New R tools for workflow management, testing, and reproducibility

Will Landau

14 October 2016

Overview

Established tools

- State of the art
  - testthat (Wickham 2011)
  - knitr (Xie 2015)
  - GitHub (GitHub, Inc. 2016)
  - packrat (Ushey et al. 2016)
  - ProjectTemplate (White 2016)
- Underutilized
  - GNU Make (Stallman and McGrath 1991)

New tools

- **downsize** (Landau 2016a)
- **remake** (FitzJohn 2016)
- parallelRemake (Landau 2016b)
- **remakeGenerator** (Landau 2016c)

downsize

Test mode

- Use `test_mode()`.

```
library(downsize)
test_mode() # scales the workflow appropriately
my_mode() # shows workflow mode
```

```
## [1] "test mode"
```

```
big_data <- data.frame(x = rnorm(1e4), y = rnorm(1e4)) # always large
my_data <- downsize(big_data) # either large or small
nrow(my_data) # responds to test_mode() and production_mode()
```

```
## [1] 6
```

```
# ...more code, time-consuming if my_data is large...
```

Production mode

- Replace `test_mode()` with `production_mode()` and leave everything else the same.

```
library(downsize)
production_mode() # scales the workflow appropriately
my_mode() # shows workflow mode
```

```
## [1] "production mode"
```

```
big_data <- data.frame(x = rnorm(1e4), y = rnorm(1e4)) # always large
my_data <- downsize(big_data) # either large or small
nrow(my_data) # responds to test_mode() and production_mode()
```

```
## [1] 10000
```

```
# ...more code, time-consuming if my_data is large...
```

Select your own test data

```
small_data = data.frame(x = 1:2, y = 3:4)
test_mode()
downsize(big = big_data, small = small_data)
```

```
##   x y
## 1 1 3
## 2 2 4
```

```
production_mode()
my_data = downsize(big = big_data, small = small_data)
dim(my_data)
```

```
## [1] 10000     2
```

Toggle subsetting

```
test_mode()
downsize(big = big_data, nrow = 4) # could have set ncol
```

```
##            x          y
## 1 -0.5673685  0.9441748
## 2  2.0583166 -0.2589098
## 3 -1.2931820 -1.6728959
## 4 -0.5739584 -0.3526358
```

```
downsize(big = big_data, dim = c(4, 1))
```

```
##            x
## 1 -0.5673685
## 2  2.0583166
## 3 -1.2931820
## 4 -0.5739584
```

Toggle subsetting

```
downsize(big = 1:16, length = 4)
```

```
## [1] 1 2 3 4
```

```
set.seed(0)
downsize(big = 1:16, length = 4, random = TRUE)
```

```
## [1] 15  4  6  8
```

```
downsize(big = 1:16, length = 4, downsize = FALSE)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
```

Toggle entire code blocks

```
test_mode()
downsize(big = {a = 1; a + 10}, small = {a = 1; a + 1})
```

```
## [1] 2
```

```
production_mode()
downsize(big = {a = 1; a + 10}, small = {a = 1; a + 1})
```

```
## [1] 11
```

Toggle entire code blocks

```
test_mode()
```

```
tmp <- downsize(
  big = {
    x = "long code"
    y = 1000
  },
  small = {
    x = "short code"
    y = 3.14
  })
```

```
x == "short code" & y == 3.14
```

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

Toggle entire code blocks

```
production_mode()
```

```
tmp <- downsize(
  big = {
    x = "long code"
    y = 1000
  },
  small = {
    x = "short code"
    y = 3.14
  })
```

```
x == "long code" & y == 1000
```

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

remake

code.R (your custom functions)

