911 Help for R Newbies

Neil J. Gunther

January 9, 2012

Contents

1 Overview

When I first tried to use R, I found it to be quite a struggle to remember how to do some of the most mundane things, like importing data in the R enviroment. There is an abundance of documentation for R (both local and on the web), as well as various online user fora. These resources, however, tend to assume that you already have a good deal of R context in your head and are otherwise very terse. This creates a significant *barrier to entry* for newbies trying to learn R.

To make matter worse, some commands are close to those found in the Unix shell[1], yet different enough to be confusing, e.g., `ls` and `grep` commands. See Sects. 6.3 and 15. Morover, like Unix man pages, the R documentation is only as useful as your ability to specifically recall the correct R command or function name.

[1]Not too surprising given that R was originally developed as S at AT&T Bell Labs

This RConsole activated resource is intended to help the R-newbie overcome those hurdles.

1.1 Rationale

After some consideration, I constructed the 911 helper in this particular way for several reasons:

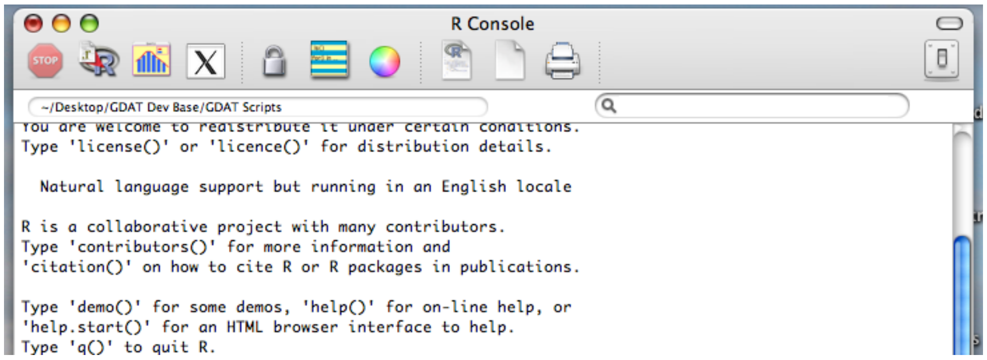
- I wanted it to be accessible as a Console command, rather than forcing you to rummage around looking for the PDF file.
- I did not want to replicate the way `help.start()` works by constructing separate documents in the `/tmp` directory.
- I did not want to go to all the trouble of creating an integrated `.Rd` file when there is no accompanying R package. Also, it does not facilitate the use of images.
- I did not want to assume that you were connected to the Internet.
- I wanted to be able to use hyperlinks to navigate quickly to the answer.
- I did not want to generate a local HTML file. Since I use LaTeX2e to generate HTML anyway, I may as well create a fully self-contained PDF and be done with it. Also, adding images makes HTML messy and less portable.

The combination of a hyperlinked PDF file and an associated R script to tell the Console where to find it, seemed to be the most expeditious way to go.

1.2 Installation

Acrobat Reader is assumed to be installed. To activate this 911 helper:

1. Locate the `911.r` and `911r.pdf` files in the *GDAT Scripts* directory/folder.
2. Move them to your preferred location.
3. Open 911.r and edit the `path911` variable appropriately. The string of characters containing the path (including slashes) must be contained within the double quotes. Check Section 1.3 for your O/S requirements.
4. From the R Console load `911.r` and follow the instructions there. Loading `911.r` can be accomplished in either of 2 ways:
   (a) As a command at the R Console prompt type: `source("~/Your Path/911.r")`. In this case, you must again supply the explicit path to your 911.r file.
   (b) From the R Console GUI using the R button (see Fig. 1). In this case, the path to your 911.r file will be determined by tracing it visually.



```
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
```

Figure 1: R Console in Mac OS X showing the set of function buttons along the top row with the R source button positioned second from the left next to the STOP sign

Of course, if you think 911 is too chauvenistic[2], you are free to rename the files to whatever you would like.

1.3 Possible Gotchas

Even though this is supposed to be the 21st century, we still have to deal with annoying discrepancies in some basic O/S commands. Hence, depending on how things are set up on your platform, you may run into to a failure mode for the 911.r script. In attempt to prevent any problems of this type, 911.r detects the type of O/S, assuming it's one of these three:

Linux: Most likely command is `gnome-open`, e.g., under ubuntu.

