Open reproducibility tools - git and lab notebooks

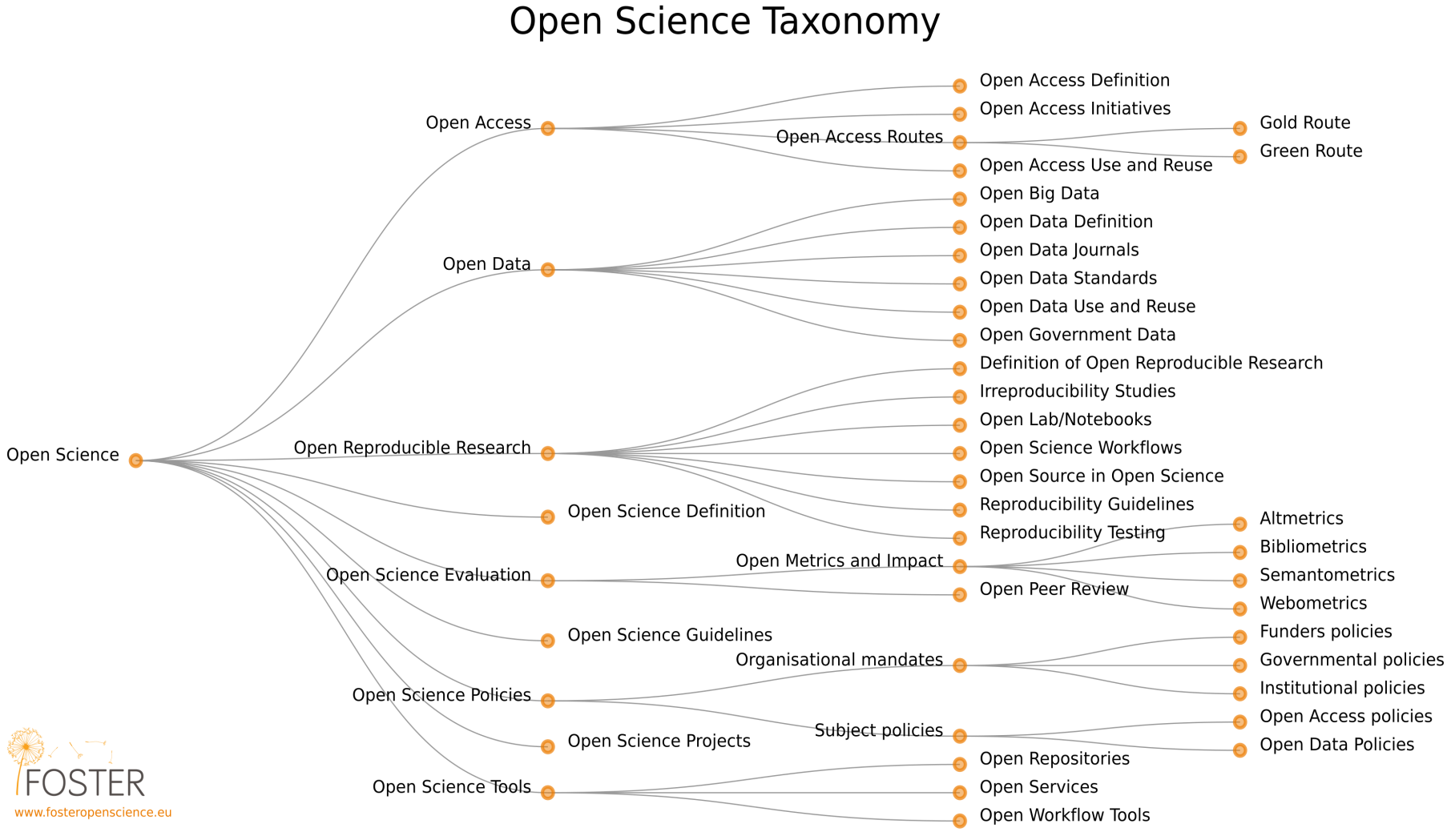
Mark Einon, Systems Manager

DPMCN / Centre for Neuropsychiatric Genetics & Genomics

Cardiff University

einonm@cardiff.ac.uk

What is Open Reproducible Research?



https://www.fosteropenscience.eu

What is Open Reproducible Research? (Cont.)

Open science involves making scientific methods, data, and outcomes available to everyone. It can be broken down into several parts, including:

- Transparency in data collection, processing and analysis methods, and derivation of outcomes
- Publicly available data and associated processing methods
- Transparent communication of results

Reproducible science is when anyone (including your future self) can understand and perform the steps of an analysis, applied to the same or even new data - achieving the same results.

