Thirteen things I wish I'd known when starting my PhD

Dean Bodenham

24 October 2019

Overview

A collection of tools and tips to make:

- Developing your statistical software
- Publishing your code
- Writing your thesis and papers

easier, more efficient — and even fun.

Towards better Science

Using these tools and approaches will allow one to:

- Become more efficient on a computer
- Write better code
- Create more reproducible research

Disclaimer

- Just my opinion
- I am not an expert
- These are just **suggestions**
- There are lots of other options
- Many of these may seem obvious

I hope you will find **one** suggestion today that is of interest to you

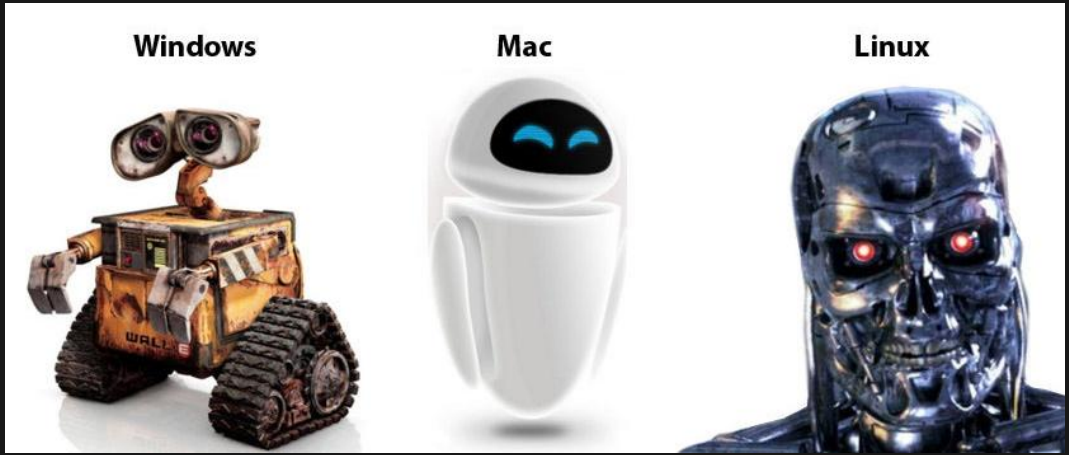
1. Use a Unix-based OS

Two popular distributions:

- Linux (many versions)
- OSX

Everything you see today is free and works with both.

Why use Unix?



source: imgur.com

Why use Unix?

- The power of the command line tools:
  - `ssh`
  - `grep`
  - `find`
  - `wget`
  - `aspell` (LaTeX spellcheck)
  - `top`
- These tools can be 'piped' together
  - e.g. find all files containing word X, made after date Y
- **Shell script**: a single file that runs multiple commands/scripts
- Don't know how to use a tool?
  - `man grep`
- Windows users: Cygwin

Useful example

```
grep --include=*.{Rmd,R} -rnw '.' -e "Linux"
```

```
grep --include=*.{Rmd,R} -rnw '.' -e "Linux"
./thirteenthings.Rmd:309:* Linux (many versions)
./thirteenthings.Rmd:353:grep --include=*.{Rmd,R} -rnw '.' -e "Linux"
./thirteenthings.Rmd:357:grep --include=*.{Rmd,R} -rnw '.' -e "Linux"
./thirteenthings.Rmd:361:## Different flavours of Linux
./thirteenthings.Rmd:484:Linux (Ubuntu):
./thirteenthings.Rmd:534: - Developed by Linus Torvalds (Linux)
./thirteenthings.Rmd:591:Linux:
```

Different flavours of Linux

Last 12 months			Last 6 months		
1	MX Linux	4303▲	1	MX Linux	4926▲
2	Manjaro	3116▼	2	Manjaro	2554▼
3	Mint	2142▬	3	Mint	2045▼
4	elementary	1527▼	4	Debian	1558▲
5	Ubuntu	1434▲	5	Ubuntu	1406▼
6	Debian	1416▲	6	elementary	1285▼
7	Solus	1041▲	7	Solus	1093▲
8	Fedora	999▲	8	Fedora	989▬
9	openSUSE	815▬	9	deepin	856▲
10	Zorin	785▬	10	Zorin	832▬

source: distrowatch.com, October 2019, top 10 of approx. 200

Cinnamon, source: linuxmint.com

Try it out

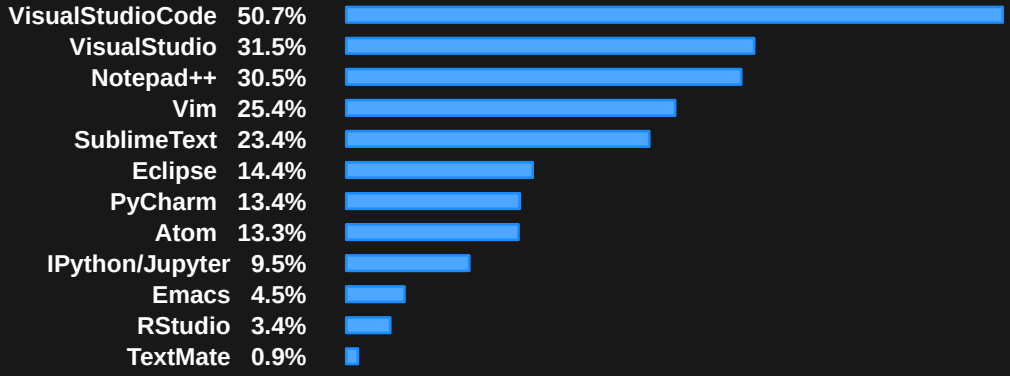
- Live USB - let's you test it out before installing
  - check wifi
  - check sound
  - check video
  - check filesystem
- Virtual machine: VirtualBox

Starting out?

