The Catch in Rye

Seeding Change and Lessons Learned

Armin *@mitsuhiko* Ronacher

Who am I?

Armin @mitsuhiko Ronacher

Things you might know I worked on: *Flask, Werkzeug, Jinja, Pygments, Sphinx, LogBook, itsdangerous, Click, MarkupSafe, Sentry, Babel, …*

… and then I disappeare

So what did I do?

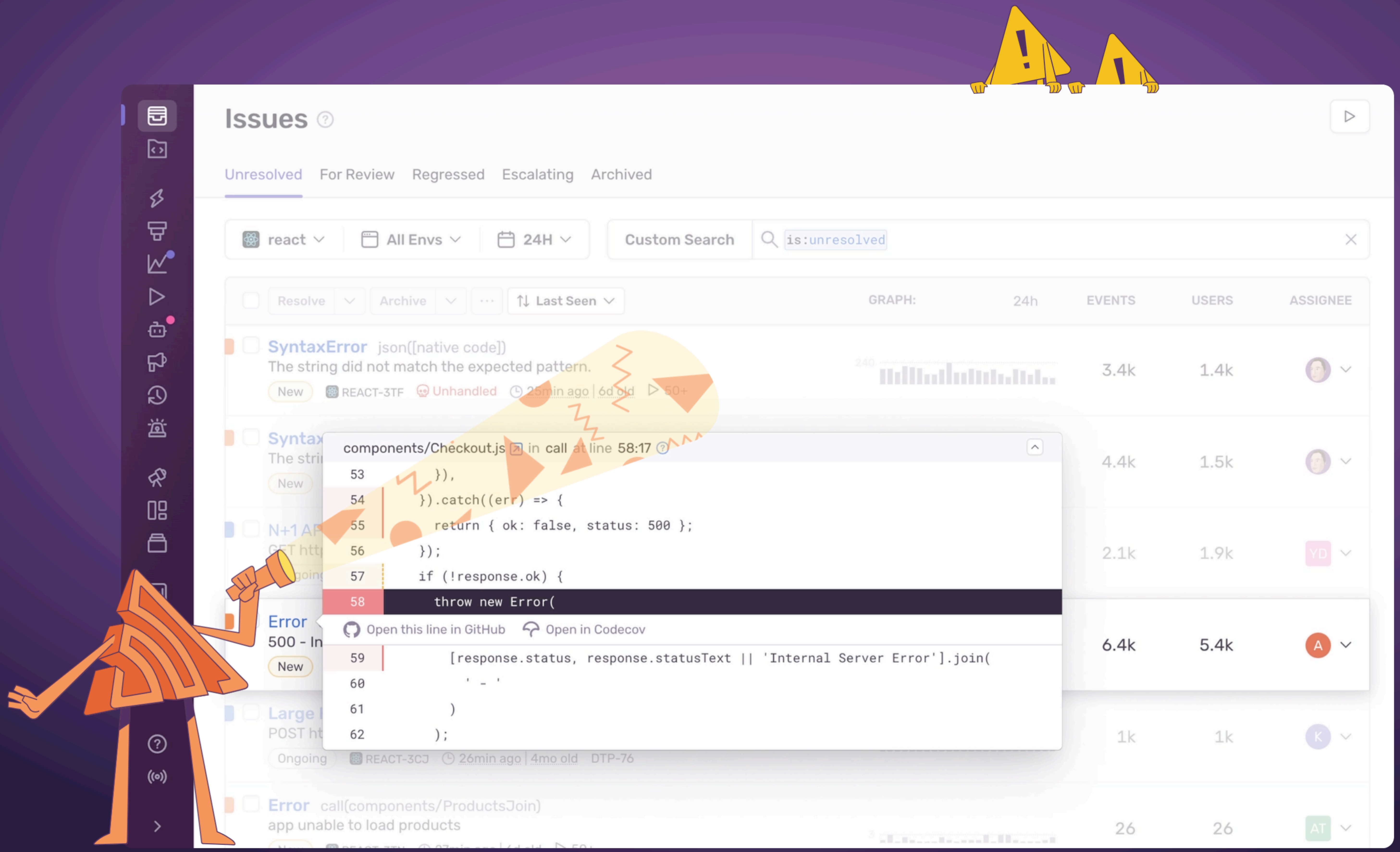
Poured a lot of time into Sentry

Started to enjoy the green pastures of Rust

SENTRY

Code breaks, fix it faster

Application monitoring software considered "not bad" by 4 million developers.



Issues

Unresolved · For Review · Regressed · Escalating · Archived

react · All Envs · 24H · Custom Search · is:unresolved

	GRAPH: 24h	EVENTS	USERS	ASSIGNEE
SyntaxError json([native code]) The string did not match the expected pattern. New · REACT-3TF · Unhandled · 25min ago | 6d old · 50+		3.4k	1.4k	
Syntax… The stri… New		4.4k	1.5k	
N+1 AP… GET http… Ongoing		2.1k	1.9k	YD
Error 500 - In… New		6.4k	5.4k	A
Large… POST ht… Ongoing · REACT-3CJ · 26min ago | 4mo old · DTP-76		1k	1k	K
Error call(components/ProductsJoin) app unable to load products		26	26	AT

components/Checkout.js in call at line 58:17

```
53      }),
54    }).catch((err) => {
55      return { ok: false, status: 500 };
56    });
57    if (!response.ok) {
58      throw new Error(
```

Open this line in GitHub · Open in Codecov

```
59        [response.status, response.statusText || 'Internal Server Error'].join(
60          ' - '
61        )
62      );
```

Rust

A language empowering everyone to build reliable and efficient software.

- Let's not kid ourselves: it's bloody complicated
- Yet as a programmer you're surprisingly productive with it
- The ecosystem has excellent DX
- The language values backwards compatibility
- The language values innovation and progress

Going back in Time (~2014)

- I picked up Rust properly when I used Python 2 actively
- Cargo was not yet a thing
- Python 3 was in a state of very slow and painful adoption

The Zen of Python

“There should be one — and preferably only one — obvious way to do it.”

The Zen of Python (cont.)

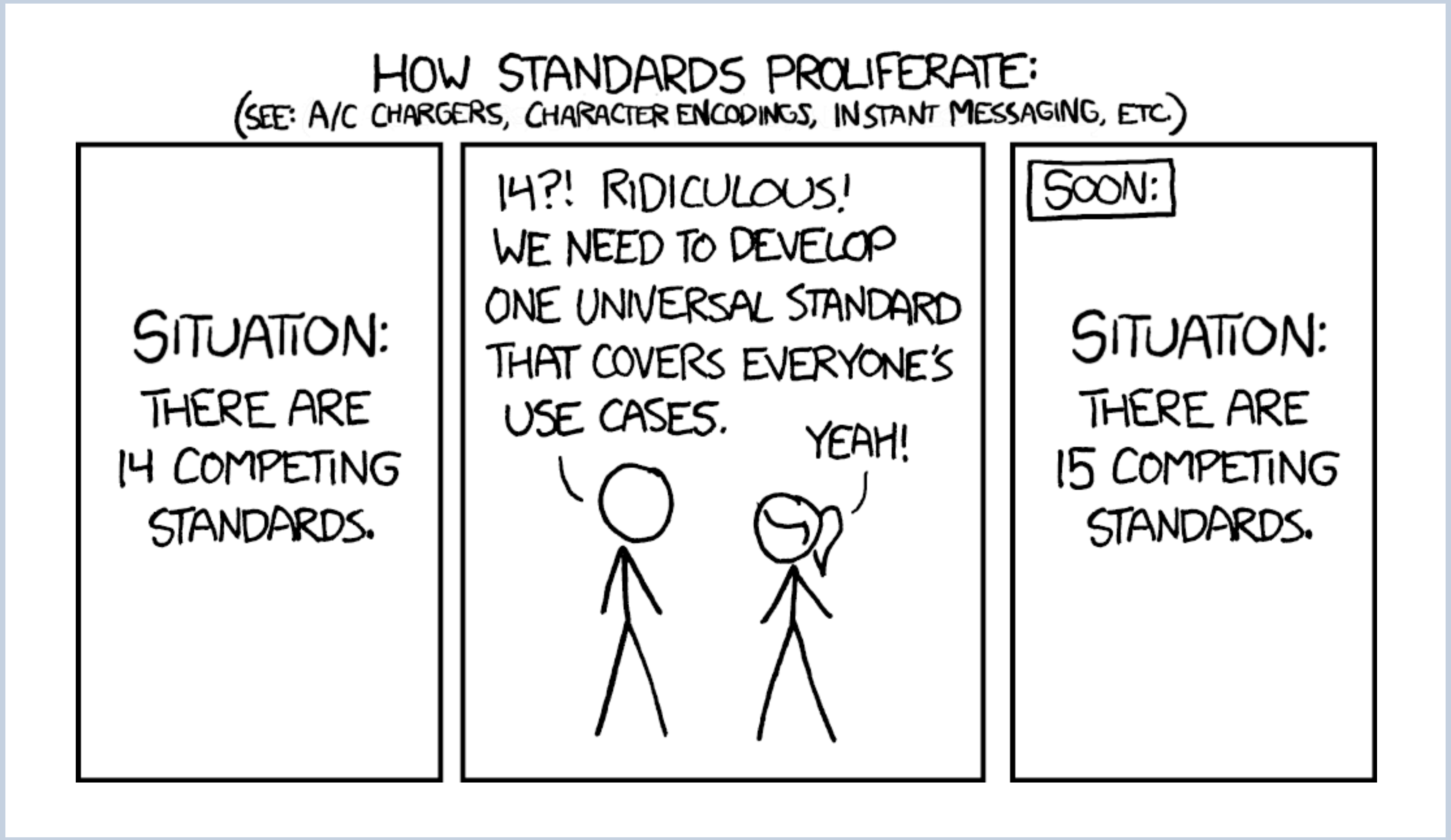
“Special cases aren't special enough to break the rules.”

