FreeBSD® JOURNAL

January/February 2022

Software and System Management Issue

CBSD Part 1

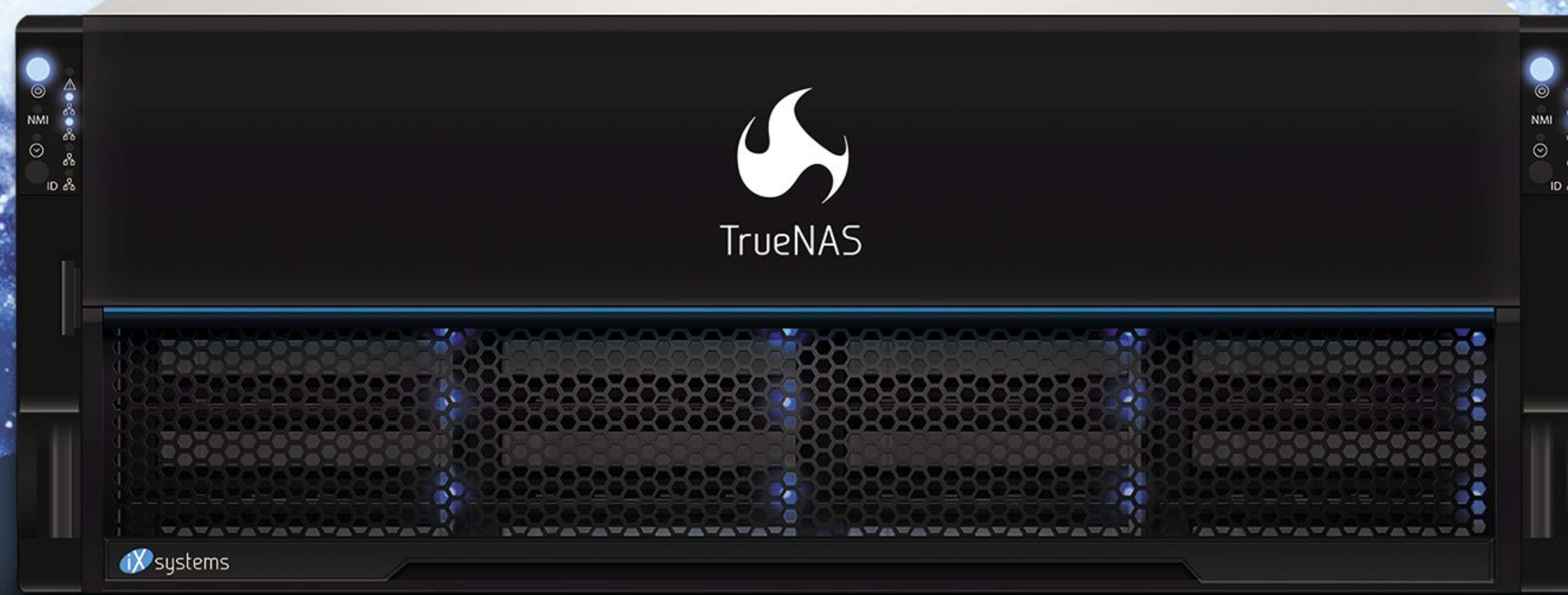
Contributing to FreeBSD Ports with git

pf_syncookies

A review of *The Kollected Kode Vicious*

TrueNAS® M-SERIES
Powerfully Scalable Enterprise Storage

OPEN STORAGE FOR ENTERPRISE WORKLOADS



UTILIZES FLASH-OPTIMIZED ZFS TECHNOLOGY
IDEAL FOR LATENCY-SENSITIVE AND BUSINESS-CRITICAL VIRTUAL MACHINES AND PHYSICAL WORKLOADS.

PERFORMANCE AND SCALE WITHOUT COMPROMISE

INTELLIGENT STORAGE OPTIMIZATION

SELF-HEALING DATA PROTECTION

UNLIMITED SNAPSHOTS AND REPLICATION

Contact iXsystems to Learn More about what TrueNAS® can do for your business!

iXsystems.com/TrueNAS | (855) GREP-4-iX

© 2020 iXsystems. TrueNAS is a registered trademark of iXsystems, Inc. All rights reserved. Intel, the Intel logo, Xeon, and Xeon Inside are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. All product names, logos, and brands are property of their respective owners.

FreeBSD® JOURNAL

Editorial Board

John Baldwin • FreeBSD Developer and Chair of FreeBSD Journal Editorial Board.

Tom Jones • FreeBSD Developer, Internet Engineer and Researcher at the University of Aberdeen.

Ed Maste • Senior Director of Technology, FreeBSD Foundation and Member of the FreeBSD Core Team.

Benedict Reuschling • Vice President of the FreeBSD Foundation Board and a FreeBSD Documentation Committer.

Mariusz Zaborski • FreeBSD Developer, Manager at Fudo Security.

Advisory Board

Anne Dickison • Marketing Director, FreeBSD Foundation

Justin Gibbs • Founder of the FreeBSD Foundation, President of the FreeBSD Foundation, and a Software Engineer at Facebook.

Daichi Goto • Director at BSD Consulting Inc. (Tokyo).

Allan Jude • CTO at Klara Inc., the global FreeBSD Professional Services and Support company.

Dru Lavigne • Author of *BSD Hacks* and *The Best of FreeBSD Basics*.

Michael W Lucas • Author of more than 40 books including *Absolute FreeBSD*, the *FreeBSD Mastery* series, and *git commit murder*.

Kirk McKusick • Treasurer of the FreeBSD Foundation Board, and lead author of *The Design and Implementation* book series.

George Neville-Neil • Past President of the FreeBSD Foundation, member of the FreeBSD Core Team, and co-author of The Design and Implementation of the FreeBSD Operating System.

Hiroki Sato • Director of the FreeBSD Foundation Board, Chair of Asia BSDCon, Member of the FreeBSD Core Team, and Assistant Professor at Tokyo Institute of Technology.

Robert N. M. Watson • Director of the FreeBSD Foundation Board, Founder of the TrustedBSD Project, and University Senior Lecturer at the University of Cambridge.

S&W PUBLISHING LLC
PO BOX 3757 CHAPEL HILL, NC 27515-3757

Publisher • Walter Andrzejewski
walter@freebsdjournal.com

Editor-at-Large • James Maurer
jmaurer@freebsdjournal.com

Design & Production • Reuter & Associates

Advertising Sales • Walter Andrzejewski
walter@freebsdjournal.com
Call 888/290-9469

FreeBSD Journal (ISBN: 978-0-615-88479-0) is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).
Published by the FreeBSD Foundation,
3980 Broadway St. STE #103-107, Boulder, CO 80304
ph: 720/207-5142 • fax: 720/222-2350
email: info@freebsdfoundation.org
Copyright © 2021 by FreeBSD Foundation. All rights reserved. This magazine may not be reproduced in whole or in part without written permission from the publisher.

LETTER from the Foundation

Welcome to the January/February issue of the *FreeBSD Journal*. A new year brings new opportunities for users and developers alike.

One source of new opportunities this year is FreeBSD's ports collection. The collection is ever-changing as it tracks changes in both third party software and the base system. Mateusz Piotrowski walks through the process of contributing a new port, while Joseph Mingrone describes some of the best practices for contributing changes to ports after the migration from Subversion to Git.

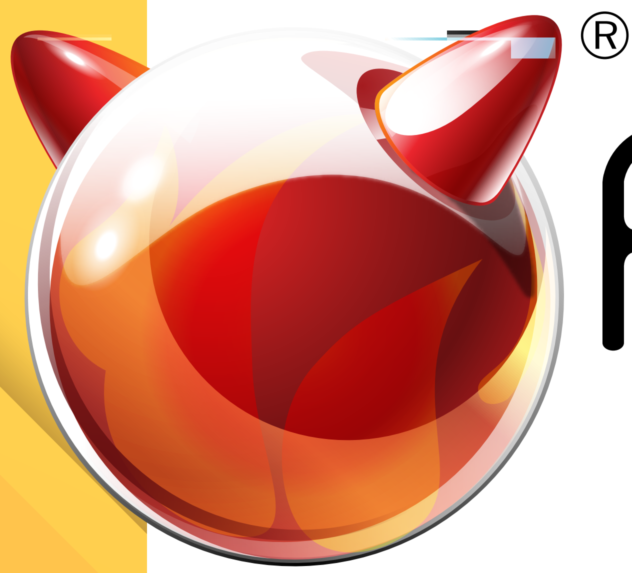
CBSD provides a framework for managing both jails and bhyve virtual machines on FreeBSD. Oleg Ginzburg provides the first installment in a series of articles introducing users to the CBSD framework and how it can be used. In a similar vein, Tom Jones introduces a new tool for creating jails.

Kristoff Provost describes yet another episode in his ongoing work maintaining FreeBSD's port of the pf packet filter.

This issue also features two conference reports from EuroBSDCon. We hope to feature more conference reports in future issues this year both from "established" BSD conferences and new venues.

As always, we love to hear from our readers. If you have feedback on any of our articles, suggestions for topics for a future article, or are interested in writing an article, please email us at info@freebsdjournal.com.

On behalf of the FreeBSD Foundation,
John Baldwin
FreeBSD Developer
and Chair of the *FreeBSD Journal* Editorial Board



January/February 2022

FreeBSD® JOURNAL

Software and System Management Issue

5 Contributing to the FreeBSD Ports Collection
By Mateusz Piotrowski

12 Contributing to FreeBSD Ports with Git
By Joseph Mingrone

27 CBSD: Part 1–Production
By Oleg Ginzburg

39 Porting OpenBSD's pf syncookie Code to FreeBSD's pf
By Kristof Provost

3 Foundation Letter
By John Baldwin

47 WIP/CFT:mkjail
By Tom Jones

49 Practical Ports
A Review of *The Kollected Kode Vicious* ***By Benedict Reuschling***

52 We Get Letters
Packaged Base ***By Michael W Lucas***

56 Conference Reports
EuroBSDCon 2021 ***By Katie McMillan and René Ladan***

62 Events Calendar
By Anne Dickison

Contributing to the FreeBSD Ports Collection

BY MATEUSZ PIOTROWSKI

Why would you use ports?

Many FreeBSD releases ago, the FreeBSD Ports Collection was the primary way to install third-party software. Users would build software they needed from ports. In theory, there were some binary packages available, but the overall support for them wasn't that great. Package management tools were cumbersome. Package repositories contained outdated packages. Building software from ports was a necessity.

The situation began to change around the time when the new pkg(8) package management tool emerged. Nowadays, FreeBSD packages repositories are one of the largest and most up-to-date in the open-source world (see the graphs on repology.org). Most FreeBSD users use binary packages instead of compiling ports themselves and it has been like that since I started using using FreeBSD, which was around version 10.3.

Although binary packages are great, people use FreeBSD Ports Collection directly all the time. Both FreeBSD maintainers and FreeBSD users use it all the time. Why would FreeBSD users use it? Because ports make it really easy to tailor binary packages to very specific needs. Would you like to rebuild the Nginx package with a custom patch? No problem. Would you like to add an unusual backend to your collectd daemon? Easy. Would you like to get a debug build of Python? No big deal.

In this article I would like to give you some insights about contributing to the FreeBSD Ports Collection. How to get started? Why to get started? What does it take to submit a patch? Keep reading if you want to know the answers to those questions.

