Package management for FORM and its libraries

Vitaly Magerya (CERN)

FORM Developer's Workshop
NIKHEF, June 23, 2026

Historical note: Issue #322

Back in 2019:

FORM package manager #322

Open

tueda opened on Sep 7, 2019 — Last edited by **tueda** — Collaborator

Many modern programming languages have official or some de fact standard package managers that download and install programs or libraries. Maybe it is interesting to consider something similar in FORM. I came up with the following new preprocessor instruction,

```
#import benruijl/forcer@v1.0.0
```

3 months later:

tueda commented on Dec 2, 2019

Though no one (except me) is interested in this,

Why manage packages?

To solve computer problems:

* Package installation, deinstallation, upgrade, etc.

To address people problems:

* Making FORM libraries visible.
* Lowering the barrier for usage (of FORM and libraries).
* Lowering the barrier for participation (in library development).
* Lowering the barrier for cooperation (between library developers).
* Lowering FORM maintainace burden (feature deprecation without breaking users).

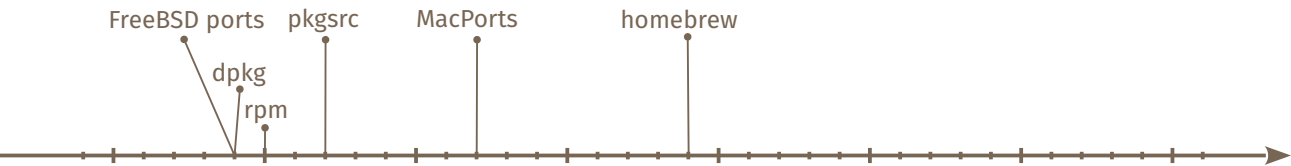
Early package management

In the beginning there was the *source distribution*:

```
<obtain the source>
<install dependencies>
./configure --some=settings && make && sudo make install
```

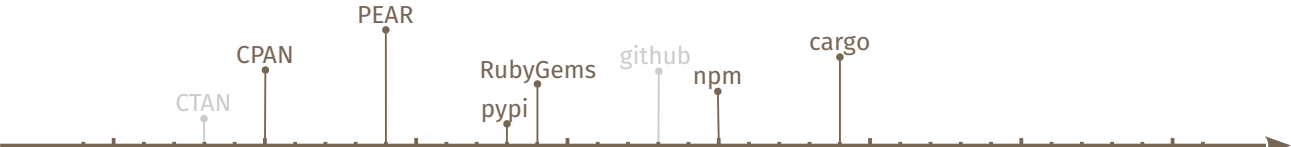
Enter *package management*:

* Making building standardized ⟶ collections of build instructions (*source packages*).
* Making installation fast & reproducible ⟶ archives of files to install (*binary packages*).
* Tracking installed files and programs ⟶ per-system package databases.



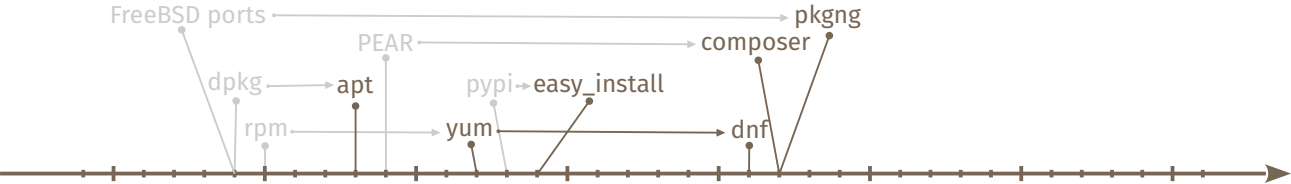
Feat: open package registries

* Anyone can submit their library.
  * No manpower needed for manual approval process.
  * Automated technical checks only.
* Each library supports standard building mechanism, acting as its own source package.
  * ... or a binary package.