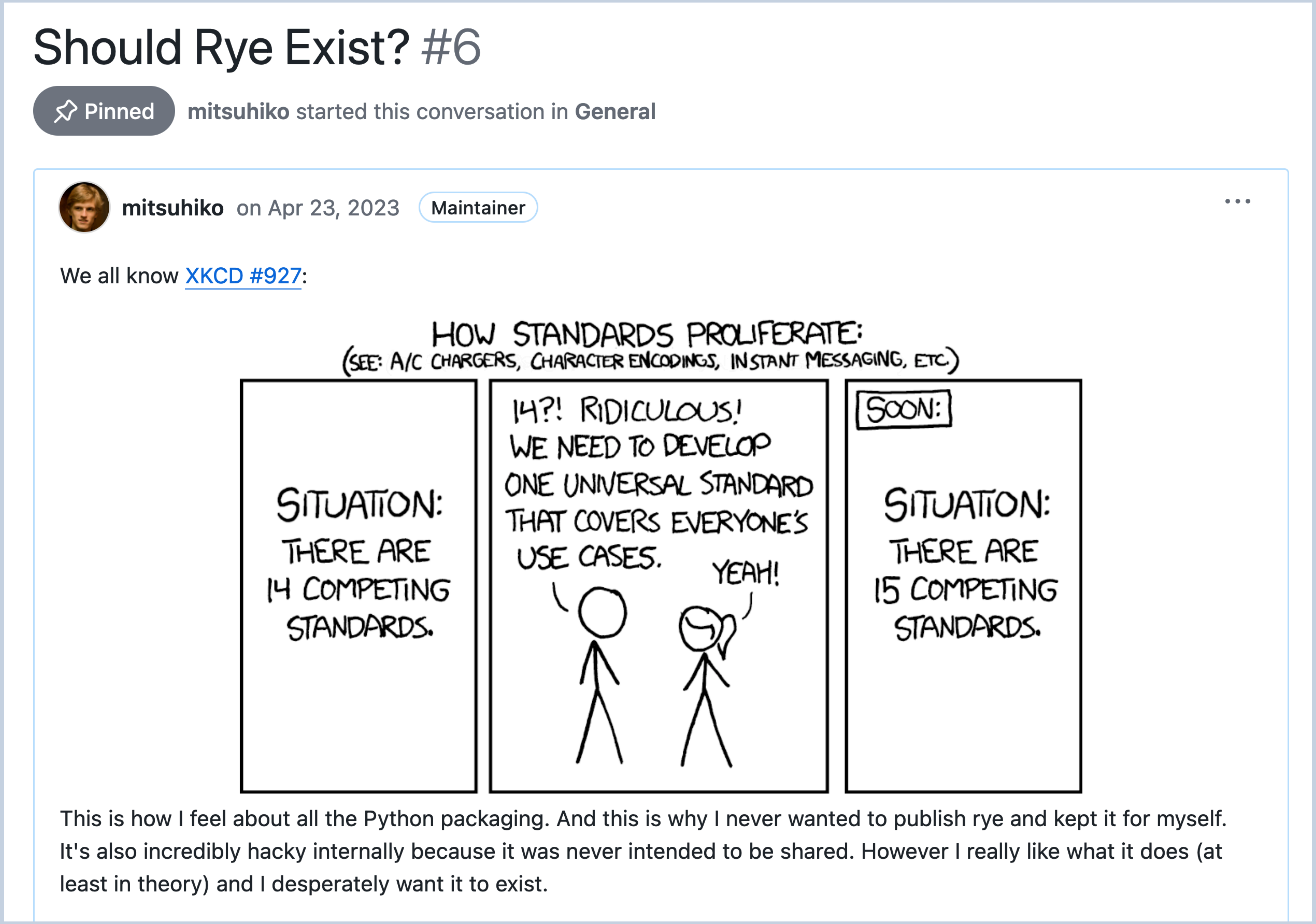
Packaging definitely isn't a special case

I Saw The Light

- Packaging doesn't have to be painful
- Downloading the compiler/interpreter doesn't have to be painful
- Switching between compiler/interpreter versions can be trivial
- One can have the same experience on Linux, macOS, and Windows

Meet Rye!


