A tutorial introduction to Make for reproducible research

Iain Davies and Daniel Nuest and Stephen J Eglen

August 28, 2020

1 Introduction

2 Introduction

Why make is useful. Originally designed into 1970s for the efficient compilation of programs. Gradually adopted to other situations.

Note the workflow here

1. Generate 2. Analyse 3. Summarise

If Generating the data takes 10 hours, and analysing takes only 2 minutes, you don't want to rerun whole pipeline to regenerate data.

3 Example problem

3.1 Estimating pi

A useful example for using Make is the dartboard method for estimating pi.

Consider a dartboard of radius 1 set in a square of side length 2. We randomly throw darts uniformly across the square and count the fraction that land in the dartboard. In expectation, this fraction will be the ratio of the area of the dartboard (pi) to the area of the square (4). Hence we estimate pi as 4 times the fraction of darts that land in the dartboard. We may also want to display the position of the darts in a diagram for the ease of future readers.

We can write programs for this process which neatly capture the three steps of workflow above:

1. Generating data. This is throwing the darts at the square and storing the position of each in an accesible format.
    - inputs.dat - a file that states n, the number of darts to be thrown.
    - throw-darts.R - a R script that reads n from inputs.dat and outputs a list of the positions of n randomly thrown darts which is written to darts.xy.
    - darts.xy - the list of positions of n darts outputted by throw-darts.R.
2. Analysis. This is calculating the fraction of the darts which landed inside the dartboard and inferring an estimate of pi.
    - count-inside.R - a R script that reads the position of each dart from darts.xy and writes whether it is inside the dartboard to inside.out.
    - inside.out - a list of logicals corresponding to whether each dart is inside the dartboard.

- estimate.R - a R script that calculates an estimate of pi inferred from the list in inside.out. The estimate is written to pi.est.
- pi.est - a file containing the estimate of pi .

3. Summary. This is displaying the position of our thrown darts and our estimate of pi in an easily readable diagram.
    - draw-figure.R - a R script that creates a diagram of the positions of the darts and estimate of pi and saves it to darts.pdf.
    - darts.pdf - a diagram displaying the darts and estimate of pi.

We can use Make to execute these programs in the correct order and without repeating steps that haven't changed.

3.2 Traditional approach - write a batch file

```
#!/bin/sh
./throw-darts inputs.dat > darts.txt
./count-inside darts.txt > inside.out
./estimate inside.out > pi.est
./draw-figure darts.txt inside.out pi.est output.pdf
```

3.3 Version 1 (R makefile)

```
.PHONY: clean all

all: darts.pdf

darts.xy: inputs.dat throw-darts.R
	Rscript throw-darts.R inputs.dat > darts.xy

inside.out: darts.xy count-inside.R
	Rscript count-inside.R darts.xy > inside.out

pi.est: inside.out estimate.R
	Rscript estimate.R inside.out > pi.est



darts.pdf: darts.xy inside.out pi.est draw-figure.R
	Rscript draw-figure.R darts.xy inside.out pi.est darts.pdf


clean:
	rm -f darts.xy inside.out pi.est darts.pdf



## This depends on https://github.com/lindenb/makefile2graph

graph.pdf: Makefile
	make -Bnd | makefile2graph | dot -Tpdf > graph.pdf
```

The above is a simple Makefile that does the job. So how does it work?

3.3.1 Anatomy of a Rule

A Makefile is a list of instructions of how to create files in a process. It does this via *rules*. *Rules* consist of a *target* file to be made, its *dependencies* and finally a *recipe* to make the target. For example, take the *rule* for darts.xy above:

```
darts.xy: inputs.dat throw-darts.R
	Rscript throw-darts.R inputs.dat > darts.xy
```

The *target* is darts.xy. This is followed by a colon then a list of *dependencies* - inputs.dat and throwdarts.R - which are the files needed to make darts.xy. Finally, the second line starts with a tab and then gives the *recipe* to make darts.xy. This is a command run in the terminal - in this case executing the R file throw-darts.R with input input.dat and piping the output into darts.xy.