Hello, my name is Mateusz, I would like to contribute to FreeBSD

This is a message I see all the time, whether on the mailing lists, Discord or IRC channels.

Typically, it gets answered with a bunch of links to bug trackers and wiki pages with project ideas. You would expect that long-time contributors love to introduce newcomers to their projects. This is true (and that is a sign of a healthy open-source community). What is also true is that long-time contributors are reluctant to reply to such messages. Why is that so? Well, they know that there is a high chance that they won't hear back from that newcomer ever again. Yes, you heard me right. Basically, this is not how you get started contributing to open source. Don't get me wrong. I used to send similar messages all the time in the past myself. Why? Because it felt like the right way to start! I had the time and motivation, I just needed the community to give me an interesting project to focus on. Isn't that simple?

It turns out that it works slightly differently. Becoming absorbed in a project listed on some project ideas page is borderline impossible. Project ideas land on wiki pages because no one had the motivation to spend the the necessary amount of time to do the work. How could it possibly be picked up by a newcomer if it doesn't spark joy even among seasoned contributors? I am not sure. Some project ideas just have to wait for their champion.

How to get started then? The truth is that you need to find the area you are passionate about on your own. Here is what you can do. Start using FreeBSD regularly. Explore the system and pay attention to what annoys you. Ask yourself such questions as:

1. How do I get my laptop to suspend automatically if the battery is running low?
2. Could the GPU setup documentation be more straightforward?
3. How cool would be to have a proper icon for Xpdf in application launchers and menus?

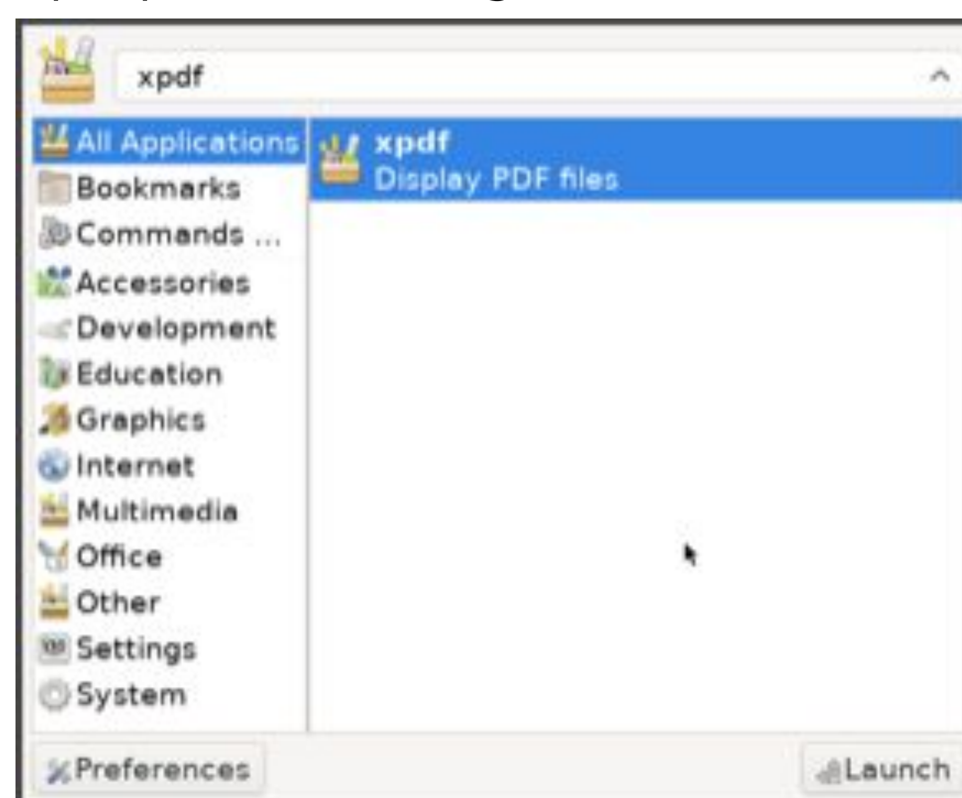
The greatest motivation to contribute comes from an itch to scratch. A problem so annoying that you decide to fix it on your own. A problem so interesting that resisting the urge to to fix it straightaway is futile. A problem so common that solving it surely will give you all the street cred at the next conference. In other words, the easiest way to start contributing is to work on something you need. Of course, you are going to get stuck at some point. What is different this time is that you have a specific problem at hand. These problems attract a good deal of attention in the community. Remember those "unreachable" seasoned contributors? Trust me, they are going to answer your questions much more often now as you seem to be quite motivated to solve some interesting problems there. Because it's much more fun helping somebody out with a problem rather than helping somebody out with finding a problem.

Scratching an Itch (Missing Xpdf Icons Edition)

This part of the article describes a workflow of developing a patch for the FreeBSD ports. When I was starting hacking on FreeBSD ports, I often wondered how others develop their patches. (Now that I think about, I am still fascinated by the efficiency of certain ports committers. Perhaps more than ever.) For some reason exact development workflows are infrequently described in the official FreeBSD documentation. Without further ado, let's see what it takes to cook up a ports patch.

First Encounter

It is a sunny Saturady evening, the snow is melting. You are computing important things on your FreeBSD desktop. You are going through your ever-growing todo list. One item at a time. The next task requires a PDF reader. No problem. Let's use our venerable Xpdf for that. So you fire up Xfce Application Finder and search for Xpdf. Everything is going smoothly. And then you see it. The Xpdf entry does not have a proper icon. Imagine that! This cannot be. We need to fix this.



Before we start hacking on the ports tree, let's understand why the Xpdf icon is missing. Xfce Application Finder generates the list of entries based on the desktop files located in `/usr/local/share/applications`.

The one called `/usr/local/share/applications/xpdf.desktop` describes the Xpdf entry. Let's see if there is something related to icons in that file.

```
$ grep -i icon /usr/local/share/applications/xpdf.desktop
Icon=xpdf
```

The icon's name is xpdf. Let's see if we can find such an icon in `/usr/local/share/icons`.

```
$ find /usr/local/share/icons -name '*xpdf*'
```

The output of our find(1) one-liner is empty. The Xpdf icon is not installed. At this point, we probably need to take a look at the ports tree.

Developing a Patch

The first thing we need is a copy of the FreeBSD ports tree. You can read up on the details in the FreeBSD Handbook (https://docs.freebsd.org/en/books/handbook/ports/#ports-using). Ultimately, the following command is all we need:

```
$ git clone https://git.FreeBSD.org/ports.git ~/ports
```

Now let's examine the Xpdf port. How do we find it among all the ports? There are a couple of different ways to do it.

The easiest way is to ask pkg(8) about the origin of the package.

```
$ pkg search -o xpdf
japanese/xpdf                  Japanese font support for xpdf
graphics/xpdf                  Display PDF files and convert them to other formats
graphics/xpdf3                 Display PDF files and convert them to other formats
graphics/xpdf4                 Display PDF files and convert them to other formats
print/xpdfopen                 Command line utility for PDF viewers
```

pkg-search(8) searches the package repository catalogue looking for package names matching "xpdf". The `-o` option tells pkg-search(8) to show the origin of the package in the output. The origin is the official term for the directory name of a port within the ports tree. This is exactly what we are looking for.

Tip: Sometimes I don't know the name of the package that installed a file I'd like to fix. In those cases I use pkg-which(8):

```
$ pkg which /usr/local/share/applications/xpdf.desktop
/usr/local/share/applications/xpdf.desktop was installed by package xpdf-4.03,1
```

OK, so we learnt that the origin of the Xpdf package is `graphics/xpdf`. Let's see what is inside the port's directory:

```
$ cd ~/ports/graphics/xpdf
$ ls
Makefile
```

Aha! For those of you, who are new to ports: what we see here does not look like a typical port. Usually, you expect to find other files like `distinfo` containing checksums of source code archives, `pkg-descr` containing a longer description of the port, and `pkg-plist` listing all the files this port installs. Let's see what's inside the `Makefile`:

```
$ cat -n Makefile
     1  VERSIONS=               3 4
     2  XPDF_VERSION?=          4
     3
     4  MASTERDIR=              ${.CURDIR}/../xpdf${XPDF_VERSION}
     5
     6  .include "${MASTERDIR}/Makefile"
```

See the `MASTERDIR` variable on line 4? It means that `graphics/xpdf` is a master port. When this port is build, it's actually `graphics/xpdf4` driving the whole process. (BTW, Xpdf version 4 is apparently the default, judging by line 2). Down the rabbit hole!

```
$ cd ~/ports/graphics/xpdf4
$ ls
distinfo    files       Makefile    pkg-descr   pkg-message pkg-plist
```

Just as expected. High time we grabbed a copy of the source code. We do it with the `extract` target defined by the FreeBSD Ports framework.

```
make extract
```

All the source code and build artifacts live in directory `./work` in the port's directory. E.g., the source code of Xpdf gets extracted into `work/xpdf-4.03`.

Tip: It is a good idea to run the `patch` target as well. The reason is that we want all the local FreeBSD ports patches applied to the freshly extracted, unmodified source code.

```
make patch
```

Alright, we've got all the basics covered. Let's start code spelunking. Are there any icons in the Xpdf sources?

```
$ find work/ -name *icon*
work/xpdf-4.03/xpdf-qt/indicator-icon-err5.svg
work/xpdf-4.03/xpdf-qt/indicator-icon-err2.svg
work/xpdf-4.03/xpdf-qt/indicator-icon1.svg
work/xpdf-4.03/xpdf-qt/indicator-icon6.svg
work/xpdf-4.03/xpdf-qt/icons.qrc
work/xpdf-4.03/xpdf-qt/xpdf-icon.svg
work/xpdf-4.03/xpdf-qt/indicator-icon7.svg
work/xpdf-4.03/xpdf-qt/indicator-icon0.svg
work/xpdf-4.03/xpdf-qt/indicator-icon-err3.svg
work/xpdf-4.03/xpdf-qt/indicator-icon-err4.svg
work/xpdf-4.03/xpdf-qt/indicator-icon3.svg
```

```
work/xpdf-4.03/xpdf-qt/indicator-icon4.svg
work/xpdf-4.03/xpdf-qt/indicator-icon-err7.svg
work/xpdf-4.03/xpdf-qt/indicator-icon-err0.svg
work/xpdf-4.03/xpdf-qt/indicator-icon-err1.svg
work/xpdf-4.03/xpdf-qt/indicator-icon-err6.svg
work/xpdf-4.03/xpdf-qt/xpdf-icon.ico
work/xpdf-4.03/xpdf-qt/indicator-icon5.svg
work/xpdf-4.03/xpdf-qt/indicator-icon2.svg
```