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

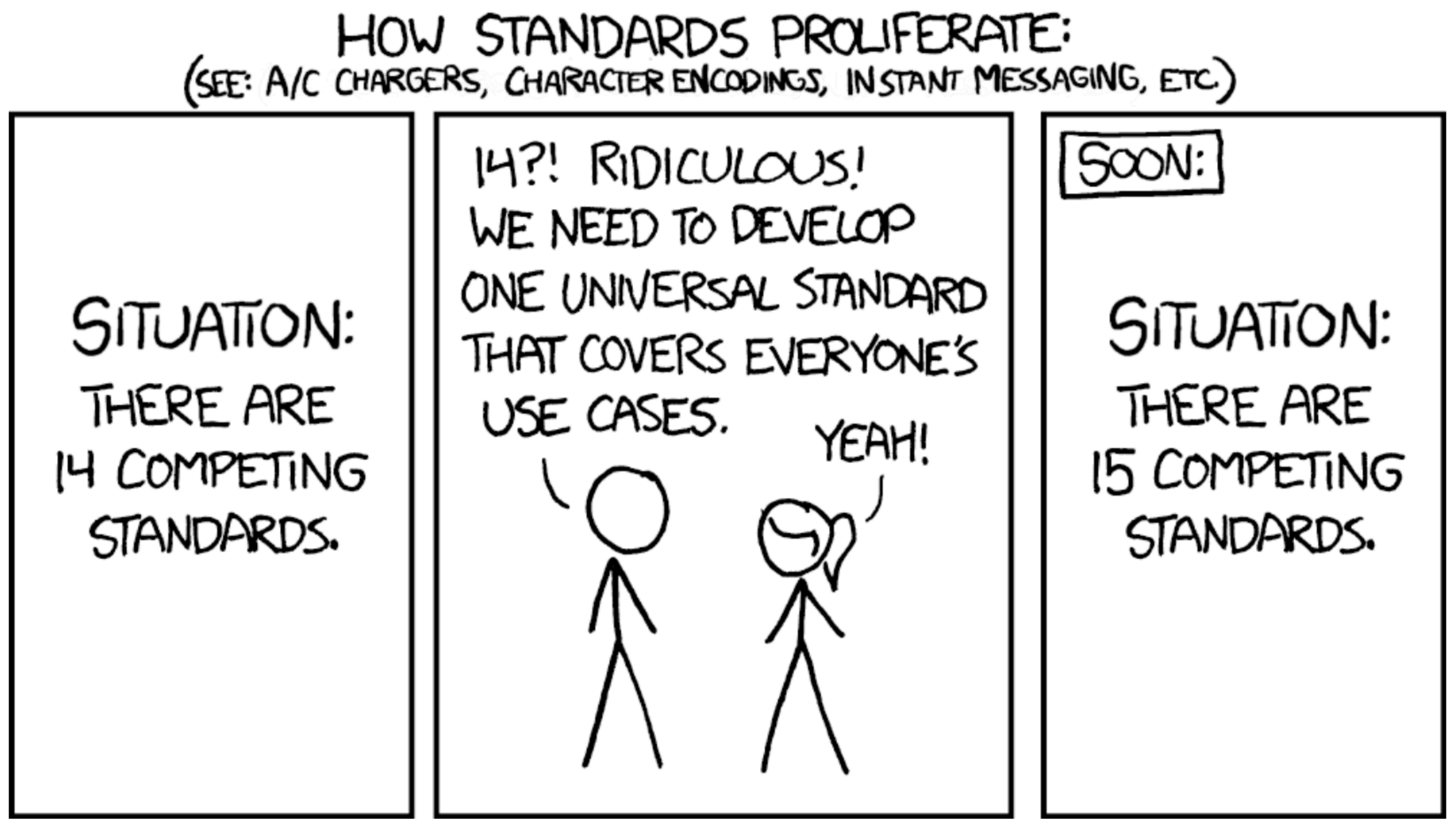


Should Rye Exist? #6

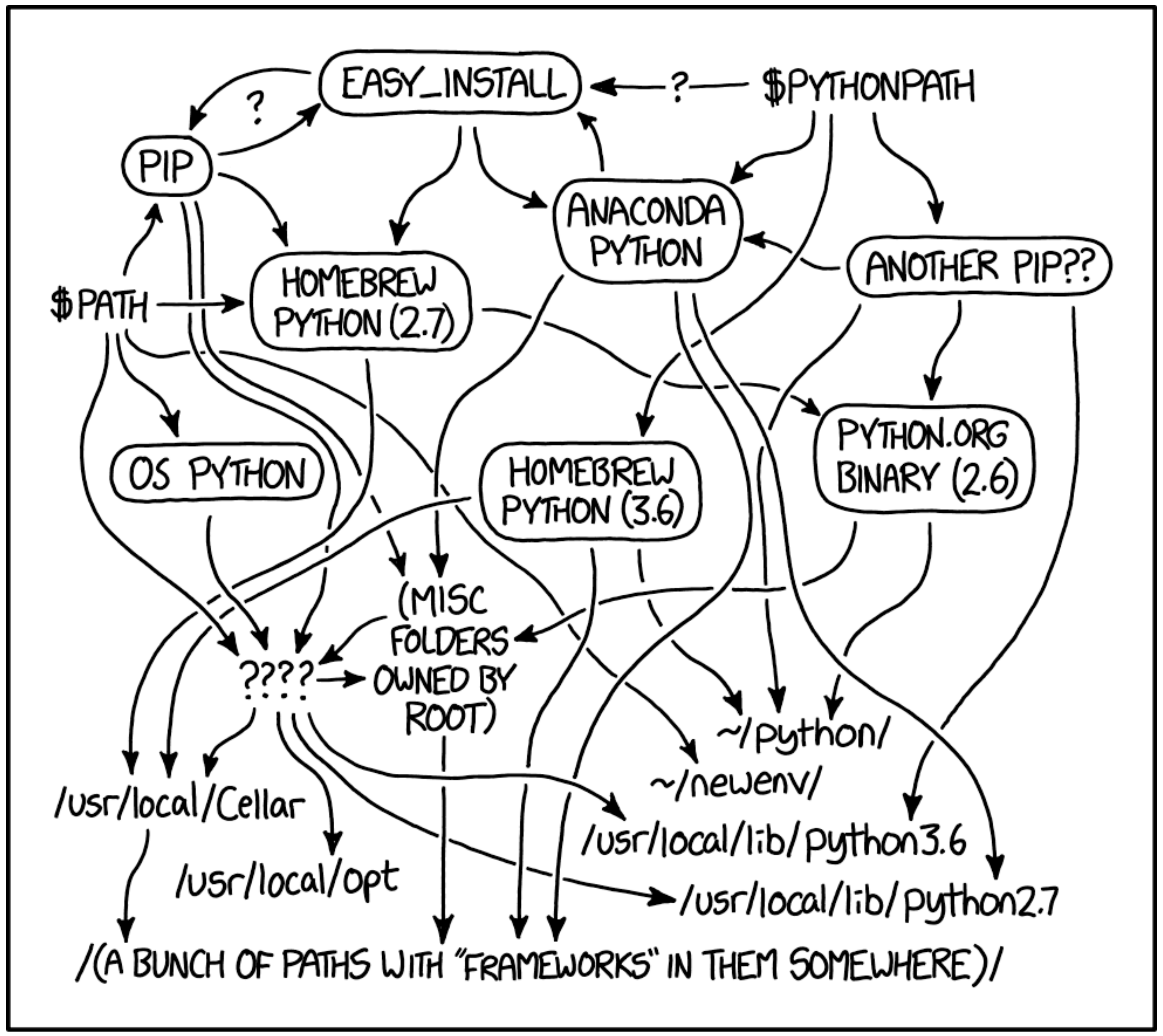
Pinned mitsuhiko started this conversation in General

mitsuhiko on Apr 23, 2023 Maintainer

We all know XKCD #927:


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

This is how I feel about all the Python packaging. And this is why I never wanted to publish rye and kept it for myself. It's also incredibly hacky internally because it was never intended to be shared. However I really like what it does (at least in theory) and I desperately want it to exist.



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

- The only goal is to dominate
- If it does not dominate, something else should
- *“I just want it solved”*

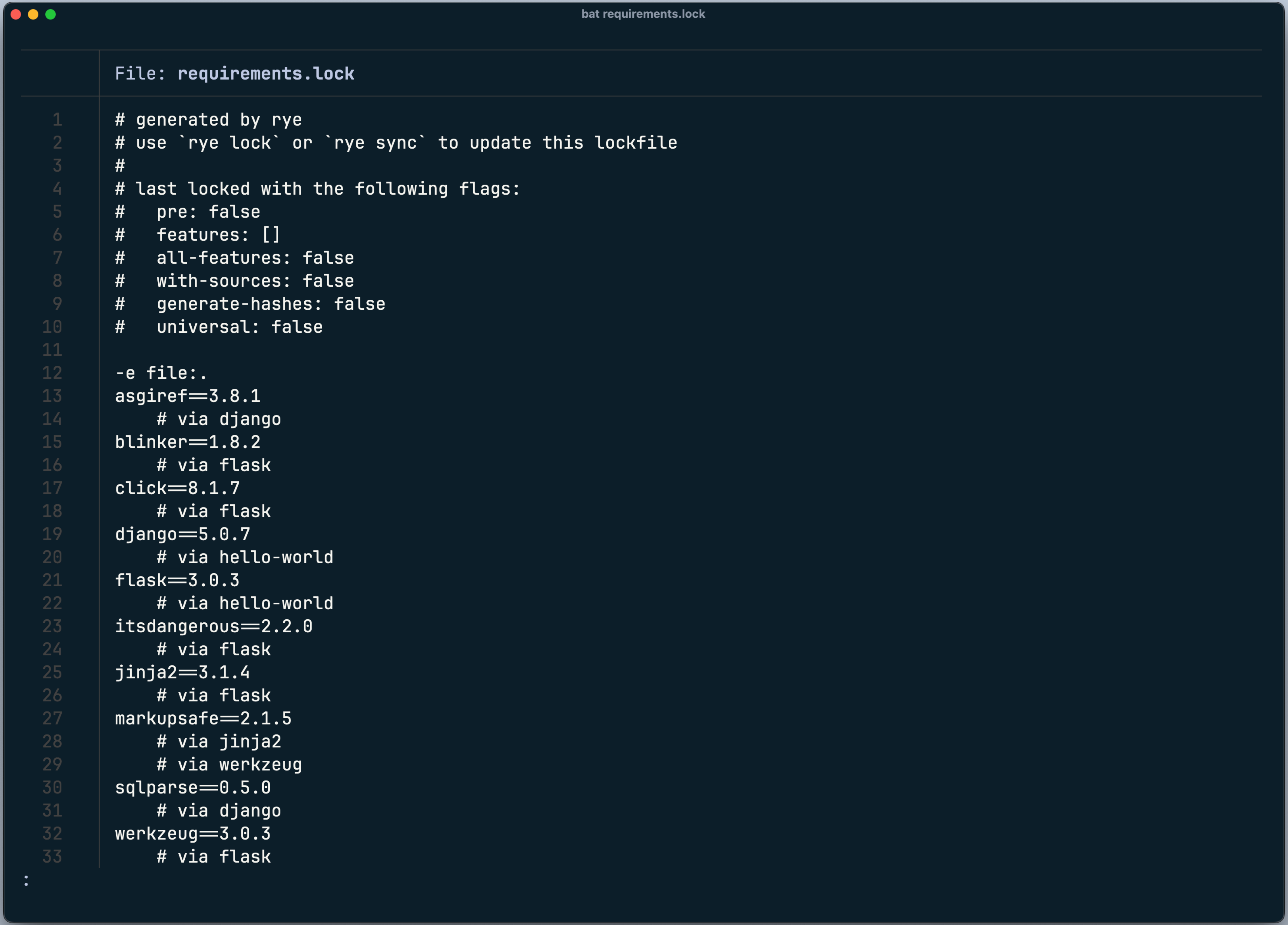
0 to 100

```
mitsuhiko at cheetah in ~
$ 
```

