Advices for students who intend to do numerical modelling at ISTO

Emmanuel Le Trong

September 26, 2025

This document is tracked here: https://github.com/calculisto/isto_numerical_modeling_guide.

Contents

1 Regarding the LaTeX typesetting system

You should write your papers using the LaTeXsystem.

1.1 Some useful LaTeX packages

Here is a list of packages that will simplify the production of quality material.

- `physics` (https://ctan.org/pkg/physics). It brings a lot of useful macros for vector operations like gradient and divergence, mathematical

functions, partial or total derivatives, matrices, etc. My favorite: instead of

```
\frac{\partial^2 f}{\partial x^2}
```

I can write

```
\pdv[2]{f}{x}
```

which is far easier to read. It automatically loads the `amsmath` package. (Which you should always use anyway.)

TODO: `physics` considered harmful.

- `bm` (https://ctan.org/pkg/bm). To get symbols in boldface, in particular Greek letters. The `physics` package provides the `\vb` macro to do that but depending on the font you use it might not be "bold enough" or even not bold at all. Note that the `bm` package produces slanted bold Latin letters while `\mathbf` and `\vb` from the `physics` package produce upright characters. Thus

```
\begin{equation}
    u \mathbf{u} \vb{u} \bm {u}.
\end{equation}
```

gives

$$u\mathbf{uu}u. \tag{1}$$

- `siunitx` (https://ctan.org/pkg/siunitx). To correctly typeset quantities with physical units. E.g.

```
\qty{18.015e-3}{\kilogram\per\mole}
```

gives $18.015 \times 10^{-3}\,\mathrm{kg\,mol^{-1}}$.

Caveat: Both `physics` and `siunitx` define the macro `\qty` to do different things. I find the later more useful. In order to use it without fuss, that I put

```
% Use siunitx \qty macro and silence the warning
\AtBeginDocument{\RenewCommandCopy\qty\SI}
\ExplSyntaxOn
\msg_redirect_name:nnn{siunitx}{physics-pkg}{none}
\ExplSyntaxOff
```

in the preamble of my document. The functionality of the `physics` version of the macro is still available under the name `\quantity`, if you need it.

- `minted` (https://ctan.org/pkg/minted). To include source code in your document. E.g. this LaTeX snippet

```
\begin{minted}[linenos,frame=lines]{cpp}
#include <iostream>
int main ()
{
    // The simplest program
    std::cout << "Hello world\n";
```

```
        }
      \end{minted}
```

renders like so

```cpp
1    #include <iostream>
2    int main ()
3    {
4        // The simplest program
5        std::cout << "Hello world\n";
6    }
```

- mhchem (https://ctan.org/pkg/mhchem). To typeset chemical formulas and reactions. E.g.

  ```
  \ce{H2O -> OH- + H+}
  ```

  gives $H_2O \longrightarrow OH^- + H^+$

- tikz (https://ctan.org/pkg/tikz). To produce drawings. For example, this ball ● is obtained with this command

  ```
  \tikz \shade[ball color=red] (1ex,1ex) circle (1ex);
  ```

  It can do a *lot* (the manual has 1321 pages) but it requires learning a new language.

- ctable (https://ctan.org/pkg/ctables). To produce what some call "professional-looking" tables. It uses the booktabs (https://ctan.org/pkg/booktabs) package whose documentation is worth reading. For example, this

  ```
  \ctable[
    caption=The price of things,pos=h,label=table1
  ]{lS@{\ \ }l}{}{
      \FL
      Product & {Price (€)} &
      \ML
      Item 1 & 22.00 &
      \NN
      Item 2 & 5.25 &
      \LL
  }
  ```

  produces the table 1.

Table 1: The price of things

Product	Price (€)
Item 1	22.00
Item 2	5.25

1.2 Some tips when writing LaTeX

(Those are *tips*, not *rules*.)

- We are in Europe, use A4 paper: add the option `a4paper` to the document class, e.g.

  `\documentclass[a4paper]{article}`

- Equations are part of the text and as such must obey the rules of grammar, especially punctuation. In practice this means that when needed, an equation shall be followed by a comma or a dot (when it is at the end of a sentence).

- Mathematical constants (e.g. the base of the natural logarithm e or the imaginary unit i) shall be typeset in roman (upright) font. So shall mathematical functions like cos, sin, ln, etc. (These are available as macros in amsmath: `\cos`, `\sin`, `\ln`, etc.) Subscripts and superscripts that refer to words (as opposed to variables) shall also be typeset in roman. For example the velocity $\bm{u}$ in the solid phase should be noted $\bm{u}_\text{s}$ instead of $\bm{u}_s$. The way to typeset in roman font inside math mode is via the `\text` macro. The velocity above was produced by this snippet `\bm{u}_\text{s}`.