- Consider starting with:
  - Mint (Ubuntu), Cinnamon
  - Manjaro KDE
- Loads of software, users, support

2. Pick a good editor

Most Popular Development Environments



87,317 responses; select all that apply

source: Stackoverflow Developer Survey 2019

Desirable editor features

- Cross platform
- Multi-language support
  - Python, R, C++, etc
  - LaTeX
- If not command line-based editor, become familiar with one
  - e.g. nano, Vim, Emacs

3. Use a terminal multiplexer

Terminal multiplexers

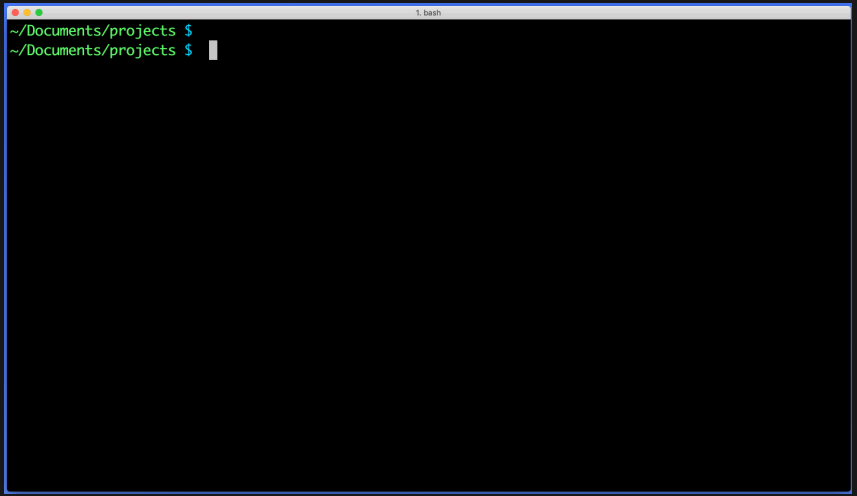
- `screen` (classic, but no longer maintained)
- `tmux` (new and growing in popularity)

`tmux` benefits

- Organise multiple terminals in one screen*
- Easily split windows into panes*
- Continue server sessions*
- Pane shading
- Save sessions after reboots: tmux-resurrect

*also `screen`

tmux demo



4. Use the server

Easily run large-scale simulations

- Your local machine is for **development**
- Queue a job on the server, and let it run overnight
- This is where knowing:
  - terminal commands
  - Vim/Emacs
  - `screen`/`tmux`

  will pay off **A LOT**.
- If no queueing system, try `nohup`
  - Example: `nuhup CMD bg &`, where `CMD` is the command to be run

5. Use Git

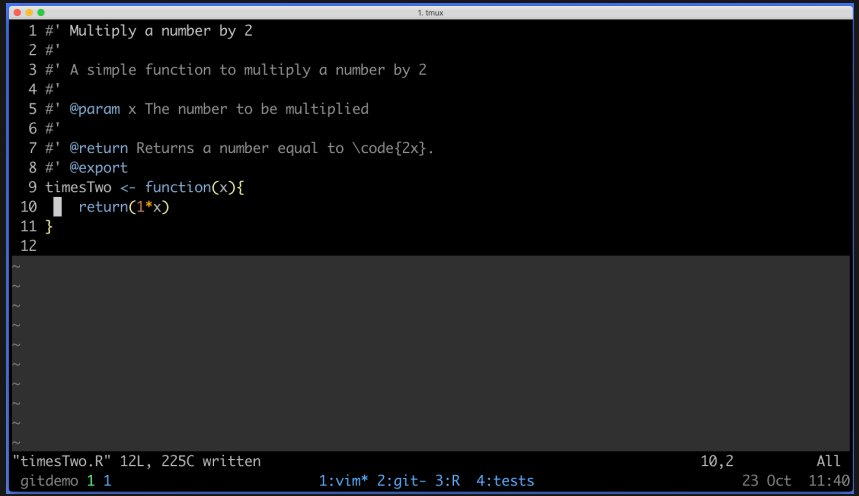
Git: version control

- Ever had code (R, C++, LaTeX...) 'stop working'?
- Git allows you to easily create checkpoints and versions
- Developed by Linus Torvalds (Linux)
- Git is FAST
- Allows you to work offline
- There are other version control systems, but Git is most popular

Git: basic commands

- Quick and easy to get started:
  - `git init`
  - `git add .`
  - `git commit`
- Recovering versions is also easy
  - Three different methods, depending on situation