Getting Pythons

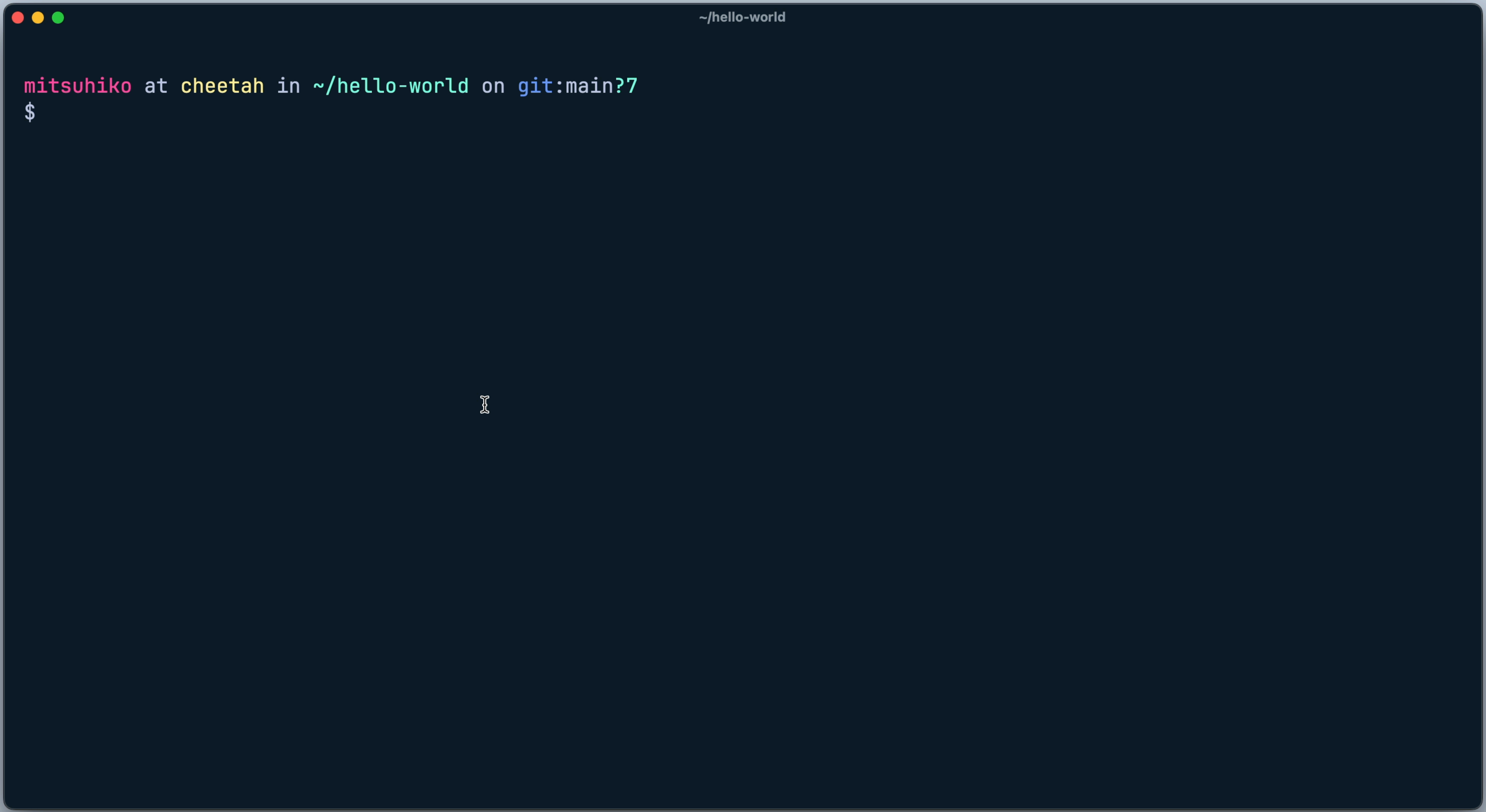
```
mitsuhiko at cheetah in ~
$
```

Lockfiles


bat requirements.lock

```
File: requirements.lock

# generated by rye
# use `rye lock` or `rye sync` to update this lockfile
#
# last locked with the following flags:
#   pre: false
#   features: []
#   all-features: false
#   with-sources: false
#   generate-hashes: false
#   universal: false

-e file:.
asgiref==3.8.1
    # via django
blinker==1.8.2
    # via flask
click==8.1.7
    # via flask
django==5.0.7
    # via hello-world
flask==3.0.3
    # via hello-world
itsdangerous==2.2.0
    # via flask
jinja2==3.1.4
    # via flask
markupsafe==2.1.5
    # via jinja2
    # via werkzeug
sqlparse==0.5.0
    # via django
werkzeug==3.0.3
    # via flask
:
```

Virtual Env Management



~/hello-world

```
mitsuhiko at cheetah in ~/hello-world on git:main?7
$
```

- I do not work for Astral
- I gave Rye's stewardship to Astral
- uv — today — is a replacement for pip-tools/pip/venv
- *uv tomorrow will fully replace the need of Rye by absorbing it in spirit*

Does it work?

- Yes, but there are issues
- Many of the issues are not technical challenges

So what the the challenges?

- Dev Dependencies
- Local Dependencies
- Workspaces
- pyproject.toml Limitations (PEP 508)
- Single Version Resolution
- Good Python Builds

Resolver is Solved

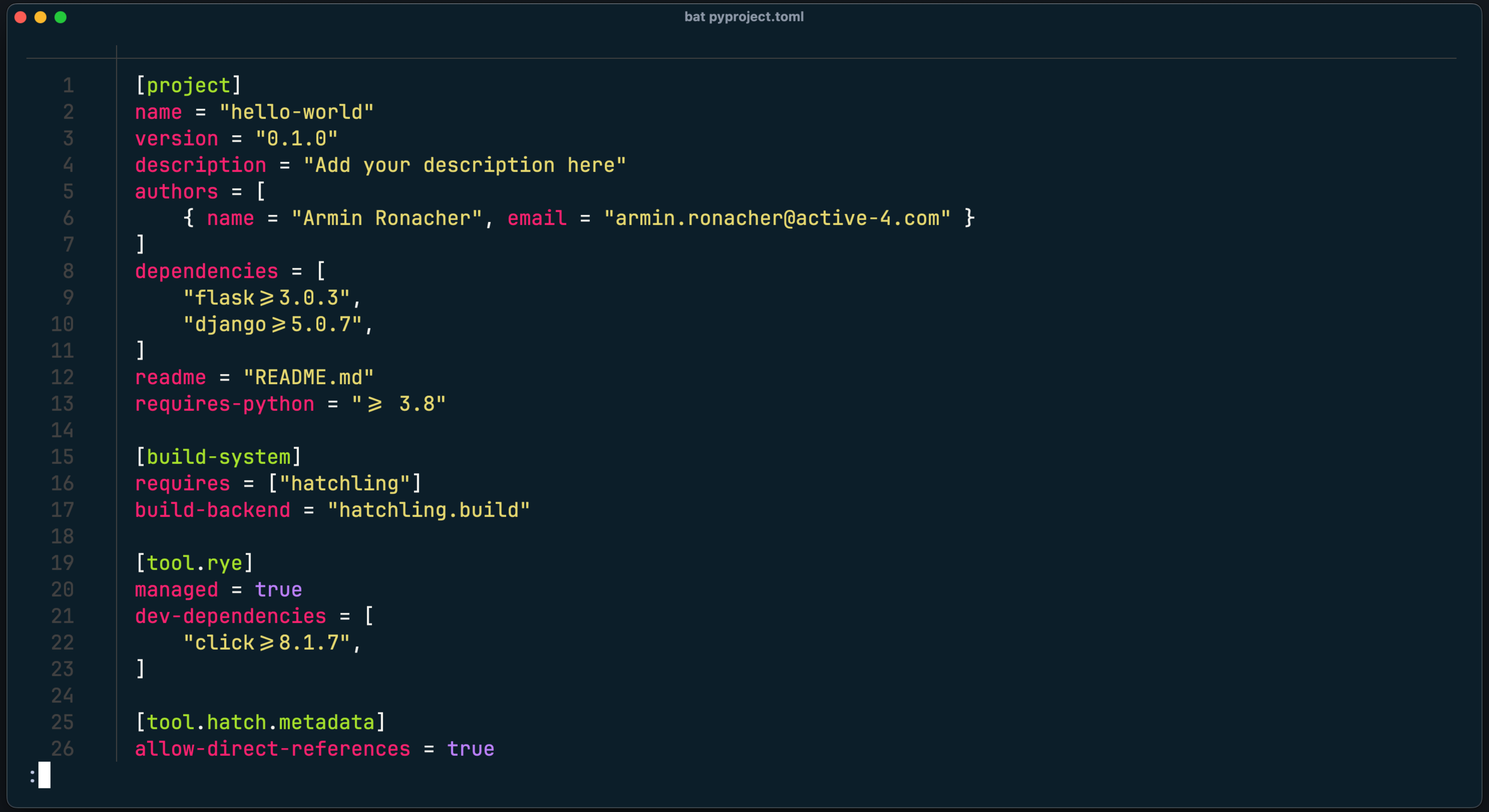
uv is pretty damn fast. You should use it.

Dev Dependencies

- Every Tool invents dev dependencies
- Some could benefit from isolation
  - black, ruff, …
- Others do not work with isolation
  - pytest, …
- Others are mixed
  - flake8, …

Dev Dependencies

- There is no standard, everyone invents one
- Potential solution:
  - reserve a "dev" extra group
  - add a "tool" dependency group?



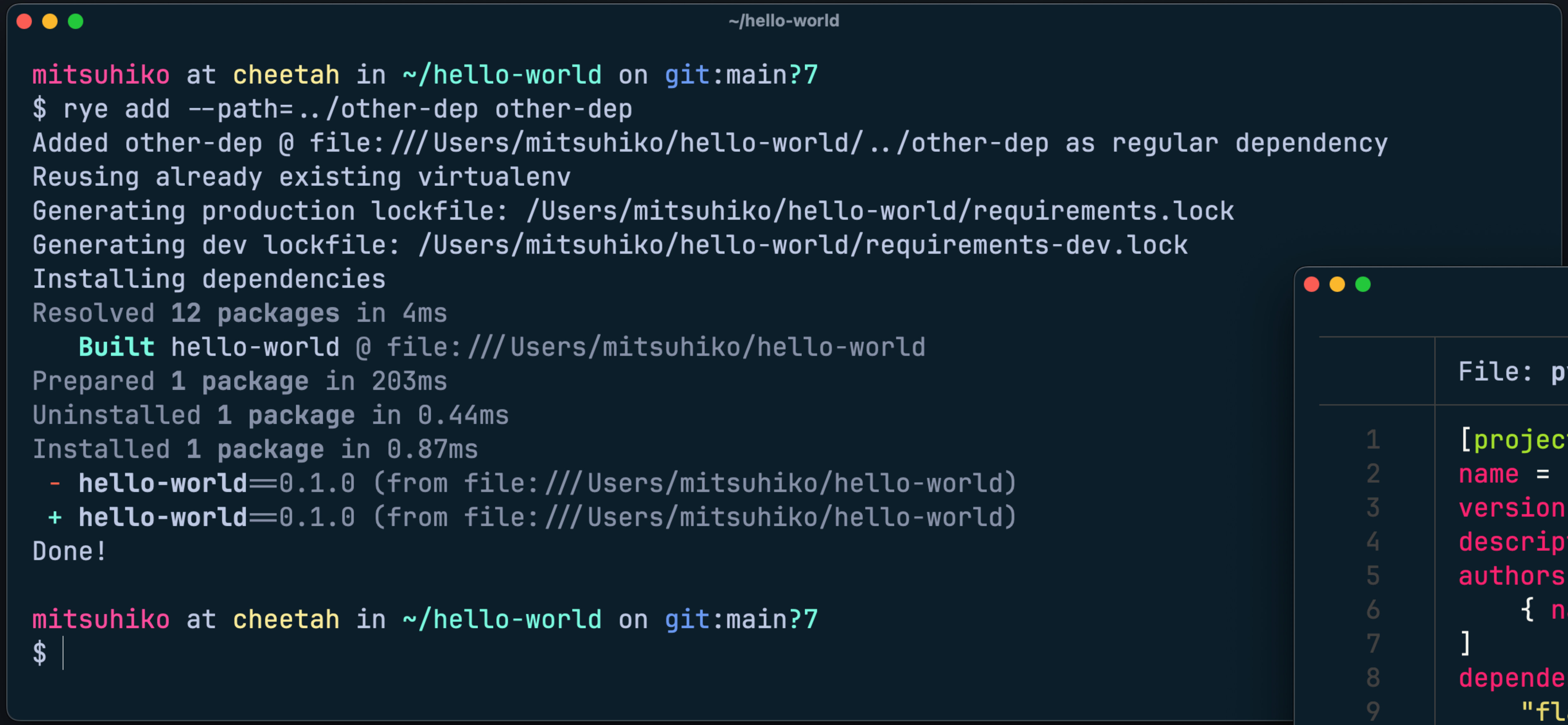
bat pyproject.toml

```toml
[project]
name = "hello-world"
version = "0.1.0"
description = "Add your description here"
authors = [
    { name = "Armin Ronacher", email = "armin.ronacher@active-4.com" }
]
dependencies = [
    "flask≥3.0.3",
    "django≥5.0.7",
]
readme = "README.md"
requires-python = "≥ 3.8"

[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"

[tool.rye]
managed = true
dev-dependencies = [
    "click≥8.1.7",
]

[tool.hatch.metadata]
allow-direct-references = true
```