A *rule* must be given for all files that are created in the process. For example, the Makefile above also gives *rules* for inside.out (a list of whether the darts are inside the dartboard), pi.est (our esimate for pi) and darts.pdf (the diagram of thrown darts).

3.4 Making Files - Importance of Dependencies

If a file has a rule in the Makefile it can be made from the command line by executing "make <filename>". For example we can execute the recipe to make darts.xy by typing the following:

```
cd v1-R
make darts.xy

## make[1]: Entering directory '/mhome/damtp/s/id318/Documents/Make_tutorial/make-tutorial
## Rscript throw-darts.R inputs.dat > darts.xy
## make[1]: Leaving directory '/mhome/damtp/s/id318/Documents/Make_tutorial/make-tutorial/
```

This executes the recipe in the command line and produces the file darts.xy as desired.

The clever part of the Make rule structure is that it determines whether a file should be remade by looking at the timestamps of the dependencies. If we repeat this command without changing any of the dependencies then Make will simply say that nothing is to be done:

```
cd v1-R
make darts.xy

## make[1]: Entering directory '/mhome/damtp/s/id318/Documents/Make_tutorial/make-tutorial
## make[1]: 'darts.xy' is up to date.
## make[1]: Leaving directory '/mhome/damtp/s/id318/Documents/Make_tutorial/make-tutorial/
```

This is the importance of dependencies - a target will only be remade if at least one of its dependencies Furthermore, before making the target Make will remake any of its dependencies if their own dependencies have a more recent timestamp and so on.

3.5 .PHONY targets

3.6 Running it (for real)

```
cd v1-R
make

## make[1]: Entering directory '/mhome/damtp/s/id318/Documents/Make_tutorial/make-tutorial
## Rscript count-inside.R darts.xy > inside.out
## Rscript estimate.R inside.out > pi.est
## Rscript draw-figure.R darts.xy inside.out pi.est darts.pdf
## make[1]: Leaving directory '/mhome/damtp/s/id318/Documents/Make_tutorial/make-tutorial/
```

This should generate a new pdf, 'darts.pdf'. If we then delete darts.pdf by accident, when we run 'make' again, it will not need to run all steps of the analysis, as the intermediate files are still present.

```
cd v1-R
rm darts.pdf
make

## make[1]: Entering directory '/mhome/damtp/s/id318/Documents/Make_tutorial/make-tutorial
## Rscript draw-figure.R darts.xy inside.out pi.est darts.pdf
## make[1]: Leaving directory '/mhome/damtp/s/id318/Documents/Make_tutorial/make-tutorial/
```

