NPL
National Physical Laboratory

NPL REPORT MS 37

GOOD PRACTICES IN SOFTWARE QUALITY FOR GIT, RELATIONSHIP WITH FAIR AND GOOD PRACTICES FOR FAIR

F.M. BROCHU

FEBRUARY 2022

Good practices in Software Quality for Git, relationship with FAIR and Good practices for FAIR

F. M. Brochu
INFORMATICS/DATA SCIENCE/SED

ISSN: 1754-2960

https://doi.org/10.47120/npl.MS37

National Physical Laboratory
Hampton Road, Teddington, Middlesex, TW11 0LW

Approved on behalf of NPLML by
Marina Romanchikova, Science Area Leader in Informatics.

CONTENTS

1 PURPOSE OF THIS DOCUMENT

This document provides guidance on how the software development tool Git can be used to improve the software quality and make your code more Findable, Accessible, Interoperable and Re-usable (FAIR). It covers good practices around using Git and provides links to NPL and external resources for integration of Git with common software tools and using Git features for software documentation.
This guide does not provide a comprehensive technical description of Git. Detailed instructions on how to install and use Git at NPL can be found here along with other NPL Gitlab resources.

2 GLOSSARY

FAIR: good practice principles for data, stands for Findable, Accessible, Interoperable, Reproducible.
SWQ: Software Quality
SWQD: Software Quality Documents
QA: Quality Assurance

3 INTRODUCTION

3.1 BACKGROUND TO GIT

Git is an open-source software "for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows (thousands of parallel branches running on different systems) (Wikipedia). It is a code repository and a version-control system like SVN and CVS before it, building on their success. It is distributed under GNU General Public License Version 2.

3.2 GITLAB

Git was released for NPL-wide usage packaged as Gitlab, an NPL-owned server to be used as a coding repository. The NPL instance had 350 active users developing over 680 projects in April 2020 and has grown significantly since.
Gitlab comes with a few useful functionalities that make software Findable and Accessible, namely a search box and easy to set access rights (either on a user-basis or NPL-wide basis). However, as Git is primarily a code repository as well as an easy way to perform collaborative code development, the Interoperability and Reusability aspects of FAIR is still up to the user to implement, by adhering to software development good practices developed by the NPL QA team.

3.3 LINK TO FAIR

This web page from the Dutch e-science centre describes 5 steps to make any software development FAIR. Step 1, "Use a publicly accessible repository with version control" is fulfilled by git and NPL Gitlab. The other steps, "Add a licence", "Register your code in a community registry", "Enable citation of your software" and

“Use a software quality checklist” are not covered by git alone, but by separate NPL software development procedures.
A companion report on FAIR practices at NPL is available here. This document describes how the FAIR principles can be applied to software.

3.4 INTEROPERABILITY/REUSABILITY FOR SOFTWARE AND GOOD PRACTICES.

Interoperability and Reusability are very hard to distinguish when it comes to software. A code is interoperable when it can run on different machines, possibly with different operating systems. A code that is interoperable is reusable.
A code that is re-usable can be rerun as is on the same machine it was developed several years later. A code that is re-usable is not necessarily interoperable, as dependencies will vary across operating systems.
NPL software development good practices already emphasise the need for software testing and maintain good documentation when it comes to running platforms and software versioning, so we are going to reformulate them in the framework of Gitlab.

4 GOOD PRACTICE IN GIT(LAB)

4.1 INTRODUCTORY MATERIAL TO GIT

This report is not designed to be a starter guide to Git, for a comprehensive introduction to Git please refer to the resources here. This report is designed to provide guidance on good practices for how to use Git to ensure software quality of your code. This report focuses on how Gitlab integrates with NPL’s digital ecosystem and corporate software.