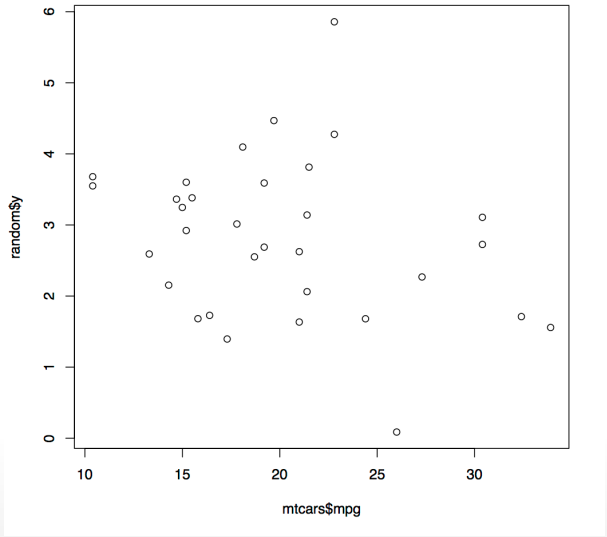
```
my_mtcars = function(){
  data(mtcars)
  mtcars
}

my_random = function(){
  data.frame(y = rnorm(32))
}

my_plot = function(mtcars, random){
  plot(mtcars$mpg, random$y)
}
```

remake.yml (the analysis plan)

```
sources: code.R
packages: MASS

targets:

  all:
    depends: plot.pdf

  plot.pdf:
    command: my_plot(mtcars, random)
    plot: TRUE

  mtcars:
    command: my_mtcars()

  random:
    command: my_random()
```

First execution

```
> library(remake)
> make()
<  MAKE > all
[ BUILD ] mtcars    |  mtcars <- my_mtcars()
[ BUILD ] random    |  random <- my_random()
[  PLOT ] plot.pdf  |  my_plot(mtcars, random) # ==> plot.pdf
[ ----- ] all
```



Repeated execution

After changing only whitespace and comments in `code.R`:

```
> make()
<  MAKE > all
[    OK ] mtcars
[    OK ] random
[    OK ] plot.pdf
[ ----- ] all
```

After changing code inside `my_random()`:

```
> make()
[  READ ]           |  # loading sources
<  MAKE > all
[    OK ] mtcars
[ BUILD ] random    |  random <- my_random()
[  PLOT ] plot.pdf  |  my_plot(mtcars, random) # ==> plot.pdf
[ ----- ] all
```

String literal arguments

```
targets:
  all:
    depends: x # x == list("y")
  x:
    command: list(I("y")) # Function I() says "y" is just a string.
```

```
targets:
  all:
    depends: x # x[[1]] stores the contents of z.csv.
  x:
    command: list("y") # Cached R object y is a dependency.
  y:
    command: read.csv("z.csv") # External file z.csv is a dependency.
```

Nested functions

- Except for `I()`, do not use nested functions in `remake.yml`.
- INCORRECT:

```
targets:
  all:
    depends: x
  x:
    command: f(g(1)) # ERROR
```

- CORRECT:

```
targets:
  all:
    depends: x
  x:
    command: h(1) # h <- function(y) f(g(y))
```

- Changing the body of `f()`, `g()`, or `h()` triggers the recomputation of `x` and downstream targets in the next call to `make()`.

remakeGenerator

Write the files for the example

```
library(remakeGenerator)
example_remakeGenerator(index = 1)
```

- `code.R`
- `workflow.R`
- `markdown.Rmd`
- `latex.Rnw`

code.R

```r
# Generate datasets
normal_dataset = function(n = 16){
  data.frame(x = rnorm(n, 1), y = rnorm(n, 5))
}

poisson_dataset = function(n = 16){
  data.frame(x = rpois(n, 1), y = rpois(n, 5))
}

# Analyze each dataset
linear_analysis = function(dataset){
  lm(y ~ x, data = dataset)
}

quadratic_analysis = function(dataset){
  lm(y ~ x + I(x^2), data = dataset)
}
```

code.R

```r
# Compute summaries
mse_summary = function(dataset, analysis){
  predictions = predict(analysis)
  mean((predictions - dataset$y)^2)
}

coefficients_summary = function(analysis){
  out = c(coefficients(analysis), "I(x^2)" = 0)[1:3]
}
```

workflow.R

```
datasets = commands(
  normal16 = normal_dataset(n = 16),
  poisson32 = poisson_dataset(n = 32),
  poisson64 = poisson_dataset(n = 64))
```

```
datasets
```

```
##      target                   command
## 1  normal16  normal_dataset(n = 16)
## 2 poisson32 poisson_dataset(n = 32)
## 3 poisson64 poisson_dataset(n = 64)
```

