto experimental sin ningún riesgo de afectar tus otros proyectos en producción o entrega.

1.12 11. Caso de uso 5 — Reportes reproducibles con Quarto / R Markdown

Escenario: generas documentos con Quarto o R Markdown (`.qmd` / `.Rmd`) que combinan texto y código R, por ejemplo reportes económicos, material didáctico o capítulos de tesis con `apaquarto`.

Por qué aplica renv aquí: `renv::dependencies()` rastrea automáticamente los bloques de código dentro de archivos `.qmd` y `.Rmd`, igual que lo hace con scripts `.R` normales, por lo que `renv::init()` y `renv::hydrate()` detectan correctamente los paquetes usados dentro de tus documentos Quarto/RMarkdown sin configuración adicional.

Flujo recomendado:

```
renv::init()      # detecta paquetes usados en tus .qmd / .Rmd
# ... renderizas el documento, revisas que todo funcione ...
renv::snapshot()  # fija las versiones que produjeron ese render
```

Advertencia importante (limitación documentada oficialmente): `renv` solo gestiona **paquetes de R**. El motor de renderizado **Pandoc**, del cual depende fuertemente `rmarkdown` para convertir el documento a su formato final (PDF, HTML, Word), **no** está empaquetado dentro del paquete `rmarkdown` ni es gestionado por `renv`. Esto significa que restaurar el lockfile garantiza las mismas versiones de paquetes de R, pero **no** garantiza, por sí solo, que obtengas exactamente el mismo render visual si la versión de Pandoc instalada en la máquina de destino es distinta. Si esto te genera problemas de reproducibilidad visual, puedes usar el paquete `pandoc` (gestor de versiones de Pandoc) como complemento.

1.13 12. Caso de uso 6 — Migrar un proyecto antiguo (sin renv) a un entorno reproducible

Escenario: tienes un proyecto de R ya avanzado (por ejemplo uno de tus repositorios `website-achalma` con scripts de análisis, o un script de tutoría antiguo) que nunca usó `renv`, y quieres empezar a protegerlo hacia adelante sin reescribir nada.

Flujo recomendado:

```
# Ubícate (setwd o abre el proyecto) en la carpeta del proyecto existente
renv::init()
```

Como se describió en la sección 5, `init()` detecta automáticamente qué paquetes ya está usando el código existente, los copia (cuando es posible) desde tu biblioteca de usuario actual hacia la nueva biblioteca de proyecto (evitando reinstalar todo desde cero), y genera el lockfile con el estado funcional actual del proyecto.

Resultado: a partir de ese momento, ese proyecto queda "congelado" en un estado conocido y reproducible, sin haber tenido que identificar manualmente la lista de dependencias ni sus versiones.

Variante — inicialización en blanco: si prefieres no instalar nada automáticamente y declarar tú mismo los paquetes desde cero, puedes inicializar con biblioteca vacía:

```
renv::init(bare = TRUE)
```

1.14 13. Caso de uso 7 — Perfiles múltiples dentro de un mismo proyecto

Escenario: un mismo proyecto necesita comportarse de forma distinta según el contexto. Ejemplos documentados oficialmente:

- Un perfil de **"desarrollo"** (`dev`) mientras pruebas y depuras código.
- Un perfil de **"producción"** (`prod`) para despliegues finales, con un conjunto más controlado de paquetes.

- Un perfil de **"shiny"** específico si el proyecto también contiene una aplicación Shiny con dependencias propias (por ejemplo `shiny`, `tidyverse`) que no necesitas cargar en el resto del proyecto.

Por qué importa: en vez de mantener varias copias del proyecto, `renv` permite que un mismo proyecto tenga **múltiples bibliotecas y lockfiles alternativos**, uno por perfil, todos viviendo dentro del mismo árbol de carpetas.

Flujo recomendado:

```
# Crear y activar el perfil "dev" como perfil por defecto del proyecto
renv::activate(profile = "dev")
```

Con esto, las rutas de biblioteca y lockfile del proyecto pasan a resolverse dentro de `renv/profiles/dev/`, en vez de las rutas por defecto.

