An introduction to R and version control

https://bradduthie.github.com/talks/intro_to_R.pdf

Brad Duthie (alexander.duthie@stir.ac.uk)

17 November 2022

Tentative schedule (09:30-12:30)

Introduction to R (09:30-11:30)

- Getting started
- Useful functions
- Custom functions
- Using loops

Introduction to version control (11:30-12:30)

- Getting started
- Using GitKraken

How does coding (specifically R) help me as a researcher?

Practical advantages of learning and using R

- R software is entirely free and open source

How does coding (specifically R) help me as a researcher?

Practical advantages of learning and using R

- R software is entirely free and open source
- Thousands R packages for specific data needs

How does coding (specifically R) help me as a researcher?

Practical advantages of learning and using R

- R software is entirely free and open source
- Thousands R packages for specific data needs
- Standard programming language for statistics

How does coding (specifically R) help me as a researcher?

Practical advantages of learning and using R

- R software is entirely free and open source
- Thousands R packages for specific data needs
- Standard programming language for statistics
- Write papers, slides, and apps in Rmarkdown

How does coding (specifically R) help me as a researcher?

Practical advantages of learning and using R

- R software is entirely free and open source
- Thousands R packages for specific data needs
- Standard programming language for statistics
- Write papers, slides, and apps in Rmarkdown
- **Write your own solutions to problems**
- **Write your own statistical analyses**
- **Create your own plots**

How does coding (specifically R) help me as a researcher?

Practical advantages of learning and using R

- R software is entirely free and open source
- Thousands R packages for specific data needs
- Standard programming language for statistics
- Write papers, slides, and apps in Rmarkdown
- **Write your own solutions to problems**
- **Write your own statistical analyses**
- **Create your own plots**

A lot of coding is Googling solutions to get the code to work.

Write your own solutions to data organisation problems

```
##           SPEI Year Tree_ID      BAI cumul_mn
## 1  0.34325052 1966  FR6201 645.3972       NA
## 2 -1.35933830 1967  FR6201 470.0363       NA
## 3  0.49415034 1968  FR6201 830.5755       NA
## 4 -1.38069918 1969  FR6201 414.0594       NA
## 5  0.79613295 1970  FR6201 977.4877       NA
## 6 -0.06371012 1971  FR6201 809.8834       NA
```

Calc. the cumul_mn for the BAI col. every year for each tree[1]:

[1] Thanks to Tom Ovenden for letting me use this example.

Write your own solutions to data organisation problems

```
##           SPEI Year Tree_ID      BAI cumul_mn
## 1  0.34325052 1966  FR6201 645.3972       NA
## 2 -1.35933830 1967  FR6201 470.0363       NA
## 3  0.49415034 1968  FR6201 830.5755       NA
## 4 -1.38069918 1969  FR6201 414.0594       NA
## 5  0.79613295 1970  FR6201 977.4877       NA
## 6 -0.06371012 1971  FR6201 809.8834       NA
```

Calc. the cumul_mn for the BAI col. every year for each tree[1]:

- Do not include *current* BAI record in cumul_mn calc
- If the `Year` `SPEI` is >1 or -1, never include the BAI value
- If the `Year - 1` `SPEI` is >1 or <-1, never include the BAI value
- If the `Year - 2` `SPEI` is >1.5 or <-1.5, never include the BAI value
- If the `Year - 3` `SPEI` is >2 or <-2, never include the BAI value

[1] Thanks to Tom Ovenden for letting me use this example.

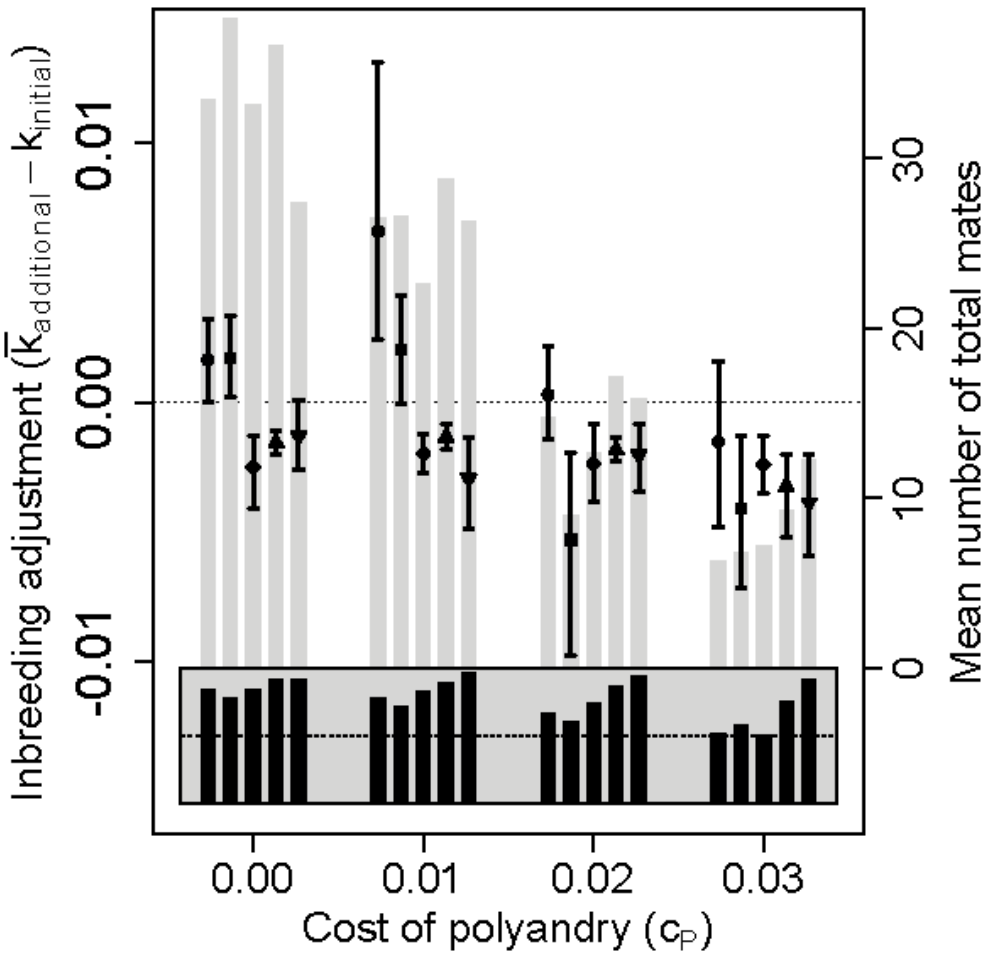
Write your own solutions to data organisation problems

Solution took 65 lines of code written in 3 custom functions.

```
tree_cumulative_mean <- function(dat){
    trees     <- unique(dat$Tree_ID);
    new_table <- NULL;
    for(tree in trees){
        sub_dat    <- dat[dat$Tree_ID == tree,];
        tree_cumul <- get_cumulative(tree = sub_dat);
        new_tree   <- update_cumul(tree = sub_dat,
                                   vec  = tree_cumul);
        new_table  <- rbind(new_table, new_tree);
    }
    return(new_table);
}
```

The above is the 'outermost' function.

