ILLINOIS

Rscript and r

Lecture 22

Dirk Eddelbuettel

STAT 430: Data Science Programming Methods (Fall 2019)

Department of Statistics, University of Illinois

Scripting with R: Rscript

A scripting frontend

- Should be installed whereever R is available
- Can access command-line arguments (examples follow)
- Now makes (base R) package `methods` available by default
  - Needed for S4 packages
  - Behavior should be indistinguishable from R
- Useful for scripting

A few basic rules

- The first line defines the "shebang":
  - `#!/bin/bash` for shell scripts
  - `#!/usr/bin/env Rscript` uses indirection trick
- "Mode" *i.e.* permissions have to be set
- A good value is 0775 or 0755: read and execute bits for everyone

A first Rscript example: **render.r**

```
#!/usr/bin/env Rscript

argv <- commandArgs(trailingOnly = TRUE)

if (length(argv) == 0) {
    cat("Usage: render.R file1 [file2 [file3 [...]]]\n")
} else {
    for (f in argv) {
        if (file.exists(f)) {
            rmarkdown::render(f)
        }
    }
}
```

A first Rscript example: `render.r`

```
#!/usr/bin/env Rscript

argv <- commandArgs(trailingOnly = TRUE)

if (length(argv) == 0) {
  cat("Usage: render.R file1 [file2 [file3 [...]]]\n")
} else {
  for (f in argv) {
    if (file.exists(f)) {
      rmarkdown::render(f)
    }
  }
}
```

Assign command-line arguments to `argv`

Checks if any arguments given

Loops over arguments, tests for file and renders from markdown

Discussion of first Rscript example: `render.r`

- `commandArgs()` needs `trailingOnly=TRUE` argument
- Else arguments given to underlying R process returned too
  - Usually not what one wants
- Otherwise standard R functionality available in a script
- Possible to use code from all installed packages at will
- Here we just call `rmarkdown::render()`, *i.e.*
  - `render()` function from `rmarkdown` package
  - without loading `rmarkdown` first

Scripting with R: R

Features

- `r` from the `littler` package can be installed from CRAN
- Not yet pre-installed on RStudio Cloud
- A simple `install.packages("littler")` works
- Then in the terminal (*not* in R):

```
mkdir ~/bin/
ln -s ~/R/x86_64-pc-linux-gnu-library/3.6/littler/bin/r ~/bin/
```

- This creates a soft-link for `r` in `~/bin` which is in `$PATH`
- Afer that `r` should be accessible for shell commands

A second r example: `render.r`

```
#!/usr/bin/env r

if (length(argv) == 0) {
  cat("Usage: render.R file1 [file2 [file3 [...]]]\n")
} else {
  for (f in argv) {
    if (file.exists(f)) {
      rmarkdown::render(f)
    }
  }
}
```

`argv` is automatically assigned

Again checks if any arguments given

Loops over arguments, tests for file and renders from markdown

stdin

- Both `Rscript` and `r` can read from `stdin`
- Starting with `echo -e "1\n2\n3"`
- We can use the same code to read and process:

```
x <- read.table(file="stdin"); print(summary(x[,1]))
```

- Process with either a pipe into `Rscript` or `r`
- In both cases `-e ' ... '` is used with expression between single quotes

Included with the packages

- `install.r` to install packages
- `install2.r` which more control and options
- `update.r` to update installed packages
- `roxy.r` to convert generate Rd documentation
- `render.r` generalizes the simple example we looked at
- `rcc.r` run `R CMD check` via the `rcmdcheck` package

More in the examples/ directory and the vignette

`docopt`

- one of several package to parse command-line options
- provides R implementation, many other languages supported
- main idea: use the `--help` output to define the option switches
- this sounds weird unless ... you ever had to do this by hand
- simple example next, many examples in `littler` package

Making Scripts Better: docopt

ILLINOIS

```
#!/usr/bin/env r
# A simple example to invoke unit tests [...]

suppressMessages({
    library(docopt)       # we need the docopt package
    library(RUnit)        # we need the RUnit package
})
doc <- "Usage: runit.r [-p PACKAGES] [--help] [FILES ...]

-p --packages PACKAGES      comma-separated list of packages to install [default: ]
-h --help                   show this help text"

opt <- docopt(doc)
for (p in strsplit(opt$packages, ",")[[1]])   # load packages, if any listed
    suppressMessages(require(p, character.only=TRUE))

for (f in opt$FILES)          # for all submitted file arguments
    runTestFile(f)            # run tests on given file

q(status=0)
```

docopt Examples

- Previous example was actual code (some whitespace removed)
- Support simple functionalities:
  - brief help on `runit.r -h` (or `--help`)
  - optionally loading listed packages (split by comma) too
  - run unit tests from a particular given file
- Argument to `doc` is a simple fixed string, it is displayed
- That string defines `-p` and `--packages` as well as help
- Also defines that trailing arguments go into `FILES`
- After `opt <- docopt(doc)` we just pick these off `opt`

cron: Automation

Automation

- One powerful things on a Unix computer is the `cron` daemon
- It permits execution at a given
  - time of day
  - hour/minute
  - weekday
  - ...
- Or a combination via (slightly technical) file called `crontab`

Automation

- In essence: if you write code that runs
- Then you can place the code into a script
- (For R code it also helps to organize it as a package)
- That script can then be executed automatically
- And that is all there is to automation

Illustration

- `crontab` uses minute, hour, day-of-month, month, day-of-week
- values that are a `*` are still the unspecified default
- also specified: user to run the the task as, script, arguments
- a real entry on my server

```
15 12   * * *    edd /home/edd/git/crp/bin/check.r
```

- this means every day (no day-of-month or week) at 15 minutes after noon the `check.r` script from the `crp` repo
- another sets log file rotation at 01:14 each Monday

```
14 01   * * 1    edd /home/edd/bin/rotate.files
```

"Data Science Programming Methods": Tie it all together

- We started with the shell, commands, shell scripting, ...
- Next came R and data manipulation and visualization tasks
- `Rscript` or `r` let you build new R-based commands
- Tools like `cron` allows you to automate running these
- (On Windows you have the Windows task scheduler too)
- Plus `git` for keeping track of your project code
- And Markdown for report generation ...

Summary of Lecture 22

Summary

- We learned about two scripting front-ends
- First `Rscript` which comes with R
- Second `r` which predated `Rscript` a little
- We looked into a simple example
- We also learned about `docopt` for command-line parsing
- A brief illustration of `cron` and `crontab` concluded