Open reproducible science emerges from the adoption of open workflows that allow methods to be easily shared, allow collaborations with others and enable data and pipelines to be openly shared, thereby contributing to greater scientific knowledge.

From https://www.earthdatascience.org/courses/intro-to-earth-data-science/open-reproducible-science/get-started-open-reproducible-science/

Also see https://lgatto.github.io/open-and-rr-2/

Why is open reproducible research being encouraged?

- A **replication crisis** has been found to exist and open science is seen as a solution, of which open reproducible research is a part.

- Many findings were found to be doubtful [1], due to:
    - Generation of new data/publications at an unprecedented rate.
    - Compelling evidence that the majority of these discoveries will not stand the test of time.
    - **Causes: failure to adhere to good scientific practice and the desperation to publish or perish**
    - This is a multifaceted, multi-stakeholder problem.
    - No single party is solely responsible, and no single solution will suffice.

1. https://en.wikipedia.org/wiki/Replication_crisis

The story so far...

- For a long time, it has been possible to be a top researcher in the field without ever having shared working, reliable code.
- Funders would still give you money
- Publishers would still publish without working code
- Peer reviewers would not insist on having verifiable code
- Unlikely that anyone would ask to repeat your findings using your scripts (also easy to fob off)
- PhDs happen without this sort of collaboration - the code probably wasn't reviewed, shared or run by anyone else, so the relevant collaboration skills are not taught.

Barriers to open reproducible research

- Impression it takes more time
- Little incentive to provide evidence of your own mistakes
- Additional skills to learn
- Change of mindset/culture required
- Activities do not contribute to promotion (unlike grants and papers)

1. https://the-turing-way.netlify.app/reproducible-research/reproducible-research

The situation is changing...

Researchers are being increasingly challenged to practice open science that goes beyond open access and open data, extending all the way to making software and data research outputs truly repeatable and reproducible.

Funders[1], journals[2] and REF[3] are making strong recommendations and sometimes mandatory policy to instil an open science culture and adopt practices successfully established in open source communities, and the rate of such activities is increasing.

These practices aim to ensure software has less defects which is understandable by other researchers and produce results using repeatable pipelines.

1. https://wellcome.org/grant-funding/guidance/how-complete-outputs-management-plan
2. https://www.nature.com/documents/GuidelinesCodePublication.pdf
3. https://www.ukri.org/our-work/supporting-healthy-research-and-innovation-culture/open-research/

Meeting open reproducible research requirements

Open reproducible research is increasingly becoming a requirement to obtain funding and publish research. To ensure open reproducible research, we may need to provide:

- Source code
- Documentation
- Executables
- Software dependencies
- Narratives of what the code is doing
- Narratives of whole workflows
- Input and output data
- Open licensing

1. https://www.software.ac.uk/blog/2018-05-22-sharing-reproducible-research-minimum-requirements-and-desirable-features

2. https://software.ac.uk/resources/guides/choosing-open-source-licence

Open reproducibility tools

- Git, Gitlab, Github, Jupyter and Rmarkdown are a few of the many tools that facilitate open reproducibility
    - As important to know why they are used as how to use them
    - Tools can change between projects and over your career

- Github is a fancy way of displaying information from the git tool
    - Git is a command line tool, where the real power & usefulness lies
    - Another alternative to the proprietary Github is the open source Gitlab

- Rmarkdown is specific to the R language, Jupyter notebooks are also popular and can be in R, python, bash and dozens of other languages
    - Both fully functioning data science lab notebooks that work in unison with git
    - Open source and widely used - aim to get exposure to both

- Zenodo is one service that allows a DOI to be attached to a version of a git repository
    - Figshare and OSF.io are some other alternatives

Git

- Git is a version control & complex communication tool that is the bread-and-butter of data science best practice
- It is essentially a command line tool with other web-based interfaces available, e.g. GitHub & GitLab
- Git fluency is currently a basic necessity for producing reproducible data science
- It is quick to understand and use basic git features, but does have a learning curve for more powerful features
    - every change to a set of text-based files has it's own ID and log message
    - ability to revert to any previous version or see differences between versions
    - gtilab/github give free, online cloud storage for your code
    - easier to get support in tackling issues in your project

