TECHNICAL REPORT
IGE–417

DEVELOPMENT PROCEDURES FOR VERSION 5.1 OF REACTOR PHYSICS CODES

A. Hébert

Institut de génie nucléaire
Département de génie mécanique
École Polytechnique de Montréal
June 5, 2026

SUMMARY

Version 5.1 is a distribution of reactor physics codes developed at the Groupe d'Analyse Nucléaire (GAN) of École Polytechnique de Montréal. This distribution is developed using modern software engineering techniques that are likely to improve its quality and help the developers in their daily work. Version 5.1 is hosted in the NEA Data Bank GitLab platform of the Organisation for Economic Co-operation and Development (OECD). Four aspects of development procedures are described in this report:

- version control of the project components
- issue tracking and spiral development management
- continuous integration (CI) of new developments
- configuration management of the codes DRAGON, TRIVAC and DONJON.

We will discuss the implementation of the development procedures and their use by Version5.1's developers. These procedures are often assimilated to a *quality assurance* (QA) plan (*plan d'assurance qualité particulier* in french), although they are conceived to facilitate the evolution of Version5.1 rather than to slow down its development.

1 Version5 basics

One of the main goal of Version5.1 is to encapsulate all reactor physics codes developed at GAN in a unique and consistent software development project, and to maintain its consistency and quality during its development. We want to avoid any duplication of code among software components and maintain consistency in LCM object specifications and module specifications. In this system, DRAGON is consider as a foundation code providing functionalities to the other codes in the system. Any LCM object or module available in DRAGON and required by another code is simply accessed from DRAGON library and is not duplicated as previously done in Version3.

Version5.1 is divided into software components, some of them producing only libraries and some producing both libraries and executables. Version5.1 development is performed under the GNU Lesser General Public License (LGPL).[1] The components are

Utilib Set of numerical analysis Fortran subroutines compiled as a library.

Ganlib Set of C and Fortran subroutines implementing the computer science layers in Version5.1, including memory management, LCM access routines, CLE-2000 macro-language[2] and utility modules. Both library and executable are produced.

Trivac Full-core finite element code (static and space-time kinetics).[3] Both library and executable are produced.

Dragon Lattice code.[4, 5] DRAGON produces a *multi-parameter reactor database* that can be used as input in full-core calculations. Both library and executable are produced.

Donjon Full-core simulation of reactor operation.[6] Both library and executable are produced.

PyGan Python3 interface to Ganlib, Trivac, Dragon and Donjon.[7] PyGan can call any CLE-2000 procedure and access any file or memory object, in read, write or read-write mode.

2 Version control of the project components

Version control is the art of managing changes to information. Although mainly used for software development, its use can be extended to manage the production of any textual document involving many contributors. A version control tool make possible the collaborative work of many developers on the same project by implementing a systematic way of making modifications. Each modification bring to the project is recorded and can be removed if required. Tools are available to manage the tedious situation where many developers are working on the same project component.

Many tools are available to perform version control. Previous versions 4 and 5 of the code were hosted on the Subversion (svn) platform at Polytechnique Montréal.[8] Version 5.1 is now hosted on the GitLab platform of the Organisation for Economic Co-operation and Development (OECD).[9] Open-access cloning of the repository is provided using the "*Clone with HTTPS*" Code option. GIT is an open-source version control system that is free, powerful, well-accepted by computer scientists and widely available.[10] GIT is a widely-used system used to keep track of the historical evolution of software, procedures, scripts non-regression tests and related documentation. We propose to use GIT for the totality of Version5.1 components. GIT can be used from command line in a UNIX shell or from a graphical user interface such as the NEA GitLab.[11]

In version control terminology, a *commit and push* (aka *check-in*) is an operation where a developer input some information in the repository and recover a corresponding *SHA-1 hash value*. Note that a commit and push operation can never be undo, the information written in the repository is written forever. However, it is always possible to perform a subsequent commit and push operation to annihilate the effect of a previous commit (known as a *revert* operation), without erasing any information. In a version control system, it is always possible to recover the state of a system corresponding to *any* SHA-1 hash value.

