Scripts in R

Introduction to Computational BioStatistics with R

Alexey Fedoseev

September 30, 2025

Installing R packages

R makes it easy to install additional packages.

```
> install.packages("ggplot2")
Installing package into '/Users/alexey/R/x86_64-pc-linux-gnu-library/3.5'
(as 'lib' is unspecified)
--- Please select a CRAN mirror for use in this session ---
...
* DONE (ggplot2)
...
```

To load the package use this command:

```
> library(ggplot2)
```

Navigation in R

Sometimes you need to know where you are, so that you can find data, and other things.

```
> getwd()
[1] "/Users/alexey/Documents/"
> setwd("code")
> getwd()
[1] "/Users/alexey/Documents/code"
> dir()
[1] "vectorUtils.R"
```

Creating your own collections of functions

As you develop your research you will build a collection of functions which do all sorts of wonderful things. How should you manage your collections of functions?

- You should only have one copy of any given function. Put it in a file and use it when you need it.
- The one copy you keep should be well-tested, well-commented, and trustworthy.
- If you have five copies of a function, how do you remember which one you should use?
- Keep your functions in files. Keep related functions in the same file.
- Name your functions and files sensibly.

But how do I access a function once I've written it?

Scripts

Let us say you have created a helpful function `vectorToString` and saved in a file `vectorUtils.R`

```
# vectorUtils.R
# Concatenates the vector using semicolons between the elements
vectorToString <- function(vec) {
  return(paste(vec, collapse = "; "))
}
```

The `source` command gives you access to the functions stored in a file. The `source` command reads and executes the whole file as if you were typing the lines at the R prompt.

```
> source("vectorUtils.R")
> popular.cars <- c("Audi", "Mercedes-Benz", "BMW")
> cat("Popular German car manufacturers are",
+     vectorToString(popular.cars), "\n")
Popular German car manufacturers are Audi; Mercedes-Benz; BMW
```

Reloading your file

If we decide to separate our values in the vector using commas instead of semicolons we need to reload the file `vectorUtils.R` in our R prompt after we made the changes.

```
# vectorUtils.R
# Concatenates the vector using commas between the elements
vectorToString <- function(vec) {
  return(paste(vec, collapse = ", "))
}
```

```
> cat("Popular German car manufacturers are",
+     vectorToString(popular.cars), "\n")
Popular German car manufacturers are Audi; Mercedes-Benz; BMW
> source("vectorUtils.R")
> cat("Popular German car manufacturers are",
+     vectorToString(popular.cars), "\n")
Popular German car manufacturers are Audi, Mercedes-Benz, BMW
```

Scripts

Putting our commands in a script `popular_cars.R` allows us to save the work.

```r
# vectorUtils.R
# Concatenates the vector using commas between the elements
vectorToString <- function(vec) {
  return(paste(vec, collapse = ", "))
}
```

```r
# popular_cars.R
source("vectorUtils.R") # Importing function vectorToString
popular.cars <- c("Audi", "Mercedes-Benz", "BMW")
cat("Popular car manufacturers are", vectorToString(popular.cars), "\n")
```

```
user@scinet code $ Rscript popular_cars.R
Popular car manufacturers are Audi, Mercedes-Benz, BMW
```

Be careful if you copy-and-paste from the R prompt into an editor. Do not copy the > and + prompts into your script.

Command line arguments

Now you want to improve your program by adding more car manufacturers. So far we can do that by changing the variable `popular.cars` in the script `popular_cars.R`. However changing the file every time is tedious. Instead we can specify the car brands as parameters for the script, therefore you do not store the values in the script but rather you receive them when you run your script from Bash.

Let us create a new file `popular_cars_cli.R` (CLI stands for Command-line interface) with the following contents.

```
# popular_cars_cli.R
args <- commandArgs(trailingOnly = TRUE)
cat("The command line arguments are", args, "\n")
```

Run your script `popular_cars_cli.R` with several parameters.

```
user@scinet code $ Rscript popular_cars_cli.R Audi Volkswagen BMW Porsche
The command line arguments are Audi Volkswagen BMW Porsche
```

Command line arguments

Now add the following lines in the file `popular_cars_cli.R`.

```
# Importing function vectorToString
source("vectorUtils.R")

# All arguments have to contain car brands
popular.cars <- args

cat("Popular car manufacturers are", vectorToString(popular.cars), "\n")
```