Create your own plots



Custom built plot with R code for an individual-based model.

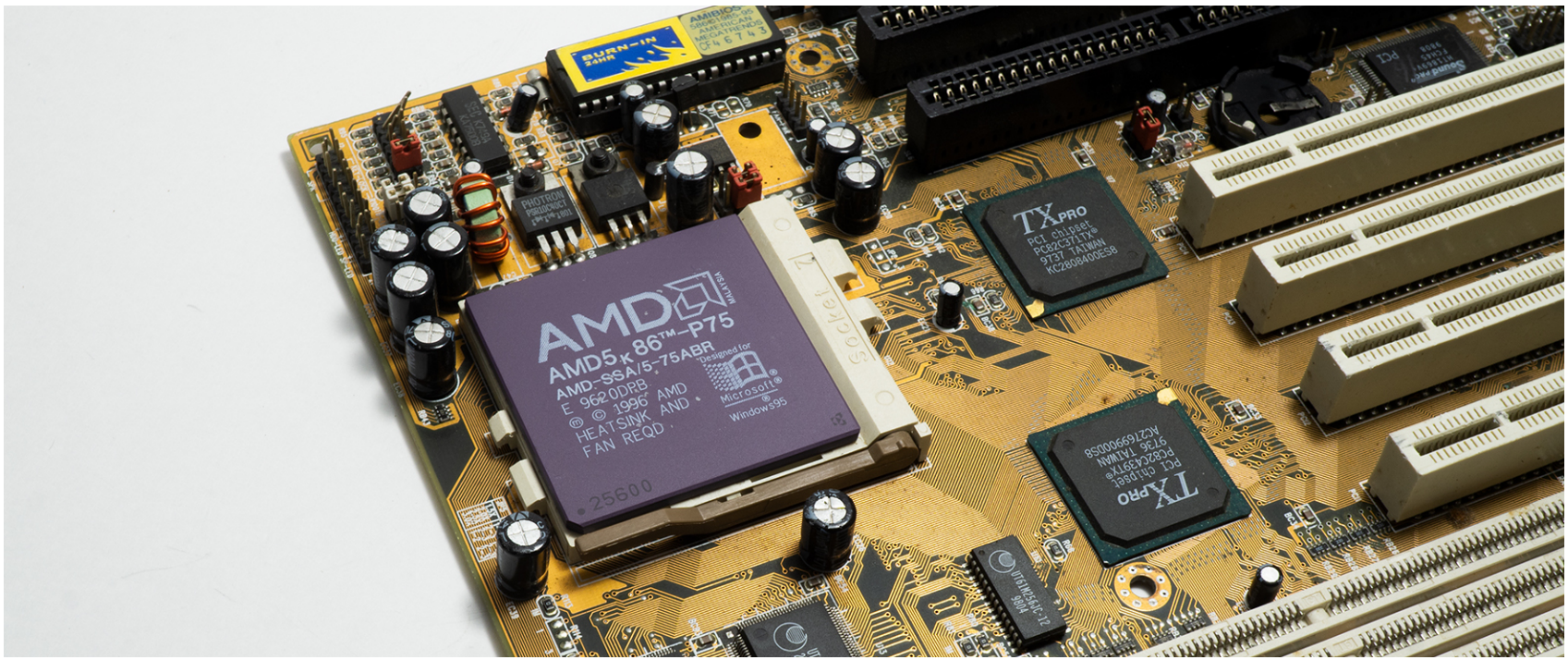
[1] Duthie AB, et al. (2016) *Evolution* 70(9), 1927–1943.

How does code actually work?

- Programmers work with **source code** (e.g., R, python)
- Computers execute **machine language** (binary)
- To get from source code to machine language, we can *compile* code or *interpret* it.

The role of the processor

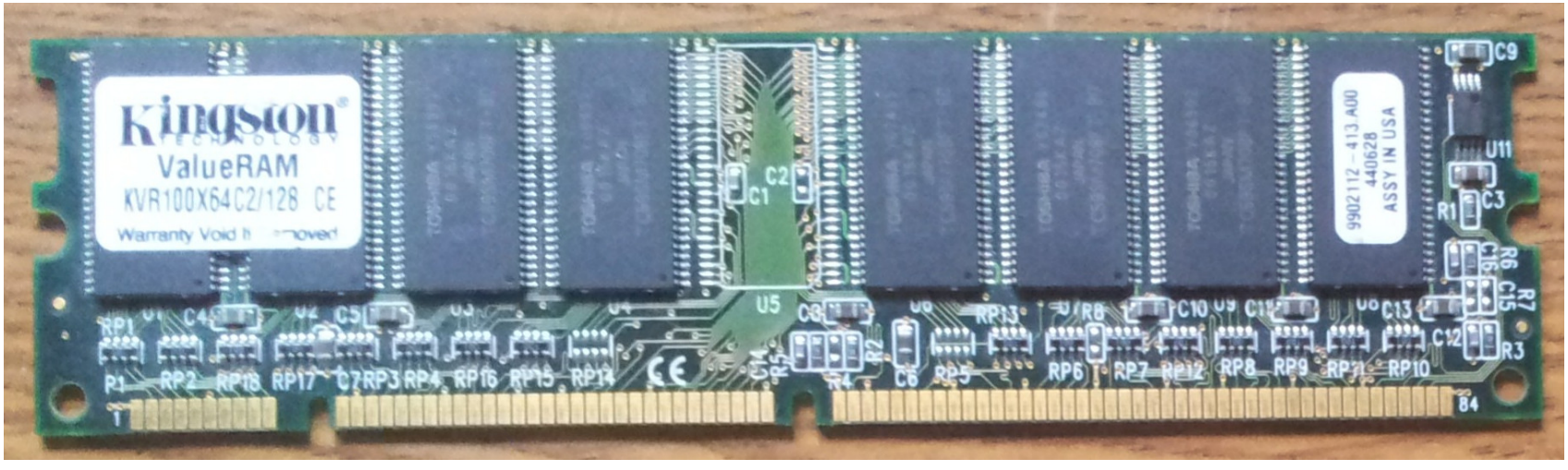
- Machine code: list of instructions written in binary (1s & 0s)
- Binary instructions sent to the *Central Processing Unit (CPU)*
  - CPUs read & write to memory, and do maths (that's it)
  - Instructions tell CPU to read & write information to memory

