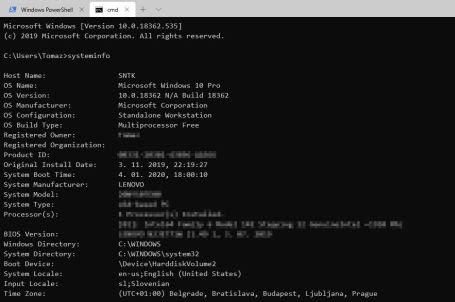
From time to time, when developing in R, working and wrangling data , preparing for machine learning projects, it comes the time, one would still need to access the operating system commands from/in R.

In this blog post, let’s take a look at some most useful [cmd commands](https://en.wikipedia.org/wiki/Cmd.exe) when using [R](https://cran.r-project.org/).  Please note, that the cmd commands apply only to windows environment, for Linux/MacOS, the system commands should be slightly changed, but the wrapper R code should remains the same.



**1. R commands to access the CMD**

R commands that are used for accessing, running and doing changes through cmd are:

* system
* system2
* shell
* shell.exec

All invoke the OS command that is specified and followed. All of the stated functions are part of the base R, so no additional packages are needed.

Let’s say, we want to start with [DIR](https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/dir) command and return the results in RStudio.

## 1 Simple dir command with order switch

system("dir /o")

system2("dir /o")

shell("dir /o")

shell.exec("dir /o")

and you will see that shell command will output the results back to console pane in R Studio, where as the others will not give you the immediate desired output. The main difference is that shell function will run the desired command under the shell (following the [POSIX standard](https://en.wikipedia.org/wiki/POSIX) or [here](http://math.furman.edu/~dcs/courses/math47/R/library/fCalendar/html/022A-PosixStandard.html) you can find POSIX functions)  and the system command will invoke OS command.

In R Base package,a set of dedicated functions is also available to create, open, append and close file and URL connections, as well as opening and loading the compressed files. Just listing the functions:

* object connections: file, url
* compression and encoding: gzfile, bzfile, xzfile, unz, pipe
* connection functions: open,close,flush
* boolean functions: isOpen, isIncomplete

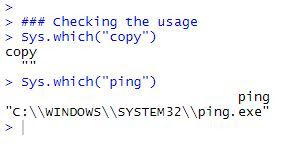
Rule of thumb (with default settings) on when to use system or shell function, shell will work always, system will only work when the system settings (paths) are correctly prepared. Example:

### Checking the usage

Sys.which("copy")

Sys.which("ping")

and the result is that system and system2 functions will work for ping CMD command, but not for copy CMD command, and later in the blogpost, also the rename, move command.



**2. Chaining the CMD commands in R**

For the simplicity, let’s continue with shell function and try how chaining can be achieved.

Following example will not work, as every time, the shell function is called, the environment is initiated. So running shell sequentially will not work!

setwd("C:\\Users\\Tomaz")

shell("dir")

shell("cd ..")

shell("dir")

Chaining the cmd commands together in one command will work:

shell("dir && cd .. && dir")

Note that this will not alter your working directory, so every time the shell is completed, you will be returned back to initiated environment. By checking:

getwd()

you can see, that the working directory has not changed. Checking before and after the R [environment variables](https://stat.ethz.ch/R-manual/R-devel/library/base/html/EnvVar.html), these will also remained intact:

Sys.getenv(c("R\_HOME","HOME"))

You can also define the variable, use the echo command, use the build in system variables like %TIME% and many more.

**3. Most typical CMD commands executed from R**

Here is the list of some CMD commands we will encapsulate in R:

* fc / compare
* rename
* move / copy
* ping (pingpath)
* systeminfo
* tasklist (taskkill)
* ipconfig / netstat

**3.1. Comparing two files in CMD**

Having capability to compare two files in CMD is just another of many ways to check if your dataset is the same.