And then run your script with several command line arguments.

```
user@scinet code $ Rscript popular_cars_cli.R Audi Volkswagen BMW Porsche
The command line arguments are Audi Volkswagen BMW Porsche
Popular car manufacturers are Audi, Volkswagen, BMW, Porsche
```

Command line arguments

Notice that the variable args is a vector of strings. It means that if you want to compare the first argument to something else, you should directly reference the first argument.

```
# popular_cars_cli.R
args <- commandArgs(trailingOnly = TRUE)
cat("The command line arguments are", args, "\n")
source("vectorUtils.R") # Importing function vectorToString
if (args[1] == "Lada") { # Lada is not a popular brand
  popular.cars <- args[-1] # Exclude Lada from popular cars
} else {
  popular.cars <- args # All arguments have to contain car brands
}
cat("Popular car manufacturers are", vectorToString(popular.cars), "\n")
```

```
user@scinet code $ Rscript popular_cars_cli.R Lada Audi Volkswagen BMW Porsche
The command line arguments are Lada Audi Volkswagen BMW Porsche
Popular car manufacturers are Audi, Volkswagen, BMW, Porsche
```

Command line arguments

We can improve our script by excluding the unpopular cars brands. Let us define a variable containing the unpopular brands and using already known to us command %in% determine which car brands we want to exclude.

```
> unpopular.cars <- c("Lada", "Opel", "Buick", "Peugeot", "Lifan")
> c("Buick", "Volkswagen") %in% unpopular.cars
[1]  TRUE FALSE
```

```
# popular_cars_cli.R
args <- commandArgs(trailingOnly = TRUE)
source("vectorUtils.R") # Importing function vectorToString
unpopular.cars <- c("Lada", "Opel", "Buick", "Peugeot", "Lifan")
popular.cars <- args[!args %in% unpopular.cars] # exclude unpopular brands
cat("Popular car manufacturers are", vectorToString(popular.cars), "\n")
```

```
user@scinet code $ Rscript popular_cars_cli.R Buick Audi Peugeot BMW Buick
Popular car manufacturers are Audi, BMW
```

Utility files and driver programs

The general framework we will use for this course is to have *utilities files* and *driver scripts*.

The *utilities files* will contain the definitions of the functions you need to work on your data (like our file `vectorUtils.R`).

It is common to have multiple utilities files, containing functions for performing different types of analyses.

The *driver script* "drives" the analysis, calling the functions in the *utilities files* as needed to accomplish whatever you are doing.

The functions in the *utilities file* are often used by a variety of different *driver scripts* (like our files `popular_cars.R` and `popular_cars_cli.R`).

Utility files

Let us add more functionality to our `vectorToString` function. Currently the function works like this:

```
> vectorToString(c("Porsche", "Mercedes-Benz", "Audi"))
[1] "Porsche, Mercedes-Benz, Audi"
```

However, I would like to see the output like that: `"Porsche, Mercedes-Benz and Audi"`. Notice that the last two elements are separated using `and` instead of a comma.

If you just go and modify our function `vectorToString` it may affect the programs that already use if functionality, so we will be more careful.

You need to decide whether the functionality you are adding to the function is small enough to just slightly modify the function without changing much the code that uses it.

If you feel that the changes are going to be significant, it is best to just leave this function as it is now and create another function with additional functionality, meaning you would have a simple version that you can use as a backup.

Improving functionality

The changes we are about to add to our function `vectorToString` are not that significant and we can actually make them without changing the code that uses this function.

Let us modify the function `vectorToString` in our file `vectorUtils.R`.

```
vectorToString <- function(vec, last.and = FALSE) {
  if (last.and & length(vec) > 1) { # Skip if vec has only one element
    # Concatenate all elements except the last one using commas
    str.with.commas <- paste(vec[-length(vec)], collapse = ", ")
    # Concatenate the last element using "and"
    str.with.and <- paste(str.with.commas, "and", vec[length(vec)])
    return(str.with.and)
  } else
    # Concatenate all elements using commas
    return(paste(vec, collapse = ", "))
}
```

Testing functionality

To avoid any surprises we need to test if our improved function still gives the same result as before.

```
> source("vectorUtils.R")
> vectorToString(c("Porsche", "Mercedes-Benz", "Audi"))
[1] "Porsche, Mercedes-Benz, Audi"
```

As we see, with the help of the default arguments we do not need to change our *driver script* `popular_cars_cli.R` as by default the function will use commas to concatenate the strings as before.