:

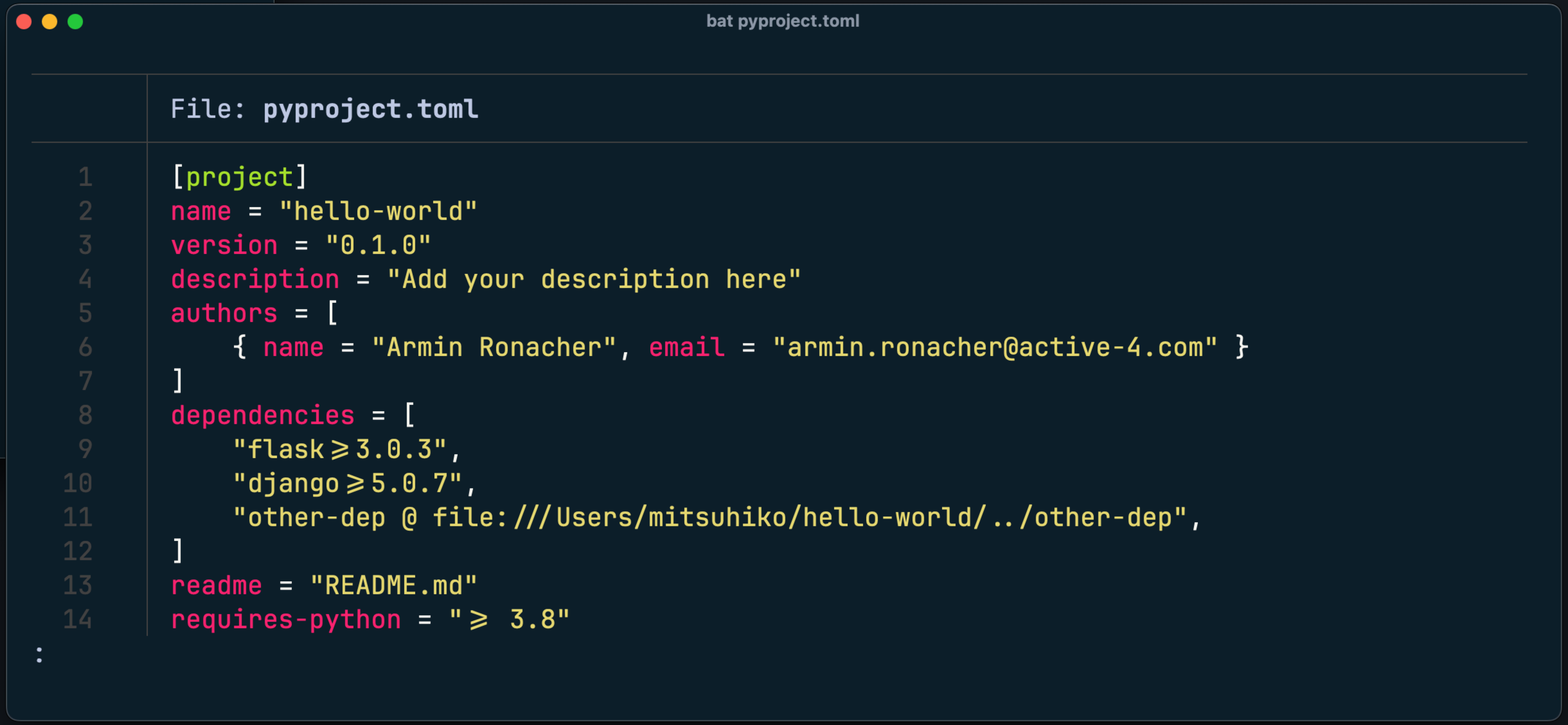
Local Dependencies

- How do you depend on a local package?



~/hello-world

```
mitsuhiko at cheetah in ~/hello-world on git:main?7
$ rye add --path=../other-dep other-dep
Added other-dep @ file:///Users/mitsuhiko/hello-world/../other-dep as regular dependency
Reusing already existing virtualenv
Generating production lockfile: /Users/mitsuhiko/hello-world/requirements.lock
Generating dev lockfile: /Users/mitsuhiko/hello-world/requirements-dev.lock
Installing dependencies
Resolved 12 packages in 4ms
   Built hello-world @ file:///Users/mitsuhiko/hello-world
Prepared 1 package in 203ms
Uninstalled 1 package in 0.44ms
Installed 1 package in 0.87ms
 - hello-world==0.1.0 (from file:///Users/mitsuhiko/hello-world)
 + hello-world==0.1.0 (from file:///Users/mitsuhiko/hello-world)
Done!

mitsuhiko at cheetah in ~/hello-world on git:main?7
$ 
```

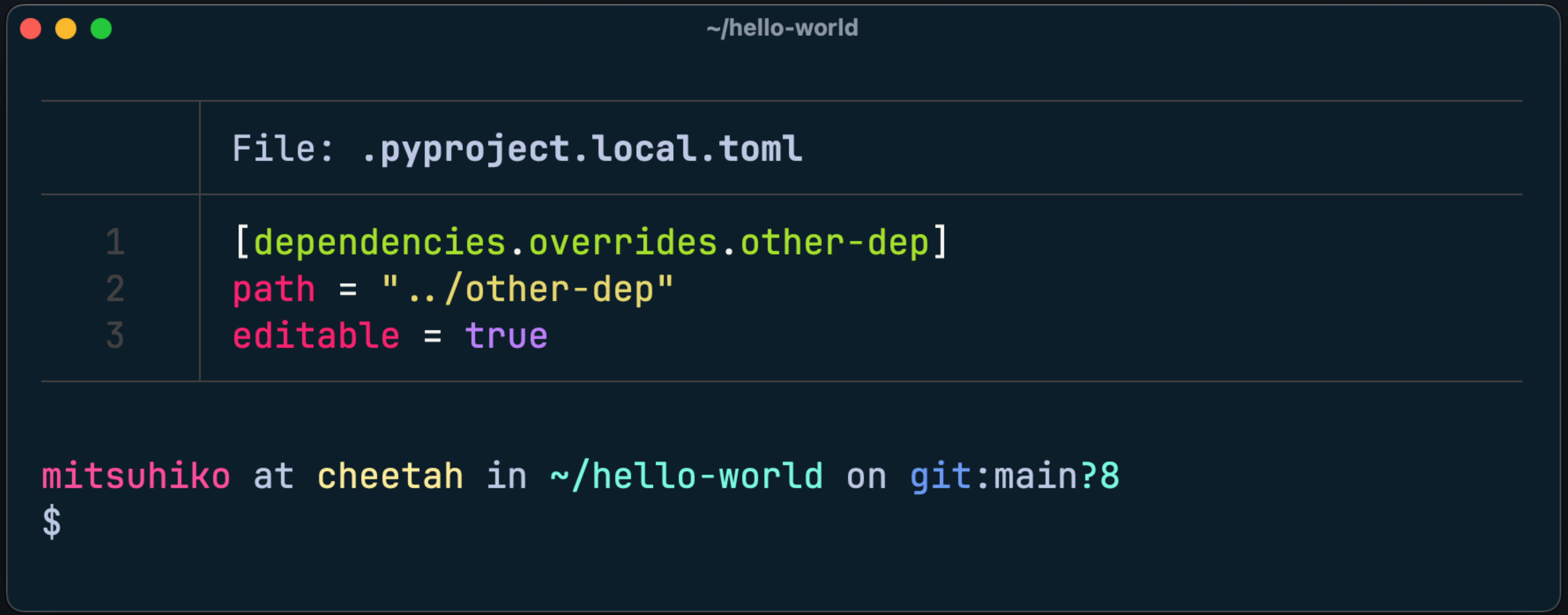


bat pyproject.toml

```
File: pyproject.toml

[project]
name = "hello-world"
version = "0.1.0"
description = "Add your description here"
authors = [
    { name = "Armin Ronacher", email = "armin.ronacher@active-4.com" }
]
dependencies = [
    "flask>=3.0.3",
    "django>=5.0.7",
    "other-dep @ file:///Users/mitsuhiko/hello-world/../other-dep",
]
readme = "README.md"
requires-python = ">= 3.8"
:
```