Excellent! `xpdf-icon.ico` and `xpdf-icon.svg` look promising. These are the files we need to get installed into `/usr/local/share/icons`. In order to do that we need edit the port's `Makefile` and expand the install targets. This port already has a `post-install` target so let's add four more lines to it. We will use `${INSTALL_DATA}` to install icon files, and `${MKDIR}` to create directories. These variables are examples of numerous wrapper variables defined in the FreeBSD Ports framework. If you'd like to know more about those variable take a look at the output of, e.g., `make -V INSTALL_DATA`. The patch should look like this so far:

```
diff --git a/graphics/xpdf4/Makefile b/graphics/xpdf4/Makefile
index bd81dd1a16be..36bd84d97e7e 100644
--- a/graphics/xpdf4/Makefile
+++ b/graphics/xpdf4/Makefile
@@ -70,5 +71,9 @@ post-install:
 		${INSTALL_DATA} ${WRKDIR}/xpdf-man.conf \
 			${STAGEDIR}${PREFIX}/etc/man.d/xpdf.conf
 		${INSTALL_DATA} ${FILESDIR}/xpdf.desktop ${STAGEDIR}${DESKTOPDIR}
+		${MKDIR} ${STAGEDIR}${PREFIX}/share/icons/hicolor/256x256
+		${INSTALL_DATA} ${WRKSRC}/xpdf-qt/xpdf-icon.ico
${STAGEDIR}${PREFIX}/share/icons/hicolor/256x256/xpdf.png
+		${MKDIR} ${STAGEDIR}${PREFIX}/share/icons/hicolor/scalable
+		${INSTALL_DATA} ${WRKSRC}/xpdf-qt/xpdf-icon.svg
${STAGEDIR}${PREFIX}/share/icons/hicolor/scalable/xpdf.svg

 .include <bsd.port.mk>
```

This is nice. Since we are installing two new files now, we need to add them to the packing list (`pkg-plist`). The list can be regenerated with `make makeplist`, but we'll do it by hand this time. Here's the patch:

```
diff --git a/graphics/xpdf4/pkg-plist b/graphics/xpdf4/pkg-plist
index e6cd3e15dd75..7eee2ae85bc6 100644
--- a/graphics/xpdf4/pkg-plist
+++ b/graphics/xpdf4/pkg-plist
@@ -10,6 +10,8 @@ libexec/xpdf/pdftotext
 %%GUI%%libexec/xpdf/xpdf
 %%GUI%%bin/xpdf
 %%GUI%%%%DESKTOPDIR%%/xpdf.desktop
+%%GUI%%share/icons/hicolor/256x256/xpdf.png
+%%GUI%%share/icons/hicolor/scalable/xpdf.svg
```

```
etc/man.d/xpdf.conf
%%DATADIR%%/man/man1/pdfdetach.1.gz
%%DATADIR%%/man/man1/pdffonts.1.gz
```

Paths in that list are relative to `${PREFIX}` (`/usr/local` by default). `%%GUI%%` at the beginning of the line means that those files are only going to be installed if this port is built with the `GUI` option enabled (apparently, some people like to have their Xpdf software headless).

The last bit we need to take care of is to bump the port's revision number. Once the change lands in the ports tree, port builders must know to rebuilt the package with our modifications. The easiest way to bump the revision is to use `portedit` (it's a part of the `portfmt` package):

```
$ portedit bump-revision -i Makefile
```

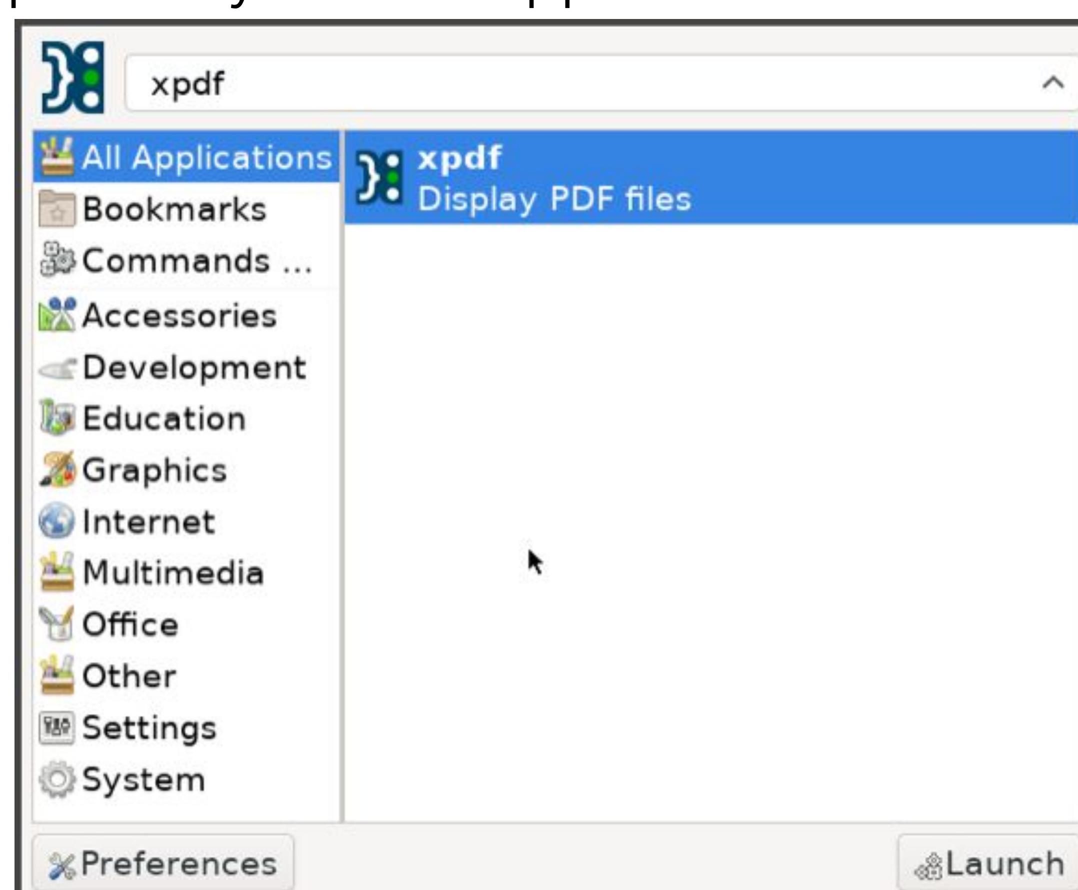
As a result we should see the following in the diff:

```
diff --git a/graphics/xpdf4/Makefile b/graphics/xpdf4/Makefile
index bd81dd1a16be..36bd84d97e7e 100644
--- a/graphics/xpdf4/Makefile
+++ b/graphics/xpdf4/Makefile
@@ -1,5 +1,6 @@
 PORTNAME=      xpdf
 PORTVERSION=   4.03
+PORTREVISION=  1
 PORTEPOCH=     1
 CATEGORIES=    graphics print
 MASTER_SITES=  https://dl.xpdfreader.com/
```

Great! Now let's test our changes. For that we need to build and reinstall Xpdf. The following command is enough. You may want to run `make missing` first and install the dependencies with pkg(8) to save time.

```
make reinstall
```

Time to check if the Xpdf entry in Xfce Application Finder has an icon now.



Success!

We need to test the patch a bit more before we submit it for review.

1. Does Xpdf still work? (Launch the newly reinstalled Xpdf and see if everything™ is in order.)
2. Does our patch work as exptected? (We've seen the icon in Xfce Application Finder so the answer is yes.)
3. Can you build Xpdf in Poudriere? (Hmm?)

Poudriere setup is well explained in the "Testing the Port" chapter of the FreeBSD Porter's Handbook: https://docs.freebsd.org/en/books/porters-handbook/testing/.

Submitting the Patch

FreeBSD Bugzilla is the service where contributors upload patches with suggested changes: https://bugs.freebsd.org. The whole process is fairly straightforward. First, you create an account and log in. Then you open a new problem report (PR) by clicking "New" in the navigation bar at the top. Remember to prefix the summary with "`graphics/xpdf4`". This way the port's maintainer will get notified about the PR (some other tips on how to write a good PR are written down here: https://wiki.freebsd.org/Bugzilla/DosAndDonts). Sometimes, in addition to opening a PR on Bugzilla, people submit their patches to Phabricator. This other service has a nicer interface for code reviews.

Oh, BTW, the problem of missing Xpdf icons is based on a true story. I've reported the issue with Xpdf on Bugzilla (https://bugs.freebsd.org/bugzilla/show_bug.cgi?id=261376) and posted my patch to Phabricator (https://reviews.freebsd.org/D33984). The Xpdf maintainer reviewed my patch and gave the green light to commit the change. Since I am a ports committer, I committed the change myself.

Getting help

One day you will embark on your own journey of contributing to the FreeBSD ports tree. Many feel overwhelmed and terrified when facing the challenge. Fear not! The FreeBSD community is always ready to help you out. IRC channels, mailing lists, forums, and most recently Discord are all great venues to talk to other FreeBSD folks and ask questions.

Most importantly, have fun hacking on FreeBSD and enjoy your time among the FreeBSD folks. See you at the zoo!

MATEUSZ PIOTROWSKI is a FreeBSD ports and documentation committer based in Berlin. He enjoys troubleshooting bugs, scripting automation, and designing robust software systems (always thoroughly documenting everything along the way). Recently, his interests have drifted toward tracing and performance engineering. When he is not hacking on the supposedly deterministic circuitry of modern software, he is exploring the ever-changing dynamics within society and culture.

Contributing to FreeBSD Ports with Git

BY JOSEPH MINGRONE

The FreeBSD ports tree was created in 1994 and tracked using CVS until July 15, 2012 when Subversion took over. A second repository conversion occurred on April 6, 2021 when the *source of truth* was migrated from Subversion to Git. As both CVS and Subversion are centralized version control systems, the required workflow changes associated with the first conversion were not as complex as with the conversion to Git, a distributed version control system.

This is not a comprehensive guide to using Git. The goal of this article is to guide those new to either Git or FreeBSD ports through *a* Git workflow that can be used to contribute to FreeBSD ports. Topics covered include:

- a brief overview of important Git concepts
- staying up to date with remote repositories
- working with branches
- committing
- modifying history
- working with Phabricator reviews
- testing changes with poudriere
- keeping track of upstream releases.

For a thorough introduction, refer to the Pro Git book and the Git Primer Chapter of the FreeBSD Committer's Guide. Also not covered is how to work with the `make` specifications that ports and the ports infrastructure are written in. This is covered in detail in the FreeBSD Porter's Handbook.

What makes Git fundamentally different from centralized version control systems like Subversion is its support for distributed workflows. Git does not require a central server that contains *blessed* copies of the versioned files because 1. copies of the repository are full clones that include meta-data and full history and 2. Git commits are snapshots of the repository rather than delta-based changes to files. Snapshots are described using hash algorithms that take as input the state of the repository and produce a deterministic hash value in the form of a hexadecimal string. If two copies of the repository are in the same state, the hash values describing the cop-

ies will be the same, whereas two repositories that differ by a single bit flip will produce different hash values. The history of a Git repository is a collection of these snapshots joined together so that each commit points to its parent commit(s).

A Git workflow may include 1. creating a local branch to develop a new feature, 2. merging the work in the feature branch with the main branch, and 3. pushing the changes to another Git repository. For a FreeBSD ports contributor, a new feature might mean creating or updating a port, or even something as simple as fixing a typo. When work in the feature branch is ready, it can be reviewed and merged with the official FreeBSD ports repository. Git branches are well suited for keeping the development of new features organized and isolated and their creation is very lightweight, as it simply involves creating a new pointer to a snapshot.