Git demo


1. tmux
#' Multiply a number by 2
#'
#' A simple function to multiply a number by 2
#'
#' @param x The number to be multiplied
#'
#' @return Returns a number equal to \code{2x}.
#' @export
timesTwo <- function(x){
return(1*x)
}
"timesTwo.R" 12L, 225C written
10,2 All
gitdemo 1 1
1:vim* 2:git- 3:R 4:tests
23 Oct 11:40

Git for backups

- Ever worried about backing up scripts?
- Get a free GitHub or Bitbucket account
  - upload Git repo's with: `git push -u origin master`
- Lost your code or want to sync on a different computer?
  - `git clone`
  - `git pull`

Using Git

- Use Git for version control with:
  - code scripts
  - LaTeX scripts
- Do not use Git with:
  - data files
- Can use it to save images, but be careful. Try `git-lfs`
- Create a `.gitignore` file to ignore certain file(type)s
- Can be used for team-based projects

6. Create R packages, not scripts

R packages

- Binds your scripts together
- Forces you to comment/document your code
- Easily integrate **tests** (more later)
- Getting started: `devtools` and `roxygen2` packages
- Check out Hadley Wickham's online book
- **Note:** similar in Python

7. Comment your code

Two types of comments

- Code description (documentation)
  - show users how to use functions
  - input, output, what the function does
  - part of `roxygen2` commands
- Algorithmic description
  - for others (and yourself!) to understand the code
  - inside the functions
  - explain why you did something in a certain way, e.g.
    - `# using a matrix here because...`

Seems annoying and time-consuming - but the payoff is **worth it**

(ever come back to code after a couple of months...?)

Example of roxygen2 comments

```
#' Multiply a number by 2
#'
#' A simple function to multiply a number by 2
#'
#' @param x The number to be multiplied
#'
#' @return Returns a number equal to \code{2*x}.
#' @export
timesTwo <- function(x){
    return(2*x)
}
```

8. Write unit tests for your code

Test your code

- **Unit tests** are pieces of code that test your functions
- Make sure your code **works** as it should
- Avoid breaking your (previously working) code
- Forces you to break your code down into logical chunks
- Can be useful for debugging
- Longer in the short term, but **shorter in the long term**
- Check out the `testthat` R package
  - Makes testing quick and easy

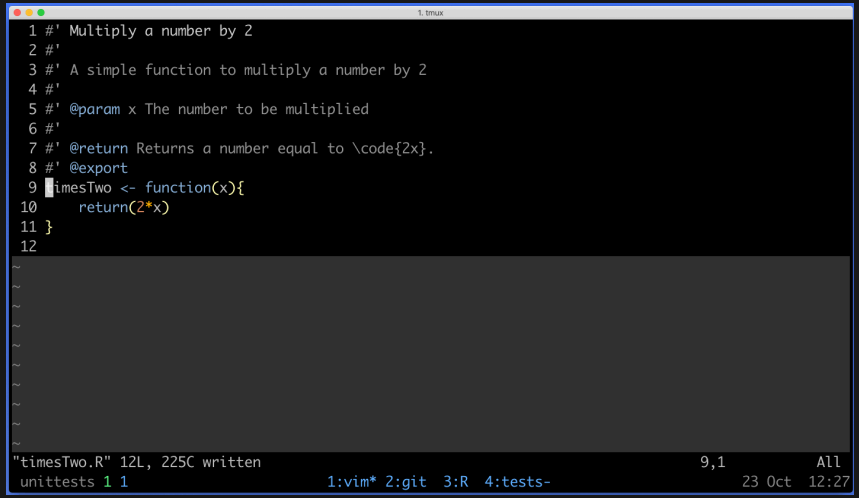
Unit testing in R

```
timesTwo <- function(x){
    return(2*x)
}
```

Unit testing in R

```
test_that("timesTwo tests", {
    expect_equal(timesTwo(1), 2)
    expect_equal(timesTwo(2), 4)
})
```

Unit testing in R



Start testing

- Hadley Wickham's tutorial
- R journal article

Note: Similar framework in Python

9. Speed up your code with Rcpp and Cython

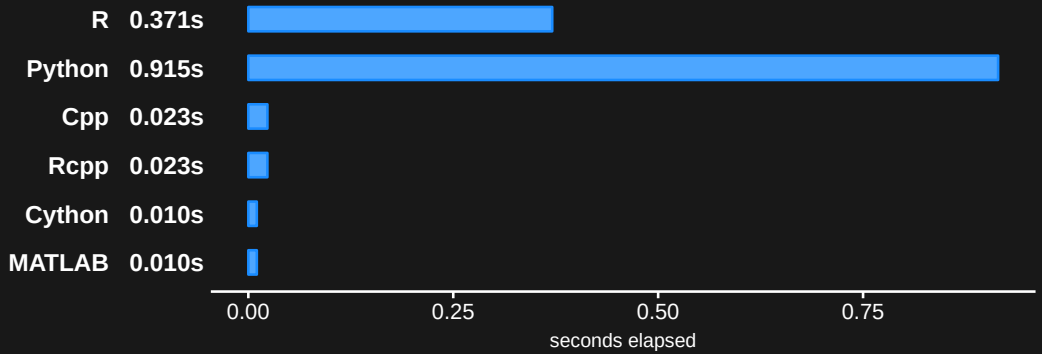
Bring the the speed of C++ to R and Python

