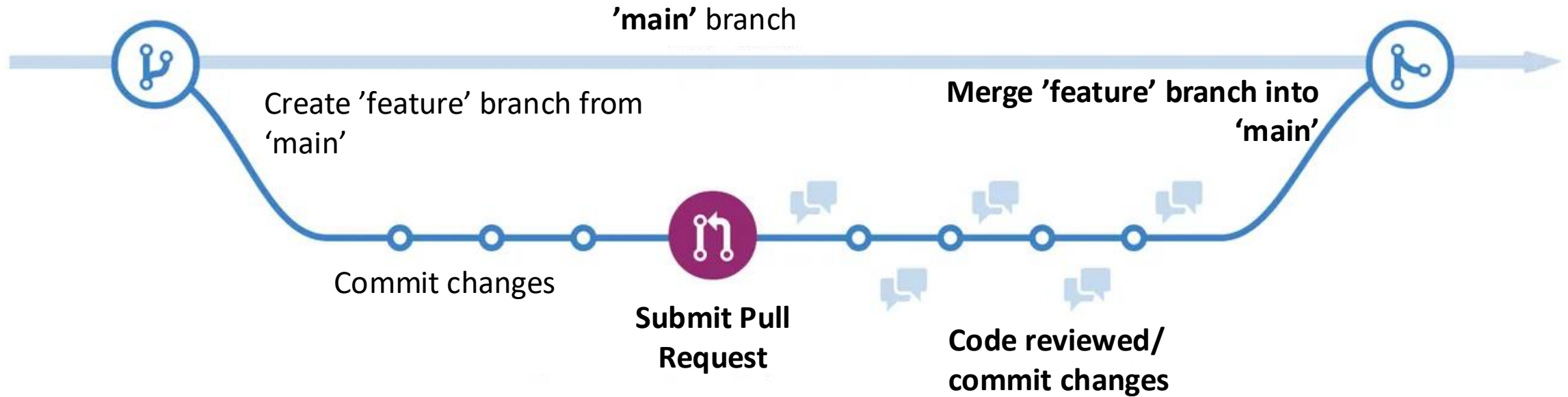


# Getting to grips with git



**Pull requests are useful for reviewing changes made to an existing code base**

# Getting to grips with git & github workshop

✓ 11 █████ README.md

<> [icon] ...

... @@ -1,2 +1,4 @@

10am

-

11am

**+ Introductory presentation**

(attendee laptops not required)

11.00

-

11:30

**+ Tea/coffee break**

(please do not bring food/drink into this room  
G.03)

11:30

-

12:30

**+ Practice reviewing R or Python in small  
groups/pairs**

(attendee laptops required)



Tuesday  
12<sup>th</sup> May 2026



Olga Polcock Room  
Lady Margaret Hall  
University of Oxford

# Workshop Expectations



This is a safe,  
supportive,  
respectful space



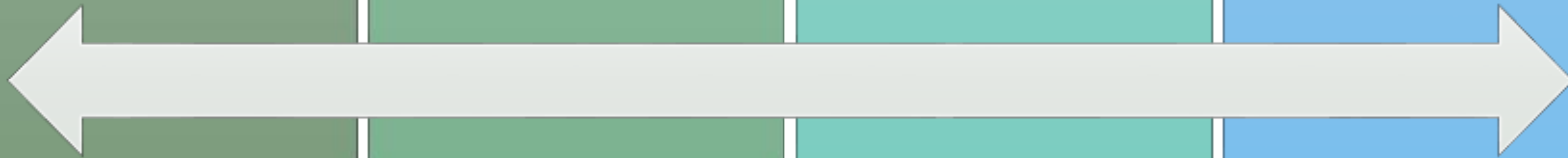
Participation is  
encouraged



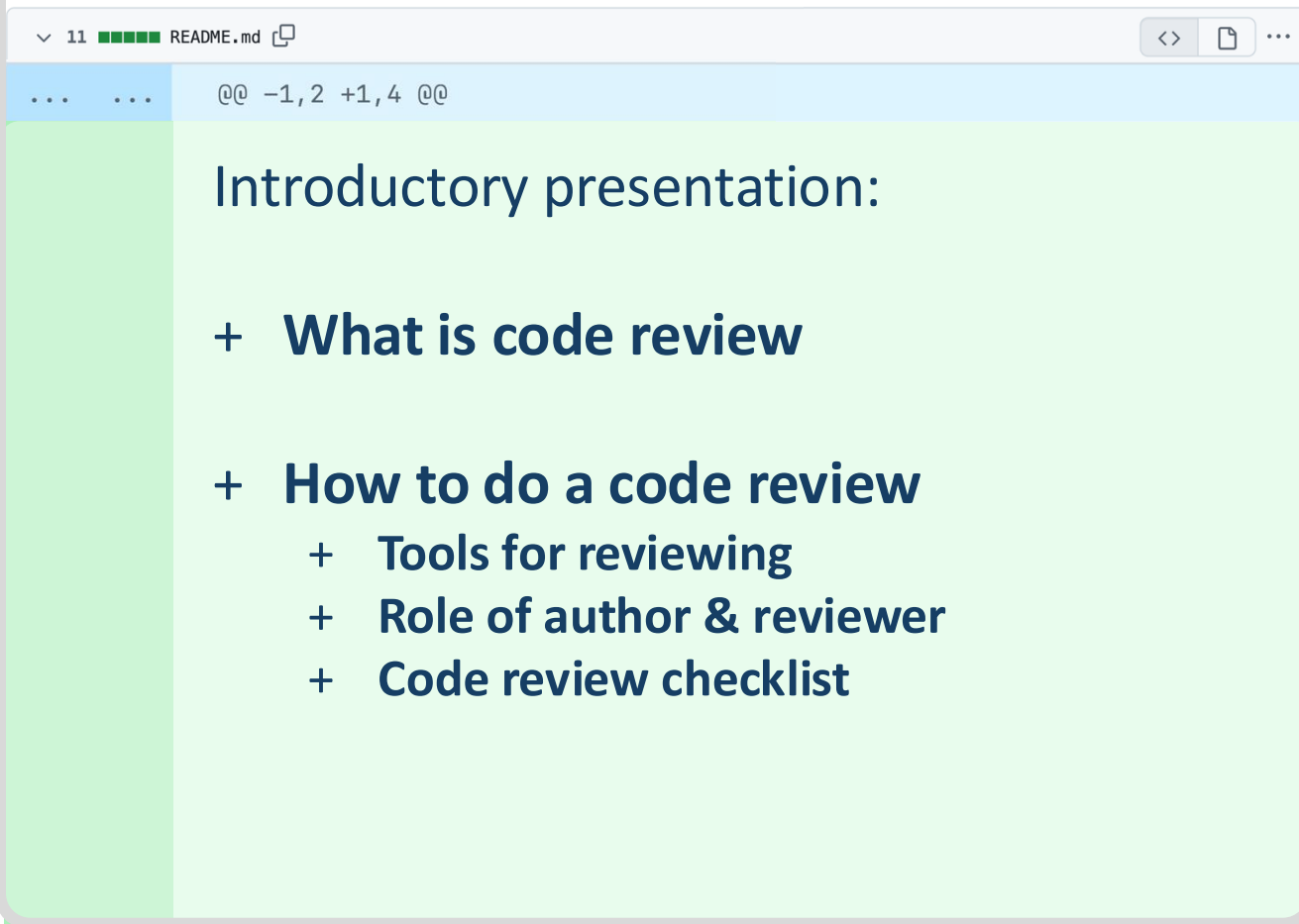
It's ok to make  
mistakes



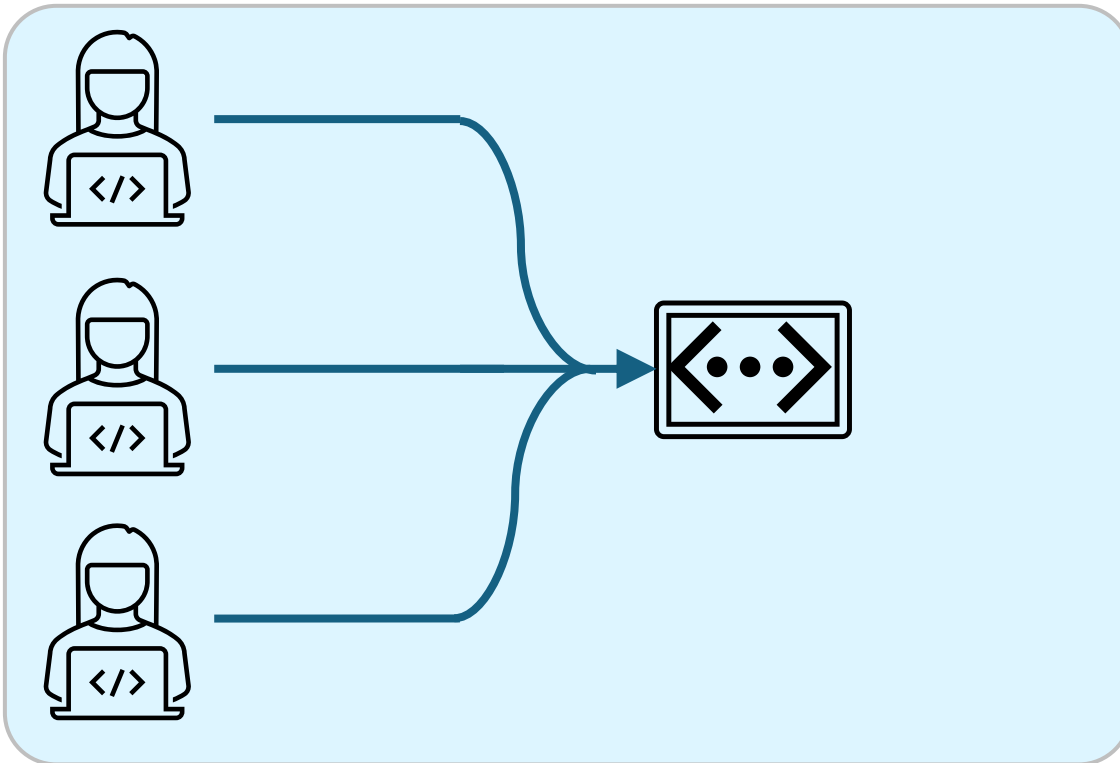
We'll use  
"Padlet" for  
questions,  
comments and  
responses in the  
practical session



