

How To Make Package Managers ~~Cry~~ Scream

Kenneth Hoste

FOSDEM 2026 Main Track | <https://fosdem.org>

Email: kenneth.hoste@ugent.be

GitHub: [@boegel](#)

BlueSky: [@boegel.bsky.social](#)

Fediverse/Mastodon: @boegel@mast.hpc.social

<https://linkedin.com/in/kenneth-hoste>

Yes, that font is Comic Sans.
No, that's not by accident...



How To Make Package Managers Cry Scream



Kenneth Hoste

FOSDEM 2026 Main Track | <https://fosdem.org>

Email: kenneth.hoste@ugent.be

GitHub: [@boegel](#)

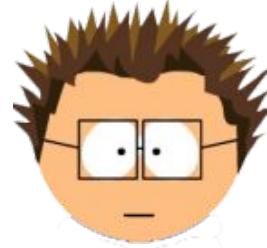
BlueSky: [@boegel.bsky.social](#)

Fediverse/Mastodon: @boegel@mast.hpc.social

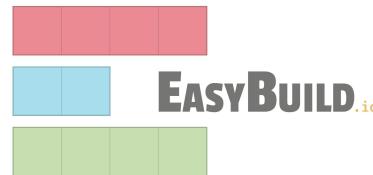
<https://linkedin.com/in/kenneth-hoste>

Yes, that font is Comic Sans.
No, that's not by accident...

whoami



- Supercomputer sysadmin @ Ghent University (Belgium) since 2010
- Open source software enthusiast for ~20 years (yes, I'm old)
- I also like family, (loud) gigs, beer (but I'm picky), stickers, dad jokes, ...
- FOSDEM attendee since 2012, (co-)organising HPC devroom since 2014
- Lead developer of [EasyBuild](#), core contributor to [EESSI](#), ...

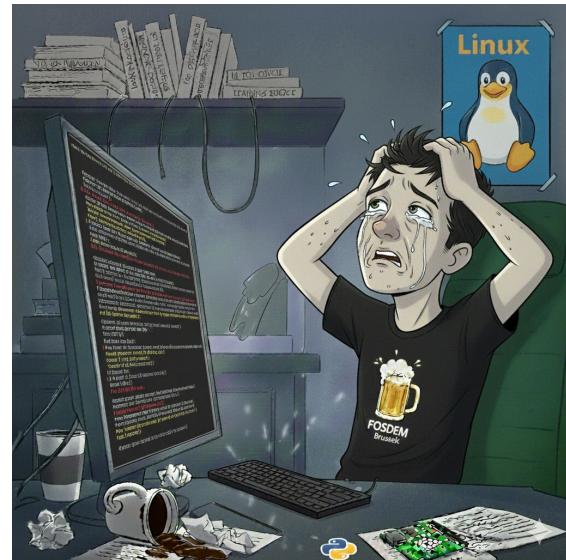


1000₂ years ago, at FOSDEM 2018

- ~25 min talk in Package Management devroom:
"How to Make Package Managers Cry"
😢
- Devroom got a bit crowded during my talk 😅
- Some people liked it!
- It even reached HackerNews (well, kind of)
 - RandomCSGeek also posted it in
["Ask HN: What is your favourite tech talk?" thread](#)
(April 2018) ❤️

<https://www.youtube.com/watch?v=NSemlYagjIU>

[https://archive.fosdem.org/2018/schedule/event/how to make package managers cry](https://archive.fosdem.org/2018/schedule/event/how_to_make_package_managers_cry)



H "how to make package managers cry"

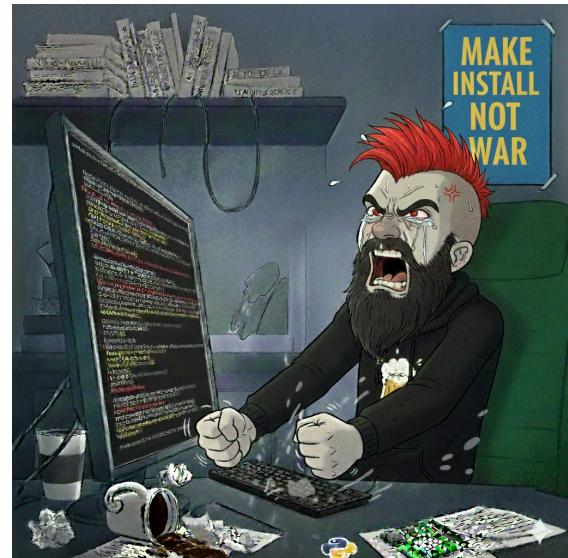
Stories Popularity All time

How to Make Package Managers Cry [video] (<https://www.hackernews.firebaseio.com/item/10002.json>)

