GitHub and other tricks to make your code accessible to others – including your future self

Randall Munroe

SICSS workshop

"GitHub and collaboration"

11 June 2024

by Marie Labussière

Introduction

Why bother?

Most of us write code without specific training!

Introduction

Why bother?

Most of us write code without specific training!

- ✓ Saving time and effort
- ✓ A condition for effective collaborations
- ✓ A matter of integrity

Introduction

Why bother?

Most of us write code without specific training!

- ✓ Saving time and effort
- ✓ A condition for effective collaborations
- ✓ A matter of integrity
  - ↪ Accessibility and replicability
  - ↪ Compliance with FAIR principles

Introduction

Ensuring reproducibility is key...and challenging!

Study by **Liu and Salganik (2019)**[1]: attempt to meet the highest standard of transparency for a special issue

[1] Liu, D. M., & Salganik, M. J. (2019). Successes and Struggles with Computational Reproducibility: Lessons from the Fragile Families Challenge. *Socius*, 5.

Introduction

Ensuring reproducibility is key...and challenging!

Study by **Liu and Salganik (2019)**[1]: attempt to meet the highest standard of transparency for a special issue

- ✗ only 2 of the 14 manuscripts were (computationally) reproducible
- ✗ after extensive review, 7 of the remaining 12 became reproducible

[1] Liu, D. M., & Salganik, M. J. (2019). Successes and Struggles with Computational Reproducibility: Lessons from the Fragile Families Challenge. *Socius*, 5.

Introduction

Ensuring reproducibility is key...and challenging!

Study by **Liu and Salganik (2019)**[1]: attempt to meet the highest standard of transparency for a special issue

- ✗ only 2 of the 14 manuscripts were (computationally) reproducible
- ✗ after extensive review, 7 of the remaining 12 became reproducible

➔ "***there was not one specific thing that caused huge difficulties***" **(p.10)**

[1] Liu, D. M., & Salganik, M. J. (2019). Successes and Struggles with Computational Reproducibility: Lessons from the Fragile Families Challenge. *Socius*, 5.

Outline

Your inputs

You

N = 19 replies – thank you!

Your inputs

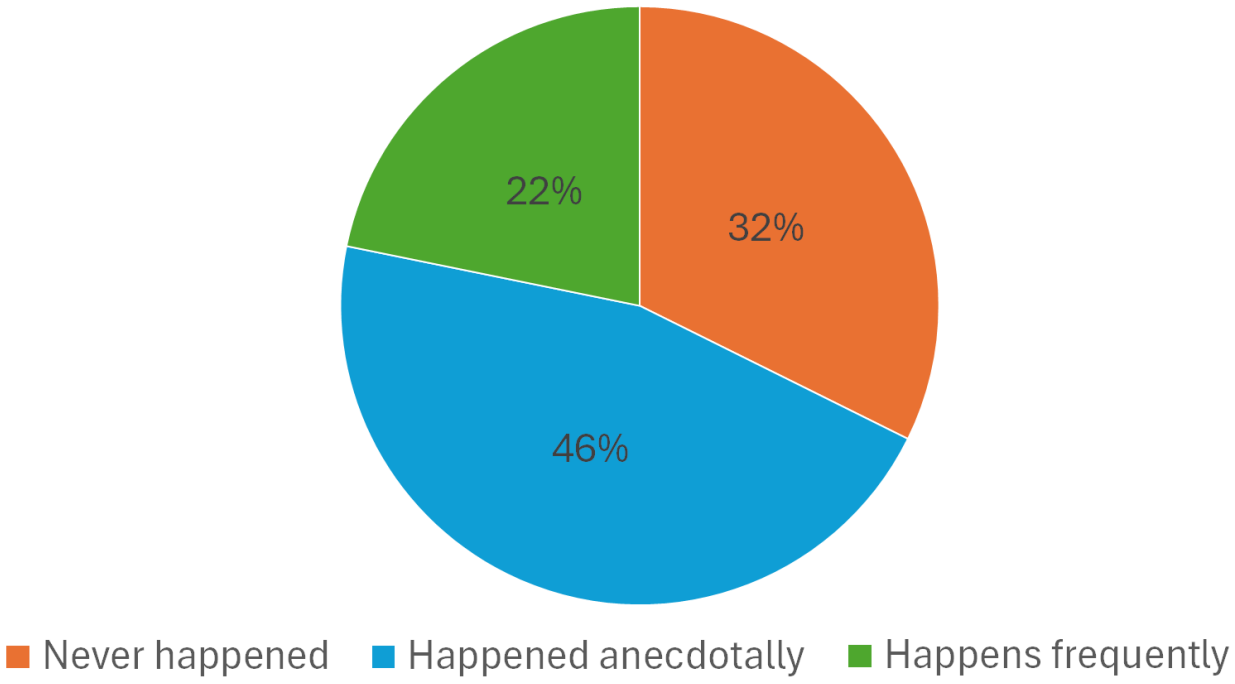
You

N = 19 replies – thank you!

- ✎ **Half** of you are familiar with GitHub but never used it (n=9/19), **half** already use it (10/19)
- ✎ Most of you **collaborate** on data preparation and analysis at least sometimes (n=14/19)

