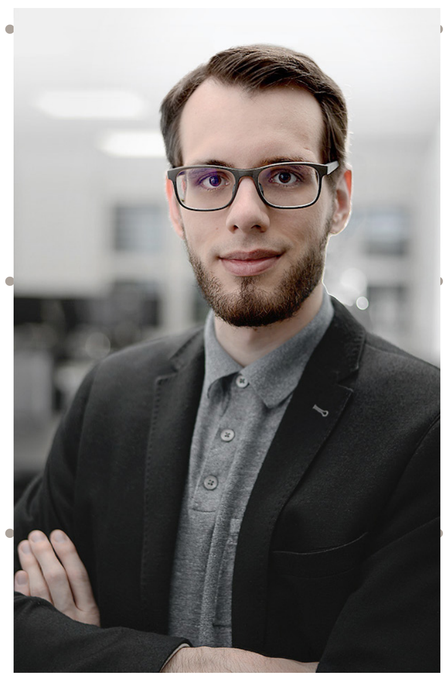
INTERVIEW

New Ports Committer: Joel Bodenmann (jbo@freebsd.org)

INTERVIEWED BY TOM JONES

TJ: Hi Joel, welcome to the project. Could you give me a little background on yourself and the sort of technology projects you enjoy working on?



JBO: I'm an electronics engineer mainly focusing on embedded systems. Usually I like working with systems that are designed to perform a specific task rather than generic computation.

TJ: FreeBSD isn't famously a great platform for embedded systems development, have you tried to work or do work style projects on FreeBSD?

JBO: I think I have to disagree with your statement. The reason why I got to FreeBSD is exactly to use it as a platform for embedded systems development. Although the world of "embedded" has changed drastically the past few years, the embedded systems I usually work with are comparably low-resource, real-time systems (i.e., your typical microcontroller based systems with < 120MHz CPU, < 128kB RAM). As such, these systems are not designed to run FreeBSD themselves, but they also wouldn't run Linux in any practical sense (nor would the requirements allow for that).

You still need a (desktop) host system for the actual development as well as surrounding support infrastructure tho. These embedded projects I usually work on can take several years from first-meeting to deployed product. One of my mood points with using Linux as a development platform is that the Linux ecosystem is in a constant state of flux. You setup your development workflow and environment on Linux based systems only to be forced to upgrade a few months later which requires to revalidate your entire workflow.

FreeBSD is well known for it's stability and coherence. Rather than implementing a new system or component from scratch because the previous one is not "good enough" anymore, FreeBSD tries to design those systems and tooling to be maintainable and expandable, vastly reducing development system management overhead.

For some projects, I do need a non-microcontroller-based system to which the microcontroller-based systems talk (i.e., for long-term data logging, orchestration etc.). In those scenarios the same benefits of FreeBSD apply: You setup and validate your system once and then just perform smaller maintenance tasks over the years, whereas with Linux-based systems I never knew what I would wake up to the next day.

A crude summary would basically state that I like to spend my time working on the actual development/engineering of the system I am working with rather than updating, debugging and revalidating my development platform just because the underlying init system, audio subsystem, hypervisor or most popular container system or similar changed three times in two years. FreeBSD checks all the boxes here. Since I migrated all of my servers, network

infrastructure and workstations to FreeBSD, I never had to guess whether my development infrastructure still boots and works when I enter the office the next day. It's just rock solid.

TJ: Have you taken on a particular area of focus in the ports tree?

JBO: So far, I have been mainly focused on handling the "low-hanging fruits" with the idea that this allows me to familiarize myself with the basics of the ports system as well as the workflows and infrastructure. At the same time, this frees up resources of the more experienced ports committers so they can spend their efforts on more complex matters.

As I am getting more comfortable with this, I hope to take on more involved tasks in the coming year such as updating ports of libraries with many consumers.

As such, my answer to your question is: No, I try to help wherever I can. I have no doubt that with increased experience I will soon find something larger to work on :)

TJ: The ports collection is massive and a lot of small tools can do heavy lifting. What are some of the low-hanging ports you've worked on so far?

Do you have any suggestions for others to find low-hanging fruit and easier first ports?

JBO: Personally, I considered two scenarios to be low-hanging fruits so far:

1. PRs with patches that already have maintainer approval, where the patch updates an existing port to a new upstream minor version. Here I'd recommend to stay clear of "fundamental" ports with a high number of consuming ports in the beginning to reduce the risk of triggering massive fall-outs.

   My reasoning here is that comparably little can go wrong with simple upstream minor version bumps and maintainers tend to be careful not to break their ports and already have the experience in what to watch out for specifically with regard to their upstream.
2. PRs which introduce a new port. These PRs tend to be "low priority" so there's comparably little pressure to get them landed quickly. Furthermore, I identified these as a great way of learning about the different systems and mechanisms that the ports framework provides that I might not have been in contact with yet.

"So far, I have been mainly focused on handling the "low-hanging fruits" with the idea that this allows me to familiarize myself with the basics of the ports system"

As for what I have worked on so far: I intentionally tried to touch various ports from various domains. I am not focusing on a particular category or type of port. Instead, I try to handle PRs of ports I know rely on something I have not yet worked with. This is a great way of getting comfortable with the various Mk/Uses/* scripts.

TJ: When you look to the future of FreeBSD what do you think the main priorities of the project should be from a porters perspective?

JBO: I think that a big part of what makes FreeBSD such an attractive OS to a variety of users is the fact that we usually follow more old-school principles. The FreeBSD project tends

to take the "slow but steady" approach rather than constantly jumping on the latest hype train re-inventing things constantly. There are, of course, drawbacks to this approach as well. For example, the ports framework lacks some features that might be considered "basic" by modern standards such as sub-packages, suggesting optional installs and similar. But I'd argue that exactly because these things are not rushed, we end up with a much more stable and easier to maintain system.

As such, rather than answering your question with a concrete list of steps or goals that need to be accomplished, my main recommendation is to stay true to this approach. Anything that ends up proving to be necessary will happen eventually, but it will usually do so in a non-forced way allowing for proper design, implementation and testing resulting in efficient use of our limited man power.

There are many bullets that one can dodge by simply not rushing or forcing progress. The expression "slow is smooth; smooth is fast" applies here in my opinion.

TOM JONES is a FreeBSD committer interested in keeping the network stack fast.

JOEL BODENMANN bio to come