Let’s create two \*.csv files with iris dataset and run the file compare in cmd:

setwd("C:\\Users\\Tomaz")

#create two files

write.csv(iris, file="iris\_file1.csv")

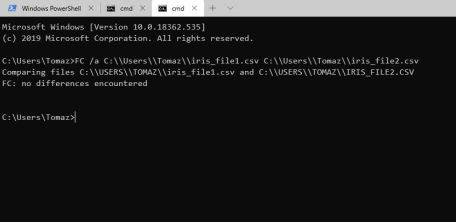
write.csv(iris, file="iris\_file2.csv")

#compare both files

shell("FC /a C:\\Users\\Tomaz\\iris\_file1.csv

C:\\Users\\Tomaz\\iris\_file2.csv")

The result information is: *FC: no differences encountered*. Same as if the command would be run from command line:



Same result could be achieved with R function that comes in base build: [all.equal](https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/all.equal?tap_a=5644-dce66f&tap_s=10907-287229).

file1 <- "C:\\Users\\Tomaz\\iris\_file1.csv"

file2 <- "C:\\Users\\Tomaz\\iris\_file2.csv"

# or in R using all.equal

all.equal(readLines(file1), readLines(file2))

Since we are in R, we would like to have results stored in data.frame. To do so, let’s use the previously mentioned System2 function. This is a newer version of System function that uses arguments and commands separately and some additional settings.

# run the same command using newer function "system2" and set the arguments

cmd\_command <- "FC"

#cmd\_args <- "/a C:\\Users\\Tomaz\\iris\_file1.csv

C:\\Users\\Tomaz\\iris\_file2.csv"

cmd\_args <- c('/a', file1, file2)

rr <- system2(command=cmd\_command,

args= cmd\_args,

stdout=TRUE,

stderr=TRUE,

wait = TRUE)

#suppose we want to store the results in data.frame

#empty dataframe

df\_rr <- data.frame(file1 = character(),

file2 = character(),

fcompare = character(),

stringsAsFactors=FALSE)

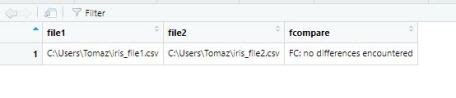
#temporary results

temp <- data.frame(file1=file1, file2=file2, fcompare=rr[2])

#bind all into dataframe

df\_rr <- rbind(df\_rr, setNames(temp, names(df\_rr)))

with results stored in data.frame when comparing multiple files.



**3.2. Renaming a file in CMD**

Keeping the file with iris dataset: *C:\\Users\\Tomaz\\iris\_file1.csv,* let’s do the renaming.

# renaming the file using shell command

shell("cd c:\\users\\tomaz && ren iris\_file1.csv iris\_file1\_copy.csv")

and with the parametrized shell function (remember the copy will not work due to *Sys.which(“copy”)* is returned empty):

file\_old\_name <- "c:\\users\\Tomaz\\iris\_file2.csv"

file\_new\_name <- "iris\_file2\_new.csv"

cmd\_command <- paste("RENAME", file\_old\_name, file\_new\_name)

# System2 Does not work

# system2(command=cmd\_command)

shell(cmd\_command)

In both ways, the files are renamed. Fortunately or not, the output in this case is none, when there is no syntax error and file exists.

**3.3. Copy a file in CMD**

Keeping the file with iris dataset: *C:\\Users\\Tomaz\\iris\_file1.csv,* let’s do also the copying.

#Copying file

shell("copy c:\\Users\\Tomaz\\iris\_file1.csv iris\_file1\_copy.csv")

and with the parametrized shell function:

orig\_file <- "c:\\Users\\Tomaz\\iris\_file1.csv"

copy\_file <- "iris\_file1\_copy1.csv"

command <- "copy"

cmd\_command <- paste(command, orig\_file, copy\_file)

shell(cmd\_command)

Now check your files.