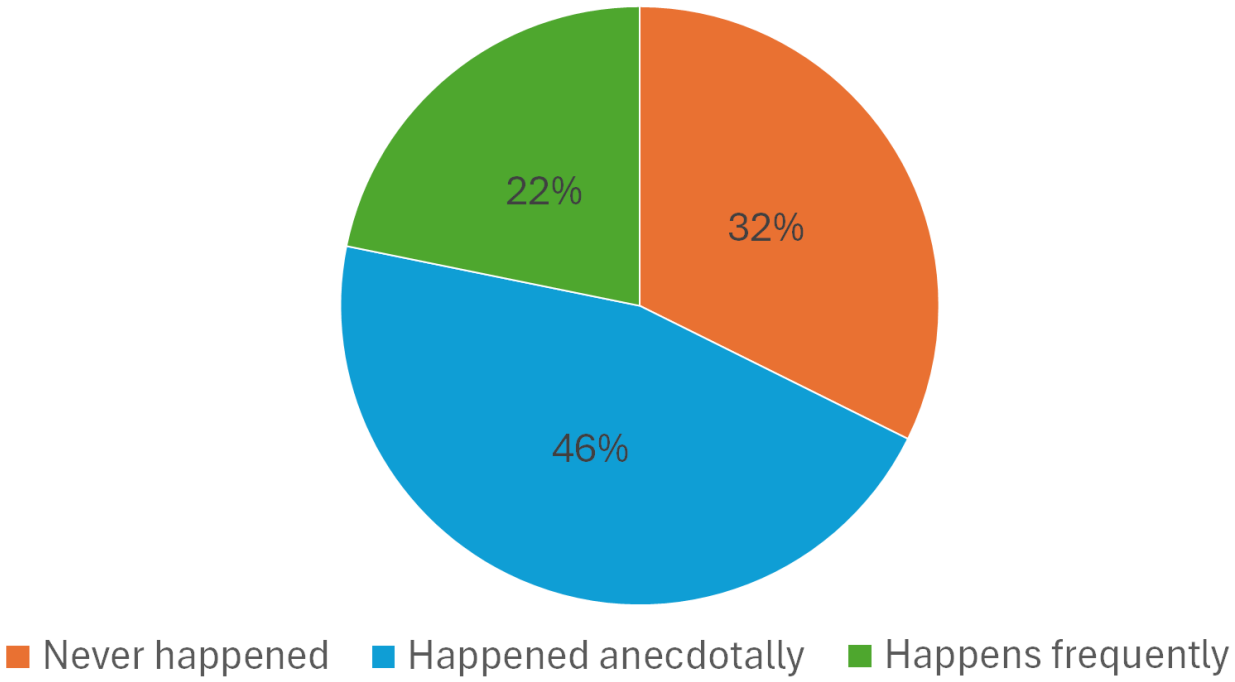
Your inputs

Your struggles with coding



Your inputs

Your struggles with coding



What never happens

- I don't remember in which order I should re-run my syntax files
- I use the wrong version of a syntax file

Your inputs

Your struggles with coding

What anecdotally happens

- I have difficulties replicating a given result (70%!)
- I don't remember why I coded something the way I did
- My code is not working because some files have been moved
- I need to update several syntax files when I make a change in the data preparation

Your inputs

Your struggles with coding

What anecdotally happens

- I have difficulties replicating a given result (70%!)
- I don't remember why I coded something the way I did
- My code is not working because some files have been moved
- I need to update several syntax files when I make a change in the data preparation

What frequently happens

- The folder(s) where my code is saved are messy
- My code works but I would not share it as it is

Your inputs

Your solutions

Half of you tried to do something about it!

"Thoroughly commenting my code"

"Organise my code better and add comments to every step"

"writing notes/log/readme file, writing better comments in script"

"A variety of improvements in the coding process, every project I do the processes (I hope) improve somewhat. Things like: folder structure, file names, extrapolating functions to a separate syntax file, informative data codebooks"

Your inputs

Your solutions

Half of you tried to do something about it!

> *"Thoroughly commenting my code"*
>
> *"Organise my code better and add comments to every step"*
>
> *"writing notes/log/readme file, writing better comments in script"*
>
> *"A variety of improvements in the coding process, every project I do the processes (I hope) improve somewhat. Things like: folder structure, file names, extrapolating functions to a separate syntax file, informative data codebooks"*

➥ Insights from computer science?

Gentzkow, Matthew and Jesse M. Shapiro. 2014. Code and Data for the Social Sciences: A Practitioner's Guide. University of Chicago mimeo.