Because much of the work with Git occurs locally, there is no single workflow that all contributors must subscribe to. Work the way that best suits you. The official FreeBSD ports repository does enforce certain conventions though. For example, we require a simple, linear history of commits, so that the history of the main branch under Git looks similar to how it looked under Subversion. To do this, certain constraints are required, which we will discuss. Other projects use different workflows that results in parallel paths in the main branch of the repository. In short, Git is flexible and there is no single workflow that suits all people or projects. Indeed, as of early 2022 there is a FreeBSD working group exploring how we can optimize the way we work with Git, so refinements may be forthcoming. With these caveats out of the way, let's explore a Git workflow that is suitable for contributing to the FreeBSD ports tree.

Installing Git

The simplest way to get Git installed on your FreeBSD system is to use the official FreeBSD package.

```
pkg install git
```

For more information about installing third-party software on FreeBSD, refer to the FreeBSD Handbook Chapter on installing applications.

Cloning the Ports Tree

If you would like to contribute a new port to the tree, but do not already have something in mind, you can start by scanning the list of requested ports on the FreeBSD Wiki. Suppose we wish to create a new port for an application currently on the list, the Nyxt browser. The first step is to clone the FreeBSD ports repository. If you are using ZFS, you may wish to create a dedicated dataset for your development ports tree.

```
zfs create zroot/usr/home/ashish/freebsd/ports
```

Of course, substitute `zroot/usr/home/ashish/freebsd/ports` for your dataset layout. Now clone the repository. You are downloading the entire repository, which includes over 40,000 ports and a 28-year history, so this will take some time.

```
git clone -o freebsd --config remote.freebsd.fetch='+refs/notes/*:refs/notes/*'
https://git.freebsd.org/ports.git ~/freebsd/ports
```

The `-o freebsd` sets the name for the default remote repository for collaboration (pulling and pushing changes). The `--config remote.freebsd.fetch='+refs/notes/*:refs/notes/*.` adds Subversion revision numbers to the notes field of commits that occurred prior to the conversion to Git. When the clone is finished, you can optionally create a child ZFS dataset where software distribution files will be stored when building ports.

```
zfs create zroot/usr/home/ashish/ports/distfiles
```

Unlike the ports themselves, which are mostly text files, the software distribution files are usually already compressed, so zfs compression can be turned off for the `zroot/usr/home/ashish/freebsd/ports/distfiles` dataset.

```
zfs set compression=off zroot/usr/home/ashish/freebsd/ports/distfiles
```

You have a few options for telling `make(1)` about the location of your ports tree. The first option is to add configuration to `/etc/make.conf`.

```
.if ${.CURDIR:M/usr/home/ashish/freebsd/ports/*}
PORTSDIR=/usr/home/ashish/freebsd/ports
.endif
```

An alternative method is to set the PORTSDIR environment variable. For example, if your shell is `zsh`, you can add the line below to `~/.zshrc`.

```
export PORTSDIR=/usr/home/ashish/freebsd/ports
```

If you plan on working with multiple ports trees, a tool like `sysutils/direnv` is useful for loading or unloading environment variables depending on the current directory.

Staying Up to Date

The ports tree is actively developed, so changes will be pushed frequently to git.freebsd.org/ports.git. To fetch the changes that occurred in the upstream FreeBSD repository, use

```
git -C ~/freebsd/ports fetch freebsd
```

Fetching gives you an opportunity to inspect what changes have been made before integrating those changes into a local branch. Here `-C ~/freebsd/ports` instructs Git to operate on the repository under `~/freebsd/ports`. If the current working directory is `~/freebsd/ports`, which from this point on is assumed, this flag can be omitted. The `freebsd` argument means fetch from that remote repository.

To list the commits that were pushed to `freebsd`'s main branch that are not part of the local main branch, run

```
git log --oneline main..freebsd/main
```

Beside the topmost hash, you will see two pointers, `freebsd/main` and `freebsd/HEAD`. `HEAD` is normally a pointer to the last commit in the branch and in this case, like `freebsd/main`, it points to the last commit in the main branch of the remote repository. If we run

```
git log --oneline freebsd/main
```

and continue down the list of commits, we will eventually see `HEAD` and `main` which both point to the last commit on the local main branch. To integrate the new commits from `freebsd/main` into our local main branch, run

```
git merge freebsd/main --ff-only
```

The `--ff-only` (fast-forward only) option means only integrate the work from `freebsd/main` into `main` if it can be done by moving the `main` branch pointer to point to the same commit as `freebsd/main`. This can only happen when the commits listed in the output of

```
git log --oneline main..freebsd/main
```

descend from the local main branch. If changes have been made to the local main branch that are not part of `freebsd/main, --ff-only` will cause the `merge` to fail. In the workflow described here, we will never make direct changes to the local main branch, so this should never be a problem, but to be safe, we can configure the merge command to always use `--ff-only` with

```
git config merge.ff only
```

As a convenience, there is a `pull` command that will do both the `fetch` and `merge`. Depending on the circumstances, using `pull` may not be wise, because you do not get the opportunity to inspect what will be integrated into your local branch. If the commits in the main branch of your ports repository are always a subset of the commits in `freebsd/main` (as recommended here), this is less of a concern. To reduce the chances of diverging from `freebsd/main` when using `git pull`, we can configure the command to only do fast-forward merges as well with

```
git config pull.ff only
```

Creating a Local Branch

Now that we can keep our repository copy up-to-date with `git.freebsd.org/ports.git`, let's *create* changes. This is where Git really shines with the use of local branches, which provide a clean and efficient way to keep work-in-progress organized. Start by creating a new feature branch to work on the new nyxt port.

```
git branch nyxt
```

Now switch to the nyxt branch using

```
git checkout nyxt
```

A shorthand for both creating and switching to a branch is

```
git checkout -b nyxt
```

To check which branch you have checked out, you can run

```
git branch --show-current
```

You may find it useful to display the current branch in your shell prompt. If your shell is zsh, you can use `shells/git-prompt.zsh` from the ports tree. A nice feature of `git-prompt-zsh` is that it updates the prompt asynchronously, so when `git status` or some other Git operation is taking time to complete, it doesn't block other work. If this appeals to you and you use a shell other than zsh, there are similar code snippets to get Git status information in your prompt if your shell is `bash`, `fish`, or `tcsh`.

First Commit

After you have hacked on your new port, it is time to commit your changes. First, let's take a look at the status of the working tree with

```
git status
```

Depending on what work you did, this may tell you that the file `www/Makefile` was modified when you added `SUBDIR += nyxt` and you should also see `www/nyxt` as untracked. When interacting with the filesystem under the repository by adding, editing, or removing files, you are interacting with Git's working tree. Before you can commit changes to the repository, you have to stage which changes will be included in the next snapshot. In Git terminology, you add files from your working tree to the index. This extra step is useful, because it gives you precise control over what goes into a commit. To add all the changes to the index, run

```
git add www/Makefile www/nyxt
```

Now `git status` will list all the modified or added files as staged and ready to be committed. Before we commit though, there are a few more one-time tasks to complete. Git has a hook feature, which is a way to execute custom scripts when certain events like committing or merging occur. To configure Git to search the location where ports-specific hooks are stored in the ports repository, with the current working directory anywhere under the repository, run

```
git config --add core.hooksPath .hooks
```

That directory contains the `prepare-commit-msg hook`, which provides a helpful template for formatting commit messages. We also want to configure the editor that will be launched to create commit messages. Git chooses the editor to launch in this order: the value of the `GIT_EDITOR` environment variable, its `core.editor` configuration variable, the `VISUAL` environment

variable, and the `EDITOR` environment variable. For example, we can tell Git to use terminal Emacs to edit commit messages with

```
git config core.editor "emacs -nw"
```

If you would like to use this editor for all your Git repositories, add the `--global option` when setting `core.editor`.

```
git config --global core.editor "emacs -nw"
```

To commit your changes run

```
git commit
```

Your editor should now be displaying the commit template, which provides tips for creating a commit message. The subject line should take the form `<part of the ports tree that is changing>: <brief overview of the change>` and ideally be under 50 characters. A good subject line might be `www/nyxt: (WIP) First attempt to port Nyxt browser`. After a blank line, the body of the commit message provides more detail. An example might be

```
Makefile is still a skeleton.
TODO:
- Add _DEPENDS
- Add license information
- Fix QL_DEPS
- Add do-build target
```

After saving and exiting the editor your changes will be committed. So far, our changes progressed from the working tree, to the staging area (index), and finally to the local repository. To inspect your commit, use `git log`, which will also confirm that the `HEAD` and `nyxt` pointers have advanced one commit ahead of the main branch pointer.

Rewriting Local History

Whereas committing with Subversion meant sending your changes to the server, committing in Git simply means recording your changes locally in a new snapshot. Thus, with Git, it is wise to commit often. When it is time to share your work with others, you can refine your local history. There are a few different ways to rewrite history. For example, if you see a typo in your latest commit message, this is a good time to fix it, since your changes are still local. To modify the most recent commit, run

```
git commit --amend
```

and amend the commit message in your editor. If you accidentally did not stage and commit your changes to `www/Makefile` in the last commit, simply stage that file before running `git commit --amend` and it will be added to the last commit. Methods for rewriting history beyond the most recent commit will be discussed later.

Testing

Before requesting a review, your new port must be tested. There are two *port linters* that can alert you about common violations. Install them with

```
pkg install portlint portfmt
```

To lint your port with portlint, from `~/freebsd/ports/www/nyxt`, run

```
portlint -AC
```

To lint your port with portclippy from the portfmt package, also from `~/freebsd/ports/www/nyxt`, run

```
portclippy Makefile
```

Be aware, while these tools are generally quite helpful, they do not catch all mistakes and they can occasionally make ill-advised suggestions. Another useful tool is `portfmt`. As the name suggests, it can help with formatting your port's Makefile.

```
portfmt -D Makefile
```

Testing with Poudriere

Section 3.4 of the Porter's Handbook describes steps to test your port. It also refers readers to Chapter 10, which includes a guide for setting up poudriere, FreeBSD's bulk package builder and port tester. That section describes the merits of testing with poudriere. "[Various] tests are done automatically when running poudriere testport. It is highly recommended that every ports contributor install and test their ports with it." That Chapter of the Porter's Handbook describes a few different ways to set up a ports tree for poudriere. When you reach that section, it makes sense to tell poudriere to use our existing ports tree with

```
poudriere ports -c -m null -M ~/freebsd/ports
```

The `-m` option tells poudriere to use the null method, i.e., use an existing ports tree found at the location specified as the argument to `-M`. Using the null method means that we will manually manage the tree, including keeping it up-to-date and checking out the appropriate branch when testing. Once you have poudriere set up, you can test your port. If you created a jail named 13amd64, you can test the new port in that jail with

```
poudriere testport -j 13amd64 www/nyxt
```

Ideally you should test your port on the various tier 1 platforms (currently 12i386, 12amd64, 13amd64, and 13arm64). To test your new port after building it, poudriere can build a package and leave the jail running with the package installed.

```
poudriere bulk -i -j 13amd64 <category>/<port>
```

It's `-i` that instructs poudriere to leave the jail running with the package installed. This is useful for testing terminal applications, but not graphical applications like nyxt.

If the port has OPTIONS, poudriere will test and build the package as the official package builder will, i.e., with the default OPTIONS chosen. If you want to test or build the package with non-default options, you can run

```
poudriere options -j 13amd64 www/nyxt
```

before `poudriere testport`... or `poudriere bulk`....

