Revista Innovación y Software
Vol. 7, No. 2, Mes Septiembre - Febrero, 2026
ISSN: 2708-0935
Pág. 50-58
https://revistas.ulasalle.edu.pe/innosoft

Esta obra está bajo una Licencia Creative Commons Atribución 4.0 Internacional.

Tipo de artículo: Artículos originales
Temática: Ingeniería de software
Recibido: 29/05/2026 | Aceptado: 23/07/2026 | Publicado: 30/09/2026

Identificadores persistentes:
DOI: 10.48168/innosoft.s30.a370
ARK: ark:/42411/s30.a370

Mejorando los llamados a paquetes de R con annotater

annotater: Enhancing library load calls in R

Luis D. Verde Arregoitia[1][0000-0001-9520-6543]*, **Juan Cruz Rodríguez**[2]

[1]Laboratorio de Macroecología Evolutiva, Red de Biología Evolutiva, Instituto de Ecología, A.C. Xalapa, Veracruz, 91073, México. luis@liomys.mx
[2]FAMAF - Facultad de Matemática, Astronomía, Física y Computación, Universidad Nacional de Córdoba. Córdoba X5000, Argentina. jcrodriguez@unc.edu.ar

*Autor para correspondencia: luis@liomys.mx

Resumen
Los paquetes y extensiones sirven para extender las capacidades de un lenguaje de programación, y trabajar con código fuente y scripts permite documentar y replicar los flujos de trabajo. Actualmente, en el entorno del lenguaje R la cantidad y diversidad de paquetes disponibles pueden ser abrumadoras. En este sentido, puede ser difícil entender el motivo o finalidad de los paquetes que se están cargando dentro de un proyecto. Para ayudar a reducir la falta de contexto que suele existir en archivos con código, una opción es agregar información directamente como comentarios en el código. Estas anotaciones dentro el código están dirigidas al lector humano y no se ejecutan, y se usan para proporcionar información adicional o aclaraciones sobre los pasos presentes en un script. `annotater` es un paquete para R para automatizar los comentarios sobre llamados a paquetes en archivos de R o en archivos de texto que permiten bloques de código ejecutable como R Markdown o Quarto (archivos .rmd y .qmd, respectivamente).

Palabras claves: código de R, comentarios, reproducibilidad

Abstract
Extensions and packages extend the capabilities of a programming language, and working with source code and scripts rather than interactively lets us document and repeat our workflows. However, in the R language ecosystem, the sheer number and diversity of existing packages can be overwhelming. In this context, the purpose of individual packages or their role in projects can become unclear. To address this lack of context, one practical approach is to add information directly to scripts using code comments. Code comments are annotations within code meant for the human reader, not the machine, meant to provide additional information or clarity to what is being executed. `annotater` *is an R package for automated commenting of library load calls in R scripts, or text-based formats that allow for embedded code blocks such as R Markdown and Quarto (*`.rmd`* and* `.qmd` *files, respectively).*

Keywords: *inline comments, R code, reproducibility*

Revista Innovación y Software
Vol. 7, No. 2, Mes Septiembre - Febrero, 2026
ISSN: 2708-0935
Pág. 50-58
https://revistas.ulasalle.edu.pe/innosoft

Introduction

R (https://www.r-project.org/) is a powerful and flexible programming language and interactive environment that is now widely used for statistical computing, data analysis and visualization, and with a growing role in production environments. R has gained wide acceptance and is now used in many areas of scientific computation [1]. As open source software in which users are free to use, change, and distribute code [2], one of R's key strengths is its extensive ecosystem of packages. Base R, or the standard installation without any additional code libraries, can already let users load, transform, analyze, and plot data. R also provides a programming syntax for building and documenting specialized tools to address tasks across multiple disciplines. Packages are the fundamental unit for sharing R code and extend the base functionality of the R language. A package bundles together code, data, documentation, and tests, and is easy to share with others [3]. Individuals and groups can create and add their own contributions to extend the language [4] and there are now tens of thousands of additional packages available in the R ecosystem.