3 points | Cmerlyn | 8 years ago | 0 comments

Today at FOSDEM 2026: the sequel !

- Initially submitted to Package Management devroom
- Got promoted to FOSDEM Main Track \o/
- More of the same, but also kinda different...
- We live in a (very) different world compared to 2018



I expect some heckling throughout the talk.

Make it happen. Do your part! 🍺

This talk is a little biased (again)

- My main experience is with installing scientific software on **supercomputers**
- Supercomputers ❤️ Linux (Windows, macOS or BSD: a lot less so...)
- We often need to **build software from source code** (it's fun, really!)
- **Software developed by scientists** can be particularly... interesting
- I generally focus on central installations on **multi-tenant systems**
- A lot of what I'll cover should also translate well to other communities...

Package managers are people too



- In this talk, package managers are the people who package software, so it can be installed/used easily (by themselves or others)
- If you “install” software for others, I consider you a package manager (ymmv)
- “First line responders”: they’re **exposed early** on to new software releases
- **Package managers don’t give up easily**, they like solving puzzles...
- They have seen a lot, but there are **more of us**, and we’re a **creative** bunch!

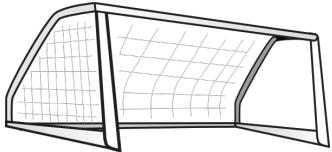
Who is this talk for?

- You like working with **computers**, coding, making them dance as you want them to
- You like showing your work to the world as **open source software**
- You don't like dealing with **people** who are using your software, or trying to use it....
 - They try to install your software in weird environments
 - They ask annoying questions, and expect you to have the answers
 - They report problems and "bugs" that you probably don't care much about
 - They even request additional changes, improvements, features, ...

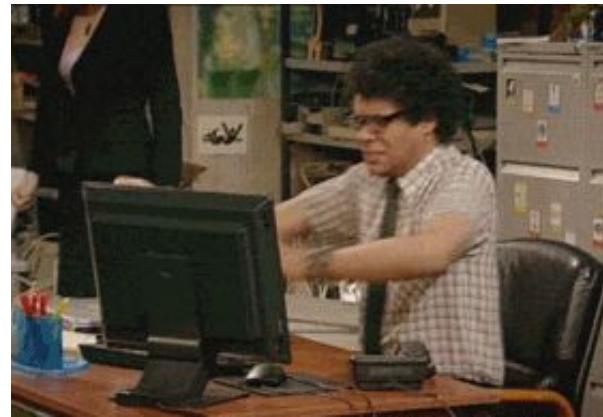
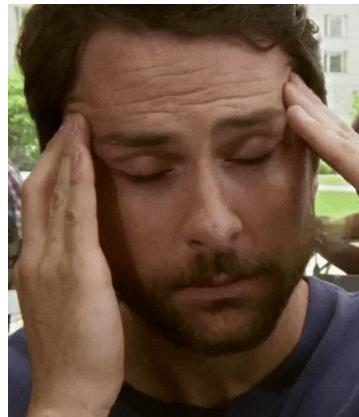
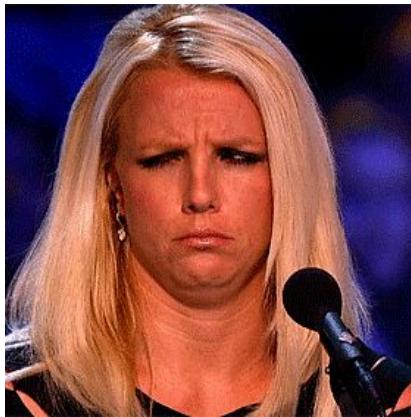
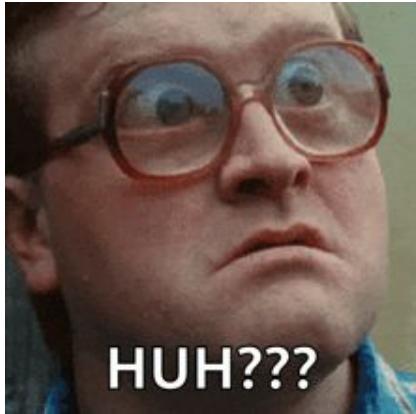
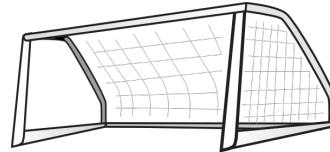
Why do we want to make package managers cry scream?!?

