Workspace and Files

In this lesson, you'll learn how to examine your local workspace in R and begin to explore the relationship between your workspace and the file system of your machine.

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] "/home/simon/Dropbox/Teaching/2017-2018/2017_720-R/fes720"
```

If you open RStudio by using the GUI to open a file, that will be the working directory. We will learn how to move about inside the computer and change the working directory later in this lesson.

Now, you can list all the objects in your local workspace using ls(). "Local workspace" refers to what is loaded or created in R's memory that can be accessed in this session. Closing R wipes this local memory clean.

```
ls()
```

```
##  [1] "course_dir"     "dest_dir"       "destrmd"        "initcode"
##  [5] "initpath"       "install_course" "keep_rmd"       "les"
##  [9] "lessonPath"     "meta"           "open_html"      "out"
## [13] "quiet"          "rmd_filename"   "unit"
```

If you are continuing from the previous lesson (Basic_Building_Blocks) without closing R or RStudio, you will likely notice that the variables you created are still in R's memory, or local workspace.

Similar to the previous lesson, let's assign 9 to x using x <- 9.

```
x <- 9
```

Now take another look at the objects that are in your workspace using ls().

```
ls()
```

```
##  [1] "course_dir"     "dest_dir"       "destrmd"        "initcode"
##  [5] "initpath"       "install_course" "keep_rmd"       "les"
##  [9] "lessonPath"     "meta"           "open_html"      "out"
## [13] "quiet"          "rmd_filename"   "unit"           "x"
```

What has changed? If "x" was already present from the previous class, you will have over-written it with the new content.

Now, let's begin to explore the contents of your computer…

Some R commands are the same as their equivalent commands on Linux or on a Mac. These include ls() and dir().

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

```
list.files()
```

```
## [1] "2017-autumn-schedule.ods" "2017-syllabus.html"
## [3] "2017-syllabus.md"         "fes720-2017-syllabus.pdf"
## [5] "fes720_Basic"             "fes720_Explore"
## [7] "fes720_Statistics"        "presentations"
## [9] "Workspace_and_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. This will either open up inside RStudio, or a new page in your web browser.

```
?list.files
```

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. We will discuss functions and arguments more later, but for now arguments are how you tell a function what exactly to do, including what data to work on. Most functions have sensible defaults.

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
```

You can see all the arguments and their defaults listed, similar to the help page for list.files(). Each argument has a default (NULL or FALSE in most cases). The important one for our purposes is the argument ‘path = “.”’. Remember that “path” is the address, or directory location, or folder location. In this case, the “.” indicates the current directory.

You can use “..” to look in the directory one level higher up the hierarchy. Type list.files(“..”).

```
list.files("..")
```

```
##  [1] "CARCHEDI-THESIS-2014.pdf"
##  [2] "class-plan.md"
##  [3] "Fall 2017 course schedule for distribution.xlsx"
##  [4] "fes720"
##  [5] "Gelman_communication_course_outline.pdf"
##  [6] "review_from_2016"
##  [7] "swirl_courses"
##  [8] "syllabus.pdf"
##  [9] "Teaching Statistics Using Small.docx"
## [10] "usingR.pdf"
```

We can also look in directories lower down the hierarchy, if there are any. Start typing list.files("./") and press the TAB key. This will bring up the autocomplete options for subdirectories, if there are any. Feel free to experiment. Remember, you can use play() to leave this lesson temporarily, and return using nxt().

So, these commands are equivalent to opening Windows Explorer or Mac Finder and opening/closing folders and subfolders.

What about creating and deleting folders and files? These operations are straightforward in a GUI, as you know.

Let's get back to work! First, we want to make sure that we can set things back to how they were if it all goes wrong … Assign the value of the current working directory to a variable called "old.dir".

```
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". Look at the arguments or help page if you are unsure what to do.

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

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. Remember that you could use autocomplete (TAB) to help with this.

```
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.

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

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

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

This file has the extension ".R". Every file has an extension, which tells the computer what kind of file it is and therefore what software applications are best able to open it. You are likely familiar with ".xls" and ".doc" files and their Mac equivalents.

The ".R" extension tells your computer that this is a file with R code, and the default will be to open it inside RStudio.

You can also tell your computer to do other things based on the file extension such as text highlighting. This will highlight different parts of the code, e.g., functions, brackets, numbers, etc.

In RStudio, go to Tools > Global Options, then the Appearance tab. Choose a colour scheme, etc. These options will become more useful when you start writing and saving code.

OK, so "mytest.R" 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] "2017-autumn-schedule.ods" "2017-syllabus.html"
##  [3] "2017-syllabus.md"         "fes720-2017-syllabus.pdf"
##  [5] "fes720_Basic"             "fes720_Explore"
##  [7] "fes720_Statistics"        "mytest.R"
```

```
##  [9] "presentations"            "testdir"
## [11] "Workspace_and_Files.Rmd"
```

We can also specifically 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  664 2017-08-31 11:58:07 2017-08-31 11:58:07
##                        atime  uid  gid uname grname
## mytest.R 2017-08-31 11:58:07 1000 1000 simon  simon
```

This function provides information on the size, various time stamps, and users of the file. These might be important in various programming contexts.

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

We will use “$” a lot more as we progress through the semester.

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(). If you check the help page or arguments, you will see that this function requires two pieces of information. The name of the file you want to copy(“from =”), and the name of the new copied file (“to =”).

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

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

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.

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.

What the $%^! does that mean?

Good question!

It means that in this function (file.path) you give R the names of the directories in the order they are found (i.e., the directory hierarchy). R then handles how these names are put together, accounting for the differences between Mac, Windows, Linux, etc.

Pass ‘folder1’ and ‘folder2’ as arguments to file.path() to make a platform-independent pathname.

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

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

The next task is to create a directory in the current working directory called “testdir2” and a subdirectory called “testdir3”, in one command.

This task involves a new idea, that of nesting functions: i.e., we can use a function as one of the arguments in another function.

This situation of nesting functions is similar to how in the previous lesson we used the variables “x”, “y”, and “z” as parts of arithmetic calculations as well as joined them together using c(). Remember?

First, 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
```

Notice also that dir.create() takes the argument ‘path’ (we have already used it when we made the directory ‘testdir/’, above). We can use file.path() here for this argument.

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)
```

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.

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

You can do this using the unlink() function. We also need to run this function recursively, so that it deletes all the files and folders within testdir/. Type unlink(x = “testdir/”, recursive = TRUE).

```
unlink(x = "testdir/", recursive = TRUE)
```

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. Thanks for playing!