Naming & Organization

Reproducible Computing @ JSM 2019

Colin Rundel

July 27, 2019

Naming things?

> There are only two hard things in Computer Science: cache invalidation and naming things.
>
> - Phil Karlton

Face it

- There are going to be files
- LOTS of files
- The files will change over time
- The files will have relationships to each other
- It'll will get complicated

Mighty weapon

- File organization and naming is a mighty weapon against chaos
- Make a file's name and location VERY INFORMATIVE about what it is, why it exists, how it relates to other things
- The more things are self-explanatory, the better
- READMEs are great, but don't document something if you could just make that thing self-documenting by definition

What works, what doesn't?

NO

```
myabstract.docx
Joe’s Filenames Use Spaces and Punctuation.xlsx
figure 1.png
fig 2.png
JW7d^(2sl@deletethisandyourcareerisoverWx2*.txt
```

YES

```
2014-06-08_abstract-for-sla.docx
joes-filenames-are-getting-better.xlsx
fig01_scatterplot-talk-length-vs-interest.png
fig02_histogram-talk-attendance.png
1986-01-28_raw-data-from-challenger-o-rings.txt
```

Three principles for (file) names

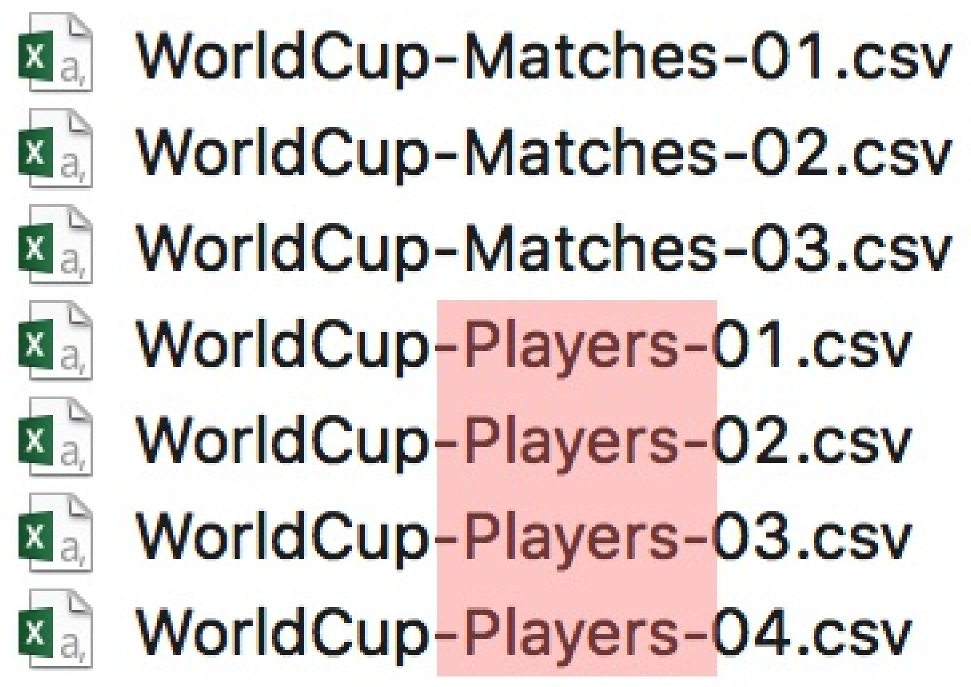
1. Human readable
2. Machine parsable
3. Plays well with OS ordering

Machine Parsable

- Search (Regular expression and globbing) friendly: Avoid spaces, punctuation, accented characters, case sensitivity
- Easy to compute on: Deliberate use of delimiters

Globbing

Excerpt of complete file listing:

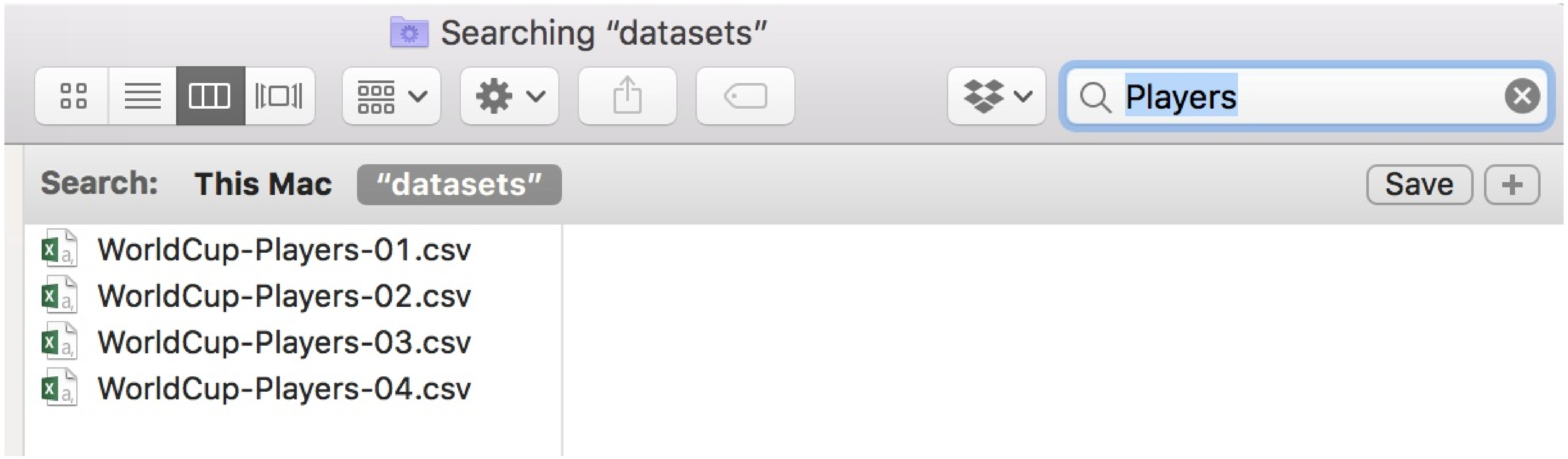


WorldCup-Matches-01.csv
WorldCup-Matches-02.csv
WorldCup-Matches-03.csv
WorldCup-Players-01.csv
WorldCup-Players-02.csv
WorldCup-Players-03.csv
WorldCup-Players-04.csv

Example of globbing to narrow file listing:

```
datasets rundel$ ls *Players*
WorldCup-Players-01.csv    WorldCup-Players-03.csv
WorldCup-Players-02.csv    WorldCup-Players-04.csv
```

Same using Mac OS Finder search facilities



WorldCup-Players-01.csv
WorldCup-Players-02.csv
WorldCup-Players-03.csv
WorldCup-Players-04.csv

Using globs or regexs in R

glob

```
> library(fs)
> dir_ls(glob = "*Players*")
WorldCup-Players-01.csv WorldCup-Players-02.csv WorldCup-Players-03.csv
WorldCup-Players-04.csv
```

regex

```
> library(fs)
> dir_ls(regexp = "Players-\\d{2}")
WorldCup-Players-01.csv WorldCup-Players-02.csv WorldCup-Players-03.csv
WorldCup-Players-04.csv
```

Punctuation

Deliberate use of `"-"` and `"_"` allows recovery of meta-data from the filenames:

- Use one to delimit units of meta-data you might want later
- Use the other to delimit words
- Stay consistent

For example:

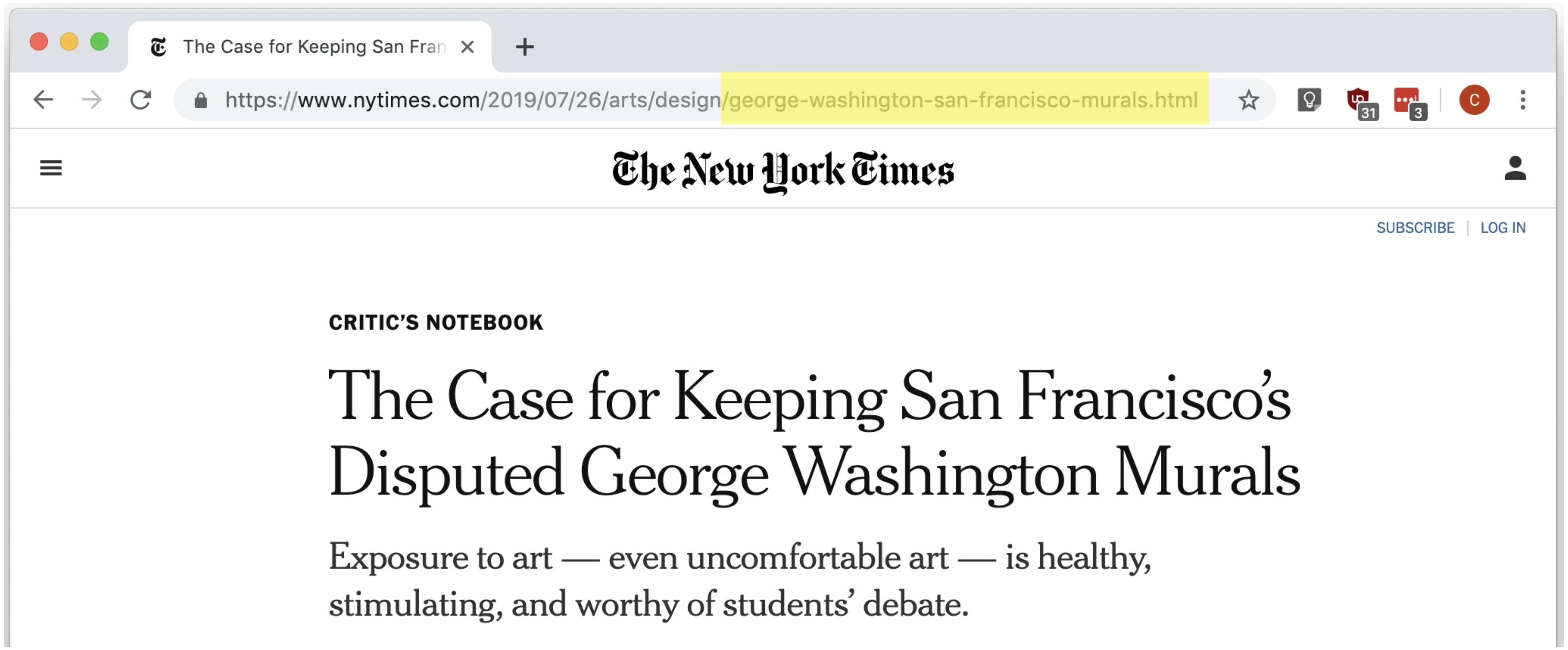
```
2019-09-01_Experiment-1_Rep-A.csv
2019-09-01_Experiment-1_Rep-B.csv
2019-09-07_Experiment-1_Rep-C.csv
2019-09-07_Experiment-2_Rep-A.csv
```