- R is often criticised for being slow
  - bad at `for` loops
- C++ is **fast**, but lacks many of R's functions
- `Rcpp` is an R package that links R and C++
- Cython is a framework for running C++ code from Python

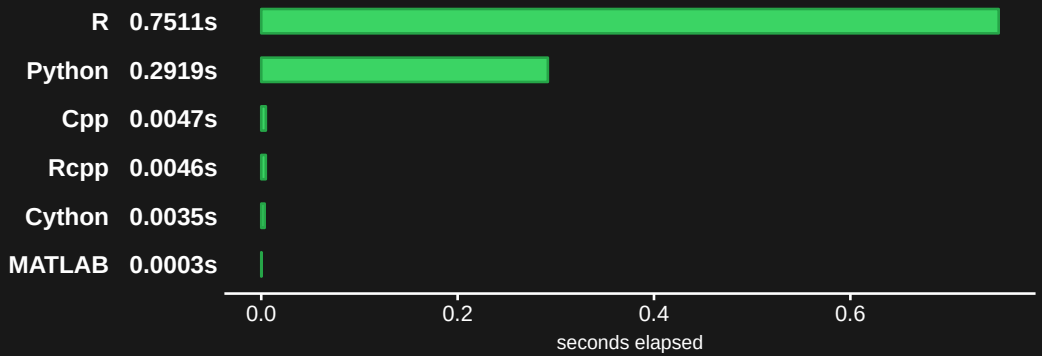
Bring the the speed of C++ to R and Python

- Code for benchmarks from julialang.org
- To reproduce these results, see https://github.com/deanbodenham/benchmarks_rpycpp
- Using `microbenchmark` package
- For linear algebra in C++, using Armadillo package
- MATLAB 2019a is used
- Check out the Rcpp gallery