- Use LaTeX macros to make your equations more readable. If you are going to do a lot of volume averaging, you should define some operators in the preamble of your document, such as

  `\newcommand{\Average}[1]{\left<#1\right>}`
  `\newcommand{\IntrinsicAverage}[2]{\left<#1\right>^{#2}}`

  which can be used in the body of the document as such

  `\begin{equation}`
  `\Average{\Psi_\beta} =`
  `\varepsilon_\beta\IntrinsicAverage{\Psi_\beta,\beta}.`
  `\end{equation}`

  and yields

  $$\langle \Psi_\beta \rangle = \varepsilon_\beta \langle \Psi_\beta \rangle^\beta. \tag{2}$$

  (Notice the dot at the end of the equation.)

- Use LaTeX macros even more. I personally grant every quantity I use a meaningful name by defining a macro. To render the Navier-Stokes equation, I would define these in the preamble

  `\newcommand{\Time}{t}`
  `\newcommand{\Density}{\rho}`
  `\newcommand{\Velocity}{\bm{u}}`
  `\newcommand{\Pressure}{p}`
  `\newcommand{\DeviatoricStress}{\bm{\tau}}`
  `\newcommand{\Gravity}{\bm{g}}`

  and then write somewhere in the body of the document

  `\begin{equation}`
  `    \pdv{\Time}\left(\Density\Velocity\right)`

```
      + \div{\left(\Density\Velocity\Velocity\right)}
      =
      - \grad{\Pressure}
      + \div{\DeviatoricStress}
      + \Density\Gravity.
   \end{equation}
```

which gives

$$\frac{\partial}{\partial t}\left(\rho \boldsymbol{u}\right) + \boldsymbol{\nabla} \cdot \left(\rho \boldsymbol{u}\boldsymbol{u}\right) = -\boldsymbol{\nabla}p + \boldsymbol{\nabla} \cdot \boldsymbol{\tau} + \rho\boldsymbol{g}. \tag{3}$$

It might appear tedious at first but will prove life-saving later if, e.g., you mix physics, chemistry and thermodynamics in your document, when inevitable naming clashes appear.

2 Working with `OpenFOAM` and `Porous4FOAM`

`OpenFOAM` is the computational fluid dynamics software we use at ISTO.

`Porous4FOAM` is a package for `OpenFOAM` dealing with reactive transport in porous media. It is developed at ISTO.

If you intend to use `OpenFOAM`, you need to know `OpenFOAM`. Read at least the user guide https://doc.cfd.direct/openfoam/user-guide. We also have books, get in touch.

If you intend to contribute to `Porous4FOAM` you need to know C++. We have books, get in touch with me if you need one. You also need to know `git`: read the book "ProGit" available here https://git-scm.com/book/en/v2.

2.1 Installing `OpenFOAM`

We are going to download and compile `OpenFOAM`. We will clone the development source tree of `OpenFOAM.org` to get the full commit history of the project. It is a valuable source of information. From this source tree, we will select the version we want (here the 13) and compile it.

1. Change directory to where you want to install OpenFOAM (here we create a new sub-directory `OpenFOAM` in the home directory).

   ```
   $ mkdir OpenFOAM
   $ cd OpenFOAM
   ```

2. Download the development source tree.

   ```
   $ git clone https://github.com/OpenFOAM/OpenFOAM-dev.git
   $ git clone https://github.com/OpenFOAM/ThirdParty-dev.git
   ```

3. Rename the directories, in order to keep several different version each in its own directory and checkout the source for the chosen version and compile.

   ```
   $ mv OpenFOAM-dev OpenFOAM-13
   $ mv ThirdParty-dev ThirdParty-13
   $ cd OpenFOAM-13
   ```

```
$ git checkout version-13
```

4. Open the file `etc/bashrc` with your favorite text editor and change the line containing

   ```
   export WM_PROJECT_VERSION=dev
   ```

   to

   ```
   export WM_PROJECT_VERSION=13
   ```

   (It should be around line 36.)

5. Initialize `OpenFOAM` and launch the compilation.

   ```
   $ source etc/bashrc
   $ ./Allwmake -j
   ```

6. Wait, the compilation might take up to several hours, depending on your hardware.

2.2 Using `OpenFOAM`

To be able to use any particular version (here again, the 13th) of `OpenFOAM` you have installed on your machine, you need to initialize it with

```
$ source $HOME/OpenFOAM/OpenFOAM-13/etc/bashrc
```

(You might get an error involving `dirname`, ignore it.) You can avoid typing this command each time you open a terminal by putting it into your `$HOME/.bashrc` file.

The next step is to create your working directory and switch to it

```
$ mkdir -p $FOAM_RUN
$ cd $FOAM_RUN
```

(The last command as a shorter alias: `run`.)

