Data Management and Software Development Skills for Fusion CDT Researchers

Tobias Slade-Harajda

12th - 13th July 2023

Day 1

Definitions

- Meta-data = data about data
- e.g. for a photo, the image would be the data, and the meta-data would be it's location, time taken, resolution etc. stored along with the image
- EPSRC-funded research data is a public good produced in the pulic interest - we have to share it
- Some jobs ask for examples of reproducibility practice or use of GitHub/GitLabs
- F.A.I.R principles in sharing data

FAIR Principle

Findable - can people find it?
Accessible - can I get it? (doesn't mean completely accessible by everyone)
Interoperable - can I use it? (i.e. a different file format)
Reusable - is it useful?

Ways of sharing data

- Lab notebook with analysis
- Zenodo, figshare
- Ampletracks
- Pre-prints to Arxiv
- Code repos (GitHub)

How to write a research data management plan (DMP)

A DMP is a ***living document*** which can be updated at any point when needed.

It outlines all stages of project lifecycle before data collection begins. Hence why it evolves over time.

DMP will cover how the data is...

- Stored
- Collected
- Protected
- Shared (with colleagues in group and external)
- Lincensing rights

A DMP will outline the overall plan, collection, processing, analysing, preserving, sharing and reusing.

Write a DMP if: it isn't already written, methods are changed, data storage location is changed, processing methods are changed, data storage/retention changed.

Data examples are covered by physical (experiment sample & hard-drive) and digital (figures, simulation outputs etc.)

Data Curators RDMP Guide :: `dmponline.dcc.ac.uk`

Logged in through institutional access Warwick.

Research data management (RDM) tools

File naming conventions:

- International date format YYYY-MM-DD
- Snake_case or hyphen-case
- Make a README.txt which explains the directory
- ## more examples here ##

Store data in accessible area i.e. University RDS, shared folder, OneDrive/other cloud based applications.

Electronic lab books like eLabFTW `https://www.elabftw.net` and Jupyter notebook/lab. Version control carried out in Git and stored in GitHub.

Use figshare `https://figshare.com` and Zenodo `https://zenodo.org` which gives a DOI of the repository of your data/meta-data, therefore you can provide a DOI link on your paper which links to a stored repository of the data

Open access publishing if possible (why? - more refs) - CC-BY licensing creative commons. If journal is closed access then can go for Gold/Green access journal. Arxiv can be useful because you don't need a publishers approval for it.

RDM for Computational Projects

Reproducing old results may occur for thesis corrections so RDM is extremely important.

Similarly, a new collaborator may want to join you with old research, would you have the data available?

TL;DR

- Run simulations with identifiable version of code
- Document and keep all inputs
- ## more comments ##

Record the version of the code, include the version number in the output, make it automatic (i.e. in a parameter/value summary - print it at the top).

Make a note of the code version, or make an environment where the code libraries stay the same throughout. If a library updates, it could change the results slightly. This should be managed properly within Warwick managed desktop through the module and environment system.

Keep track of code runs; the system you use may differ to others, but as long as you know which version you use and keep track of it. Nested directories named accordingly works well for a small number of changes to simulation parameters. Could also keep the directories "flat" with a README or other record of which each run was. Could even utilise an SQL database to properly order the parameters of simulations.

Post-processing should be automated as much as possible - make functions and call them for each run/simulation.

Can create a DOI for data stored on machines such as ARCHER2 and university based machines.

Zenodo can store up to 10Gb so good for figures, code and some small simulation output.

Include a statement in your papers such as that in figure **??**. ## example of statement ##

Ampletracks *(more useful for physical samples)*

Helps keep track of data records (i.e. sample obtained in experiment) and can create "child" records of the original. For example, a steel ingot which was cut in half, and then each half was processed differently. Need to input what has happened to each, but provides clear record of each even if original is no longer available. Most importantly you can create a QR code for a given sample which tags it and can be repeatedly used.

Why use Git?

Widely used by repositories for code. Can link accounts. Can link/push to Zenodo.

- Make changes
- Track progress
- Share work

Local use helps you save on storage space and keep working on one file while keeping track of the different versions and change history.

Term	*Definition*
Repository (repo)	a project under version control
History	timeline of changes to your repo
Clone	make a local copy of a repo, including its history
Commit	a labelled set of specific change to files in your repo
Push / pull	synchronise history with another copy of your repo
Working tree	the current state of your project and files on disk
Track	keep under version control

Table 1: Caption

Using Git remotely involves the collaborative use where branches are created. This helps make the workflow easier to understand, maintain the main branch and implement new features in the code.