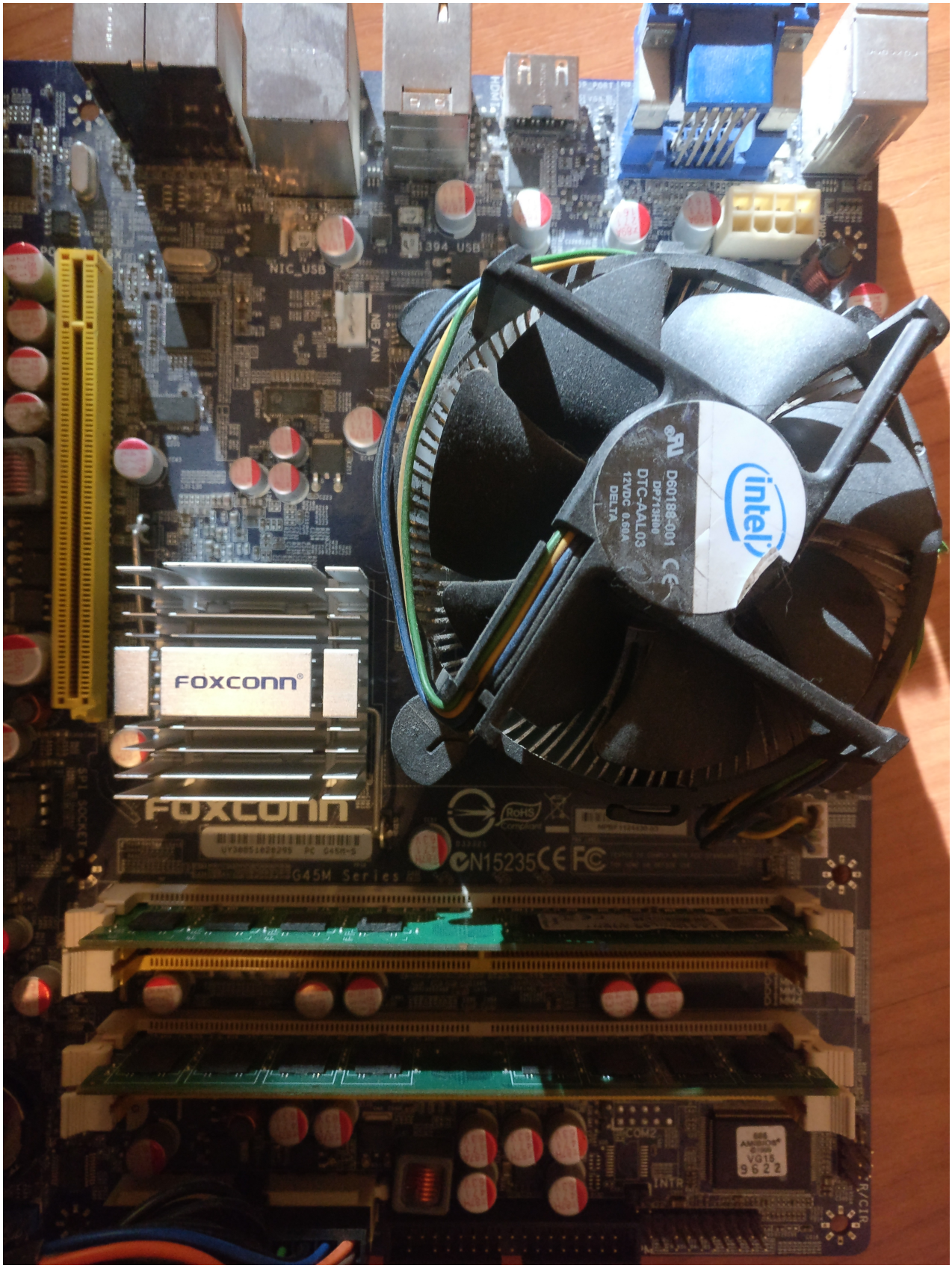


[1] **Image:** Public domain

The role of memory

- Random-access memory (RAM, or just ‘memory’) is separate from the CPU, and holds data that can be read and changed
  - Memory exists as binary digits (‘bits’) of ones and zeros
  - Bits are grouped in chunks of eight to make one ‘byte’: *Unit of data storage large enough to hold any basic character*.




NIC_USB
1394_USB
NB_FAN
FOXCONN
FOXCONN
G45M Series
intel
D60188-001
DF713RH0
DTC-AAL03
12VDC 0.60A
DELTA
RoHS
N15235
COM2
INTR

Getting started in the R console

Relevant links

- Installing R (https://www.r-project.org)
- Installing Rstudio (https://posit.co/downloads/)
- Use Rstudio cloud (https://rstudio.cloud)
- Guided learning (https://swirlstats.com/)

Switch to notes to practice:

- Calculations in the R console
- Assigning variables
- Using Rscripts to run code

https://bradduthie.github.io/data/Bumpus_data.csv

Functions in R

Functions outwith base R available in packages

- Comprehensive R Archive Network includes 18000+ packages
- Packages include specialised functions
- Access with ‘install.packages‘ and ‘library‘

```
install.packages("ggplot2");
library("ggplot2");
ggplot(data = dat, mapping = aes(x = wgt, y = totlen))
       + geom_point();
```

Custom functions can be written in R too with the `function` function.

A custom function in R

Convert from Fahrenheit to Celsius

```
F_to_C <- function(F_temp){
    C_temp <- (F_temp - 32) * 5/9;
    return(C_temp);
}
```

Highlight the whole function and run it, then you can use it.

```
F_to_C(F_temp = 70);
```

```
## [1] 21.11111
```

Now write a custom function for C to F!

Loops in R

A loop repeats the same set of instructions (i.e., 'code') across a particular set of conditions

Loops in R

A loop repeats the same set of instructions (i.e., 'code') across a particular set of conditions

Suppose you want to print the following sequence:

$1, \frac{1}{2}, 3, \frac{1}{4}, \ldots, 999, \frac{1}{1000}$

Loops in R

A loop repeats the same set of instructions (i.e., 'code') across a particular set of conditions

Suppose you want to print the following sequence:

$1, \frac{1}{2}, 3, \frac{1}{4}, \ldots, 999, \frac{1}{1000}$

How would you do it in R (without a loop)?

Loops in R

A loop repeats the same set of instructions (i.e., 'code') across a particular set of conditions

Suppose you want to print the following sequence:

$1, \frac{1}{2}, 3, \frac{1}{4}, \ldots, 999, \frac{1}{1000}$

How would you do it in R (without a loop)?

How would you explain what you want to do (verbally)?

Loops in R

A loop repeats the same set of instructions (i.e., 'code') across a particular set of conditions

Suppose you want to print the following sequence:

$1, \frac{1}{2}, 3, \frac{1}{4}, \ldots, 999, \frac{1}{1000}$

How would you do it in R (without a loop)?

How would you explain what you want to do (verbally)?

1. For each integer from 1 to 1000
2. If the number is odd, print it
3. If the number is even, divide by the number then print it
4. Stop when when finished printing

What is a loop?

A loop repeats the same set of instructions (i.e., 'code') across a particular set of conditions

Suppose you want to print the following sequence:

$1, \frac{1}{2}, 3, \frac{1}{4}, \ldots, 999, \frac{1}{1000}$

How would you do it in R (without a loop)?

How would you explain what you want to do (verbally)?