The basic principle behind version control is to keep the project information in a *version control database* or *repository* and to record every operation performed on this repository. The information in the repository is organized with a directory structure, as shown in Fig. 1. Each directory contains a hidden sub-directory named `.git` with version control information.

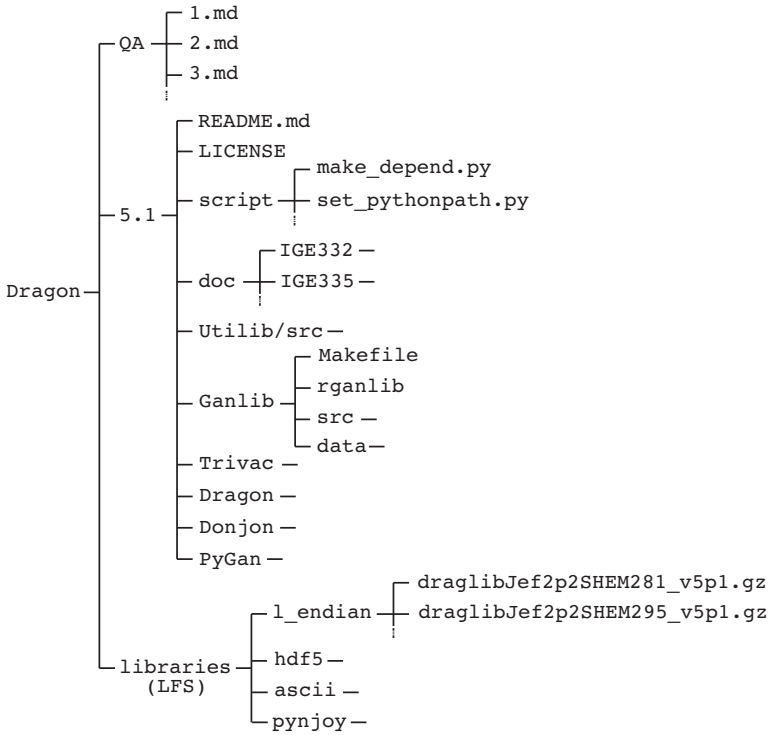


Figure 1: Directory layout in the GIT repository

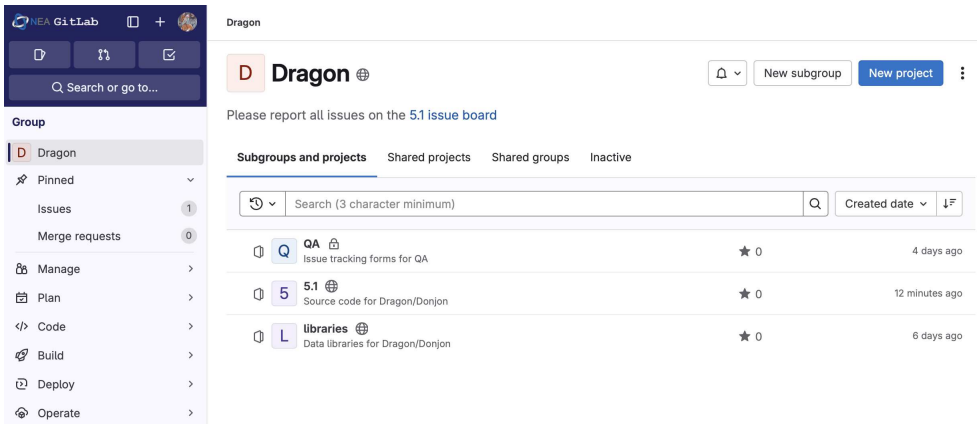


Figure 2: The Dragon5.1 NEA Gitlab

The Dragon system on GitLab is represented in Fig. 2. It includes three repositories:

1. The `QA` repository contains *issue tracking information*. The issue form `1.md` describes correction made in response of the specific user or developer concern number 1. An issue may need one or many *commits* and *pushes* to the `5.1` repository. Information in the `QA` repository is recovered from data available in the 5.1 *Issues* database of GitLab, including the list of modified components.

