R gnumaker: easy Makefile construction for enhancing reproducible research

Peter Baker
School of Public Health
< p.baker1@uq.edu.au >

10 July 2019

The University Of Queensland
AUSTRALIA

Section 1

Background

My Background

- Many years as a statistical consultant
  - for NSW Agriculture, CSIRO, UQ Public Health
  - to agricultural, genetics, medical and epidemiological researchers
- Statistical software
  - GENSTAT, Minitab, SAS, SPSS, STATA, S, BUGS, JAGS, ...
  - R (almost) exclusively since 1998
- Other software for managing data analysis/reporting
  - Make & Version Control (cvs, svn, git)
  - GNU Make & Version Control since early 1990s
  - literate programming: Sweave, Knitr, R Markdown, ...

Real world consulting

Are these scenarios familiar?

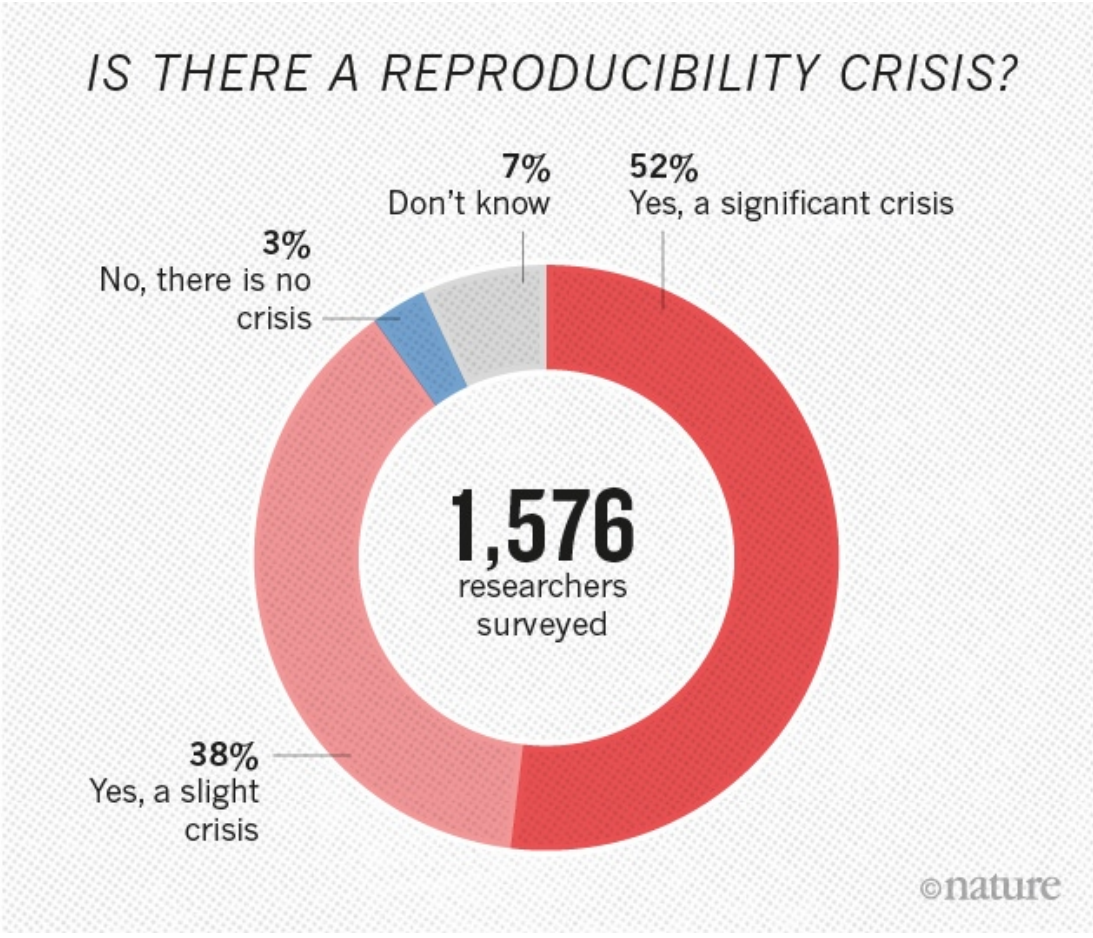
- I have a very simple question that will only take 5 minutes. I won't need to see you again
- We have several data points that need deleting. Can you rerun the analysis, insert the new tables and plot into our report by 4pm today?
- The journal got back to us: Can you rerun the analysis to take account critisicms of our method? Its not the project we did last year but the one in 2014