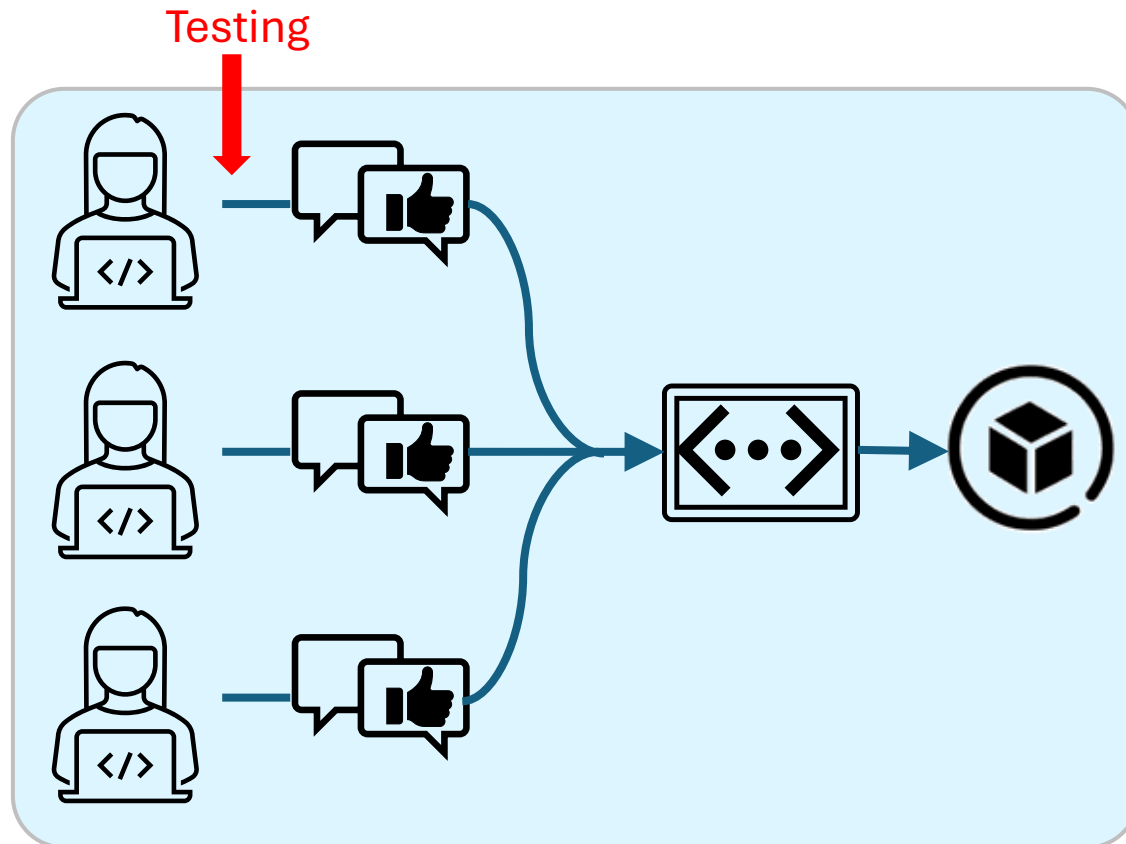
# Getting to grips with git



# Coding practices in industry



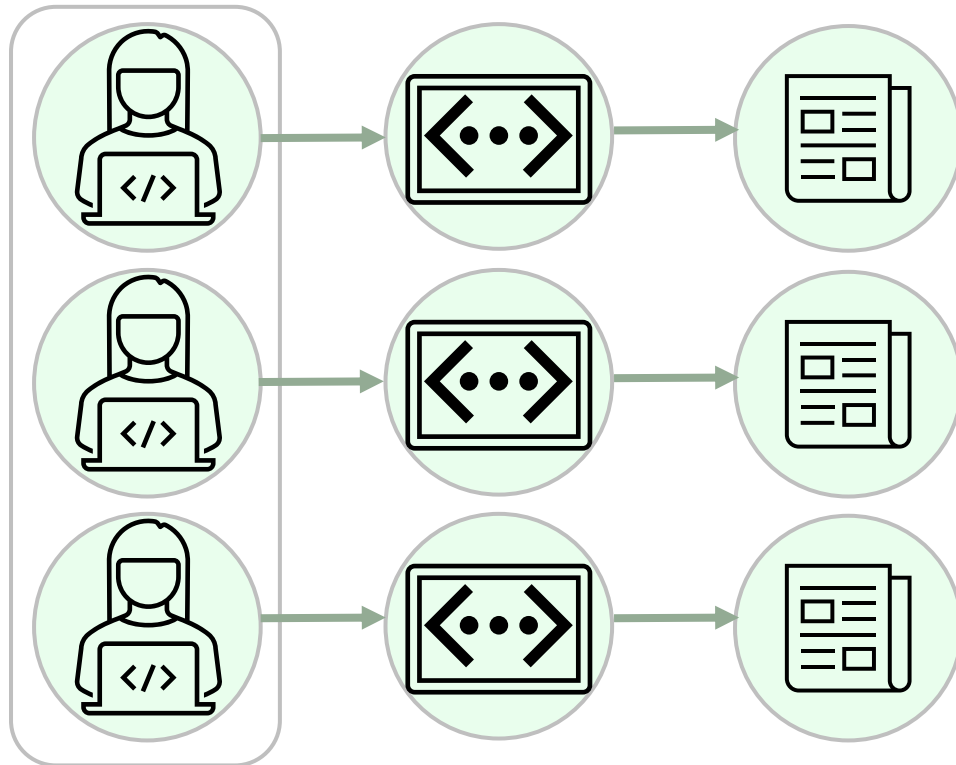
# Coding practices in industry



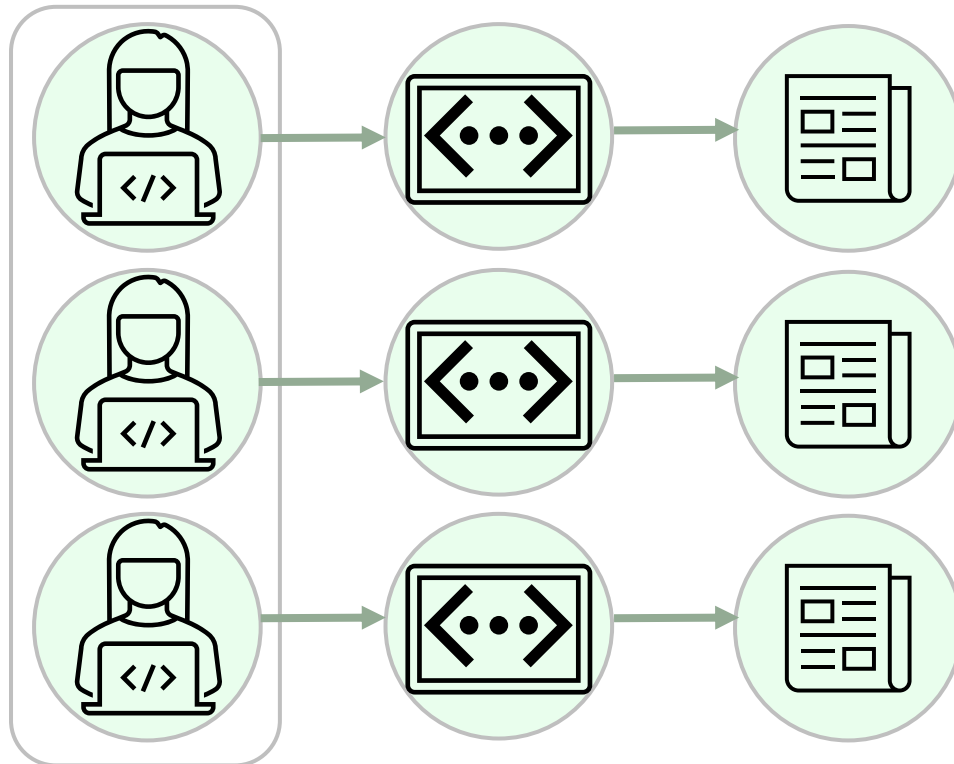
## Why code review is important in industry

- Mistakes happen
- ↑ code quality
- Opportunity to learn
- Discover bugs earlier
- ↑ maintainability of code