2. The `5.1` repository contains the overall Dragon/Trivac/Donjon project, including the following information:

   - Source code (Fortran, C, Python, CLE-2000, etc.)
   - Non-regression tests and data examples
   - Makefiles and configuration scripts
   - Execution scripts
   - Latex documentation.

3. The `libraries` repository contains cross-section library information required to run non-regression tests and data examples. Information in this repository is kept in *Large File Storage* (LFS) format, adapted to the management of big binary files. This repository has four directories:

   `l_endian:` Little-endian binary files

   `hdf5:` HDF5-formatted files

   `ascii:` ascii-formatted files

   `pynjoy:` PyNjoy data files used to construct the Draglibs with branch `1-include-module-...` of NJOY-2016.[13]

2.1 Initial setup of the project

The initial setup of the projet was made in September 2025, with assistance from the NEA Data Bank. One objective was to keep the *quality assurance* (QA) procedures of previous Versions 4 and 5 setups with Subversion.[14] Another objective was to include new Continuous Integration (CI) procedures, not available with the Subversion setup.

- QA procedures are adressing a long-term requirement. Development of Dragon is realized over many decades and involves many developers. Each modification in the code should be motivated by a user or developer need, called an *issue*. QA data is keeping track of the relations between issues and modifications. Any development has the potential to break an existing capability of the code, something called a *regression*. If a regression is observed years after a modification was done, QA data is essential to identify the corresponding trigging issue.

In the previous Subversion system, QA data was collected and managed by a system of pre-commit and post-commit Python scripts. QA information was committed in the *same* repository containing the code source. With GitLab, a different system was implemented by the NEA staff, with the same objective. A companion repository named `QA` was created to hold QA data. This information is recovered from the 5.1 *Issues* database of GitLab using YAML scripts. Any issue should therefore be created ou reopened by clicking the *Issues* menu key of GitLab in the repository named `5.1`. The QA information is created at the moment where the issue is closed. If an issue is reopened and reclosed, a new entry is created, collection old and new data related to this issue, including information appended by the developer in charge of the issue. The developer must provide enough information to ensure that future developers (this may be decades later) will understand the motivation for the 5.1 repository modification.

The information recovered by the YAML script is transcripted into *card-indices* in the QA repository. Each card-index represent a single development issue.

- CI procedures are adressing a short-term requirement. It is important for any development to avoid breaking an existing capability of the code. Many programming strategies should be used:

  1. The NEA GitLab includes *CI pipelines* that are systematically executed each time a *push* or *merge* operation is required. These pipelines consist of installing the code and executing non-regression tests on the NEA servers. Currently, executable objects and Docker images are constructed for Ubuntu 22 and Ubuntu 24 (Linux systems) and a subset of non-regression tests are executed. The Latex documentation is also constructed. In case of failure, the *push* or *merge* operation is cancelled.
  2. Success of the CI pipelines in GitLab is not enough to validate a code modification. The developer should also execute the complete set of non-regression tests on its personnal system using the `make tests` command. Personnal systems used by developers are MacOS or Linux laptops (including WSL). These systems are very permissive and don't always detect programmation issues. For important modifications, we also recommand more extensive tests on exotic systems such as IBM AIX or SUN Solaris. Different compilers such as LLVM Flang/Clang can also be used.