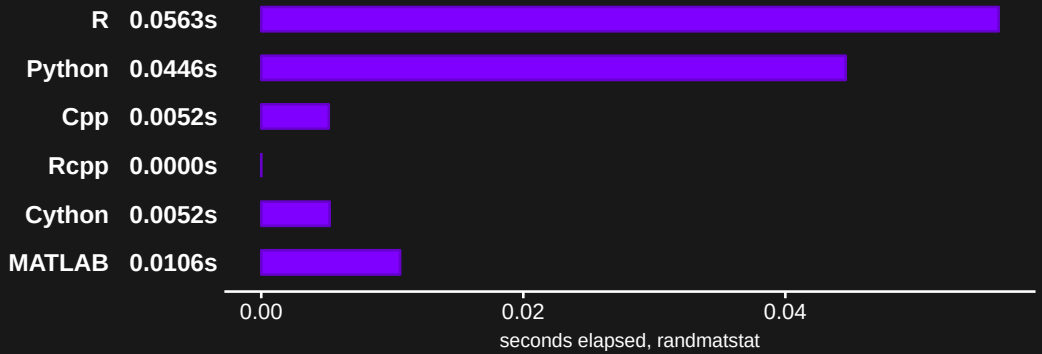
Pisum, nested for loop



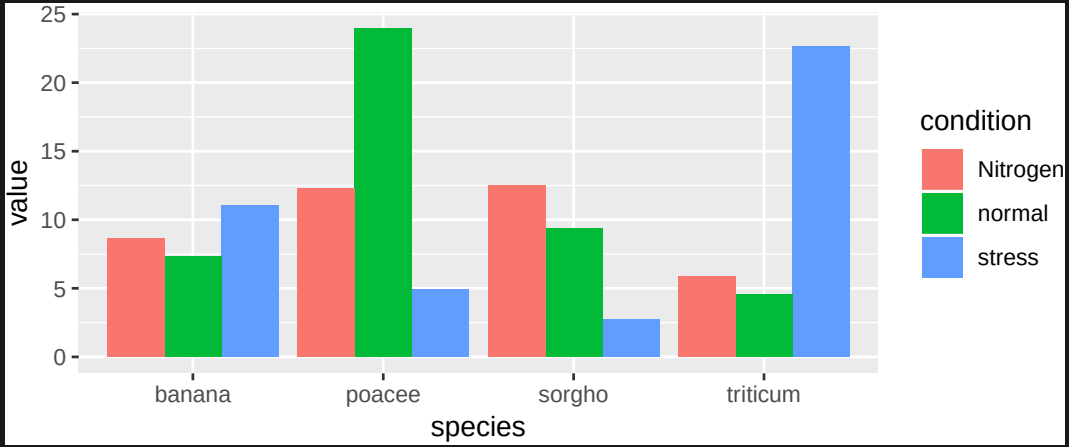
Recursive Fibonnaci

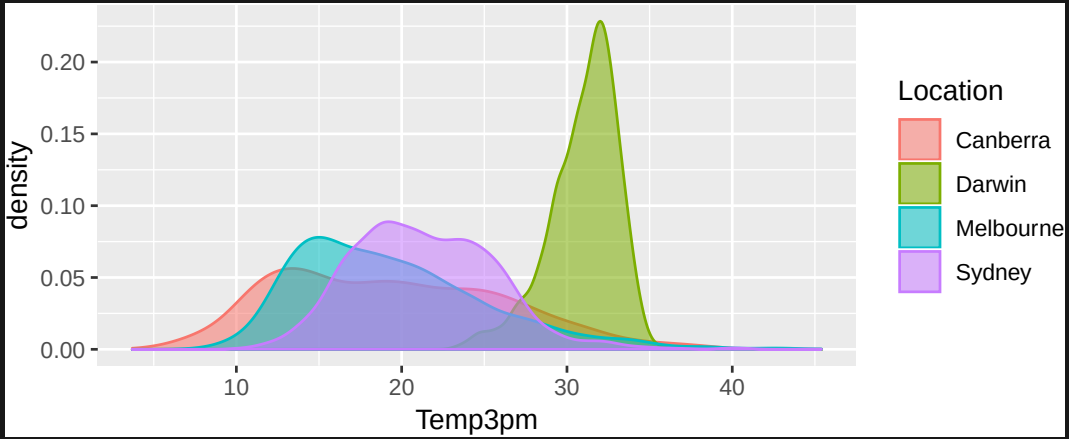


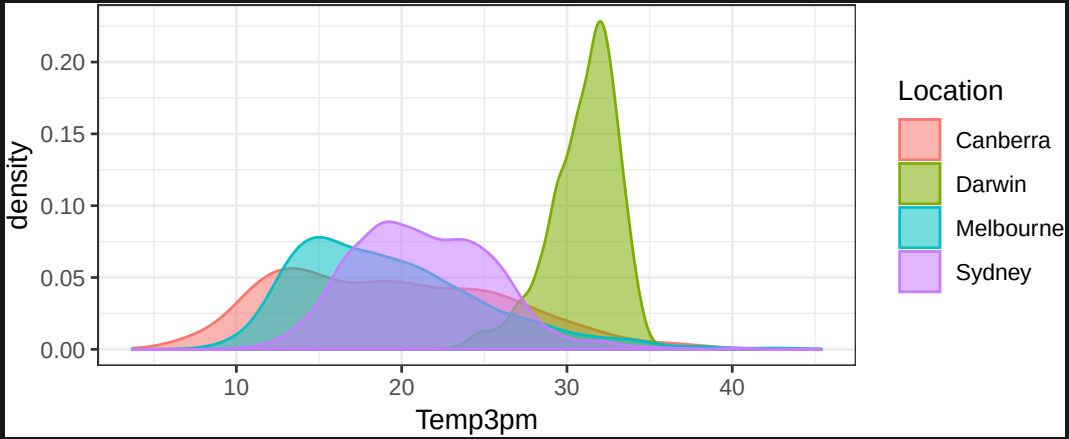
Random matrix statistics

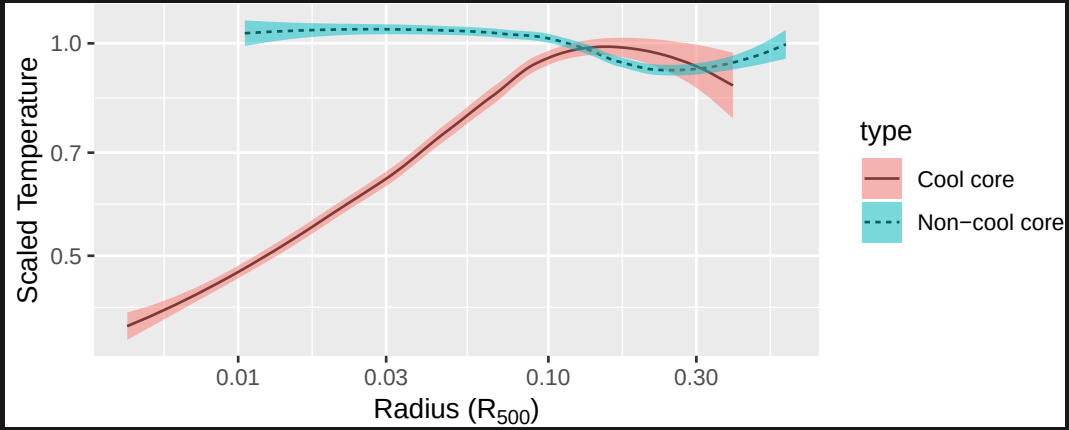


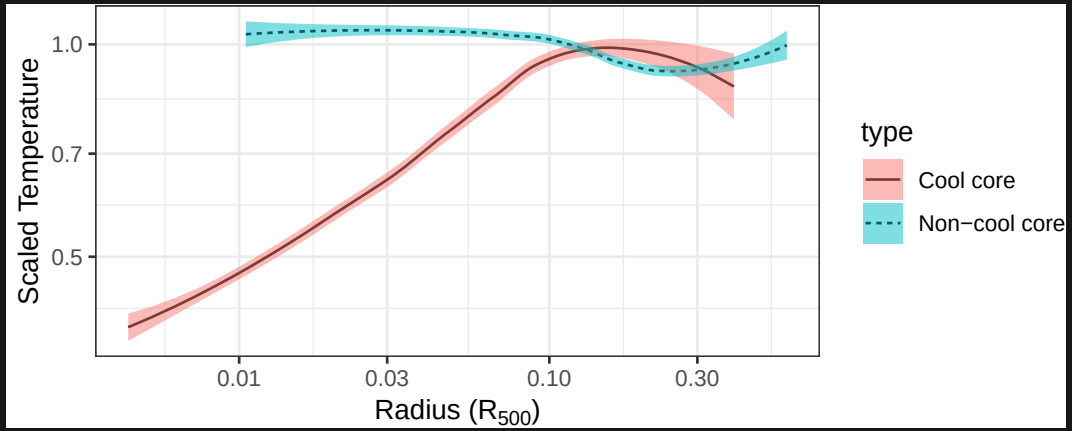
10. For nice plots, use `ggplot2`

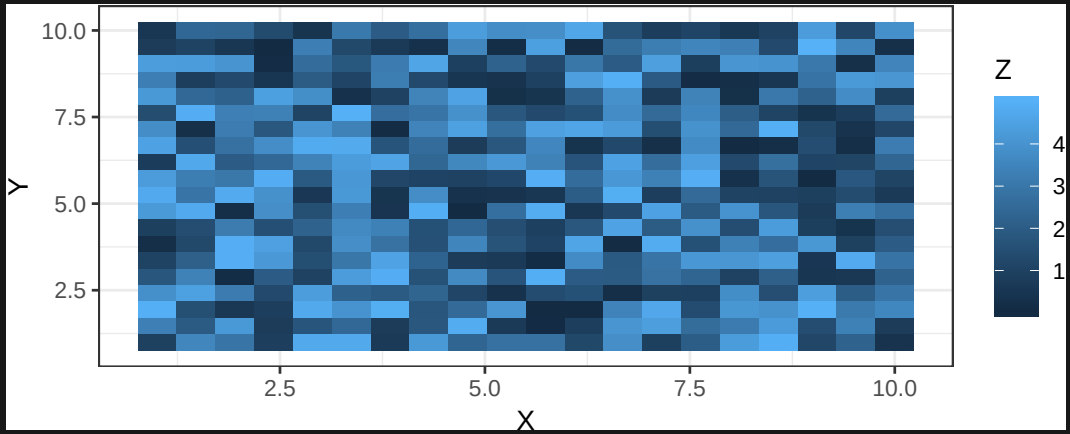

25
20
15
10
5