Poudriere also creates a repository that pkg can use to install packages. If you want to install the package on the same system as poudriere, you have to configure pkg to use it. From [PKG.CONF(5)](), a local configuration can be placed under *usr/local/etc/pkg/repos*. The name of the file is not important, but it must have a `.conf suffix`. To set a local repository configuration and disable the default official repository configured in `/etc/pkg/FreeBSD.conf`, `create/usr/local/etc/pkg/repos/local.conf` with

```
FreeBSD: {
 enabled: no
}
Poudriere: {
 url: "file:///usr/local/poudriere/data/packages/13amd64-default"
}
```

The path given above assumes poudriere's default repository location, the repository based on the 13amd64 jail, and the default ports tree.

If you want to serve packages to remote hosts, you will need to configure a web server. Poudriere also has a web interface that can display information about current and past builds. If your webserver is nginx, you can configure it to host poudriere's interface and repository with a server entry like this in `nginx.conf`.

```
server {
    listen 80 accept_filter=httpready;
    listen 443 ssl;

    server_name pkg.example.org;

    root /usr/local/share/poudriere/html;

    ssl_certificate /usr/local/etc/dehydrated/certs/example.org/fullchain.pem;
    ssl_certificate_key /usr/local/etc/dehydrated/certs/example.org/privkey.pem;

    # If you use dehydrated as a Lets Encrypt client
    location /.well-known/acme-challenge {
```

```
    alias /usr/local/www/dehydrated;
    }

    location /data {
        alias /usr/local/poudriere/data/logs/bulk;

    # Allow caching dynamic files but ensure they get rechecked
    location ~* ^.+\.(log|txz|tbz|bz2|gz)$ {
        add_header Cache-Control "public, must-revalidate, proxy-revalidate";
    }

    # Don't log json requests as they come in frequently and ensure
    # caching works as expected
    location ~* ^.+\.(json)$ {
        add_header Cache-Control "public, must-revalidate, proxy-revalidate";
        access_log off;
        log_not_found off;
    }

    # Allow indexing only in log dirs
    location ~ /data/?.*/(logs|latest-per-pkg)/ {
        autoindex on;
    }

    break;
 }

 location /repo {
    alias /usr/local/poudriere/data/packages;
    autoindex on;
 }
}
```

If you want to display poudriere's package building logs in the browser, tell nginx about text files with a `.log` suffix by editing the `text/plain` line in Nginx's `mime.types` to contain

```
text/plain log txt;
```

After restarting nginx with `service nginx restart`, point your browser to `http://pkg.example.org` to see poudriere's web interface.

Rewriting History to Prepare for Review

Before sharing your work, the commit history should be well organized, including the commit logs and the number of commits. Suppose the history on your nyxt branch contains seven WIP (work in progress) commits.

```
% git log --oneline
061be9ca5d98 (HEAD -> nyxt) www/nyxt: (WIP) ready for testing
cddad2b5886b www/nyxt: (WIP) Add missing www/Makefile entry
e42f79383312 www/nyxt: (WIP) Add build and install targets
807099e08e33 www/nyxt: (WIP) Fix QL_DEPENDS
3cc5f266b434 www/nyxt: (WIP) Complete _DEPENDS
80d098cd8367 www/nyxt: (WIP) Add license information
9ec91c5fb244 www/nyxt: (WIP) First attempt to port Nyxt browser
9f77e9601564 (freebsd/main, freebsd/HEAD, main) net-im/toxic: upgrade to v0.11.2
```

The commits above the `freebsd/main, freebsd/HEAD`, and `main` pointers are those in your `nyxt` branch that you want to clean up.

```
git rebase -i main
```

will show a log of the commits in your local nyxt branch. The `-i` option means the rebase will be interactive. We specify the commit preceding the subset of commits we wish to modify. In this case it is easiest to specify that commit with the `main` pointer. We could have also used tilde syntax, i.e., `HEAD~7` which means seven commits before HEAD, but it's tedious to count the seven commits.

This is what you should see in your editor.

```
pick 9ec91c5fb244 www/nyxt: (WIP) First attempt to port Nyxt browser
pick 80d098cd8367 www/nyxt: (WIP) Add license information
pick 3cc5f266b434 www/nyxt: (WIP) Complete _DEPENDS
pick 807099e08e33 www/nyxt: (WIP) Fix QL_DEPENDS
pick e42f79383312 www/nyxt: (WIP) Add build and install targets
pick cddad2b5886b www/nyxt: (WIP) Add missing www/Makefile entry
pick 061be9ca5d98 www/nyxt: (WIP) Ready for testing

# Rebase 9f77e9601564..061be9ca5d98 onto 9f77e9601564 (7 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup [-C | -c] <commit> = like "squash" but keep only the previous
#                    commit's log message, unless -C is used, in which case
#                    keep only this commit's message; -c is same as -C but
#                    opens the editor
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
```

```
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
# .       specified); use -c <commit> to reword the commit message
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
```

The history is written so that older commits are at the top. The comments below list all the commands we can use. We instruct Git on to how to modify history by placing these commands next to the commits. The default command beside each commit is `pick`, i.e., keep the commit as is. Here, we want to squash these WIP commits into a single commit for review. To squash the six latest commits into the first commit, change the `pick` command to squash in these bottom six commits.

```
pick 9ec91c5fb244 www/nyxt: (WIP) First attempt to port Nyxt browser
squash 80d098cd8367 www/nyxt: (WIP) Add license information
squash 3cc5f266b434 www/nyxt: (WIP) Complete _DEPENDS
squash 807099e08e33 www/nyxt: (WIP) Fix QL_DEPENDS
squash e42f79383312 www/nyxt: (WIP) Add build and install targets
squash cddad2b5886b www/nyxt: (WIP) Add missing www/Makefile entry
squash 061be9ca5d98 www/nyxt: (WIP) Ready for testing
```

When you save and quit your editor, Git will complete the rebase, then show you the log messages in your editor, so that you can write a new log message for the new, single commit. Here is an example commit message that we might want to use when sharing our work with others for review.

```
www/nyxt: New port for the Nyxt browser

Nyxt is a keyboard-driven web browser designed for power users.
Inspired by Emacs and Vim, it has familiar key-bindings and is
infinitely extensible in Lisp.
WWW: https://nyxt.atlas.engineer/
```

Refer to the November 2020 Journal article for a deeper discussion on Writing Good FreeBSD Commit Messages. Now `git log --oneline` will show a single commit in our `nyxt` branch.

```
7392483f6147 (HEAD -> nyxt) www/nyxt: New port for the Nyxt browser
9f77e9601564 (freebsd/main, freebsd/HEAD, main) net-im/toxic: upgrade to v0.11.2
```

Another way we will want to rewrite the history is by rebasing our work in the **nyxt** branch on top of an up-to-date main branch. First update the main branch.

```
git checkout main
git pull
```

Then switch back to the **nyxt** branch and tell Git to do the rebase.

```
git checkout nyxt
git rebase main
```

If all goes well, **git log** will show your commits in the **nyxt** branch descending from the latest commits from the main branch. If conflicting changes were made in **freebsd/main** and your **nyxt** branch, Git will inform you which files have conflicts and give you the opportunity to manually resolve them.

```
~/freebsd/ports [nyxt|✔] % git rebase main
Auto-merging www/Makefile
CONFLICT (content): Merge conflict in www/Makefile
error: could not apply 531d9081dfb1... Add new entry for nyxt browser
hint: Resolve all conflicts manually, mark them as resolved with
hint: "git add/rm <conflicted_files>", then run "git rebase --continue".
hint: You can instead skip this commit: run "git rebase --skip".
hint: To abort and get back to the state before "git rebase", run "git rebase
--abort".
Could not apply 531d9081dfb1... Add new entry for nyxt browser
```

We can see the conflict is in **www/Makefile** and Git tells us what options we have to resolve the conflict manually. Here is an example of what we might see in **www/Makefile**

```
<<<<<<< HEAD
SUBDIR += nyan
||||||| parent of 531d9081dfb1 (Add new entry for nyxt browser)
=======
SUBDIR += nyxt
>>>>>>> 531d9081dfb1 (Add new entry for nyxt browser)
```

In this case, it is straightforward to manually fix the conflict. We want to add our entry for **nyxt** below the new entry for **nyan**. After editing the file so it looks like

```
SUBDIR += nyan
SUBDIR += nyxt
```

tell Git that we are ready to continue with

```
git add www/Makefile
git rebase --continue
```

Rebasing your feature branch onto an updated main branch is something you will do often enough that you may want to use a convenience script to do it in one step. Here is a simple example. Run `rum` from the feature branch to do the rebase in one step.

```
#!/bin/sh

# rum, r_ebase onto u_pdated m_ain
#
# Usage: rum
#
# globals expected in ${HOME}/.ports.conf with sample values
# No leading '/' on directory names means they are relative to $HOME
# portsd='/usr/home/ashish/ports' # ports directory

. "$HOME/.ports.conf"

usage () {
 cat <<EOF 1>&2
Usage: ${0##*/}
EOF
}
############################################ main
[ $# != 0 ] && { usage; exit 1; }

[ -n "${portsd##/*}" ] && portsd="${HOME}/$portsd"

# current branch
cb="$(git -C "$portsd" branch --show-current)"

if [ -z "$cb" ]; then
  printf "Could not determine the current branch.\\"
  exit 1
elif [ "$cb" = "main" ]; then
  printf "The main branch is checked out.\\n"
  exit 1
fi

git -C "$portsd" checkout main && \
  pull && \
  git -C "$portsd" checkout "$cb" && \
  git rebase main
```

Submitting Work for Review

Now we are ready to submit our work for review. FreeBSD currently has two ways to do this. Bugzilla is used for submitting bugs and Phabricator is used for reviewing source code changes. Both accept patches, but Phabricator has helpful features that are missing from Bugzilla, such as allowing reviewers to add comments specific to one or more lines of the patch. To cover both methods, let's create a review in Phabricator, then a new bug in Bugzilla that points to the Phabricator review.

FreeBSD Phabricator Reviews

To begin using FreeBSD's Phabricator instance for code review at https://reviews.freebsd.org, you must first create an account, then install the arcanist command line tool.

```
pkg install arcanist-php80
```

Set up `~/.arcrc` with the required certificates by running

```
arc install-certificate https://reviews.freebsd.org
```

and follow the instructions. Next, configure Arcanist to use https://reviews.freebsd.org as the default URI.

```
arc set-config default https://reviews.freebsd.org/
```

To submit your review, from the nyxt branch run

```
arc diff --create main
```

This will create a new review with all the commits in the nyxt branch. In this example, we squashed our commits into a single commit, so the revision will be created with that single commit. When your editor opens, you will have the opportunity to edit the fields that are part of the revision. The top line will be the subject of your commit log, `www/nyxt: New port for the Nyxt browser` and the summary will contain the rest of the commit log. Under test plan, you can list what you did to test the port. For example, if you did `poudriere testport` for each of the supported versions on the tier 1 architectures, you could write

```
poudriere testport 12/13 amd64/aarch64
```

You must also add at least one reviewer. If you have one or more ports committers that you have been working with, you can add their usernames here. For example

```
Reviewers: ashish rene
```

You can also specify group reviewers, which are of the form `#group_name` such as `#ports_committers`. The Subscribers: field, like `Reviewers:` takes a list of users, but these users do not reject or approve your work. When reviewers request changes, you can update the revision with

```
arc diff --update <revision>
```

where <revision> is the revision ID and takes the form DXXXXX. It can be found in the email sent to your address when you created the revision. For example, if your revision is found at https://reviews.freebsd.org/D33314, then use D33314 as <revision>.