Real world consulting

Somtimes reproducibility is hard;

- No matter what clients/funders/bosses say, what happens is often very different
- **All** these situations need to be well organised and well documented
- Standardised systems help
- Good computing tools help this process too

Reproducibility



1,500 scientists lift
the lid on reproducibility *Nature*

Source: Monya Baker (2016)

A *DRY* creek near home

DRY versus WET workflows

- *DRY:*
  - Don't Repeat Yourself

DRY versus WET workflows

- *DRY:*
  - Don't Repeat Yourself
- *WET:*
  - Write Everything Twice
  - We Enjoy Typing
  - Waste Everyone's Time
- Copy-cut-and-paste writing/reporting is *WET*

Workflow of data analysis cycle

1. Plan
2. Document
3. Organise
4. Carry out analysis
5. Communicate results
6. Iterate through steps 1 to 5 and refine process

Long provides a good overview for *Stata* (Long 2009)

Workflow of data analysis cycle

1. Plan
2. Document
3. Organise
4. Carry out analysis
5. Communicate results
6. Iterate through steps 1 to 5 and refine process

Long provides a good overview for *Stata* (Long 2009)

- We can use GNU Make for Steps 3-6

Complex project directory structure

```
complex_demo/myRproject
├── admin
├── analysis
│   └── Makefile
├── data
│   ├── codebook
│   ├── derived
│   └── original
├── doc
│   ├── original
│   └── references
├── lib
├── readCleanData
│   └── Makefile
├── reports
│   └── Makefile
└── Makefile
```

Section 2

Why GNU Make?

Make and reproducible research

> *I would argue that the most important tool for reproducible research is not Sweave or knitr but GNU Make.*
> *Karl Broman*
> *Source: https://kbroman.org/minimal_make/*

Many talks tout R Markdown as being the basis of reproducible research but statisticians "don't just write simple reports. . . "

I argue that the three most useful tools we can use to aid the data analysis workflow and facilitate reproducible research are

1. GNU Make
2. Git
3. R Markdown (and R)

(or alternatives)

Why GNU Make?

Rerunning analysis

Choices:

- Manually
  - need to document steps heavily
  - still may forget something
- **GNU Make**
  - automates
  - only rerun steps needed
  - keeps track of the process
    - but need to read *make*

We can use alternatives to GNU Make but how reproducible (stable) are they?

Why GNU Make?

GNU Make

- is **defacto standard**
- aids **reproducibility**:
  - GNU Make (1976) changes at glacial pace (should work OK in 5-10 years?)
  - use GNU Make to (re)run anything you can run from command line
  - modular operation: break down into smaller tasks to facilitate reproducible research (reporting)
  - we specify what depends on what and then *make* only updates necessary files
- **documents** workflow
- works well in tandem with `git`

NB: can **automate** *make* in RStudio/ESS/IDE

Section 3

How to use GNU Make

Targets and dependencies

`Makefiles` specify target files and dependency files:

```
target_file: dependency_file_1 dependency_file_2 ...
<TAB> command 1
<TAB> command 2
<TAB> command 3
```

- make compares the times that files were saved
- if dependencies are 'newer' than targets then commands are run

Note that command lines begin with a tab not spaces

WWW: Be careful if cutting and pasting from webpages:
TABS become SPACES

Targets and dependencies

Here is a simple `Makefile` that we might use just to read the data:

```
read.Rout: read.R bmi2009.dta
<TAB> R CMD BATCH read.R
```

- make compares the times that files were saved
- if dependencies are 'newer' than targets then R BATCH command is run
- `read.Rout` is **target** on LHS :
- `read.R` and `bmi2009.dta` are **dependencies**

