

9.3: Common pitfalls in R scripting

Patient: Doc, it hurts when I do this.

Doctor: Don't do that.

To those readers who want to dig deeper, this section continues to explain why R scripts do not sometimes work, and how to solve these problems.

Advices

Use the Source, Luke!..

The most effective way to know what is going on is to look on the source of R function of interest.

Simplest way to access source is to type function name without parentheses. If the function is buried deeper, then try to use `methods()` and `getAnywhere()`.

In some cases, functions are actually not R code, but C or even Fortran. Download R source, open it and find out. This last method (download source) works well for simpler cases too.

Keep it simple

Try not to use any external packages, any complicated plots, any custom functions and even some basic functions (like `subset()`) without absolute need. This increases reproducibility and makes your life easier.

Analogously, it is better to avoid running R through any external system. Even macOS R shell can bring problems (remember history issues?). RStudio is a great piece of software but it is prone to the same problem.

Learn to love errors and warnings

They help! If the code issues error or warning, it is a symptom of something wrong. Much worse is when the code does not issue anything but produce unreliable results.

However, warnings sometimes are really boring, especially if you know what is going on and why do you have them. On macOS it is even worse because they colored in red... So use `suppressWarnings()` function, but again, only when you know what you are doing. You can think of it as of headache pills: useful but potentially dangerous.

Subselect by names, not numbers

Selecting columns by numbers (like `trees[, 2:3]`) is convenient but dangerous if you changed your object from the original one. It is always better to use longer approach and select by names, like

When you select by name, be aware of two things. First, selection by one name will return `NULL` and can make new column if anything assigned on the right side. This works only for `[]` and `$`:

Second, negative selection works only with numbers:

Try to avoid name your objects with reserved words (`?Reserved`). Be especially careful with `T`, `F`, and `return`. If you assign them to any other object, consequences could be unpredictable. This is, by the way another good reason to write `TRUE` instead of `T` and `FALSE` instead of `F` (you cannot assign anything to `TRUE` and `FALSE`).

It is also a really bad idea to assign anything to `.Last.value`. However, using the default `.Last.value` (it is not a function, see `?Last.value`) could be a fruitful idea.

If you modified internal data and want to restore it, use something like `data(trees)`.

The Case-book of Advanced R user

The Adventure of the Factor String

By default, R converts textual string into factors. It is useful to make contrasts but bring problems into many other applications.

To avoid this behavior in `read.table()`, use `as.is=TRUE` option, and in data frame operations, use `stringsAsFactors=FALSE` (or the same name global option). Also, always control mode of your objects with `str()`.

A Case of Were-objects

When R object undergoes some automatic changes, sooner or later you will see that it changes the type, mode or structure and therefore escapes from your control. Typically, it happens when you make an object smaller:

Data frames and matrices normally *drop dimensions* after reduction. To prevent this, use `[, , drop=FALSE]` argument. There is even function `mat.or.vec()`, please check how it works.

Factors, on other hand, *do not* drop levels after reductions. To prevent, use `[, drop= TRUE]`.

Empty zombie objects appear when you apply malformed selection condition:

To avoid such situations (there are more pitfalls of this kind), try to use `str()` (or `Str()` from `asmisc.r`) every time you create new object.

A Case of Missing Compare

If missing data are present, comparisons should be thought carefully:

A Case of Outlaw Parameters

Consider the following:

Problem is that R frequently ignores illegal parameters. In some cases, this makes debugging difficult.

However, not all functions are equal:

And some functions are even more weird:

The general reason of all these different behaviors is that functions above are internally different. The first case is especially harmful because R does not react on your misprints. Be careful.

A Case of Identity

Similar by consequences is an example when something was selected from list but the name was mistyped:

This is not a bug but a *feature* of lists and data frames. For example, it will allow to grow them seamlessly. However, mistypes do not raise any errors and therefore this might be a problem when you debug.

The Adventure of the Floating Point

This is well known to all computer scientists but could be new to unexperienced users:

What is going on? Elementary, my dear reader. Computers work only with 0 and 1 and do not know about floating points numbers.

Instead of exact comparison, use “near exact” [all.equal\(\)](#) which is aware of this situation:

A Case of Twin Files

Do this small exercise, preferably on two computers, one under Windows and another under Linux:

On Linux, there are two files with proper numbers of dots in each, but on Windows, there is only one file named [Ex.pdf](#) but with *three* dots! This is even worse on macOS, because typical installation behaves like Windows but there are other variants too.

Do not use uppercase in file names. And do not use any other symbols (including spaces) except lowercase ASCII letters, underscore, 0–9 numbers, and dot for extension. This will help to make your work portable.

A Case of Bad Grammar

The style of your scripts could be the matter of taste, but not always. Consider the following:

This could be interpreted as either

or

Always keep spaces around assignments. Spaces after commas are not so important but they will help to read your script.

A Case of Double Dipping

Double comparisons do not work! Use logical concatenation instead.

There is no [c\(\)](#) for factors in R, result will be not a factor but numerical codes. This is concerted with a nature of factors.

However, if you really want to concatenate factors and return result as a factor, [?c](#) help page recommends:

A Case of Bad Font

Here is a particularly nasty error:

Unfortunately, well-known problem. It is always better to use good, visually discernible monospaced font. Avoid also lowercase “l”, just in case. Use “j” instead, it is much easier to spot.

By the way, error message shows the problem because it stops printing exactly where is something wrong.

This page titled 9.3: Common pitfalls in R scripting is shared under a [Public Domain](#) license and was authored, remixed, and/or curated by Alexey Shipunov via source content that was edited to the style and standards of the LibreTexts platform.