Submitting Bugzilla Bug Reports

To create a new Bugzilla bug, point your browser to https://bugs.freebsd.org and click the `New` link at the top of the page. If you are not logged in to the FreeBSD Bugzilla instance, you will be prompted to do so. If you do not have a FreeBSD Bugzilla account, you can use the link on the login page to create a new one.

From here, you choose the `Ports & Packages` link since we are creating a new port and choose `Individual Port(s)` for the `Component`. For ports-specific bugs, the bug's subject line can be the commit subject prefixed with [NEW PORT], i.e., [NEW PORT] `www/nyxt: New port for the Nyxt browser`. If the port isn't new, the `category/port` prefix will automatically assign the bug to the maintainer of the port. In the description you can add the rest of the commit message and any other information helpful for others reading the bug. If you created a Phabricator review, add it to `See also`.

When your new port is accepted and pushed to `git.freebsd.org/ports.git`, your new job as the maintainer of the port begins. For an outline of the responsibilities of port maintainers, refer to the The challenge for port maintainers article. To keep up-to-date with upstream, portscout is a helpful service to alert you when there is a new release, so you can submit a port update. If upstream uses GitHub, you can also be alerted to new releases by following the `Watch` and `Custom` links, then check `Releases` on the project's page. When it's time to update your port and the changes are simple (e.g., only DISTVERSION/distinfo changes), submitting a Phabricator review may not be necessary. From a Git feature branch, you can create a patch using `git format-patch main` and attach it to a new Bugzilla bug. With Git, we now have more flexibility when crediting contributors for their work. When you submit a patch this way and a committer pushes it to `git.freebsd.org/ports.git`, `git log` will give you credit for your work. Even if you submit a traditional diff, committers have the option to set you as the author.

Opinionated Conclusions

Change can be hard. Many FreeBSD developers and contributors who dedicated significant time to becoming productive using Subversion were reluctant to change to a new version control system, especially one so fundamentally different. We lost some practical features like simple, monotonically increasing commit revisions and deterministic history retention when directories and files are moved within the repository. However, after three quarters of year, most indications suggest developers and the wider community are pleased and productive with the change. It is difficult to isolate the cause of certain outcomes, but the number of commits to the ports tree from the conversion date until the time of writing, 2021-04-06 to 2021-12-31 is 29,238. This is 1,748 more than the number for the same time last year. Let's hope this is a continuing trend in contributions to the ports tree.

JOE MINGRONE is a FreeBSD ports developer and works for the FreeBSD Foundation. He lives with his wife and two cats in Dartmouth, Nova Scotia, Canada.

CBSD Part 1—Production

BY OLEG GINZBURG

In 2012, I worked as an IT Systems Administrator for Nevosoft, a small game developer that had the entire server infrastructure developed on the base of the FreeBSD OS. At that time, no one knew about Kubernetes and Docker, but by virtue of the FreeBSD Jail, the company's servers benefitted by having all the components separated and each service used an independent Jail container. Nowadays, FreeBSD boasts a dozen (or even more) container orchestration programs, although in 2012, there was not a wide choice. ezjail was available, but it was a solution for a Standalone Server. Our installation had thirty to forty physical servers, each of which launched from ten to twenty containers. Except for the primitive create actions, deleting, launching, and cloning containers, the company's system administrators needed more advanced capabilities, such as container management on remote servers, container migration from one server to another, and the ability to save a container as a portable image. Those results were achieved by creating simple shell scripts. In 2013, however, the necessity of applying the proprietary software on the servers caused the company to begin the migration process to Linux. Because by this time, all created shell scripts were on a par with ezjail in the sense of their capabilities, and in certain instances, they added unique features (for example, there was initially the TUI—text-based user interface), a decision was made to combine the scripts collection and publish them in the FreeBSD Ports Tree under the same title. This was the beginning of the CBSD Project.

Comparison with Other Management Systems

Today, FreeBSD supports at least thirty titles of utilities for managing containers and virtual machines. There is bhyve on the FreeBSD platform and from there the list grows steadily longer. 2022 marks CBSD's decennary—it is kept current and continues to expand. Thus far, this is one of the oldest virtual environment management systems on the FreeBSD platform. The virtual environments are not only intended to mean containerizing based on jail, but also support for virtual machines based on bhyve, XEN and QEMU / NVMM hypervisors.

The development was based on the following concepts and philosophy:

- Focus both on single-mode installations and on those consisting of multiple hosts;
- Have the opportunity to integrate with other solutions;
- Be open for changes and think in big-picture terms: the bigger idea, the larger the potential; whereas small ones usually do not lend themselves to growth;
- Remain simple and flexible in-use.

"Simplicity is the ultimate sophistication" – Leonardo da Vinci

A wide range of tool kits have sprung up around CBSD: web interface, [message broker] service for delivering tasks to distributed CBSD nodes, API service, and thin client for working with it.

Finally, CBSD is a native FreeBSD product, not a Linux solution porting attempt.

Learn more about the project objective: https://www.bsdstore.ru/en/cbsd_goals_ssi.html

Learn more about the project development: https://www.bsdstore.ru/en/cbsd_history_ssi.html

Child Projects

CBSD is not only a product for the end user, but also one of the elements involved in complex solutions building that can save a lot of person-hours by delegating functions for creating and managing CBSD virtual environments. As follows, there are various self-contained projects and distributions for demonstration and gaining insights into the re-use of CBSD work:

- **Reggae** (developed by Goran Mekić) uses CBSD for DevOps tasks automation;
- The distribution kit https://k8s-bhyve.convectix.com/ (**k8s-bhyve**) shows the ability to quickly (from a few seconds to 1 minute) launch Kubernetes Clusters set up on a bhyve hypervisors;
- The distribution kit https://clonos.convectix.com/ has the ability to construct a WEB/UI interface over CBSD for creating containers and virtual machines bhyve;
- The distribution kit https://myb.convectix.com/ (**MyBee**) enables work with cloud images through CBSD without using the UI and CLI: it is possible to get virtual machines having sent one request to the CBSD API using an HTTP request (for example, through the use of the curl utility) or using the nubectl thin client (https://github.com/bitcoin-software/nubectl).

CBSD and Jail Container: Practical Application

Let's get better acquainted with available CBSD jail operation methods. The site has a description of the initial CBSD installation and customization process, and we have assumed that you already have the run-time version. There are various ways to set up containers:

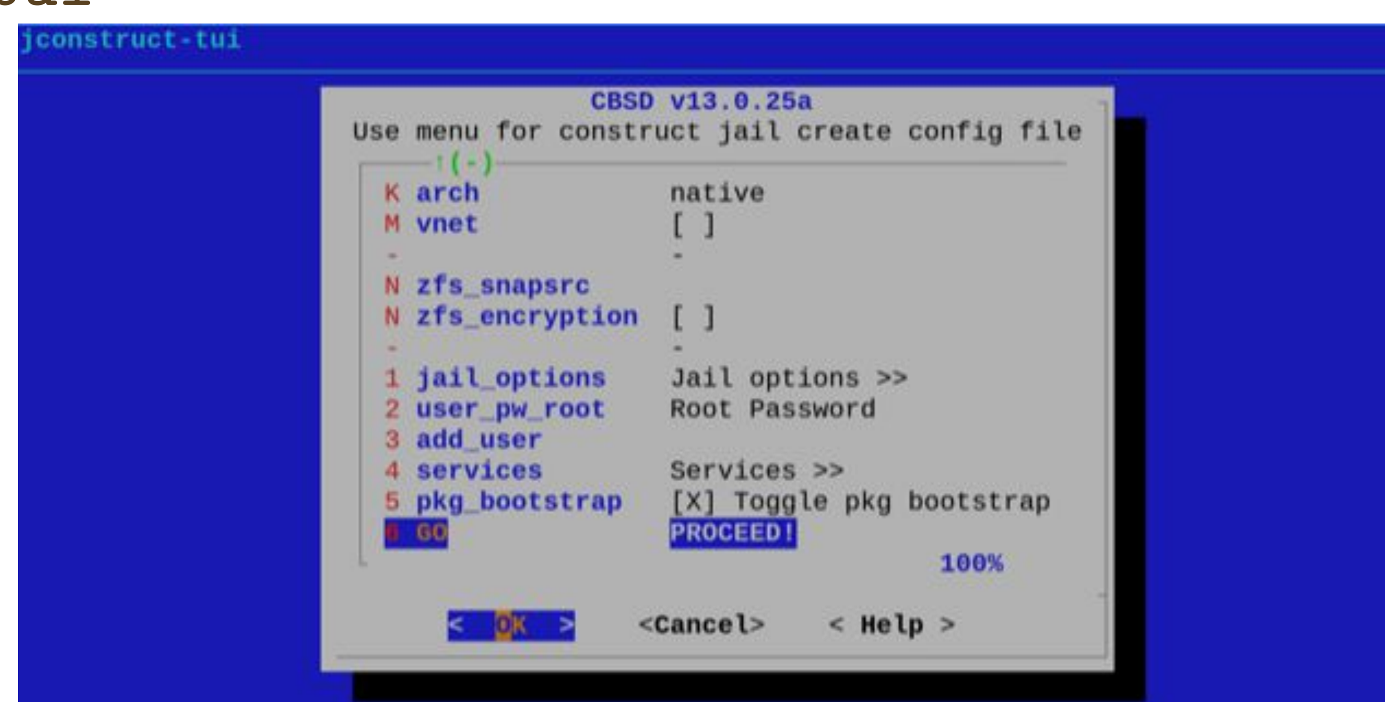
Option 1: in the command line dialog format. This method does not require learning a variety of possible jail parameters because the script will recall them:

```
cbsd jconstruct
```

```
root@cbsd:/ # cbsd jconstruct
---------[CBSD v.13.0.25a]---------
Welcome to jcreate config constructor script.

For DIALOG-based menu please use jconstruct-tui utility
-----------------------
Proceed to construct? [yes or no]
y
Jail name. Name must begin with a letter / a-z /  and not have any special symbols: -,.=% e.g
: jail2

Jail Fully Qualified Domain Name e.g: jail2.my.domain

Jail IPv4 and/or IPv6 address e.g: 172.16.0.17

Jail base source version e.g: native

Target architecture, i386/amd64 or qemu-users arch e.g: native

1,yes - Jail have personal copy of base system with write access, no NULLFS mount. 0,no - rea
d-only and NULLFS ( 0 - nullfs base mount in RO, 1 - no nullfs, RW ) e.g: 0

1,yes - Jail have shared /usr/src tree in read-only (0 - no, 1 - yes): e.g: 0
```

Option 2: in dialogue format through TUI. This option is also appropriate for beginners, as—in line with the first option—knowledge of possible arguments and commands is not required:

```
cbsd jconstruct-tui
```



The output of both dialogs is the text configuration file generation with the collection of parameters for the jcreate script, which you can call directly from the TUI interface or command line.

Option 3: Using a configuration file or command-line arguments.