2.2 Daily use of the code

The main use of the code consists to run datasets using its last development version. Here are the main steps:

1. Clone the code. If you have an account on the NEA databank, type

   ```
   git clone git@git.oecd-nea.org:dragon/5.1.git
   ```

   If not, type

   ```
   git clone https://git.oecd-nea.org/dragon/5.1.git
   ```

   If you have already cloned the code and want to upgrade it to the latest version, type

   ```
   cd 5.1
   git pull
   ```

2. Clone the cross-section libraries. The complete set of libraries is requiring a bulk of disk space; you may want to download only a few of them. If you have an account on the NEA databank, type

   ```
   git lfs clone git@git.oecd-nea.org:dragon/libraries.git
   ```

   If not, type

   ```
   git lfs clone https://git.oecd-nea.org/dragon/libraries.git
   ```

3. Define environment variables in your `.profile` or `.zprofile` file. Add lines similar to those in the following box. Double check the versions corresponding to the dependencies available in your OS.

```
# Support for HDF5
export HDF5_INC="/opt/homebrew/Cellar/hdf5/2.1.1/include" # HDF5 include directory
export HDF5_API="/opt/homebrew/Cellar/hdf5/2.1.1/lib"     # HDF5 C API

# Support for Python3
export FORTRANPATH=/opt/homebrew/Cellar/gcc/15.2.0_1/lib/gcc/15/ # libgfortran.a
export PYTHONPATH=/opt/homebrew/lib/python3.14/site-packages/

# Support for OpenMP
export LDFLAGS="-L/opt/homebrew/opt/libomp/lib"
export CPPFLAGS="-I/opt/homebrew/opt/libomp/include"

# Support for flang-new compiler
export LIBRARY_PATH=/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/lib
export LIBRARY_PATH="$LIBRARY_PATH: \
       /Library/Developer/CommandLineTools/usr/lib/clang/21/lib/darwin"
export LLVMTOOLS=/opt/homebrew/Cellar/flang/22.1.3/lib # libfortran.a

# Support for GLOW
export SALOME_DIST=$HOME/Salome9_GLOW/Salome9
export PYTHONPATH="$PYTHONPATH:$HOME/Salome9_GLOW/glow"
echo "Salome Distribution: ${SALOME_DIST}"
```

Environment variables are required to support external capabilities. Dragon5.1 can run without these capabilities, but they are generally required for production use. Among them:

Python3 The Python3 interpreter and API are required to support PyGan.

OpenMP OpenMP provides *high performance computing* (HPC) capabilities.

hdf5 HDF5 is a modern direct-access file system embedding a hierarchical structure.

GLOW The acronym GLOW stands for *Geometry Layout Oriented Workflow* and allows the definition of complex 2D geometries using a *Constructive Solid Geometry* (CSG) approach.[15]. GLOW is based on the Salome9 platform.

flang Flang is the Fortran compiler of the LLVM suite. By default, gfortran is used.

Execution of the non-regression tests uses the `make tests` command. For example, to execute the Dragon tests, including HDF5, OpenMP, proprierary apolib and GLOW Python scripts, type

```
cd Dragon
make tests hdf5=1 openmp=1 apolib=1 salome=1
```

Other options allow the replacement of gfortran/gcc with flang/clang (`llvm=1`), ifort/icc (`intel=1`) or nvfortran/nvc (`nvidia=1`).

To execute a specific dataset, type

```
cd Dragon
./rdragon salmacro.x2m
```

To clean a directory, without erasing the `bin` and `lib` directories, type

```
cd Dragon
make clean
```

The `make` operation is recursive, meaning that its execution in the Donjon directory also install Ganlib, Trivac and Dragon. The `make clean` operation is also recursive in all cases, except for PyGan.