Mac OS X: The command is `open`. The explicit full path name is required in the `911` variable.

Window: Most likely command is `open`, e.g. Windows 7. The explicit full path name is required in the `911` variable, e.g., `C:\\your path\\`. Notice that you need `\\` to escape the first \ and another to escape the '9' in 991.r

If a failure still occurs in spite of 911.r attempting to accommodate O/S vagaries, a good strategy for resolving te problem is to check the correct form of the command in the shell or windows commander and edit 911.r accordingly.

[2]Emergency telephone numbers differ around the world so, you might prefer to use your local number. On the other hand, since September 11, 2011 in New York City, "911" has become more universally recognizable.

2 General Help

2.1 help.start()

The most general help documentation can be obtained by issuing the following command in the R Console:

```
> help.start()
```

which will launch a local HTML document in your browser.

2.2 help.911()

Launches this guide.

3 Where Am I?

Although the R environment is relatively platform-independent, sometimes you'd like to know what you're running and what it's running on.

3.1 What version of R?

```
> version

platform       powerpc-apple-darwin8.11.1
arch           powerpc
os             darwin8.11.1
system         powerpc, darwin8.11.1
...
version.string R version 2.8.1 (2008-12-22)
```

3.2 What hardware is this?

The Mac OS X command is:

```
> system("system_profiler SPHardwareDataType")
Hardware:

    Hardware Overview:

      Machine Name: Power Mac G4
      Machine Model: PowerMac3,6
      CPU Type: PowerPC G4  (3.2)
      Number Of CPUs: 1
      CPU Speed: 1 GHz
      ...
```

4 What Time is It?

Timestamps are critical to the discipline of doing any kind of analysis. The data and time are most easily accessed using the command:

```
> date()
[1] "Sun Dec 18 11:36:43 2011"
```

In a similar vein, it is recommended that you include both creation and update timestamps as comments in your R scripts. This can be done simply via copy/pasting the output of `date` into your R script.

```
# Created by NJG on Fri Jul 22 11:06:44 2011
# Updated by NJG on Mon Dec 19 10:51:43 2011
```

4.1 UNIX time

4.2 Time zones

5 Clear the Console

This is most easily done using the menu item:

Menu: Edit → Clear Console

Alternatively, these keyboard commands can be used:

PCs: Control-L

Mac: Clover-Option-L

6 Where is My File?

6.1 Which directory?

```
> getwd()
# assign the current dir to a variable name
# in case you want to restore it later
curd <- getwd()
```

6.2 But I wanna be here!

```
> setwd("~/You/Someplace/Else/")
```

6.3 Where's that file?

To see what files are in the current directory:

```
> dir()
```

Not to be confused (although it will be) with `ls()`:

```
> ls()
 [1] "f.x"      "fit0"     "fit1"     "fit2"     "help.911"  ...
 ...
```

which shows R objects that are currently active in memory.

7 Get My Data Now!

7.1 External data file

Large amounts of data are best read into R from a file:

```
bigData <- read.table("~/You/../file.dat", header=FALSE)
```

Or, if your data is exported from Excel:

```
csvData <- read.csv("~/You/../file.csv", header=TRUE)
```

7.2 Embed your data

One problem with reading external data files is that you have to make sure that the correct file is accessible. If the amount of data is not very big, it may be better to incorporate it directly into your R script using the following trick:

```
localData <- read.table(textConnection("ColA ColB
2 10
4 25
6 32
8 48
10 57"),
header=TRUE,sep="\t")
closeAllConnections()
```

8 Plot My Data

If your data frame is called "localData" then you can make a simple xy-plot:

```
> plot(localData$ColA, localData$ColB,type="b")
```

Issue `?plot` for more details from the base R package.