- Package managers make it easier for people to install & use your software
- We don't want that, since it eventually only results in more work for us
- We want package managers to give up on packaging your software
- If that doesn't work, we at least want to frustrate them, slow them down, ...
- This talk includes a bunch of techniques & tricks you can use
- Any package managers in the room?
 - If you're sitting next to a package manager, distract them!





Goals



Spoiler: Topics we'll be covering in this talk

- LLMs
- Naming is hard
- Terminology
- Code structure
- Documentation
- Hosting
- Self "packaging"
- Dependencies (surprise!)
- 🏆 Prizes (huh?!)
- Hidden tricks
- Build tools (and then some)
- Testing
- Programming languages
- Compilers
- Versioning

And some examples along the way...



WARNING

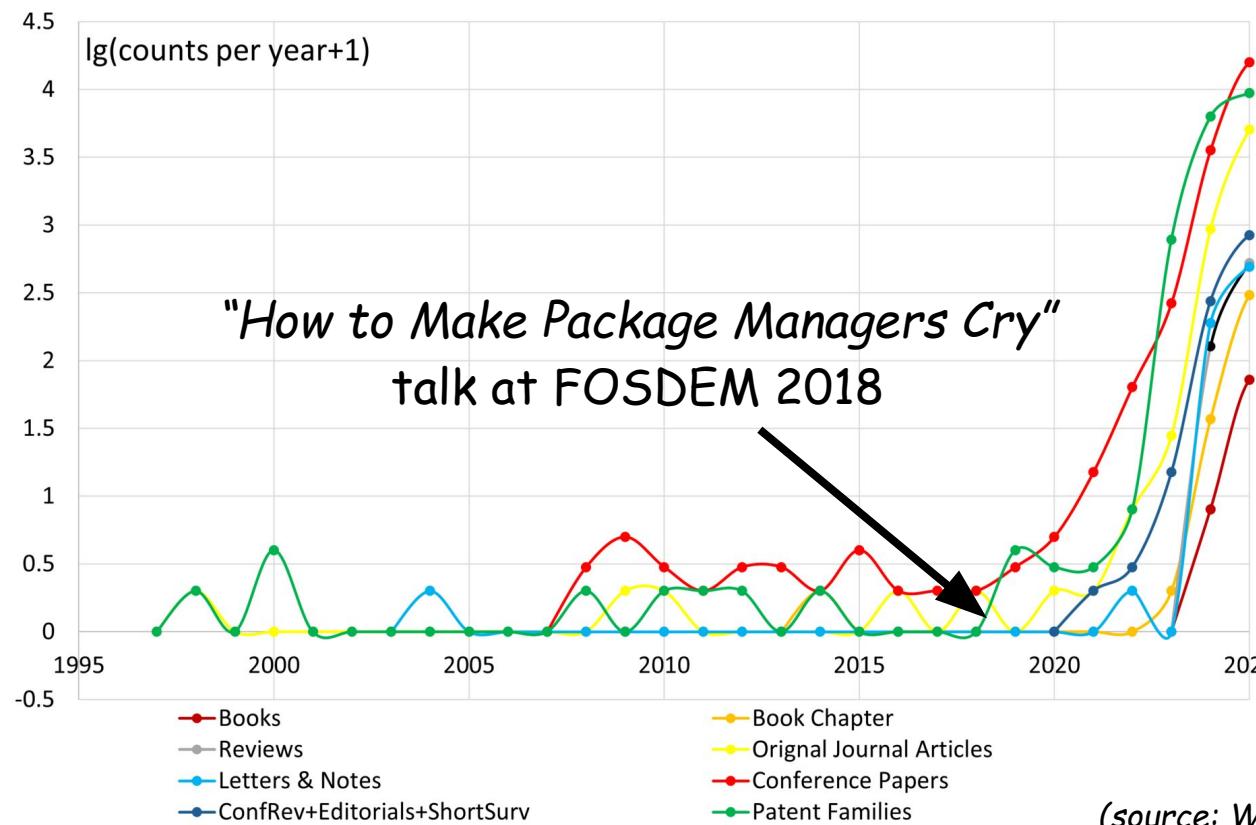
May contain traces of **SARCASM**

This talk is meant to be an eye-opener regarding
bad practices in software installation procedures.

Please do NOT interpret the given 'advice' as genuine.

I do NOT want to insult particular people or projects.

What caused explosion of publications on LLMs starting 2018?



LLMs can help us make package managers cry scream

- Let's get this out of the way, shall we...
- No need to learn a lot about the tools & techniques you will hear in this talk \o/
- If somebody complains, blame it on hallucinations of the LLM you used
- Tell package managers that they should let LLMs install the software instead
- Proposal for `install.md` standard to tell LLMs how to install your software

<https://www.mintlify.com/blog/install-md-standard-for-l1m-executable-installation>

- Please don't use this, may lead to better docs for package managers!



Choose the name of your project wisely

- Make sure it's **hard to search for**
- Aim for **maximum confusion** with other things/projects (animals! mythology!)
- Your project name should **imply something that's totally false**
- Use "**funny" characters**: special ASCII characters, Unicode (trademark sign!), ...
- Come up with ways to trigger frequent **typos**
- Be as **inconsistent** as you can, do what you can to encourage others to do the same

Some examples of naming done well

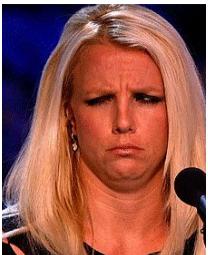
- Rust is a well-known programming language, but also a color, a popular game, a chemical process, a movie, ... 😳