Local Dependencies

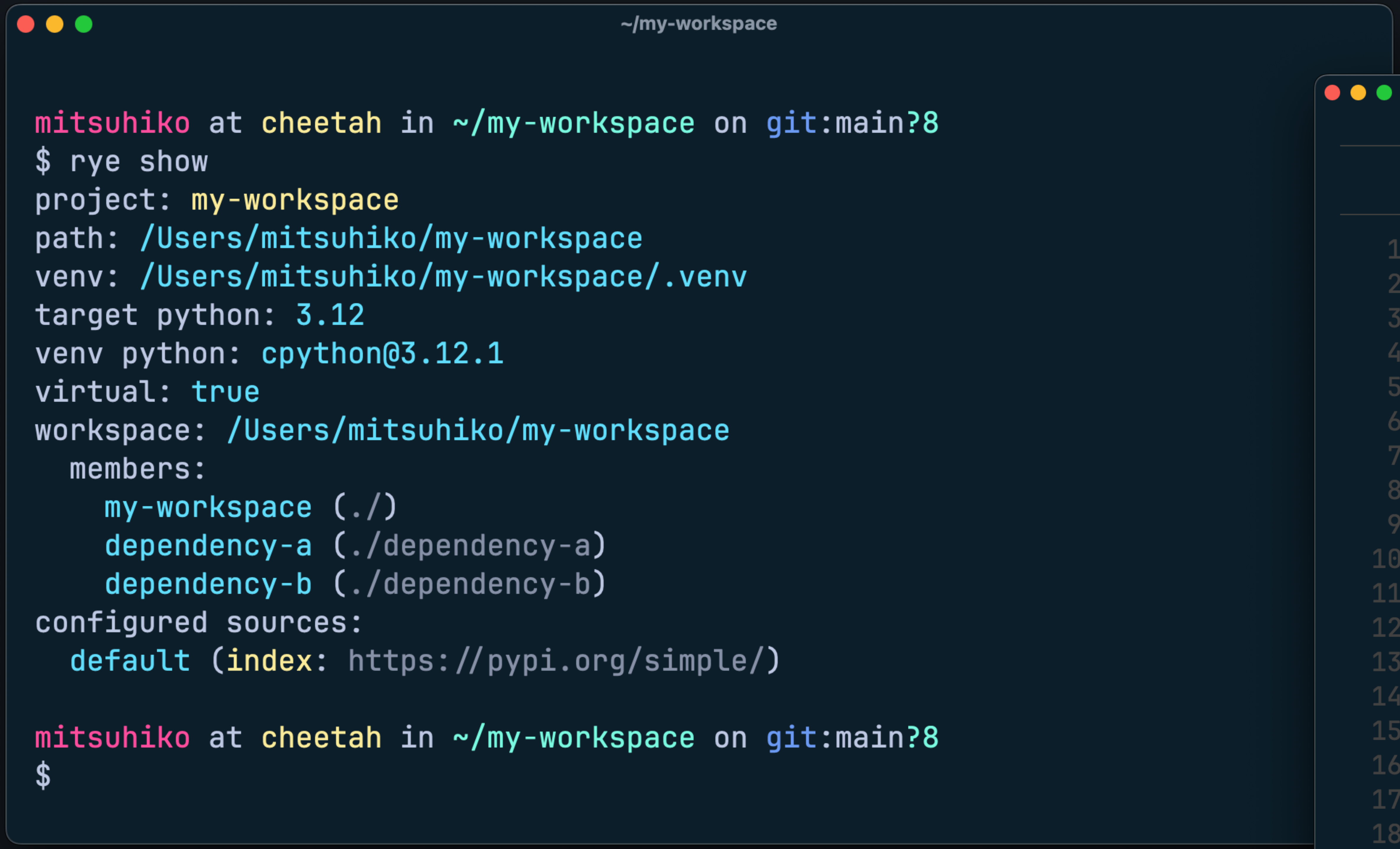
- What about temporary overrides?
- What about editable installs?
- No standard relative path URL syntax
- Potential solution: adjacent config to override packages



```
File: .pyproject.local.toml

[dependencies.overrides.other-dep]
path = "../other-dep"
editable = true

mitsuhiko at cheetah in ~/hello-world on git:main?8
$
```

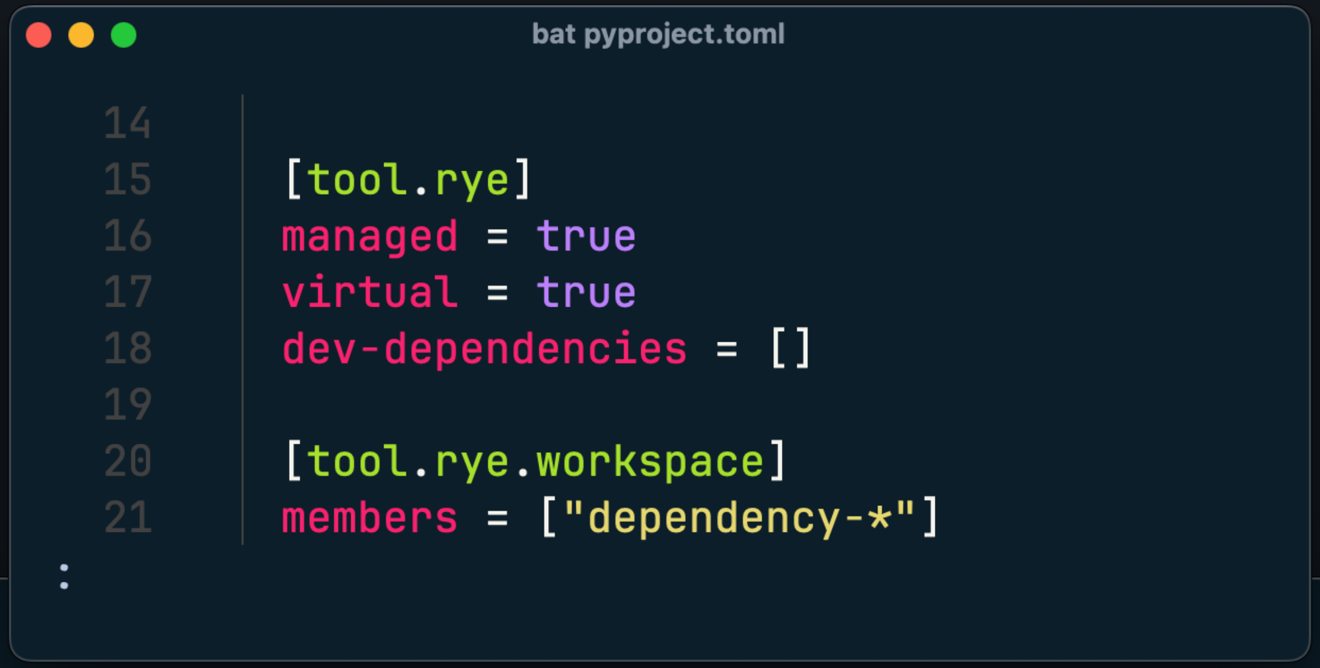
Workspaces

- Multi-dependency projects are important



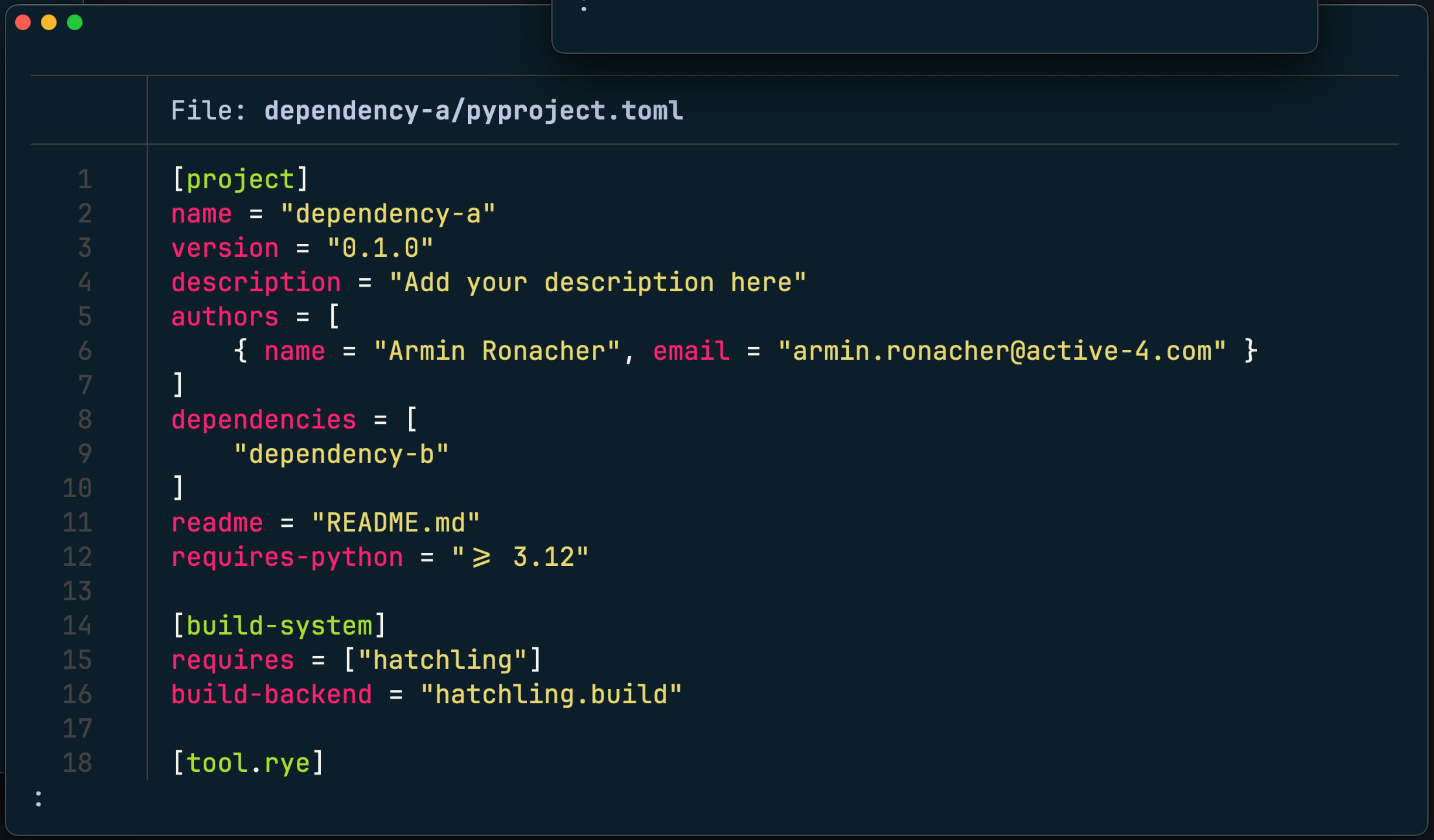
~/my-workspace

```
mitsuhiko at cheetah in ~/my-workspace on git:main?8
$ rye show
project: my-workspace
path: /Users/mitsuhiko/my-workspace
venv: /Users/mitsuhiko/my-workspace/.venv
target python: 3.12
venv python: cpython@3.12.1
virtual: true
workspace: /Users/mitsuhiko/my-workspace
  members:
    my-workspace (./)
    dependency-a (./dependency-a)
    dependency-b (./dependency-b)
configured sources:
  default (index: https://pypi.org/simple/)

mitsuhiko at cheetah in ~/my-workspace on git:main?8
$
```



bat pyproject.toml

```

[tool.rye]
managed = true
virtual = true
dev-dependencies = []

[tool.rye.workspace]
members = ["dependency-*"]
:
```