- For $x = 1, 2, 3, ..., 999, 1000$
  - Check if x is even
  - If x is not even, then print x
  - If x is even, then print $1/x$
- Stop when all x values have been considered

Using a for loop in R

```
for(x in 1:1000){          # The loop starts here



    # Do everything within these brackets,
    #     in the order set by 1:1000
    #     i.e., for x = 1, then x = 2,
    #     then x = 3, ..., then x = 1000

    # Finish the loop only after 'x' has
    #     substituted for each value



} # The loop ends here
```

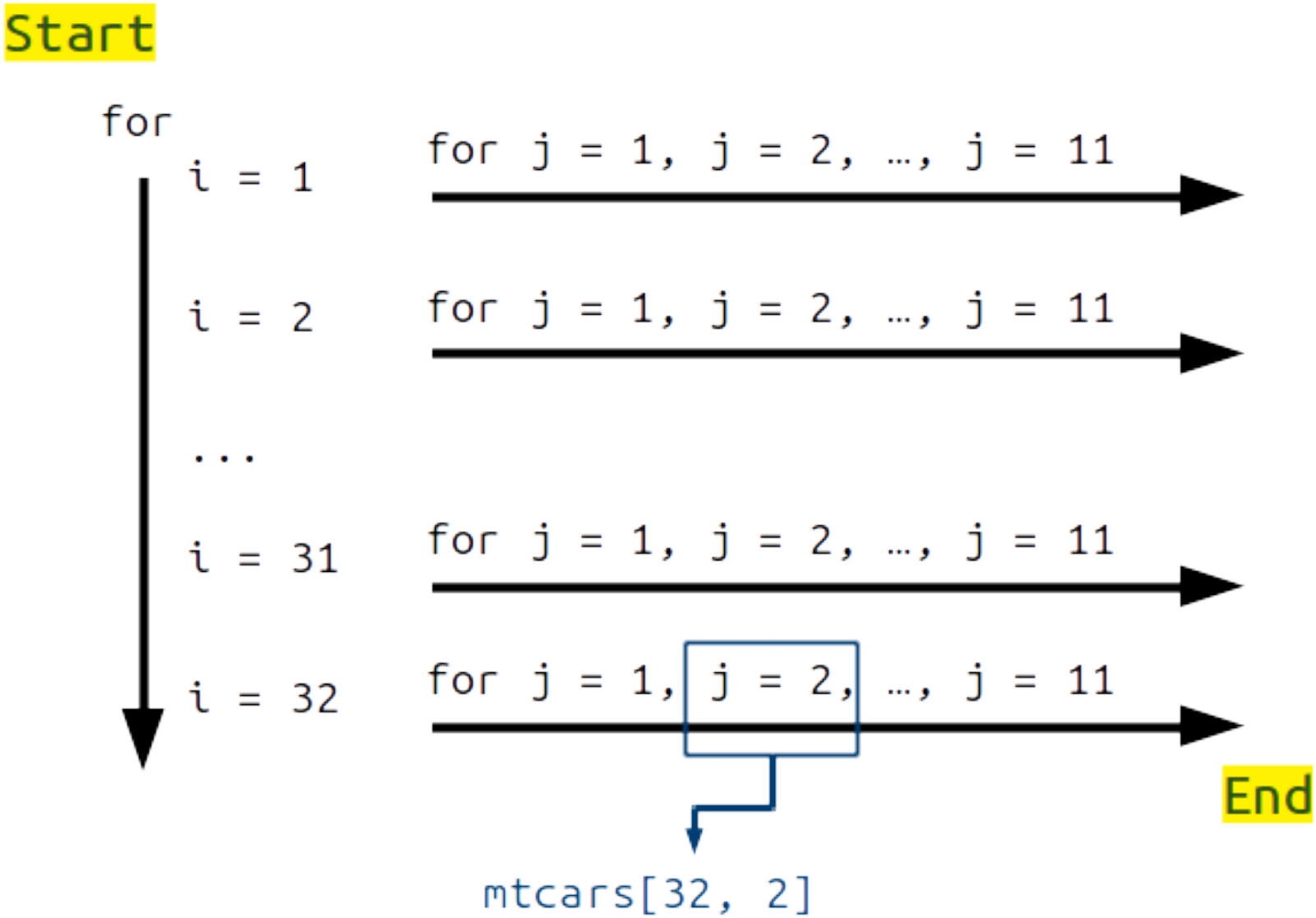
Using a for loop in R

```
for(x in 1:1000){              # The loop starts here

    is_odd <- TRUE;            # First assume'x' is odd
    if(x %% 2 == 0){           # If 'x' is not odd
        is_odd <- FALSE;       # Set to false
    }                          # Now know if 'x' is odd

    if(is_odd == TRUE){        # If 'x' is odd,
        print(x);              # then print 'x'
    }else{                     # Else it is even,
        print(1/x);            # so print 1/x
    }
} # The loop ends here
```

Loops can be inside other loops

```
data(mtcars) # Read in R table of data about cars
rows <- dim(mtcars)[1]; # Get total mtcars rows
cols <- dim(mtcars)[2]; # Get total mtcars columns
for(i in 1:rows){               # for each row
    for(j in 1:cols){           # for each column
        print(mtcars[i, j]); # print the value
    }
}
```

Loops can be inside other loops



While loops in R

Same idea as a `for` loop, but different termination condition

```
counter <- 200; # Set a counter outside the loop
while(counter > 0){ # Keep looping while counter > 0

    print(counter);

    counter <- counter - 1; # Avoid infinite loop

} # The loop ends here
```

Now practice some loops using the notes!

Using version control in R and beyond

- Understand what version control is and how it can be integrated into your work flow

Using version control in R and beyond

- Understand what version control is and how it can be integrated into your work flow
- Focus on practical skills for research
  - Learn and reinforce knowledge on how to use **key skills** effectively
  - Focus on GitHub and GitKraken software

Using version control in R and beyond

- Understand what version control is and how it can be integrated into your work flow
- Focus on practical skills for research
  - Learn and reinforce knowledge on how to use **key skills** effectively
  - Focus on GitHub and GitKraken software
- Hands-on practice setting up and using version control in your own work with accompanying notes for guidance