Day 2

Using Git

Access presentation files here: `https://livewarwickac-my.sharepoint.com/:b:/g/personal/u2156745_live_warwick_ac_uk/EYqHDhkOdhhKj8YzqK7DVNwBIvQ8Qs9zCt9csh2soMlqKQ?e=g0jSw2`

Basic process of using Git is:

- Make/edit files as you normally would (`your work goes here`)
- Add files you want to upload/push using `git add <NAME OF FILES SEPARATED BY SPACES>` (can choose what you've been working on, or can use `git add --all` to add all of the changes to the next commit)
- Commit these files to a push, this is where you add a message using `git commit -m "YOUR MESSAGE HERE"`
- Push the commit to your GitHub using `git push`. On first push you'll need to add the origin of the GitHub repo using `git push --set-upstream origin <NAME OF BRANCH>`, but after that you can just use the standard git push

In this morning session we learnt how to:

- **Create a repo**sitory, or clone an existing repository
- **Make changes** in the repository **and commit** them, adding a label and a message
- **Push** the changes to GitHub to synchronize
- **Revert** commits (`git revert <first 7 digits of commit ID>`) if something has been changed that you don't want

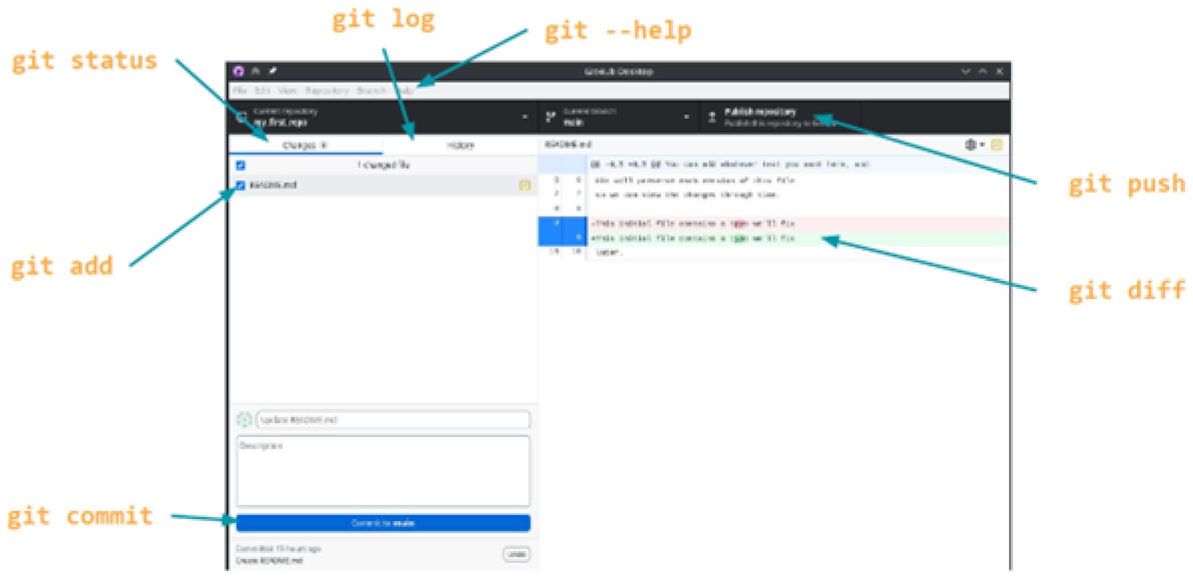

Figure 1: Git Bash commands as seen in Git-Desktop

Branch terms	*Definition*
Branch	Set of changes in your repository, being tracked in parallel to your ‘main’ branch
Merge	Brings changes from one branch into another
Conflict	Situation where competing changes have been made to some part of your repository
Pull Request	A way of reviewing, labelling, and discussing a merge before it takes place
Fork	Makes a copy linked to the original repo

Intro to Automation

Running through exercises to demonstrate the use of git, python automation of functions and use of pytest to make sure that each function behaves as expected.

The examples we’re running through (from `https://github.com/PlasmaFAIR/intro_to_automation/`) include:

- ☑ 1 - Reusable functions (on Miller surfaces)
- ☐ 2 - Packaging
- ☐ 3 - Inputs
- ☐ 4 - Testing

Can also use git push to automatically test all the functions and git will return you whether they all work and can be uploaded `https://edbennett.github.io/python-testing-ci/06-continuous-integration/index.html`. This is good because for large code you don’t need to run the tests yourself on your local machine. **Will block you from pushing broken code to your repo.**