From there you should be able to run some tutorials as explained in the user guide https://doc.cfd.direct/openfoam/user-guide.

2.3 Exploring `OpenFOAM` source code

After initializing `OpenFOAM`, go to the source directory

```
$ source $HOME/OpenFOAM/OpenFOAM-13/etc/bashrc
$ cd $FOAM_SRC
```

(The last command as a shorter alias: `src`.)

From there you have access to the full commit log, down to the year 2014

```
$ git log
```

The commit messages can be an invaluable source of documentation, with informations nowhere else to be found. As an example, check this commit message

```
$ git log -1 968e60
```

or this one

```
$ git log -1 65ef2c
```

or this one about a recent functionality you might not be aware of

```
$ git log -1 476bb42b
```

If you are interested in a particular file, you can list only the commits that modified it. Let's go to the location of the file of interest

```
$ cd $FOAM_APP/modules/multiphaseEuler
```

List the commits that modified a specific file

```
$ git log momentumPredictor.C
```

or parts of that file, given by a line range

```
$ git log -L 48,58:momentumPredictor.C
```

or just a function in that file

```
$ git log -L :prePredictor:multiphaseEuler.C
```

2.4 Using `Porous4FOAM`

2.5 Contributing to `Porous4FOAM`

There are at the moment two versions of `Porous4FOAM`: the legacy one, written essentially by Cyprien Soulaine, and the new one which is an on-going rewrite from scratch.

The repositories for `Porous4FOAM` are hosted (for now) on `GitHub`. The legacy one is here https://github.com/csoulain/porousMedia4Foam and the new one is here https://github.com/calculisto/porousMedia4Foam. (Beware, they have the same name! If you want to use both, you need to clone them in different directories.)

If you want to contribute the `Porous4FOAM` here is a summary of the steps to follow

1. You should have an account on `GitHub`. From there, fork the project.

2. Clone the repository somewhere in your `$FOAM_RUN` directory (either the original repo or your fork).

3. Immediately make a branch! All your contribution should go in that branch.

4. Write some code **and some tests** (your code must be thoroughly tested), commit them to your branch. All your commits **must be signed**!

5. When the time comes to submit your work for integration in the main repo, make a pull request and send it to me.

Let's see that in more detail. (There are even more details in the `git` section of this document.)

2.5.1 Fork and clone

On `GitHub`, go to the home page of the repository you want to clone and fork it, there is a button for that. A fork is necessary to ba able to make a pull request later. On your computer change directory to your `OpenFOAM` working directory and clone your fork of the repository.

```
$ cd $FOAM_RUN
$ git clone https://github.com/<your_username>/porousMedia4Foam \
    porousMedia4FoamLegacy
```

(Here I cloned into a new directory called `porousMedia4FoamLegacy`.)

TODO: what about the `v11legacy` branch?

2.5.2 Make a branch, write code, commit

Make what is called a topic branch in which you will implement your new feature, and switch to it. All your contributions must be done in a new branch, this makes the life of everybody simpler.

```
$ git branch my_feature
$ git switch my_feature
```

Write code (and tests!) and commit them. Your commits must be signed. This means that you must have configured `git` with your name and email address and use the `-s` option of the `git commit` command (see the `git` section).

2.5.3 Issue a pull request

When you are ready to share your contribution with the world, you need to create a pull request and submit it to the maintainer of `Porous4FOAM`. Push your branch to your repository

```
$ git push -u my_feature
```

and make a pull request as explained in the `git` section.

TODO:

3 Introduction to git

You should read the book "ProGit" (available online https://git-scm.com/book/en/v2), at least the chapters "Getting started", "Git basics" and "Git basics".

I present here only the bare minimum you should know about `git`.

3.1 First time configuration

As you are required to sign off your commits when contributing to `Porous4FOAM`, git has to know your name and email address. This is done like so

```
$ git config --global user.name "Emmanuel Le Trong"
$ git config --global user.email "emmanuel.le-trong@cnrs.fr"
$ git config list
```

The last command should show

```
user.name=Emmanuel LE TRONG
user.email=emmanuel.le-trong@cnrs-orleans.fr
```

Do this only once.

3.2 Fork and clone

We have a sample project, you can fork it (this is done on `GitHub` web site, there is a button for that). After the fork, you should have yous own copy of the repository in your account. Clone it

```
$ git clone https://github.com/<yous_username>/hello
$ cd hello
$ git status
```

The last command should show

```
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

which tells me

- which branch I'm on
- this branch is tracking a remote branch and is up to date with it
- nothing has been changed yet

3.3 Create a branch

New development should go in a dedicated branch.

