SEE TEXT ONLY

this month In FreeBSD

BY DRU LAVIGNE

Over the next few issues, we'll be taking a closer look at some of the new features making their way into the 2016 releases and the developers behind those features.

This month we chat with ALLAN JUDE. Allan has been a very busy contributor since receiving his doc commit bit in 2014 and his src commit bit in 2015. Several of his projects will make it into this year's 10.3 and 11 releases.

Q Tell us a bit about yourself. How did you get started with FreeBSD and what is your involvement with the FreeBSD Project?

A I started as a FreeBSD user back in 1999 when I wanted to run my own IRC server. After downloading the software from the Internet, I found there was no .exe in the archive. After asking around some, I was told that I would need a "shell account" to be able to run the software. I found a provider that would rent me such a shell account for around $10/month. Now-a-days you can rent a virtual machine with about four times as much RAM and even more CPU for about the same price, even with 16+ years of inflation.

I was fairly comfortable with the command line, but had never used anything other than DOS and Windows, so there was a lot to learn. As I learned more and more, I eventually stopped renting shell accounts and started selling them. In 2002 I started my own business renting IRC servers and shell accounts, and quickly became quite good at securing my servers against malicious users. As my knowledge expanded, I started also doing consulting work, setting up web-hosting stacks like Apache/MySQL/PHP for people, custom compiled from source.

When I went to college, I was lucky enough to go to a school that actually taught Linux and BSD together. There I met some other BSD users and first heard about this wonderful thing called BSDCan. Sadly, rather than going to my first BSDCan in 2006, I didn't make it to one until 2012. When I finally did attend, it changed everything. I didn't get all that much out of my first BSDCan, other than the excitement and energy that was in the air, and meeting a few people I had only heard about on mailing lists. I decided that not only would I come back next year, but I'd submit a talk, and share some of my BSD sysadmin experience with the audience. Why wait for next year? So I submitted the proposal to EuroBSDCon 2012 in Poland, and was accepted. Giving my first talk was quite an experience, but it was well received, and the audience asked insightful questions. I've been to almost every BSD conference I could manage since then.

In 2013, on the way to BSDCan, I decided that the documentation in the FreeBSD Handbook on ZFS was lacking a lot of detail, so I started writing. I attended my first doc sprint, and got up to speed on how to write documentation for the project. In August of 2013, a fan of my sysadmin podcast, TechSNAP.tv, asked me about starting a BSD-centric podcast. At first I didn't think there would be enough news to sustain a weekly show, but after the viewer, TJ, who became our first producer, and I began writing up overviews of a few weeks' shows, I quickly learned otherwise. After a short search, Kris Moore was tapped to be my cohost. Then, a few months later, on my flight to EuroBSDCon 2013 in Malta, I started looking at what could be done to make the FreeBSD installer handle ZFS better. My first draft was sloppy and full of errors, but an especially helpful committer worked with me to get it into shape. Before long, it was part of the operating system. BSDCan 2014 included a big surprise, being granted my doc commit bit. Shortly after BSDCan 2015, I was granted my src commit bit.

Now, I use FreeBSD every day at my day job, ScaleEngine.com, a video streaming company, except on Wednesdays, when I do the BSDNow.tv podcast with Kris Moore. Then my evenings are spent hacking on diverse projects, from documentation, to standardizing configuration files across utilities, to improving the command line interface of ZFS, and adding features to the FreeBSD bootcode and loader.

Q For some time, you have been working with UCL. What is UCL exactly and how does it compare with other configuration languages? What advantages does it provide over traditional configuration files?

A UCL, or the Universal Configuration Language, was designed by another FreeBSD Developer, Vsevolod Stakhov <cebka@freebsd.org>, for his own utility, rspamd. It consists of a library, libucl, that can parse the config files into objects that the application can use to configure itself, and which can emit modified config files back out. The idea is to have a config file that is easy for a human to read and write, while at the same time being able to manipulate it programmatically—a config format that any experienced sysadmin or user will understand and appreciate.

The UCL config syntax itself is based on the syntax of NGINX and Bind, but slightly modified in a way to make it also compatible with a loose interpretation of JSON:

```
category {
  key = value;
}

othercategory {
  subcategory {
    max_size = 10kb
    expiration = 3d
    array1 = [1, 2, 3]
    array2 = [
      thing1,
      thing2,
      thing3,
    ]
  }
}
```

UCL syntax has a number of features that make it easier for humans to write. It does not require the semicolon terminator at the end of a line. It supports arrays, but unlike JSON, allows the last item in a list to be followed by a comma, to reduce the diff as items are added to the list. There is also "syntax sugar," where values can contain units, like k (1000), kb (1024), h (hours), and d (days). Booleans can be specified as any of: true/false, on/off, or yes/no. UCL also supports comments, in all three popular styles, including single line (//) and multi-line (/* ... */) C comments, and the standard hash (#) character that many config file formats use.

