2. Workspace and Files

Solutions to Swirl's R Programming Exercises

06-19-2022

Acknowledgements: R Language Concepts and code questions (with minor modifications) are used here from the swirl package. https://www.r-project.org/nosvn/pandoc/swirl.html

Important note: We don't require to use library(swirl) and swirl() here because we are not going to run the R script in RStudio Console.

In this lesson, you'll learn how to examine your local workspace in R.Because different operating systems have different conventions with regards to things like file paths, the outputs of these commands may vary across machines.

However it's important to note that R provides a common API (a common set of commands) for interacting with files, that way your code will work across different kinds of computers.

Let's jump right in so you can get a feel for how these special functions work! Determine which directory your R session is using as its current working directory using getwd().

```
getwd()
```

```
## [1] "C:/r-basics/swirl/2_WorkspaceAndFiles"
```

Some R commands are the same as their equivalents commands on Linux or on a Mac. Both Linux and Mac operating systems are based on an operating system called Unix. It's always a good idea to learn more about Unix!

```
x <- 9
```

List all the files in your working directory using list.files() or dir().

```
list.files()
```

```
## [1] "mytest2.R"           "mytest3.R"           "testdir"
## [4] "testdir2"            "Workspace_Files.pdf" "Workspace_Files.rmd"
```

As we go through this lesson, you should be examining the help page for each new function. Check out the help page for list.files with the command ?list.files.

```
list.files()
```

```
## [1] "mytest2.R"           "mytest3.R"           "testdir"
## [4] "testdir2"            "Workspace_Files.pdf" "Workspace_Files.rmd"
```

One of the most helpful parts of any R help file is the See Also section. Read that section for list.files. Some of these functions may be used in later portions of this lesson.

Using the args() function on a function name is also a handy way to see what arguments a function can take.

```
 args(list.files)
```

```
## function (path = ".", pattern = NULL, all.files = FALSE, full.names = FALSE,
##     recursive = FALSE, ignore.case = FALSE, include.dirs = FALSE,
##     no.. = FALSE)
## NULL
```

Use the args() function to determine the arguments to list.files().

```
 args(list.files)
```

```
## function (path = ".", pattern = NULL, all.files = FALSE, full.names = FALSE,
##     recursive = FALSE, ignore.case = FALSE, include.dirs = FALSE,
##     no.. = FALSE)
## NULL
```

Assign the value of the current working directory to a variable called old.dir <- getwd()

```
old.dir <- getwd()
```

We will use old.dir at the end of this lesson to move back to the place that we started. A lot of query functions like getwd() have the useful property that they return the answer to the question as a result of the function.

Use dir.create() to create a directory in the current working directory called "testdir".

```
dir.create("testdir")
```

```
## Warning in dir.create("testdir"): 'testdir' already exists
```

We will do all our work in this new directory and then delete it after we are done. This is the R analog to "Take only pictures, leave only footprints."

Set your working directory to "testdir" with the setwd() command.

```
setwd("testdir")
```

In general, you will want your working directory to be someplace sensible, perhaps created for the specific project that you are working on. In fact, organizing your work in R packages using RStudio is an excellent option. Check out RStudio at http://www.rstudio.com/

Create a file in your working directory called "mytest.R" using the file.create() function.

```
file.create("mytest.R")
```

```
## [1] TRUE
```

This should be the only file in this newly created directory. Let's check this by listing all the files in the current directory.

```
 list.files()
```

```
## [1] "mytest.R"            "mytest2.R"           "mytest3.R"
## [4] "testdir"             "testdir2"            "Workspace_Files.pdf"
## [7] "Workspace_Files.rmd"
```

Check to see if "mytest.R" exists in the working directory using the file.exists() function.

```
file.exists("mytest.R")
```

```
## [1] TRUE
```

These sorts of functions are excessive for interactive use. But, if you are running a program that loops through a series of files and does some processing on each one, you will want to check to see that each exists before you try to process it.

Access information about the file "mytest.R" by using file.info().

```
file.info("mytest.R")
```

```
##          size isdir mode               mtime               ctime
## mytest.R    0 FALSE  666 2022-07-09 01:56:43 2022-07-09 01:56:43
##                        atime exe
## mytest.R 2022-07-09 01:56:43  no
```

You can use the $ operator — e.g., file.info("mytest.R")$mode — to grab specific items.

```
 file.info("mytest.R")$mode
```

```
## [1] "666"
```

Change the name of the file "mytest.R" to "mytest2.R" by using file.rename().

```
file.rename("mytest.R", "mytest2.R")
```

```
## [1] TRUE
```

Your operating system will provide simpler tools for these sorts of tasks, but having the ability to manipulate files programatically is useful. You might now try to delete mytest.R using file.remove('mytest.R'), but that won't work since mytest.R no longer exists. You have already renamed it.

Make a copy of "mytest2.R" called "mytest3.R" using file.copy().

```
 file.copy("mytest2.R", "mytest3.R")
```

```
## [1] FALSE
```

You now have two files in the current directory. That may not seem very interesting. But what if you were working with dozens, or millions, of individual files? In that case, being able to programatically act on many files would be absolutely necessary. Don't forget that you can, temporarily, leave the lesson by typing play() and then return by typing nxt().

Provide the relative path to the file "mytest3.R" by using file.path().

```
file.path("mytest3.R")
```

```
## [1] "mytest3.R"
```

You can use file.path to construct file and directory paths that are independent of the operating system your R code is running on. Pass 'folder1' and 'folder2' as arguments to file.path to make a platform-independent pathname.

```
file.path("folder1", "folder2")
```

```
## [1] "folder1/folder2"
```

Take a look at the documentation for dir.create by entering ?dir.create . Notice the 'recursive' argument. In order to create nested directories, 'recursive' must be set to TRUE.?dir.create will show you the docs.

```
?dir.create
```

```
## starting httpd help server ... done
```

Create a directory in the current working directory called "testdir2" and a subdirectory for it called "testdir3", all in one command by using dir.create() and file.path().

```
dir.create(file.path("testdir2", "testdir3"), recursive=TRUE)
```

```
## Warning in dir.create(file.path("testdir2", "testdir3"), recursive = TRUE):
## 'testdir2\testdir3' already exists
```

Go back to your original working directory using setwd(). (Recall that we created the variable old.dir with the full path for the orginal working directory at the start of these questions.)

```
setwd(old.dir)
```

It is often helpful to save the settings that you had before you began an analysis and then go back to them at the end. This trick is often used within functions; you save, say, the par() settings that you started with, mess around a bunch, and then set them back to the original values at the end. This isn't the same as what we have done here, but it seems similar enough to mention.

After you finish this lesson delete the 'testdir' directory that you just left (and everything in it)

Take nothing but results. Leave nothing but assumptions. That sounds like 'Take nothing but pictures. Leave nothing but footprints.' But it makes no sense! Surely our readers can come up with a better motto . . .

In this lesson, you learned how to examine your R workspace and work with the file system of your machine from within R.