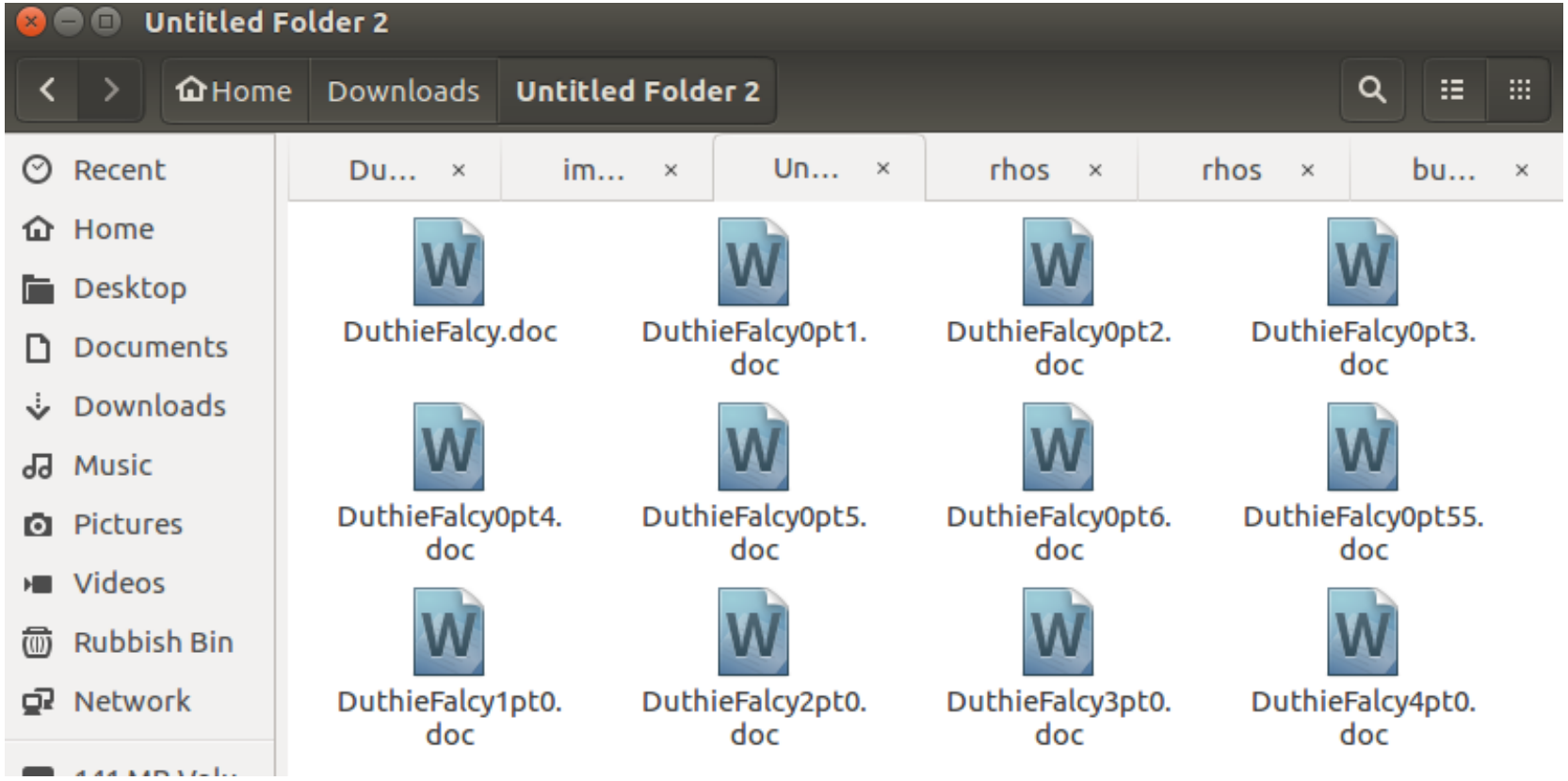
---

https://bradduthie.github.io/notes/vc_notes.html

Rough outline of version control workshop

1. What is version control, and why use it?
2. Getting set up – good file management
3. The GitKraken interface and simple commits
4. Setting up GitHub, pushing and pulling
5. Branching using GitKraken
6. Merging and merge conflicts
7. Forking and cloning using GitHub
8. Independent work using version control

What is version control, and why use it?



What version control software does

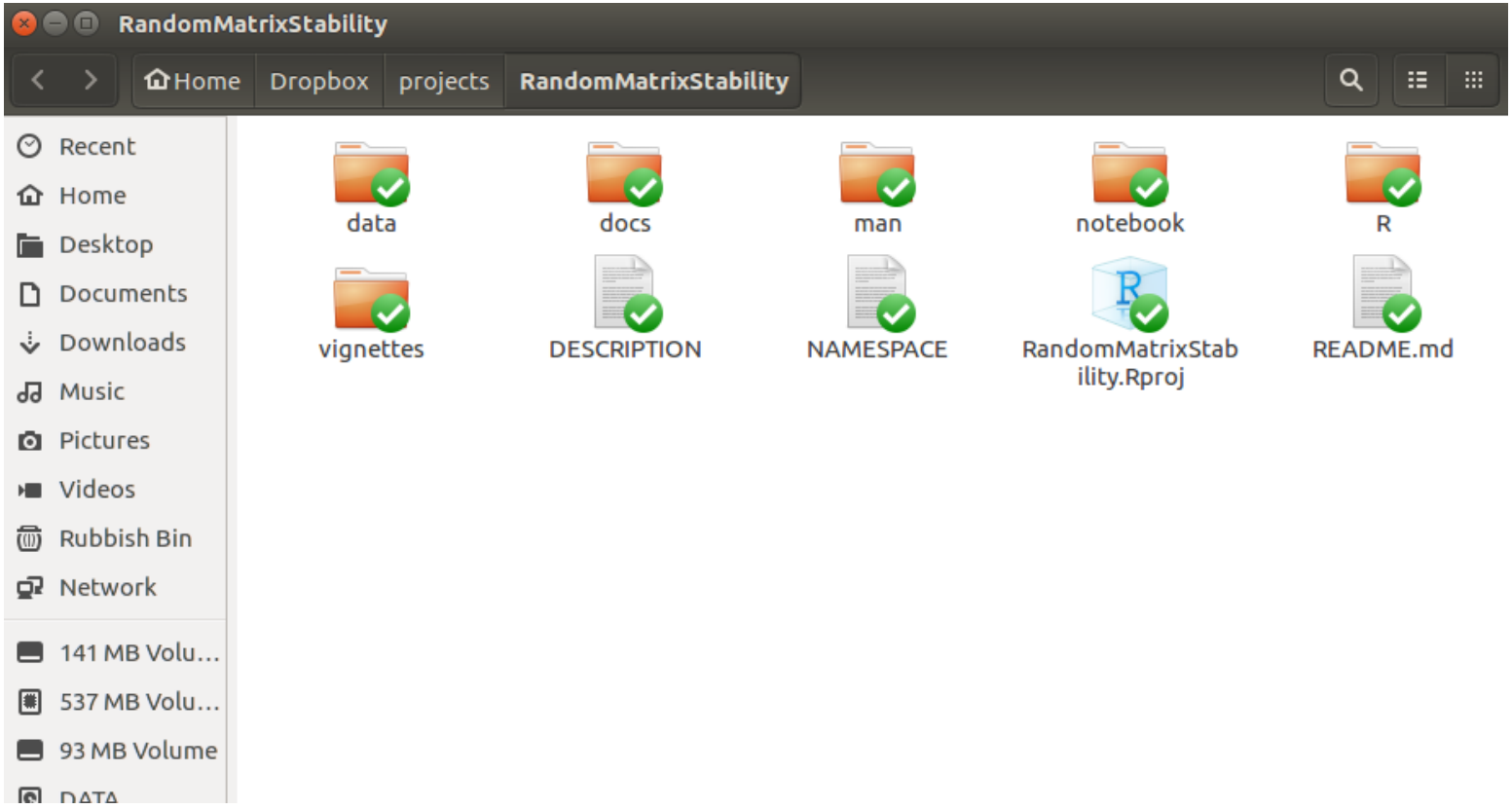
- Software that records changes you make to files over time
  - Manage different *versions* of files (no need to 'Save As...')
  - Recover old files, keep track of file changes
  - Collaborate with others on shared files

