Make and Makefiles for workflows

Hannah Holland-Moritz

April 13, 2020

Overview

Resources/Links/Inspiration:

Today's presentation is heavily inspired by:

1. Sofware Carpentry's lesson in makefile
   - https://swcarpentry.github.io/make-novice/
2. Karl Broman's "minimal make" tutorial
   - https://kbroman.org/minimal_make

Today's Topics

1. Introduction to Make
2. Anatomy of a makefile
3. Phony targets
4. Makefiles for an R workflow

Why you should care

Scenario 1:

You realize you made a mistake in the lab or field and now need to replace your old data with the new, more accurate data. You do so and re-run your analysis. Then you receive data from a collaborator that you've been waiting on for months. Yay! Some of your figures depend on this data and some don't. You updated your code and then re-run the whole thing from start to finish just to be sure. The collaborator tells you that one of the samples in the previous data they sent you was a mistake. You should remove it from the analyses. You now need to run everything again but without this one data point. etc. etc.

Scenario 2:

Your advisor or a reviewer asks you to redo an analysis in a slightly different way. You internally groan. Although you have all the code and the change is relatively small, the change is at the very beginning of your computational pipeline. You are going to have to re-run 20 steps one at a time to incorperate the change. Some of the steps take quite a long time to run but others take only a short time, very few of the steps take a convenient amount of time in which you could get something useful done. You spend an entire day of work pressing "enter" and doing small tasks on the side.

Scenario 3:

You make a small change to a script that results in changes to only one of your figures, because you can't remember the exact dependencies of your pipeline, you re-run the whole pipeline to make sure every thing is updated. Or worse, you don't re-run the whole pipeline and one of your statistics is incorrect. You can't understand why you are getting this weird outcome. It takes you months to realize the issue. Or you never do. Oops.

Scenario 4:

A collaborator asks you for your code so they can build on some of your analyses. You say yes of course, then you realize, they won't be able to run this part of it with out some other part of the pipeline. Or if they run one script (step 4) they will need to run the scripts for step 1 and step 2 (but not step 3) in order for it to work properly. You realize you now have several choices 1) send them all of your code and write them a lengthy email explaining the idiosyncracies of your pipeline (and hope you don't forget anything), 2) run the analysis for them instead of giving them the code, 3) create a custom analysis script just for their use. To your dismay each of these options will use a lot of time (either you or your collaborator's) even though the initial request was fairly straightforward.

Why should we be using makefiles?

1. Reproducible analysis with minimal brain input

Why should we be using makefiles?

1. Reproducible analysis with minimal brain input
2. Self-documenting analysis pipeline

Why should we be using makefiles?

1. Reproducible analysis with minimal brain input
2. Self-documenting analysis pipeline
3. More efficient use of your time and energy

Challenges of make

1. Weird rules and idiosyncrasies

Challenges of make

1. Weird rules and idiosyncrasies
2. Can have cryptic error messages

Challenges of make

1. Weird rules and idiosyncrasies
2. Can have cryptic error messages
3. Requires a bit of knowledge about the unix shell

Today's goal:

Learn to use `make` for your pipelines with minimal background knowledge.

Introduction to `make`

`make` is a program that executes instructions in a makefile

- `make` looks for a file called "Makefile" and runs all of the instructions in the makefile
- By default it assumes your makefile is called "Makefile", but you can specify different filename if you want using the `-f` flag

How to run make:

```
[you@localhost make_example]$ make
```

Alternatively if your makefile is called "mymake.txt":

```
[you@localhost make_example]$ make -f mymake.txt
```

Anatomy of a makefile

Anatomy

```
# Makefile structure
target: dependency
    action
```

- *target*: output files that you are trying to make
- *dependency*: the files (including scripts and data), that your outputs depend upon
- *action*: a command that can be run in a terminal that describes what to do with the dependencies.

Anatomy of a makefile

Makefile

```
# Makefile structure
clean_data.txt : clean_data.R raw_data.txt
    R CMD BATCH clean_data.R

Figure_01.pdf fig_01_data.RDS: make_fig_01.R clean_data.txt
    R CMD BATCH make_fig_01.R

Figure_02.pdf fig_02_data.RDS: make_fig_02.R clean_data.txt
    R CMD BATCH make_fig_02.R
```

Important points (1):

The first line must be on one line. But you can break it up into multiple human-readable lines with line breaks (backslashes) for easy reading.

With a line break

```
Figure_01.pdf \ # <- line break
fig_01_data.RDS: make_fig_01.R clean_data.txt
    R CMD BATCH make_fig_01.R
```

Important points (2):

The second line **must** begin with a TAB, not spaces.

Action lines must begin with tab

```
# Makefile structure
clean_data.txt : clean_data.R raw_data.txt
    R CMD BATCH clean_data.R # This line starts with TAB
```

Important points (3):

The action line is either a command that can be run in the terminal, or a make-specific command.