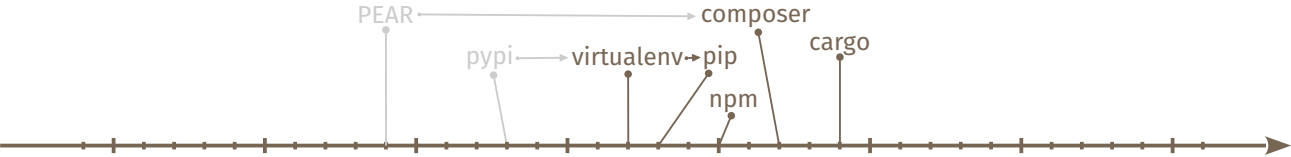
Feat: automated dependency resolution

* Installing dependent packages automatically.
* Choosing versions of dependent packages to ensure compatibility.
  * For example, pySecDec dependencies: “numpy>=1.23,<3”, “sympy>=1.10.1,!=1.11.*,!=1.12.*,<2”, ...
  * Modern package managers employ SAT solvers (and more) to find a compatible set of versions.



Feat: isolated local environments

* Setup an *isolated* set of libraries for each local “project” (i.e. directory).
* Prevents conflicts with incompatible library versions installed system-wide, or user-wide.
* Allows different versions in different directories.



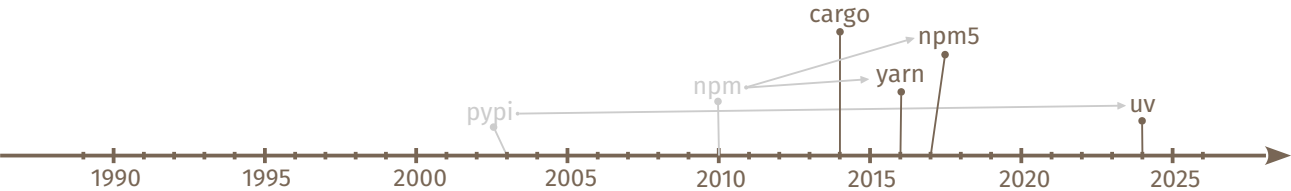
Feat: lock files

To prevent updates of dependencies of dependencies to break your program: arXiv:2505.04834

* Test the program with a set of dependencies.
* Write the exact version of each package in the full dependency tree into a *lock file*.
* Add the lock file to the code repository.

→ All users can reproduce the known-good build.

This is needed to keep applications with many dependencies stable.

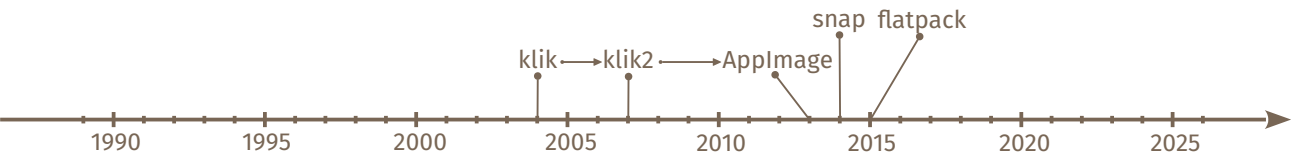


Feat: whole-system containers

* Package a whole Linux system, aside from the kernel, into an archive. Start/stop it on demand.
* Ensures reproducible execution across multiple programs, and no version conflicts.
* Used instead of package managers to package complex software.
  * E.g. multiple interdependent programs, like websites (server+database+background services).

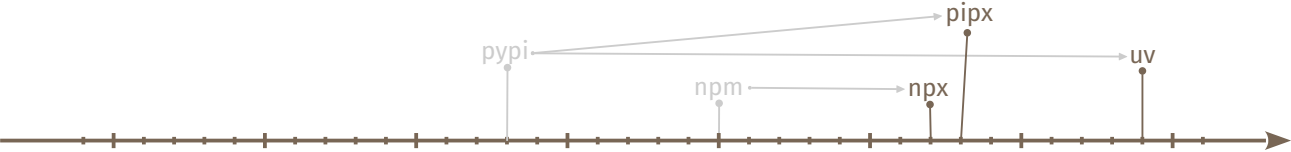
Feat: whole-program containers

* Store a program and all needed files and libraries (down to the C library) in an archive.
* When the program is executed: unpack, run, throw away once done.
* Ensures predictable execution of one program, no version conflicts.



Feat: transient installation

* Install a package and all of its dependencies on-the-fly in a temporary environment.
* Throw away the environment when the program is done.



