

Contributing to Dynare

Sébastien Villemot (CEPREMAP)

25 November 2022

1st Dynare Workshop for Advanced Users (JRC, Ispra)

Examples of contributions

- Answering questions on the forum
- Reporting a bug
- Fixing a bug
- Adding a new feature
- Improving the performance of an algorithm
- Adding a test case in the test suite
- Improving the documentation

Our community

- People
 - Core developers (“Dynare Team”)
 - Advisory committee
 - Occasional contributors
 - Users
- Communication channels
 - Forum
 - GitLab instance
 - Mailing lists (info@dynare.org, dev@dynare.org)
 - 3 annual events: summer school, JRC workshop, conference
- [Code of Conduct](#)

Working with GitLab


Our GitLab instance

- GitLab
 - Source code hosting facility (also known as a *forge*)
 - Built around git repositories
 - Additional features: issues, merge requests, continuous integration, milestones, wiki
 - Our own GitLab instance: <https://git.dynare.org>
- Official Dynare repository: <https://git.dynare.org/Dynare/dynare>
- Generic repository URL: <https://git.dynare.org/namespace/project> where *namespace* can be a group name (e.g. Dynare) or a user name (e.g. sebastien)
- List of all repositories of the Dynare group: <https://git.dynare.org/Dynare>
- Most repositories are publicly visible, but you need to [create your account](#) to contribute

Exploring a repository on GitLab

- Source tree
- Commit history
- Branches (master = development branch; 5.x = current stable)
- Tags (one tag per release; also used for alpha and beta versions)
- Issues
- Merge requests (MR)
- Continuous integration (CI)

Reporting a bug or suggesting a new feature

- Bug reports and feature suggestions are *issues* in GitLab
- Issues are attached to a repository \Rightarrow decide whether to create the new issue against the main Dynare repository or against a sub-component (preprocessor, dseries...); if in doubt, choose the main Dynare repository
- Verify that the issue has not already been created
 - search the [issues list](#)
 - if it is a bug, check the [known bugs wiki page](#)
- Click on the *New issue* button (on the [issues list](#), or under the  icon in the top bar)
 - add a title and a description
 - add label(s) if relevant