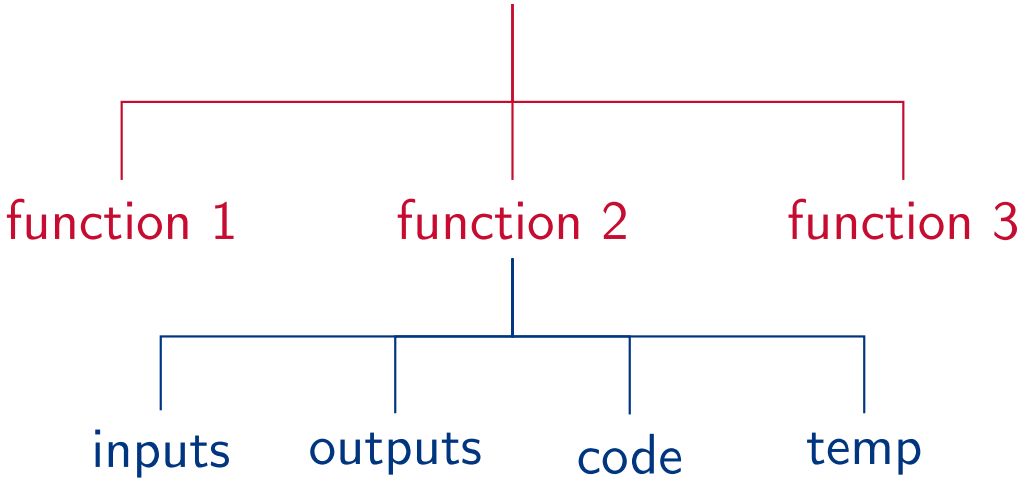
Outline

Golden rules

1. Data and code **organisation**:

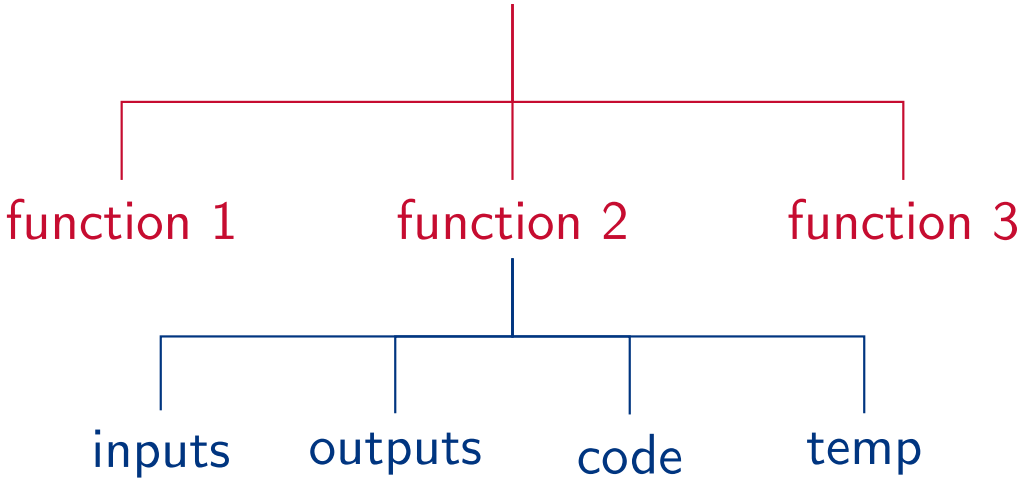
Golden rules

1. Data and code **organisation**: make it modular



Golden rules

1. Data and code **organisation**: make it modular



- Make directories portable: relative paths
- Each directory is controlled by a single master script

Golden rules

2. Write a single script that executes all code from beginning to end.

Golden rules

2. Write a single script that executes all code from beginning to end.
3. Use a **version control** software to track successive versions of the code

   $\neq$ date and authorship tags (e.g., `analysis17112022ML.py`)

Golden rules

2. Write a single script that executes all code from beginning to end.
3. Use a **version control** software to track successive versions of the code

   $\neq$ date and authorship tags (e.g., `analysis17112022ML.py`)

"You didn't spend six years in grad school so you could type in today's date all over the place."
(Gentzhow and Shapiro 2014, 13)

Golden rules

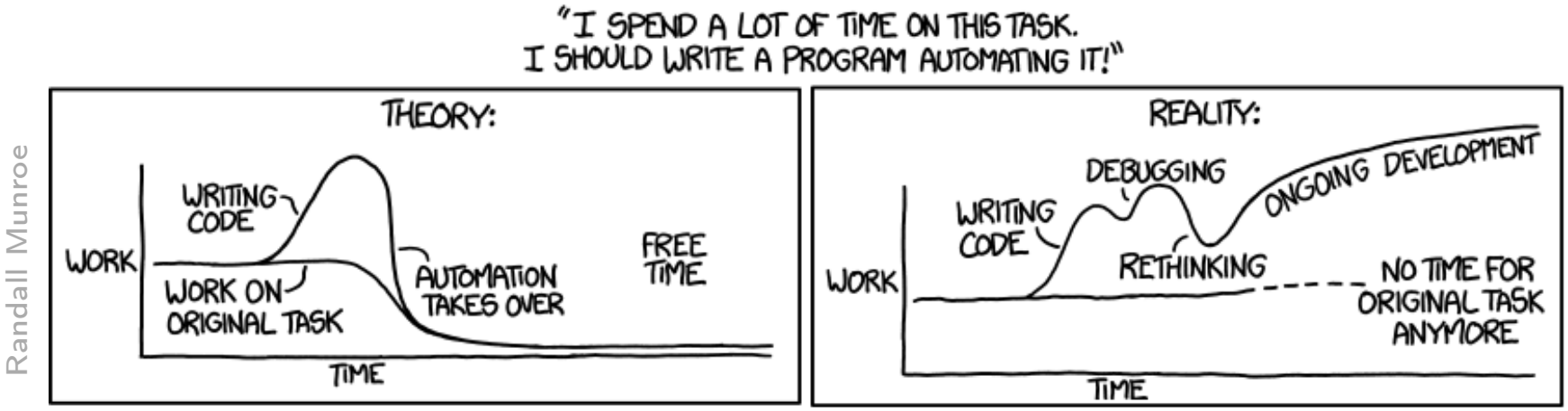
2. Write a single script that executes all code from beginning to end.
3. Use a **version control** software to track successive versions of the code

   $\neq$ date and authorship tags (e.g., `analysis17112022ML.py`)

"You didn't spend six years in grad school so you could type in today's date all over the place."
(Gentzhow and Shapiro 2014, 13)

✓ Complete run of the directory before checking in
(two times the charm!)

Golden rules

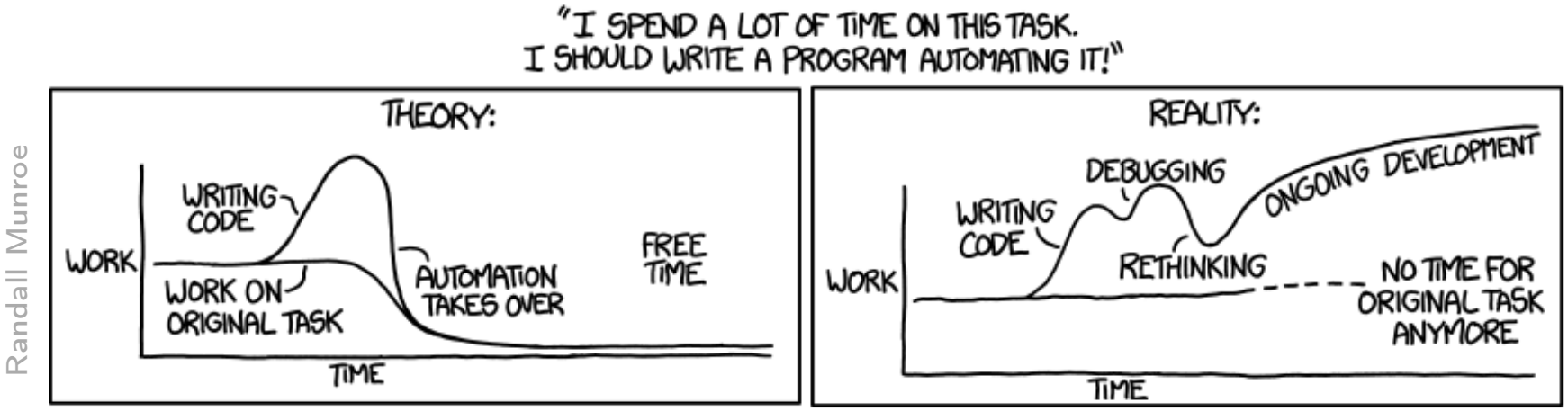


Randall Munroe

4. **Abstract** to eliminate redundancy and/or improve clarity. Otherwise, don't abstract!