To add more data points or curves to an existing plot window, use:

```
> points(dataA)
> lines(dataB)
```

9 What Does This Function Do?

To get help concerning a particular function, type the following in the R Console:

```
> ?function
```

To find out how a particular function does what it does, just type the function name without the the prepended question-mark and without any appended parens:

```
> function
```

If the function is implemented in the R language, you will be able to read the source. For example, this should look a bit familiar:

```
> help.911
function() {
setwd(path911)
system("open 911r.pdf")
}
```

Otherwise, it will display something like:

```
> mean
function (x, ...)
UseMethod("mean")
<environment: namespace:base>
```

when the implementation is in some other compiled language.

10 Stop My Script!

To abort a running script in R, use the `stop()` function.

```
> ost <- "unknown"
> if(ost == "unknown") { stop("Unknown value of OST") }
Error: Unidentified operating system
```

Notice how it also prepends `Error:` to the output before aborting script execution.

11 Use library or require?

Which is correct: `library(pdq)` or `require(pdq)`? In practice, there is not much functional difference. If the package does not exist:

library: Throws an error and STOPS.

require: Returns FALSE and keeps going.

12 Assignment Operators

There are 3 types of assignment operator in R:

1. `<-`
2. `=`
3. `<<-`

The left-arrow construction `<-` is the most ubiquitous and can be used under any circumstances, The other assignment operator is `=` and is most commonly used in pre-defined functions for assigning a value to an argument in that function, e.g., `plot(x, type = "p", main = "My Plot")`

Historically, the left-arrow notation derives from the existence of a single key $\leftarrow$ on AT&T computers and APL keyboards when the S language was defined. Since no such character is available in ASCII, it has to be typed as 2 characters. Hence, `x <- 2` should be read as: "x gets two" in R parlance.

12.1 Quick keys for $\leftarrow$

The argument that it is easier to type to the single key `=` instead of `<-` is bogus because the following function-key combinations can be used:

MacOS X: Type the key combination: Option and –

Linux: Type the key combination: Alt and –

Windows: Type the key combination: Alt and –

You can also do an assignment as `2 -> X` but this is not advisable for either safe programming or code readability.

12.2 Double Arrow

Essentially `<<-` is used to update a global variable from within a local scope, such as a function.

```
> s <- 19
> t <- "foo"
> funny <- function(){ s <- 111; t <<- "bar"; }
> funny()
> c(s,t)
[1] "19"  "bar"
```

Notice that the variable `t` got updated, whereas variable `s` did not.

13 x++ in R?

Short answer: No! But you can get close by using the operators package on CRAN.

```
> require(operators)
> (x <- 20)
[1] 20
> (x %+=% 1)
[1] 21
```

which is identical to `x += 1` in C syntax and gives the same result as `x++`. More generally:

```
> (x %+=% 10)
[1] 31
```

If you examine the accompanying Reference manual, you'll see a very broad class of operators that use this same syntax.

Remark 1 (Caution) *The above operators should not be confused with the similar syntax used for modulus and integer division, which are part of the R base.*

```
> 5%%2   # mod
[1] 1
> 5%/%2  # div
[1] 2
```

14 a ? b : c in R?

The C language has the useful ternary operator which can be used in a statement like:

```
y = x == 1 ? 2 : 3;
```

so that if x is TRUE (1) then y = 2, otherwise y = 3. The R syntax for ternary operation uses the `ifelse()` function:

```
> x <- FALSE
> y <- ifelse(x == TRUE, 2, 3)
> y
[1] 3
```

15 String Manipulation

String manipulation and regexp examples. For more details type `?regex` in the RConsole. (see Section 9)

15.1 String variable

Like Perl, a string-valued variable (or object) is defined by characters within double quotes.

```
> s <- "This is a string of characters"
> is.character(s)
[1] TRUE
> str(s)
 chr "This is a string of characters"
> length(s)
[1] 1
> nchar(s) # Count characters
[1] 30
```

There is no limit to the size of a string; any amount of characters, symbols, or words can make up your strings.

15.2 String split

The function `strsplit` in R acts like `split` in Perl.

```
> strsplit(s," ")
[[1]]
[1] "This"       "is"         "a"          "string"     "of"         "characters"
> ss<-strsplit(s," ")
> is.character(ss)
[1] FALSE
```

```
> str(ss) # it's a list object
List of 1
 $ : chr [1:6] "This" "is" "a" "string" ...
> length(ss)
[1] 1
> ssu<-unlist(ss)
> ssu
[1] "This"       "is"         "a"          "string"     "of"         "characters"
> is.character(ssu)
[1] TRUE
> str(ssu) # it's a string object again
 chr [1:6] "This" "is" "a" "string" "of" "characters"
> length(ssu) # Count words
[1] 6
```

TBC ...

