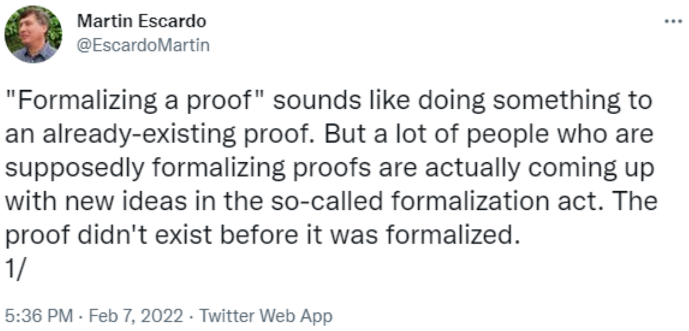
Getting Lean up and Running

Matej Penciak

Northeastern University

February 11, 2022
Slides available at:
`https://github.com/mpenciak/Lean-Seminar-Sp2022`

A good tweet



Martin Escardo
@EscardoMartin

"Formalizing a proof" sounds like doing something to an already-existing proof. But a lot of people who are supposedly formalizing proofs are actually coming up with new ideas in the so-called formalization act. The proof didn't exist before it was formalized.
1/

5:36 PM · Feb 7, 2022 · Twitter Web App

`https://twitter.com/EscardoMartin/status/1490816890414972930`

Parts of the toolchain

0. Bash (or any other shell)
1. Lean 3
   - lean
   - elan
   - leanpkg
2. Git
3. Python 3
   - mathlibtools
4. VS Code or emacs
   - Including the lean extension

Shell

This should only matter if you're on Windows. There are a number of options to get a Unix-like shell on Windows (msys2, cygwin) but the recommended way is to get the git bash client from
`https://gitforwindows.org/`
(this has the added benefit of dealing with step 2 as well).

If you're feeling fancy pretty much everything below works with other shells as well (zsh, fish, ...) with maybe slight modifications to add things to your path variable when the time comes.

Lean

Lean is distributed with handy install scripts most platforms. Just follow the instructions for your particular setup at `https://leanprover-community.github.io/get_started.html`. They should all be fairly painless (except maybe on new Macs with M1 chips)

What was installed:

- lean (Lean 3): This is the essence of Lean. It includes the interpreter
- elan: (Kind of like venv for Python or cabal for Haskell)
- leanpkg: A built-in dependency and package manager (like pip for Python)

Git

Git (for those of you who are unaware) is a handy program that creates repositories and can handle version control of said repos.

Windows: Hopefully you got it along with Git Bash above.

Linux: Most (all?) distros should already have it installed. If not just use your package manager.

OSX: A few options are available to install it. Probably the easiest is to get the installer binary `https://sourceforge.net/projects/git-osx-installer/`.

Python

Again, all Linux distributions and OSX come with Python pre-installed so this only affects Windows users.On Windows just install the latest version of Python at `https://www.python.org/`.

It may seem strange that we need Python in order to work with Lean, and to a certain extent this is true and will change with Lean 4.

Until then you will need a python utility called leanproject. Get it with `python -m pip install mathlibtools`.

VS Code

Lean has a limited list of editors that support interactive theorem proving. This list includes VS Code and emacs (and unofficially neovim `https://github.com/Julian/lean.nvim/`).

The easiest to get up and running is VS Code (**Opinion:** Which all of you should be using anyway)
`https://code.visualstudio.com/`
(for non debian Linux distros, use your package manager)

You will also need the Lean extension. Don't get lean4 (version v0.0.63), get lean (version v0.16.45).

Lean Projects

If everything is working correctly you should be able to run

```
leanproject get
https://github.com/mpenciak/Lean-Seminar-Sp2022.git
```

and download these slides, all the demos, and everything that comes after that.

`leanproject` is an amalgamation scripts that run leanpkg to set up the structure of the Lean project and add mathlib as a dependency, and git to get all the dependencies.

Keep up to date by running `leanproject pull` occasionally.

Lean Projects

If everything is working correctly you should be able to run

```
leanproject get
https://github.com/mpenciak/Lean-Seminar-Sp2022.git
```

and download these slides, all the demos, and everything that comes after that.