Para activar un perfil solo durante una sesión puntual, sin cambiar el valor por defecto del proyecto:

```
Sys.setenv(RENV_PROFILE = "dev")
```

o, desde la línea de comandos (por ejemplo en tu `zsh`), antes de lanzar R:

```
export RENV_PROFILE=dev
```

Para volver al perfil estándar:

```
renv::activate(profile = "default")
```

Declarar dependencias específicas de un perfil: se hace en el archivo `DESCRIPTION` del proyecto, con un campo dedicado por perfil, por ejemplo:

```
Config/renv/profiles/shiny/dependencies: shiny, tidyverse
```

1.15 14. Caso de uso 8 — Reproducibilidad a largo plazo (archivar un proyecto terminado)

Escenario: terminaste una investigación, tesis o reporte y quieres archivarlo de forma que, dentro de dos o tres años, alguien (incluido tú mismo) pueda volver a ejecutarlo exactamente igual, aunque CRAN ya haya avanzado varias versiones de los paquetes involucrados.

Flujo recomendado al cerrar el proyecto:

```
renv::snapshot()   # última fotografía del entorno funcional final
```

Y conservar junto al código fuente:

- El archivo `renv.lock`.
- El `.Rprofile` del proyecto.
- La carpeta `renv/` (sin la subcarpeta `library`, que es regenerable).

Al reabrir el proyecto archivado, meses o años después:

```
renv::restore()
```

`renv` reinstalará las versiones exactas registradas, incluso si en CRAN ya existen versiones más nuevas de esos mismos paquetes, siempre que los binarios/fuentes originales sigan disponibles en algún repositorio accesible.

Limitación honesta que reconoce la propia documentación: si un paquete fue instalado originalmente desde un binario que posteriormente deja de estar disponible (por ejemplo, retirado de un repositorio), `renv` intentará compilarlo desde el código fuente como alternativa, pero esto puede fallar si faltan las dependencias de sistema necesarias para compilar (ver la sección de dependencias de sistema en la guía de instalación de R/RStudio). Por esta razón, **renv no es una garantía absoluta de reproducibilidad eterna**, pero sí resuelve la parte más importante y más frecuente del problema: las versiones de paquetes de R.

1.16 15. Caso de uso 9 — Despliegue en servidor / Docker / Posit Connect

Escenario: necesitas que tu análisis o aplicación (por ejemplo una app Shiny, o un pipeline de reportes) se ejecute de forma idéntica en un servidor, contenedor Docker, integración continua (CI), o en una plataforma de publicación como Posit Connect.

Por qué aplica renv aquí: el mismo lockfile que usas en tu laptop es el artefacto que le indica al entorno remoto exactamente qué instalar. La documentación oficial cubre tres escenarios de despliegue específicos:

- **Integración continua (CI)**: en pipelines de CI (por ejemplo GitHub Actions), se puede usar `renv::restore()` como paso de configuración del entorno antes de correr pruebas, garantizando que el CI use las mismas versiones que tu máquina de desarrollo.
- **Docker**: se recomienda copiar el `renv.lock` (y la infraestructura de renv) dentro de la imagen y ejecutar `renv::restore()` como parte del `Dockerfile`, de modo que la imagen final tenga exactamente el entorno de paquetes esperado. La caché de renv puede aprovecharse entre builds de imágenes para acelerar reconstrucciones.
- **Posit Connect**: al publicar contenido (apps Shiny, reportes Quarto/RMarkdown) hacia Posit Connect, el servidor usa el `renv.lock` del proyecto para reconstruir el entorno de paquetes necesario en el servidor remoto.

Flujo conceptual mínimo para cualquiera de estos casos:

```
# En tu máquina de desarrollo, antes de desplegar
renv::snapshot()
```

```
# En el entorno remoto / imagen / pipeline
# (usualmente en un script de arranque o en el Dockerfile)
```

```
renv::restore()
```

Para los detalles específicos de sintaxis de cada plataforma (ejemplo de `Dockerfile`, configuración de un workflow de GitHub Actions, opciones de Posit

Connect), consulta los artículos oficiales dedicados: *Using renv with continuous integration*, *Using renv with Docker* y *Using renv with Posit Connect*, enlazados en la sección de referencias.

1.17 16. Caso de uso 10 — Desarrollo de un paquete de R propio

Escenario: estás desarrollando tu propio paquete de R (por ejemplo, una herramienta interna para tus prompts/scripts educativos, o un paquete que planeas publicar en CRAN).