Reflecting its widespread use across disciplines, R boasts a rich and active community of users and developers that contribute to package development, ensuring that there are packages for virtually any task in any scientific discipline or in industry. This collaborative environment is conducive to innovation and diversity, with new features continuously added and improved upon [5]. As a downside, the sheer number and variety of existing packages can be overwhelming. In this context, the purpose of individual packages or their role in a given project or workflow can become unclear. When opening a script, the role of a loaded package may not be evident, requiring manual investigation and interactively stepping through the code to understand its purpose. This might mean interrupting the workflow to check the documentation or search the web to understand more about the loaded packages. This context switching [6] can slow down code review, collaboration, and ultimately reduce productivity, especially when there are many dependencies or when code is shared between users with different backgrounds, skill levels, and personal preferences. For example, users can choose from or mix and match functions, methods and objects from different package 'families'for data manipulation (such as the dplyr/data.table dialetcs [7]), spatial data work, or statistical modeling frameworks.

In addition, tracking the exact versions and sources of loaded packages is important for ensuring the reproducibility of analyses and results. In research, the replication of results and findings is the standard by which scientific claims are evaluated [8]. For a robust, reproducible analysis, script-based workflows (i.e. documenting the sequence of tasks from start to finish) are crucial. These scripts should contain all the steps that link raw data to clean data to final analysis outputs [9]. Furthermore, when code and supporting data for a particular analysis are provided, information on the computational environment used to perform the analyses should also be shared to ensure accurate replication. In the context of R and R packages, and especially those in fast-moving fields, using the stable vs. development version of a package might mean the difference between a

workflow failing or succeeding or between obtaining the same results as the ones users wish to replicate. The specific source and version of an R package used in a script can be critical, as different versions may contain different features, bug fixes, or produce slightly different results, highlighting the importance of tracking this information for reproducibility.

Good data management and reproducible research or programming practices include creating seprate data dictionaries, readme files, and detailed instructions for reproducing the code being shared [10]. Additionally, another way to record these important details is within the code itself as code comments. Ultimately, the practice of adding comments to package load calls offers numerous benefits without breaking the execution of code or being overly distracting. These annotations, meant for human readers [11], can significantly aid in teaching and learning R, streamline code and data sharing among collaborators, improving the replicability of research materials, and generally lead to faster comprehension of what is happening within a script.

annotater

Most scripts load numerous packages, which may not have self-explanatory names and are often loaded without mentioning their purpose, source, or which specific functions and datasets are actually used. This lack of explicit information does not imply bad coding practices. In fact, it is often said that well written code does not need any inline comments when the purpose for each step is obvious [12]. However, adding useful information as unobtrusive comments can lead to self-documented and understandable code, ultimately improving individual and collaborative workflows. The `annotater` package was developed to address this unmet need in R for improving code comprehension regarding loaded packages in scripts.

Identifying which parts of a script rely on specific packages and their components can be challenging, making it harder to refactor code, manage dependencies, or identify unused packages. Identifying unused packages within a script is particularly valuable practice for several reasons. Loading unnecessary packages introduces overhead, consuming memory and potentially slowing down script execution. Moreover, having users install and attach packages that are never actually used can lead to confusion, frustration, unnecessary system-level dependencies, or even namespace conflicts in which functions with the same name become masked and the order in which libraries were attached to the search path becomes important. Streamlining the list of loaded packages makes the code cleaner and easier to understand, especially during first contact with a new script or code base.

Features and examples

Upon installation, R packages already include useful details that can be leveraged to automate the creation of these informative comments. For an installed package, the `packageDescription()` function parses and returns its `DESCRIPTION` file, which is a basic building block for all packages that records useful information. `annotater` uses regular expressions to identify package load calls in a script or text file, and can gather relevant information for each loaded package from either the DESCRIPTION file or by parsing all the code in a script to note the functions, methods, or datasets being used from each package.

These annotations can be particularly useful for sharing code with others, as a way to provide immediate context about why each package is being used and for what purpose. Code can be annotated interactively using the package functions or through addins (an interactive approach to execute functions from a package through a drop-down menu) in the popular RStudio IDE. Support for using these addins in the the newer Positron IDE is forthcoming.

The following annotations are supported. The code blocks below show the output of the different features on small scripts. These examples have line breaks which are not present in the actual output.