4.2 FIRST POINT OF CALL FOR DOCUMENTATION: THE MASTER README.MD FILE

The Gitlab platform provides all projects with its own wiki and README.md files, both of which can be used for documentation. While the wiki stays on the server, the README.md file is integral to the project development and downloaded along all project code by any developer making a copy of the software (an operation called “cloning” in git jargon). An example of a root “README.md” file can be seen from the Gitlab screenshot show in Figure 1.

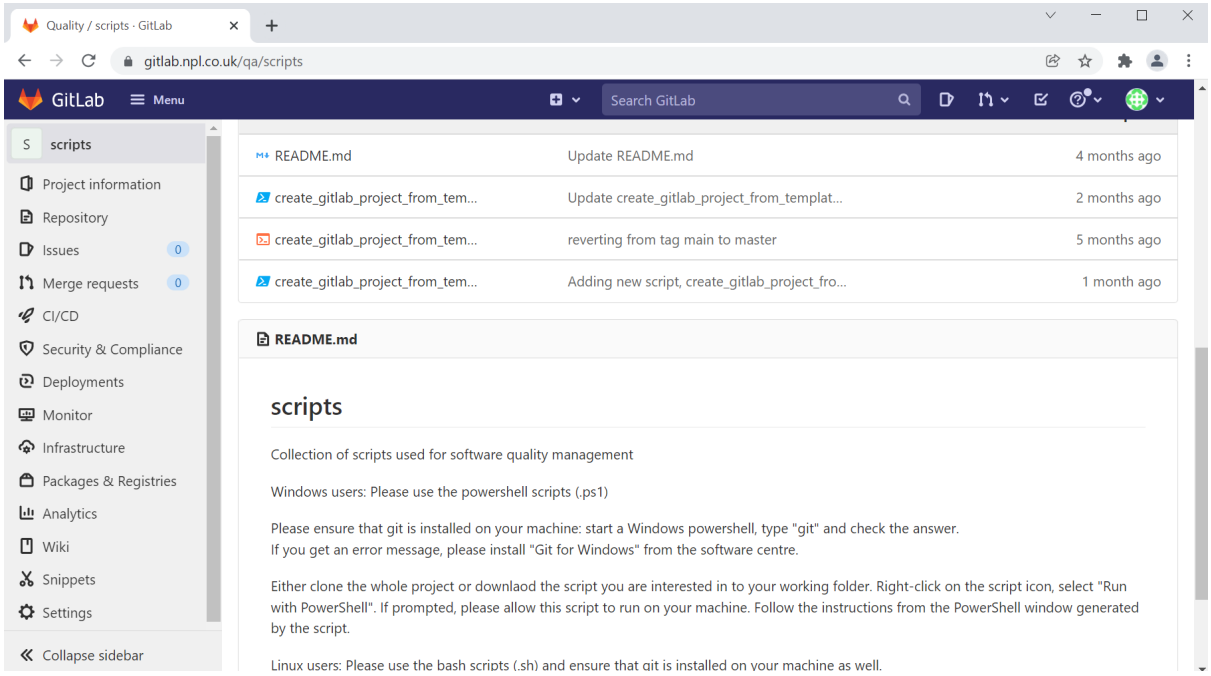

Figure 1: Snapshot of a Gitlab project main page, including file listing and root README.md file

As such, we recommend keeping the wiki for general information with respect to the software, and using the README.md file to describe:

- The requirements to run the software (operating system, name and version (e.g Windows 10), compiler/interpreter, name and version (e.g python 3.7) and all external dependencies, name and version (e.g anaconda 2.0.1, numpy 1.0.15).
- The steps needed to install the software and dependencies from scratch.

These two points are essential for Reusability: a software will be made obsolete if any of its external dependencies are no longer findable or if the relevant version is not known.

- The software functionality, the repository contents (file names and description) as requested by NPL software QA.
- If the project contains a testing suite, instructions to run it, including the description of expected input and output data format, supported protocols and other pre-requisites.

4.3 SOFTWARE QUALITY DOCUMENTATION

Software quality is an important aspect of developing code at NPL. Existing NPL procedures such as QF-59, software integrity level, user requirements and M013. The software quality documents are useful for users to understand the code's purpose, structure, functionality etc. and are an essential part of the NPL process for maintaining software quality.