0
value
banana
poacee
sorgho
triticum
species
condition
Nitrogen
normal
stress

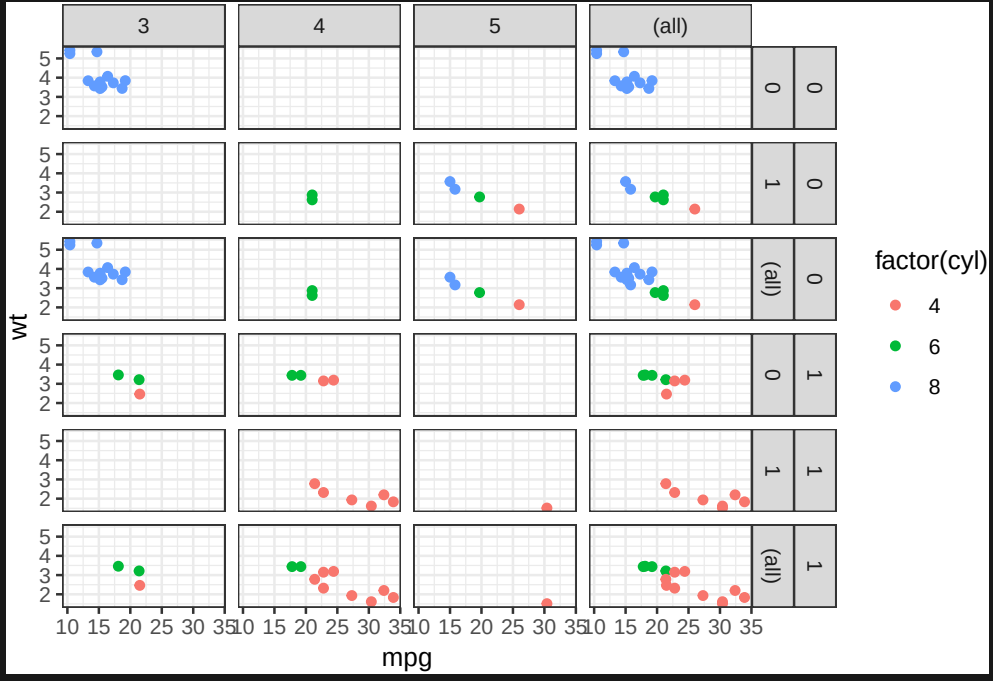

density
0.20
0.15
0.10
0.05
0.00
10
20
30
40
Temp3pm
Location
Canberra
Darwin
Melbourne
Sydney


density
0.20
0.15
0.10
0.05
0.00
10
20
30
40
Temp3pm
Location
Canberra
Darwin
Melbourne
Sydney


Scaled Temperature
1.0
0.7
0.5
0.01
0.03
0.10
0.30
Radius (R_{500})
type
Cool core
Non−cool core