Add package titles

Package titles are one line descriptions of what a package does. These titles are always plain text, their length is generally consistent (<65 characters to avoid truncation in different listings), and in the case of packages from CRAN (the Comprehensive R Archive Network), the wording is reviewed by a dedicated team to ensure informative and concise titles. Adding titles next to each load call can already provide a good notion of why a package is being loaded, especially for those with names that are derived from acronyms, abbreviations, wordplay, international languages, or references to popular culture.

```
library(brms)    # Bayesian Regression Models using 'Stan'
library(caper)   # Comparative Analyses of Phylogenetics and Evolution in R
library(readr)   # Read Rectangular Text Data
library(picante) # Integrating Phylogenies and Ecology
```

Add package installation sources and versions

R packages can be obtained from various sources and repositories, reflecting the collaborative and evolving nature of the R community. The most common repository is the Comprehensive R Archive Network (CRAN), which hosts a vast collection of stable, programatically-tested, and human-reviewed packages. Other sources

include Bioconductor, specializing in packages for bioinformatics and computational biology, and development platforms like GitHub, GitLab, Bitbucket, and Codeberg, where developers share the latest package versions, often before official release. annotater supports various sources including CRAN, GitHub, GitLab, Bioconductor, Posit Package Manager (RSPM), and the R-universe (https://r-universe.dev/).

```
library(brms)    # [github::paul-buerkner/brms] v2.22.11
library(caper)   # CRAN v1.0.3
library(readr)   # Posit RSPM v2.1.5
library(picante) # CRAN v1.8.2
```

All functions in `annotater`, including those for adding source and version, are compatible with both `library()` and `p_load()` calls when loading packages with the `pacman` [13] package management tools. Most annotation functions in the package align the commenting or hash symbol vertically for readability, and the source and version annotation function also adds the source of the package in square brackets before the version number.

```
# add source and version to pacman call
library(readr) # Posit RSPM v2.1.5
pacman::p_load(
  caper,   # CRAN v1.0.3
  picante  # CRAN v1.8.2
)
```

Identify the functions and datasets being used from each package

Code in a script can be parsed and tokenized so that all functions or datasets that appear in a script can be identified, and then matched with the exported functions or datasets from each package being loaded in a script.

For functions and methods:

```
library(brms)    # No used functions found
library(caper)   # No used functions found
library(readr)   # read_csv
library(picante) # df2vec
```

```
dat <- read_csv("mdata.csv")
df2vec(dat, colID = Y1)
```

For datasets:

```
library(caper)   # shorebird.data
library(readr)   # No loaded datasets found
library(picante) # No loaded datasets found
```

```
data(shorebird)
hist(shorebird.data$F.Mass)
```

Expand popular metapackages into their loaded components

Metapackages or package universes are ways of easily installing and loading a set of related packages using a single command. While this approach of loading multiple packages using an 'alias' is convenient, especially for interactive work, loading metapackages may attach many more functions, methods, and datasets than are strictly required for a particular workflow. This can increase overhead and the potential for conflicts and confusion. Therefore, explicitly loading only the specific component packages needed can be advantageous for efficiency and clarity. `annotater` can expand `tidyverse` [14], `easystats` [15], and `tidymodels` [16] into their core components. After this step, other functions can then identify unused packages to streamline a script and/or make the code easier to follow.

The call `library(tidyverse)` will be expanded into:

```
####
library(ggplot2)
library(tibble)
library(tidyr)
library(readr)
library(purrr)
library(dplyr)
library(stringr)
library(forcats)
```

```
library(lubridate)
####
```

The component packages are listed on separate lineas and in the same order as they are loaded by the metapackage, and the block of code is delimited by comment lines intentionally.

Add session information

`annotater` supports adding R and RStudio versions, platform, and operating system of the local machine as comments to the beginning of a script.

On a machine running Linux:

```
# R version 4.5.0 (2025-04-11)
# Platform: x86_64-pc-linux-gnu
# Running under: Linux Mint 22.1
# Rstudio 2025.5.0.496 (Mariposa Orchid)
#
```

On the same machine a year later:

```
# R version 4.6.0 (2026-04-24)
# Platform: x86_64-pc-linux-gnu
# Running under: Linux Mint 22.3
# Rstudio 2026.4.0.526 (Globemaster Allium)
#
```

Concluding remarks

`annotater` is available on GitHub, CRAN, and R-universe and has a dedicated website for documentation (https://annotater.liomys.mx/). Since its release on CRAN, the package has been downloaded over 17,000 times. Community adoption can be inferred from public code searches for patterns commonly generated by `annotater`. For example, GitHub code searches for strings such as "`) # CRAN v`", "`) # Create elegant`", and "`) # A grammar`" —which typically annotate library calls for packages like `dplyr` and `ggplot2`— return

over 2,000 results. These matches to likely fingerprints of package use suggest that a significant number of users are using `annotater` to automatically append version and title comments to their package load calls.