```
$ git git switch -c new_feature
$ git status
```

The last command should show

```
On branch new_feature
nothing to commit, working tree clean
```

which tells me that I indeed switched to the branch `new_feature`.

3.4 Contribute to the project

I add a new file `AUTHORS` and add a comment to the existing `hello.hpp` file

```
$ echo "Emmanuel LE TRONG" > AUTHORS
$ vim hello.hpp
$ git status
```

The last command should show

```
On branch new_feature
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
```

```
  (use "git restore <file>..." to discard changes in working directory)
        modified:   hello.hpp

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        AUTHORS

no changes added to commit (use "git add" and/or "git commit -a")
```

which tells me that there is a modified file and a new (untracked) file. We will now register our changes in two commits, one introducing the comment and another the new file. The first one

```
$ git add hello.hpp
$ git status
```

The last command should show

```
On branch new_feature
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   hello.hpp

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        AUTHORS
```

which tells me that the modified file is ready to be committed. Let's commit it

```
$ git commit -s -m "Add a comment to hello.cpp::say_hello"
```

This shows

```
[new_feature e042a59] Add a comment to hello.cpp::say_hello
 1 file changed, 2 insertions(+), 1 deletion(-)
```

which tells me that a new commit has been created. Here I have used the -m option of git commit to provide a short commit message directly on the command line. I also have used the option -s to sign off my commit. The command

```
$ git show e042a59
```

displays the full commit

```
commit e042a59268a750b9eff856f135f3854023dee9d2 (HEAD -> new_feature)
Author: Emmanuel LE TRONG <emmanuel.le-trong@cnrs-orleans.fr>
Date:   Wed Sep 24 16:09:20 2025 +0200

    Add a comment to hello.cpp::say_hello

    Signed-off-by: Emmanuel LE TRONG <emmanuel.le-trong@cnrs-orleans.fr>

diff --git a/hello.hpp b/hello.hpp
index 1241ffd..6729b19 100644
--- a/hello.hpp
+++ b/hello.hpp
@@ -1,9 +1,10 @@
 #pragma once
```

```
-#include <iostream>
+#include <string>

    namespace
 calculisto::sample_project
 {
+// This function says hello
    void
 say_hello (std::string const& who = "world");
 } // namespace calcul_isto::sample_project
```

Notice the `Signed-off-by` line. In the diff you can see that not only did I add a comment but I also replaced an include directive in the file.

Before doing the second commit, let's check the log

`$ git log`

gives

```
commit e042a59268a750b9eff856f135f3854023dee9d2 (HEAD -> new_feature)
Author: Emmanuel LE TRONG <emmanuel.le-trong@cnrs-orleans.fr>
Date:   Wed Sep 24 16:09:20 2025 +0200

    Add a comment to hello.cpp::say_hello

    Signed-off-by: Emmanuel LE TRONG <emmanuel.le-trong@cnrs-orleans.fr>

commit 00c3f8c0a0ef551910dc7bae0f8c95c2cee47611 (origin/main, origin/HEAD, main)
Author: Emmanuel LE TRONG <emmanuel.le-trong@cnrs-orleans.fr>
Date:   Wed Sep 24 10:52:28 2025 +0200

    Add a README

    Signed-off-by: Emmanuel LE TRONG <emmanuel.le-trong@cnrs-orleans.fr>

commit b95e57c96edf0cf97eba4fe8ec47c603a4735a58
Author: Emmanuel LE TRONG <emmanuel.le-trong@cnrs-orleans.fr>
Date:   Wed Sep 24 10:40:42 2025 +0200

    Initial Commit

    Signed-off-by: Emmanuel LE TRONG <emmanuel.le-trong@cnrs-orleans.fr>
```

This lists the commits and also tells me where the branches are pointing to.

Let's do the second commit

```
$ git add AUTHORS
$ git commit -s -v
```

I didn't use the `-s` option so `git` opened an editor so I can type my commit message. The `-v` options shows the diff in the editor to remind me of what has changed. The working tree is now clean

`$ git status`

gives

```
On branch new_feature
nothing to commit, working tree clean
```

Let's say that I am now ready to share my work.

3.5 Publish your contribution

First I push my new branch to my repository

```
$ git push -u origin new_feature
```

gives

```
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 763 bytes | 381.00 KiB/s, done.
Total 6 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), done.
remote:
remote: Create a pull request for 'new_feature' on GitHub by visiting:
remote:      https://github.com/le-migou/hello_fork/pull/new/new_feature
remote:
To https://github.com/le-migou/hello_fork
 * [new branch]      new_feature -> new_feature
branch 'new_feature' set up to track 'origin/new_feature'.
```

We are going to use `GitHub` pull request feature which is slightly simpler than `git`'s. Go to your fork's home page, there should be a big green button labelled "Compare & pull request". Click it and you should be able to give your pull request a title and a description before creating it. The maintainer of the project will now be able to merge (accept) your contribution or not.