[2] See Sandi Metz's blog post "The Wrong Abstraction".

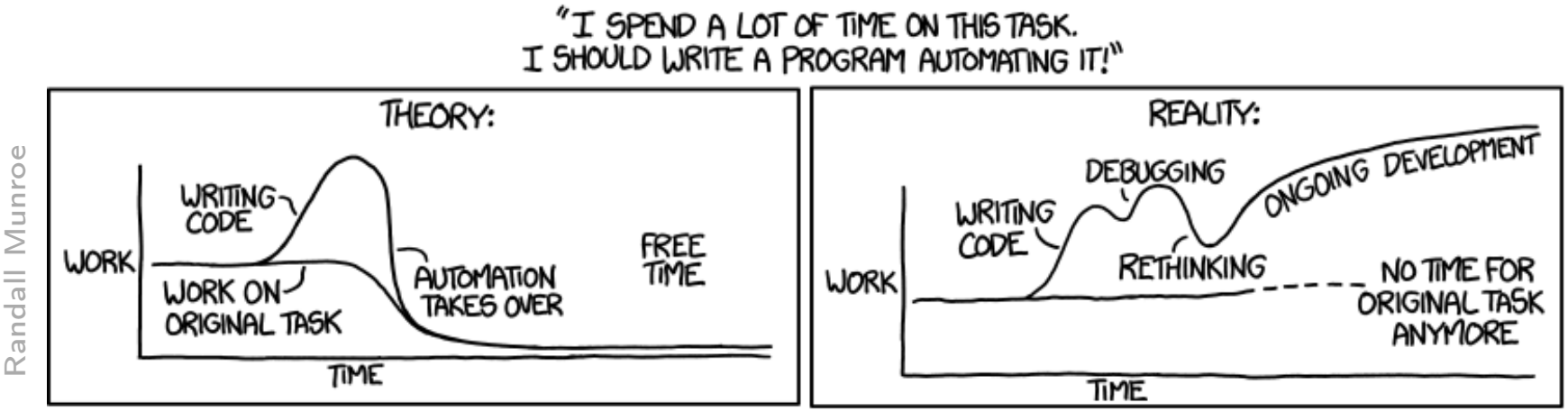
Golden rules



4. **Abstract** to eliminate redundancy and/or improve clarity. Otherwise, don't abstract!
   - ✗ code clones → `Don't repeat yourself (DRY)`
   - ✓ `Avoid hasty abstractions (AHA)`

[2] See Sandi Metz's blog post "The Wrong Abstraction".

Golden rules

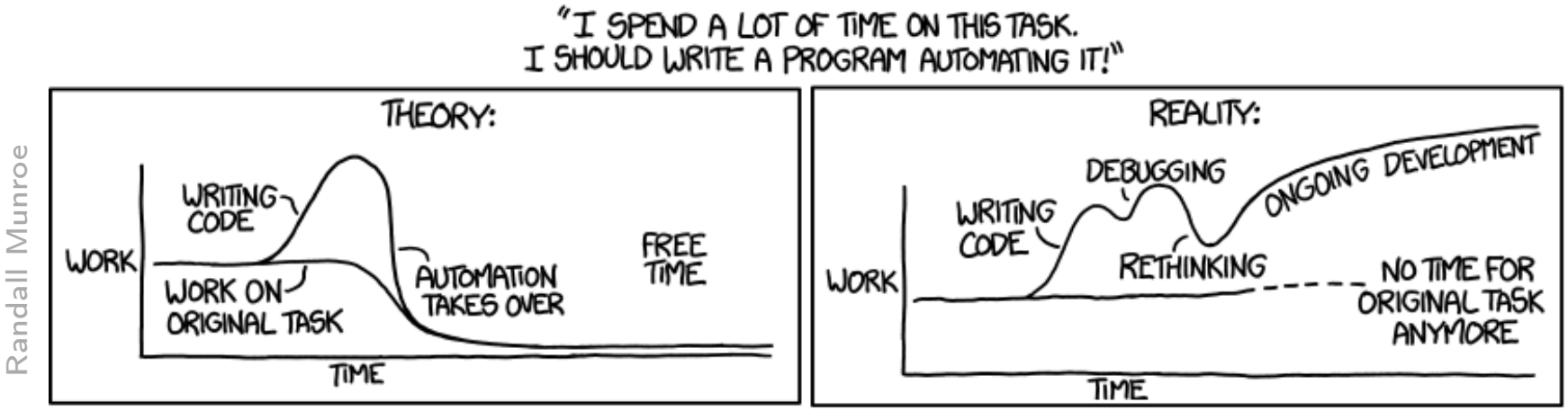


Randall Munroe

4. **Abstract** to eliminate redundancy and/or improve clarity. Otherwise, don't abstract!
   - ✗ code clones → `Don't repeat yourself (DRY)`
   - ✓ `Avoid hasty abstractions (AHA)`
     - ↪ "Duplication is far cheaper than the wrong abstraction"[2]

[2] See Sandi Metz's blog post "The Wrong Abstraction".

