

Package Management Learnings from Homebrew



Mike McQuaid



Mike McQuaid

<https://mikemcquaid.com>

Why?



brew install
works: no-one cares



**brew install
breaks: we're fucked**



my toilet
works: no-one cares



my toilet
breaks: shit everywhere



Act 1

5.0.0

"Homebrew is slow"



"Bundler is slow"



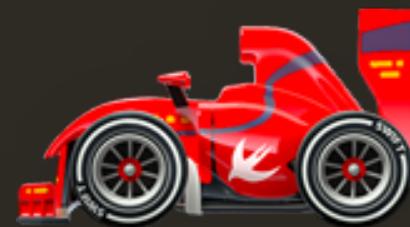
"Cargo is fast"



"uv is fast"



"Bundler is fast"



"\$brew in Rust is fast"

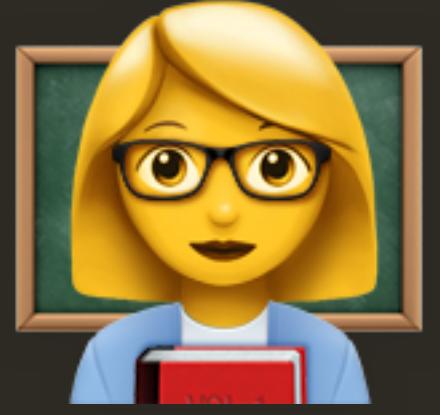




`HOMEBREW_DOWNLOAD_CONCURRENCY=auto`



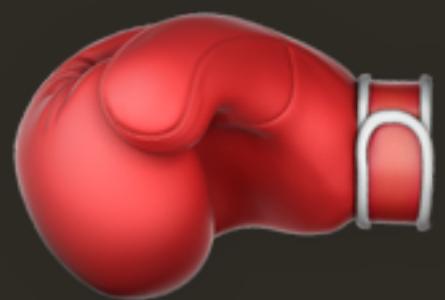
`HOMEBREW_USE_INTERNAL_API=1`



brew prof



ghcr.io



bash

"Homebrew is fast(er)"



Act 2

support

"Homebrew compiles"



"Homebrew extracts"



Homebrew on macOS x86_64

Homebrew on macOS x86_64

Homebrew on Linux x86_64

Homebrew on macOS x86_64

Homebrew on Linux x86_64

Homebrew on macOS arm64

Homebrew on Linux arm64

Homebrew on macOS x86_64

Homebrew on Linux x86_64

Homebrew on macOS arm64

Homebrew on Linux arm64

Homebrew on ...?

Homebrew on macOS 26

Homebrew on macOS 15

Homebrew on macOS 14

Homebrew on macOS 26 x86_64

Homebrew on macOS 26 arm64

Homebrew on macOS 15 x86_64

Homebrew on macOS 15 arm64

Homebrew on macOS 14 x86_64

Homebrew on macOS 14 arm64

Homebrew on macOS 26 x86_64

Homebrew on macOS 26 arm64

Homebrew on macOS 15 x86_64

Homebrew on macOS 15 arm64

Homebrew on macOS 14 x86_64

Homebrew on macOS 14 arm64

Homebrew on Linux x86_64

Homebrew on Linux arm64

"Homebrew is slow"



Homebrew on macOS 26 arm64

Homebrew on macOS 15 arm64

Homebrew on macOS 14 x86_64

Homebrew on macOS 14 arm64

Homebrew on Linux x86_64

Homebrew on Linux arm64

26 - 09

Homebrew on macOS 27 arm64

Homebrew on macOS 26 arm64

Homebrew on macOS 15 arm64

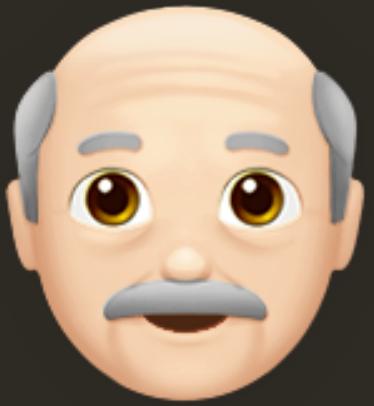
Homebrew on Linux x86_64

Homebrew on Linux arm64

"Homebrew
hates me"



**"Homebrew
cannot support me"**

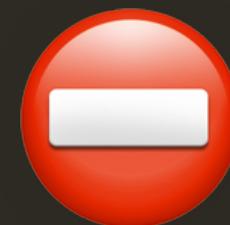




--no-quarantine



odeprecated



odisabled



deleted

Platform Support

Support for different platforms (“targets”) are organized into three tiers, each with a different set of guarantees. For more information on the policies for targets at each tier, see the [Target Tier Policy](#).

Targets are identified by their “target triple” which is the string to inform the compiler what kind of output should be produced.

Component availability is tracked [here](#).

Tier 1 with Host Tools

Tier 1 targets can be thought of as “guaranteed to work”. The Rust project builds official binary releases for each tier 1 target, and automated testing ensures that each tier 1 target builds and passes tests after each change.

Tier 1 targets with host tools additionally support running tools like `rustc` and `cargo` natively on the target, and automated testing ensures that tests pass for the host tools as well. This allows the target to be used as a development platform, not just a compilation target. For the full requirements, see [Tier 1 with Host Tools](#) in the Target Tier Policy.

All tier 1 targets with host tools support the full standard library.



Homebrew Documentation

Support Tiers

Homebrew defines three support tiers to help users understand how well Homebrew is expected to work on different systems.