- Single-letter names like C and R are hard to use in search engines 👍
- EasyBuild has over 250 configuration options (wasn't this supposed to be easy?!)
- "Ask in the Spack Slack for help to install Spark in a stack using a spec."
- Python packages on PyPI being forced to use _ instead of - in filenames
- OpenSpeedShop (a pipe!), LaTeX (max. camel-case), Ωmega (❤️ Unicode), ...

Particular example of well-chosen project name: PRRTE

- PRRTE is a key dependency of Open MPI (for distributed computing workloads)
- Short for “PMIx Reference Runtime Environment”
- PMIx is a standard, short for “Process Management Interface - Exascale”
- Both are kind of hard to pronounce (prrrrrr-tee? pretty? pee-em-ai-ex? pim-ex?) 
- Inconsistent naming across docs, GitHub, API, configuration, commands, ...

The project is formally referred to in documentation by “PRRTE”, and the GitHub repository is

[pr rte](https://github.com/pr rte).



Note

We have found that most users do not like typing the two consecutive  letters in the name. Hence, all of the internal API symbols, environment variables, MCA frameworks, and CLI executables all use the abbreviated   (one , not two) for convenience.

<https://docs.pr rte.org>

Names you should consider for your next project

- **Phoenix** (or derivatives like Feniks, Fenix, ...) - “rise from the ashes”
- **Hydra** - the “multiple heads” can mean many things (parallelism, ...)
- Anything else from Greek/Roman **mythology**: Thor, Helix, Titan, Orion, Zeus, ...
- Some play on **snakes**, to join the club: Python, Conda, Mamba, Cobra, Viper, ...
- **Open<something>** - especially if some aspect of the project is not open at all
- **Alpha<something>** - interesting if you want to win a Nobel Prize

Terminology

- **Don't use standard terminology** for things, invent your own (like "crates" in Rust)
- Use tools that have their own **custom terminology**
- Overload **commonly used terminology** even more by also using it;
"modules" is a particularly good one: Linux kernel modules, Fortran modules,
CMake modules, Python modules, Ansible modules, environment modules, ... 
- Try to **distort well established terminology** by giving your own spin to it:
"In my project, interface actually means something slightly different..."