**3.4. Ping**

Since ping CMD is available with the sys.which() function, we can use it directly as:

URL <- '[wordpress.com](http://tomaztsql.wordpress.com)'

cmd\_command <- "Ping"

cmd\_args <- c('-n 1', URL)

system2(command=cmd\_command,

args=cmd\_args,

stdout=TRUE,

stderr=TRUE,

wait = TRUE)

And since we want to check many URLs, let’s create an empty data.frame to store the result and a function to walk through the list.

URLs <- c("[google.com](http://google.com)", "[youtube.com](http://youtube.com)")

#empty dataframe

df\_rr <- data.frame(URL = character(),

reply = character(),

package = character(),

stringsAsFactors=FALSE)

ping\_fn <- function(url) {

system2("ping",c(url,' -n 1'),

stdout=TRUE,

stderr=TRUE)

}

for (i in 1:length(URLs)){

site <- print(URLs[i])

rr <- ping\_fn(site)

temp <- data.frame(URL=rr[2], reply=rr[3], package=rr[6])

df\_rr <- rbind(df\_rr, setNames(temp, names(df\_rr)))

}

head(df\_rr)

and the result is binded in a single data.frame:

**3.5. Systeminfo**

Assume, you need the information about the system not only about the R environment. Some of the R functions are available here:

# R

R.Version()

Sys.info()

.Platform

And all are very useful. But if you need Systeminfo CMD information into your R IDE environment, save it in data.frame, using system2() function is a way to go.

# Using system2

cmd\_command <- "systeminfo"

rr <- system2(command=cmd\_command,

stdout=TRUE,

stderr=TRUE,

wait = TRUE)

and getting the result:



So grabbing the e.g.: “System Manufacturer” just walk to the 13th position in the vector:

rr[13]

**3.6. Tasklist**

Getting the list of all the running task

cmd\_command <- "tasklist"

rr <- system2(command=cmd\_command,

stdout=TRUE,

stderr=TRUE,

wait = TRUE)

# getting the results into something more readable format

dd <- tibble::enframe(rr)

stack(dd)[1]

and can get all the additional information from the data.frame.

In addition, you can also kill a specific task

# kill a specific task

shell("taskkill /F /PID 6816")

with the returned information: “*SUCCESS: Sent termination signal to the process with PID 6816.*“

**3.7 Ipconfig / netstat**

Getting the list IpConfig information or all the netstat information, the task is the same:

cmd\_command <- "netstat"

rr <- system2(command=cmd\_command,

stdout=TRUE,

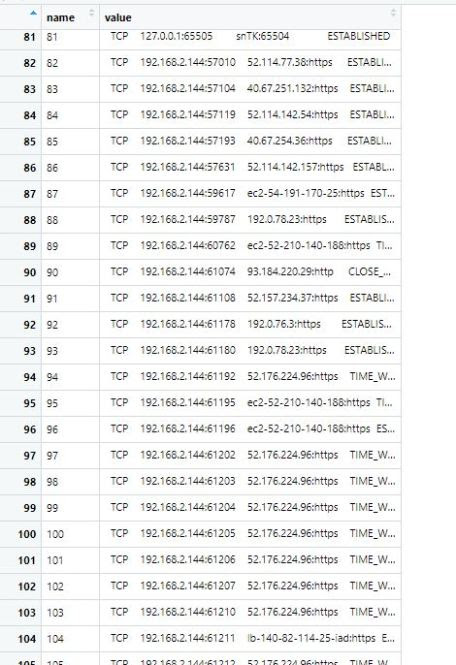
stderr=TRUE,

wait = TRUE)

and storing the results with tibble:

dd <- tibble::enframe(rr)

and the results are same as in CMD:



**4. Conclusion**

Using CMD in R IDE will for sure not be your daily driver when crunching numbers and wrangling data. But when you will need to operate closely with your files, datasets, system or tasks, system2 and shell functions are absolute winners to do the job.