Running make

If either *read.R* or *bmi2009.dta* changes

- target *read.Rout* will be older
- regarded as being **out of date**

Run *make* by typing *make* at the command line or pressing the appropriate button in your *IDE*

If *read.R* newer, *R CMD BATCH read.R* is run

If *read.Rout* is newer, then

```
$ make
make: 'read.Rout is up to date'.
```

Pattern Rules (Automatic Rules)

- Don't want to write rules every time
- Need automatic (pattern) rules
- `GNU Make` has pattern rules for many languages
  (C, C++, Fortran, Ratfor, Yacc, Lex, Info Texinfo, Tex)

Problem: *GNU Make* does not have rules for statistical languages like *R*, *Stata*, *SPSS*, *SAS*, *GENSTAT*, *Perl*, *Python*, ...

Solution: Define pattern rules, eg

```
%.Rout: %.R
<TAB> R CMD BATCH $<
```

Pattern rules look pretty much like normal rules except

- the wild card symbol % is used before the file extension (pattern)
- $< is automatic variable: the filename of **first dependency**

Pattern Rules (Automatic Rules)

Now easily write `Makefiles` by specifying targets and dependencies

```
report.pdf: report.Rmd plots.pdf

plots.pdf: plots.R read.Rout

read.Rout: read.R bmi2019.dta
```

as long as we have appropriate pattern rules.

r-rukes.mk rules

Pattern rules provided for

- Statistics packages (and related)
  - R
  - Sweave
  - R Markdown
  - Stata
  - SAS
  - PSPP
- Data science
  - Python
  - Perl

Caveat: `Windows` and `macOS` users may need a *better* GNU Make

- `Windows`: install latest `RTools`
- `macOS`: install `gmake` via homebrew https://brew.sh/

In practice

- don't need to write rules every time
- *include* pattern rules from a file
- a selection of rules available at github (Baker 2019) https://github.com/petebaker/r-makefile-definitions

Simply **include r-rules.mk** at end of file

```
include ~/lib/r-rules.mk
```

or similarly on Windows

```
include C:/MyLibrary/r-rules.mk
```

or in system wide directory like `/usr/local/include`

```
include r-rules.mk
```

Handwritten Simple Makefile

```
## File:    Makefile
## Purpose: Simple Example

.PHONY: all
all: report1.pdf report2.docx

## reports 1&2 depend on results of 'linmod.Rout' & '*.Rmd'
report1.pdf: report1.Rmd linmod.Rout
report2.docx: report2.Rmd linmod.Rout

## data analysis:  dependent on 'linmod.R' and 'read.Rout'
linmod.Rout: linmod.R read.Rout

## read in data: depends on 'read.R' and 'simple.csv'
read.Rout: read.R simple.csv

## include R pattern rule definitions from file
include r-rules.mk
```

Section 4

The gnumaker R package

Do I really have to learn another language?

Q: What if I don't know GNU Make?

Do I really have to learn another language?

Q: What if I don't know GNU Make?

A: Generate Makefile with `gnumaker` package

- get `R` to write the `Makefile`
- uses pattern rules available at github
- get help on available pattern rules
- check consistency (is dependency graph a DAG?)
- plot the file dependency graph (DAG)

Generate Makefile with `create_makefile` (1)

targets: list for targets and dependencies

```
library(gnumaker)
gm1 <-
  create_makefile(
    targets = list(read = c("read.R", "simple.csv"),
      linmod = c("linmod.R", "read"),
      rep1 = c("report1.Rmd", "linmod")))
```

Default target files are produced from first file. eg

- `read.Rout` from `read.R`
- `report1.pdf` from `report1.Rmd`

Dependencies may include previous steps in the process

- `rep1` depends on `linmod` (`linmod.Rout`)
- `linmod` depends on `read` (`read.Rout`)

