# Make your own make

Jan 3, 2018

## Introduction

One of my favorite features of Cargo is that it is **not** a general purpose build tool. This allows Cargo to really excel at the task of building Rust code, without usual Turing tarpit of build configuration files. I have yet to see a complicated Cargo.toml file!

However, once a software project grows, it's almost inevitable that it will require some tasks **besides** building Rust code. For example, you might need to integrate several languages together, or to setup some elaborate testing for non-code aspects of your project, like checking the licenses, or to establish an elaborate release procedure.

For such use-cases, a general purpose task automation solution is needed. In this blog post I want to describe one possible approach, which leans heavily on Cargo's built-in functionality.

xtask specification is a modern version of this idea.

## Existing Solutions

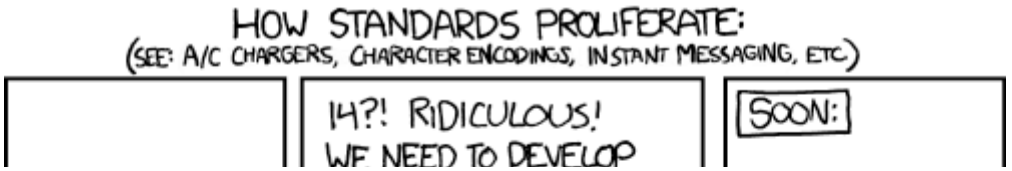
The simplest way to automate something is to write a shell script. However there are few experts in the arcane art of shell scripting, and shell scripts are inherently platform dependent.

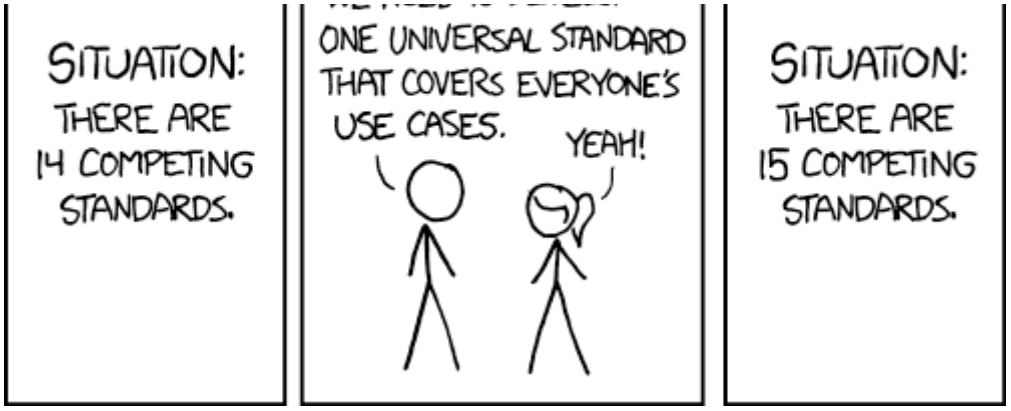
The same goes for make, with its many annoyingly similar flavors.

Two tools which significantly improve on the ease of use and ergonomics are just and cargo make. Alas, they still mostly rely on the shell to actually execute the tasks.

## Reinventing the Wheel

Obligatory XKCD 927:






An obvious idea is to use Rust for task automation. Originally, I have proposed creating a special Cargo subcommand to execute build tasks, implemented as Rust programs, in this thread. However, since then I realized that there are built-in tools in Cargo which allow one to get a pretty ergonomic solution. Namely, the combination of workspaces, aliases and ability to define binaries seems to do the trick.

## Elements of the Solution

If you just want a working example, see this commit.

A typical Rust project looks like this

```
frobnicator/
  Cargo.toml
  src/
    lib.rs
```

Suppose that we want to add a couple of tasks, like generating some code from some specification in the RON format, or grepping the source code for `TODO` marks.

First, create a special `tools` package:

```
frobnicator/
  Cargo.toml
  src/
    lib.rs
  tools/
    Cargo.toml
    src/bin/
      gen.rs
      todo.rs
```

The `tools/Cargo.toml` might look like this:

```toml
# file: frobnicator/tools/Cargo.toml

[package]
name = "tools"
version = "0.1.0"
authors = []
# We never publish our tasks
publish = false

[dependencies]
# These dependencies are isolated from the main crate.
serde = "1.0.26"
serde_derive = "1.0.26"
file = "1.1.1"
ron = "0.1.5"
```

Then, we add a `[workspace]` to the parent package:

```toml
# file: frobnicator/Cargo.toml

[workspace]
members = ["tools"]
```

We need this section because `tools` is not a dependency of `frobnicator`, so it won't be picked up automatically.

Then, we write code to accomplish the tasks in `tools/src/bin/gen.rs` and `tools/src/bin/todo.rs`.

Finally, we add `frobnicator/.cargo/config` with the following contents:

```
# file: frobnicator/.cargo/config

[alias]
gen  = "run --package tools --bin gen"
todo = "run --package tools --bin todo"
```

Voilà! Now, running `cargo gen` or `cargo todo` will execute the tasks!

Discussion on /r/rust.