# Coding practices in academia



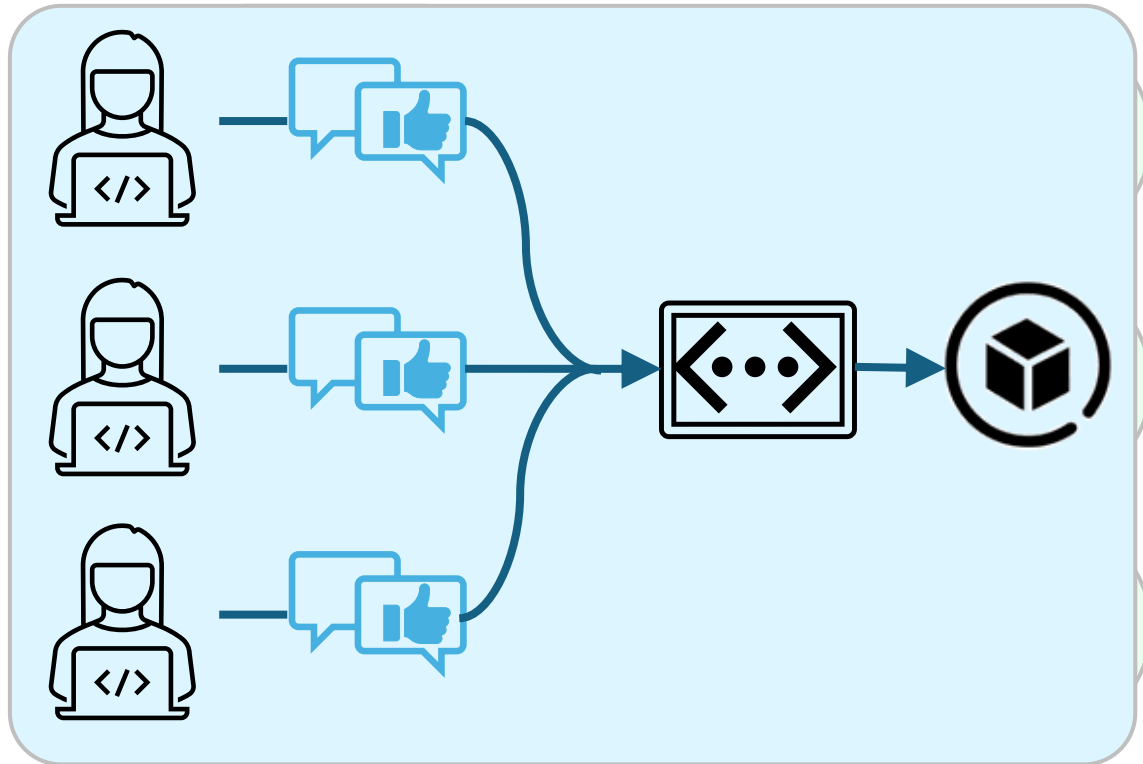
# Coding practices in academia



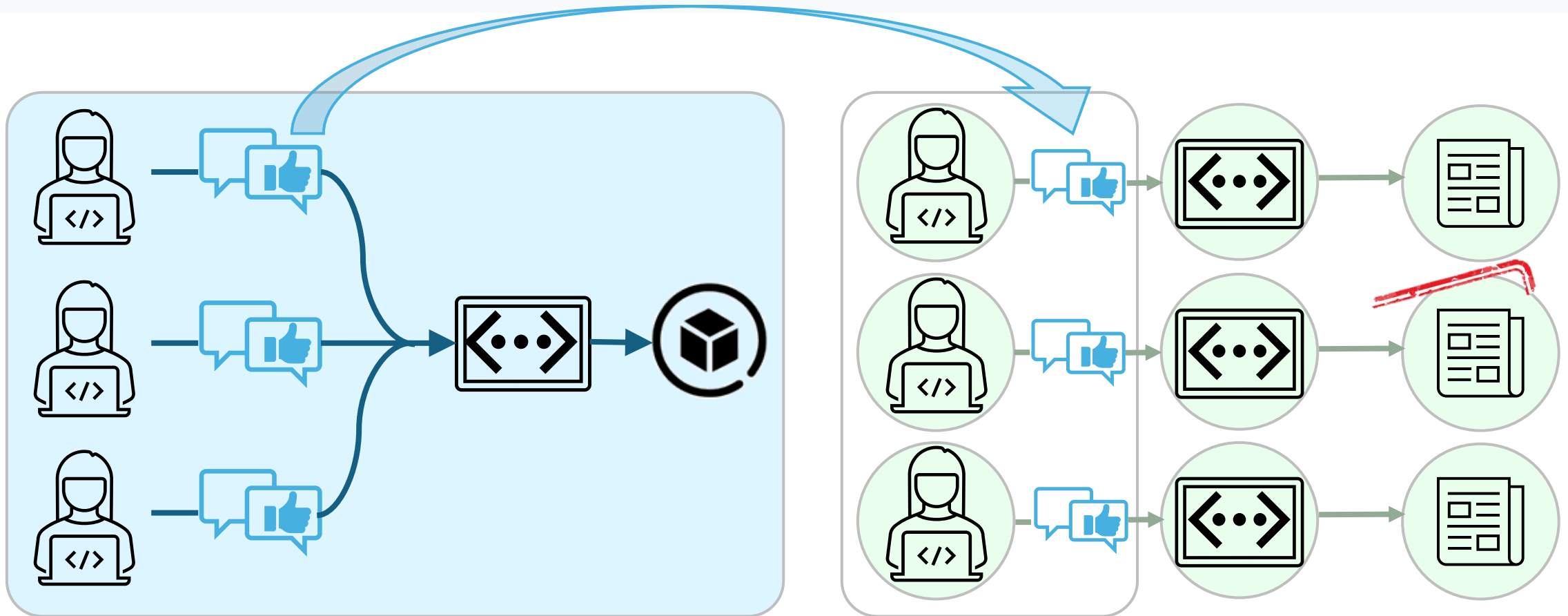
## Why code review is important in academia

- Mistakes happen
- ↑ code quality → reproducibility
- ↑ research quality
- Opportunity to learn
- Discover bugs earlier
- ↑ maintainability → reproducibility

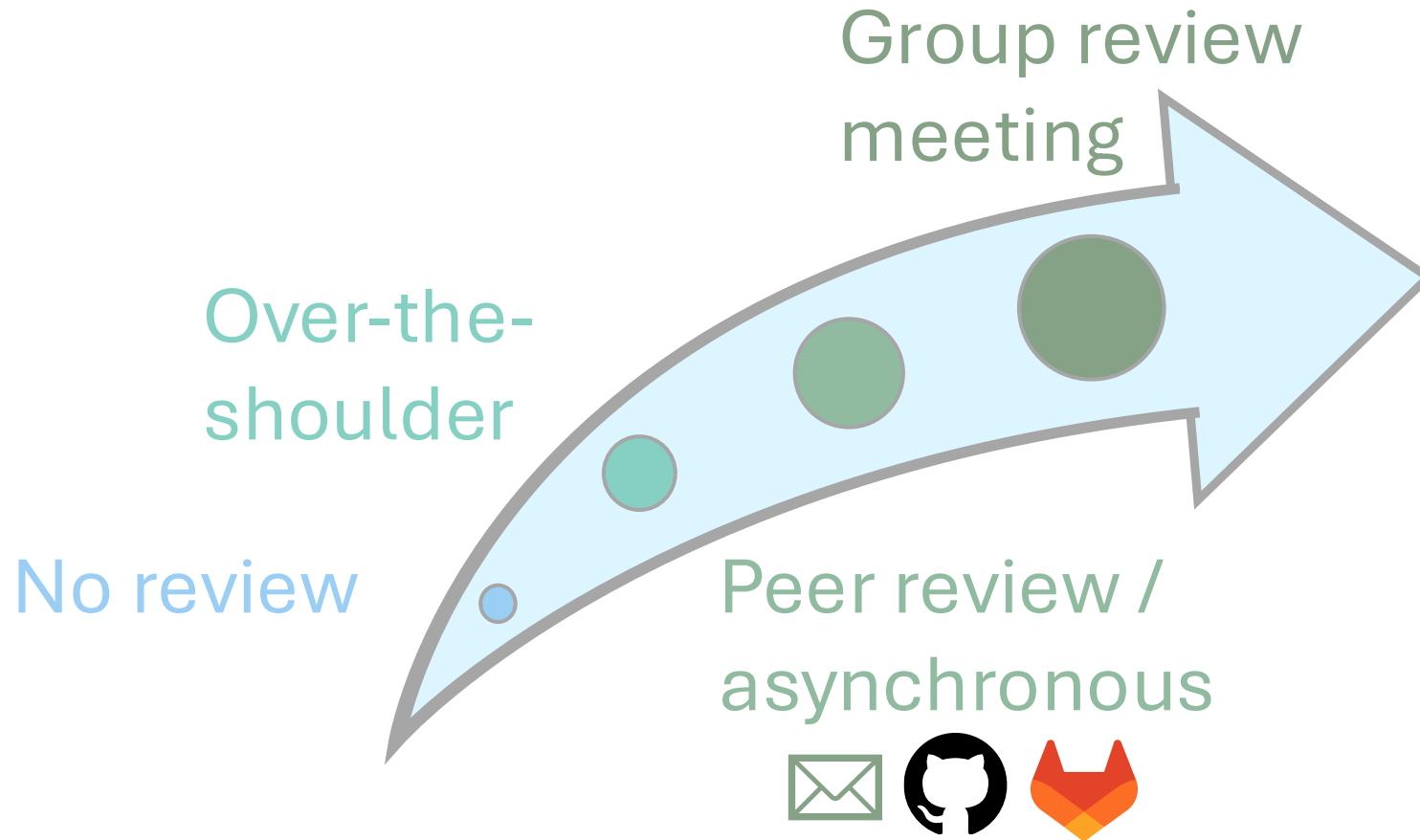
# Apply industry methods to academia



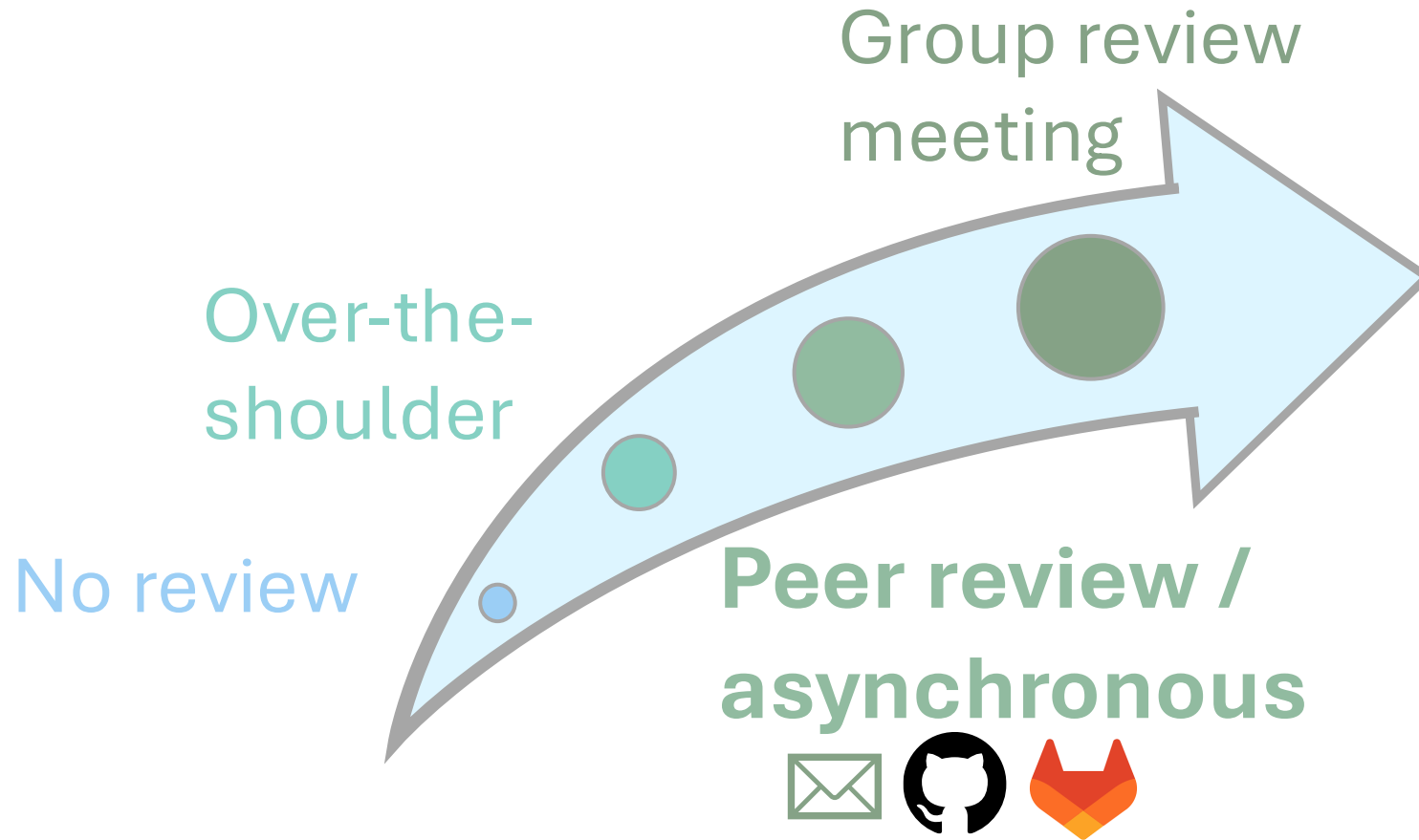
# Apply industry methods to academia



# Types of code review



# Types of code review



## Summary so far...

- Coding in industry vs academia is different
- Code review is probably also going to be different

Next...