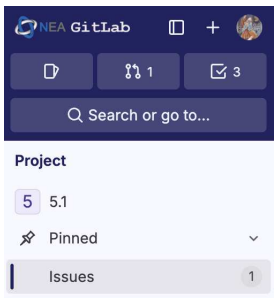
2.3 Daily operations on the repository

During the life of the project, selected developers are allowed to perform read and/or write operations on the repository. We will cover the more frequent cases. More information can be found in Ref. 10. Note that the developer must log into its GitLab account to perform these operations. A developer should first add the following lines to its `.profile` or `.zprofile` file:

```
#GitLab setup
if [ -z "$SSH_AUTH_SOCK" ] ; then
  eval `ssh-agent -s`
  ssh-add
fi
```

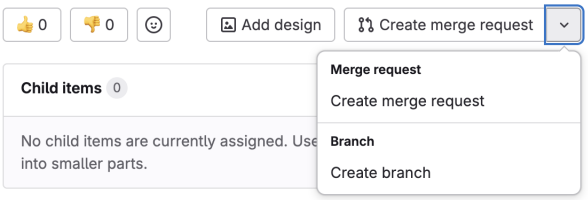
A developer receive a request to correct the code or implement a new capability. The workflow proceeds in three steps.

1. An issue should be opened in GitLab

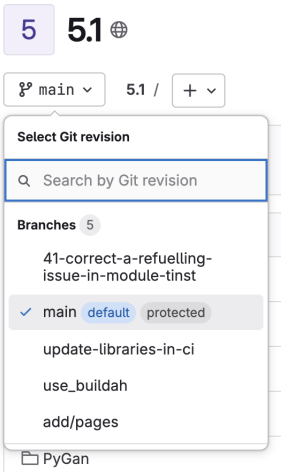


Press the *Issues* key and press the *New issue* key (for a new issue) or *Closed* (for reopening a previous issue). For a new issue, provide a title and a description. The description should include the origin of the request, the date and a short description of the issue. If you are reopening a previous issue, provide the issue information in the *Activity* box and press the *Reopen issue* key.

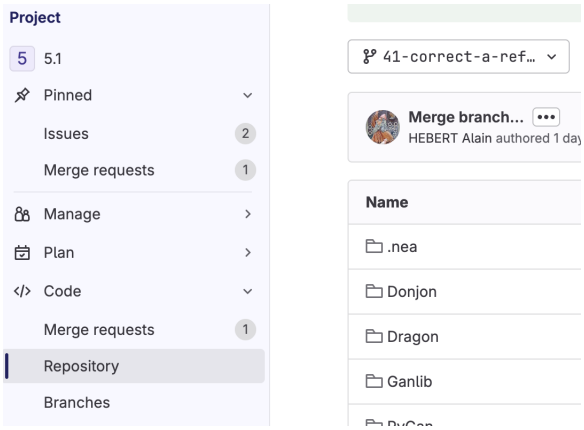
Next, press the *create branch* key:



After this operation, a branch is created in GitLab to hold the issue modifications before its merge in the main



You may notice the index of the issue (here 41) and its name (`41-correct-a-refuelling...`. If you press this name, you access the branch itself:



There is a key in page *Branches* to cut the name of the issue, so that you could paste it later:

```
41-correct-a-refuelling-issue-in-module-tinst
```

If you open an issue by mistake, you can delete it by pressing the *Edit* key on the right upper corner. Next, press the three vertical dots and select *Delete issue*.

2. After the branch corresponding to the issue is created on GitLab, the developer goes back to its personnal system and clone the branch locally:

```
bit clone -b 41-correct-a-refuelling-issue-in-module-tinst \
    git@git.oecd-nea.org:dragon/5.1.git
```

The developer performs the correction, runs the non-regression tests and update the documentation, if required. At any time, the developer may check the status of its branch using

```
cd 5.1
git status .
```

The next step is to *stage* any modification using command `git add` and perform a local commit of them:

```
cd 5.1
git add Dragon/src/EVODRV.f Donjon/src/TINST.f Donjon/src/TINMIC.f
git commit -m '#41: Correct a refuelling issue in module TINST:'
```

Note that the issue index is embedded in the commit message. The next step is to push the commit on GitLab:

```
cd 5.1
ssh-add ~/.ssh/id_ed25519
git push -u origin HEAD
```