These tiers describe the level of compatibility, automation coverage, and community support that the project actively maintains. They also set expectations for how we handle issues, pull requests, and regressions.

Tier 1

A Tier 1 configuration is considered fully supported. These configurations receive the highest level of CI coverage and are prioritized during issue review and formula development.

<https://docs.brew.sh/Support-Tiers>

Open Source Maintainers Owe You Nothing

19 March 2018

This post is heavily inspired by my experience over the last ten years participating in the open source community and eight years as a maintainer of Homebrew (which I've maintained longer than anyone else at this point).

Burnout is a big problem for open source software maintainers. This is avoidable; maintainers can have fun, keep healthy and be productive working long-term on open source projects. How? By realising they have zero obligations to any other maintainers, contributors or users of their software even if they have personally benefited from the project (e.g. through self-promotion or donations).

Is there any basis to state that maintainers have no obligations? In fact, yes: in open source licenses themselves. Let's start by looking at the most popular open source license used on GitHub: the MIT license.

“ The software is provided “as is”, without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall the authors

<https://mmq.lol/nothing>

Act 3

Money

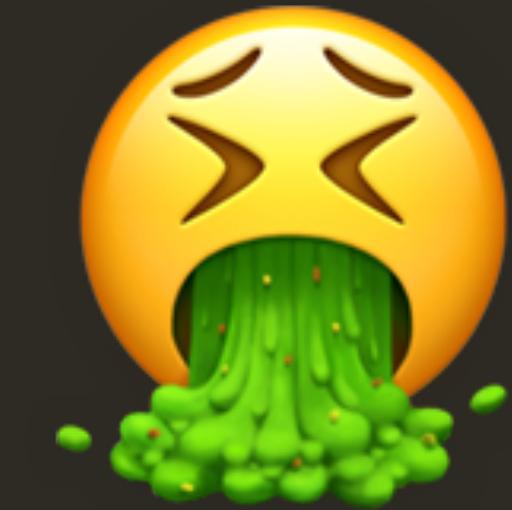
Money

Code

coede

humans

self promotion



Robot Pedantry, Human Empathy

09 June 2021

Homebrew was the first open source project I've maintained where I've had to review and merge contributions from other users. Homebrew is also one of the most active community projects on GitHub with a consistently small team of maintainers (always under thirty in total, always under ten doing work every week). As a result I've had to figure out over the last twelve years how best to manage large numbers of contributions from users in pleasantly and efficiently for both maintainers and contributors.



Manual Process

I've [written before about how Homebrew's CI system has evolved over time](#) but not what I've had to learn to make it work as well as it does.

In the earlier days of Homebrew all review and testing was manual. This involved a maintainer checking out a GitHub pull request onto their local Homebrew installation and verifying it worked as expected. I'm obsessed in my

<https://mmq.lol/robot>

The Open Source Contributor Funnel (or: Why People Don't Contribute To Your Open Source Project)

14 August 2018

Homebrew, the macOS package manager I maintain, is one of the most active community projects on GitHub. We regularly attract large numbers of new contributors and valuable, first-time open source contributions. We've done this by thinking about our users, contributors and maintainers going through a "contributor funnel". If you're wondering why people aren't contributing to your open source project (😭): thinking this way will help you fix this.

Who uses your software?

Let's start with defining groups of open source project users by looking at how they are interacting with your project.

-  the **users** of an open source project are the people who use the software but aren't submitting code, documentation or performing issue

<https://mmq.lol/funnel>

Open Source Economics (is not what you think)

27 October 2021

“Open Source Economics” and the “Open Source Economy” are regularly discussed in the context of how to improve open source software’s sustainability, contributor diversity and ecosystem quality. Too often, though, the use of the word “economics” brings incorrect assumptions about the problems to be solved.



What aren’t “Open Source Economics”?

When most people hear the term “economics” they tend to think about how flows around an economy and, particularly in a capitalist economy, how the allocation of capital affects the throughput of businesses operating in a free market.

As a result, when people hear the term “open source economics” they tend to jump to the same conclusions: it’s about how flows between and is invested in open source projects. This is not a bad conclusion, it results in the

<https://mmq.lol/economics>

Saying No

20 January 2022

I've developed a bit of a reputation inside and outside work as being pretty good with "saying no" and setting boundaries. In this post, I will help you do so, too.

Why say "no"?

I've been consuming a lot of Brené Brown's [podcasts](#) and read [one of her books](#) recently, and they've got me thinking about what's at the root of being able to say "no" regularly and well.



<https://mmq.lol/no>

Open Source Maintainers Owe You Nothing

19 March 2018

This post is heavily inspired by my experience over the last ten years participating in the open source community and eight years as a maintainer of Homebrew (which I've maintained longer than anyone else at this point).

Burnout is a big problem for open source software maintainers. This is avoidable; maintainers can have fun, keep healthy and be productive working long-term on open source projects. How? By realising they have zero obligations to any other maintainers, contributors or users of their software even if they have personally benefited from the project (e.g. through self-promotion or donations).

Is there any basis to state that maintainers have no obligations? In fact, yes: in open source licenses themselves. Let's start by looking at the most popular open source license used on GitHub: the MIT license.

“ The software is provided “as is”, without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall the authors

<https://mmq.lol/nothing>

Acts

- 1 Default to fast
- 2 Publish support reality
- 3 Optimise for maintainers

You!



Make something faster



1 sentence "supported"



Set maintainer boundary

Questions?



mike@brew.sh

<https://mikemcquaid.com>