- Methods used for asynchronous code review

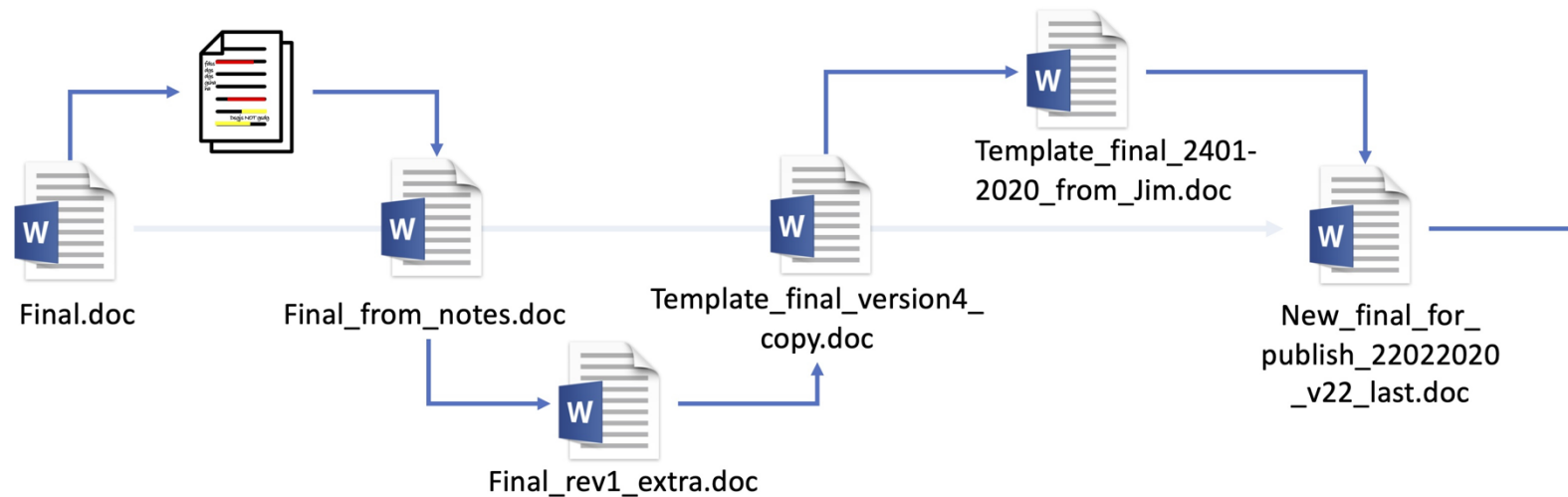
## Industry uses “pull requests” for peer review

# What is git... git branches... pull requests?

## How I think simple version control would be

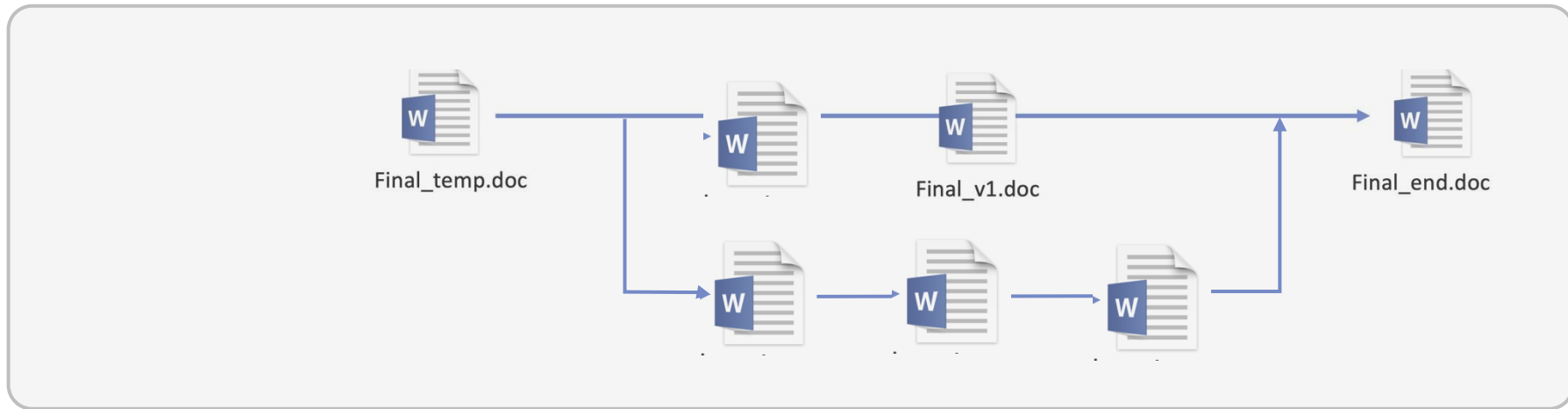


## How version control actually is



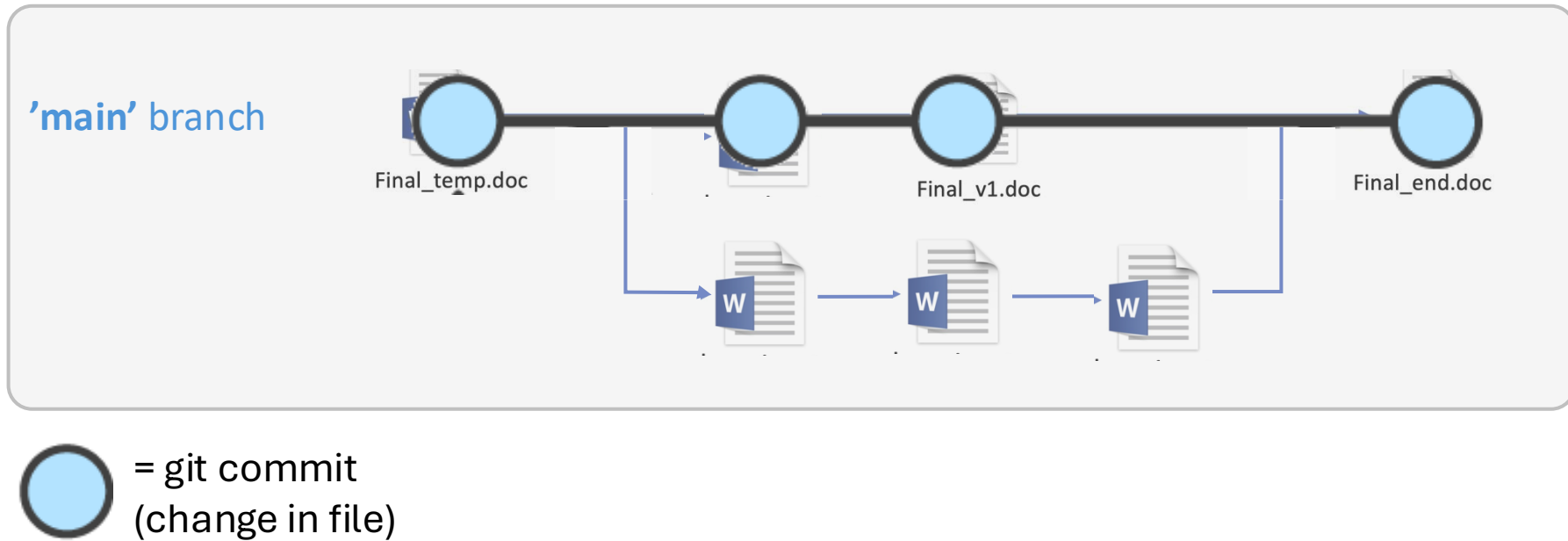
# What is git... git branches... pull requests?

**git = version control system**



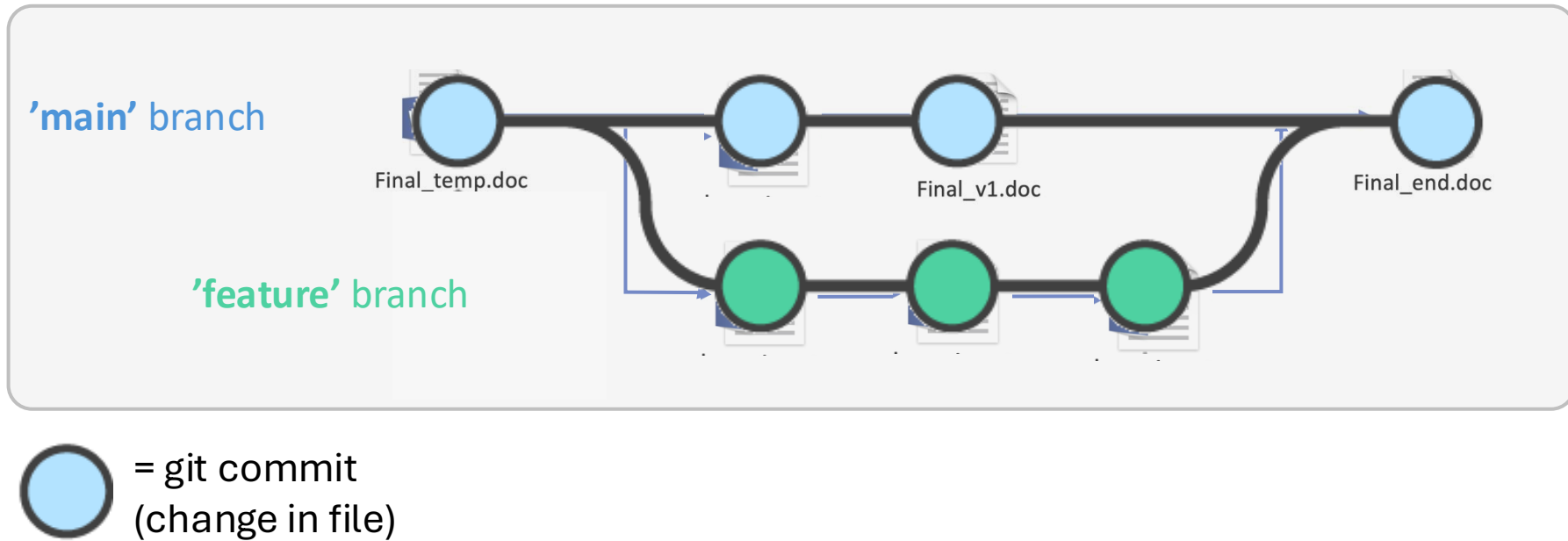
# What is git... git branches... pull requests?