Since software quality documents need to be version controlled, we generally recommend including them into the same Gitlab repository as the code. However, in certain cases such as projects with public releases, the SWQD may be too sensitive to be shared alongside the code. In such cases, we recommend to store software quality documents in a separate area with different permissions. Such separation can

be achieved by using Gitlab “modules”.

Gitlab organises projects as “folders” and allows splitting into subfolders, or “modules” of a given project. Each module has its own set of permissions, allowing to distribute work and responsibilities among developers. This functionality is useful for large scale projects and can also be used to separate software quality documents from the code.

To help users organise their Gitlab project and generate the required software quality structure, one could develop a dedicated script that will automatically provide a minimum set of software quality documents when a new Gitlab project is generate. The Time and Frequency group are using Git to generate SWQD automatically using Git, and we will aim to combine the two approaches in a future extension of this work.

4.4 VERSIONING RECOMMENDATIONS

Recording code and documentation version numbers is important for software quality. For Git it is recommended that:

- In the case of collaborative development where branching is necessary, do provide a human readable version number or “tag” via the command line git tag, once merging has been done and a collective release (internal or external) has been agreed. “Internal” release is an intermediate development step towards “external” (or full) release to the user.
- In case of single developer project, a readable version number should be allocated on versions marking a significant development milestone, to help the auditors making sense of the development strategy.

Git provides a nice way to ensure that each new version is unique, but unfortunately it means the default version number is unreadable by humans. On the other hand, traditional numbering does not make real sense when several concurrent development branches are active and can be subjective, with many different approaches which would work well. By making use of tags in Git you combine advantages of both methods.

Git version frequency

- A new version is generated by committing developments to the repository. It could be to a branch or the master repository. Each commit comes with a comments message describing the changes made It is essential (critical in the case of collaborative development) to describe what you have done in a clear, but exhaustive way, and name the files affected (as several files might change during a single commit). The comments message will be part of the project history and therefore read by the auditors, so it is important to have meaningful details.
- Because of the need for details for each commit, it is recommended having a maximum of 3 to 4 significant changes for each commit. More would make the version description hard to read.
- As a consequence of the two previous points, we recommend committing changes often in small steps.

4.5 CONTINUOUS INTEGRATION/CONTINUOUS DEVELOPMENT AND TESTING

NPL SWQ procedure leaves the testing mechanisms and procedures open to the developers. It would be useful to educate people on testing methods and approaches for Gitlab, including test-driven development (also known as CI/CD in Gitlab jargon). This process is recommended for the developers who are already experienced with Gitlab and beyond the scope of this report. For the curious, this Gitlab page will introduce the concept of CI/CD and automated testing.

5 SUMMARY RECOMMENDATIONS

This guide covered information on how to use Git version control system to improve software quality and support FAIR principles in code management. A summary of key recommendations numbered R1, R2 etc. is given below.
R1: Always create a README.md document in the top-level folder.
This document should contain at least:

- Author details and contact information
- Summary of project and/or code
- Content list
- Example use or test case
- Include software quality procedures and forms

R2: Commit the Software quality documents directly along the code in Gitlab, unless they are too sensitive for release, in which case the project should be subdivided into a public subproject containing the code, and a private subproject containing the Software Quality documents.
R3: Commit changes often and in small quantities, so the changes can be well described and easy to follow up.
R4: Use human readable tags for releases completing a milestone.

There are advanced Gitlab functionalities such as the Issues tracker for bug management, Issues board for planning, merge requests for code reviews, the CI/CD instance for automated code testing, etc. that will be worth exploring but are beyond the scope of this report. Current statistics show a very low adoption rate of these provided functionalities across NPL, showing that Gitlab is currently just used as a code repository.

6 GITLAB INTEGRATION WITH COMMON NPL SOFTWARE TOOLS

In this section we overview how Gitlab integrates with NPL supported software.