workflow.R

```
analyses = analyses(
  commands = commands(
    linear = linear_analysis(..dataset..),
    quadratic = quadratic_analysis(..dataset..)),
  datasets = datasets)
```

```
analyses
```

```
##                target                         command
## 1     linear_normal16     linear_analysis(normal16)
## 2    linear_poisson32    linear_analysis(poisson32)
## 3    linear_poisson64    linear_analysis(poisson64)
## 4  quadratic_normal16  quadratic_analysis(normal16)
## 5 quadratic_poisson32 quadratic_analysis(poisson32)
## 6 quadratic_poisson64 quadratic_analysis(poisson64)
```

workflow.R

```
summaries = summaries(
  commands = commands(
    mse = mse_summary(..dataset.., ..analysis..),
    coef = coefficients_summary(..analysis..)),
  analyses = analyses, datasets = datasets,
  gather = strings(c, rbind))
```

```
summaries[3:8,]
```

```
##                    target                                        command
## 3     mse_linear_normal16      mse_summary(normal16, linear_normal16)
## 4    mse_linear_poisson32    mse_summary(poisson32, linear_poisson32)
## 5    mse_linear_poisson64    mse_summary(poisson64, linear_poisson64)
## 6  mse_quadratic_normal16   mse_summary(normal16, quadratic_normal16)
## 7 mse_quadratic_poisson32 mse_summary(poisson32, quadratic_poisson32)
## 8 mse_quadratic_poisson64 mse_summary(poisson64, quadratic_poisson64)
```

workflow.R

```
summaries[1:2,]
```

```
##   target
## 1   coef
## 2    mse
##
## 1 rbind(coef_linear_normal16 = coef_linear_normal16, coef_linear_poisson32 = coef
## 2                 c(mse_linear_normal16 = mse_linear_normal16, mse_linear_poisson
```

workflow.R

```
output = commands(coef.csv = write.csv(coef, target_name))
```

```
output
```

```
##     target                          command
## 1 coef.csv write.csv(coef, target_name)
```

workflow.R

```
plots = commands(mse.pdf = hist(mse, col = I("black")))
plots$plot = TRUE
```

```
plots
```

```
##    target                       command plot
## 1 mse.pdf hist(mse, col = I("black")) TRUE
```

workflow.R

```
reports = data.frame(target = strings(markdown.md, latex.tex),
  depends = c("poisson32, coef, coef.csv", ""))
reports$knitr = TRUE
```

```
reports
```

```
##        target                    depends knitr
## 1 markdown.md poisson32, coef, coef.csv  TRUE
## 2   latex.tex                            TRUE
```

workflow.R

```
targets = targets(datasets = datasets, analyses = analyses,
  summaries = summaries, output = output, plots = plots,
  reports = reports)
```

```
str(targets)
```

```
## List of 34
##  $ all                      :List of 1
##   ..$ depends:List of 6
##   .. ..$ : chr "datasets"
##   .. ..$ : chr "analyses"
##   .. ..$ : chr "summaries"
##   .. ..$ : chr "output"
##   .. ..$ : chr "plots"
##   .. ..$ : chr "reports"
##  $ datasets                 :List of 1
##   ..$ depends:List of 3
##   .. ..$ : chr "normal16"
##   .. ..$ : chr "poisson32"
```

workflow.R

Write `remake.yml` and `Makefile`.

```
workflow(targets, sources = "code.R", packages = "MASS",
  begin = c("# Prepend this", "# to the Makefile."))
```

`workflow()` uses `parallelRemake::write_makefile()`.

Distribute targets over HPC jobs/nodes

- For clusters with the SLURM job scheduler (SchedMD 2016),

```
workflow(...,
  begin = c(
    "SHELL=srun",
    ".SHELLFLAGS= <ARGS> bash -c"))
```

  where `<ARGS>` = additional arguments to `srun`.

- For details, see Lindenbaum (2014).