Particularidad de este caso: cuando `renv` se usa dentro de un proyecto de **desarrollo de paquete** (es decir, un proyecto que contiene su propio archivo `DESCRIPTION` de paquete, no solo de análisis), el comportamiento es ligeramente distinto al de un proyecto de análisis normal: en vez de usar siempre la biblioteca de proyecto, `renv` puede usar una biblioteca externa al directorio del proyecto, para evitar mezclar la infraestructura del propio paquete en desarrollo con su entorno de pruebas.

Casos típicos dentro de este escenario:

- **Probar contra la última versión disponible en CRAN** de tus dependencias (recomendado si planeas publicar el paquete en CRAN, ya que CRAN exige compatibilidad con el ecosistema actual):

```
renv::restore()
```

- **Probar contra versiones de desarrollo** de otras dependencias (por ejemplo si dependes de una rama no publicada aún de otro paquete tuyo o de un colaborador), declarando esas dependencias de desarrollo en el campo `Remotes` del `DESCRIPTION` del paquete.
- **Integración continua del propio paquete**: es habitual usar `renv` dentro del flujo de CI del paquete (por ejemplo en GitHub Actions) para acelerar la instalación de dependencias de prueba aprovechando la caché global, en vez de reinstalar todo desde cero en cada ejecución del pipeline.

Importante: si vas a enviar tu paquete a CRAN, **no debe incluirse la infraestructura de renv** dentro del tarball fuente que se sube a CRAN; `renv` es una herramienta para tu flujo de desarrollo, no parte del paquete distribuido.

1.18 17. Referencia rápida de funciones

Función	Propósito
`renv::init()`	Inicializa renv en el proyecto actual (crea biblioteca, lockfile y `.Rprofile`)
`renv::init(bare = TRUE)`	Inicializa con biblioteca vacía, sin instalar nada automáticamente
`renv::snapshot()`	Registra en `renv.lock` el estado actual de la biblioteca del proyecto
`renv::restore()`	Reinstala en la biblioteca del proyecto las versiones exactas del `renv.lock`
`renv::install("pkg")`	Instala un paquete (alternativa a `install.packages()`, soporta GitHub, Bioconductor, etc.)
`renv::update()`	Actualiza todas las dependencias del proyecto a sus últimas versiones disponibles
`renv::upgrade()`	Actualiza únicamente el propio paquete `renv`
`renv::dependencies()`	Lista los paquetes que el código del proyecto está usando actualmente
`renv::hydrate()`	Instala en la biblioteca de proyecto los paquetes detectados por `dependencies()`
`renv::status()`	Compara el estado actual de la biblioteca contra lo registrado en el lockfile

Función	Propósito
`renv::history()`	Muestra el historial de versiones anteriores del lockfile (vía Git)
`renv::revert()`	Revierte el lockfile a una versión anterior del historial
`renv::activate()`	Activa renv en la sesión actual (o cambia de perfil con `profile=`)
`renv::deactivate()`	Desactiva renv en el proyecto (mantiene los archivos)
`renv::deactivate(clean = TRUE)`	Elimina por completo la infraestructura de renv del proyecto
`renv::paths$root()`	Devuelve la ruta de la caché/raíz global de renv en el sistema

---

1.19 18. Límites de renv: lo que NO resuelve

La propia documentación oficial es explícita sobre esto, y conviene tenerlo presente para no asumir una falsa sensación de seguridad total:

- **Versión de R**: `renv` registra qué versión de R se usó, pero no instala ni cambia versiones de R por ti (no puede, porque corre dentro de la propia R). Para gestionar múltiples versiones de R en una misma máquina, la documentación oficial sugiere herramientas externas como `rig`.
- **Pandoc**: como se explicó en el caso de uso 5, Pandoc no viaja dentro del lockfile de `renv`, aunque `rmarkdown`/Quarto dependan fuertemente de él.
- **Sistema operativo, librerías de sistema, versión del compilador**: mantener una imagen de sistema "estable" es un problema aparte; la solución recomendada oficialmente para esto es **Docker**, no `renv` por sí solo.

- **Disponibilidad futura de binarios**: si un paquete fue instalado desde un binario que luego deja de publicarse, `restore()` intentará compilar desde fuente, lo cual puede fallar si faltan dependencias de sistema.

En palabras de la propia documentación: hacer un proyecto reproducible siempre exige reflexión, no solo el uso mecánico de una herramienta. `renv` resuelve muy bien **una** parte importante del problema (los paquetes de R), no todo el problema de reproducibilidad de punta a punta.

