PharmaSUG 2025 - Paper OS-229

Identifying Breaking Changes in R Packages: pkgdiff

David J. Bosak, Archytas Clinical Solutions

ABSTRACT

SAS® software is famous for its stability and backward compatibility. SAS programs written in the 1980s will still work today, often with few or no changes. R is a different story. In R, programs you wrote a few months ago and were working perfectly suddenly can have errors or warnings. After some investigation, you realize a package you were using has changed, and broken your program. This is called a “breaking change”. The purpose of this paper is to explore an R package named “pkgdiff”, which aims to identify and help manage breaking changes. The paper will summarize the major functions of the package, and explain how it can be used to reduce breakages in your code.

INTRODUCTION

If you have worked with R for any length of time, you have probably experienced a “breaking change”. This is when a program you wrote some months ago all the sudden gets an error or warning, even though nothing has changed in your program. Upon investigation, you discover, to your great surprise, that a function you were using from a particular package has been completely removed from the package. Or perhaps it was renamed to something else. In either case, your code is now broken, and requires maintenance to keep running properly.

Open-source software has a reputation for being chaotic like this. R package developers are not required to maintain backward compatibility with previous versions of the package. They can remove a function or function parameter at any time, and for any reason. Sometimes they will give you a warning in advance of such removal. Sometimes they will not.

The purpose of the **pkgdiff** package is to identify breaking changes in R packages. **pkgdiff** aims to give you an objective way of discovering and anticipating breaking changes, with a view toward increasing the stability and reliability of your code. As the use of R spreads in the pharmaceutical industry, tamping down the chaos in our R programs has become increasingly important.

The **pkgdiff** package meets this need. **pkgdiff** functions are easy to use. You can use them in scripts, and also use them on the command line. They give you an easy way to assess the stability of a package, determine if a package upgrade will break your code, and identify which functions or function parameters will break. With **pkgdiff**, you can even assess the stability of an entire repository.

This paper will present an overview of **pkgdiff**. The paper will provide a brief description of the major functions, and examples for each. The paper will then give you some general tips on how to reduce breaking changes in your code base.

Before describing the functions, let’s first understand how this package works.

HOW IT WORKS

R packages are published to an organization called the Comprehensive R Archive Network (CRAN). CRAN maintains a web site with all publicly available R packages, and some basic information about them. CRAN keeps all versions of a package, including superseded versions. These packages live on in the CRAN archive, and the archive is publicly available. Anyone can download any version of any package that has ever been published to CRAN.

The **pkgdiff** package works by pulling the current and previous versions of packages from CRAN. It then downloads them, and extracts information from each of the package versions. This information is then compared and consolidated into R objects. These R objects are then passed back to the user, where they may be viewed or queried programmatically.

VIEWING INFORMATION ABOUT A PACKAGE

The most basic thing we can do with **pkgdiff** is view some information about a package. Viewing information about a package is accomplished with the function `pkg_info()`. This function will extract some fields from the package description file. You can display these fields in the console, or access them from code. Here is an example:

```
> pkg_info("stringr")
# A package info object: stringr package
- Version: v1.5.1
- Release Date: 2023-11-14
- Title: Simple, Consistent Wrappers for Common String Operations
- Maintainer: Hadley Wickham <hadley@posit.co>
- License: MIT + file LICENSE
- Description: A consistent, simple and easy to use set of wrappers around
the fantastic 'stringi' package. All function and argument names (and
positions) are consistent, all functions deal with "NA"'s and zero
length vectors in the same way, and the output from one function is
easy to feed into the input of another.
- Depends: R (>= 3.6)
- Imports: cli, glue (>= 1.6.1), lifecycle (>= 1.0.3), magrittr, rlang (>= 1.0.0), stringi
(>= 1.5.3), vctrs (>= 0.4.0)
- Suggests: covr, dplyr, gt, htmltools, htmlwidgets, knitr, rmarkdown, testthat (>= 3.0.0),
tibble
- Downloads/Month: 1147815
- Repository: CRAN
- Cached: TRUE
- Functions: 68
```

The example above shows some description fields for the “stringr” package. The “stringr” package was written by Hadley Wickham at Posit®. As you can see, it is a very popular package, with over 1,000,000 downloads per month.

The `pkg_info()` function allows you to view this information very easily in the R console. It is even easier than navigating to the “stringr” web site.