Recap: Machine parsable

- Easy to search for files later
- Easy to narrow file lists based on names
- Easy to extract info from file names, e.g. by splitting
- New to regular expressions and globbing? be kind to yourself and avoid
  - Spaces in file names
  - Punctuation
  - Accented characters (Unicode in general)
  - Different files named `foo` and `Foo`

Human readable

- Name contains info on content
- Connects to concept of a slug from semantic URLs



The Case for Keeping San Fran

https://www.nytimes.com/2019/07/26/arts/design/george-washington-san-francisco-murals.html

The New York Times

SUBSCRIBE | LOG IN

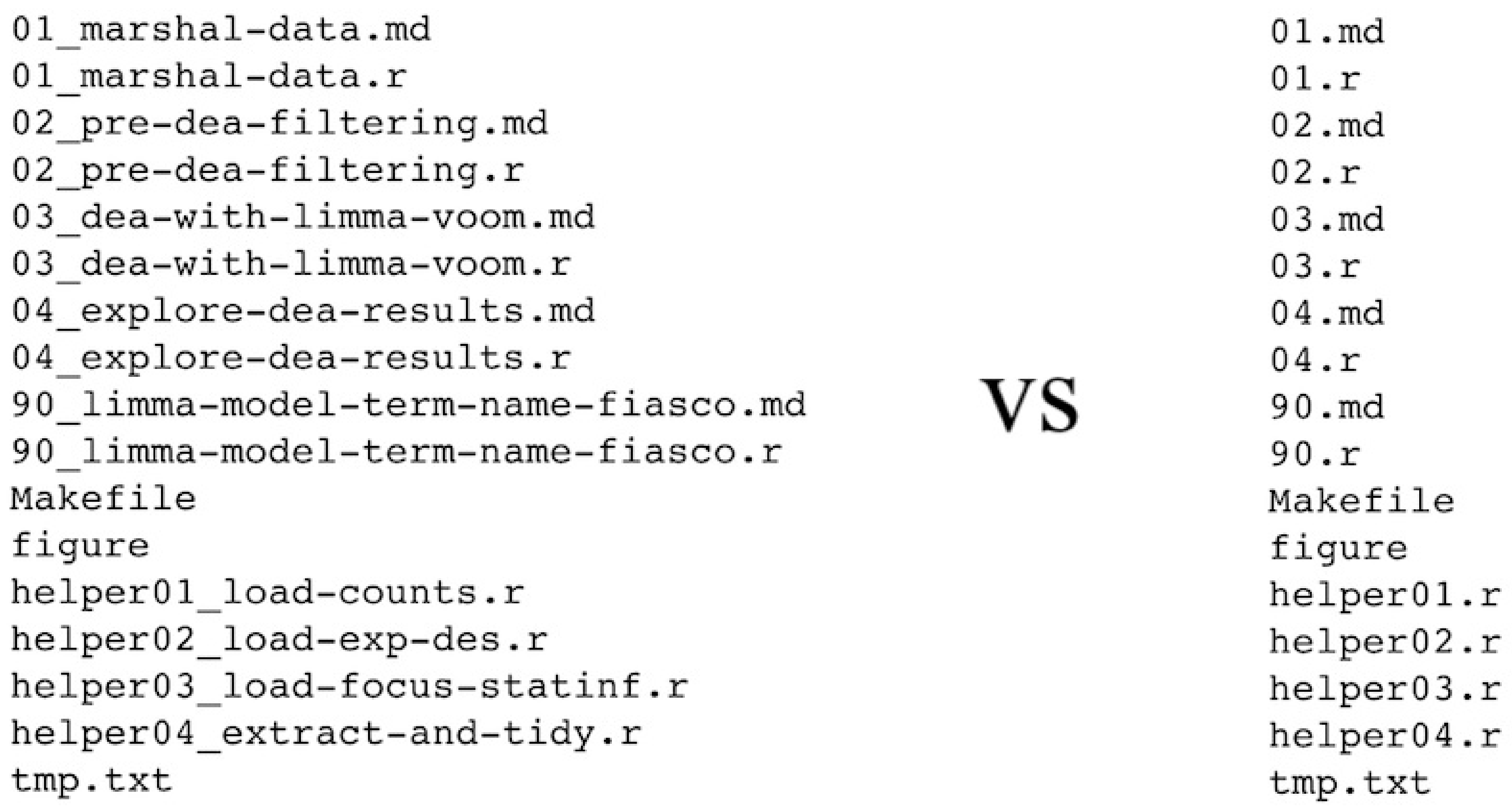
CRITIC'S NOTEBOOK

The Case for Keeping San Francisco's Disputed George Washington Murals

Exposure to art — even uncomfortable art — is healthy, stimulating, and worthy of students' debate.