Golden rules



Randall Munroe

4. **Abstract** to eliminate redundancy and/or improve clarity. Otherwise, don't abstract!
   - ✗ code clones → `Don't repeat yourself (DRY)`
   - ✓ `Avoid hasty abstractions (AHA)`
     - ↪ "Duplication is far cheaper than the wrong abstraction"[2]
   - ✓ unit testing + document

[2] See Sandi Metz's blog post "The Wrong Abstraction".

Golden rules

5. **Don't write documentation you will not maintain!**

   $\rightarrow$ source of contraction between code and comments

Golden rules

5. **Don't write documentation you will not maintain!**
   - → source of contraction between code and comments
   - → the code should be self-documenting

Golden rules

5. **Don't write documentation you will not maintain!**
   - → source of contraction between code and comments
   - → the code should be self-documenting
     - ↪ look for *executable documentation*
       - assertions:

```
def custom_function(x):
    if x<=0:
        raise ValueError("Value x should be strictly positive")
    else:
        return np.log(x)+1/np.exp(x)
```

Golden rules

5. **Don't write documentation you will not maintain!**
   - → source of contraction between code and comments
   - → the code should be self-documenting
     - ↪ look for *executable documentation*
       - assertions
       - type hinting / checking (`mypy`):

```
def update_df(df: pd.DataFrame,
              clf: str,
              acc: float,
              split:float = 0.5,
              remarks: List[str] = []
              ) -> pd.DataFrame:
```

Golden rules

5. **Don't write documentation you will not maintain!**
   - → source of contraction between code and comments
   - → the code should be self-documenting
     - ↪ look for *executable documentation*
       - assertions
       - type hinting / checking (`mypy`):
   - ✓ comment on what is counter-intuitive or may cause incorrect behavior

Golden rules

6. Improve your **code style**
   - make it short and intuitive

Golden rules

6. Improve your **code style**
   - make it short and intuitive
   - use descriptive names (pronounceable, searchable,)

Golden rules

6. Improve your **code style**
   - make it short and intuitive
   - use descriptive names (pronounceable, searchable,)
   - define constants instead of using "magic numbers"

```
# Not recommended
def roll_dice():
    return random.randint(0, 4)
            # what is 4 supposed to represent?

# Recommended
DICE_SIDES = 4

def roll_dice():
    return random.randint(0, DICE_SIDES)
```

Golden rules

6. Improve your **code style**
   - make it short and intuitive
   - use descriptive names (pronounceable, searchable,)
   - define constants instead of using "magic numbers"
   - break complicated calculations into pieces

Golden rules

6. Improve your **code style**
   - make it short and intuitive
   - use descriptive names (pronounceable, searchable,)
   - define constants instead of using "magic numbers"
   - break complicated calculations into pieces
   - separate fast from slow code

Keep in mind

The sunk cost fallacy

"The sad truth is that the more complicated and incomprehensible the code, i.e. the deeper the investment in creating it, the more we feel pressure to retain it"[3]

[3] Sandi Metz's blog post "The Wrong Abstraction"

Keep in mind

The sunk cost fallacy

"The sad truth is that the more complicated and incomprehensible the code, i.e. the deeper the investment in creating it, the more we feel pressure to retain it"[3]

↪ ***Refactoring*** is key: need to constantly improve the internal structure of the code, without changing its behavior (Fowler, 1999)

[3] Sandi Metz's blog post "The Wrong Abstraction"

Keep in mind

The sunk cost fallacy

"The sad truth is that the more complicated and incomprehensible the code, i.e. the deeper the investment in creating it, the more we feel pressure to retain it"[3]

↪ ***Refactoring*** is key: need to constantly improve the internal structure of the code, without changing its behavior (Fowler, 1999)

- remove unnecessary code
- improve variable names
- add loops or functions

[3] Sandi Metz's blog post "The Wrong Abstraction"

Keep in mind

Trade-off between time spent coding error-handling and time spent debugging

Keep in mind

Trade-off between time spent coding error-handling and time spent debugging

Trade-off between storage and CPU time

Keep in mind

Trade-off between time spent coding error-handling and time spent debugging

Trade-off between storage and CPU time

- efficient formats for storing: **serialize**, e.g. XML, JSON, BSON, pickle (Python), rds (R).

Keep in mind

Trade-off between time spent coding error-handling and time spent debugging

Trade-off between storage and CPU time

- efficient formats for storing: **serialize**, e.g. XML, JSON, BSON, pickle (Python), rds (R).

Caveat beware of running memory (RAM)

Keep in mind

Trade-off between time spent coding error-handling and time spent debugging

Trade-off between storage and CPU time

- efficient formats for storing: **serialize**, e.g. XML, JSON, BSON, pickle (Python), rds (R).

Caveat beware of running memory (RAM)

- use memory-efficient data structures, e.g. arrays in Python

Keep in mind

Trade-off between time spent coding error-handling and time spent debugging

Trade-off between storage and CPU time

- efficient formats for storing: **serialize**, e.g. XML, JSON, BSON, pickle (Python), rds (R).

Caveat beware of running memory (RAM)

- use memory-efficient data structures, e.g. arrays in Python
- manually clean your environment if necessary (`gc.collect()`)

Outline

GitHub

Why using Github?

PhD comics

GitHub

Why using Github?

An version control tool to...

- ✓ ..track every change in your code
- ✓ ...make sure you can always replicate your results
- ✓ ...collaborate effectively

What is GitHub?

Quick history:

- Open Source community
- Created in 2007
- Based on Git (Linux)

What is GitHub?

Quick history:

- Open Source community
- Created in 2007
- Based on Git (Linux)
- A property of Microsoft since 2018
- Competitors: Bitbucket, GitLab and SourceForge

Figures in 2022

- ☆ 83 million developers
- ☆ more than 200 million repositories
- ☆ including at least 28 million *public* repositories

Key advantages of Github

1. Effective version control & issue tracking tool
2. Facilitates collaborations
3. A web-based repository hosting service

A web-based repository hosting service

GitHub is a community!