You can also check out all the versions of a package. You can do that with the `pkg_versions()` function, like this:

```
> pkg_versions("stringr")
   Package Version              FileName    Release Size
1  stringr   1.5.1  stringr_1.5.1.tar.gz 2023-11-14 172K
2  stringr   1.5.0  stringr_1.5.0.tar.gz 2022-12-02 172K
3  stringr   1.4.1  stringr_1.4.1.tar.gz 2022-08-21 133K
4  stringr   1.4.0  stringr_1.4.0.tar.gz 2019-02-10 133K
5  stringr   1.3.1  stringr_1.3.1.tar.gz 2018-05-10 116K
6  stringr   1.3.0  stringr_1.3.0.tar.gz 2018-02-19 116K
7  stringr   1.2.0  stringr_1.2.0.tar.gz 2017-02-18  92K
8  stringr   1.1.0  stringr_1.1.0.tar.gz 2016-08-19  62K
9  stringr   1.0.0  stringr_1.0.0.tar.gz 2015-04-30  34K
10 stringr   0.6.2  stringr_0.6.2.tar.gz 2012-12-06  20K
11 stringr   0.6.1  stringr_0.6.1.tar.gz 2012-07-25  20K
12 stringr     0.6    stringr_0.6.tar.gz 2011-12-08  20K
13 stringr     0.5    stringr_0.5.tar.gz 2011-06-30  18K
14 stringr     0.4    stringr_0.4.tar.gz 2010-08-24  16K
15 stringr     0.3    stringr_0.3.tar.gz 2010-02-15  11K
16 stringr     0.2    stringr_0.2.tar.gz 2009-11-16  10K
17 stringr  0.1.10 stringr_0.1.10.tar.gz 2009-11-09 6.8K
```

The above data frame shows every release of the “stringr” package, the version number, the release date, etc. This table is useful if you just want to know what all the versions of a package are, or are looking for a particular version number. More information on `pkg_info()` and `pkg_versions()` can be found on the **pkgdiff** web site here https://pkgdiff.r-sassy.org/reference/pkg_info.html and here https://pkgdiff.r-sassy.org/reference/pkg_versions.html.

ASSESSING PACKAGE STABILITY

The next thing you might want to know about a package is its stability. You can examine the stability of a package with the function `pkg_stability()`. Just call the function, and pass the name of the package you are interested in:

```
> pkg_stability("stringr")
# A stability score: stringr package
- Age: 15.33 years
- Score: 98.3
- Assessment: Very Stable
- Version Range: 0.1.10/1.5.1
- Release Range: 2009-11-09/2023-11-14
- Release Count: 17
- Breaking Releases: 4
- Data:
   Package Version              FileName    Release   Size AF AP RF RP BC TF
1  stringr   1.5.1  stringr_1.5.1.tar.gz 2023-11-14   172K  0  0  0  0  0 68
2  stringr   1.5.0  stringr_1.5.0.tar.gz 2022-12-02   172K 19  6  0  0  0 68
3  stringr   1.4.1  stringr_1.4.1.tar.gz 2022-08-21   133K  0  0  0  0  0 49
4  stringr   1.4.0  stringr_1.4.0.tar.gz 2019-02-10   133K  3  3  0  0  0 49
5  stringr   1.3.1  stringr_1.3.1.tar.gz 2018-05-10   116K  0  0  0  0  0 46
6  stringr   1.3.0  stringr_1.3.0.tar.gz 2018-02-19   116K  6  0  3  0  1 46
7  stringr   1.2.0  stringr_1.2.0.tar.gz 2017-02-18    92K  1  2  0  0  0 43
8  stringr   1.1.0  stringr_1.1.0.tar.gz 2016-08-19    62K  4  1  0  0  0 42
9  stringr   1.0.0  stringr_1.0.0.tar.gz 2015-04-30    34K 12  4  0  2  1 38
10 stringr   0.6.2  stringr_0.6.2.tar.gz 2012-12-06    20K  0  0  0  0  0 26
11 stringr   0.6.1  stringr_0.6.1.tar.gz 2012-07-25    20K  1  0  0  0  0 26
12 stringr     0.6    stringr_0.6.tar.gz 2011-12-08    20K  1  0  0  0  0 25
13 stringr     0.5    stringr_0.5.tar.gz 2011-06-30    18K  3  0  0  0  0 24
14 stringr     0.4    stringr_0.4.tar.gz 2010-08-24    16K  3  1  2  0  1 21
15 stringr     0.3    stringr_0.3.tar.gz 2010-02-15    11K  5  0  1  0  1 20
16 stringr     0.2    stringr_0.2.tar.gz 2009-11-16    10K  1  1  0  0  0 16
17 stringr  0.1.10 stringr_0.1.10.tar.gz 2009-11-09   6.8K 15 15  0  0  0 15
```

Now this is getting interesting!