Example

Which set of file(name)s do you want at 3 a.m. before a deadline?



```
01_marshal-data.md
01_marshal-data.r
02_pre-dea-filtering.md
02_pre-dea-filtering.r
03_dea-with-limma-voom.md
03_dea-with-limma-voom.r
04_explore-dea-results.md
04_explore-dea-results.r
90_limma-model-term-name-fiasco.md
90_limma-model-term-name-fiasco.r
Makefile
figure
helper01_load-counts.r
helper02_load-exp-des.r
helper03_load-focus-statinf.r
helper04_extract-and-tidy.r
tmp.txt
```

VS

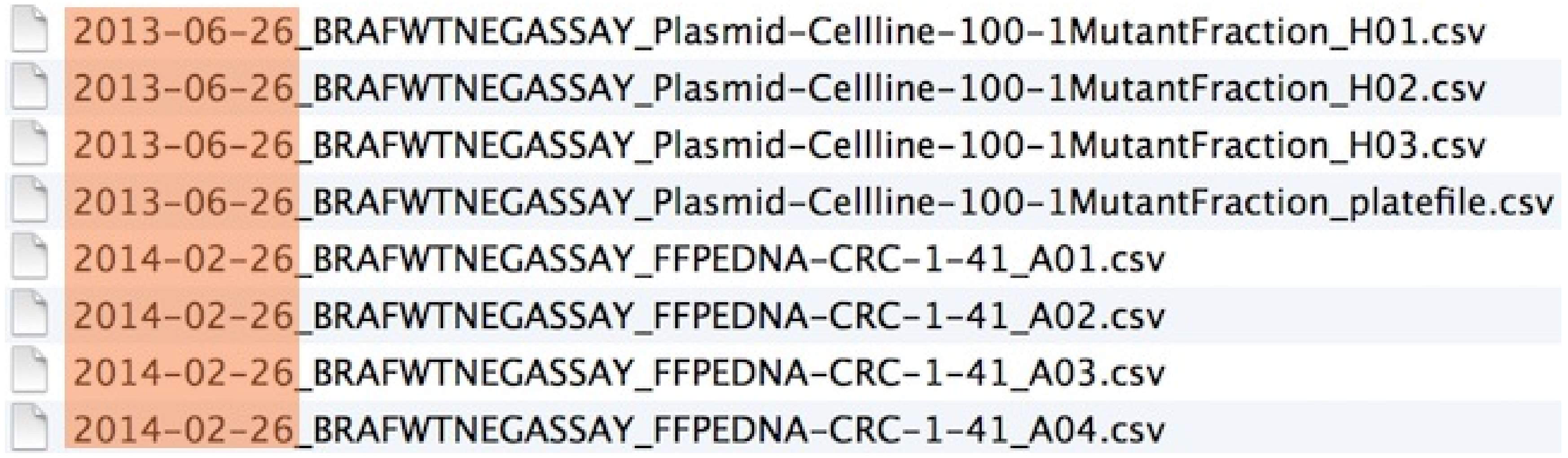
```
01.md
01.r
02.md
02.r
03.md
03.r
04.md
04.r
90.md
90.r
Makefile
figure
helper01.r
helper02.r
helper03.r
helper04.r
tmp.txt
```

Plays well with OS default ordering

- Put something numeric first
- Use the ISO 8601 standard for dates
- Left pad other numbers with zeros