What version control software does

- Software that records changes you make to files over time
  - Manage different *versions* of files (no need to 'Save As...')
  - Recover old files, keep track of file changes
  - Collaborate with others on shared files

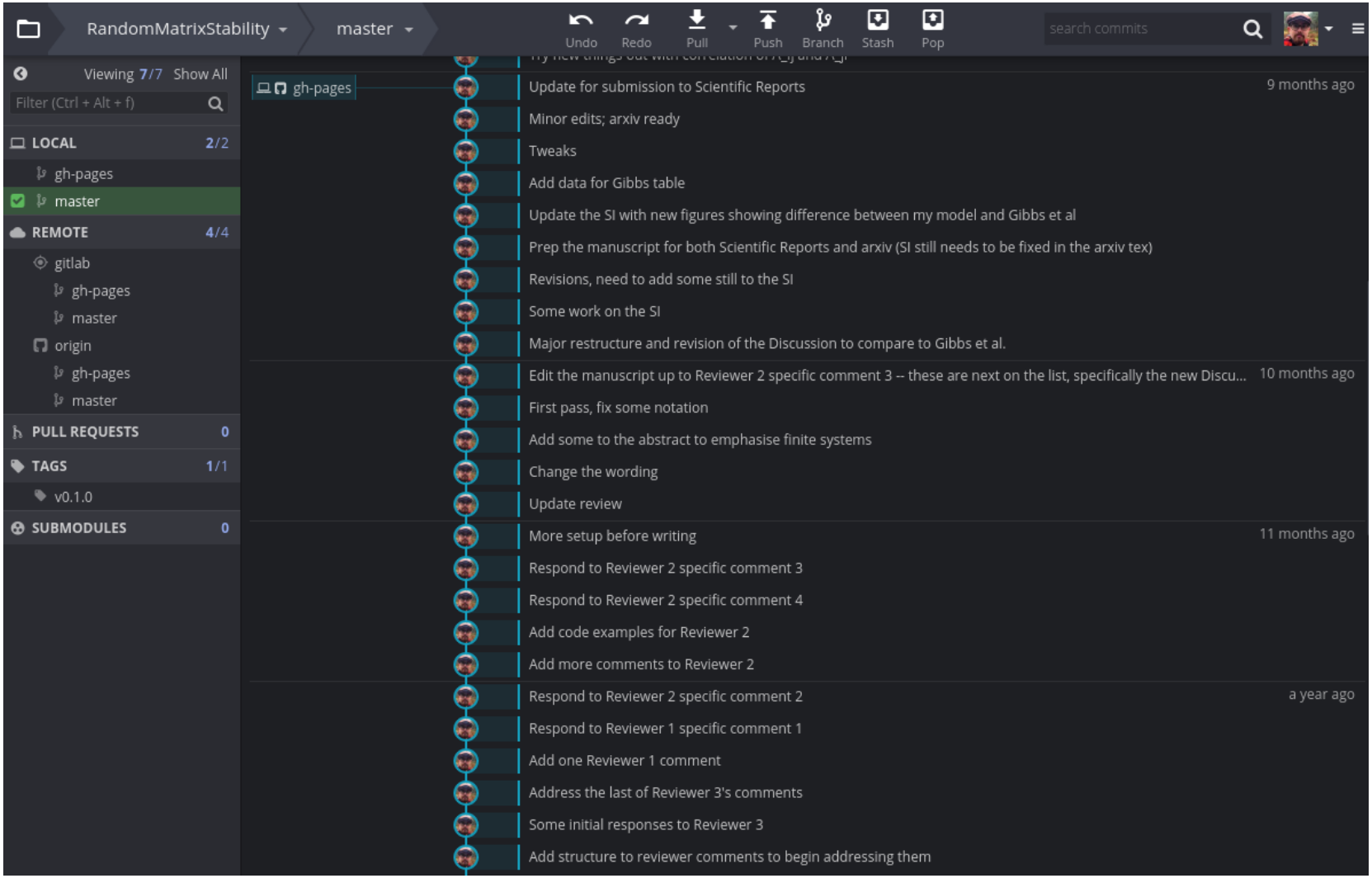
- **Put more intuitively**, version control takes a snapshot in time (called a '**commit**') of all the files in one of your folders (called '**repositories**')
  - Visualise changes to your files over time
  - Look at the differences between file versions
  - Record who changed files, and what they changed

Inside of a project on version control



Folders (a.k.a, 'repositories') include all data files, R code, notes, manuscript drafts, etc.

Full annotated timeline of folder changes (GitKraken)



Full annotated timeline of folder changes (GitHub)

Commits on Mar 1, 2019

Some work on the SI

bradduthie committed on 1 Mar 2019 ✓

5d29331 <>

Major restructure and revision of the Discussion to compare to Gibbs ... ...

bradduthie committed on 1 Mar 2019 ✓

…et al.

a40ede5 <>

Commits on Feb 27, 2019

Edit the manuscript up to Reviewer 2 specific comment 3 -- these are ... ...