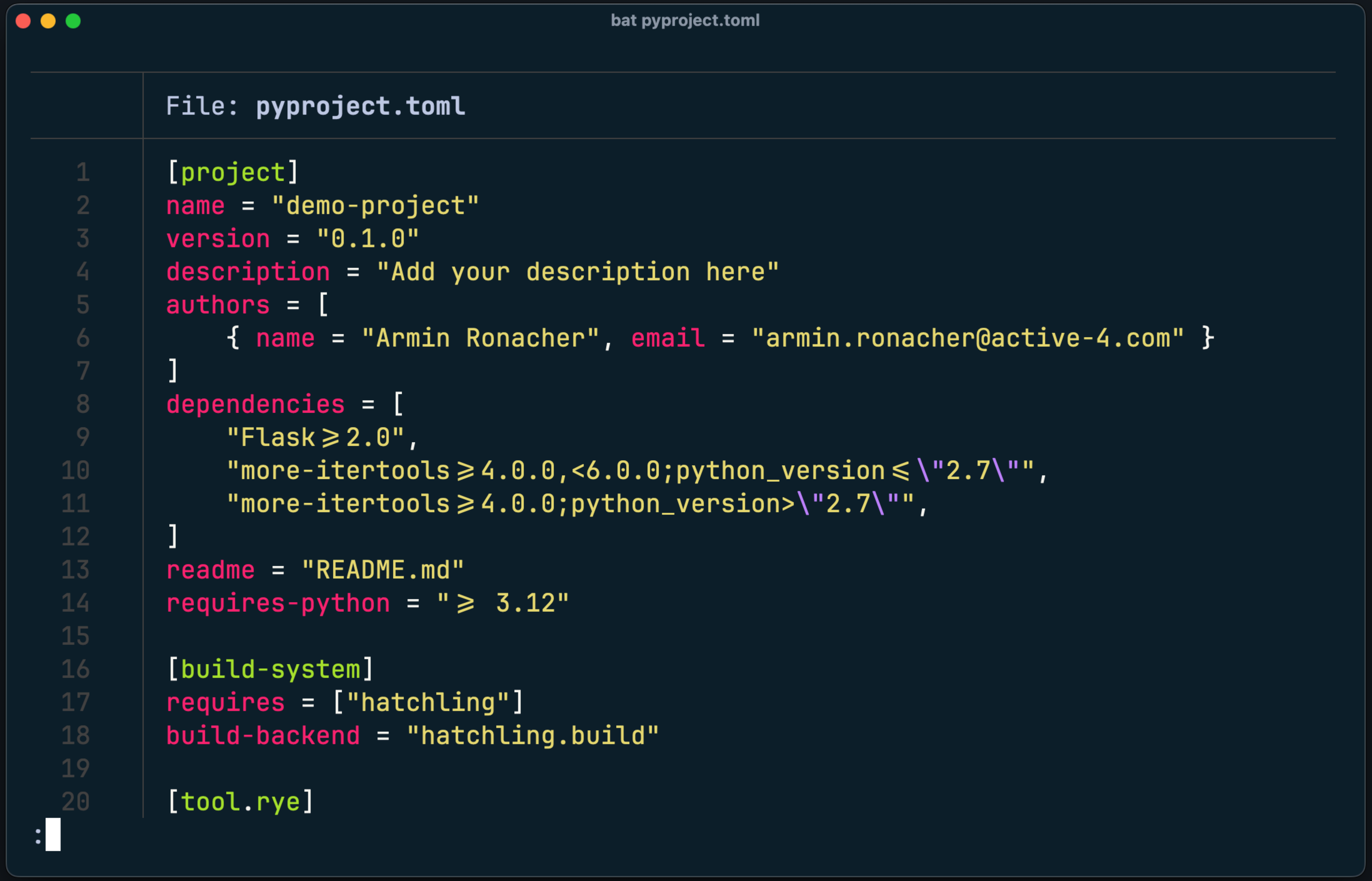
```
File: dependency-a/pyproject.toml

[project]
name = "dependency-a"
version = "0.1.0"
description = "Add your description here"
authors = [
    { name = "Armin Ronacher", email = "armin.ronacher@active-4.com" }
]
dependencies = [
    "dependency-b"
]
readme = "README.md"
requires-python = ">= 3.12"

[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"

[tool.rye]
:
```

Workspaces

- But they don't work well yet
- They are Rye proprietary
- Again run into challenges with relative paths

pyproject.toml Limitations

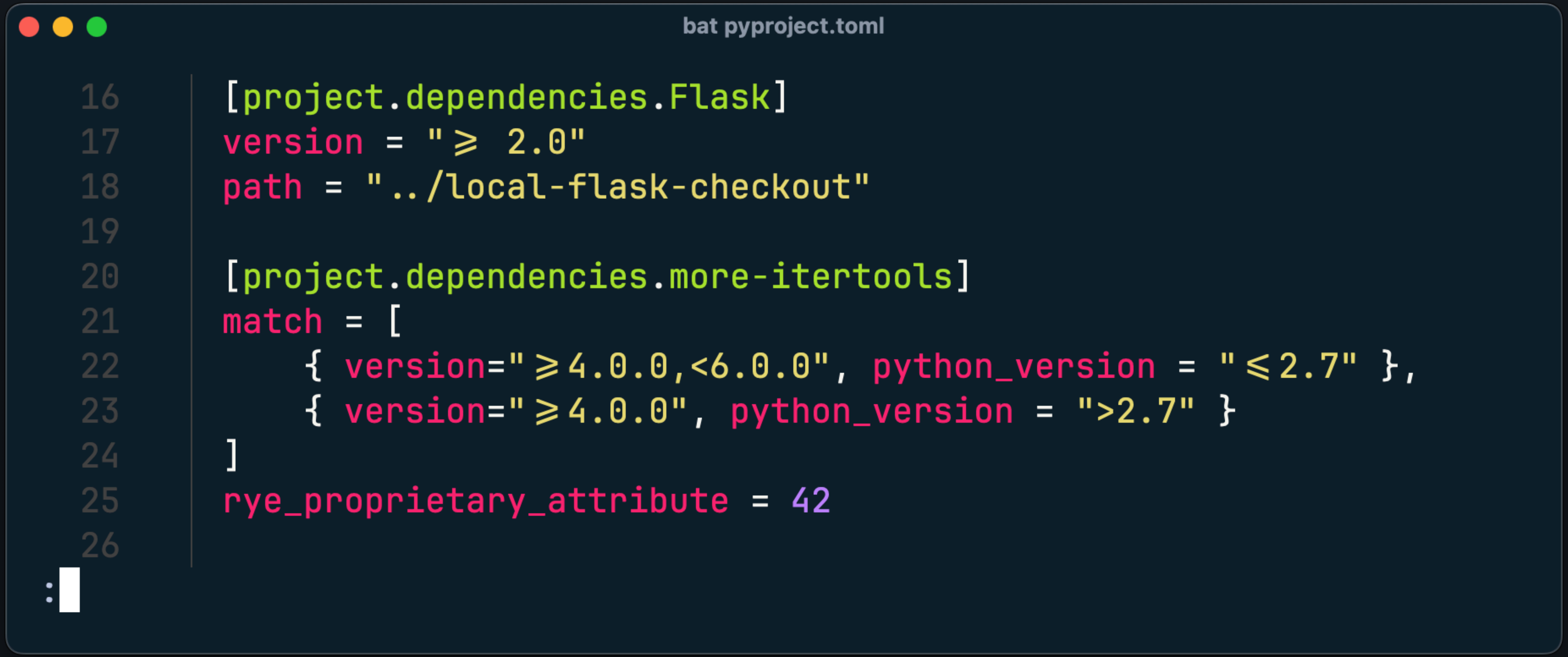
- Dependency string array is too limiting
    - Where do you store dependency attached meta information?
    - Impossible to encode even more into these strings without breaking already existing tools
    - Who can write these strings?



```
File: pyproject.toml

[project]
name = "demo-project"
version = "0.1.0"
description = "Add your description here"
authors = [
    { name = "Armin Ronacher", email = "armin.ronacher@active-4.com" }
]
dependencies = [
    "Flask≥2.0",
    "more-itertools≥4.0.0,<6.0.0;python_version≤\"2.7\"",
    "more-itertools≥4.0.0;python_version>\"2.7\"",
]
readme = "README.md"
requires-python = "≥ 3.12"

[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"

[tool.rye]
:
```

pyproject.toml Limitations

- Why do you need meta information?
- Pick the right index (PyPI vs internal)
- Git checkout, local paths, multi-version matches
- Tool specific proprietary (even if only temporary) extra information