16 Packages

16.1 Installing more packages

To install a new R package called "Pkg" as an R Console command:

```
> install.packages("Pkg")
```

Note the use of quotes in the argument.

I prefer to use the GUI Package manager 2a and Package installer 2b invoked from the R menu bar.

16.2 Package dependencies

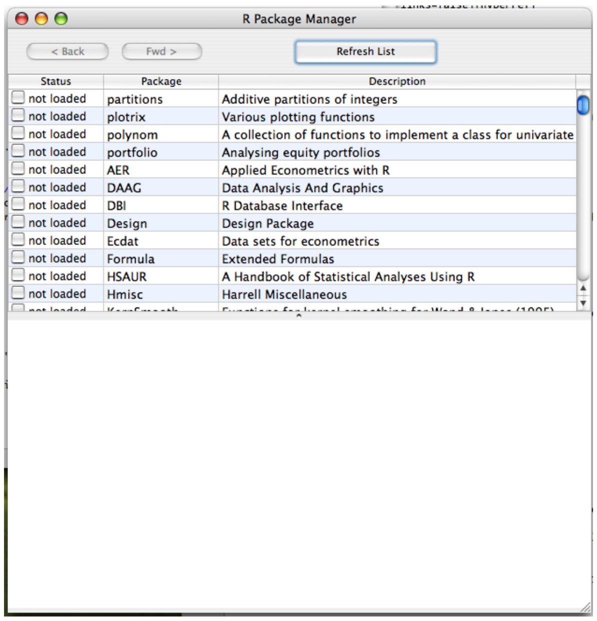
How do I distinguish function `foo` when it appears with the same name in two different R packages?

Use path dependent calls, which uses the same syntax as Perl:

```
pkgA::foo()
pkgB::foo()
```

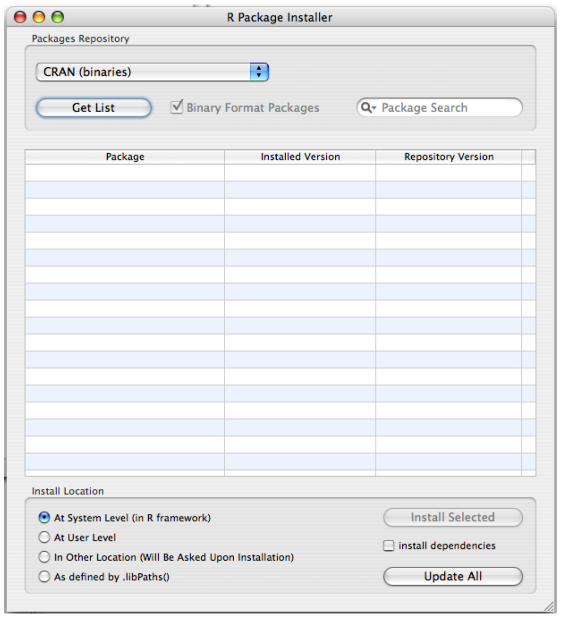
It's also a good idea to use explicit package names when employing PDQ:

```
library(pdq)
pdq::CreateNode()
pdq::SetDemand()
```



Status	Package	Description
not loaded	partitions	Additive partitions of integers
not loaded	plotrix	Various plotting functions
not loaded	polynom	A collection of functions to implement a class for univariate
not loaded	portfolio	Analysing equity portfolios
not loaded	AER	Applied Econometrics with R
not loaded	DAAG	Data Analysis And Graphics
not loaded	DBI	R Database Interface
not loaded	Design	Design Package
not loaded	Ecdat	Data sets for econometrics
not loaded	Formula	Extended Formulas
not loaded	HSAUR	A Handbook of Statistical Analyses Using R
not loaded	Hmisc	Harrell Miscellaneous

(a) Package manager



Packages Repository: CRAN (binaries)

[✓] Binary Format Packages

Package	Installed Version	Repository Version

Install Location

[✓ At System Level (in R framework)]
At User Level
In Other Location (Will Be Asked Upon Installation)
As defined by .libPaths()

[ ] install dependencies

(b) Package installer

Figure 2: Package management GUIs in Mac OS X

17 PDQ in R

Currently, PDQ is not part of CRAN and must be installed separately from PerfDynamics.com or SourceForge.