bradduthie committed on 27 Feb 2019 ✓

…next on the list, specifically the new Discussion paragraph

d411fd5 <>

Commits on Feb 22, 2019

First pass, fix some notation

bradduthie committed on 22 Feb 2019 ✓

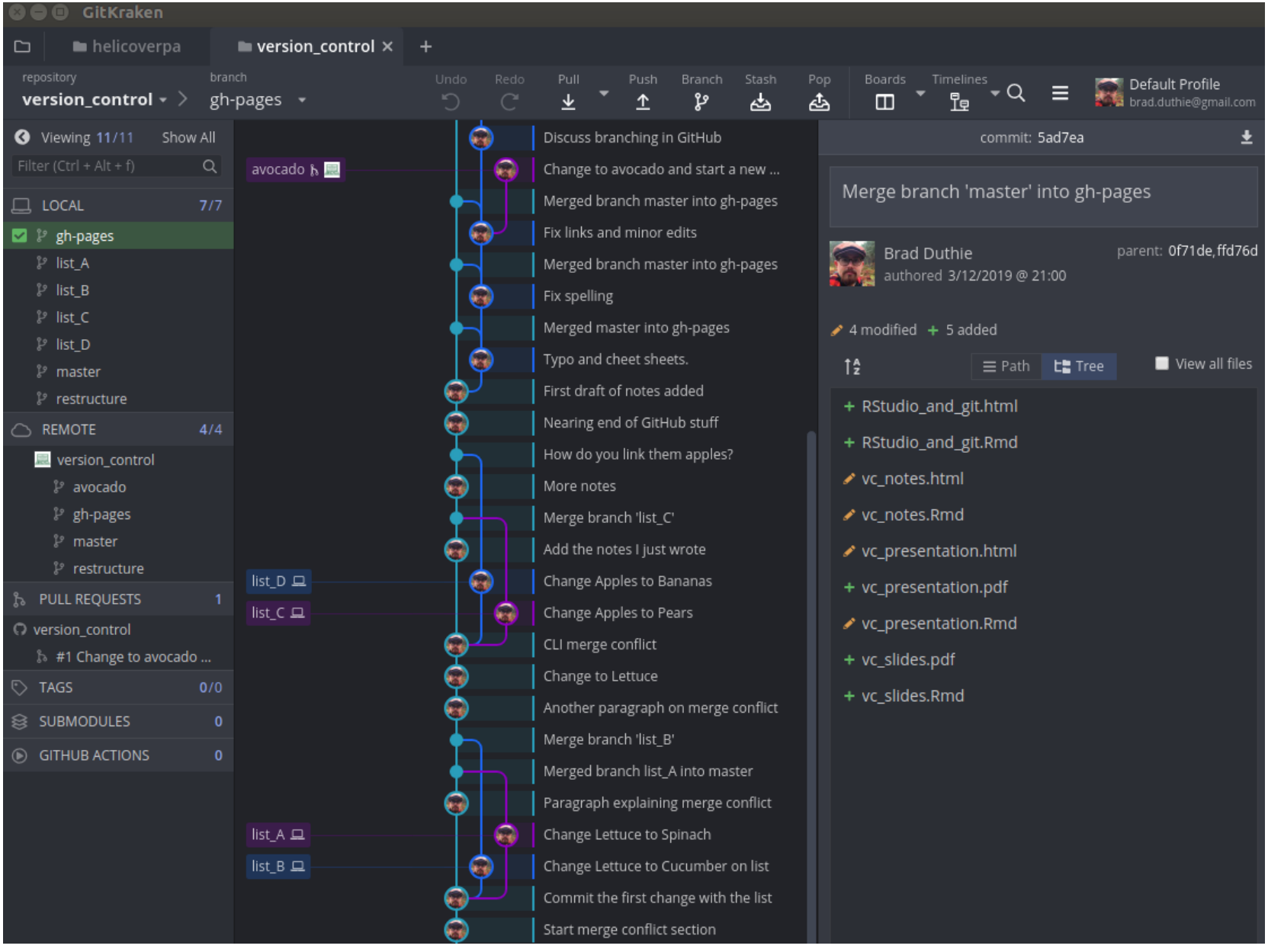
39f854b <>

Add some to the abstract to emphasise finite systems

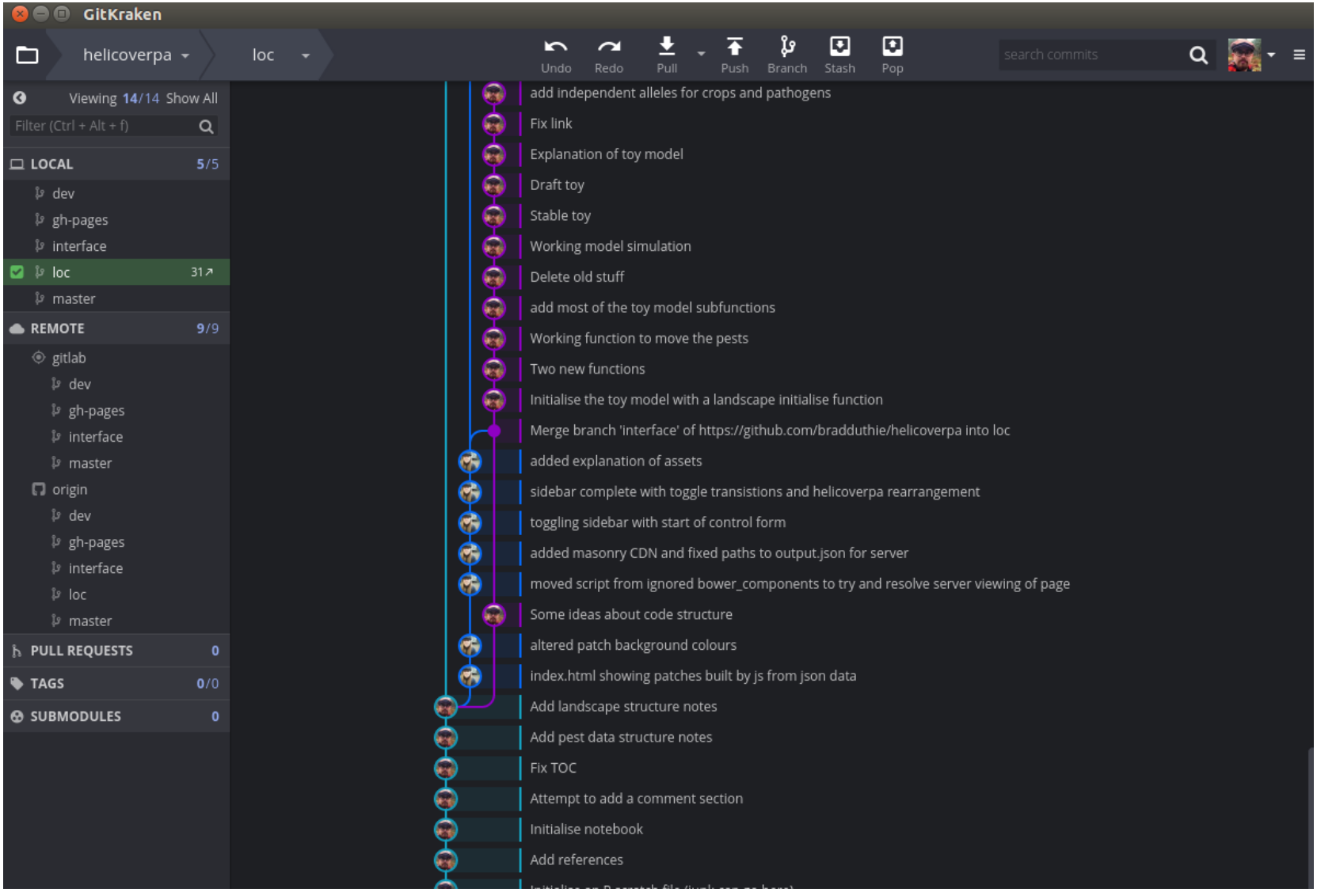
bradduthie committed on 22 Feb 2019 ✓

612aa4b <>

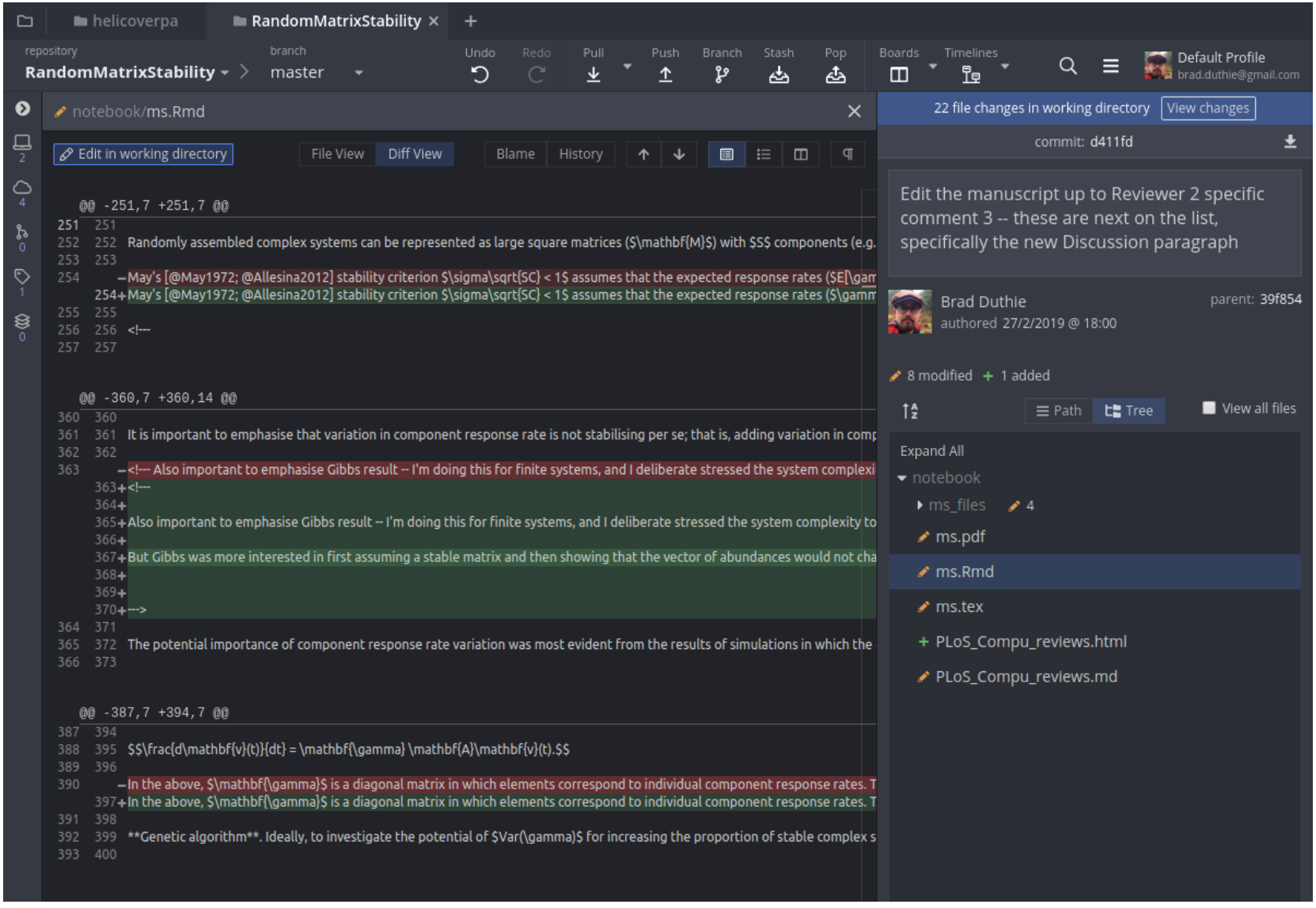
Parallel versions ('branches') of a folder (GitKraken)



Collaborative history or a shared folder (GitKraken)



Clear breakdown of what has changed (GitKraken)



Version control makes science easier

- **Organises files** by avoiding 'save as' multiple versions
  - analysis_1.R
  - analysis_2.R
  - analysis_FINAL.R
  - analysis_FINAL_no_really_this_time.R

Version control makes science easier

- **Organises files** by avoiding 'save as' multiple versions
  - analysis_1.R
  - analysis_2.R
  - analysis_FINAL.R
  - analysis_FINAL_no_really_this_time.R
- **Provides a clear history** of what you have done, when, and why (through commit comments)

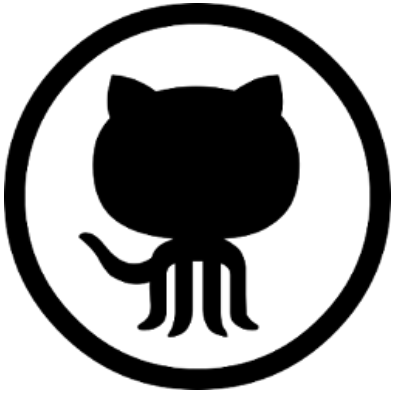
Version control makes science easier

- **Organises files** by avoiding 'save as' multiple versions
  - analysis_1.R
  - analysis_2.R
  - analysis_FINAL.R
  - analysis_FINAL_no_really_this_time.R
- **Provides a clear history** of what you have done, when, and why (through commit comments)
- **Saves time** by avoiding loss of data, analysis, or writing when integrating with GitHub

Version control makes science easier