Code structure & where stuff is located

- It's up to you how you structure your project, it's your project!
- It does not need to make sense to others (only to you, sort of)
- Distribute your source code across lots of different (deep) subdirectories
- Configuration & build tools look in standard places, so hide your stuff well
- You don't need to install binaries into `<prefix>/bin`, and nobody can make you
- Likewise for libraries in `lib(64)`, header files in `include`, etc.

Documentation

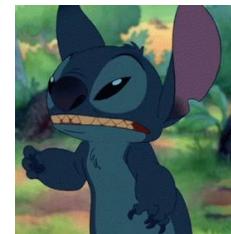
- You need to have some docs, you can't get away with not having any at all
- That doesn't it mean it can't be helpful to make package managers ~~cry~~ scream...
- Your documentation can be:
 - Minimal
 - Incomplete
 - Out of date
 - Confusing
 - Long and boring
 - Scattered across many different places
 - Barely searchable
 - Only a research paper, nothing more
 - Only available in an annoying format (like PDF)
 - Only (comments in) the source code

Hosting (code, website, releases)

- Host your project “somewhere else”, not on the usual platforms
 - Force package managers to have an account everywhere, just to ask questions
 - Make them learn the slightly different (and constantly changing) interface
 - Self-hosted GitLab is an interesting option (especially in a walled garden)
- Don't use Git, there are many several alternatives (check out CVS!)
- Be inconsistent, scatter releases: tag versions (some as release), mirror with infrequent sync, only upload some releases to PyPI, have an outdated website, ...
- Include source tarballs or zip files in a Git repository (yes, it *is* allowed)
- Split your project across (many) different repositories (cfr. AMD ROCmTM)

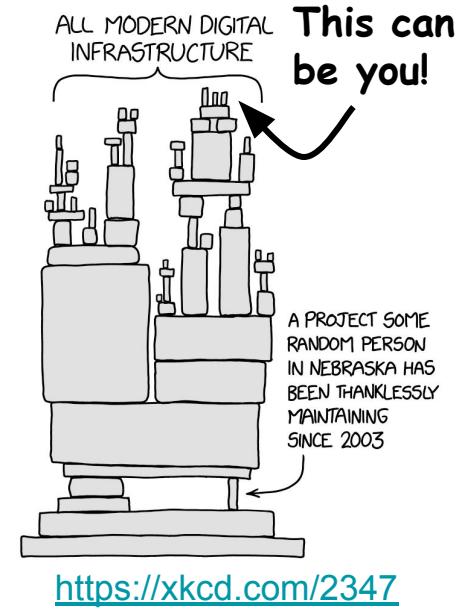
"Packaging" things yourself

- Take matters into your own hands, beat package managers at their own game
- Only provide instructions to do system-wide installation on Ubuntu (for example)
 - Some people *will* think that Ubuntu a hard requirement for your software...
 - Multiple cases of "*I tried to run sudo apt install example, but it didn't work*" from researchers using a supercomputer running a RHEL-based distro...
- Only provide a **Docker file** as installation instructions
- Release your software **only as a Jupyter notebook** (no other source files!)
Hat tip: "*I don't like notebooks*" talk: <https://youtu.be/7jiPeIFXb6U?si>
- Consider creating your very own **package manager** (cfr. AMD's TheRock™)



Dependencies

- More dependencies is better. Always.
- Choose them well, aim for maximum likelihood of trouble
- Look for **obscure** dependencies in particular
- Adopt newly **emerging** libraries ASAP
- You don't need to actually use all those dependencies in your code, **be reasonable**
 - Just make your configuration tool require them, error out if they're missing
 - Add include statements in your code, but don't use any of the provided functions
 - If somebody complains, just state that you're "future-looking"



<https://xkcd.com/2347>

Can you win prizes with making package managers cry scream?