1.20 19. Solución de problemas comunes

1.20.1 19.1. `renv::restore()` falla al intentar compilar un paquete desde fuente

Suele deberse a dependencias de sistema faltantes (ver la guía de instalación de R/RStudio, sección de "Dependencias de sistema para paquetes de R"). Instala la librería de sistema indicada en el mensaje de error y reintenta.

1.20.2 19.2. Un colaborador abre el proyecto y no ve activarse renv automáticamente

Verifica que se haya hecho commit de `.Rprofile`, `renv.lock`, `renv/settings.json` y `renv/activate.R` al repositorio Git. Si alguno de estos archivos quedó fuera del control de versiones (por ejemplo por un `.gitignore` mal configurado), el autoarranque de renv no ocurrirá en la máquina del colaborador.

1.20.3 19.3. Quiero saber si mi biblioteca actual coincide con el lockfile

```
renv::status()
```

Esto compara la biblioteca actual del proyecto contra lo registrado en `renv.lock` y reporta diferencias (paquetes instalados que no están en el lockfile, o viceversa).

1.20.4 19.4. Necesito volver a una versión anterior del lockfile (no solo a la última)

```
renv::history()   # ver versiones anteriores registradas en Git
renv::revert()    # revertir a una versión específica
```

1.20.5 19.5. El proyecto ya no necesita renv

```
renv::deactivate()              # desactiva pero conserva los archivos
renv::deactivate(clean = TRUE)  # elimina toda la infraestructura de renv del proyect
```

Para eliminar también la caché global de renv del sistema (afecta a *todos* tus proyectos con renv, úsalo con cuidado):

```
root <- renv::paths$root()
unlink(root, recursive = TRUE)
```

Y finalmente, si ya no quieres usar renv en ningún proyecto:

```
utils::remove.packages("renv")
```

1.21 20. Referencias oficiales

- **renv — Sitio oficial de documentación**: https://rstudio.github.io/renv/
- **renv — Introducción (vignette principal)**: https://rstudio.github.io/renv/articles/renv.html
- **renv — Instalación de paquetes y caché**: https://rstudio.github.io/renv/articles/package-install.html
- **renv — Fuentes de paquetes (CRAN, GitHub, Bioconductor, etc.)**: https://rstudio.github.io/renv/articles/package-sources.html
- **renv — Perfiles de proyecto**: https://rstudio.github.io/renv/articles/profiles.html
- **renv — Desarrollo de paquetes con renv**: https://rstudio.github.io/renv/articles/packages.html
- **renv — Integración continua (CI)**: https://rstudio.github.io/renv/articles/ci.html

- **renv — Uso con Docker**: https://rstudio.github.io/renv/articles/docker.html
- **renv — Uso con Posit Connect**: https://rstudio.github.io/renv/articles/rsconnect.html
- **renv — Preguntas frecuentes (FAQ)**: https://rstudio.github.io/renv/articles/faq.html
- **renv — Referencia de funciones**: https://rstudio.github.io/renv/reference/index.html
- **renv — Repositorio oficial en GitHub**: https://github.com/rstudio/renv
- **RStudio IDE — Guía de entornos con renv**: https://github.com/rstudio/rstudio/blob/main/docs/user/rstudio/ide/guide/environments/r/renv.qmd

---

Edison Achalma Economista — Universidad Nacional de San Cristóbal de Huamanga (UNSCH) Ayacucho, Perú ORCID: 0000-0001-6996-3364

2 Publicaciones Similares

Si te interesó este artículo, te recomendamos que explores otros blogs y recursos relacionados que pueden ampliar tus conocimientos. Aquí te dejo algunas sugerencias:

1. 011 Instalacion De R
2. 012 Configurar Entorno Virtual R
3. 013 Lo Que Debemos Saber De R
4. 02 Manipulacion De Datos
5. 03 Visualizacion De Datos
6. 04 Modelo De Machine Learning I Analisis Exploratorio
7. 05 Modelo De Machine Learning Ii Modelo De Clasificacion
8. 06 Modelo De Machine Learning Iii Modelo De Regresion
9. 07 Modelo De Machine Learning Iv Tex Mining

Esperamos que encuentres estas publicaciones igualmente interesantes y útiles. ¡Disfruta de la lectura!