`leanproject` is an amalgamation scripts that run leanpkg to set up the structure of the Lean project and add mathlib as a dependency, and git to get all the dependencies.

Keep up to date by running `leanproject pull` occasionally.

It may be worth putting all of your completed exercises in a separate folder, so they don't accidentally get deleted in a merge.

Lean Projects structure

1. `leanpkg.toml` File that keeps track of the package information, dependencies, and more
2. `leanpkg.path` File that tells leanpkg where to look for dependencies when we type things like `import data.nat.basic`
3. `.gitignore` File that tells what files/folders git can ignore.
4. `./src/` Put all of your .lean files in this folder!
5. `./_target/` This is where all of the dependencies are installed for the project. Note this is in .gitignore because leanproject knows where to look for them from `leanpkg.toml`

Lets get to actually using Lean!

Either run `leanproject new <name>` to start a new project with mathlib dependencies or `leanproject get <name>` to start working on an ongoing project.

Everything should be working if when you open `/src/week1/Demo1.lean` your editor doesn't yell at you, and the "lean infoview" panel opens automatically.

Navigating Mathlib

Mathlib is huge and built up of a ton of files. You will often find yourself needing to figure out what's already been done, where it's written down, and how to import and use that specific result

Navigating Mathlib

Mathlib is huge and built up of a ton of files. You will often find yourself needing to figure out what's already been done, where it's written down, and how to import and use that specific result

Any leanproject has a copy of mathlib which you can navigate through in a variety of different ways:

1. Just scrolling through the folders. The folders are organized by subject area, and lemma names are chosen to be descriptive.
2. Searching using the built-in VS Code search.
3. Using the online mathlib documentation `https://leanprover-community.github.io/mathlib_docs/index.html`.
4. Other handy QOL additions (ctrl-click, links in the infoview, etc...).

Importing and dealing with namespaces

For example, if I need a basic result about group actions that I found in
`_target/deps/mathlib/src/group_theory/group_actions/basic.lean`
write: `import group_theory.group_action.basic` at the top of the file.

Importing and dealing with namespaces

For example, if I need a basic result about group actions that I found in
`_target/deps/mathlib/src/`
`group_theory/group_actions/basic.lean`
write: `import group_theory.group_action.basic` at the top of the file.

This is because `leanpkg.toml` has `_target/deps/mathlib/src` in its path, so all we need to tell Lean is how to get to our file from there.

Importing and dealing with namespaces

For example, if I need a basic result about group actions that I found in
`_target/deps/mathlib/src/group_theory/group_actions/basic.lean`
write: `import group_theory.group_action.basic` at the top of the file.

This is because `leanpkg.toml` has `_target/deps/mathlib/src` in its path, so all we need to tell Lean is how to get to our file from there. Also imports have to be the first lines of the file (weird quirk about Lean)

Using the lemmas

Another way Lean helps reduce conflicts been different object names is by using *namespaces*. You will likely have seen something like `namespace mul_action`, which tells you everything in that particular section has to be cited as `mul_action.what_I_want`.

Using the lemmas

Another way Lean helps reduce conflicts been different object names is by using *namespaces*. You will likely have seen something like `namespace mul_action`, which tells you everything in that particular section has to be cited as `mul_action.what_I_want`.

Note: Namespaces are separate from import location. This is similar to languages like C and C++ (what Lean was originally written in), but should be contrasted with how things work in Python.

Using the lemmas

Another way Lean helps reduce conflicts been different object names is by using *namespaces*. You will likely have seen something like `namespace mul_action`, which tells you everything in that particular section has to be cited as `mul_action.what_I_want`.

Note: Namespaces are separate from import location. This is similar to languages like C and C++ (what Lean was originally written in), but should be contrasted with how things work in Python.

A quick way to see if you've gotten the import rights is to use `#check object_name`. If Lean can properly find that object, then it'll report back with it's type.

Goals for next time

Try to finish up the Natural Number Game `https://www.ma.imperial.ac.uk/~buzzard/xena/natural_number_game/`

Work on the short exercise sheet found at `src/week2/exercises.lean` where you'll go on a scavenger hunt through mathlib!