Scaled Temperature
1.0
0.7
0.5
0.01
0.03
0.10
0.30
Radius (R_{500})
type
Cool core
Non−cool core


10.0
7.5
5.0
2.5
Y
2.5
5.0
7.5
10.0
X
Z
4
3
2
1


3
4
5
(all)
0
1
(all)
wt
mpg
factor(cyl)
4
6
8

11. Reproducible results and plots

Reproducible results

- Save every experiment with key results
  - May need to run again/tweak at a later stage
  - Combine with Git
- All results should appear by running **one command**
- Write a README for each experiment

Reproducible figures

- Less obvious, but just as important
- Use a two-step approach:
  - One script to generate the data
  - Another script to generate the figure from the data

Putting it all together

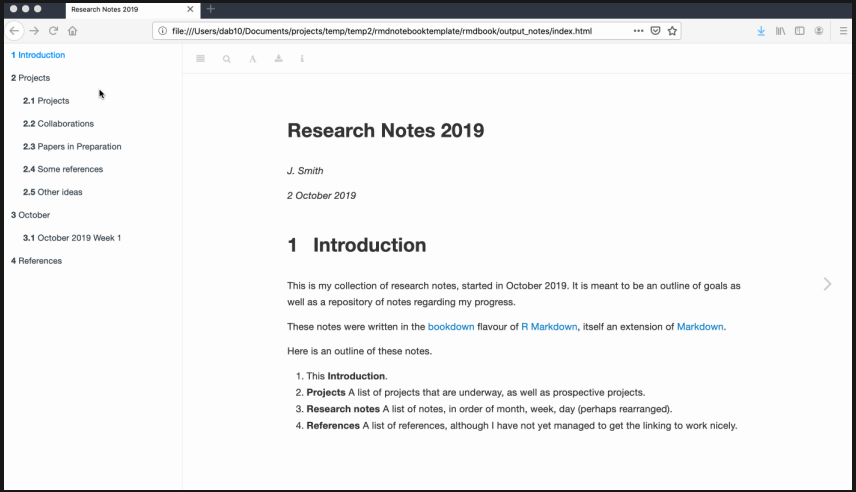
- Step 1: Run code to generate results (long time, server)
  - results saved in csv/RData file.
- Step 2: Script to create plot from data file (short time, laptop)
  - e.g. pdf, probably many revisions
- Step 3: Shell script to both generate results and create plot
  - Test on laptop with short simulation
  - Used in the future when needing to reproduce results
  - Make a note of the Git commit/package version number for generating data
- Example of the general advice: “write as you go”

12. R Markdown Journal

Keep track of research goals and results

- As a scientist, keep track of:
  - research efforts
  - results
  - things that worked
  - things that didn't work
  - goals
- Pen/paper notebooks are great
- R Markdown provides a framework for a digital version
- Allows output to html, pdf and other formats

Rmd notebook output



Template

A template I use can be found at:

https://github.com/deanbodenham/rmd_notebook_template

13. Use a reference manager

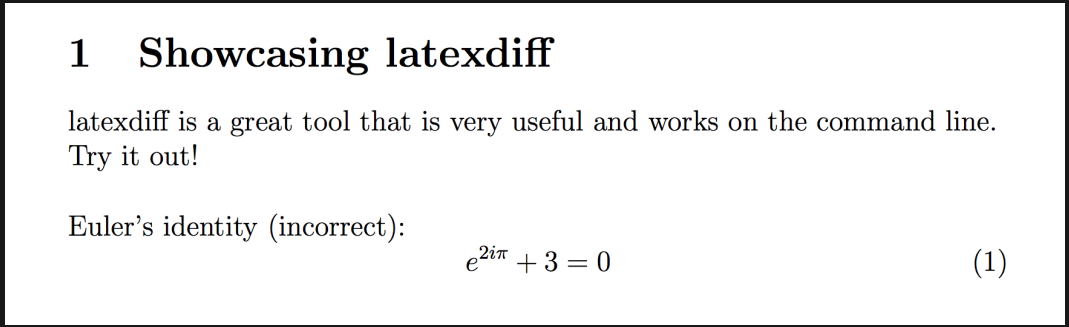
Keep track of papers and citations

- Examples:
  - Mendeley
  - Zotero
- Organise your papers
  - What you've already read
  - What you've still got to read
  - Notes on papers (later searchable)
- When citing paper/reference in your work, include comment with page/line number

14. git-latexdiff

Check differences between tex files

- Originally, `latexdiff` works with two files as input
- `git-latexdiff` works with **versions of the same file** with Git



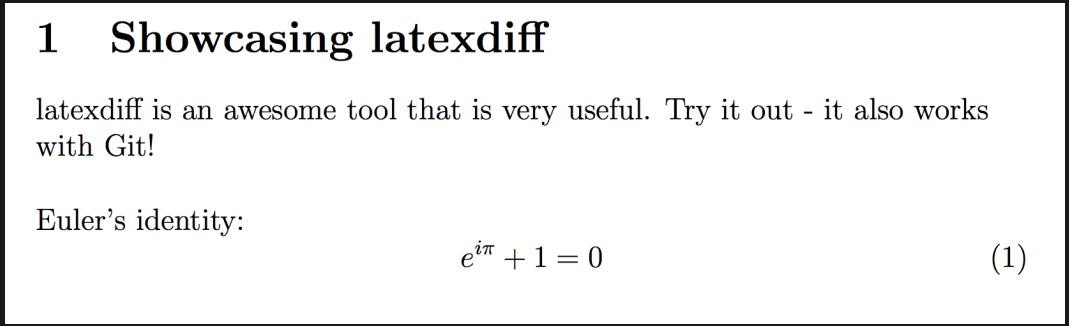
1 Showcasing latexdiff

latexdiff is a great tool that is very useful and works on the command line. Try it out!