It is worth noting that Large Language Model (LLM) tools can now generate inline explanations for loaded packages based on the context in a file and on the model's training data about what different packages do. Users of chatbots, integrated assistants, and autocompletion engines may now be familiar with how LLM-powered tools readily add comments next to package load calls. However, `annotater` represents a more parsimonious approach with distinct practical advantages. `annotater` runs locally in R, requiring no internet access, incurring no usage fees, and eliminating the need for setting up local models or managing API keys. Most importantly, package information is obtained directly from users' installations, avoiding hallucinations or other issues related to the source and copyright of training data for external LLM tools.

The `annotater` package provides a valuable solution by offering a non-invasive method to automatically add informative comments alongside package load calls. By annotating scripts with package titles, repository sources, versions, and even the functions and datasets being used, `annotater` significantly enhances code clarity and provides essential information for reproducibility and maintenance.

Acknowledgments

The authors acknowledge the LatinR (`https://latinr.org/`) community (Conferencia Latinoamericana sobre Uso de R en Investigación + Desarrollo) for ongoing feedback and promotion of the package.

Author contributions

Luis D. Verde Arregoitia: Conceptualization, Software, Supervision, Writing – original draft, Writing – review & editing. Juan Cruz Rodríguez: Software, Writing – original draft, Writing – review & editing.

Referencias

[1] S. J. Eglen, "A quick guide to teaching R programming to computational biology students," *PLOS Computational Biology*, vol. 5, no. 8, p. e1000482, 2009. [Online]. Available: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1000482

[2] N. Thieme, "R generation," *Significance*, vol. 15, no. 4, pp. 14–19, 2018. [Online]. Available: https://rss.onlinelibrary.wiley.com/doi/10.1111/j.1740-9713.2018.01169.x

[3] H. Wickham and J. Bryan, *R Packages*, 2nd ed. O'Reilly Media Inc., 2023.

[4] J. M. Chambers, *Extending R*. Chapman and Hall/CRC, 2017.

[5] K. Bodwin, "Keynote: Keep R weird," presented at UseR!, Salzburg, Austria, 2024.

[6] G. Wilson. (2021) Task interruption in software development projects. It Will Never Work in Theory. [Online]. Available: https://neverworkintheory.org/2021/08/09/task-interruption-in-software-development-projects.html

[7] V. Arel-Bundock. (2025) data.table vs. base vs. dplyr. Arelbundock.com. [Accessed March 9, 2026]. [Online]. Available: https://arelbundock.com/posts/dt_tb_df/

[8] R. D. Peng, "Reproducible research in computational science," *Science*, vol. 334, no. 6060, pp. 1226–1227, 2011. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.1213847

[9] N. Cooper and P. Y. Hsing, *A Guide to Reproducible Code in Ecology and Evolution*. British Ecological Society, 2017. [Online]. Available: https://www.britishecologicalsociety.org/wp-content/uploads/2017/12/guide-to-reproducible-code.pdf

[10] C. Lewis, *Data Management in Large-Scale Education Research*, 1st ed. Chapman and Hall/CRC Press, 2024.

[11] A. Filazzola and C. Lortie, "A call for clean code to effectively communicate science," *Methods in Ecology and Evolution*, vol. 13, no. 10, pp. 2119–2128, 2022. [Online]. Available: https://besjournals.onlinelibrary.wiley.com/doi/abs/10.1111/2041-210X.13961

[12] G. Wilson, J. Bryan, K. Cranston, J. Kitzes, L. Nederbragt, and T. K. Teal, "Good enough practices in scientific computing," *PLOS Computational Biology*, vol. 13, no. 6, p. e1005510, 2017. [Online]. Available: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005510

[13] T. W. Rinker and D. Kurkiewicz, "pacman: Package management for R," Buffalo, New York, 2018. [Online]. Available: http://github.com/trinker/pacman

[14] H. Wickham *et al.*, "Welcome to the Tidyverse," *Journal of Open Source Software*, vol. 4, no. 43, p. 1686, 2019. [Online]. Available: https://joss.theoj.org/papers/10.21105/joss.01686

[15] D. Lüdecke *et al.*, "easystats: Framework for easy statistical modeling, visualization, and reporting," 2022, cRAN.

[16] M. Kuhn and J. Silge, *Tidy Modeling with R: A Framework for Modeling in the Tidyverse*. O'Reilly Media, Inc., 2022.