The `pkg_stability()` function tells us that the “stringr” package is over 15 years old, has had 17 releases, and has had 4 breaking releases. A “breaking release” is a release where at least one function or function parameter was removed from the previous version. The “stringr” package has a stability score of 98.3, which is “very stable”. The table at the bottom shows us every package version, the release date, and some data about how many functions were added or removed from the previous version.

Here is a dictionary explaining the meaning of the stability data columns:

Column	Description
Package	The package name
Version	The package version
FileName	The name of the package file
Release	The release date
Size	The package file size
AF	Added functions
AP	Added parameters
RF	Removed functions
RP	Removed parameters
BC	Breaking change
TF	Total number of functions

The stability data shows us exactly which versions had breaking changes, and how many. This information can be very useful when planning a version upgrade, or when making package selection decisions.

For more information on the `pkg_stability()` function, see the function reference at https://pkgdiff.r-sassy.org/reference/pkg_stability.html. For an elaboration on how the stability score is calculated, see the stability score vignette at https://pkgdiff.r-sassy.org/articles/pkgdiff-stability.html.

COMPARING PACKAGE VERSIONS

Notice that the stability data for the "stringr" package shows that the package had a breaking change between versions "1.2.0" and "1.3.0". Is there a way that we can see what those changes were?

Yes! Differences between package versions can be discovered using the `pkg_diff()` function. This function compares two versions of a package, and determines the differences in terms of functions and function parameters. The `pkg_diff()` function is actually the core functionality of the **pkgdiff** package. Most of the functions in **pkgdiff** are based off this comparison utility.

Let's see how it works:

```
> pkg_diff("stringr", "1.2.0", "1.3.0")
# A difference object: stringr package
- Comparing: v1.2.0/v1.3.0
- Breaking Changes: TRUE
- Added Functions:
  - str_flatten()
  - str_glue()
  - str_glue_data()
  - str_remove()
  - str_remove_all()
  - str_squish()
- Removed Functions:
  - ignore.case()
  - perl()
  - str_join()
```

The above comparison shows us that between "stringr" versions "1.2.0" and "1.3.0", six functions were added and three functions were removed. It further shows us the names of the changed functions. This information would be very handy if you happened to be upgrading "stringr". You could then search your code and find out if you were using any of the removed functions.

Note that the return value of the `pkg_diff()` function is an object. The added and removed functions are returned as lists, and can be manipulated programmatically.

For more information on the `pkg_diff()` function, see the function documentation at https://pkgdiff.r-sassy.org/reference/pkg_diff.html.

ASSESSING ENTIRE REPOSITORY

The above package comparison is nice if you are only upgrading one package. But what if you are upgrading several packages? Or even an entire repository?

pkgdiff provides tools to assess packages in bulk. It can gather stability scores on an entire repository, and also determine breakages for an entire repository.

Yet before examining these tools, first we need to find out what exactly is in our repository.

PACKAGES IN REPOSITORY

The `pkg_repo()` function can tell us what packages and versions of those packages are in a specified repository. The function is easy to use. You can find out the packages in your current repository like this:

```
> pkg_repo(ver = "current")
# A package repo object
- Repo Version: 4.4
```

```
- Packages:
           Package    Version
1            abind      1.4-8
2          admiral      1.1.1
3       admiraldev      1.1.0
4             ards      0.1.1
5         askpass      1.2.0
6       assertthat      0.2.1
7        backports      1.5.0
8        base64enc      0.1-3
9             bigD      0.3.0
10     BiocManager    1.30.23
11             bit      4.0.5
12           bit64      4.0.5…
```

The `pkg_repo()` function returns a data frame that contains a row for each package in the repository, and the version of that package. If you want the packages from your current repository, pass the special value "current" on the version parameter. You can also pass a specific version number if you have multiple repositories.

For more information on the `pkg_repo()` function, see the documentation at https://pkgdiff.r-sassy.org/reference/pkg_repo.html.

REPOSITORY STABILITY

Earlier we looked at how to assess the stability of a single package. But what if you want to assess more than one package? Is there an easy way to do that?