- Yes, you can!
- AlphaFold (<https://github.com/google-deepmind/alphafold>)
- Tool to predict how proteins fold based on its sequence by Google DeepMind

Installation and running your first prediction

You will need a machine running Linux, AlphaFold does not support other operating systems. Full installation requires up to 3 TB of disk space to keep genetic databases (SSD storage is recommended) and a modern NVIDIA GPU (GPUs with more memory can predict larger protein structures).

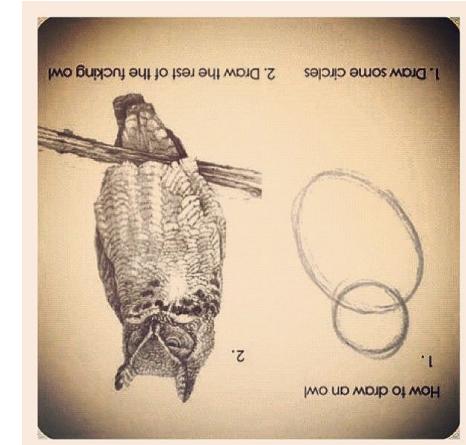
Please follow these steps:

1. Install Docker.

- Install [NVIDIA Container Toolkit](#) for GPU support.
- Setup running [Docker as a non-root user](#).

2. Clone this repository and cd into it.

```
git clone https://github.com/deepmind/alphafold.git  
cd ./alphafold
```



Can you win prizes with making package managers cry scream?

- Yes, you can!
- AlphaFold (<https://github.com/google-deepmind/alphafold>)
- Tool to predict how proteins fold based on its sequence by Google DeepMind

5. Build the Docker image:

```
docker build -f docker/Dockerfile -t alphafold .
```

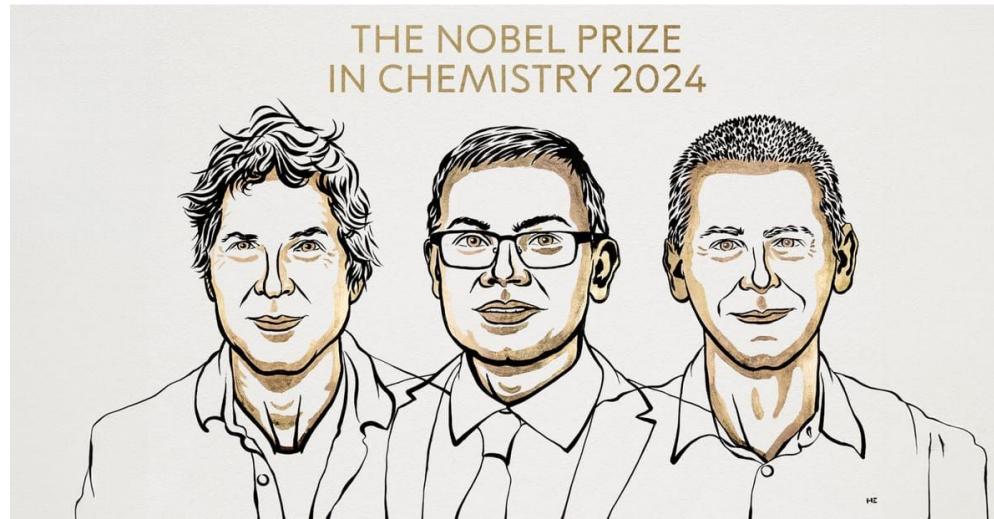
If you encounter the following error:

```
W: GPG error: https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64 I: 
E: The repository 'https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64' is not signed.
```

use the workaround described in [#463 \(comment\)](#).

Can you win prizes with making package managers ~~cry~~ scream?

- Yes, you can!
- AlphaFold (<https://github.com/google-deepmind/alphafold>)
- Tool to predict how proteins fold based on its sequence by Google DeepMind
- 🏆 Prize?