**git = version control system**

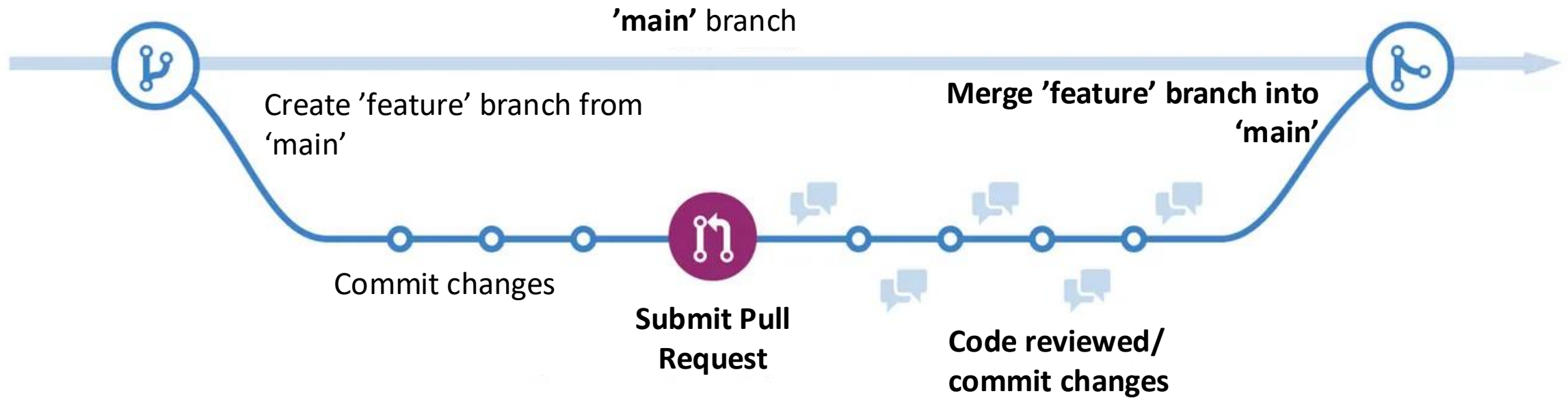


# What is git... **git branches**... pull requests?

**git = version control system**

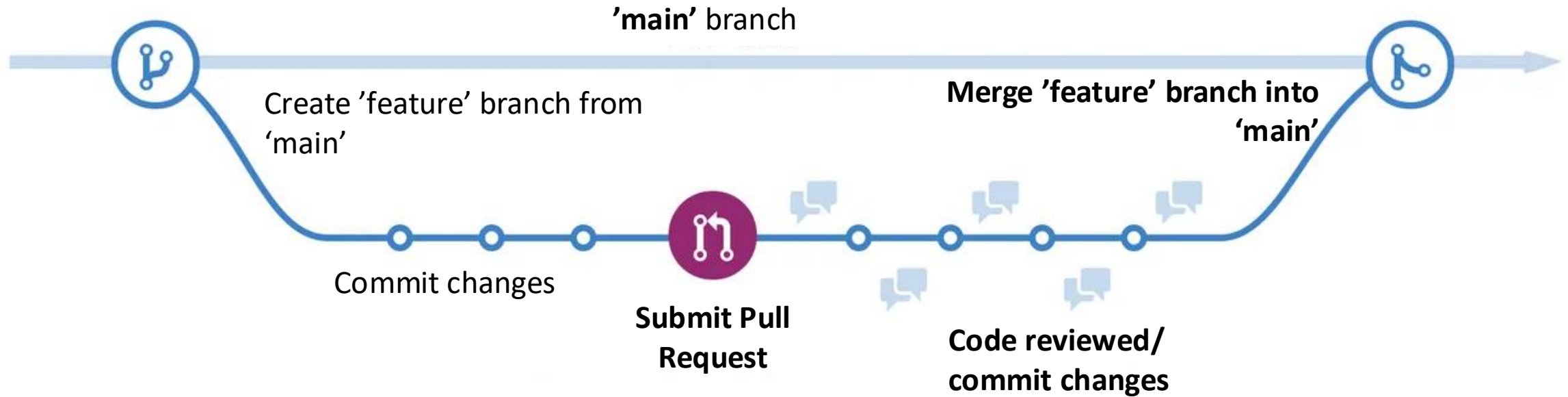


# What is git... git branches... pull requests?



**Pull requests are useful for reviewing changes made to an existing code base**

## Industry uses “pull requests” for peer review

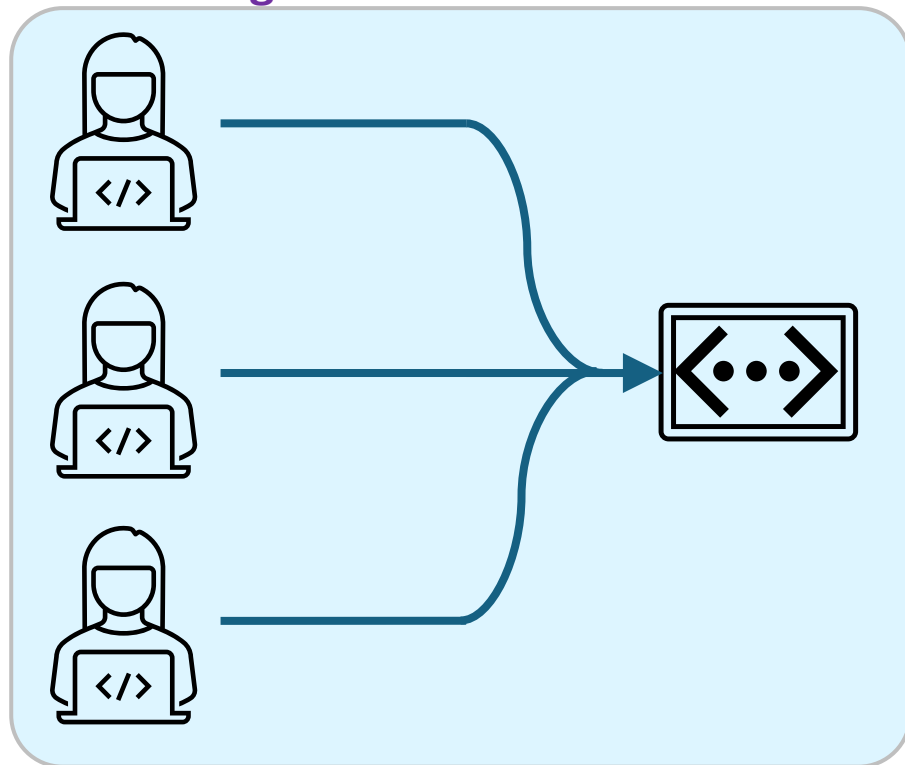


**Pull requests are useful for reviewing changes made to an existing code base**

# “Pull requests” for peer review in academia?

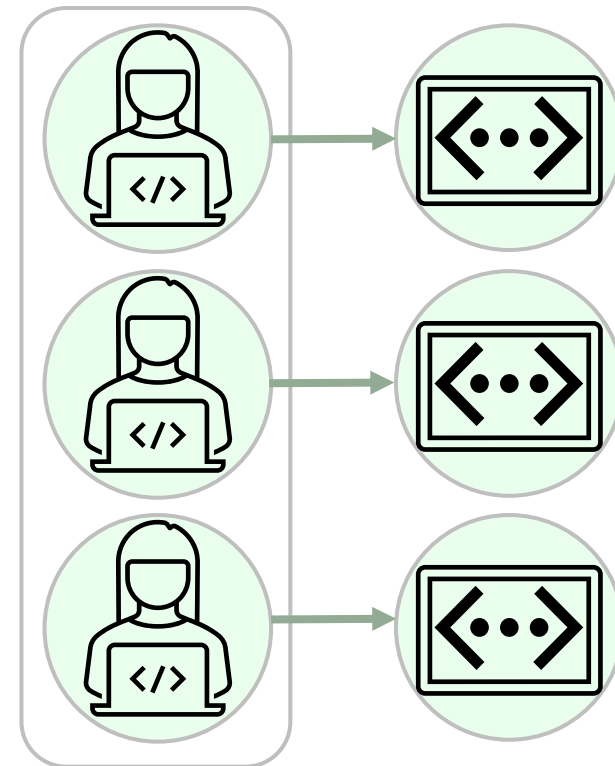
## Industry:

Changes made to an existing code base  
**Changes to code are reviewed**



## Academia:



Each researcher/project = new code base  
**Not just changes that need reviewing**









**Do we need something  
less advanced and more  
suitable for academia?**

# Oxford Code Review Network

<https://github.com/OxfordCodeReviewNet/forum>



OxfordCodeReviewNet / forum



<> Code

Issues 2


Pull requests

Actions

Projects

Security

Insights

 forum Public

Watch 32

Fork 8

Star 103


master 1 Branch 0 Tags










Go to file

t

+


<> Code

 **tlestang** Fix link to OxCRN intro video 50543ef · 5 years ago 60 Commits


 .github/ISSUE_TEMPLATE	Update issue templates	5 years ago
 CONTRIBUTING.md	Add CONTRIBUTING.md	5 years ago
 LICENSE	Add license	5 years ago
 README.md	Fix link to OxCRN intro video	5 years ago
 floobits.gif	Add section on Floobits and jitsi. Add comments abo...	5 years ago
 guidelines_for_reviewers.md	Mention C++ style guides instead of core guidelines	5 years ago
 remote.md	Minor fixes	5 years ago
 tmate-client-side.gif	Add missing gifs	5 years ago
 tmate-server-side.gif	Add missing gifs	5 years ago

README

CC-BY-SA-4.0 license

 **Oxford code review network**

[Code review guidelines](#) | [Getting started](#) | [Tools for remote code reviews](#) | [Guidelines for reviewers](#) | [Events](#)



Want to get feedback on your code? Could use a second pair of eyes to track down a bug? Interested in reviewing other researchers' code?

