Daniel Parks • daniel@demonhorse.com • 503-913-9333

My calling is building tools for humans and I've spent the last two decades doing exactly that.

I have extensive experience taking projects from conception to production. I spent years helping small businesses define and build web applications and tools. I've both built applications from the ground up and added features to old, hairy codebases. I know when to go fast and dirty, and when to go slow and clean.

I'm looking for a non-management technical leadership position. I'm flexible about what languages are involved, but I'm particularly interested in writing Rust, Python, Go, or JavaScript. I hope to work closely with other engineers — I believe that teams succeed when members support each other and grow together.

Work experience

2020 to present — Open source contributions

After leaving Puppet, I took a few years to resolve a medical issue. While looking for a new position I've spent my time making contributions to open source projects. I've described a few of my own projects below.

2015 to 2019 — Puppet — Senior Software Engineer, Manager of Engineering

I joined the Puppet SRE team to build tools for the team and their clients. Notable projects included an automatic GitHub mirror in Python, tools to manage OpenStack, and an automatic server inventory system in Ruby.

As a manager, I helped my team grow from a group of individuals working on their own projects into a focussed team. In addition, I managed vendors, improved our agile process, and significantly reduced our on call burden.

2007 to 2014 — Contractor

I worked with clients to define the *right* requirements and make the *right* trade-offs. Often clients needed an outside perspective to see what was and wasn't necessary.

For example, I worked with an e-commerce client to evaluate applications for on-site product review. We chose to build a tightly scoped, custom review system which saved them more than $10,000 a year compared to a SaaS solution.

A typical project involved a PHP or Ruby backend on top of a MySQL or PostgreSQL database. Frontends were primarily rendered server-side, but generally had interactive functionality implemented in Javascript and CSS. I worked extensively on all parts of the stack, from writing CSS and slicing images to designing database schemas and configuring servers.

I also built related tools, such as an importer for a single massive XML file containing 20,000 products, and a web crawler that scraped car ads from dealer web sites.

2001 to 2006 — Verinform — Software Engineer

I was an early employee at Verinform, a start-up building software for teaching hospitals. Our product was a web application written in PHP with an Oracle database; this predates even Prototype.js so we used raw JavaScript and CSS.

I built our user interface framework, the third party scheduling software integration, and a duty hours tracking interface. The duty hours interface remains my proudest achievement — I built a beautiful, easy-to-use UI that residents (doctors in training) actually enjoyed using despite having little incentive to do so. It ended up being a major selling point for the software.

Open source projects

See demon.horse/portfolio for a more extensive list.

yanki — github.com/danielparks/yanki

Yanki is a Python command line tool made to simplify creating thousands of video flashcards from YouTube videos.

I designed Yanki to be an authoring tool. It can produce flashcards in the popular Anki format, or it can produce static HTML that can be hosted on any web server. Most people will find the HTML app much easier to use.

Internally, Yanki includes a simple parser for deck definition files, and uses ffmpeg to trim, crop, and/or slow video. Since ffmpeg is primarily single threaded when encoding media, Yanki uses Python's `asyncio` to run multiple instance in parallel and significantly cut down the time needed to process thousands of flashcards.

git-status-vars — github.com/danielparks/git-status-vars

I configure my shell prompt to display information about the current git repository, e.g. the name of the current branch, whether or not there are uncommitted changes, etc. Loading that information required multiple calls to `git` which added a noticeable delay after every command, including trivial ones like `cd`. Git-status-vars uses `libgit2` to quickly read common repo information, then prints the information in shell variable form. The output can be `eval`ed and the variables can be used in a prompt.

htmlize — github.com/danielparks/htmlize

Htmlize is a Rust crate that *correctly* encodes and decodes HTML text.

Encoding entities is easy and fast, but decoding entities correctly has two major edge cases: not all entities have to end with a semicolon, e.g. `&copy` (©), and some entities contain other entities as a prefix, e.g. `&copysr;` (℗).

After significant experimentation, I chose two fast decode algorithms. They produce identical results, but switching between them trades faster compile times (with a perfect hash) for faster run times (with a massive `match` tree generated by my crate matchgen).

Last updated October 23, 2025