Hidden tricks

- You may need to take some measures to **keep yourself sane**...
- Implement **undocumented knobs** and features to make things easier for you
- Put stuff behind **environment variables** that nobody knows about
 - Make sure they are hard to find
 - Do not use project name as prefix
 - Use magic required values!
 - Example code in Python:

```
# BAD example
skip_flaky_tests = os.getenv('EXAMPLE_SKIP_FLAKY_TESTS')
if not skip_flaky_tests:
    run_tests()
```

```
# good example!
PROJ_PREF = 'EX'
key_parts = ['sk' + 'ip', 'fla' + 'ky', 'te' + 'sts']
env_var_name = PROJ_PREF + '_' + '_'.join(key_parts).upper()
s = os.getenv(env_var_name)
if s != 'YeS_Skip_em':
    run_tests()
```

Build tools

- Choose your build tools wisely... Aim for maximum damage
- CMake is always a good choice, it confuses and annoys everyone
 - For inspiration:
 - <https://xallt.github.io/posts/cmake-is-a-pain-in-the-ass>
 - <https://twdev.blog/2021/08/cmake>
 - From <https://news.ycombinator.com/item?id=34589687>:



▲ girvo on Jan 31, 2023 | parent | prev | next [-]

I would give up my firstborn to never have to deal with CMake again.

Build tools (part deux)

- You should **combine multiple** build tools, there's no reason to stick to just one
 - Call cmake, make, ... from setup.py (sure you can)
 - Use a script named configure.py, implement it in Perl 😈
- Have all of these in your repository for your Python project:
setup.py, setup.cfg, requirements.txt, pyproject.toml, environment.yml
- Guess what: build tools have **dependencies** too! \o/
- It doesn't stop with choosing one (or more), it's also about **how you use** them
- Think outside the box: use CMake as a *runtime dependency* in your software!
- If CMake is not enough for you, use Bazel instead (guaranteed success)



Build tools (yes, there's more)

- Do not read the docs of the tools you're using, or only use the outdated stuff
- Do not use well established tools according to best practices
 - Look up classic CMake vs modern CMake, use the former (well duh!)
 - Do still require that the *very latest* CMake version is used!
 - With a bit of luck, someone will hit some unexpected "regression" in CMake...
 - Pro tip: get alerts for new CMake releases via <https://newreleases.io>
- Beast mode: deliberately use hard to spot "typos" in filenames and commands
 - Example: ./nnake -f CMakeLitst.txt > pyproject.toml
 - Some day a package manager *will* fall for it and waste their weekend!

Testing

- Package managers like to be able to **test** the installation of your software
- They often **lack domain knowledge** for doing so well, **use that to your benefit**
- Aim for a test suite that **takes forever** to run
- **Flaky tests are your friend** (you know which ones to ignore)
- Don't pass on requiring **extra dependencies** to run your tests!
 - Package managers really hate having to do even more work just to run your tests

Particular example of test suites done well: PyTorch

- PyTorch has a massive test suite...
- **More than 250,000 tests**, takes ~36h to run on a single (recent, beefy) system
- Significant changes to the structure of the test suite every PyTorch release \o/
- Example test suite result for PyTorch 2.7.1 (as produced by EasyBuild):

```
WARNING: 52 test failures, 0 test errors (out of 261883):  
distributed/_composable/fsdp/test_fully_shard_state_dict (1 failed, 1 passed, 4 skipped, 0 errors)  
distributed/test_store (1 failed, 32 passed, 0 skipped, 0 errors)  
dynamo/test_compiler_bisection (1 failed, 6 passed, 0 skipped, 0 errors)  
higher_order_ops/test_invoke_quant (1 failed, 13 passed, 0 skipped, 0 errors)  
higher_order_ops/test_invoke_subgraph (1 failed, 19 passed, 1 skipped, 0 errors)  
inductor/test_aot_inductor (2 failed, 313 passed, 102 skipped, 0 errors)  
...
```

52 failing tests out of 261k => 99.98% of tests passing, is that good enough? 🤔

- A package manager person would be very tempted to try and reach 100%...

Live poll

- I will count to 3, and then you all SCREAM what you think is the correct answer
- Here comes the question...
- No screaming yet! Wait for the countdown!
- Which programming language is most likely to make package managers scream?
- Think about it first, no screaming yet!
- Are you ready for the countdown?

Live poll: programming language to make 'em SCREAM?

