

2.6: How to start with R

Launching R

Typically, you launch R from the desktop icon or application menu. To launch R from the terminal, type:
—and you will see the R screen.

It is even possible to launch R on the remote UNIX server without any graphical system running. In that case, all plots will be written in one PDF file, [Rplots.pdf](#) which will appear in the working directory

If you know how to work with R, it is a good idea to check the fresh installation typing, for example, [plot\(1:20\)](#) to check if graphics works. If you are a novice to R, proceed to the next section.

First steps

After you successfully opened R, it is good to understand how to exit. After entering empty parentheses, be sure to press [Enter](#) and answer “n” or “No” on the question:

This simple example already shows that any *command* (or *function*, this is almost the same) in R has an *argument* inside round brackets, parentheses. If there is no argument, you still need these parentheses. If you forget them, R will show the *definition* of the function instead of quitting:

(For the curious, “bytecode” means that this function was compiled for speed, “environment” shows the way to call this function. If you want to know the function code, it is not always work to call it without parentheses; see the reference card for more advanced methods.)

How to know more about function? Call the *help*:

or simply



Now back to the [?q](#). If you read this help text thoroughly, you might conclude that to quit R *without being asked anything*, you may want to enter [q\("no"\)](#). Please try it.

“no” is the *argument* of the exit function [q\(\)](#). Actually, not exactly the argument but its *value*, because in some cases you can skip the *name* of argument. The name of argument is [save](#) so you can type [q\(save="no"\)](#). In fact, most of R functions look like [function\(name="value"\)](#); see more detail in Figure 2.6.1.

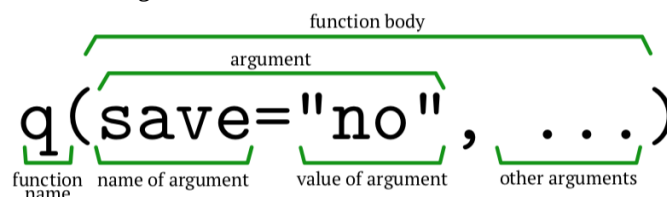


Figure 2.6.1 Structure of R command.

R is pretty liberal about arguments. You will receive same answers if you enter any of these variants:

^[2]As you see, arguments are matched by *name* and/or by *position*. In output, R frequently prints something like [\[1\]](#), it is just an *index* of resulted number(s). What is [round\(\)](#)? Run [?round](#) to find out.)

It is possible to mess with arguments as long as R “understands” what you want. Please experiment more yourself and find out why this

gives the value you probably do not want.

If you want to *know arguments* of some function, together with their default values, run [args\(\)](#):

There is also an [example\(\)](#) function which is quite useful, especially when you learn plotting with R. To run examples supplied with the function, type [example\(function\)](#). Also do not forget to check [demo\(\)](#) function which outputs the list of possible *demonstrations*, some of them are really handy, saying, [demo\(colors\)](#).

Here R shows one of its basic principles which came from Perl language: *there always more than one way to do it*. There are many ways to receive a help in R!

So default is to ask the “save” question on exit. But why does R ask it? And what will happen if you answer “yes”? In that case, two files will be written into the R working directory: binary file [.RData](#) and textual file [.Rhistory](#) (yes, their names start with a dot). First contains all objects you created during the R session. Second contains the full history of entered commands. These files will be *loaded automatically* if you start R from the same directory, and the following message will appear:

[\[Previously saved workspace restored\]](#)

Frequently, this is *not* a desirable behavior, especially if you are just learning R and therefore often make mistakes. As long as you study with this book, we strongly recommend to answer “no”.

If you by chance answered “yes” on the question in the end of the previous R session, you might want to remove unwanted files: (*Be extremely careful* here because R deletes files silently! On macOS, file names might be different; in addition, it is better to uncheck Read history file on startup in the Preferences menu.)

If you are bored from answering questions again and again, and at the same time do not want to enter `q("no")`, there is a third way. Supply R starting command with option `-no-save` (it could be done differently on different operation systems), and you will get rid of it^[3].

How to type

When you work in R, the previous command could be called if you press “arrow up” key (↑). This is extremely useful and saves plenty of time, especially when you need to run the command similar to the preceding. On some systems, there is also *backward search* (`Ctrl+R` on Linux) which is even more efficient than arrow up.

If you mistakenly typed the long command and want to wipe it without supplying to R, there is `Ctrl+U` key (works on Linux and Windows).

If you run R in the terminal which has no apparent way to scroll, use `Shift+PgUp` and `Shift+PgDn`.

Another really helpful key is the `Tab`. To see how it works, start to type long command like `read.t...` and then press `Tab`. It will call *completion* with suggests how to continue. Completion works not only for commands, but also for objects, command arguments and even for file names! To invoke the latter, start to type `read.table(")` and then press `Tab` once or twice; all files in the working directory will be shown.