Execution

Execute directly with `remake`

```
remake::make()
```

or access `remake` through the `Makefile` for parallel computing.

```
system("make -j 4")
```

Inspect output in an R session

```
parallelRemake::recallable() # List available cached targets
x <- parallelRemake::recall("x") # load target `x` into session
```

Also see

```
remake::dump_environment()
remake::create_bindings()
remake::delete_bindings()
```

Extend remakeGenerator

Extend remakeGenerator

- Use `expand()` to generate multiple replicates of each target.

```
datasets = commands(
  normal16 = normal_dataset(n = 16),
  poisson32 = poisson_dataset(n = 32),
  poisson64 = poisson_dataset(n = 64))
```

```
datasets
```

```
##      target                   command
## 1  normal16  normal_dataset(n = 16)
## 2 poisson32 poisson_dataset(n = 32)
## 3 poisson64 poisson_dataset(n = 64)
```

Extend remakeGenerator

- Use `expand()` to generate multiple replicates of each target.

```
expand(datasets, values = c("rep1", "rep2"))
```

```
##           target                  command
## 1  normal16_rep1  normal_dataset(n = 16)
## 2  normal16_rep2  normal_dataset(n = 16)
## 3 poisson32_rep1 poisson_dataset(n = 32)
## 4 poisson32_rep2 poisson_dataset(n = 32)
## 5 poisson64_rep1 poisson_dataset(n = 64)
## 6 poisson64_rep2 poisson_dataset(n = 64)
```

Extend remakeGenerator

- Symbols `..dataset..` and `..analysis..` are built-in wildcard placeholders.
- You can define your own (for example, `MY_DATA`).

```
analyses = commands(
  linear = linear_analysis(MY_DATA),
  quadratic = quadratic_analysis(MY_DATA))
```

```
analyses
```

```
##      target                        command
## 1    linear    linear_analysis(MY_DATA)
## 2 quadratic quadratic_analysis(MY_DATA)
```

Extend remakeGenerator

- Be sure to `evaluate()` your wildcards.

```
evaluate(analyses, wildcard = "MY_DATA", values = c("data1", "data2"))
```

```
##             target                      command
## 1    linear_data1    linear_analysis(data1)
## 2    linear_data2    linear_analysis(data2)
## 3 quadratic_data1 quadratic_analysis(data1)
## 4 quadratic_data2 quadratic_analysis(data2)
```

- Full demo:

```
example_remakeGenerator(index = 2)
```

Obtain the packages

```
install.packages("downsize")
library(devtools)
install_github("richfitz/remake")
install_github("wlandau/parallelRemake")
install_github("wlandau/remakeGenerator")
```

- Using `system("make")` for Makefiles on Windows requires Rtools.
- `parallelRemake` and `remakeGenerator` will be submitted to CRAN after `remake` is released.

Read more

- GNU Make: gnu.org/software/make/
- remake: github.com/richfitz/remake
- parallelRemake: github.com/wlandau/parallelRemake
- remakeGenerator: github.com/wlandau/remakeGenerator
- downsize: github.com/wlandau/downsize
- blog post: will-landau.com/2016/06/14/workflow/
- See links:

```
downsize::help_downsize()
parallelRemake::help_parallelRemake()
remakeGenerator::help_remakeGenerator()
```

References

FitzJohn, Rich. 2016. “Remake: Make-Like Declarative Workflows in R.” https://github.com/richfitz/remake.

GitHub, Inc. 2016. “GitHub.” https://github.com.

Landau, Will. 2016a. . https://github.com/wlandau/downsize.

———. 2016b. “ParallelRemake: An R Package to Accelerate Remake Workflows.” https://github.com/wlandau/parallelRemake.

———. 2016c. “RemakeGenerator: Generate Large Remake-Style Workflows with Minimal Code.” https://github.com/wlandau/remakeGenerator.

Lindenbaum, Pierre. 2014. “Parallelizing GNU #Make 4 in a #SLURM Infrastructure/Cluster.” http://plindenbaum.blogspot.com/2014/09/parallelizing-gnu-make-4-in-slurm.html.

SchedMD. 2016. “SLURM Workload Manager Version 16.05.” http://slurm.schedmd.com/.

Stallman, Richard, and Roland McGrath. 1991. . Boston: Free Software Foundation.

Ushey, Kevin, Jonathan McPherson, Joe Cheng, Aron Atkins, and JJ Allaire. 2016. . https://CRAN.R-project.org/package=packrat.

White, John Myles. 2016. . https://CRAN.R-project.org/package=ProjectTemplate.

Wickham, Hadley. 2011. “Testthat: Get Started with Testing.” 3: 5–10. http://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf.

Xie, Yihui. 2015. . 2nd ed. Boca Raton, Florida: Chapman; Hall/CRC. http://yihui.name/knitr/.