R-bloggers

R news and tutorials contributed by hundreds of R bloggers

- Home
- About
- RSS
- add your blog!
- Learn R
- R jobs���
- Contact us

## Welcome!

Here you will find daily **news and tutorials about R**, contributed by hundreds of bloggers. There are many ways to **follow us -**
By e-mail:

Your e-mail here

Subscribe

50942 readers BY FEEDBURNER

On Facebook:

**If you are an R blogger yourself** you are invited to add your own R content feed to this site (**Non-English** R bloggers should add themselves- here)

## Jobs for R-users

- (Junior) Data Analyst – Berlin
- Associate Researcher (Data Analytics)
- Lead Data Scientist @ Stabio, Ticino, Switzerland
- Senior Scientist, Translational Informatics
- Senior Scientist, Translational Informatics @ Vancouver, BC, Canada

## Recent Posts

- Using bwimge R package to

in images of natural structures
- New package: GetEdgarData
- Easy introduction to Offensive Programming
- Le Monde puzzle [#1114]
- upcoming AI-related courses
- Free R/datascience Extract: Evaluating a Classification Model with a Spam Filter
- Parsing Sda Pages
- Super Solutions for Shiny Apps #4 of 5: Using R6 Classes
- Simulating data with Bayesian networks
- JAMA retraction after miscoding – new Finalfit function to check recoding
- Finding free science books from Springer
- Merge MLP And CNN in Keras
- Shiny 1.4.0
- Strange Attractors: an R experiment about maths, recursivity and creative coding
- What are Your Use Cases for rOpenSci Tools and Resources?

## Other sites

- Jobs for R-users
- SAS blogs

# R Code – Best practices

September 1, 2018
By The R Trader

Nothing is more frustrating than a long piece of code with no standard way of naming elements, presenting code or organizing files. It's not only unreadable but more importantly not reusable. Unfortunately, unlike other programming languages, R has no widely accepted coding best practices. Instead there has been various attempts to put together a few sets of rules. This post is trying to fill the gap by summarizing and/or extracting what I found relevant in those various attempts. It also includes some tips I came up with after years of using R on a daily basis.

**1 – Naming conventions**

R has no naming conventions that are generally agreed upon. As a newcomer to R it's useful to decide which naming convention to adopt.

- There are 5 naming conventions to choose from:
  - *alllowercase*: e.g. adjustcolor
  - *period.separated*: e.g. plot.new
  - *underscore_separated*: e.g. numeric_version
  - *lowerCamelCase*: e.g. addTaskCallback
  - *UpperCamelCase*: e.g. SignatureMethod
- Strive for names that are concise and meaningful
- Generally, variable names should be nouns and function names should be verbs.
- Strive for names that are concise and meaningful
- Generally, variable names should be nouns and function names should be verbs.
- Not exported and helper functions always start with "."
- Local variables and functions are all in small letters and in "." syntax (do.something, get.xyyy). It makes it easy to distinguish local vs global and therefore leads to a cleaner code.
- File names should be meaningful and end in .R.
- Pick one naming convention and stick to it. My suggestion:
  - **Files**: underscore_separated, all lower case: e.g. numeric_version
  - **Functions**: period.separated, all lower case: e.g. my.function
  - **Variables**: lowerCamelCase: e.g. addTaskCall

## 2 – Files organisation

They way files are organised helps making the code more readable. Similarly, the way the code is organised within a file has a significant impact on readability. Files might also have specific purposes. Some might contain only functions that will be used by other files, some might be used to update packages etc…

### 2.1 – How to organise the files within a project?

- Use the project facility of RStudio each time you start working on a new project
- Keep all of the source files for a project in one directory and use relative paths to access them
- Separate files that contain functions that will be used by other parts of the code from the core of the code
- Consider what working directory you are in when sourcing a script.

### 2.2 – How to organise the code within each file?

- Start each file with a comment saying who wrote it and when, what it contains, and how it fits into the larger program.
- Then load all required packages.
- Then source required files if any
- Then your code starts
- Break code up into separate files (generally <2000–3000 lines).

```
#############################################################
## Stuff I have to do
## thertrader@gmail.com - Feb 2018
#############################################################

##############################
# 0 - Load librairies
##############################
library(zoo)
library(xts)

##############################
# 1 - Source file
##############################
dataPath &amp;amp;lt;- "C:/some_directory/some_sub_directory/"
dataFile &amp;amp;lt;- "some_functions.R"
source(paste0(dataPath,dataFile))

##############################
# 2 - Start my code
##############################
myPlot &amp;amp;lt;- plot(data,type="l")
```

### 2.3 – Files of functions

- Write functions (and even a package) to automate things. Packages require a lot of discipline, documentation, and structure, which really help to enforce best practices