- To run an R script in the terminal use either `R CMD BATCH myscript.R` or `Rscript myscript.R`.

Action lines must be executable in the terminal

```
# Makefile structure
clean_data.txt : clean_data.R raw_data.txt
    R CMD BATCH clean_data.R # executable in the terminal
```

Important points (4):

If you accidentally forget a line break or use spaces instead of a tab for the second line you'll see the following error:

Missing seperator error

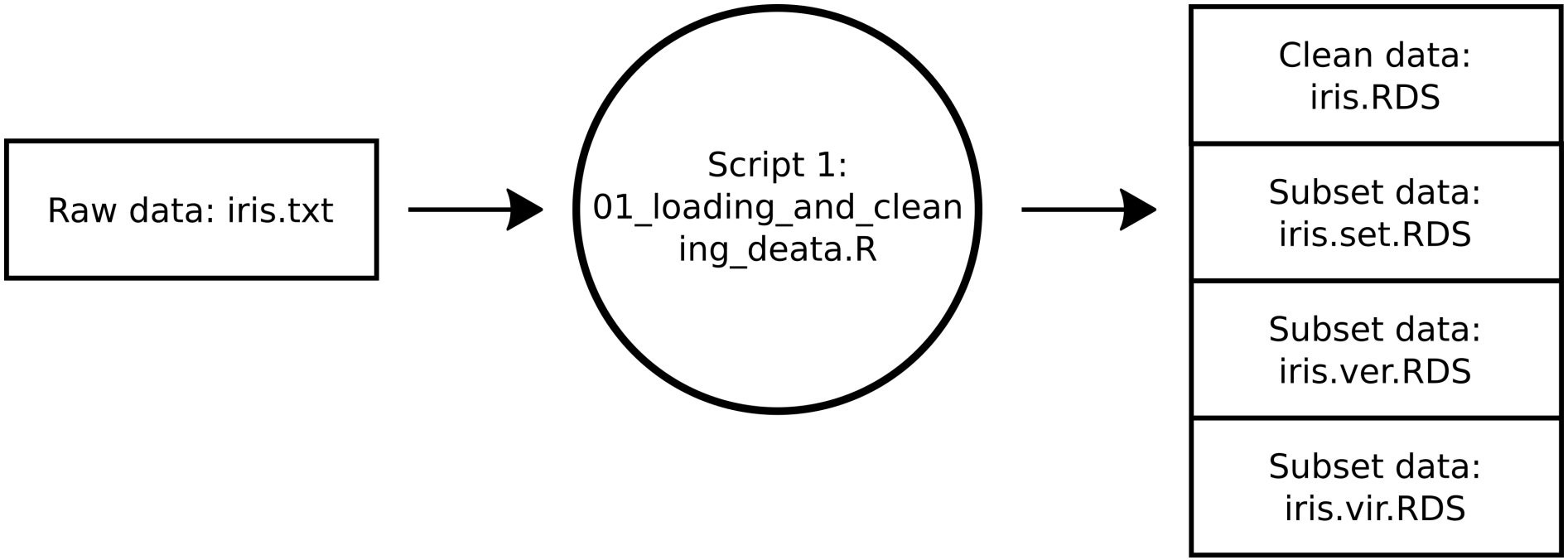
```
[you@localhost make_example]$ make
Makefile:5: *** missing separator.  Stop.
```

If your action line is not able to run, you'll get a much more cryptic error.

Execution error

```
[you@localhost make_example]$ make
make: 01_loading_and_cleaning_data.R: Command not found
make: *** [Makefile:9: iris.RDS] Error 127
```

Your turn: Write a makefile for the following workflow



Your turn:

Makefile_00

Makefile

```
# Cleaning Data ========================================

# Files from 01_loading_data_and_cleaning_mapping_file.R
iris.RDS \
iris.set.RDS \
iris.ver.RDS \
iris.vir.RDS: 01_loading_and_cleaning_data.R iris.txt
    R CMD BATCH 01_loading_and_cleaning_data.R
```

Your turn:

Running your makefile

“Phony” targets

Shortcuts

Makefiles can be made less repetative using shortcut variables. Shortcut variables can also be used to prevent typos when filenames change.

Shortucut Variables

- $@ = "the target of the current rule"
- $^ = "all the dependencies of the current rule"
- $< = "the first dependency of the current rule"
- $(@D) = "the directory part of the target"
- $(@F) = "the file part of the target"
- $(<D) = "the directory part of the dependency"
- $(<F) = "the file part of the dependency"

For a more in-depth tutorial on reducing repetition in your makefile see steps 3 - 7 in https://swcarpentry.github.io/make-novice/

Troubleshooting and common errors

1. Missing separator error
2. Makefile always runs all code (i.e. ciruclar dependencies)

Self-Documentation

Makefiles for an R workflow