

# BMI 826-003

## Tools for Reproducible Research

Karl Broman

Biostatistics & Medical Informatics, UW–Madison

`biostat.wisc.edu/~kbroman`

`github.com/kbroman`

`@kwbroman`

Course web: [bit.ly/tools4rr](https://bit.ly/tools4rr)

Reproducible

Reproducible

vs.

Replicable

Reproducible

vs.

Correct

# Levels of quality

- ▶ Are the tables and figures reproducible from the code and data?
- ▶ Does the code actually do what you think it does?
- ▶ In addition to **what** was done, is it clear **why** it was done?  
(e.g., how were parameter settings chosen?)
- ▶ Can the code be used for other data?
- ▶ Can you extend the code to do other things?

# Basic principles

- ▶ Everything via code
- ▶ Everything automated
  - Workflow and dependencies clearly documented
- ▶ Get the data in the most-raw form possible
- ▶ Get any/all data and meta-data possible
- ▶ Keep track of the **provenance** of all data files
- ▶ Be self-sufficient

# Why do we care?

- ▶ Avoid embarrassment
- ▶ More likely correct
- ▶ Save time, in the long run
- ▶ Greater potential for extensions; higher impact

Your closest collaborator is you six months ago,  
but you don't reply to emails.



# What could go wrong?

- ▶ "The attached is similar to the code we used."
- ▶ "Where did this data file come from?!"
- ▶ "Can you repeat the analysis, omitting subject X?"
- ▶ "This part of your script is now giving an error."

# Need to avoid

- ▶ Open a file to extract as CSV
- ▶ Open a data file to do even a slight edit
- ▶ Paste results into the text of a manuscript
- ▶ Copy-paste-edit tables
- ▶ Copy-paste-adjust figures

# Basic tools

- ▶ Automation with Make
- ▶ Unix command line
- ▶ Latex and Markdown
- ▶ Knitr
- ▶ Version control with git
- ▶ R packages
- ▶ Python (or Ruby or Perl)

# Other topics

- ▶ Organizing projects
- ▶ Writing clear code
- ▶ Don't Repeat Yourself (DRY)
- ▶ Testing and debugging
- ▶ Handling big jobs
- ▶ Licenses; human subjects data

# Don't Repeat Yourself

- ▶ In code, in documentation, etc.
- ▶ Repeated bits of code are harder to maintain  
Write a function
- ▶ Use documentation systems like Roxygen2  
Documentation in just one place
- ▶ Make use of others' code

# This course

- ▶ Brief intro to various tools and concepts
- ▶ Try everything out as we go along  
Ask questions!
- ▶ I don't know everything  
Make suggestions!
- ▶ Project
  - Write a bit of R code
  - Use version control
  - Make it an R package
  - Write a vignette

# Automation with GNU Make

- ▶ Make is for more than just compiling software
- ▶ The **essence** of what we're trying to do
- ▶ Automates a workflow
- ▶ Documents the workflow
- ▶ Documents the dependencies among data files, code
- ▶ Re-runs only the necessary code, based on what has changed

# Example Makefile

```
# Example Makefile for a paper
mypaper.pdf: mypaper.bib mypaper.tex Figs/fig1.pdf Figs/fig2.pdf
    pdflatex mypaper
    bibtex mypaper
    pdflatex mypaper
    pdflatex mypaper

# cd R has to be on the same line as R CMD BATCH
Figs/fig1.pdf: R/fig1.R
    cd R;R CMD BATCH fig1.R fig1.Rout

Figs/fig2.pdf: R/fig2.R
    cd R;R CMD BATCH fig2.R fig2.Rout
```



# Fancier example

```
FIG_DIR = Figs

mypaper.pdf: mypaper.tex ${FIG_DIR}/fig1.pdf ${FIG_DIR}/fig2.pdf
    pdflatex mypaper

# One line for both figures
${FIG_DIR}/%.pdf: R/%.R
    cd R;R CMD BATCH $(<F)

# Use "make clean" to remove the PDFs
clean:
    rm *.pdf Figs/*.pdf
```

# How do you use make?

- ▶ If you name your make file `Makefile`, then just go into the directory containing that file and type `make`
- ▶ If you name your make file `something.else`, then type `make -f something.else`
- ▶ Actually, the commands above will build the **first** target listed in the make file. So I'll often include something like the following.

```
all: target1 target2 target3
```

Then typing `make all` (or just `make`, if `all` is listed first in the file) will build all of those things.

- ▶ To build a specific target, type `make target`. For example, `make fig1.pdf`