Setting up local and personal repositories

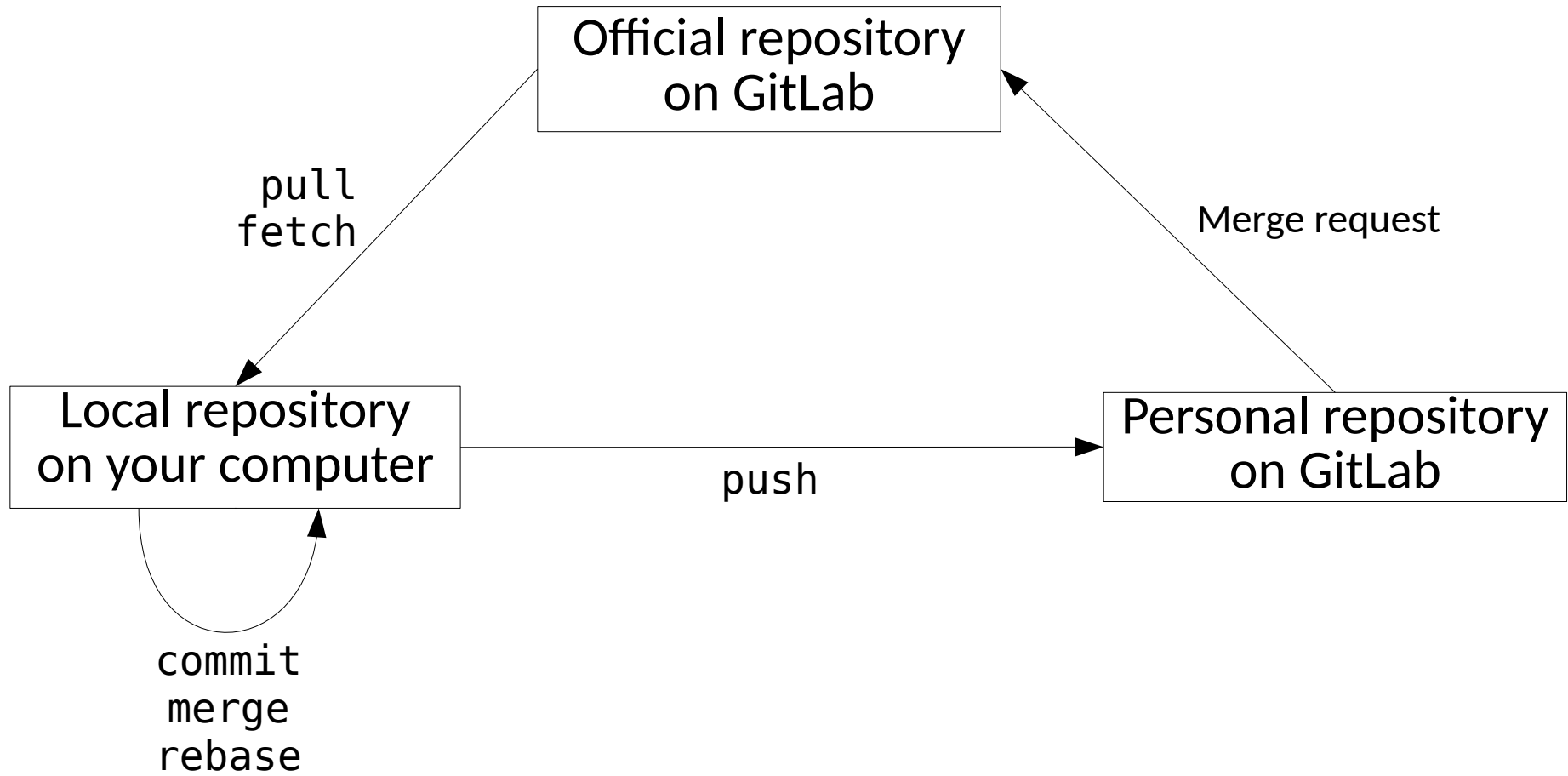
- Suppose you want to contribute to the official Dynare repository
- Clone the repository locally on your computer

```
git clone --recurse-submodules https://git.dynare.org/Dynare/dynare.git
```
- Create your personal Dynare repository on GitLab
 - go to the GitLab page of the [official Dynare repository](#)
 - click on the *Fork* button on the upper right corner
 - your personal GitLab repository is now at
`https://git.dynare.org/username/dynare`
- Link your local repository to your personal GitLab repository

```
git remote add personal https://git.dynare.org/username/dynare
```

NB: `personal` in the above command is just a nickname, you can use whatever you like


Git workflow



Git workflow in commands

- Downloading commits from the official repository on GitLab
 - `git pull --recurse-submodules origin`
 - Or, if you have unmerged local commits
 - `git fetch origin`
 - `git rebase origin`
 - NB: origin is optional (if local repository was cloned from official one)
- Adding commit(s) locally
 - edit files
 - `git add <files>`
 - `git commit`
- Pushing to your personal repository on GitLab
 - `git push personal`
 - NB: you may need to add the `--force` option if you rebased or amended commit(s)

Creating a merge request (MR)

- Suppose you pushed commits to your personal GitLab repository, that you now want to merge into the official repository
- Open the page of your personal GitLab repository:
`https://git.dynare.org/username/dynare`
- Click on the  icon in the top bar, then *New merge request*
- Select `master` as the source branch, and leave the rest as it is
- Click on *Compare branches and continue*
- Choose a title and optionally a description and relevant label(s)
- Click on *Create merge request*
- Your merge request is now listed among the [merge requests](#) against the official repository; the Dynare Team will be notified and will review it

Continuous integration (CI)

- After each push, GitLab runs a *pipeline* (a sequence of *jobs*) to verify that nothing is broken by the new commits, and to build *artifacts*
- **Dynare pipeline:**
 - builds binaries and documentation
 - runs the test suite (on MATLAB R2022b, optionally on R2014a and Octave)
 - creates the Windows and macOS installers, and the source tarball
 - uploads the installers, the source tarball and the documentation to the Dynare website (NB: this step is skipped on personal forks)
- Artifacts (e.g. installers) and logs can be manually downloaded from the job page (before they expire)
- No pipeline will be created if the commit message contains `[skip ci]`

The source code of Dynare

Structure of the Dynare source tree

- `preprocessor`: source code of the preprocessor (C++)
- `matlab`: core computational routines (MATLAB/Octave)
- `mex`: source code of MEX files (C++, Fortran)
- `doc`, `examples`: reference manual and other documentation
- `tests`: automated test suite
- `windows`, `macOS`: production of platform-specific installers
- `contrib`: third-party code
- `scripts`: misc utilities for developers
- `configure.ac`, `Makefile.am`, `m4`: build system