Let us test the new functionality.

```
> vectorToString(c("Porsche", "Mercedes-Benz", "Audi"), TRUE)
[1] "Porsche, Mercedes-Benz and Audi"
```

Testing functionality

After verifying that the function works properly we can turn on the new functionality in the *driver script* `popular_cars_cli.R`.

```
# popular_cars_cli.R
args <- commandArgs(trailingOnly = TRUE)
source("vectorUtils.R") # Importing function vectorToString
unpopular.cars <- c("Lada", "Opel", "Buick", "Peugeot", "Lifan")
popular.cars <- args[!args %in% unpopular.cars] # exclude unpopular brands
cat("Popular car manufacturers are",
    vectorToString(popular.cars, last.and = TRUE), "\n")
```

```
user@scinet code $ Rscript popular_cars_cli.R Porsche Mercedes-Benz Audi
Popular car manufacturers are Porsche, Mercedes-Benz and Audi
```

Testing functionality

Once you have written the basic functionality of your program you need to test it with different parameters to see how it works in the situations you did not think of in the first place, for example using only one brand:

```
user@scinet code $ Rscript popular_cars_cli.R Porsche
Popular car manufacturers are Porsche
user@scinet code $ Rscript popular_cars_cli.R
Popular car manufacturers are
```

Testing functionality

It would be nice to show an example on how to use our script to someone who does not know how to run it.

```
# popular_cars_cli.R
args <- commandArgs(trailingOnly = TRUE)
# Show the usage example if no parameters were specified
if (length(args) == 0) {
   cat("usage example: Rscript popular_cars_cli.R Porsche Mercedes-Benz\n")
   quit()
}
...
```

```
user@scinet code $ Rscript popular_cars_cli.R
usage example: Rscript popular_cars_cli.R Porsche Mercedes-Benz
```

Notice that we do not need `else` in our `if` statement since the command `quit()` completely stops the program and no code below this command will run.

Testing functionality

We also need to correctly handle the situation when only one car brand is specified.

```
# popular_cars_cli.R
...
popular.cars <- args[!args %in% unpopular.cars] # exclude unpopular brands
if (length(popular.cars) > 1) {
   manufacturers.string <- "manufacturers are"
} else if (length(popular.cars) == 1) {
   manufacturers.string <- "manufacturer is"
} else { # Show the full list of unpopular cars
   cat(vectorToString(unpopular.cars, last.and = TRUE),
      "are not very popular.\n")
   quit()
}
# Create a full sentence using proper copulas (is or are)
cat("Popular car", manufacturers.string,
    vectorToString(popular.cars, last.and = TRUE), "\n")
```

Summing up

Now we have a functional program!

```
user@scinet code $ Rscript popular_cars_cli.R
usage example: Rscript popular_cars_cli.R Porsche Mercedes-Benz
user@scinet code $ Rscript popular_cars_cli.R Lifan
Lada, Opel, Buick, Peugeot and Lifan are not very popular.
user@scinet code $ Rscript popular_cars_cli.R Lada Buick
Lada, Opel, Buick, Peugeot and Lifan are not very popular.
user@scinet code $ Rscript popular_cars_cli.R Lifan Buick Porsche
Popular car manufacturer is Porsche
user@scinet code $ Rscript popular_cars_cli.R Porsche
Popular car manufacturer is Porsche
user@scinet code $ Rscript popular_cars_cli.R Porsche Mercedes-Benz
Popular car manufacturers are Porsche and Mercedes-Benz
user@scinet code $ Rscript popular_cars_cli.R Porsche Mercedes-Benz Audi
Popular car manufacturers are Porsche, Mercedes-Benz and Audi
```

Further improvement

Our program works quite well, however there are still ways to improve it.

One of them is to use the command `tolower` when comparing the strings. It is useful if someone spells `lada` instead of `Lada`.

```
> tolower(c("Lada", "Mercedes-Benz"))
[1] "lada"          "mercedes-benz"
> c("lada","opel") %in% tolower(c("Lada","Opel","Buick","Peugeot","Lifan"))
[1] TRUE TRUE
```

While developing your program you will find several use cases that you have not thought of beforehand. It is normal to revise your code accordingly and take care of such use cases later.

Meanwhile, instead of letting your program to crash unexpectedly, just display a simple message like “this feature will be implemented in later versions” to remind yourself what to focus on later and let others know that you are aware of this use case.