Since PDQ is a library of queueing analysis functions, it is invoked in R via:

```
library(pdq)
```

17.1 PDQ Scripts

Because it takes several PDQ functions to establish even the simplest model, it is recommended that you construct your PDQ models as R scripts, rather than entering each PDQ function as a separate R Console command.

17.2 Open Model

An open queueing model is one where there is a potentially unbounded flow of requests into a service facility.

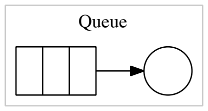
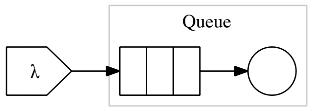
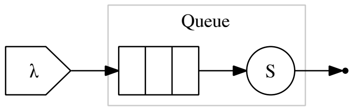
Unbounded means that the rate of flow into the server could be greater than the service rate ($\lambda > \mu$), in which case the queue would become infinitely long and the queueing model cannot be solved. PDQ will report an error condition in this case and cease computation until it is fixed.

The standard solution technique for an open model is called "canonical" or CANON in PDQ parlance.

Example 1 *Checkout stand*

```
library(pdq)
arate <- 0.5   # lambda
stime <- 1.0   # service time S
pdq::Init("Open Model")
pdq::CreateNode("checkout", CEN, FCFS)
pdq::CreateOpen("traffic", arate)
pdq::SetDemand("checkout", "traffic", stime)
pdq::Solve(CANON)
pdq::Report()
```

has the following semantics in PDQ:

Action	Example	Function	Diagramatically
Initialize PDQ		*pdq::Init*	
Define queue	*Checkout stand*	*pdq::CreateNode*	
Define workload	*Store traffic*	*pdq::CreateOpen*	
Define service time	*Purchase time*	*pdq::SetDemand*	
Solve the model		*pdq::Solve*	
Show results		*pdq::Report*	

17.3 Closed Model

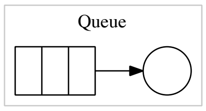
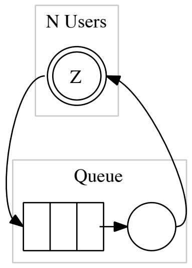
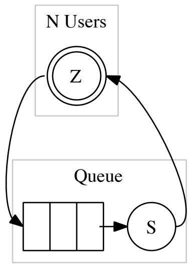
A closed queueing model is one where only a finite number of requests ($N \ll \infty$) are allowed to flow in the queueing system; no requests can exit or enter from outside. Therefore, the queueing system can never be Unbounded in the sense of an open model.

Example 2 *Load test system*

```
library(pdq)
clients <- 100   # N finite users
thinkz  <- 30    # thinks time Z
stime   <- 1.0   # service time S
```

```
pdq::Init("Closed Model")
pdq::CreateNode("SUT", CEN, FCFS)
pdq::CreateClosed("script", TERM, clients, thinkz)
pdq::SetDemand("SUT", "script", stime)
pdq::Solve(EXACT)
pdq::Report()
```

has the following semantics in PDQ:

Action	**Example**	**Function**	**Diagramatically**
Initialize PDQ		*pdq::Init*	
Define queue	*System under test*	*pdq::CreateNode*	
Define workload	*Client scripts*	*pdq::CreateClosed*	
Define service time	*Scripted action*	*pdq::SetDemand*	
Solve the model		*pdq::Solve*	
Show results		*pdq::Report*	

There two solution techniques for a closed model:

1. When $N < 100$, use `EXACT` method. This algorithm consumes memory.
2. When $N > 100$, use `APPROX` method. Consumes far less memory.

17.4 Tandem Queues

TBD

17.5 Parallel Queues

TBD

17.6 Approximating Closed with Open Models

TBD

18 IDEs for R

There are a number of so-called integrated development environments for R:

- RStudio FOSS
- Revolution commercial