The Dynare preprocessor

- Standalone executable, run at an early stage by the dynare command
- Role:
 - parses the `.mod` file (possibly with a macro-processing step)
 - performs input validation and sanity checks
 - performs several model transformations (auxiliary variables, Ramsey optimality conditions...)
 - computes the static version of the model (for the steady state)
 - computes block decomposition
 - computes symbolic model derivatives
 - writes model information for consumption by MATLAB or Octave
- Written in C++ (`.cc` and `.hh` extensions)

Core computational routines

- Routines for:
 - perturbation solution
 - simulation: perturbation, perfect foresight, purely backward
 - estimation: Bayesian, classical full information, classical partial information (methods of moments)
 - identification and sensitivity analysis
 - optimal policy
- Written in MATLAB/Octave (.m extension)
- Organized in subdirectories of the `matlab` directory
 - some (recent) subdirectories are *packages* in the MATLAB terminology (“+” prefix)
 - others (older) are vanilla folders that need to be added to the MATLAB path
 - ideally we will migrate most subdirectories to packages

MEX files

- Functions written in lower-level language that can be called directly from MATLAB or Octave
- Used for accelerating performance-critical sections of algorithms, *e.g.*:
 - specialized Kronecker products
 - solutions to polynomial matrix equations (cycle reduction, logarithmic reduction, discrete Lyapunov)
 - k -order perturbation simulation
 - law of motion of particles within particle filtering estimation
 - construction of the stacked Jacobian of the perfect foresight problem
- Written in C++ or Fortran
NB: we're currently trying to migrate most MEX files to (modern) Fortran, since researchers in numerical methods usually find it easier than C++

Documentation

- Reference manual
 - Under `doc/manual/` subdirectory
 - Written in [reStructuredText](#) (RST), a lightweight markup language
- Other documents in LaTeX under `doc/` or `preprocessor/doc/`
- Example `.mod` files under `examples/`
- Wiki pages targeted at both users and developers:
<https://git.dynare.org/Dynare/dynare/-/wikis/home>
- Also, resources listed on the Dynare website:
<https://www.dynare.org/resources/>

Test suite

- Critical element of our quality assurance
- Includes two types of tests:
 - *integration* tests: complete `.mod` files, possibly including MATLAB commands for checking the value of the results
 - *unit* tests: for testing specific subroutines, such as a given function or algorithm
- Adding an integration test is easy:
 - add a `.mod` file under the `tests/` directory
 - append its filename to the `MODFILES` variable in `tests/Makefile.am`
- The procedure for adding a unit test is detailed in [CONTRIBUTING.md](#)

Submodules

- A *git submodule* is a git repository used as a subdirectory of another git repository
- Several submodules in Dynare repository:
 - preprocessor
 - dseries, reporting (under matlab/modules/)
 - particle filtering routines (under matlab/particles/)
 - unit testing framework (under matlab/utilities/tests/)
 - third-party code under contrib/
- Dealing with submodules is a bit tricky, but you can mostly ignore them (unless you want to modify one of them)
- However do not forget to pass `--recurse-submodules` option to `git pull` to keep local submodules up-to-date

Compilation

- The preprocessor and MEX files are written in C++ and Fortran and thus need to be compiled (their *binaries* have to be built from *source*)
- If not under Linux, a development environment needs to be installed:
 - [MSYS2](#) for Windows
 - [Homebrew](#) for macOS
- Detailed instructions for building from source are in [README.md](#)
- However, if you do not plan to contribute to the preprocessor or the MEX files, a simpler alternative is to use the binaries included in the unstable snapshot of Dynare:
 - install the latest snapshot on your computer
 - copy the preprocessor and MEX binaries to your local git repository
 - regularly repeat the above two steps with newer snapshots

Coding guidelines

- Comment your code
- Document your changes (reference manual, wiki pages)
- Add tests
- Legal information in file header
 - Copyright (ideally): Dynare Team
 - License: GNU General Public License, version 3 or later (GPL-3+)
- Style requirements
 - Follow indentation rules
 - Use spaces instead of tabulations
 - Use the LF end of line (not CR+LF)
 - A good editor can automate these
- See the [coding guidelines](#) and [coding resources](#) wiki pages