J. Random Hacker

Dr. Emma Carelton

PHIL 2990

29 November 2024

A Thesis of Modernlangs

In a brief comment she gave me, Molly (whom prompted the creation of this essay) remarked that Rust and Golang are inherently products of modern-day capitalism, feature a very narrow worldview, are Western-centric, and exhibit classist ideology. While this may seem a tall tale, I wish to explore this idea in the following paragraphs. Despite being a citizen of the modern global village bought about by the Internet, I always like to take a step back and examine where we came from; I also like to explore the merits of various ideologies as they relate to computing. As much as I am an engineer, I am also a critic: this essay will explore some of the criticism of Rust and Golang that we both seem to have.

The leading thesis that Molly made was that Rust (and, more importantly, the Cargo build system) makes a lot of brash assumptions -- some of which may not be true for a large amount of the world. The standard tooling for Rust projects is to use the Cargo utility: it provides package management and program compilation/maintenance in one combined package. This is a pivotally useful feature, yes, but introduces a whole sort of downsides. For one, when using Cargo to build a Rust program, you must load the dependencies beforehand. If you are forced to install a program from Cargo rather than your operating system's package manager/load the binaries yourself, you must download up to hundreds of crates from crates.io (paraphrasing Molly here). Take, for example, you are stranded in rural Mississippi and do not have a stable internet connection. As much as I am connected to the world at large, I find myself in these shoes very

often -- I will find myself in a somewhat desolate place many times throughout the year, where the only form of internet connection that is available is a ˜8000 baud packet radio connection via amateur radio. This is not suitable at all to download all of the dependencies for a Rust package, source *or* binary. Likewise, if you are building with Cargo, you cannot take use of any libraries your operating system already has installed, because Rust does not fancy reusing libraries already on disk. If you are on a slow or metered Internet connection, this operation is already unnecessary and very time-consuming.

Do contrast this with most Linux distributions and BSD systems. I can easily compute a list of every dependency a standard C program will need on a Linux system by doing a dependency tree traversal of the Linux package build script or BSD port description file. If I already have a version of a library on disk that is compatible, I do not need to fetch some other specific version (since most C libraries are forwards-compatible). This is in direct contrast to Rust and Go's build systems -- Go fares better in this regard, but you are still fundamentally downloading a large amount of possibly-duplicated information. Do realize that there are plenty of people in this world that (either through choice or by a lack of any other option) use offline package stores to put together their operating system: Debian still has the means to use apt from a CD, as do a few other Linux distributions. Even if you are on a slow internet connection, having a disc with all of the packages you need on it can be beneficial; it is a shame that Rust and Golang do not foster this mindset. Instead, you are totally dependent on a third-party service that you have little means of effectively replicating to provide you everything you need for the complete supply chain.

Now, a common objection is that most Rust code is on GitHub or GitLab -- this is not much better, since these are still single points of failure. Assuming that 90% of Rust projects are

hosted on GitHub, 8% on GitLab, with the remaining 2% on Sourcehut, this is still not a uniform distribution. If GitHub went down even for a moment, that would be rather crippling; contrast this to the older-and-grander years of open source, when most projects were stored all over the internet. Major projects that were commonly downloaded were mirrored to, in those days, primarily servers hosted at universities. If MIT's source-and-binary Linux repository server went down, there were archives available elsewhere. Now, this brings an interesting remark from a series of important essays written in the 90s: Eric Raymond compared and contrasted two open source development models that he called "the cathedral" and "the bazaar." The bazaar model is commonly followed by most modern-day open source projects (including Rust), where the source code is the primary thing that is distributed, and prospective developers can easily get involved. The inverse of this is the cathedral model, where the working source repositories are difficult to access, not commonly accessed, or otherwise not available to access; this is the model used by, for example, GCC. You download a GCC release tarball from one of a thousand mirrors, rather than download the source code from a singular Git repository on GitHub. The bazaar model is known for being more democratic, but this association fails the instant one realizes that it is purely dependent on centralized infrastructure (GitHub, et cetera).