Comparing package managers

	Apt, yum, etc (Distro p.m.)	Spak (HPC p.m.)	Cargo (Rust p.m.)	NPM (JS p.m.)	PyPI (Python p.m.)
Mature & well known	☺	☹	☺	☺	☺
Language-independent	☺	☺	😐	😐	😐
Handles native (C/C++) dependencies	☺	☺	☺	😐	😐
Open package registry	☹	☹	☺	☺	☺
Per-user installation	☹	☺	☺	☺	☺
Per-project installation	☹	☺	☺	☺	☺
Transient installation	☹	☹	😐	☺	☺
Optimized builds	☹	☺	😐	😐	😐
You already have it™	☹	☹	☹	☹	☺

Specific proposal

Package FORM, and libraries using Python ecosystem (PyPI, pip, uv, setuptools)!

https://pypi.org/
https://pip.pypa.io/
https://docs.astral.sh/uv/
https://setuptools.pypa.io/
https://setuptools-scm.readthedocs.io/

* Add full FORM installation as the package “form”.
  - 🚫 Currently the name “form” is taken by a name squatter.
    Temporary workaround: “form-bin”.
* Add FORM libraries with a “form-” prefix (“form-color”, “form-hyperform”, etc).
* Apply for a “Programming language :: FORM” category on pypi.org for FORM packages.

* * *

“But PyPI is for Python”.

* Other non-Python packages already on PyPI: clang-format, cmake, ninja, pkgconf, ziglang.
* pySecDec already bundles a FORM binary. https://pypi.org/project/pysecdec/
* Libraries like Takahiro’s python-form could depend on form-bin. https://pypi.org/project/python-form/

Demonstration

What is required of FORM?

1. Choose include paths to cooperate with local installation.
   * Pip will install FORM binary into `<prefix>/lib/python3.<n>/site-packages/form/form`.
   * If so, add `-I <prefix>/lib/python3.<n>/site-packages/form-packages/` automatically.
     - Maybe include other paths too. Basically, play along with whatever Python does.
     - Libraries are expected to install files into `<prefix>/lib/python3.<n>/site-packages/form-packages/libname/`.
     - Then, they can be imported with e.g. `#include libname/libname.h`
2. Setup a build system to produce binary packages in PyPI-compatible format.
   * This build system could live in the FORM repo, or in a separate repo.
   * Dependencies (GMP, MPFR, and FLINT) will need to be included.
   * Separate builds for separate glibc versions are commonly used.
     - Some testing w.r.t. C++ ABI stability is still needed (was a problem for pySecDec in the past).
     - Maybe a fully static binary is possible instead.
3. Release often. (Please!)

What is required of library authors?

Assume *the root* of your source is in FORMPATH.

* Use `#include name/myhelper.inc`

Add `pyproject.toml` to the source.

Tag a version number:

* `git tag v1.2.3`

Build:

* `python -m build`.

Register on pypi.org (or, first on test.pypi.org):

* See tutorials at packaging.python.org.

Upload with:

```
python -m twine upload \
dist/name-1.2.3-py3-none-any.whl
```

```
[project]
name = "form-color"
description = "A FORM package for calculating color group coefficients."
dependencies = ["form-bin>=4.0.0"]
readme = "README.md"
license = "GPL-3.0-or-later"
authors = [
   {name = "T. van Ritbergen"},
   {name = "A. N. Schellekens"},
   {name = "J. A. M. Vermaseren"}
]
maintainers = []
classifiers = [
  "Development Status :: 6 - Mature",
  "Topic :: Scientific/Engineering :: Physics"
]
dynamic = ["version"]

[project.urls]
Homepage = "https://www.nikhef.nl/~form/maindir/packages/color/color.html"
Repository = "https://github.com/magv/form-color"
Documentation = "https://www.nikhef.nl/~form/maindir/packages/color/color.pdf"

[build-system]
requires = ["setuptools>=79", "setuptools-scm[simple]>=9.2"]
build-backend = "setuptools.build_meta"

[tool.setuptools.package-dir]
"form-packages.color" = "color"
```

"Take version from git tags".

Subdirectory that will get installed.

Guide for users / pip

Install pip: https://pip.pypa.io/

* Comes with Python. You already have it!