Generate Makefile with `create_makefile` (2)

target.all: list for `.PHONY` `all` targets at top of file (always made)

```
gm1 <-
  create_makefile(
    targets = list(read = c("read.R", "simple.csv"),
      linmod = c("linmod.R", "read"),
      rep1 = c("report1.Rmd", "linmod"),
      rep2 = c("report2.Rmd", "linmod")),
    target.all = c("rep1", "rep2"))
```

- targets `rep` and `rep2` will be made, or
- files `report1.pdf` and `report2.pdf` will be made once all previous steps made first

Generate Makefile with `create_makefile` (3)

all.exts: character vector of file name extensions for `.PHONY all` target

```
library(gnumaker)
gm1 <-
  create_makefile(
    targets = list(read = c("read.R", "simple.csv"),
      linmod = c("linmod.R", "read"),
      rep1 = c("report1.Rmd", "linmod"),
      rep2 = c("report2.Rmd", "linmod")),
    target.all = c("rep1", "rep2"),
    all.exts = list(rep1 = "pdf", rep2 = "docx"))
```

- `report1.pdf` and `report2.docx` will be made once all previous steps made first

Generate Makefile with `create_makefile` (4)

comments: list to override default coments

```
gm1 <-
 create_makefile(
  targets = list(read = c("read.R", "simple.csv"),
    linmod = c("linmod.R", "read"),
    rep1 = c("report1.Rmd", "linmod"),
    rep2 = c("report2.Rmd", "linmod")),
  target.all = c("rep1", "rep2"),
  all.exts = list(rep1 = "pdf", rep2 = "docx"),
  comments =
   list(linmod = "plots and analysis using 'linmod.R'"))
```

Changes default comment for `linmod`

What's in gm1? (1)

```
gm1
```

```
## Makefile:
##  [1] "# .PHONY all target which is run when make is invoked"
##  [2] ".PHONY: all"
##  [3] "all: report1.pdf report2.docx"
##  [4] ""
##  [5] "# report1.pdf depends on report1.Rmd, linmod.Rout"
##  [6] "report1.pdf: report1.Rmd linmod.Rout"
##  [7] ""
##  [8] "# report2.docx depends on report2.Rmd, linmod.Rout"
##  [9] "report2.docx: report2.Rmd linmod.Rout"
## [10] ""
## [11] "# plots and analysis using 'linmod.R'"
## [12] "linmod.Rout: linmod.R read.Rout"
## [13] ""
## [14] "# read.Rout depends on read.R, simple.csv"
## [15] "read.Rout: read.R simple.csv"
## [16] ""
## [17] "# include GNU Makfile rules. Most recent version available at"
## [18] "# https://github.com/petebaker/r-makefile-definitions"
## [19] "include ~/lib/r-rules.mk"
## [20] ""
## [21] "# remove all target, output and extraneous files"
```

What's in gm1? (2)

```
## [1] "# remove all target, output and extraneous files"
## [2] ".PHONY: cleanall"
## [3] "cleanall:"
## [4] "\trm -f *~ *.Rout *.RData *.docx *.pdf *.html *-syntax.R *.RData"

##
## Makefile DAG:

## A graphNEL graph with directed edges
## Number of Nodes = 10
## Number of Edges = 10
```

```
class(gm1)
```

```
## [1] "gnu_makefile"
```

```
write_makefile(gm1, file = "Makefile.demo")
```

```
## File: Makefile.demo written at Wed Jul 10 12:21:42 2019
```

Contents of Makefile.demo (1)

```
# File: Makefile.demo
# Created at: Wed Jul 10 12:21:42 2019

# Produced by gnumaker:  0.0.0.9005 on R version 3.6.0 (2019-04-
# Before running make, please check file and edit if necessary

# .PHONY all target which is run when make is invoked
.PHONY: all
all: report1.pdf report2.docx

# report1.pdf depends on report1.Rmd, linmod.Rout
report1.pdf: report1.Rmd linmod.Rout

# report2.docx depends on report2.Rmd, linmod.Rout
report2.docx: report2.Rmd linmod.Rout

# plots and analysis using 'linmod.R'
linmod.Rout: linmod.R read.Rout
```

Contents of Makefile.demo (2)