1

Live poll: programming language to make 'em SCREAM?

2

Live poll: programming language to make 'em SCREAM?

3

Live poll: programming language to make 'em SCREAM?

SCREAM

(some programming language, not just something random)



THE programming language to make 'em SCREAM

- Pfft, where do we begin...
- C++ standard library is a gem
- Error messages produced by C++ compilers are... waaaauw
- Templates in C++
- Metaprogramming, what a concept (*run the code at compile time!*)
- Both Linus Torvalds and the FBI recommend you shouldn't use C++. **So use it!**
- Bonus points for the name...
- Don't take my word for it, see "*The worst programming language of all time*"
<https://www.youtube.com/watch?v=7fGB-hjc2Gc> (rant of 2 hours 9 min!)



Random quote: "If you like C++, then you don't know it well enough"

Other programming languages you should consider using

- **Python**: the installation tools! `setuptools`! `Conda`! Python 4 on the horizon?!
- **Rust**: different enough to make your head spin, lots of custom terminology
- **FORTRAN**: since 1956, still in use (scientific software), #12 in TIOBE Index
- **C#**: just in case C++ isn't enough...
- **JavaScript**: the inconsistencies! Npm!
- **Go weird**: OCaml, Haskell, Prolog (WTF?!), Erlang, F#, Lisp, Bash, Scheme, Ada, ...
- There's a new programming language born every day, adopt them early on!

Compilers

- A.k.a. how to make C++ even worse than it already is...
- They're huge, complex, and impossible to avoid for any serious work
- **Screenfilling cryptic error messages** with C++ (it's worth repeating, admit it)
- Defaults change frequently (cfr. https://gcc.gnu.org/gcc-15/porting_to.html)
- **Compilers have dependencies too!** They even depend on each other!!!
- **Embrace the diversity**, explore alternatives, don't stick to **GCC**
- Try to make sure that **GCC can not be used** for some reason, and force package managers to use another set of compilers...

- Implemented in C++ \o/
- Only generic pre-built binaries available on PyPI (don't look up WheelNext!)
- Still uses Bazel as configuration & build tool \o/
- Clang compiler is now preferred over GCC
- Magic environment variables everywhere, but nowhere to be found in docs
 - They all start with `$TF_` (not `$TENSORFLOW_`)



≡  TensorFlow

Search results

No Results

[Search for `TF_NEED_CLANG` on Google](#)

Versioning

- Don't waste the opportunity to confuse people through **software versions**
- Suggest that you're using **semantic versioning**, but violate it (cfr. Python, Lua, ...)
- Start with calendar versioning (20260131), release 1.0, then switch to zero-over (0.x)
 - Let package managers figure out what the latest release is...
- HDF5
 - Stable releases have **even** minor number (1.8, 1.10)
 - Development "releases" (?) have an **odd** minor number (like 1.9, 1.11)
 - No more since HDF5 2.0, since they adopted semantic versioning (but, **why**!?)
 - https://support.hdfgroup.org/documentation/hdf5/latest/_releasenotes.html
- Encourage **forks**, especially without changing the name (see OpenFOAM!)

Things we did not get to cover (in detail)

- Python installation tools. Oh boy... (pro tip: stay away from uv !)
- Containers: build your own messy world, and get others to use it willingly somehow
- Downloading stuff during building/testing (yes cargo, I'm looking at you...)
- Misleading error messages, errors that are really just warnings (or vice versa), error messages that go out of there way to hide the actual problem (pip!), etc.
- Recent and upcoming CPU families (for scientific computing): Arm, RISC-V
It's about to get a whole lot worse very soon! \o/
- What else?! Come talk to me, I'll be here all FOSDEM, let's have a  !



How To Make Package Managers Cry Scream



Kenneth Hoste

FOSDEM 2026 Main Track | <https://fosdem.org>

Email: kenneth.hoste@ugent.be

GitHub: [@boegel](#)

BlueSky: [@boegel.bsky.social](#)

Fediverse/Mastodon: @boegel@mast.hpc.social

<https://linkedin.com/in/kenneth-hoste>

Yes, that font is Comic Sans.
No, that's not by accident...