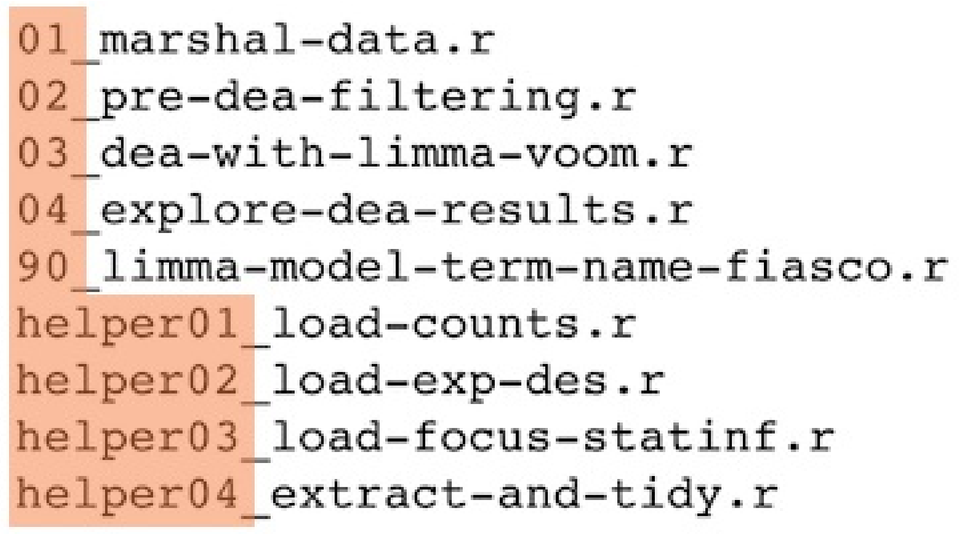
Examples

Chronological order:



```
2013-06-26_BRAFWTNEGASSAY_Plasmid-Cellline-100-1MutantFraction_H01.csv
2013-06-26_BRAFWTNEGASSAY_Plasmid-Cellline-100-1MutantFraction_H02.csv
2013-06-26_BRAFWTNEGASSAY_Plasmid-Cellline-100-1MutantFraction_H03.csv
2013-06-26_BRAFWTNEGASSAY_Plasmid-Cellline-100-1MutantFraction_platefile.csv
2014-02-26_BRAFWTNEGASSAY_FFPEDNA-CRC-1-41_A01.csv
2014-02-26_BRAFWTNEGASSAY_FFPEDNA-CRC-1-41_A02.csv
2014-02-26_BRAFWTNEGASSAY_FFPEDNA-CRC-1-41_A03.csv
2014-02-26_BRAFWTNEGASSAY_FFPEDNA-CRC-1-41_A04.csv
```

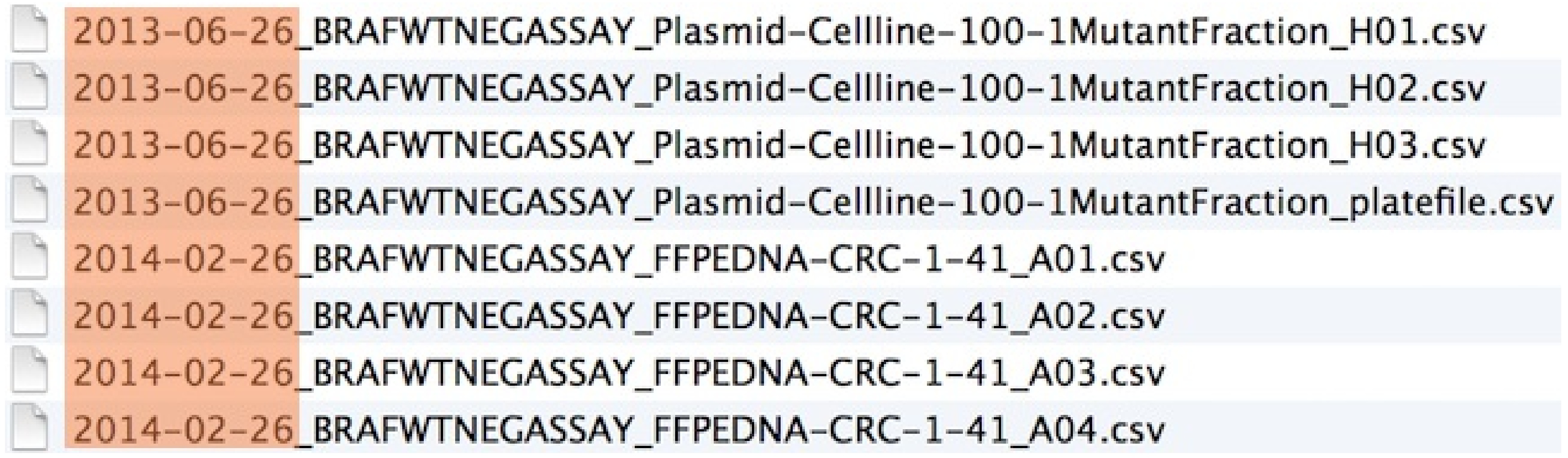
Logical order: Put something numeric first



```
01_marshal-data.r
02_pre-dea-filtering.r
03_dea-with-limma-voom.r
04_explore-dea-results.r
90_limma-model-term-name-fiasco.r
helper01_load-counts.r
helper02_load-exp-des.r
helper03_load-focus-statinf.r
helper04_extract-and-tidy.r
```