Learning git

- Software Carpentry have a well-pitched comprehensive introductory course at https://swcarpentry.github.io/git-novice/ (https://swcarpentry.github.io/git-novice/) (3-4 hrs)
- Git is a tool (mechanism) and doesn't dictate a process (policy) on how to use it
- Git can be used by a sole developer, a team or an entire community
- Git is like chess, rules easy to learn but competency comes from practising
- Always choose/define a process! Having a best-practice process has a greater bearing on quality than the people involved
    - (Code peer review is the next most fruitful best practice a project could adopt)

1. https://sdlambert.github.io/2015/04/09/git-workflow-for-solo-development/ git workflow for solo development

2. https://osf.io/4yp9a/?action=download Curating Research Assets: A Tutorial on the Git Version Control System

3. https://git-scm.com/book/en/v2/GitHub-Contributing-to-a-Project GitHub - contributing to a Project

4. https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005510 Good enough practices in scientific computing

Example sole developer git workflow

Setup a local repository with a remote copy

- Initialise a local git repository
- Create a remote repository
- Create a git repo README.txt & LICENCE.txt
- Add the README, LICENCE and any existing files to the local repository, ensuring the git commit message follows best practice
- Push the local committed changes to the remote repository

Make and version a change

- Edit some files in the local repository, making changes to do one complete, logical thing only. E.g. fix a bug, add a feature or some tests
- Test the changes, ensuring the code runs with no regressions
- Add the new file modifications to the local repository, ensuring the git commit message follows best practice
- Push the local committed changes to the remote repository

Example sole developer git workflow (Cont.)

Make a big change

- Create a git branch in which to develop the desired change
- Make and commit several logical changes to develop the big change
- Test the big change, ensuring the code runs with no regressions
- Rebase branch onto the HEAD of master
- Push the local committed changes to the remote repository

Shared team workflow

This would use branches more extensively and adopt peer review as a gateway to getting changes merged into the shared branch/repository

Demonstration

- Set up a git repo. Push to gitlab and github

Data science notebooks

- These are plain-text documents that mix markdown narrative with code and optionally the code output, which is often graphs and tables
    - A plain-text format is essential for git to be able to manage changes effectively
- They are split into a series of 'cells' of markdown or code. The code can be executed in-situ and exported to html, pdf, LaTeX, slideshows, dashboards and other formats
- As an alternative to code scripts, they can also record and provide a narrative of an experiment and allow interaction with all parts of an analysis, not just the results
- Along with git, notebooks facilitate peer review of analysis code
- Jupyter notebooks are the most widely used notebook, with Rmarkdown more popular in some fields
- These slides were created in a Jupyter Notebook!

1. https://www.datacamp.com/community/blog/jupyter-notebook-r
2. https://www.datacamp.com/community/tutorials/tutorial-jupyter-notebook

Rmarkdown

- Rmarkdown is technically non-interactive, but the newer R notebooks add this functionality to mimic Jupyter notebooks in this regard - so that cells can be edited and run in the same document interactively
- RStudio (via the knitr package) takes the Rmarkdown (.Rmd) file, containing only code and markdown and produces a .html file with code, markdown, when executed as a 'batch' style operation
    - In comparison, Jupyter uses .ipynb as default recording all three, and the jupytext package allows just the code and markdown to be stored as a .py/.R/.java file
    - Two separate files allow analysis steps to be stored in git without having to manage differences in runs with different results
- Markdown itself is a simple plain text format with some easy formatting characters for headings, bold/italics, tables etc.
    - The flavour of markdown does vary between R, Jupyter, wikis etc - be careful!

Demonstration

- Create an Rmarkdown doc and push to the repos

Jupyter notebooks

- Jupyter is a web-based application that allows editing of Jupyter notebook files, stored as plain text with an .ipynb extension, containing markdown, code and output
    - Stored in plain text JSON format
    - Use the jupytext package to also save a .R/.py/.sh file with just plain text code and markdown
- Code cells can be run independently in any order interactively (watch out!), and saved at any time
    - Unlike the static R knitr package, Jupyter has a language specific 'kernel' constantly running in the background that executes the code, and keeps state of variables and data frames etc, much like an interactive shell
- As with git, R & Jupyter notebook skills come from practice using them

Demonstration

- Print a pdf of the notes & push to the repos
- Create a DOI

Further best practices that could be adopted

- Remove manual analysis steps - put into code form and automate
- Version control of code & associated documents (e.g. licence/guides)
- Share discoverable workflows/code - use a gitlab group
- Use 'Agile' checklists, guides and workflow documentation - light touch and generic
- Pipeline validation & testing
- Project issue trackers
- Peer review of code
- Ensuring feedback loops exist to improve practices - development processes are projects themselves
- Foster a community of practitioners