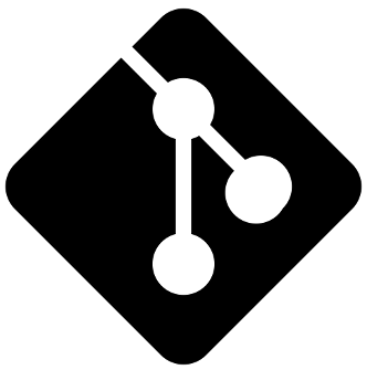
- **Organises files** by avoiding 'save as' multiple versions
  - analysis_1.R
  - analysis_2.R
  - analysis_FINAL.R
  - analysis_FINAL_no_really_this_time.R
- **Provides a clear history** of what you have done, when, and why (through commit comments)
- **Saves time** by avoiding loss of data, analysis, or writing when integrating with GitHub
- **Gives peace of mind** to experiment by removing any fear of breaking something that you know works

Version control can help open science



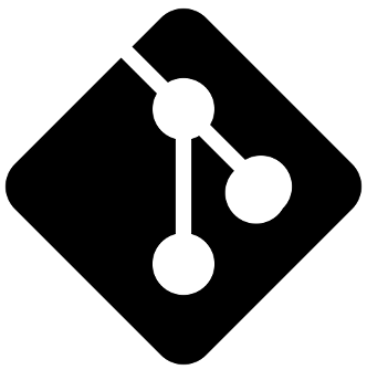
- Transparent record of data collection, analysis, and writing
- Record publicly available on GitHub, Bitbucket, or GitLab
- GitHub repository can be copied, reproduced, and discussed
- git and GitHub can track individual contributions to a project

Most researchers use git (and GitHub)



- Free and open-source
- Separate from GitHub

Most researchers use git (and GitHub)



- Free and open-source
- Separate from GitHub
- Works across platforms
  - Windows
  - Linux
  - Mac
- Invented by Linus Torvalds

Why focus on using GitKraken?

- Free to download and use
- Easy GitHub integration
- Graphical user interface
- Visualisation of repository

Reference documents and contact

Documents and data used

- https://bradduthie.github.io/talks/intro_to_Rcoding.pdf
- https://bradduthie.github.io/notes/R_intro_notes.html
- https://bradduthie.github.io/notes/vc_notes.html
- https://bradduthie.github.io/data/Bumpus_data.csv

Contact me

- alexander.duthie@stir.ac.uk
- @bradduthie@ecoevo.social
- https://github.com/bradduthie