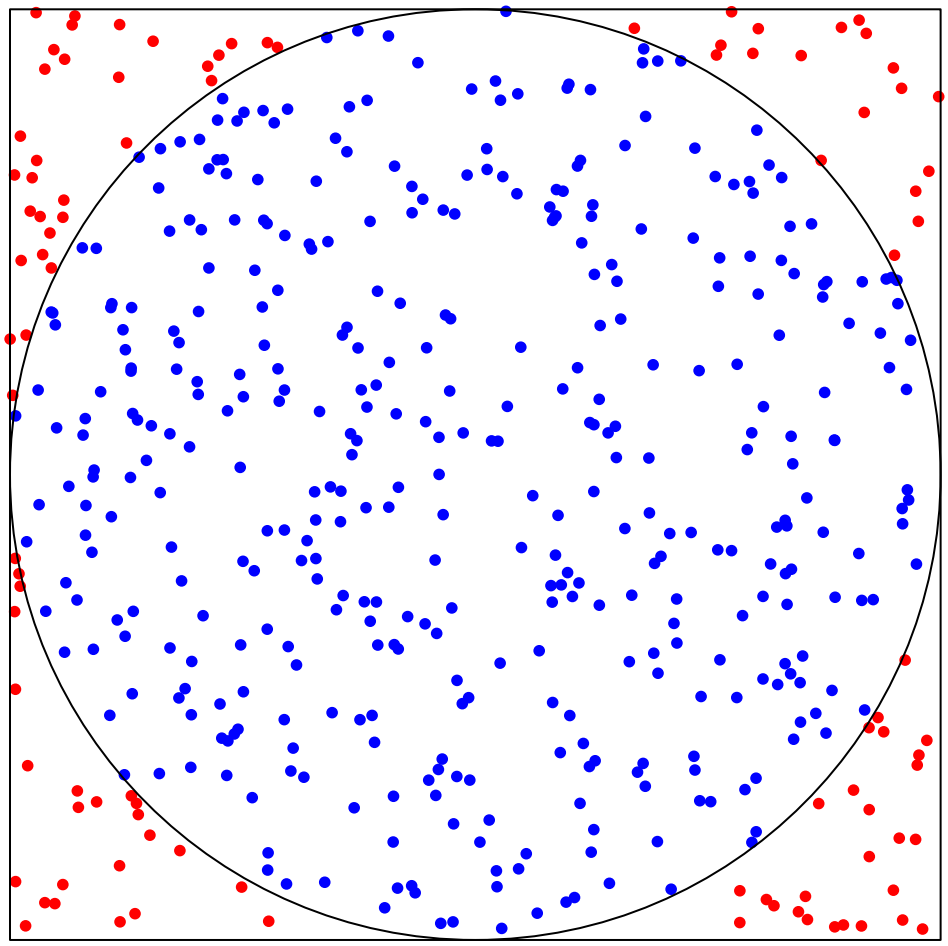

Figure 1: Example output file, darts.pdf, created by "make" command. Blue (or red) points are those that were determined to be inside (or outside) the circle. The estimate of pi, given in the title, was then given as 4*d/n, where d is the number of darts inside the circle and n is the total number of darts thrown.