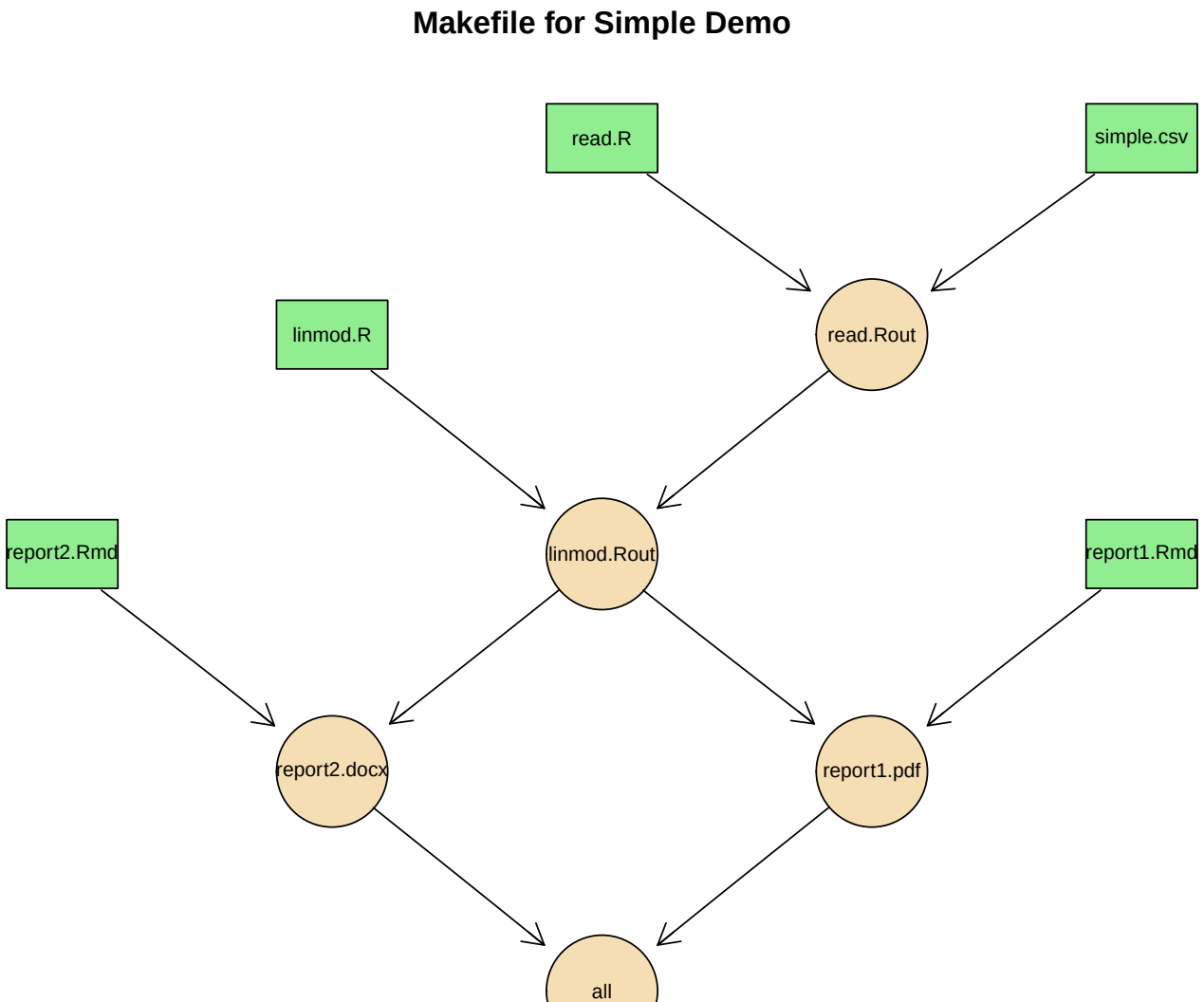
```
# read.Rout depends on read.R, simple.csv
read.Rout: read.R simple.csv

# include GNU Makfile rules. Most recent version available at
# https://github.com/petebaker/r-makefile-definitions
include ~/lib/r-rules.mk

# remove all target, output and extraneous files
.PHONY: cleanall
cleanall:
    rm -f *~ *.Rout *.RData *.docx *.pdf *.html *-syntax.R *.RDa
```