Search for a package (e.g. biterm model), raise issues

Example: `maximtrp/bitermplus`

A web-based repository hosting service

GitHub is a community!

Search for a package (e.g. biterm model), raise issues

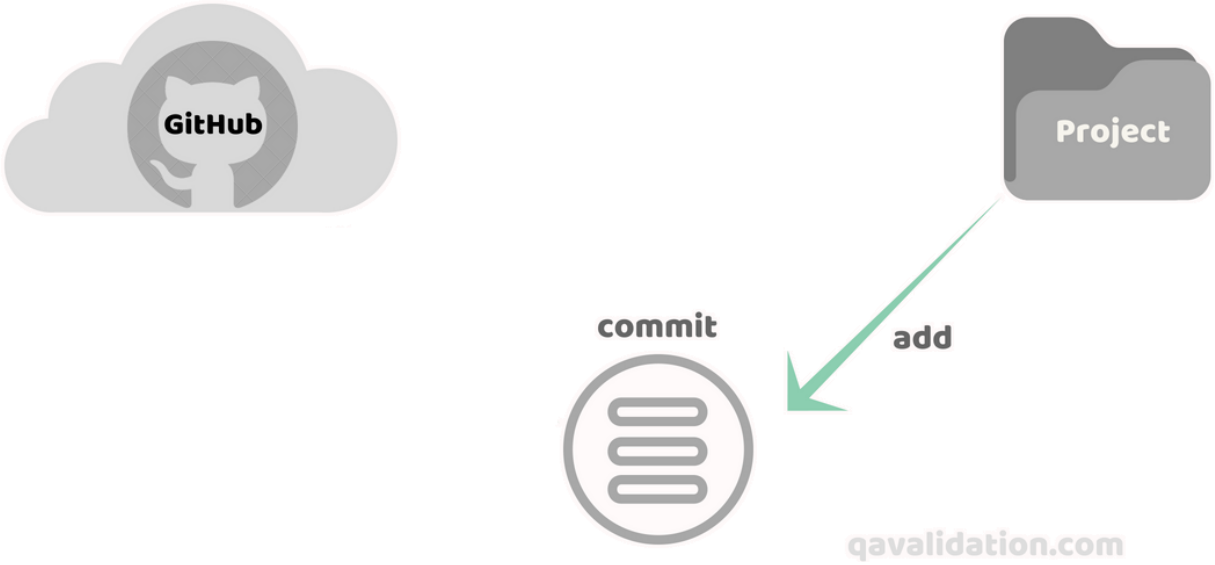
Example: `maximtrp/bitermplus`

- ✓ Date of creation and update frequency
- ✓ Quality of the code & documentation
- ✓ Are the author(s) responsive?