In the context of the `/.ssh directory`, `id_ed25519` (often abbreviated as `id_ed` in listings or shorthand) is the default file name for a private SSH key that uses the Ed25519 algorithm. You can generate keys with the 'ssh-keygen' command:

```
ssh-keygen -t ed25519
```

The command output:

```
Generating public/private ed25519 key pair.
Enter file in which to save the key ($HOME/. ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in $HOME/.
```

A few additional GIT commands are useful:

- To suppress modifications in a file named `<<filename>>` and revert to its official version, type

```
git checkout -- <<filename>>
```

- To list committed files before push:

```
git show --name-status HEAD
```

- To view modifications in a file named `<<filename>>`, type

```
git diff <<filename>>
```

- If the modifications in `<<filename>>` have been staged, type

```
git diff --staged <<filename>>
```

- To enable LFS coding of library files, use the "`git lfs track`" command in the `library` directory and do a "`git add`" of file `.gitattributes`:

```
cd libraries
git status .
git lfs track <path/filename>
git add .gitattributes
```

3. The information is sent to the NEA GitLab after the `git push` is completed and the CI pipelines are automatically trigerred. You connect back to GitLab to follow their execution. If the pipelines failed, you may try to correct the commit or to just restart the pipelines using GitLab.

   After pipelines success, you create a merge request and perform the merge on GitLab:

Finally, perform the merge. The CI pipelines are automatically trigerred one more time. After pipelines success, you may want to close the issue, so that the QA *card-index* is produced. Print this *card-index* and include it in the QA binder.

References

[1] GNU Lesser General Public License. See https://www.gnu.org/licenses/lgpl-3.0.en.html.

[2] R. Roy, *The CLE-2000 Tool-Box*, Report IGE–163, Institut de Génie Nucléaire, École Polytechnique de Montréal, Montréal, Québec (1999).

[3] A. Hébert, "TRIVAC, A Modular Diffusion Code for Fuel Management and Design Applications", *Nucl. J. of Canada*, Vol. 1, No. 4, 325-331 (1987).

[4] G. Marleau, A. Hébert and R. Roy, "New Computational Methods Used in the Lattice Code Dragon," *Int. Top. Mtg. on Advances in Reactor Physics*, Charleston, USA, March 8-11, 1992.

[5] G. Marleau, A. Hébert and R. Roy, "A User Guide for DRAGON Version5", Report IGE-335, École Polytechnique de Montréal, Institut de Génie Nucléaire (2020).

[6] A. Hébert, J. Sekki and R. Chambon, "A User's Guide for DONJON Version5," Technical Report IGE-344, École Polytechnique de Montréal, Institut de Génie Nucléaire (2020).

[7] A. Hébert and R. Roy, "The Ganlib5 kernel guide," Technical Report IGE-332, École Polytechnique de Montréal, Institut de Génie Nucléaire (2020).

[8] B. Collins-Sussman, B. W. Fitzpatrick and C. Michael Pilato, "Version Control with Subversion," O'Reilly Media Inc., USA, June 2004. See http://subversion.tigris.org.

[9] Dragon 5.1 GitLab website, September 2025. See https://git.oecd-nea.org/dragon.

[10] S. Chacon and B. Straub, "Pro Git," Apress Publishing, USA, May 2026. See https://git-scm.com/book/en/v2.

[11] NEA Data Bank GitLab platform. See https://databank.io.oecd-nea.org/.

[12] Linux Subsystem for Linux. See https://learn.microsoft.com/en-us/windows/wsl/.

[13] NJOY-2016, October 2025. See https://git.oecd-nea.org/njoy.

[14] A. Hébert, "Development procedures for Version5 of reactor physics codes", Report IGE-374, École Polytechnique de Montréal, Institut de Génie Nucléaire (2014).

[15] Glow website, January 2026. See https://github.com/newcleo-dev-team/glow.