The `repo_stability()` function can assess the stability of a vector of packages. Here we can pass a vector of packages to the function, and it will return a stability score for each one:

```
> pkgs <- c("curl", "dbplyr", "dplyr", "purrr", "stringr", "tidymodels")
> repo_stability(pkgs)
Getting stability score for package 'curl'...
Getting stability score for package 'dbplyr'...
Getting stability score for package 'dplyr'...
Getting stability score for package 'purrr'...
Getting stability score for package 'stringr'...
Getting stability score for package 'tidymodels'...
# A repo stability object
- Run Datetime: 2025-03-15 16:16 UTC
- Summary:
     Package     FV    LV         FR         LR TR BR Score        Assessment
1       curl    0.2 6.2.1 2014-11-20 2025-02-19 51  1  98.0       Very Stable
2     dbplyr  1.0.0 2.5.0 2017-06-09 2024-03-19 23 10  74.1          Unstable
3      dplyr    0.1 1.1.4 2014-01-16 2023-11-17 45 20  87.5 Somewhat Unstable
4      purrr  0.1.0 1.0.4 2015-09-28 2025-02-05 18  6  92.1            Stable
5    stringr 0.1.10 1.5.1 2009-11-09 2023-11-14 17  4  98.3       Very Stable
6 tidymodels  0.0.1 1.3.0 2018-07-27 2025-02-21 14  0 100.0           Perfect
```

Awesome! This feature allows you to easily determine if a set of packages meet your desired level of stability.

Here is a dictionary describing the meaning of the repo stability data columns:

Column	Description
Package	The package name
FV	First version
LV	Latest version
FR	First release date
LR	Last release date
TR	Total number of releases

BR	Number of breaking releases
Score	The stability score
Assessment	The stability assessment

Using the `pkg_repo()` function from above, you may then gather stability data on your entire repository:

```
> repo <- pkg_repo(ver = "current")
> res <- repo_stability(repo$Package)
```

Be prepared for the above code to run for some time. The function needs to retrieve stability data for potentially hundreds of packages. When the function returns, it will show you the stability score for each package in your repository, displayed as above. Here is the full documentation for `repo_stability()`: https://pkgdiff.r-sassy.org/reference/repo_stability.html.

PLANNING A VERSION UPGRADE

Let's say you are planning to upgrade your R version, and pull down the latest versions of all packages in your repository. Wouldn't it be great to know if there will be any breaking changes?

You can find out if there will be any breaking changes using the `repo_breakages()` function. The function accepts two data frames, representing the source package set and the target package set. You can create the source and target package sets using the `pkg_repo()` function from above. Let's see how:

```
# Create source package set
r1 <- pkg_repo(ver = "current")

# Create target package set
# Pass package vector from source set
r2 <- pkg_repo(r1$Package, ver = "latest")
```

You now have two data frames, mapping the current versions of the packages in your repo to the latest versions on CRAN. Next we can use `repo_breakages()` to look for breaking changes between the two version sets:

```
# Look for breakages in repo upgrade
res <- repo_breakages(r1, r2)
Comparing admiral v1.1.1/v1.2.0
Comparing admiraldev v1.1.0/v1.2.0
Comparing askpass v1.2.0/v1.2.1
Comparing BiocManager v1.30.23/v1.30.25
Comparing bit v4.0.5/v4.6.0
Comparing bit64 v4.0.5/v4.6.0-1
Comparing broom v1.0.6/v1.0.7
Comparing bslib v0.8.0/v0.9.0
Comparing car v3.1-2/v3.1-3…
```

Again, this function can take some time to run. When `repo_breakages()` completes, it will return a data frame with a row for each package, the source and target versions, and a flag showing whether or not there are breaking changes for that package:

```
# Show breakages table
res$Summary
           Package   Version1   Version2 Breakages
1            abind      1.4-8      1.4-8     FALSE
2          admiral      1.1.1      1.2.0      TRUE
3       admiraldev      1.1.0      1.2.0      TRUE
4             ards      0.1.1      0.1.1     FALSE
5          askpass      1.2.0      1.2.1     FALSE
6       assertthat      0.2.1      0.2.1     FALSE
7        backports      1.5.0      1.5.0     FALSE
8        base64enc      0.1-3      0.1-3     FALSE
```

```
9             bigD      0.3.0      0.3.0     FALSE
10     BiocManager    1.30.23    1.30.25     FALSE
11             bit      4.0.5      4.6.0     FALSE
12           bit64      4.0.5    4.6.0-1      TRUE…
```