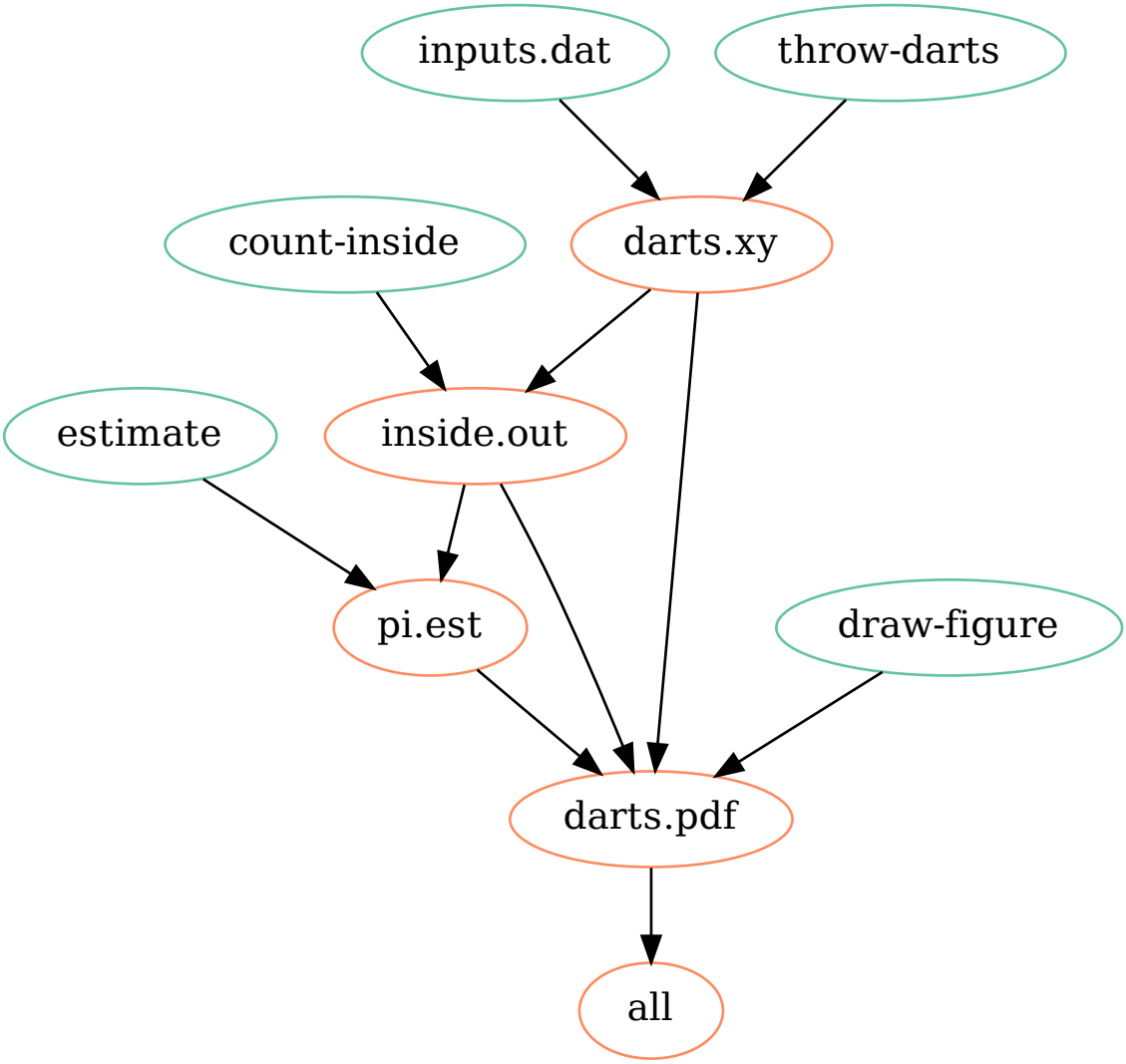

Figure 2: DAG for Makefile version 1. green ellipses correspond to those files that are up to date and do not need to be regenerated; red ellipsis are those files that need to be remade.

3.7 DAG

One nice thing about using a Makefile is usually you can visualize the dependency graph. e.g. Figure 2. Input files are clearly shown in green, and everything else in red can be regenerated.

(This figure is generated by analysing the structure of the output from the make program, thanks to a program from `makefile2graph`)

3.8 Explain what PHONY targets are

Two common PHONY targets are the ”all” and ”clean” targets. ”make all” by convention is a request to regenerate all the files in the directory, rather than to generate a file called ”all”. Likewise, ”clean” is a convention to remove all the files that can be deleted.

4 ersion 2 – generate lots of simulations

In version 2, we want to highlight how to make rules more flexible. Example application here would be generating lots of simulation runs, rather than just one.

```
# Tell Emacs this is a -*- Makefile -*-

.PHONY: all clean

REPEATS := $(shell seq 1 15)

TARGETS := $(patsubst %, pi-%.est, $(REPEATS))

all: $(TARGETS)

darts-%.xy: inputs.dat
	./throw-darts inputs.dat > $@

inside-%.out: darts-%.xy
	./count-inside $^ > $@


pi-%.est: inside-%.out
	./estimate $^ > $@


# https://blog.jgc.org/2015/04/the-one-line-you-should-add-to-every.html
print-%: ; @echo $*=$($*)


clean:
	rm -f $(TARGETS)
```

Try to generate B=15 samples, and then show a histogram of the distribution of the B estimates of pi.

```
cd v1-R
make -f Makefile2
cat pi-*est

## make[1]: Entering directory '/mhome/damtp/s/id318/Documents/Make_tutorial/make-tutorial
## ./throw-darts inputs.dat > darts-1.xy
## /bin/sh: 1: ./throw-darts: not found
## Makefile2:12: recipe for target 'darts-1.xy' failed
## make[1]: *** [darts-1.xy] Error 127
## rm darts-1.xy
## make[1]: Leaving directory '/mhome/damtp/s/id318/Documents/Make_tutorial/make-tutorial/
## cat: 'pi-*est': No such file or directory
```

Describe make -j8 to run this in parallel.
NOTE: intermediate files are not kept.
Perhaps show a 4x3 grid of 12 simulations?

5 What next

Further reading (books); other programs that build upon idea of make (snakemake, drake).
`https://www.frontiersin.org/articles/10.3389/fninf.2016.00002/full`
Portable make code: `https://github.com/markpiffer/gmtt`

5.1 Acknowledgements

Mozilla CODECHECK for funding