The power comes from the native support for variables, macros, and includes. Variables are set by the application, for the user to use in their configuration file, like $HOST for the system hostname. Macros allow the application that is interpreting the configuration file to extend the configuration language with additional functionality. The includes system allows additional files to be included into the configuration. Includes supports priorities, so the application or user can control which value is used if a setting is defined in two different bits of configuration. Support for GLOB patterns and search paths allow multiple files to be included, such as /usr/local/etc/appname/*.conf. There is also support for remote includes, with optional signature verification.

libucl itself can interpret UCL, JSON, YAML, and Msgpack. It can also output the configuration in all four of these formats. An application that uses libucl to parse its config file, will accept any of these; as to the application, they all look the same.

Q FreeBSD is transitioning to using UCL. Can you describe what users can expect in the upcoming 10.3 and 11.0 releases of FreeBSD?

A Time constraints and compatibility concerns mean that users of 10.3 will not see any changes. However, starting with 11.0, a number of config files will change format. For compatibility and ease of transition, all of the tools will continue to accept their original file formats, but will gain support for the new UCL-based config files. One of the first things I implemented as part of the UCL conversion was a version identifier in the config file that will make future transitions easier, and allow applications to detect when they are given a new config file. Users who upgrade an existing machine to 11.0 will have the option to just continue using their existing config files, or try to convert them. Users who install a fresh 11.0 system will have the new UCL config files by default. The only complication here is documentation: since the old format is still supported, the documentation will need to be retained, if for no other reason than to be a resource to those who are converting their configuration files.

At this time, there is no plan for tools to automatically convert old config files to the new format, but there is nothing stopping someone from adding one to the ports tree.

My initial targets for conversion are: newsyslog.conf, login.conf, jail.conf, pw.conf, and wpa_supplicant.conf.

There are a lot of config files to cover, so I am always interested in hearing from people who want to help, or even just comment on which config files they would like to see converted next.

I would also like to revive my bhyveucl (github.com/allanjude/bhyveucl) project. Originally it was a shell script that read a UCL config file and wrote the complex bhyve command line to launch the VM as described, and configure the network as required. It was sidelined when I joined a project to implement the config file parsing directly in bhyve itself. That project was put on indefinite hold at the request of the bhyve authors, as they are working on a number of enhancements, including USB support, a plugin architecture for networking and storage, and a host of other things. These will require a much more expressive config file than I had originally designed. However, that work is taking longer than expected, so there may be value in reviving bhyveucl. Even once the work in bhyve is completed, the configuration file may be rather complex, and a utility like bhyveucl that can take a simpler config file and convert it into the more expressive "machine description" that bhyve will require may still be of great value.

"At this time, there is no plan for tools to automatically convert old config files to the new format, But there is **nothing stopping someone from adding to the ports tree.**"

—ALLAN JUDE

Q Improvements are also being made to bsdinstall to provide support for ZFS boot environments. For readers unfamiliar with this feature, can you provide an overview of its benefits? What type of work was needed to add this support and when will it be available to users?

A ZFS boot environments are a way to have multiple root (/) file systems and switch between them at reboot in order to revert a problematic upgrade, or to dual boot multiple versions of the operating system. They can be managed manually, but there is a utility in ports, sysutil/beadm, that provides a nice user interface. Each boot environment is a ZFS dataset, although in most cases, it is a ZFS clone of the existing file system, so takes no additional space until changes are made. If you clone your / file system before an upgrade, and the upgrade does not work as expected, just use 'beadm activate oldbootenv' and reboot, and your system will boot from the clone of / from before the upgrade. Other file systems, like users' home directories, are not affected.

The initial support for ZFS boot environments was introduced to the FreeBSD installer in FreeBSD 10.0. When you use the 'Automatic root-on-ZFS' mode in the installer, it creates poolname/ROOT/default, which will be your first boot environment.

There are two ways to use boot environments. PCBSD creates a new environment before each upgrade, starts a jail chrooted in that environment, and does the upgrade there, then reboots into that environment. I personally prefer to just use the 'default' boot environment, and create clones of that before each upgrade, in case I need to roll back. Another key factor is deciding what should be included in the boot environment, and what should remain untouched when switching between them. The base operating system (/bin, /sbin, /usr/bin, /usr/sbin, and /etc) is usually included, but it can depend on your environment if you want /usr/local (where applications installed with pkg are put) to be unique to each environment, or stable across them all.

The problem with boot environments as they shipped in 10.0–10.2 is that if you end up with one that doesn't boot correctly, the only way to switch is by manually manipulating the loader prompt, or booting from a live CD/USB and switching the 'active' boot environment.