Outline

A version control & issue tracking tool

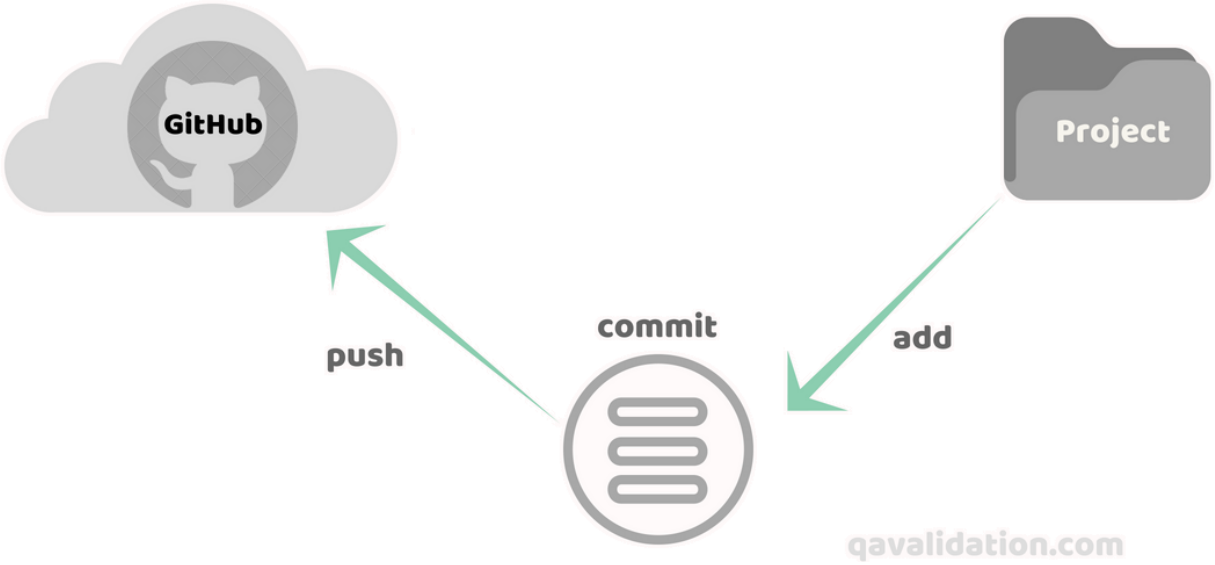
A version control & issue tracking tool



Commit (log of) changes in your local directory

- ✓ Each commit is assigned a unique ID (SHA)
- ✓ Easy comparison of scripts

A version control & issue tracking tool



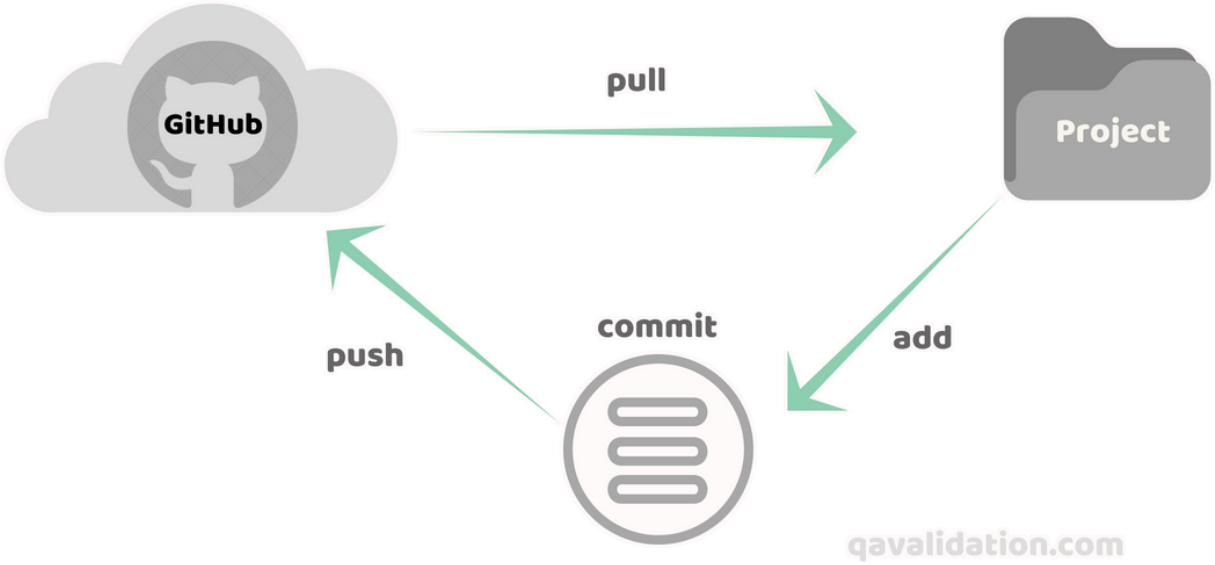
Commit (log of) changes in your local directory

- ✓ Each commit is assigned a unique ID (SHA)
- ✓ Easy comparison of scripts

Push commits into Github

- ✓ Remote version: others can access them

A version control & issue tracking tool



Commit (log of) changes in your local directory

- ✓ Each commit is assigned a unique ID (SHA)
- ✓ Easy comparison of scripts

Push commits into Github

- ✓ Remote version: others can access them

Illustration: using GitHub for version control

- Create a new repository on GitHub.com
- Clone the repository and links it to a local folder using GitHub Desktop
- Add some code on the local branch
- Commit the code using GitHub Desktop
- Push the code to the remote repository using GitHub Desktop

Extra remarks on creating a repository

License

Choose an Open Source Initiative (OSI)-approved license,
e.g. **BDS/MIT**
Need help? `https://choosealicense.com/` and Morin et al. (2012)

Extra remarks on creating a repository

License

Choose an Open Source Initiative (OSI)-approved license,
e.g. **BDS/MIT**
Need help? `https://choosealicense.com/` and Morin et al. (2012)

git.ignore

Beware of data and large files!

Extra remarks on creating a repository

License

Choose an Open Source Initiative (OSI)-approved license,
e.g. **BDS/MIT**
Need help? `https://choosealicense.com/` and Morin et al. (2012)

git.ignore

Beware of data and large files!

Other parameters

Add collaborator(s), add rules, manage pull requests

Practice: using GitHub for version control

Your turn!

- Create a new repository on GitHub.com
- Clone the repository and links it to a local folder using GitHub Desktop
- Add some code on the local branch
- Commit the code using GitHub Desktop
- Push the code to the remote repository using GitHub Desktop

Next step: creating branches

Branches enable you to develop code without affecting the rest.
$\rightarrow$ It is the parallel version of a repository that does not affect the main branch

Next step: creating branches

Branches enable you to develop code without affecting the rest.

$\rightarrow$ It is the parallel version of a repository that does not affect the main branch

$\neq$ **Fork**: create a new repository

Next step: creating branches

Branches enable you to develop code without affecting the rest.
$\rightarrow$ It is the parallel version of a repository that does not affect the main branch

$\neq$ **Fork**: create a new repository

Advantages:

- ✓ develop new code
- ✓ only update your previous code when you are sure
- ✓ suitable for collaboration

Outline

GitHub Desktop vs. command line

Slides made by Dieuwke Zwier

Two alternative ways to interact with Git(Hub)

→ All Git commands we used (and more) can *also* be executed on the command line

→ **Trade-off**: user-friendly GUI vs. speed, customisation and advanced features

GitHub Desktop vs. command line

Slides made by Dieuwke Zwier

Two alternative ways to interact with Git(Hub)

→ All Git commands we used (and more) can *also* be executed on the command line

→ **Trade-off**: user-friendly GUI vs. speed, customisation and advanced features

First-time setup: set user name & email and SSH authentication

Version control using the command line

Clone remote repository

`git clone "<URL>"` (navigate to desired directory first!)

Version control using the command line

Clone remote repository

`git clone "<URL>"` (navigate to desired directory first!)

Make changes to remote repository

1. `git status`: check status of local repository
2. `git add .`: add (all) changes to staging area
3. `git commit -m "<commit message>"`: save staged changes in local repository
4. `git push`: push changes to remote repository

Version control using the command line

Clone remote repository

`git clone "<URL>"` (navigate to desired directory first!)

Make changes to remote repository

1. `git status`: check status of local repository
2. `git add .`: add (all) changes to staging area
3. `git commit -m "<commit message>"`: save staged changes in local repository
4. `git push`: push changes to remote repository

Pull changes from remote repository

`git pull`

Version control using the command line

Clone remote repository

`git clone "<URL>"` (navigate to desired directory first!)