Dependency file graph (DAG)

```
plot(gm1, main = "Makefile for Simple Demo",
     attrs = list(node = list(fontsize = 16)))
```



info_rules (1)

Display information about pattern rules with

```
info_rules(list.all = TRUE)
```

```
## Dependency file name extensions:
##  [1] "_Article.Rnw"          "rnw"
##  [3] "Rnw"                   "_beamer-handout.pdf"
##  [5] "_beamer-handout.Rmd"   "Rmd"
##  [7] "rmd"                   "_Handout.Rnw"
##  [9] "_Notes.Rnw"            "_Present.Rnw"
## [11] "_Handout.pdf"          "snw"
## [13] "Snw"                   "pl"
## [15] "PL"                    "py"
## [17] "PY"                    "r"
## [19] "R"                     "sps"
## [21] "SPS"                   "do"
## [23] "DO"                    "sas"
## [25] "SAS"                   "tex"

## NULL
```

info_rules (2)

Display information about particular pattern rule with

```
info_rules("Rmd")
```

```
Possible filename extensions for 'Rmd':
 [1] "_beamer-handout.Rmd" "_beamer.pdf"
 [3] "_ioslides.html"      "_slidy.html"
 [5] "_tufte.pdf"          "-syntax.R"
 [7] "docx"                "html"
 [9] "odt"                 "pdf"
[11] "pptx"                "rtf"

Default: 'html'

Example rule:
example1.html: example1.Rmd dep_file2 dep_file3

Other options are available for R Markdown files, such as:

example1_ioslides.html: example1.Rmd dep_file2 dep_file3
```

info_rules (3)

info_rules("Rmd") (continued)

Other options are available for R Markdown files, such as:

example1_ioslides.html: example1.Rmd dep_file2 dep_file3
example1_beamer.pdf: example1.Rmd dep_file2 dep_file3

to produce ioslide and beamer presentation formats.

An R syntax file can be produced with
    make example1-syntax.R
and a similar rule can be specified if necessary with
example1-syntax.R: example1.Rmd dep_file2 dep_file3

What if I need more help/customisation?

- type 'make help' at command line
- set variables (globally). eg

```
## include rules
include ~/lib/r-rules.mk
RMARKDOWN_PDF_OPTS = \"bookdown::pdf_document2\"
```

- set target specific variables
- Multiple targets
- Mixed `Sweave` `knitr`

See (Baker 2019) for more details

Section 5

Conclusions

Summary: GNU Make

The three most useful tools we can use to aid the data analysis workflow and facilitate reproducible research are

1. GNU Make
2. Git
3. R Markdown (and R)

GNU Make

- useful for efficient modular workflow,
- documents workflow
- good documentation (GNU Make manual, Graham-Cumming (2015), Mecklenburg (2004))
- many alternative build systems but few mature or used widely (eg see `Drake`, `Remake`, `Scons`)

Conclusions

The `gnumaker` R package can help you get started.

Using simple lists, it produces

- Makefiles
- checks DAG of relationships
- plots DAG
- provides help/overview of pattern rules

Future enhancements

- help on environmental variables
- generic DAG plot for all `Makefiles`

Thank you

References

Baker, Monya. 2016. "1,500 Scientists Lift the Lid on Reproducibility." *Nature News* 533 (7604): 452. https://doi.org/10.1038/533452a.

Baker, Peter. 2019. "Using GNU Make to Manage the Workflow of Data Analysis Projects." *Journal of Statistical Software (Accepted Jan 2019)*.

Graham-Cumming, John. 2015. *The GNU Make Book*. No Starch Press.

Long, J. Scott. 2009. *The Workflow of Data Analysis Using Stata*. StataCorp LP.

Mecklenburg, Robert. 2004. *Managing Projects with GNU Make*. 3rd ed. O'Reilly Media, Inc.