About

A central repository to coordinate code reviews between researchers at the University of Oxford

Readme

CC-BY-SA-4.0 license

Activity

Custom properties

103 stars

32 watching

8 forks

Report repository


Releases


No releases published

Packages

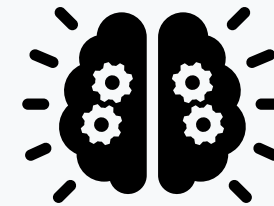
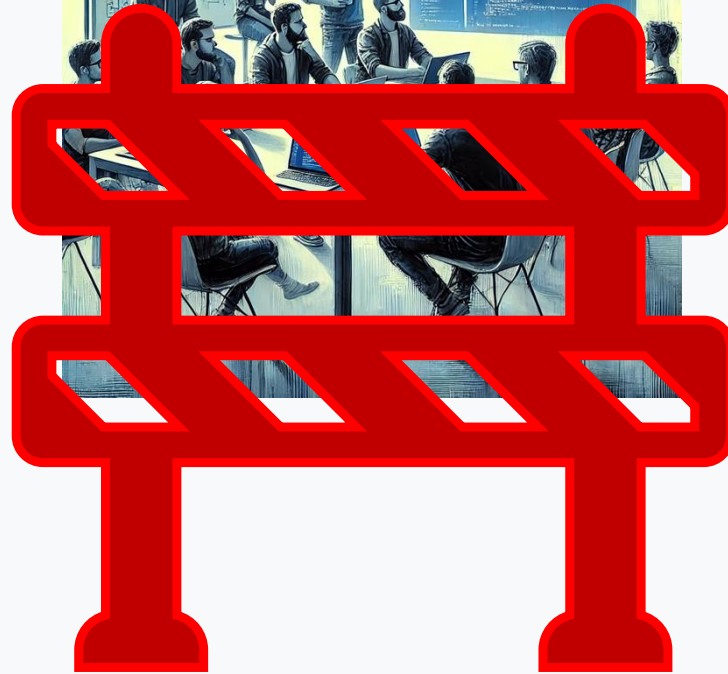
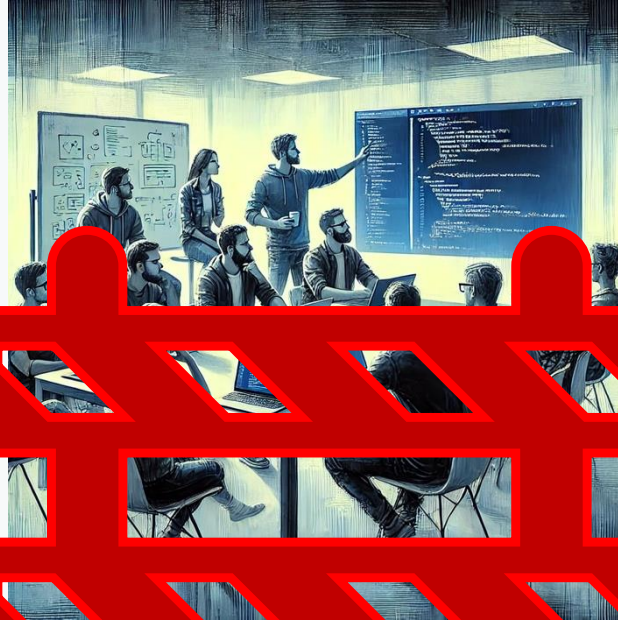
No packages published

Contributors 2

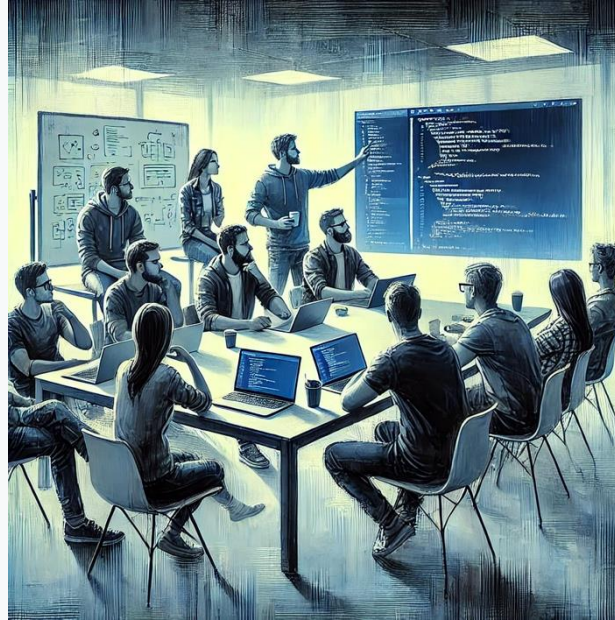
 **tlestang** Thibault Lestang

 **fcooper8472** Fergus Cooper

# Barriers for code review



# Barriers for code review



**Reviewing the code is more important  
than the tools used to review**

# Personal experience of code review

- Didn't use pull requests or GitHub issues...
- My code was on a shared GitHub repository
- New coding language to me - nextflow
- My colleague looked at the script I was working on
- We had regular meetings where I would walkthrough/explain the code
- Restructured logic of code execution, learnt new functions, etc.

# Summary so far...

- Tools for code review balanced with reducing barriers

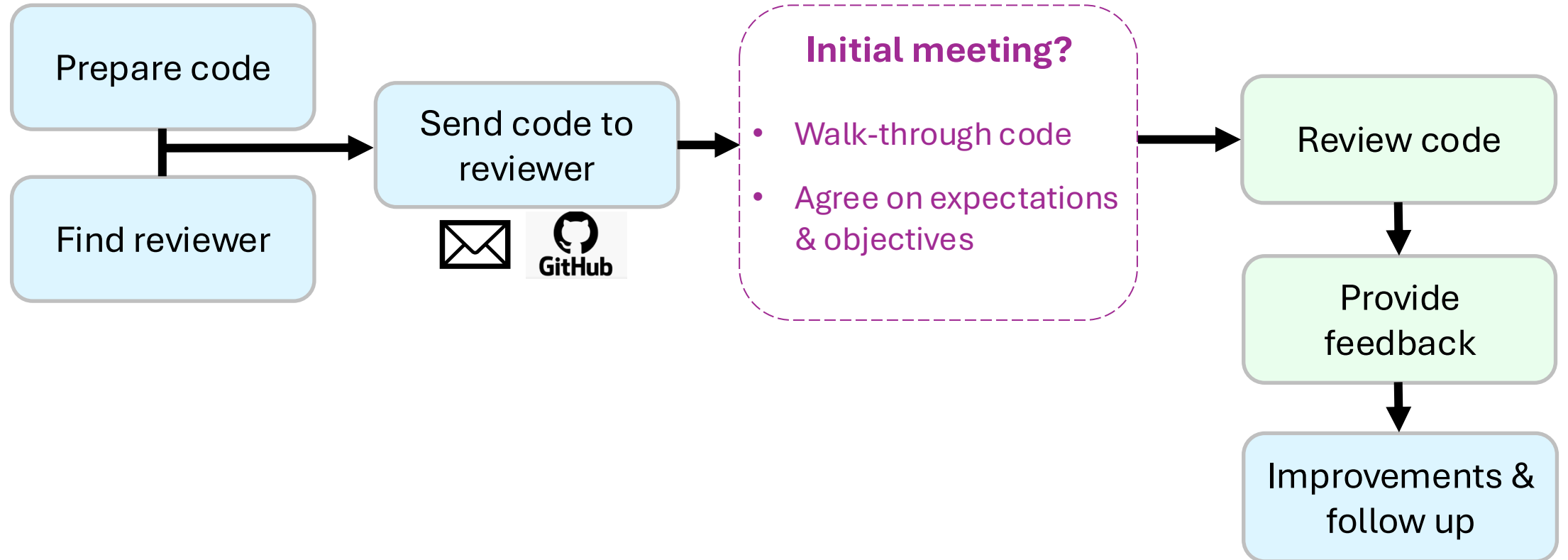
Next...

- Explain how a code review is done (role of author & reviewer)
- Things to look for in code when reviewing (checklist)

# Overview - How to do a code review

Role of code author

Role of code reviewer



# Preparing code for review

## Role of code author

- Identify **50 to 200 lines** of code you want reviewed

- Make it easy to review

- Decide the **focus** of the review

- Ask for **specific feedback**

- Be prepared to **walk away**



**documentation**



**logical order of execution**



# Finding a reviewer

## Role of code author

- Any researcher that codes is a good candidate
- Any coding experience is useful
- Colleagues in your research group,  
collaborators,  
mentors,  
students...

# How to do a code review

Role of code reviewer

# How to do a code review

## Role of code reviewer



- **Simply ask questions**
- Get the code author to explain their code in detail..  
*“Can you help me understand what this line of code does?”*
- Leads to them finding bugs & areas of improvements themselves

# How to do a code review

## Role of code reviewer



- **Check code logic**
- Does the code do what is expected?

# How to do a code review

## Role of code reviewer



Be kind, give **personal opinions**  
rather than imperative statements

*“I think this function’s name could be improved”*

*NOT*

*“You should rename this function”*

# Variable names

Use descriptive names that convey intent



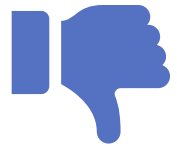
```
1 def calc(a, b):  
2     return a * b
```



```
1 def calculate_area(width, height):  
2     return width * height
```

# Hard coded values

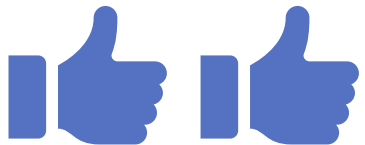
Avoid “magic numbers”



```
1 for (int i = 0; i < 26; i++) {
```



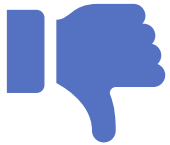
```
1 alphabetLength = 26;  
2 for (int i = 0; i < alphabetLength; i++) {
```



```
1 alphabetLength = alphabetData.size();  
2 for (int i = 0; i < alphabetLength; i++) {
```

# Duplicated code

Don't repeat yourself (DRY)



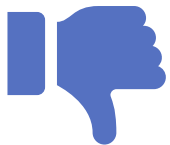
```
1 dfA <- filter(df, group == "A")
2 analysisA <- t.test(dv~condition, data = dfA)
3
4 dfB <- filter(df, group == "B")
5 analysisB <- t.test(dv~condition, data = dfB)
```



```
1 subtest <- function(data, level) {
2   sub_df <- filter(data, group == level)
3   t.test(dv~condition, data = sub_df)
4 }
5
6 analysisA <- subtest(df, "A")
7 analysisB <- subtest(df, "B")
```

