LAMMPS Code Clinic 2022

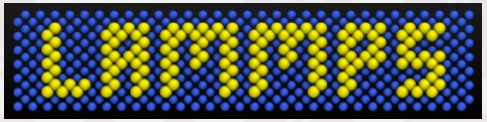
Dr. Axel Kohlmeyer

LAMMPS Core Developer

Associate Dean for High-Performance Computing,
College of Science and Technology

Temple University
Philadelphia PA, USA

a.kohlmeyer@temple.edu



LAMMPS Development Process

- "Continuous release" process:
  - → LAMMPS code should always be usable
  - → known bugs are fixed as quickly as possible
  - → automated checks for compilation and style
  - → unit tests and regression tests
- Public Git repository on GitHub
  - → 3 main branches: *develop*, *release*, *stable*
  - → changes merged to *develop*
  - → releases with new features every 4-8 weeks
  - → stable releases 1-2 per year
- **All** contributions are included via Pull Requests

Suggested Git/GitHub Workflow (I)

- Create a GitHub account, upload ssh key
- Go to https://github.com/lammps/lammps in your web browser and create a fork
- On your desktop create a clone of your fork:
  ```
  git clone git@github.com:<UserID>/lammps.git
  ```
- Set up access to the upstream repository
  ```
  git remote add upstream git@github.com:lammps/lammps.git
  git fetch upstream
  ```
- Connect “develop” branch to upstream
  ```
  git checkout develop
  git branch -u upstream/develop
  ```

Suggested Git/GitHub Workflow (II)

- Create a feature branch for your work
  ```
  git checkout -b improve-errors develop
  ```
- Work on changes, commit frequently when a related group of changes is done, push to fork
  ```
  git push origin -u improve-errors
  ```
- When you have a sufficient set of changes, submit your branch as a pull request
- You may be asked to make additional changes, LAMMPS developers may add changes, too.
  ```
  git checkout improve-errors; git pull
  […]
  git commit; git push
  ```

Suggested Git/GitHub Workflow (III)

- You may work on multiple feature branches concurrently and switch between them
- It may be needed to update “your” *develop* branch or your feature branches to include changes from upstream:

```
git checkout develop; git pull
git checkout improve-errors
git merge develop
```

- **Never** commit any changes to develop, to undo

```
git checkout develop
git reset upstream/develop
git checkout improve-errors
```

Suggested Git/GitHub Workflow (IV)

- Before submitting a pull request, test that your code compiles and passes all checks (the ci.lammps.org server will also run these checks and some more and block merging on failures):
  - `cd src; make check`
  - `cd doc; make html; make spelling`
  - `cd build; make ; ctest`
- Running tests with ctest requires compilation with CMake and -DENABLE_TESTING=on
- https://docs.lammps.org/Howto_cmake.html
  https://docs.lammps.org/Howto_github.html

Prerequisite Software

- For Code Clinic development, you should configure LAMMPS with as many packages and features enabled as possible (to detect conflicts)
- Prequisites for Fedora/RedHat/Ubuntu Linux can be seen from the singularity definition files.
- For Windows you are best off using Visual Studio 2022 and install MS-MPI package + SDK
- If you don’t want to change your local installation you can consider using a virtual machine or singularity/apptainer container

Project 1: Improve Error Messages

- Detailed basic explanations are at: https://www.lammps.org/workshops/Aug22/project1/
- Error messages should remain brief (1-2 lines of text) Just add the extra bit of information needed to figure out what the cause is from the manual.
- You can sometimes find inspiration here: https://docs.lammps.org/Errors_messages.html
- Errors with complex explanations or multiple causes should have an explanatory paragraph in the manual and just call the utils::errorurl() function pointing there. This is a last resort and should be avoided.

Project 2: Modernize LAMMPS

- Detailed basic explanations are at: https://www.lammps.org/workshops/Aug22/project2/
- You are advised to only do one kind of change in a given feature branch. That will make the review easier. Use multiple feature branches if necessary.
- This project is rather open ended. Please ask for assistance before you do larger changes or if you are unsure about how far to refactor the code.
- Please also consider applying coding style changes as they are outlined here: https://docs.lammps.org/Modify_style.html
  Often code is formatted to72 cols while we use 100 now

Project 3: Update External Code

- Detailed basic explanations are at:
  https://www.lammps.org/workshops/Aug22/project3/
- For this project the workflow for the lammps repository would need to be changed to the lammps-plugins repo, if the code is intended to be included there.
  Submission for inclusion to either repository is optional
- Update manual for yet undocumented issues
  https://docs.lammps.org/Developer_updating.html
- Please also consider applying coding style changes as they are outlined here:
  https://docs.lammps.org/Modify_style.html
  Often code is formatted to72 cols while we use 100 now

Project 4: Modernize the Manual

- Detailed basic explanations are at: https://www.lammps.org/workshops/Aug22/project4/
- For this project using Linux is **required**. The build process is only tested on Linux.
- For a changes including ":math:" commands (or roles) the translation of the manual also needs to be checked with "make pdf" which requires a fairly complete LaTeX distribution with several packages. Issues with math expressions will not always be visible in HTML and PDFLaTeX is also more strict in what is allowed to use.
- For ".. versionadded::" you can check the release notes: https://github.com/lammps/lammps/releases

Project 5: Adding Tests

- Detailed basic explanations are at: https://www.lammps.org/workshops/Aug22/project5/
- Tests exist at multiple levels:
  - Tests of individual functions and standalone classes
  - Tests of individual LAMMPS commands
  - Tests for the C++, C, and Fortran Library interfaces
  - Tests for the LAMMPS Python module (in Python)
  - Tests for complex LAMMPS operations on force styles (pair, bond, etc.) using generic executables and input files with reference data in YAML format
- Tests use googletest or Python unittest
- Enable code coverage to detect untested code paths

Project Difficulty Levels

- Project 1 requires basic C++ knowledge
- Project 2 requires a solid C++ understanding including some features added in C++11
- Project 3 requires average C++ understanding
- Project 4 requires no programming knowledge, (unless you call writing LaTeX programming)
- Project 5 requires a solid C++ understanding **and** a good understanding of using LAMMPS
- You welcome to contribute to one or multiple

Communication Among Us

- The preferred way to communicate during the Code Clinic event will be using Slack, and by preference in the #general channel to keep it public so that everybody can learn and know.
- If needed, you can also request a private Zoom session, e.g. if you need to share your screen to demonstrate an issue.
- Emails should go to developer@lammps.org
- Please keep in mind that we are in a wide range of different time zones, so please be patient.

Questions?