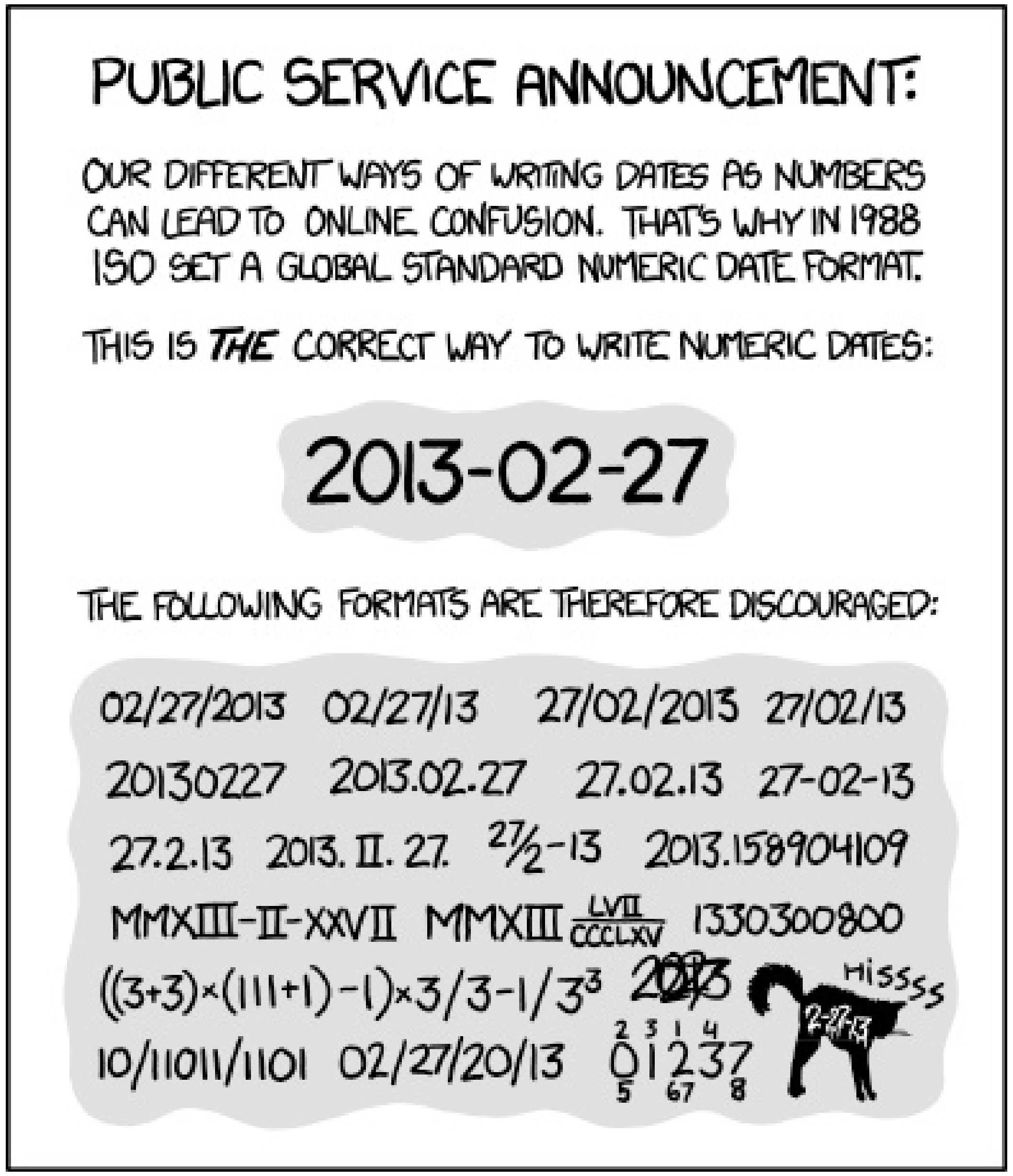
Dates

Use the ISO 8601 standard for dates: YYYY-MM-DD



```
2013-06-26_BRAFWTNEGASSAY_Plasmid-Cellline-100-1MutantFraction_H01.csv
2013-06-26_BRAFWTNEGASSAY_Plasmid-Cellline-100-1MutantFraction_H02.csv
2013-06-26_BRAFWTNEGASSAY_Plasmid-Cellline-100-1MutantFraction_H03.csv
2013-06-26_BRAFWTNEGASSAY_Plasmid-Cellline-100-1MutantFraction_platefile.csv
2014-02-26_BRAFWTNEGASSAY_FFPEDNA-CRC-1-41_A01.csv
2014-02-26_BRAFWTNEGASSAY_FFPEDNA-CRC-1-41_A02.csv
2014-02-26_BRAFWTNEGASSAY_FFPEDNA-CRC-1-41_A03.csv
2014-02-26_BRAFWTNEGASSAY_FFPEDNA-CRC-1-41_A04.csv
```

ISO8601



PUBLIC SERVICE ANNOUNCEMENT:

OUR DIFFERENT WAYS OF WRITING DATES AS NUMBERS CAN LEAD TO ONLINE CONFUSION. THAT'S WHY IN 1988 ISO SET A GLOBAL STANDARD NUMERIC DATE FORMAT.