Euler's identity (incorrect):

$$e^{2i\pi} + 3 = 0 \tag{1}$$

Figure 1: Version 1



1 Showcasing latexdiff

latexdiff is an awesome tool that is very useful. Try it out - it also works with Git!

Euler's identity:

$$e^{i\pi} + 1 = 0 \tag{1}$$

Figure 2: Version 2

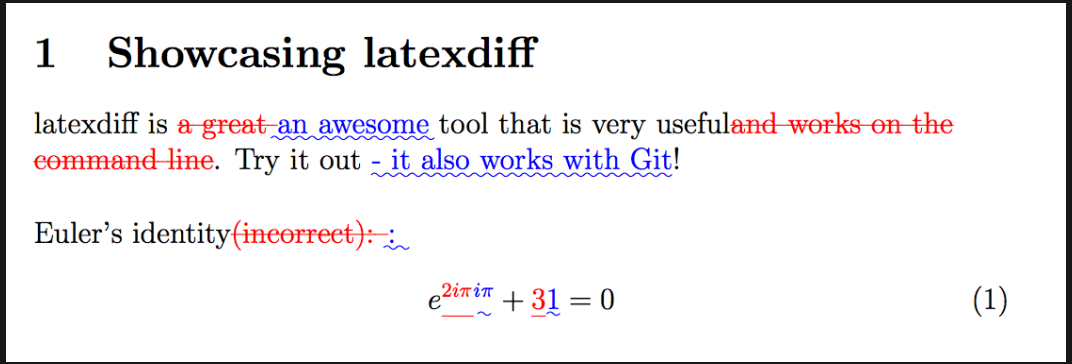


Figure 3: Differences with `latexdiff`

Easy Command:

- Compare current version and two versions back:

```
git latexdiff HEAD~2 --main main.tex
```

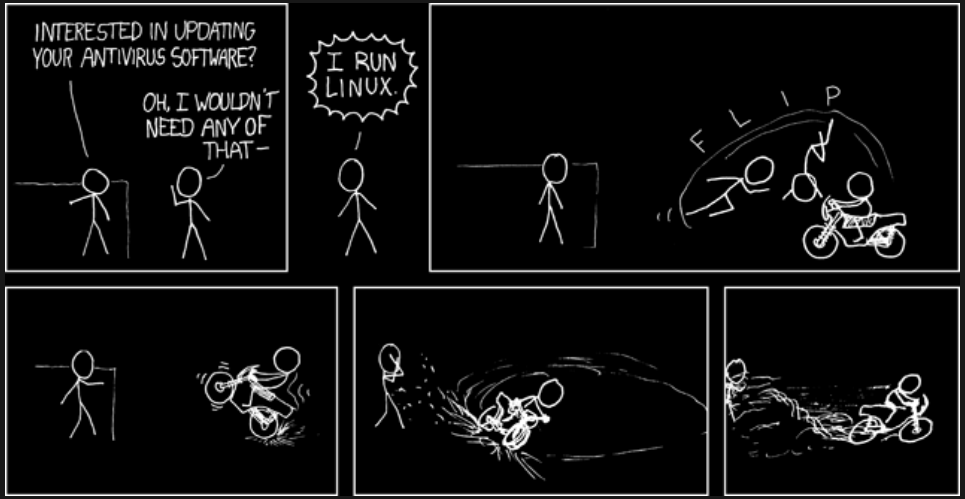
- Full credit to:
  - Frederik Tilmann, creator of `latexdiff`
  - Matthieu Moy, creator of `git-latexdiff`

Summary

1. Use a Unix-based OS
2. Choose a good editor
3. Use tmux for terminal management
4. Use the server for running experiments
5. Use Git for version control (mention `git-lfs`)
6. Create R packages, rather than scripts
7. Comment your code
8. Write unit tests for your code
9. Speed up your code with Rcpp and Cython
10. For nice plots `ggplot2`
11. Reproducible results and plots
12. Rmarkdown research journal
13. Use a reference manager
14. `git-latexdiff`

15. Things I Learnt

Questions?



source: http://xkcd.com/272/

Slides at https://github.com/deanbodenham/tenthings

1. Use a Unix-based OS
2. Choose a good editor
3. Use tmux for terminal management
4. Use the server for running experiments
5. Use Git for version control (mention `git-lfs`)
6. Create R packages, rather than scripts
7. Comment your code
8. Write unit tests for your code
9. Speed up your code with Rcpp and Cython
10. For nice plots `ggplot2`
11. Reproducible results and plots
12. Rmarkdown research journal
13. Use a reference manager
14. `git-latexdiff`