Compared to the previous methods, this one is complicated in that it requires knowledge of the configurable parameters' names, but it is adequate for automation of environments development. You can synthesize a configuration file template as follows:

```
jname="myjail1"
ver=13.0
baserw=0
pkglist="misc/mc shells/bash"
astart=0
ip4_addr="DHCP"
...
```

Launch: `cbsd jcreate jconf=<path_to_file>`

Also, it is possible to specify the options as arguments without the configuration file:

```
cbsd jcreate jname=myjail1 ver=13.0 baserw=0 pkglist="misc/mc
shells/bash" astart=0 ip4_addr="DHCP"
```

The configuration file and arguments can be combined:

```
cbsd jcreate jconf=<path_to_file> ip4_addr="10.0.0.2" runasap=1
```

Option 4: Vagrant-like style: CBSDfile.

This is another notation for describing CBSD virtual environments allowing both: describe container settings and operate customization while creating (for example, copy files to the container file system and configure the service). Notwithstanding the fact that in one CBSD file you can describe an infinite number of environments, create and delete them by calling `cbsd up` and `cbsd destroy`, this form of notation is user friendly for a certain directory structures having one directory describing only one environment, for example: https://github.com/cbsd/cbsdfile-recipes/tree/master/jail

Also, the distinctive capability of environments created through the CBSD file format is not with local environments and through the CBSD API.

Jail Templates

We will not dwell upon the typical containers working operations, since they are described on the site. We'll go over some innovations of the CBSD project aimed to easily reference in jail—in particular, working with container templates. Templates are methods of the container description and configuration. As for the container, it can be exported to a portable image. A container can contain a standard runtime environment, but in most cases, they are used for services/application isolation and distribution through an image. There are advantages and disadvantages of this. Some of the advantages of the container approach are the speed of service deployment, no effect on the basic system environment, and the capability to commit the versions of dependencies with which your application is fully operational.

Referring to disadvantages of this approach, there is the issue of security, especially when using a container or image without a self-contained build script (template). This makes an operational software update to a container difficult (for example, to fix 0-day vulnerabilities) and it makes backdoors more likely to occur that are intentionally or accidentally left by image collectors. Considering the complexity of installing certain software, it is not unusual to find a container with a configured service hand-emplaced or with a lost assembly instruction.

Another disadvantage is the complexity of configuring services to the containers. Some images maintain the capability to configure a limited number of parameters through the environment variables, though not always sufficient. One of the examples is dynamic configurations, in the case of a need to add and configure multiple virtual hosts (vhost) for the WEB server and setup several databases in the DBMS. For such tasks, there is a specific software segment called configuration management. The best known products in this category are Ansible, Chef, Puppet, Rex, SaltStack. However, it is general practice that they are all optimized for work in classic environments. CBSD can combine the full power of configuration managers and a container-based approach to get containers with management services. There are two kinds of templates used in CBSD:

• static (classic), basically including install software only. An example of such a template for CBSD is the sambashare container.

Similar templates can be found in other projects: for instance, the nginx template through the example of Fockerfile (focker project):

```
base: freebsd-latest

steps:
  - run:
      - ASSUME_ALWAYS_YES=yes IGNORE_OSVERSION=yes pkg install nginx
```

Or here's an example of the RabbitMQ template for the BastilleBSD project, where the **CMD** file content is:

```
service rabbitmq restart
```

PKG file content:

```
rabbitmq
python27
```

These templates are of limited usefulness, as this is an alternate form of writing two lines in shell:

```
pkg install -y rabbitmq python27
service rabbitmq restart
```

There are more complex templates that cover the copies of the service configuration files and, in some cases, those followed by accompanied processing through complex structures from sed/awk while container configuration.

As a general principle:

a) This is an irreversible operation and designed for one-time operation: there's no way to roll back or change the configuration without re-creating the container;

b) such templates are highly demanding for their maintenance: even after a minor service update in the container, the source configuration file can be substantially changed;

c) there is no guarantee that the service will be operable, or, in the case of configurator error, that it will roll back to the previous run-time version;

d) it's a difficult line to draw between sed/awk operating and switching from it to use config management programs.

Here is the solution to the problem and the main CBSD project message: do not reinvent the wheel, but re-use someone else's work whenever you can. Many developers maintain configuration management modules, so we recommend that for reasons of saving time you use their work for delegating configuration with utilities that are native for this. With this objective in view, the forms script has been included in CBSD since 2016. The CBSD Forms functions are to get and save user parameters in the YAML format in the form appropriate for the configuration module—a peculiar kind of middleware. **Puppet** was chosen as the configuration management system, but any other framework can be used.

Preparing CBSD for Using Templates

To put things into perspective, let's create a jail1 container and apply the redis service template to it.

As it stands, it is expected that CBSD is installed and set up correctly: you can run a container and the cbsd dhcpd command gives a client the addresses that have access to the Internet (for example through NAT)--this is a necessary criterion for pkg operation and packages installation in the container.

1) Git must be installed for the CBSD plug-in installation from the public repository https://github.com/cbsd. Install git (~220 MB) or its light version - git-lite (~40 MB), if you have not yet done so:

```
pkg install -y git-lite
```

2) Create the system container cbsdpuppet1 from the profile included in the base distribution of CBSD. This is done once on each new CBSD host:

```
cbsd jcreate jname=cbsdpuppet1 jprofile=cbsdpuppet
```

The cbsdpuppet1 container should not be run, as it fulfils one function—it contains the puppet7 installed package used by CBSD to configure containers.

3) Install the CBSD puppet module—this is optional functionality and is not included in the basic distribution. The module contains the puppet modules checked by the CBSD project. This is done once on each new CBSD host:

```
cbsd module mode=install puppet
```

4) Install the module for the redis service configuration. This is done once on each new CBSD host:

```
cbsd module mode=install forms-redis
```

5) Create a container with any arbitrary name where we are about to get the redis service, for example: jail1:

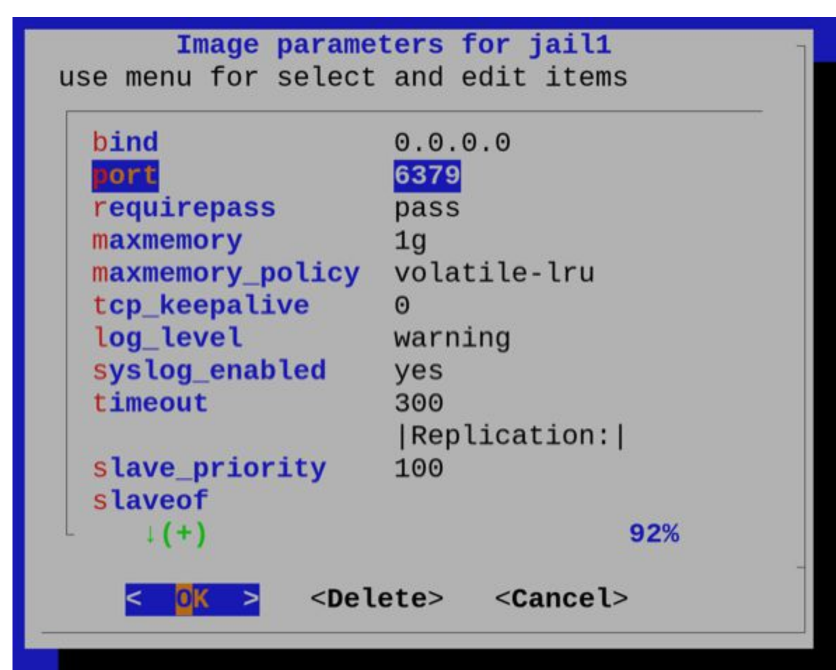
```
cbsd jcreate jname=jail1 runasap=1
```

(Note: the ***runasap=1*** *argument means the container will be run immediately).*

6) And the last step: pick up the redis service template to our jail1 container:

```
cbsd forms module=redis jname=jail1
```

There will be the familiar TUI interface with the parameters of the Redis module which can be configured at your discretion:



Let's retain the default arguments and pick them by the [COMMIT] operation. It may take a while before the script will finish running (depending on the Internet connection speed, since the module installs the redis server from the official repository `pkg.FreeBSD.org`) so you'll have to double check that the service in the container is installed and operating:

```
~ # cbsd jexec jname=jail1 sockstat -4l
USER     COMMAND    PID   FD PROTO  LOCAL ADDRESS         FOREIGN ADDRESS
redis    redis-serv 8910  7  tcp4   172.16.0.13:6379      *:*
```

By re-using the cbsd forms call, you can reconfigure the service at any time. Moreover, the forms hold the previous values and always output current data during initialization. For example, let's change an array of parameters:

- set the port to **7777**;
- set up a password to connect to redis server;
- set the maxmemory parameter to **4g**;
- set the maxmemory_policy parameter to **noeviction**.

Check on the state after the new parameter's application:

```
~ # cbsd jexec jname=jail1 sockstat -4l
USER     COMMAND    PID   FD PROTO  LOCAL ADDRESS         FOREIGN ADDRESS
redis    redis-serv 12587 7  tcp4   172.16.0.13:7777      *:*

~ # cbsd jexec jname=jail1 grep '^maxmemory' /usr/local/etc/redis.conf
maxmemory 4g
maxmemory-policy noeviction
```

Most Puppet modules commit to incorrect data validation, the configuration file validation, the service state. Therefore, the chance of getting a non-serviceable service due to invalid parameter input is considerably less than with statical templates.

Now that we've gotten to know the TUI interface, let's turn to automation. The cbsd forms permit acceptance of the parameters for a template through environment variables dispensing with interactive dialogs. In order to see what variables a particular template accepts, use the **vars** argument with indication of the desired module:

```
~ # cbsd forms module=redis vars
H_BIND H_PORT H_REQUIREPASS H_MAXMEMORY H_MAXMEMORY_POLICY
H_TCP_KEEPALIVE H_LOG_LEVEL H_SYSLOG_ENABLED H_TIMEOUT H_SLAVE_PRIORITY
H_SLAVEOF
```

Forms add the **H_** prefix to the names of regular parameter to cut the likelihood of potential conflicts with global variables of the system. Let's reconfigure the redis port the third time (set it to the value of **9999**), but not in an interactive mode (inter=0):

```
env H_PORT=9999 cbsd forms module=redis jname=jail1 inter=0
```

Give attention to the output of values in applying the template:

```
root@cbsd:/ # env H_PORT=9999 cbsd forms module=redis jname=jail1 inter=0
redis formfile for jail1: updated
puppet jname=jail1 module=redis mode=apply debug_form=0
applying puppet manifest for: redis
generate manifest and save to: /usr/jails/jails-system/jail1/puppet/manifest/redis.pp
generate hieradata (1) and save to: /usr/jails/jails-system/jail1/puppet/hieradata/redis.yaml
puppet: apply local...
/usr/sbin/sysrc -qf /usr/jails/jails-data/jail1-data/etc/rc.conf redis_enable="YES"
redis_enable: YES -> YES
Module path: /tmp/cbsdpuppet/modules/public
/tmp/puppet/bin/puppet apply /tmp/cbsd/puppet_root/manifest/init.pp --show_diff --hiera_confi
g=/tmp/cbsdpuppet/hiera.yaml --modulepath=/tmp/cbsdpuppet/modules/public --log_level=crit
***
JAIL1_REDIS_BIND="0.0.0.0"
JAIL1_REDIS_PORT="9999"
JAIL1_REDIS_REQUIREPASS="pass"
JAIL1_REDIS_MAXMEMORY="1g"
JAIL1_REDIS_MAXMEMORY_POLICY="volatile-lru"
JAIL1_REDIS_TCP_KEEPALIVE="0"
JAIL1_REDIS_LOG_LEVEL="warning"
JAIL1_REDIS_SYSLOG_ENABLED="yes"
JAIL1_REDIS_TIMEOUT="300"
JAIL1_REDIS_SLAVE_PRIORITY="100"
root@cbsd:/ #
```

Exported Variables

This is an informational output of the exported CBSD variables as a result of the module's work. The parameterization format can be set by mask through the configuration file `forms_export_vars.conf` in the directory `~cbsd/etc`.