6.1 MATLAB

MathWorks provides some recipes about integration of git with MATLAB and Simulink.

6.2 LABVIEW

A description of LabView and Gitlab integration can be found here

6.3 COMSOL

This will be covered in coming NMS cross-theme development with Ed Dickinson.

6.4 PYTHON

There are a few python modules providing easy access to the Gitlab server API, allowing to do atomic operations on the project itself, like python-GitLab. Covertly, the Gitlab documentation addresses well how-to setup a testing (CI/CD) environment for project written in python.

A Yammer thread by Alvise Vianello (26/05/2021) quoted below outlines how to install python packages from GitLab via python package installer *pip* and PyPi by using this tutorial. The demo URL from the tutorial should be modified to accommodate NPL Gitlab repository structure:

```
gitlab.npl.co.uk/api/v4/projects/<project id>/packages/pypi/
```

"<project id> is the ID of the Gitlab project you are using as a Package Registry. Your packages will then appear under Packages and Registries > Package Registry, and you can setup pip to look for packages in that repository before looking on the official PyPI repository." A tutorial on configuring pip to use custom package index can be found here.

7 CASE STUDY: ROYAL FREE COVID-19 RESPONSE

In this section we provide exemplar case studies of using Gitlab for NPL projects. First, we introduce the original project, links to the Gitlab code and good practice here. Secondly, we present a generalisation of these codes in to a 'Toolkit' so that other users in NPL with similar problems can benefit from this code. In this case we also provide descriptions of good practice used here.

7.1 INITIAL PROBLEM AND SPECIFIC CODE

As part of the COVID-19 response, NPL's Data Science and Medical Physics teams worked with the Royal Free London NHS Foundation Trust (RFL) on several projects. These included:

- **Optimising hospital flow** – Managing hospital flow to separate areas of the hospital in to Covid and Covid-free areas. This was also extended to modelling the hospital flow to support optimal use of wards, rooms, beds and taking into account key requirements and constraints such as oxygen usage.
- **Optimising the cancer pathway** - Optimise the RFL cancer pathways which have interdependencies such as Radiology and Pathology, in order to meet national targets for diagnosing patients within a given time frame.
- **Analysis of Biomarkers** – Analysing coagulation markers and routine lab tests for Covid patients in order to identify possibly markers of disease severity for optimal treatment planning.

For more detailed information please visit the website case study and watch Elizabeth Cooke's presentation of this work at the Celebrating Science Lecture on

Data Science Solutions to Healthcare Problems (starting at 27:20). For each of these projects developed software consisting of multiple files and functions to read in, process, analyse and visualise data from the RFL in various formats. Code for all projects is available in NPL's Gitlab repository though access is restricted due to the sensitive nature of the data. These codes have been generalised for use in other areas in an NPL wide repository which is available here and detailed in the next section.

7.2 GENERALISATION OF CODE

The code developed in the COVID-19 response with RFL was generalised to be reused for related problems as many of the tasks are generic. The toolkit is available here and open to all NPL users. This shows several benefits of Gitlab, including code reuse across projects, branching (a testing/development branch and a released branch), version control in code development. The project contains a very detailed README.md which:

- outlines the creator and contact details
- content of the repository
- requirements to run the code
- the codes available (including a description)
- Example testing codes and their requirements
- Module requirements for packages used by the code

Further benefits from Gitlab that are utilised in this case study

- Datasets for the example problems
- User documentation in a designated folder
- Issues board

8 USEFUL WEB LINKS

Last accessed 25/01/2022.

https://threedots.tech/post/automatic-semantic-versioning-in-gitlab-ci/
https://connect.appypie.com/apps/Gitlab/integrations/sharepoint
http://artokai.net/2016/SPFXContinuousIntegration/
https://best-practice-and-impact.github.io/qa-of-code-guidance/intro.html
https://realpython.com/what-is-pip/#installing-packages-from-your-github-repositories