Make changes to remote repository

1. `git status`: check status of local repository
2. `git add .`: add (all) changes to staging area
3. `git commit -m "<commit message>"`: save staged changes in local repository
4. `git push`: push changes to remote repository

Pull changes from remote repository

`git pull`

Add `--help` to any command to learn more

Outline

Facilitating collaborations

i. Everyone works on their own **forks**

ii. **Pull request:** merge later to the master branch after validation

   ↪ Help to manage merge conflicts

iii. You can **pull** the new master branch to your local copy

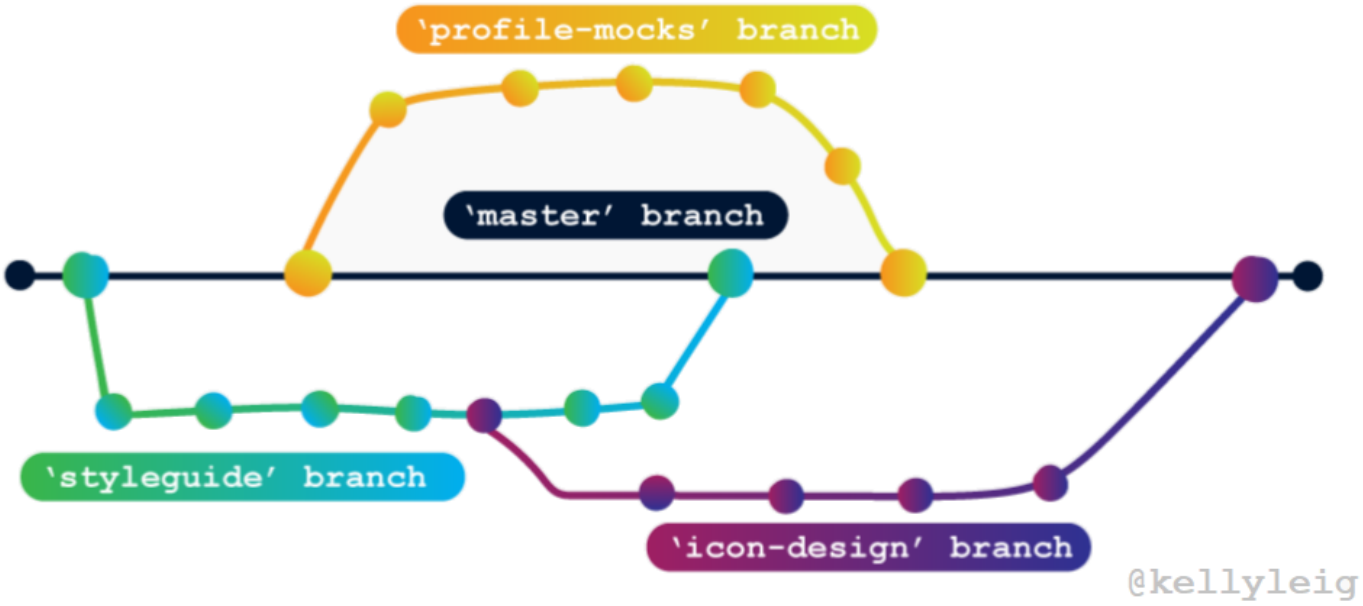


Illustration: collaborating with Github

- *A* creates a new project with some initial code
- *B* pulls the repository and works on a fork to change the code
- *B* makes a pull request to update the project when ready
- *A* reviews the changes and approves/rejects the change

Practice: collaborating with Github

Practice in pairs using one of the repositories you created previously!

Follow the steps:

1. *A* creates a new project with some initial code
2. *B* pulls the repository and works on a fork to change the code
3. *B* makes a pull request to update the project when ready
4. *A* reviews the changes and approves/rejects the change

Miscellaneous

- GitHub automatically merges differences between branches, but **merge conflicts** occur if people change the same lines of code or delete files that are used by others
  $\hookrightarrow$ you need to manually review these conflicts before merging
- Is your repository up to date?
  Don't forget to **sync your fork** and **pull**!
- **Collaborative function** to have equal rights
  $\hookrightarrow$ use feature branches and create **branch protection rules**
- Use **issues** to flag problems but also to divide work among collaborators

Other useful tools

✗ GitHub is often not accessible in secured data environments (e.g., CBS remote access)

Other "post-coding" options:

- Open Science Framework (OSF)
- Harvard DataVerse
- Zenodo
- ODISSEI code library for LISS/register data
- Computational reproducibility with fake data, offline GitHub workflow

Example: Eva Zschirnt's replication material on OSF

Interoperability

Combine GitHub with other platforms!

Zenodo	create a DOI for your GitHUb repository
Figshare	
OSF	link a repository to your OSF project
Jekill	create and host a website on GitHub

Conclusion

- ✓ Improving your coding practices is an ongoing and never-ending process!
- ✓ You can save a lot of time and energy in the long run
- ✓ Adopt best practices and convince your co-authors!

Useful resources

- Software Carpentry
- Gentzkow, Matthew and Jesse M. Shapiro (2014) Code and Data for the Social Sciences: A Practitioner's Guide. University of Chicago mimeo.
- Riverol et al. (2016) Ten Simple Rules for Taking Advantage of Git and GitHub. *PLoS Comput Biol* 12(7).
- Wilson G, Aruliah DA, Brown CT, Chue Hong NP, Davis M, Guy RT, et al. (2014) Best Practices for Scientific Computing. *PLoS Biol* 12(1).
- Morin, A., Urban, J., & Sliz, P. (2012). A Quick Guide to Software Licensing for the Scientist-Programmer. *PLoS Comput Biol*, 8(7), e1002598.
- Chacon, S. and Straub, B. (2014). Pro Git. Apress.
- Python style guide

References I

Fowler, M. (1999). *Refactoring: improving the design of existing code*. The Addison-Wesley object technology series. Addison-Wesley, Boston, 28. printing edition.

Liu, D. M. and Salganik, M. J. (2019). Successes and Struggles with Computational Reproducibility: Lessons from the Fragile Families Challenge. *Socius: Sociological Research for a Dynamic World*, 5:237802311984980.

Morin, A., Urban, J., and Sliz, P. (2012). A Quick Guide to Software Licensing for the Scientist-Programmer. *PLoS Computational Biology*, 8(7):e1002598.