THIS IS *THE* CORRECT WAY TO WRITE NUMERIC DATES:

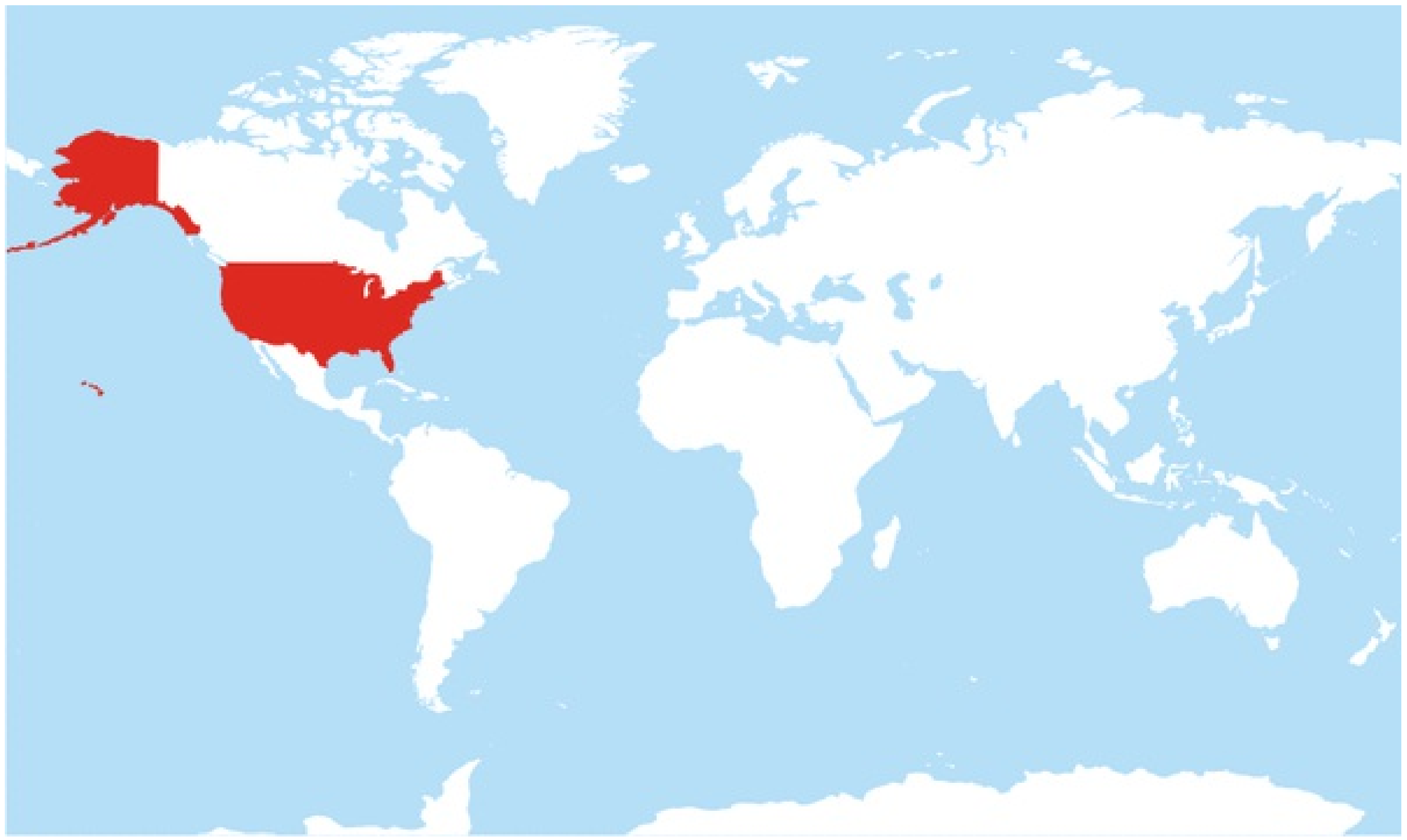
2013-02-27

THE FOLLOWING FORMATS ARE THEREFORE DISCOURAGED:

02/27/2013 02/27/13 27/02/2013 27/02/13
20130227 2013.02.27 27.02.13 27-02-13
27.2.13 2013. II. 27. $^{27}\!/\!_{2}$-13 2013.158904109
MMXIII-II-XXVII MMXIII $\frac{LVII}{CCCLXV}$ 1330300800
((3+3)×(111+1)-1)×3/3-1/3³ 2013
10/11011/1101 02/27/20/13 01237

Source: XKCD #1179

Comprehensive map of all countries that use the MM-DD-YYYY format



Source: https://twitter.com/donohoe/status/597876118688026624.

Left pad other numbers with zeros

```
> ls
01_data-cleaning.R
02_fit-model.R
10_final-figs-for-publication.R
```

If you don’t left pad, you get this:

```
> ls
10_final-figs-for-publication.R
1_data-cleaning.R
2_fit-model.R
```

Organizing your workflow

There is no one formula

that will work for all projects, but use an organization that will allow

- you to come back to the project a year later and resume work fairly quickly
- your collaborators to figure out what you did and decided and what files they need to look at
- works with your tools not against them

Tip 1 - Use Projects / Project Folders

Specifically within RStudio, but also more generally as an organizing principal.

- Use one (master) folder per project
    - Everything related to that project needs to live within that folder. (e.g. data, scripts, etc.)
    - IDEs, git, R sessions are all designed around this principal (don't fight it)
    - Organize related files within your folder (e.g. `data/`, `scripts/`, `figures/`)

Aside - Raw data is sacrosanct

Raw data is foundational to reproducibility and it is critical to have a auditable log of any changes at all times.

Create a folder for it, put it there and never touch it.