User-wide FORM installation:

Install: `pip install --user --break-system-packages form-bin form-color`

Upgrade: `pip install --user --break-system-packages -U form-bin form-color`

Uninstall: `pip uninstall --user --break-system-packages form-bin form-color`

Installation into the current directory:

Prepare: `python3 -m venv .venv; source .venv/bin/activate`

Install: `./.venv/bin/pip install form-bin form-color`

Upgrade: `./.venv/bin/pip install -U form-bin form-color`

Uninstall: `./.venv/bin/pip uninstall form-bin form-color`

Guide for users / uv

Install uv: https://docs.astral.sh/uv/

* Download from github.com/astral-sh/uv/releases and unpack (single executable).

User-wide FORM installation:

Install:	`uv tool install form-bin --with form-color`
Upgrade:	`uv tool upgrade form-bin`
Uninstall:	`uv tool uninstall form-bin`

Installation into the current directory:

Prepare:	`uv venv; source .venv/bin/activate`
Install:	`uv install form-bin form-color`
Upgrade:	`uv pip install -U form-bin form-color`
Uninstall:	`uv pip uninstall form-bin form-color`

Packaging optimized builds

What about optimized builds for different CPU extensions? E.g. AVX, FMA, BMI, etc?
Surely, everyone should compile with `-march=native -mtune=native`, right?

Let's see benchmark timings:

Arch	chromatic	color	fmft	forcer	forcer-exp	hyperform	ibp-box2l	ibp-mbox1l	ibp-npbox2l	ibp-pent1l	ibp-tri3l	minceex	mincer	mzv-dm	sort-disk	sort-large	sort-small	trace
x86-64-v1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
x86-64-v2	~	~	~	~	~	~	+2%	+2%	~	~	+1%	~	~	~	~	~	~	~
x86-64-v3	-8%	-3%	~	-2%	~	~	~	~	-5%	-2%	-1%	-5%	+4%	~	~	~	+3%	+2%
x86-64-v4	-9%	+3%	~	-3%	~	~	-4%	-2%	-5%	-3%	~	-7%	~	~	~	~	+3%	+8%
"Native"	-7%	+2%	~	-1%	~	~	-3%	~	-5%	-5%	~	-7%	+6%	+3%	~	+7%	+9%	+7%

FORM v5.0.0, `tform -w8`, GMP 6.3, MPFR 4.2.2, FLINT 3.5, GCC 15.2, Zen4 AMD server

Takeaway: x86-64-v3 (~AVX2) and x86-64-v4 (~AVX512) are potentially useful, but more insight is needed.

Packaging optimized builds: strategies

Build implementations for different CPUs, and switch to an appropriate implementation, *at some point*:

I. At the function level.
   * I.e. compile a function multiple times for different CPU flags, and select the correct one at runtime.
   * GNU __attribute__((ifunc)) automates this (used by e.g. glibc).
     - Will patch the PLT (procedure linkage table) entry of the function at the first call.
     - No overhead on later calls.
   ⃠ Need GMP & FLINT to cooperate, which they don't.

II. At the program level.
   * I.e. compile the whole FORM multiple times, and select the correct binary at program start.

III. At the package level.
   * I.e. provide separate form[x86-64-v3] and form[x86-64-v4] packages.
   * Some packages do it, but it's super inconvenient for the user.

Call to action

Let's take FORM packaging from 1994 to 2026!

For that we need:

* approval & participation of the FORM developers;
* approval & participation of the library authors.

Short term goal (*this week*):

* FORM and multiple main libraries easily installable in a modern-package-manager way.

Long term goal:

* People improving libraries that they didn't write themselves.
* New FORM libraries that depend on other FORM libraries.

Open issues

If lib-A defines macro X(), and lib-B defines macro X(), how to avoid conflict?

* Name prefixes. Make sure lib-A defines libA_X(), and lib-B defines libB_X().
* Package namespaces? ⟶ That would be a new FORM feature.
  * Nontrivial interaction with preprocessor.

If lib-A defines some *input* names to be symbols, but lib-B defines them to be indices, how to cooperate?

* Prefixes and namespaces solve this for *internal* names, not for *external* ones.