# Complex if else statements

Flatten nested conditional statements with guard clauses



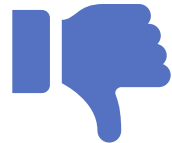
```
1 calculate_value <- function(x) {  
2   if (x > 0) {  
3     if (x < 10) {  
4       return(x * 2)  
5     } else {  
6       if (x < 20) {  
7         return(x * 3)  
8       } else {  
9         return(x * 4)  
10      }  
11    }  
12  } else {  
13    return(0)  
14  }  
15 }
```



```
1 calculate_value <- function(x) {  
2   if (x <= 0) return(0)  
3   if (x < 10) return(x * 2)  
4   if (x < 20) return(x * 3)  
5   return(x * 4)  
6 }
```

# Long functions

Functions should be short and do one thing



```
1 long_function <- function() {  
2   # Step 1: Do a lot of things  
3   # Step 2: More things  
4   # Step 3: Even more things  
5 }
```



```
1 step_one <- function() { ... }  
2 step_two <- function() { ... }  
3 step_three <- function() { ... }  
4  
5 long_function <- function() {  
6   step_one()  
7   step_two()  
8   step_three()  
9 }
```

# Obscure lines

Resist clever one-liners



```
1 result = reduce(sum, map(square, filter(positive, map(double, data))))
```



```
1 filtered_data = filter(positive, data)
2 doubled_data = map(double, filtered_data)
3 squared_data = map(square, doubled_data)
4 result = reduce(sum, squared_data)
```

# File paths

All file references should use relative paths, not absolute paths.



```
1 patientRecords <- read.csv("C:/Users/Username/project/data/patientRecords.csv")
```



```
1 patientRecords <- read.csv("data/patientRecords.csv")
```

“here” R package: <https://here.r-lib.org/>

# File names

Name files so both people and computers can easily find things



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

three principles for (file) names

machine readable

human readable

plays well with default ordering

Use YYYY-MM-DD format for dates



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

# Unintended behaviour

Validate inputs to prevent unintended behaviour or errors



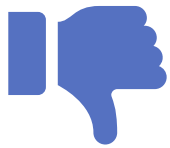
```
1 calculate_scaled_log <- function(value) {  
2   log_value <- log(value)  
3   scaled_value <- log_value * 10  
4   return(scaled_value)  
5 }
```



```
1 calculate_scaled_log_good <- function(value) {  
2   if (value <= 0) {  
3     stop("Input must be a positive number for log()")  
4   }  
5   log_value <- log(value)  
6   scaled_value <- log_value * 10  
7   return(scaled_value)  
8 }
```

# Comments

Explain the “why” not the “what”



- Redundant comments
- Complicated comments

```
1 # Subtract the mean age from age
2 centeredAge <- data$age - mean(data$age)
```



- Warnings of consequences
- Assumptions made

```
1 # Mean-center age to improve interpretation,
2 # reduce multicollinearity, and better model
3 # individual age-related changes over time
4 # in longitudinal trajectories.
```

# Documented code

Functions and classes should contain docstrings



```
1  #' Descending order
2  #'
3  #' Transform a vector into a format that will be sorted in descending order.
4  #' This is useful within [arrange()].
5  #'
6  #' @param x vector to transform
7  #' @export
8  #' @examples
9  #' desc(1:10)
10 #' desc(factor(letters))
11 #'
12 #' first_day <- seq(as.Date("1910/1/1"), as.Date("1920/1/1"), "years")
13 #' desc(first_day)
14 #'
15 #' starwars %>% arrange(desc(mass))
16 desc <- function(x) {
17   obj_check_vector(x)
18   -xtfrm(x)
19 }
```

# Documented code

Functions and classes should contain docstrings



```
1 def desc(x):  
2     """  
3     Descending order  
4     Transform a vector into a format that will be  
5     sorted in descending order.  
6     This is useful within [arrange()].  
7  
8     Parameters:  
9     x (array-like): The vector to transform.  
10  
11    Returns:  
12    array-like: A transformed version of `x` for  
13    descending sorting.  
14  
15    Example:  
16    >>> desc([1, 2, 3])  
17    [-1, -2, -3]  
18    """  
19    # function code  
20    return x
```

# Documented data

Be clear, consistent, and provide context

- Names (i.e., the column names)
- Labels/description
- Codings (e.g., 1 = always, 5 = never)
- Data type (e.g., binary, continuous)
- Descriptives (e.g., min, max)
- Data units (e.g., mg/L, months)
- Missing values (e.g., NA, 999)

# Coding Style

“Good coding style is like correct punctuation: you can manage without it, but it sure makes things easier to read.”

- Notation and naming



- Syntax (spacing, indentations, line length)
- Commenting guidelines
- And more...

Google style guide on many languages: <https://google.github.io/styleguide/>

R style guide: <https://style.tidyverse.org/>

# Automate what can be automated

Code author can use a linter to automate coding style checks before review



- **R – lintr** R package

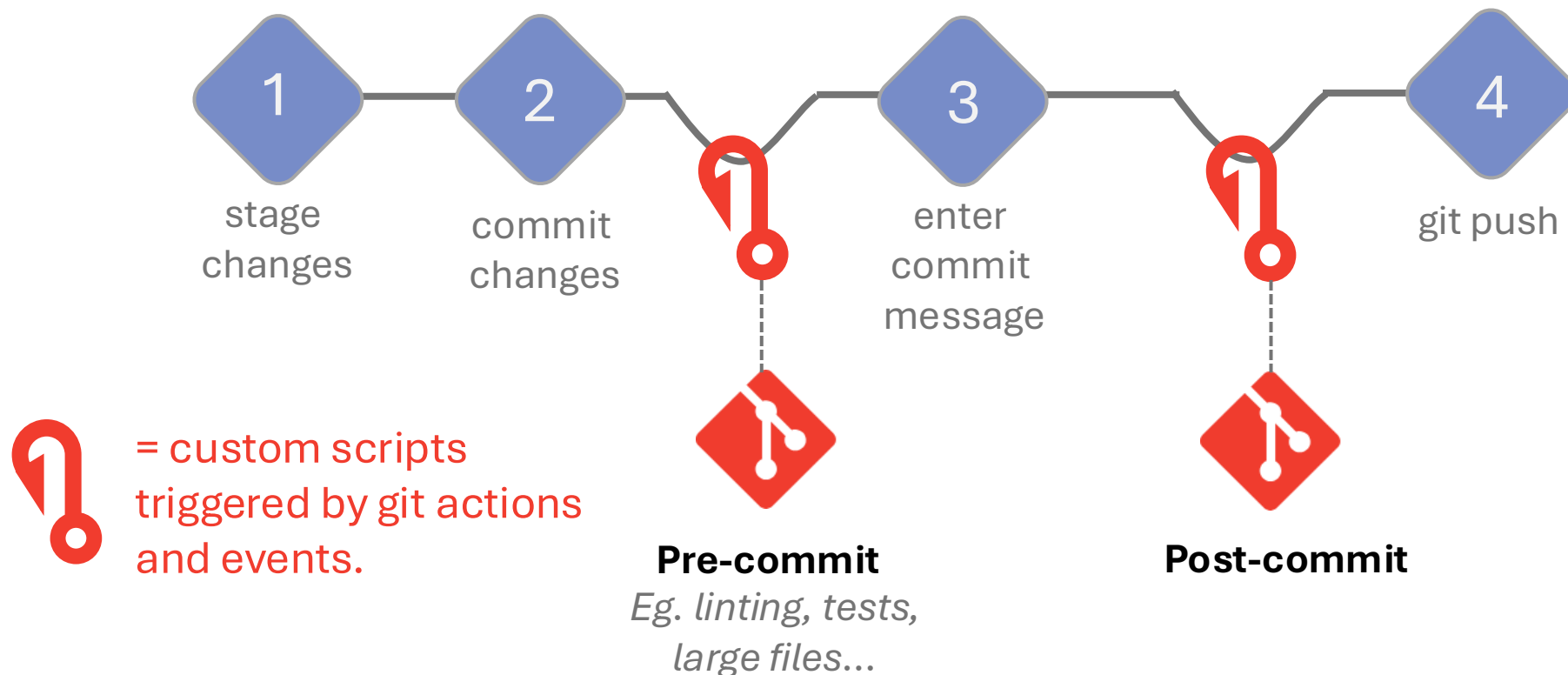
*“lintr provides static code analysis for R. It checks for adherence to a given style, identifying syntax errors and possible semantic issues, then reports them to you so you can take action.”*

- **Python** - list of linters:

*<https://github.com/vintasoftware/python-linters-and-code-analysis>*

# Using git hooks and pre-commits

Code author can use automated pre-commit checks before review



# Summary of checklist

- Variable names
- Hard coded values/magic numbers
- Duplicated code
- Complex if else statements
- Long functions
- Obscure lines
- File paths
- File names
- Unintended behavior
- Comments
- Documented code
- Documented data
- Coding style (with automated checks using linter/git hooks)

**Other things for a reviewer to  
check?**



Padlet



## Code Review in Academia Workshop

Ideas for tools/methods for code review



...

Pull requests



Talked about how industry uses pull requests

Other things to check for when reviewing code



...

Variable names



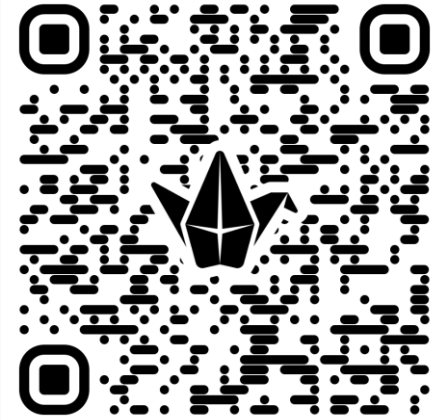
Use descriptive names that convey intent

Other comments



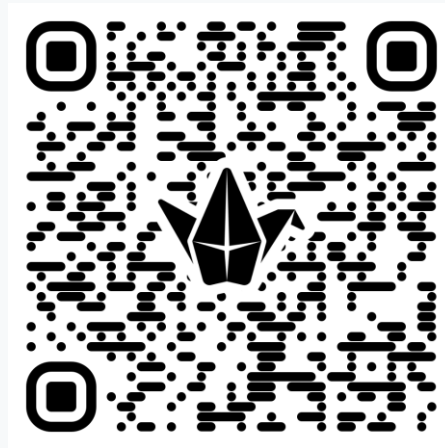
...