Fundamentally, Rust and Golang represent a hijacking of leftist ideals when applied to software. Large businesses have hijacked open source developers as they squeeze money of a product that, by all definitions, contains the word "free." Microsoft and GitHub produced the Copilot AI model without any consideration for the will of the open source developers it learned from, for example. Rust and Golang also fundamentally pit poorer people at a disadvantage: if you are not wealthy or do not wish to spend money on a fast computer, you will be waiting long periods of time to compile your Rust and Golang programs, thanks to spotty binary availability.

If you are someone that actively wishes to live a simpler lifestyle and use older computers (as they solve your problems just fine), you are automatically locked out of software support just by programming language alone. I myself often use older computers for many things, simply because they are simpler to repair, are of leaner designs, and service most computing needs for me (email, word processing, etc) just fine. I, however, cannot compile the latest and greatest Rust or Golang programs on a 20-year-old Alpha AXP UNIX workstation. I do understand that I am at fault for making this choice in the first place, but, keep in mind, that 20-year-old Alpha is running the latest GNU/Linux operating system, just devoid of Rust and Golang programs (with shockingly good performance, too).

I grew up quite computationally poor; actually, I was poor in many regards. If I wanted to make music, I didn't go out and buy a synthesizer with money that I did not have, I added an electronic-sounding rank to my 1927 theatre pipe organ. If I wanted to play a video game, I searched around an archive of CDs I had for some killer game from 1999. Needless to say, all of this kept me quite entertained, and I didn't need an Internet connection for any of it (even though I was, by all regards, self-imposing these limits upon myself). I found myself playing second fiddle to some of my less-than-fortunate friends: when I was young, some of my friends did not have stable Internet connections. They, however, wanted to use Linux, because I taught it to them at school. My solution to their Internet problem? I burnt them all stacks of DVDs of whatever release of Debian was common in those days (8 or 9, I believe it was). They would install the OS, catalog the discs, and install packages. Rather than fetching the packages from the Internet, it prompted them to insert a disc into the DVD drive. I saw firsthand what computing poverty was really about, and I find it quite odd that Rust and Golang programmers would sit here and tell me that their dependency on cut-and-dry solutions is good for people like my friends. No,

they are obviously not. So many people box themselves into this pipeline of "store the source code on GitHub, rely on dependencies from crates.io, build and deploy to Docker Hub, download image from Docker Hub, and run on a VPS somewhere." There are so many loss-of-control variables involved in that equation, I struggle to rationalize any situation in which this is good for the "little guy." Microsoft decides you're some kind of anti-Microsoft dissident? There goes your source repository! Better hope your PC that has the only copy of the repo doesn't go up in smoke! Burning too much compute time on GitHub Actions? Too bad, you won't be using any compute time anymore! Docker Hub gets blown away thanks to a configuration error or miss-slipping hand of their system administrator? There goes a vital part of your build pipeline!

Fundamentally, I do not believe that Rust and Golang are setting good precedents for the future of computing. How will people like me, who actively maintain legacy systems in the current era, have to deal with all that "legacy" Rust code in 2050? How am I going to have any hope of tracking down every exact version of each dependency for a Python script written yesterday in twenty years? I can easily go acquire the libraries needed to compile a 20-year-old 3D shooter, they came free with the installation of my compiler. I cannot say it will be easy to build a 200-dependency Rust or Golang program in the future. We are not taking sufficient steps in the current age to prepare future generations to be stewards of the software we make now. I am fully aware that this is not an issue unique to Rust and Golang, but could be extended to all modern computing in general with some adequate rhetorical work; I am merely selecting Rust and Golang since the supporters and users of both tend to be very left-leaning, yet they are being clearly blindsighted by capitalism in its purest form. When you use the standard Rust build pipeline, you are being fully dependent on a cathedral-model-style organization that you cannot assume has your best interests tomorrow in mind. This is the opposite of democratization, this is

actually quite a case of strong centralization. I do hope people will realize how anti-them the Rust, Golang, and Python ecosystems are, and I do hope that we can, one day, mass-replicate source code repositories with commit history for all to grab! I see a bright future, where everything plays along nicely, but we must first build it for that to become a reality.