- Put function definitions at the top of your file (if not too many). Function names can also be retrieved directly within RStudio.
- Each function should have a single, focused task
- If a function starts to get really complicated, consider separating parts out as separate functions. (Think reuse)
- Precede each function with a comment regarding its task and the format of the input and output.

```
#############################################################
## Date Functions
##
## dd.mm.yyyy.to.date: 22.10.2004 to a date
## yyyy-mm-dd.to.date: 2008-07-22 to a date
## dd-mm-yyyy.to.date: 12-05-2001 to a date
##
## thertrader@gmail.com - Jan 2008...
############################################################

## 22.10.2004 to a date
dd.mm.yyyy.to.date &amp;amp;lt;- function(theDate) {
 myDate &amp;amp;lt;- as.Date(theDate, format = "%d.%m.%Y")
 return(myDate)
}

## 2008-07-22 to a date
yyyy-mm-dd.to.date &amp;amp;lt;- function(theDate) {
 myDate &amp;amp;lt;- as.Date(theDate, format = "%Y-%m-%d")
 return(myDate)
}

## 12-05-2001 to a date
dd-mm-yyyy.to.date &amp;amp;lt;- function(theDate) {
 myDate &amp;amp;lt;- as.Date(theDate, format = "%m-%d-%Y")
 return(myDate)
}
```

#### 2.4 – Files with packages and addins

- R and packages can be updated with the installr command on a (Windows) computer that already has R installed but when installing R on a brand new computer or a new operating system another method is needed
- For installation on a brand new computer or a new operating system, It's very handy to keep a file with all packages and addins that you need and install them right after R has been installed for the first time. Below is a template file

```
##################################################
## Install Package Automatically - to run after R has been upgraded/installed
##
## thertrader@gmail.com - Aug 2010...
##################################################

install.packages(c(
"data.table",
"DEoptim",
"devtools",
"dplyr",
"DT",
"dygraphs",
"ggplot2",
"xts",
"PerformanceAnalytics"))

install.packages("taskscheduleR", repos = "http://www.datatailor.be/rcube", type = "source")
```

### 3 – Syntax

- Place spaces around all infix operators (=, +, -, <-, etc.).
- Use <-, not =, for assignment.
- Use comments to mark off sections of code.
- Comment your code with care. Comments should explain the why, not the what
- Each line of a comment should begin with the comment symbol and a single space
- An opening curly brace should never go on its own line and should always be followed by a new line; a closing curly brace should always go on its own line, unless followed by else.
- Always indent the code inside the curly braces.
- Keep your lines less than 80 characters.This is the amount that will fit comfortably on a printed page at a reasonable size. If you find you are running out of room, this is probably an indication that you should encapsulate some of the work in a separate function.

### 4 – Miscellaneous

- Use version control when you start sharing code. RStudio ships with integrated facilities to access GitHub and SVN
- Keep track of versions (of data, of functions).
- Always start with a clean environment instead of saving the workspace.
- Keep track of session information in your project folder.
- Keep track of the memory used by your program.
- Have someone else review your code: hence this document

This post has been written using my own experience and the following documents:

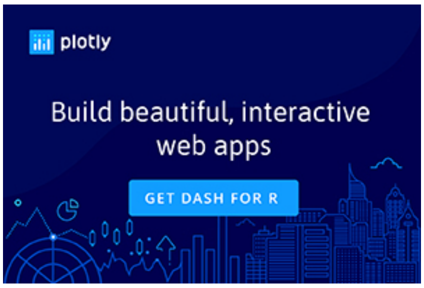
- R Style Guide
- Efficient R Programming
- What best practices do you use for programming in R? (StackOverflow)
- The State of Naming Conventions in R
- Best Practices for writing R
- Advanced R: Style Guide
- Consistent Naming Conventions in R
- Google's R Style Guide

## Search R-bloggers

## Most visited articles of the week

## Sponsors

DataCamp
Learn R
by doing.


plotly
Build beautiful, interactive web apps
GET DASH FOR R

TRY R Studio Team
DOWNLOAD QUICKSTART VM

Beginner's Guide to
Spatial, Temporal and Spatial-Temporal Ecological Data Analysis with R-INLA
Zuur, Ieno, Saveliev

ODSC
BUILD AI AND ML SKILLS IN R
SAN FRANCISCO
OCT 29TH – NOV 1ST

R-TRAININGS | FRANKFURT
Einführung & Machine Learning
November 2019
JETZT ANMELDEN
eoda

GitLab
DevOps toolchain too c
Learn to ship coc
200% faster.
GET THE WHITEPAPI

## Jobs for R users

- (Junior) Data Analyst – Berlin
- Associate Researcher (Data Analytics)
- Lead Data Scientist @ Stabio, Ticino, Switzerland
- Senior Scientist, Translational Informatics
- Senior Scientist, Translational Informatics @ Vancouver, BC, Canada
- Data Scientist @ New York, United States
- Empirical Research Librarian

**Full list of contributing R-bloggers**

**R-bloggers** was founded by Tal Galili, with gratitude to the R community.
Is powered by WordPress using a bavotasan.com design.