Remember that all brackets (braces, parentheses) and quotes *must be always closed*. One of the best ways to make sure of it is to *enter opening and closing brackets together*, and then return your cursor into the middle. Actually, graphic R on macOS does this by default.

Pair also all quotes. R accepts two types of quotes, single `'...'` and double `"..."` but they *must be paired with quote of the same type*. Good question is when do you need quotes. In general, *quotes belong to character strings*. Rule of thumb is that objects *external* to R need quotes whereas *internal* objects could be called without quotes.

R is sensitive to the case of symbols. Commands `ls()` and `Ls()` are *different*! However, spaces do not play any role. These commands are the same:

Do not be afraid of making errors. On the contrary,

Make as many mistakes as possible!

The more mistakes you do when you learn, the less you do when you start to work with R on your own.

R is frequently literal when it sees a mistake, and its *error messages* will help you to decipher it. Conversely, R is perfectly silent when you do well. If your input has no errors, R usually says *nothing*.

It is by the way really hard to crash R. If nevertheless your R seems to hang, press `Esc` button (on Linux, try `Ctrl+C` instead).

Yet another appeal to users of this book:

Experiment!

Try unknown commands, change options, numbers, names, remove parentheses, load any data, run code from Internet, from help, from your brain. The more you experiment, the better you learn R.

How to play with R

Now, when we know basics, it is time to do something more interesting in R. Here is the simple task: convert the sequence of numbers from 1 to 9 into the table with three columns. In the spreadsheet or visual statistical software, there will be several steps: (1) make two new columns, (2–3) copy the two pieces into clipboard and paste them and (4) delete extra rows. In R, this is just one command:

(Symbol `<-` is an *assignment* operator, it is read *from right to left*. `bb` is a new R *object* (it is a good custom to name objects with double letters, less chances to intersect with existent R objects). But what is `1:9`? Find it^[4] yourself. Hint: it is explained in few pages from this one.)

Again from the above: How to select the sample of 100 trees in the big forest? If you remember, our answer was to produce 100 random pairs of the coordinates. If this forest is split into 10,000 squares (100×100), then required sample might look like:

(First, `expand.grid()` was used above to create all 10,000 combinations of square numbers. Then, powerful `sample()` command randomly selects 100 rows from whatever number of rows is in the table `coordinates`. Note that your results will be likely different since `sample()` uses the *random number generator*. Finally, this `samples.rows` was used as an *index* to randomly select 100 rows (pairs of coordinates) from 10,000 combinations. What is left for you now is to go to the forest and find these trees :-))

Let us now play dice and cards with R:

(Note here `outer()` command which combines values, `paste()` which joins into the text, `rep()` which repeats some values, and also the `replace=TRUE` argument (by default, `replace` is `FALSE`). What is `replace=FALSE`? Please find out. Again, your results could be different from what is shown here. Note also that `TRUE` or `FALSE` must always be fully uppercased.)

Overgrown calculator

But the most simple way is to use R as an advanced calculator:

(Note that you can skip leading zero in decimal numbers.)

The more complicated example, “`log10(((sqrt(sum(c(2, 2))))^2)*2.5)`” will be calculated as follows:

1. The vector will be created from two twos: `c(2, 2)`.
2. The sum of its elements will be counted: `2+2=4`.
3. Square root calculated: `sqrt(4)=2`.
4. It is raised to the power of 2: `2^2=4`.
5. The result is multiplied by 2.5: `4*2.5=10`.
6. Decimal logarithm is calculated: `log10(10)=1`.

As you see, it is possible to embed pairs of parentheses. It is a good idea to count opening and closing parentheses before you press [Enter](#); these numbers must be *equal*. After submission, R will open them, pair by pair, from the deepest pair to the most external one.

So R expressions are in some way similar to Russian doll, or to onion, or to artichoke (Figure 2.6.2), and to analyze them, one should peel it.



Figure 2.6.2 You may think of R syntax as of “artichoke”.

Here is also important to say that R (similar to its TeX friend) belongs to one of the most deeply thought software. In essence, R “base” package covers almost 95% needs of the common statistical and data handling work and therefore external tools are often redundant. It is wise to keep things simple with R.

If there are no parentheses, R will use precedence rules which are similar to the rules known from the middle school.

For example, in `2+3*5` R will multiply first (`3*5=15`), and only then calculate the sum (`2+15=17`). Please check it in R yourself.

How to make the result 25? Add parentheses.

Let us feed something mathematically illegal to R. For example, square root or logarithm of `-1`:

If you thought that R will crash, that was wrong. It makes `NaN` instead. `NaN` is *not a number*, one of *reserved words*.

What about division by zero?

References

1. There is command `Xpager()` in the `asmisc.r` collection of commands, it allows to see help in the separate window even if you work in terminal.
2. Within parentheses immediately after example, we are going to provide comments.
3. By the way, on Linux systems you may exit R also with `Ctrl+D` key, and on Windows with `Ctrl+Z` key.
4. Usually, small exercises are boldfaced.

This page titled 2.6: How to start with R 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.