[https://tinyurl.com/  
code-review-padlet](https://tinyurl.com/code-review-padlet)



Please add  
your  
comments  
and ideas

# Questions?

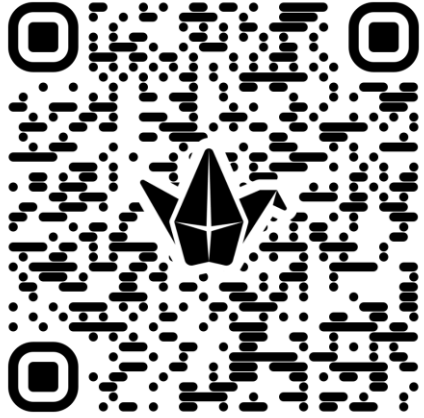


Link to padlet:

<https://tinyurl.com/code-review-padlet>

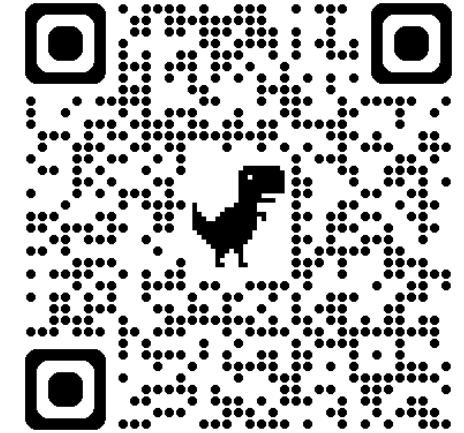
Please add your comments and ideas

# Code Review in Academia Workshop



Link to padlet:  
<https://tinyurl.com/code-review-padlet>

11 <div></div> README.md	
@@ -1,2 +1,4 @@	
10am - 11am	<del>+ Introductory presentation</del> (attendee laptops not required)
11:00 - 11:30	<b>+ Tea/coffee break</b> <b>(please do not bring food/drink into this room G.03)</b>
11:30 - 12:30	<b>+ Practice reviewing R or Python in small groups/pairs</b> (attendee laptops required)



Link to practical session:  
<https://tinyurl.com/code-review-practical>

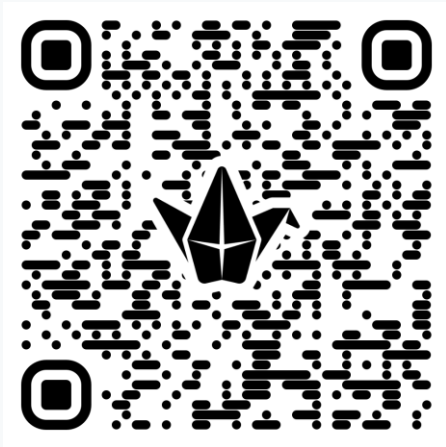
← Open these links on your laptop for practical session →

# Practical Session Starting Now!

## 1. Open these links on your laptop

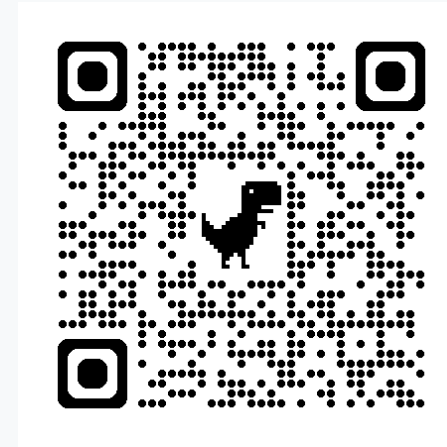
Link to padlet:

<https://tinyurl.com/code-review-padlet>



Link to practical session:

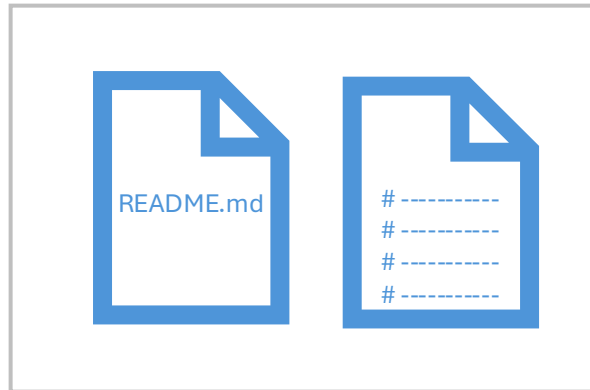
<https://tinyurl.com/code-review-practical>



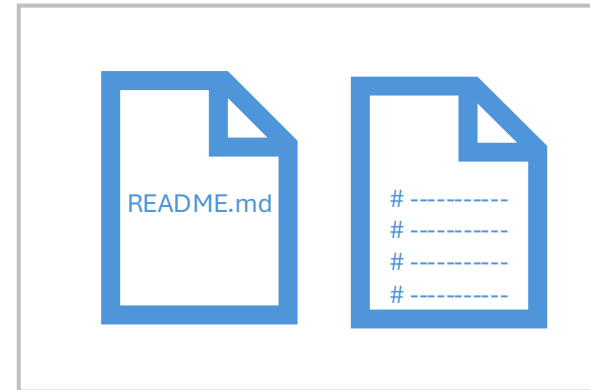
Links are also in the Eventbrite reminder email

# Code snippets (x2) to review in pairs/small groups

~10-15 mins in pairs/groups +  
~5-10 mins whole room discussion



~10-15 mins in pairs/groups +  
~5-10 mins whole room discussion

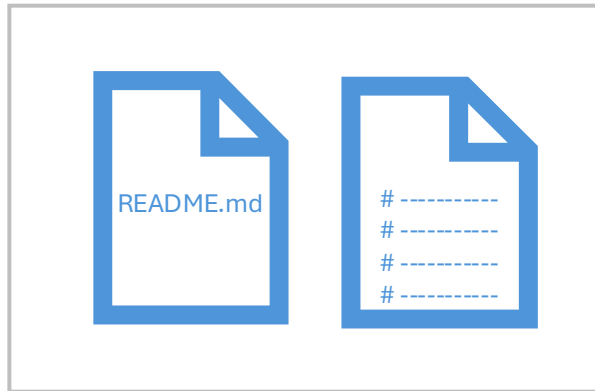


[https://github.com/EleanorSC/code-review-workshop/tree/main/practical\\_session](https://github.com/EleanorSC/code-review-workshop/tree/main/practical_session)



# First exercise: 01-code-review-fitbit/

~10-15 mins in pairs/groups +  
~5-10 mins whole room discussion

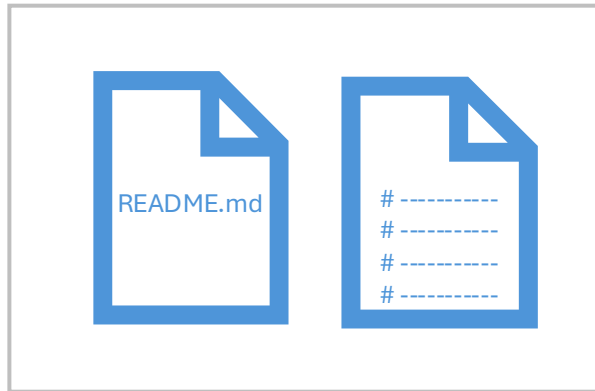


code-review-workshop/  
practical\_session/  
01-code-review-fitbit/  
├— README.md  
├— exploreStepsData.R  
└— exploreStepsData.py

- Pair up or form a group of 3 with the person next to you.
- Choose either the R or Python script to review together.
  - Original script was written in R
  - If your group has different R/Python experience, choose one language for this exercise and switch languages for the next exercise.
  - The languages are similar, so you can still follow the checklist.
- Read the README.md for context/documentation
- Use the checklist from the presentation as a guide.
- Write down any suggested changes in 10 mins.
  - Either in a word doc/note pad
  - Or in the padlet: <https://tinyurl.com/code-review-padlet>

# First exercise: 01-code-review-fitbit/

~10-15 mins in pairs/groups +  
~5-10 mins whole room discussion

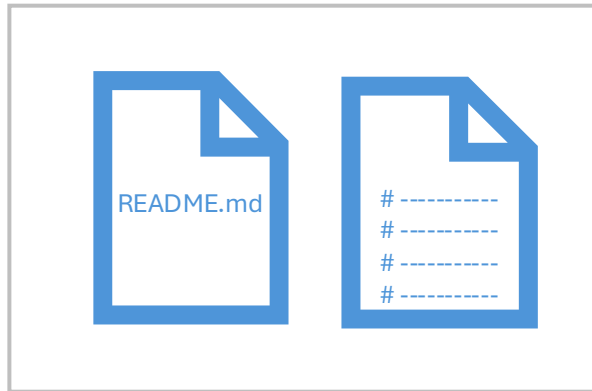


***WHOLE ROOM DISCUSSION  
STARTING NOW!***

code-review-workshop/  
practical\_session/  
01-code-review-fitbit/  
├— README.md  
├— exploreStepsData.R  
└— exploreStepsData.py

# Second exercise: 02-code-review-data-cleaning/

~10-15 mins in pairs/groups +  
~5-10 mins whole room discussion

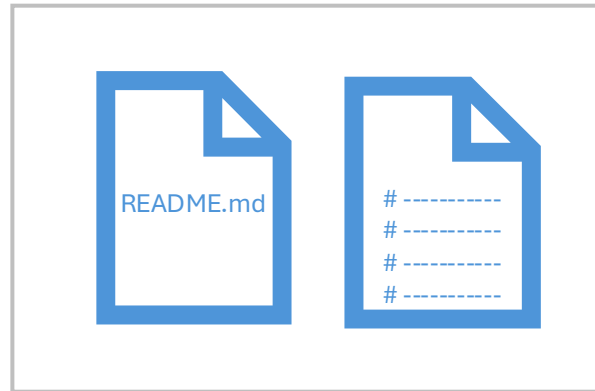


code-review-workshop/  
practical\_session/  
02-code-review-data-cleaning/  
├── README.md  
├── cluster\_ukb.R  
└── cluster\_ukb.py

- Stay in the same pair/group
- Choose either the R or Python script to review together.
  - Original script was written in Python.
  - *If your group has different R/Python experience, choose the language you didn't use in the first exercise.*
- Read the README.md for context/documentation
- Use the checklist from the presentation as a guide.
- Write down any suggested changes in 10 mins.
  - *Either in a word doc/note pad*
  - *Or in the padlet: <https://tinyurl.com/code-review-padlet>*

# Second exercise: 02-code-review-data-cleaning/

~10-15 mins in pairs/groups +  
~5-10 mins whole room discussion



***WHOLE ROOM DISCUSSION  
STARTING NOW!***

```
code-review-workshop/  
  practical_session/  
    02-code-review-data-cleaning/  
      ├── README.md  
      ├── cluster_ukb.R  
      └── cluster_ukb.py
```

# Questions?



**[tinyurl.com/code-review-feedback](https://tinyurl.com/code-review-feedback)**

We would be grateful for any feedback to improve future workshops