I have been working on an additional menu option in the beastie loader menu that allows you to select a different boot environment. This allows you to quickly, easily, and safely switch between the different root datasets at boot time. What sets this apart from the way it is currently done in PCBSD with GRUB, or in IllumOS, is that rather than reading the list of boot environments from a configuration file, the list is generated by examining the ZFS pool itself, so the list is always up to date.

The other complication is encryption. If the user GELI encrypts their ZFS pool, they currently require a UFS partition, or a second not-encrypted ZFS pool to sort the kernel on, so that support for GELI encryption could be loaded to decrypt their file system. In both cases, it is not possible to support boot environments, because the kernel does not reside on the boot where the environments were created.

I have been working on solving this issue by implementing support for GELI decryption in the bootcode and loader. I hope that this work will also make it into 10.3. This will allow booting om a single encrypted ZFS pool, offering full support for boot environments, even in the presence of encryption.

Q You have written extensively on ZFS, including the ZFS chapter of the FreeBSD Handbook and the ZFS Mastery series of books with Michael W Lucas. How did you get interested in ZFS and what benefits has it provided to you and your business?

A I first got interested in ZFS when my company needed to store large quantities of video files, in a flexible and safe way. I had little experience with enterprise storage, having never used anything more complex than motherboard BIOS-assisted mirroring on any of my machines. I found the concept of pooled storage and copy-on-write to be very interesting and to fit our needs quite well. The administration and configuration interface for ZFS was also extremely easy and powerful, and I quickly became very comfortable with it. The more I learned about ZFS, the more I wanted to share it with everyone.

I have learned a lot since setting up that first ZFS server in 2011. At our company each customer gets their own ZFS dataset, so we can more easily move customers between servers, manage their snapshots independently, and assign quotas. When I first deployed ZFS, we just had one big dataset for video, and each customer had a directory. The problem was that if a customer purged a large number of their videos, or cancelled their account, we did not regain that space until the snapshots were destroyed. We did not want to lose the snapshots of our other customers' data, while at the same time being able to get that space back, so we have transitioned to using a separate dataset for each customer. We also use the ZFS space accounting system to bill our customers.

The biggest advantage to our business has been the flexibility, resiliency, and tunability of ZFS. With the tools provided by FreeBSD and ZFS, not to mention DTrace, we can easily monitor and diagnose any performance problems. Having the details of each disk directly exposed to the operating system, rather than hidden behind a RAID controller has also saved time and made managing failing disks easier.

Q The BSD Now podcast is into its third year. Have you seen any trends in BSD usage or perception over that time?

A When the idea for the show was first proposed, I didn't think it would work. After we worked on it for a while and actually started it, I was pretty confident in it. The biggest surprise to me has been that neither Kris nor I have burned out yet, and still enjoy doing the show each week. I think a big part of that is the very positive feedback we get from the community. My personal favorite part of the show is the interviews.

I have definitely felt an uptick in the perception and adoption of BSD since the launch of the show. We get feedback every week about someone new trying out, or switching over to, a BSD. My concern is the retention ratio, how many of them stick with it.

I wonder what things the projects can do to help newcomers get over those initial hurdles, and become lifetime users like I have. I think the biggest complaint we see is hardware support on newer laptops, and I hope that is something that can be addressed going forward.

Q What other projects are you working on or plan to start in the near future?

A In addition to more work on implementing UCL config files in FreeBSD, and my command line utility for working with UCL called uclcmd (github.com/allanjude/uclcmd), I have a number of ideas for new ZFS features. Along with a system to track changes to the ZFS command line interface, for use by scripts that automate parts of ZFS, I am also working on implementing the additional hashing algorithms that were recently added to ZFS to the FreeBSD kernel, so they can be used on ZFS. One of them, SHA512t256, which is a SHA512 truncated to 256 bits since that is the maximum size of the checksum in ZFS, is approximately 50% faster than a regular SHA256 when calculated on 64bit x86 hardware. I have also been working on converting utilities in FreeBSD to libxo, a library that makes the utilities able to output JSON and XML, in addition to the regular text output.

My slightly loftier goal is a project called Zoro, a successor to sysutil/zxfer, to manage ZFS replication. Zoro would manage snapshot creation and retention, bookmarks, and replication. Some of the ideas I have for it will be best implemented with a little help from the ZFS side, so it might lead to some new ZFS features as well.

I have my fingers in a number of different areas of the system, either trying to make my own job easier, allow FreeBSD to reach more of the potential I see in it, or solving problems that others have, in hopes of increasing adoption of FreeBSD and ZFS. ●

Dru Lavigne is a Director of the FreeBSD Foundation and Chair of the BSD Certification Group.