29 Jun 2025

Tools I love: mise(-en-place)

tools cli mise

Once in a while you get introduced to a tool that instantly changes the way you work. For me, mise is one of those tools.

mise is the logical conclusion to a lot of the meta-tooling that exists around language-specific version and package managers like asdf, nvm, uv, pyenv etc. It makes it exceptionally easy to install, use, and manage software. It also allows you to manage environment variables and declare tasks (run commands).

Trying out new tools

The first step in getting an intuitive understanding of what mise can help you with is to use it to install a tool. Pick your favorite and try it out; it supports *a lot*!

I recently read about `jj` in Thorsten Ball's newsletter and decided to try it out (again). I crossed my fingers and hoped that `jj` was one of the tools supported by mise and, to my delight, it was! The process looked something like this:

```
$ jj
command_not_found_handler:5: command not found: jj

$ mise use jj
mise ~/projects/examples_mise/mise.toml tools: jj@0.30.0

$ jj version
jj 0.30.0

$ cd ..

$ jj version
command_not_found_handler:5: command not found: jj

$ cd eaxmples_mise
$ jj version
jj 0.30.0
```

As the above shows, with mise we're just one command away from installing and trying out a new tool, e.g. `mise use jj`.

In the above we that mise printed `mise ~/projects/examples_mise/mise.toml tools: jj@0.30.0`. This tells us that mise has created (or updated) the mise configuration *on that path*. We also see that if we cd out of `~/projects`, the `jj` command is no longer available. If we cd back into `~/projects/examples_mise`, it becomes available again; unless you explicitly install tools globally, mise will only make the tools available which are mentioned in a `mise.toml` file on the path from your current directory to the root of your file system. That of course means that we could potentially meet multiple `mise.toml` files when going back up to the root of the file system. Mise handles this by concatting the configurations and overwriting conflicting configurations, letting the file furthest down the tree win.

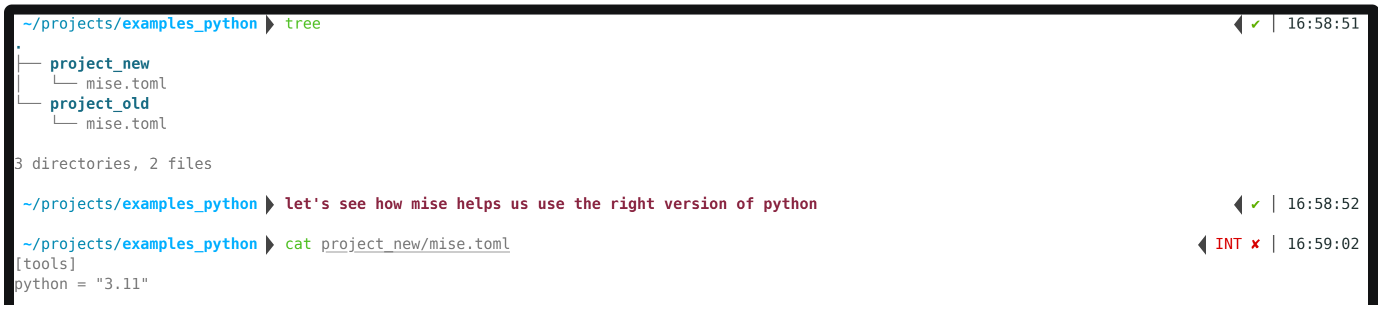
This is a clever design as it allows us to configure different versions of the same tool to be available in different directories. Let's have a look at what the `mise.toml` file looks like:

```
[tools]
jj = "latest"
```

If we want a specific version of `jj` to be installed in a specific directory, we just update the toml file to say e.g. `jj = "0.30.0`.

Managing multiple versions of a tool

Let's see what it looks like to use mise to manage Python versions for two projects with different requirements:



```
 ~/projects/examples_python ▶ tree                                         ◀ ✔ │ 16:58:51
.
├── project_new
│   └── mise.toml
└── project_old
    └── mise.toml

3 directories, 2 files

 ~/projects/examples_python ▶ let's see how mise helps us use the right version of python   ◀ ✔ │ 16:58:52

 ~/projects/examples_python ▶ cat project_new/mise.toml                    ◀ INT ✘ │ 16:59:02
[tools]
python = "3.11"
```

```
$ tree
.
```

```
├── project_new
│       └── mise.toml
└── project_old
    └── mise.toml

$ cat project_new/mise.toml
[tools]
python = "3.11"

$ cat project_old/mise.toml
[tools]
python = "3.8"

$ cd project_new
$ python --version
Python 3.11.13

$ cd ../project_old
$ python --version
Python 3.8.20
```

When we cd into one of the directories listed above, mise automatically makes the version of the tool configured in `mise.toml` available to us. If it isn't already installed, mise will install it for us. The implication of this is that you can commit a `mise.toml` to your repository, and anyone that has mise installed will automatically get and use the expected dev tools when they enter the project directory. And when it's time to upgrade a dev tool, you can just update the version number in `mise.toml` and everyone will start using the new version!

Use in CI/CD pipelines

The fact that mise makes tools available to you according to the `mise.toml` file in your current working directory has further implications: it's not just developer machines that can benefit from using mise; CI/CD pipelines can benefit greatly as well! When you use mise in your pipelines, you avoid the problem of having out of sync versions between developer and build machines. You get to have a single place where you can configure the version of your dev tools everywhere!

As I mentioned in the beginning, besides managing dev tools, mise also allows you to declare and run so-called tasks. Think of a task as an advanced invocation of a bash script. Even if we use tasks as just plain bash scripts (they can do a lot more), it can be a major advantage to declare common operations such as building, testing, linting etc. as mise tasks, since all developers get access to them and will run their commands in exactly the same way every time.

If you're diligent in your naming, you can even make the experience of building or testing across projects identical.

The following are examples of some very simple Python-related tasks declared in `mise.toml`:

```toml
[tasks.install-deps]
run = ["uv pip install -r requirements.txt"]

[tasks.test]
run = ["pytest ."]
```

Adding this to `mise.toml` will make the commands `mise install-deps` and `mise test` available. Again, if you check this in to your repo, the commands will be available to all developers and pipelines. And reusing these names in your rust project means that you can use the same commands to tell cargo to install your crates or run your tests.

Once you've declared your tasks you should of course also use them in your CI/CD pipeline. Doing this makes you less dependent on the particular yaml syntax and arbitrary requirements of your provider, and makes it easier to move to another one if you need to. It also ensures that there's a standard way to build and test your code, helping to further reduce the amount of "it works on my machine".

There's a lot of depth to what you can use mise to help you automate. It's a lovely tool and I hope I've spiked your interest enough to give it a try!

Security concerns

Although this is a very obvious problem, I want to make it explicit: a major concern of all software dependency management is control of your supply chain; how easy is it for somebody to insert malicious code into a binary you will run hugely impacts the integrity of your systems and data. Depending on your industry, it might not be feasible to use mise as it's pretty opaque where your dependencies will be downloaded from.

Michael Bang
@micvbang · Follow

I'm hoping to find the time to write a series of posts over the summer on tools that I love. Here's the first one which I fell in love with just 3 months ago: mise

blog.vbang.dk/2025/06/29/too…

6:02 PM · Jun 29, 2025

4 Reply Copy link

Read 1 reply