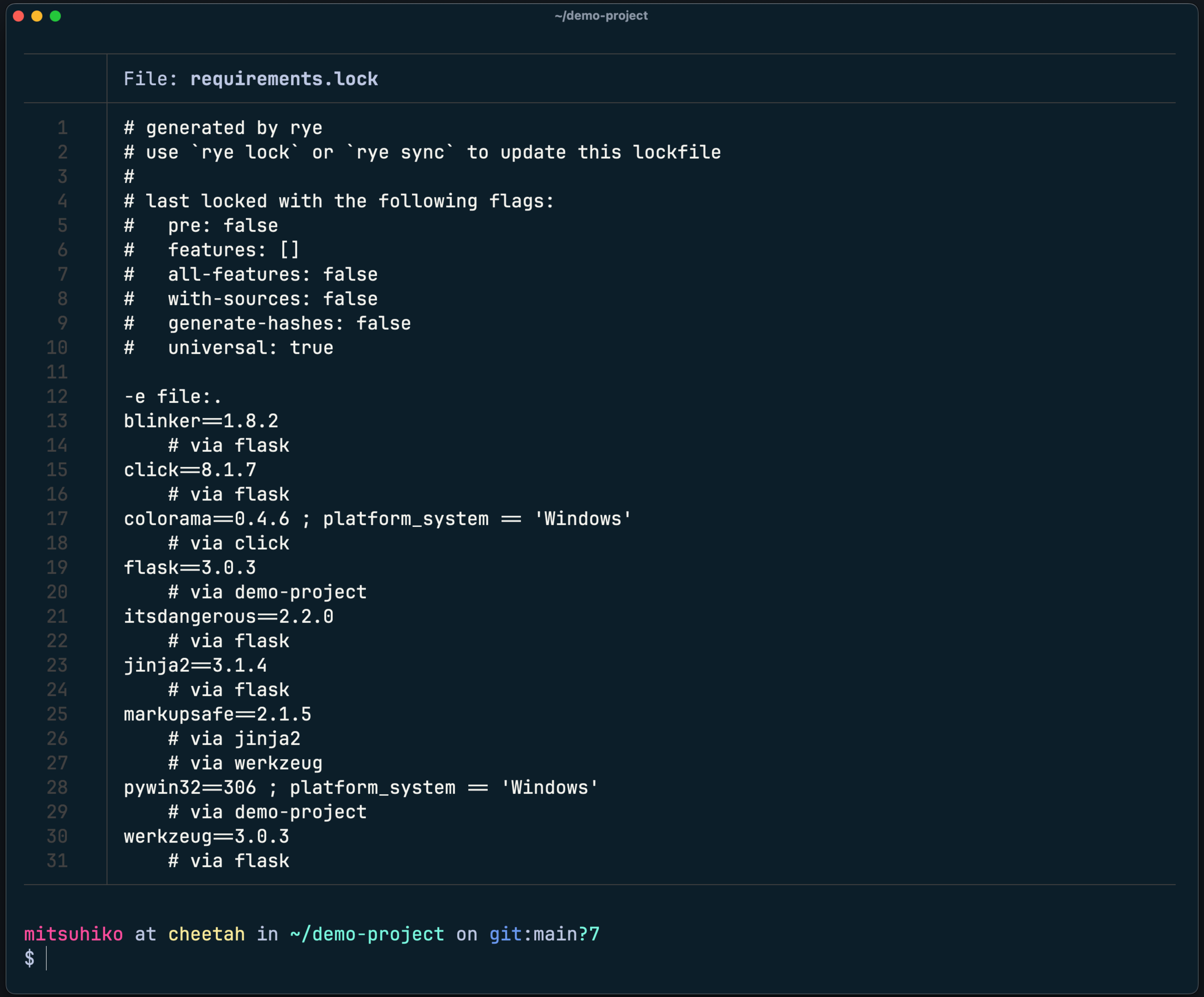
```
[project.dependencies.Flask]
version = "≥ 2.0"
path = "../local-flask-checkout"

[project.dependencies.more-itertools]
match = [
    { version="≥4.0.0,<6.0.0", python_version = "≤2.7" },
    { version="≥4.0.0", python_version = ">2.7" }
]
rye_proprietary_attribute = 42

:
```

Other Issues with pyproject.toml

- Dynamic metadata is a bad idea
- Already countless of proprietary extensions by different tools
- Many different ways to define licenses
- Complex resolutions caused by markers

Portable Locking

- rye/uv support experimental universal locking
- it does not yet have a stable and supported cross platform lock format
- The problem is "not easy"



~/demo-project

```
File: requirements.lock

# generated by rye
# use `rye lock` or `rye sync` to update this lockfile
#
# last locked with the following flags:
#   pre: false
#   features: []
#   all-features: false
#   with-sources: false
#   generate-hashes: false
#   universal: true

-e file:.
blinker==1.8.2
    # via flask
click==8.1.7
    # via flask
colorama==0.4.6 ; platform_system == 'Windows'
    # via click
flask==3.0.3
    # via demo-project
itsdangerous==2.2.0
    # via flask
jinja2==3.1.4
    # via flask
markupsafe==2.1.5
    # via jinja2
    # via werkzeug
pywin32==306 ; platform_system == 'Windows'
    # via demo-project
werkzeug==3.0.3
    # via flask

mitsuhiko at cheetah in ~/demo-project on git:main?7
$
```

Single Version Resolution

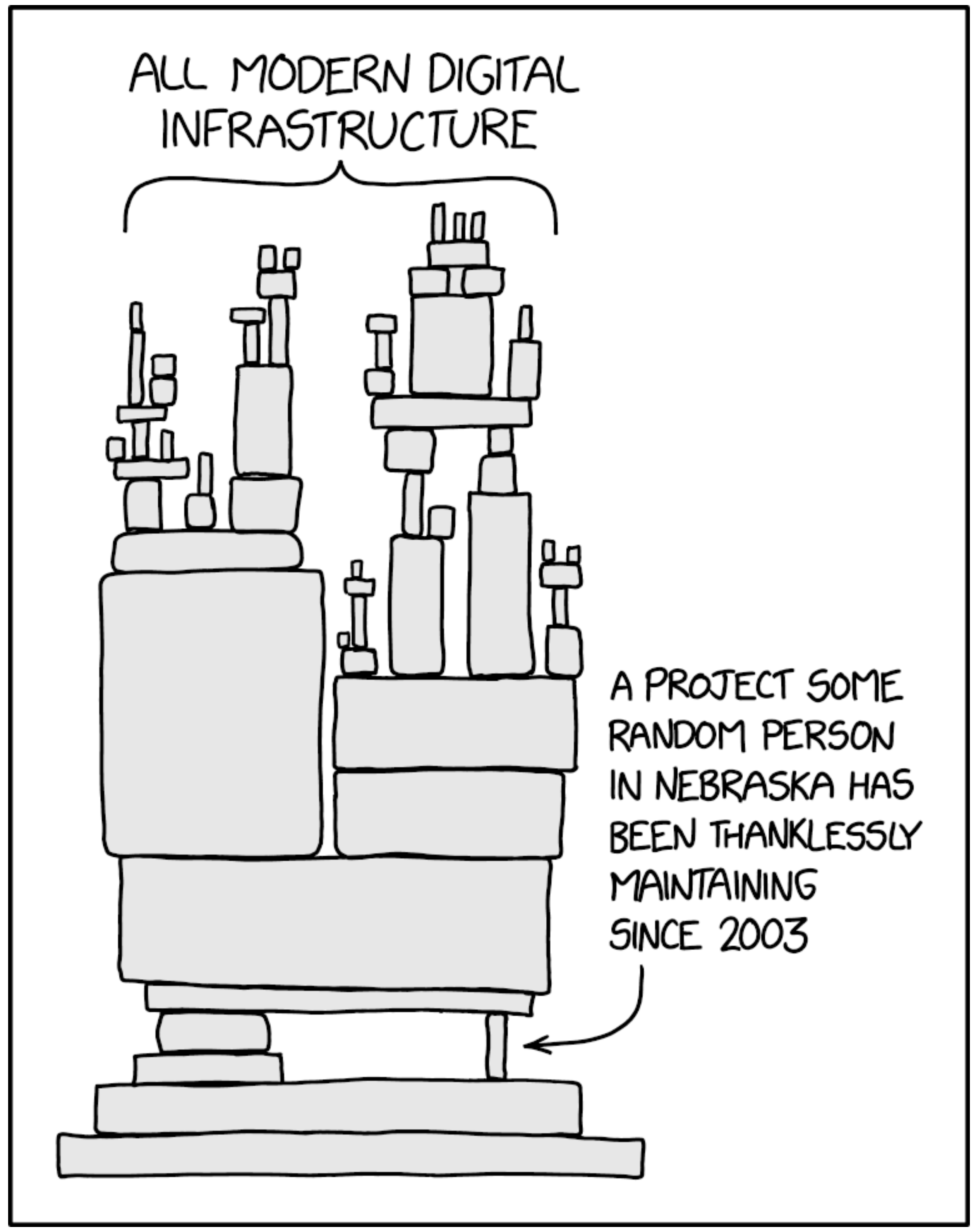
- a >= 1
- a < 1
- How can you ever find a solution?
- Rust/Node: permits multi-version resolutions
- Issues: sys.modules (though solvable), C-extension modules (CABI)

Technically Solvable

- https://github.com/mitsuhiko/multiversion/
- Demonstration of multi-version imports on Python 2
- What would be the benefit? Smoother ecosystem upgrades

Good Python Builds

- We need PEP 711: PyBI: a standard format for distributing Python Binaries
- indygreg builds are great, but have portability issues (bad CFLAGS, missing readline, ...)
- Not an official project, run by a single person


ALL MODERN DIGITAL INFRASTRUCTURE
A PROJECT SOME RANDOM PERSON IN NEBRASKA HAS BEEN THANKLESSLY MAINTAINING SINCE 2003

We are so close to solving it

*What can stand in the way
is only ourselves*

Beware: “I got this”

fin.

- https://x.com/mitsuhiko
- https://rye.astral.sh/
- https://github.com/astral-sh/uv