If there are breaking changes, the function will include a difference object for each package with breaking changes. You can access these difference objects using the "Details" item on the repo breakages object:

```
# Show details
res$Details
# A difference object: admiral package
- Comparing: v1.1.1/v1.2.0
- Breaking Changes: TRUE
- Added Functions: 3
- Added Parameters: 6
- Removed Functions:
  - country_code_lookup()
  - dose_freq_lookup()
- Removed Parameters:
  - consolidate_metadata(): check_keys
  - derive_extreme_event(): ignore_event_order
  - derive_param_exposure(): analysis_var, summary_fun, filter
  - derive_summary_records(): filter, analysis_var, summary_fun
  - derive_var_joined_exist_flag(): first_cond, filter
  - derive_var_merged_summary(): new_var, analysis_var, summary_fun
  - derive_vars_merged(): match_flag
  - event_joined(): first_cond
  - filter_joined(): first_cond, filter
  - get_summary_records(): analysis_var, summary_fun…
```

The summary table and detail comparisons give you some idea of what the effect of your upgrade will be. Whether or not you actually have breaking changes will of course depend on whether you are using any of the removed functions or parameters. You can use the above information to run scans on your code to look for any removed items. Hopefully this will help you gauge the effect of the upgrade, and help you plan your activities more effectively.

For additional information about `repo_breakages()`, see the function documentation on the web site: https://pkgdiff.r-sassy.org/reference/repo_breakages.html.

TIPS TO REDUCE BREAKING CHANGES

Having seen how to use the major functions in the **pkgdiff** package, let's now present some general tips to reduce and manage breaking changes in R. There are two broad categories: advance tips and deferred tips.

ADVANCE TIPS

Advance tips are things you can to up front to reduce breaking changes, before you start programming:

- **Reduce Dependances**: The first thing you can do to reduce breaking changes is to reduce the number of packages you use in the first place. Every package you use exposes you to potential breaking changes from that package. So, obviously, the fewer package you use, the less likely it is that you will have breaking changes.
- **Learn Base R**: Base R is far more stable than contributed packages. The R Core team is acutely aware that the entire R ecosystem is dependent on their work. They try very hard to make the Base R packages backward compatible and as stable as possible. You can therefore make your code base more stable by choosing Base R functions over equivalent functions from contributed packages.
- **Select Stable Packages**: Of course, there are some cases when Base R does not provide the functionality that you need, or does not provide it in a convenient enough way. In those cases, you will have to add some contributed packages. Best practice, however, is to select the most

stable packages possible. If there are multiple packages that have the functionality you need, use `repo_stability()` to evaluate your choices. The information provided by that function will allow you to pick the most stable and reliable package that meets your requirements.

DEFERRED TIPS

Once you have already selected a set of packages, and started writing code against it, the Advance Tips above will no longer work. Here are some tips for reducing breaking changes once your code is written:

- **Perform Code Reviews**: Programmers just learning R tend to use a lot of packages unnecessarily. They think they need to use a different package for almost every line of code. However, it is not true. A lot of things can be done in Base R, without exposing your code to potential chaos. Your organization should get in the habit of performing code reviews: let senior programmers look over the code of junior programmers, and teach them how to write R code with the minimum number of dependencies.
- **Scan Code for Known Breaking Changes**: Once your repo is in place, and you are getting ready to start upgrading, you can use the functions in **pkgdiff** to anticipate upgrade problems. You can use `pkg_diff()` and `repo_breakages()` to identify trouble, and help make your upgrades smoother.
- **Code Refactoring**: If you still find yourself inundated with breaking changes, it might be time to start eliminating troublesome packages. Identify the worst offenders and start refactoring your code to get rid of problematic packages. You can use `pkg_stability()` and `repo_stability()` to identify unstable packages and perhaps select better options.

CONCLUSION

For programmers used to programming in SAS®, the number of breaking changes in R can be very surprising and distressful. The **pkgdiff** package allows you to get a better handle on these changes, and can help you reduce them in several ways. First, **pkgdiff** gives you a way to identify and select the most stable packages up front. Then once you select and start using those packages, it can help you anticipate breaking changes, and deal with them in a more organized and systematic manner. With conscientious usage, **pkgdiff** can reduce the breaking changes chaos, and make your R programming experience more pleasant.

REFERENCES

Bosak, David. “Introduction to **pkgdiff**”, 2025. Available at https://pkgdiff.r-sassy.org.

ACKNOWLEDGMENTS

The author would like to thank Brian Varney and Kevin Putschko of Experis for their valuable suggestions and feedback during the early stages of **pkgdiff** development. The package is much improved because of their contributions.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

David J. Bosak
Archytas Clinical Solutions
dbosak01@gmail.com
www.r-sassy.org