BAD:

```
project/
  - data/
```

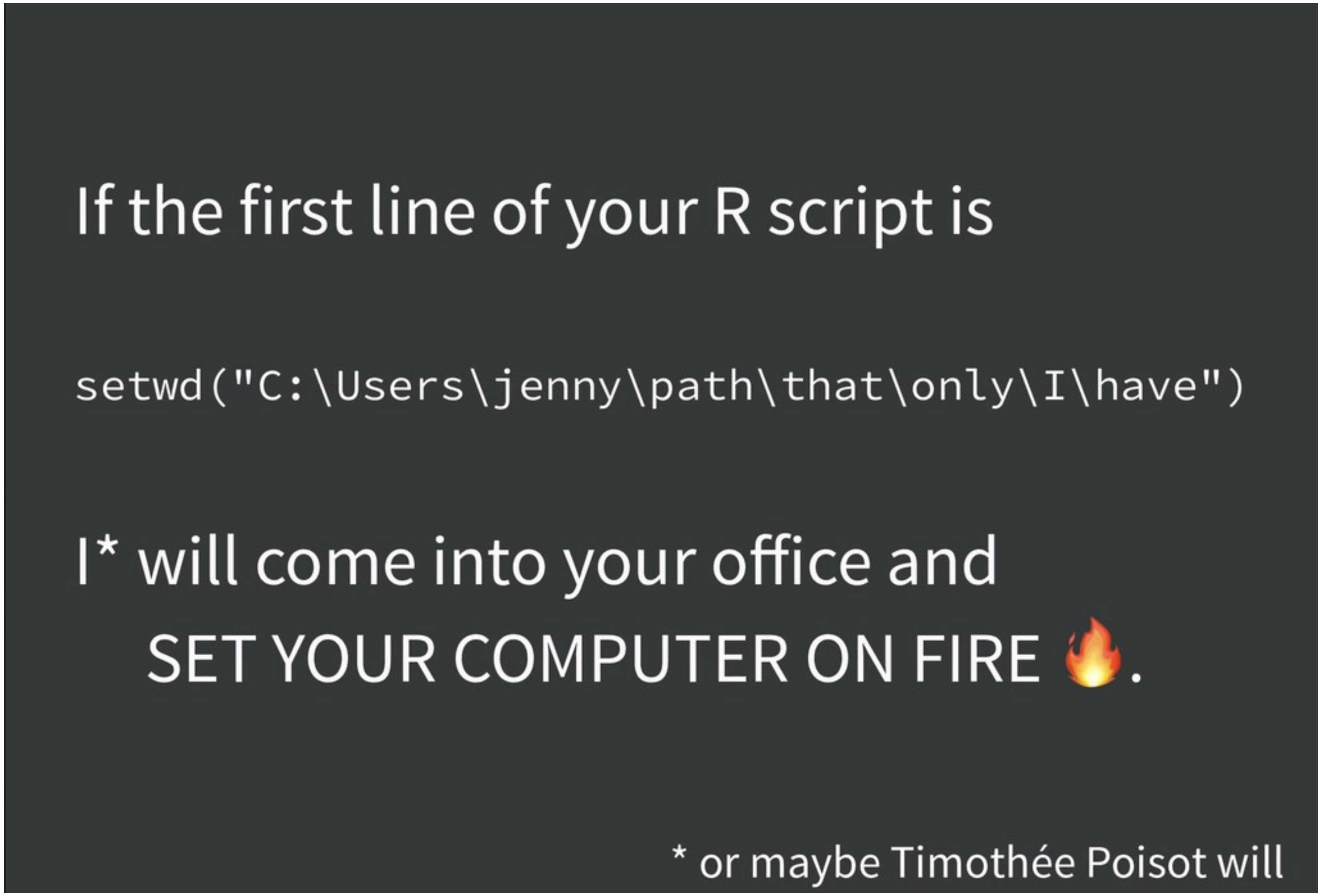
vs.

GOOD

```
project/
  - data-raw/
  - data-clean/
```

```
project/
  - data/
    - raw/
    - clean/
```

Relative vs absolute paths


If the first line of your R script is

```
setwd("C:\Users\jenny\path\that\only\I\have")
```

I* will come into your office and
SET YOUR COMPUTER ON FIRE 🔥.

* or maybe Timothée Poisot will

Source: Jenny Bryan's Zen and the Art of Workflow Maintenance

Working directories

Keeping track of working directories can be painful.

Take a project that looks something like the following:

```
project/
  - project.Rproj
  - data/
    - raw/
      - data.csv
    - clean/
  - script/
    - 01_clean.R
```

When I run `01_clean.R` what is its working directory?

Using here

`here` is a package that tries to simplify the process by identifying the root of your project, `project/` in this case and then providing relative paths from that root directory to everything else in your project.

```
here::here()
## [1] "/home/rundel/Desktop/project"

here::here("data/raw", "data.csv")
## [1] "/home/rundel/Desktop/project/data/raw/data.csv"
```

Tips

- Keep / protect the raw data
- Give yourself less rope
- Avoid monoliths (modularize your code / scripts into logical steps)
- Keep the life cycle of data in mind