Look into its default value:

https://github.com/cbsd/cbsd/blob/v13.0.18/etc/defaults/forms_export_vars.conf

These values can be automatically exported to a file or various Service Discovery services such as Consul. In this case, your cluster gets these values automatically, which can be used to build a **SOA** (Service Oriented Architecture), but that's a theme for another time.

Along with redis, here are other templates for services configuration: repositories named modules-forms-XXXX. For example, try to use these templates:

- modules-forms-memcached
- modules-forms-mysql
- modules-forms-grafana
- modules-forms-rabbitmq
- modules-forms-postgresql
- modules-forms-elasticsearch

Note: regardless of the conventional 1 service-1 container concept, in regard to the CBSD forms you can apply multiple templates to the same environment, having gotten all services in it at once.

How It Works

The Puppet modules (or any other similar system) contain the parameters that in 99% of cases have the value parameter form and are usually parameterized in YAML format. For user convenience, CBSD offers a TUI interface for working with basic forms, where, in a similar manner to HTML forms, the following elements can exist:

- radio button (the boolean type's values are true/false, yes/no);
- checkbox elements;
- dropdown menu of known certain values;
- the custom inputbox for relative user input;

To hold parameters in a universal form, CBSD forms uses the SQLite3 database—a describing form for the parameters input which stores the input value. The recording (writing) format is universal and serves for both TUI dialogs autogeneration (CBSD forms) and WEB/HTML forms (ClonOS), as the following examples clearly show. Let's create a SQLite3 database using the format required by CBSD forms:

```
sqlite3 /tmp/myforms.sqlite <<EOF
BEGIN TRANSACTION;
CREATE TABLE forms (  idx INTEGER PRIMARY KEY AUTOINCREMENT, mytable
VARCHAR(255) DEFAULT '', group_id INTEGER DEFAULT 1, order_id INTEGER
DEFAULT 1, param TEXT DEFAULT NULL, desc TEXT DEFAULT NULL, def TEXT
DEFAULT NULL, cur TEXT DEFAULT NULL, new TEXT DEFAULT NULL, mandatory
INTEGER DEFAULT 0, attr TEXT DEFAULT NULL, xattr TEXT DEFAULT NULL, type
VARCHAR(255) DEFAULT 'inputbox', link VARCHAR(255) DEFAULT '', groupname
VARCHAR(128) DEFAULT ''  );

INSERT INTO forms (
mytable,group_id,order_id,param,desc,def,cur,new,mandatory,attr,type,link,
groupname ) VALUES ( "forms", 1,0,"-","BSDMag poll: favorite BSD stuff
on",'-','','',1, "maxlen=128", "delimer", "", "" );
INSERT INTO forms (
mytable,group_id,order_id,param,desc,def,cur,new,mandatory,attr,type,link,
groupname ) VALUES ( "forms", 1,1,"FreeBSD","enter favorite FreeBSD tools,
e.g: sysrc",'bsdinstall','','',1, "maxlen=60", "inputbox", "", "" );
INSERT INTO forms (
mytable,group_id,order_id,param,desc,def,cur,new,mandatory,attr,type,link,
groupname ) VALUES ( "forms", 1,2,"DragonFlyBSD","enter favorite DFLY
tools. e.g: checkpoint",'checkpoint','','',1, "maxlen=60", "inputbox",
 "", "" );

CREATE TABLE system (  helpername TEXT DEFAULT NULL, helperdesc TEXT
DEFAULT NULL, version INTEGER DEFAULT 0, packages INTEGER DEFAULT 0,
have_restart INTEGER DEFAULT 0, longdesc TEXT DEFAULT NULL, title TEXT
DEFAULT ''  );
INSERT INTO system VALUES('bsdmag',NULL,201607,'','','','');
COMMIT;
EOF
```

And let's talk more about some of these lines. For us, the most important table columns are:

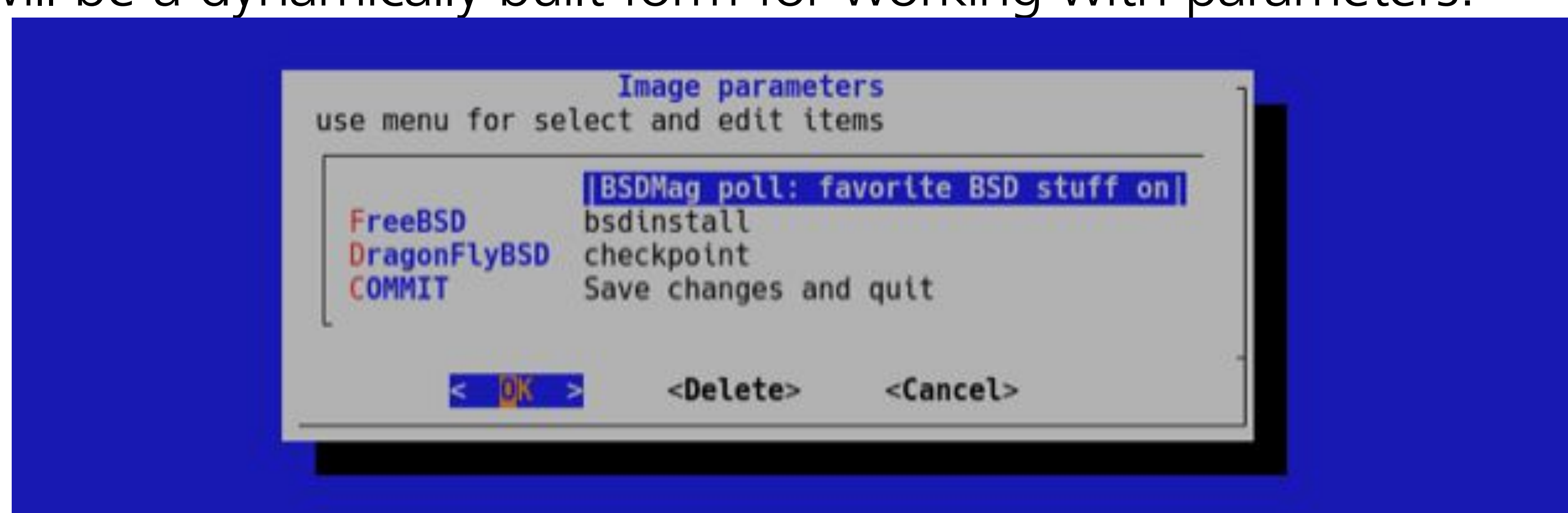
* **param** – the parameter-name directly, the value of which we want to get;
* **desc** – its relevant description, where appropriate;
* **def** – the default value;
* **new** – new value inputted by the user;
* **type** – type of a field: inputbox, delimiter, password, group_add, group_del, radio, check-box.

Two parameters can be added in the next two lines, and the values we want to get are: FreeBSD and DragonFlyBSD, each having its own default value and both have the inputbox field format, so users are free to enter an arbitrary string value.

Run the cbsd forms script on this form:

```
cbsd forms formfile=/tmp/myforms.sqlite
```

The output will be a dynamically built form for working with parameters:



The form structure for the redis service is more complex as it contains the field elements of boolean and dropdown types. Let us look at the table:

idx	mytable	group_id	order_id	param	desc	def	cur	new	mandatory	attr	xattr	type	link	group
1	forms	1	1	bind	Bind: default is 0.0.0.0	0.0.0.0			1	maxlen=60	NULL	inputbox	bind_autocomplete	
2	forms	1	2	port	Port: default is 6379. 0 - not listen on a TCP socket	6379			1	maxlen=60	NULL	inputbox	port_autocomplete	
3	forms	1	3	requirepass	Requirepass: Require clients to issue AUTH <PASSWORD>				0	maxlen=30	NULL	password		
4	forms	1	4	maxmemory	MaxMemory: Don't use more memory than the specified amount of byte	1g			1	maxlen=128	NULL	inputbox		
5	forms	1	5	maxmemory_policy	maxmemory policy	1	1		1	maxlen=128	NULL	select	memory_policy_select	
6	forms	1	6	tcp_keepalive	tcp_keepalive	0			1	maxlen=128	NULL	inputbox		
7	forms	1	7	log_level	log_level; default is: warning	4			1	maxlen=128	NULL	radio	log_level	
8	forms	1	8	syslog_enabled	syslog_enabled	2	2		1	maxlen=128	NULL	radio	syslog_noyes	
9	forms	1	9	timeout	timeout	300			1	maxlen=128	NULL	inputbox		
10	forms	1	10	-	Replication:	-			1	maxlen=128	NULL	delimer		
11	forms	1	11	slave_priority	slave-priority	100			1	maxlen=128	NULL	inputbox		
12	forms	1	12	slaveof	slaveof: ip port				0	maxlen=128	NULL	inputbox		

We take as our example the table `memory_policy_select`, the `memory_policy parameter` variation table of the select type:

redis.sqlite → memory_policy_select

Browse | Structure | SQL | Search | Insert | Export | Import | Rename | Empty | Drop

Show : 30 row(s) starting from record # 0 as a Table

Showing rows 0 - 5, Total: 6 (Query took 0.0001 sec)
SELECT * FROM "memory_policy_select" LIMIT 0, 30

← T →	id	text	order_id
Edit Delete	1	volatile-lru	5
Edit Delete	2	allkeys-lru	4
Edit Delete	3	volatile-random	3
Edit Delete	4	allkeys-random	2
Edit Delete	5	volatile-ttl	1
Edit Delete	6	noeviction	0

Check All / Uncheck All *With Selected:* Edit Go

In the WEB interface, the auto-generated form (the screenshot was taken from ClonOS) looks like this:

Where the maxmemory policy parameters are in a specific dropdown element:

The TUI interface offers a similar choice:

Note: all CBSD TUI dialogs can be described by similar SQL-based forms which are currently used only for work with templates.

After getting and processing the parameters, after the cbsd forms, the cbsd puppet module intervenes in proceedings. Here's what happens next:

- the overlay file system mounting (having used nullfs) of the puppet7 package files from the cbsdpuppet1 system container to the `/tmp/XXX` directory to the destination container (jail1);
- YAML file presentation with parameters for the module to the destination container (jail1);
- fetching the puppet apply instruction to the configurable container (jail1) to apply the puppet manifest from `the /tmp/XXX directory`.

For example, this is what the nullfs mounting points look like when parameters are applied:

```
root@cbsd:/ # mount -t nullfs | grep cbsdpuppet
/usr/local/cbsd/modules/puppet.d on /usr/jails/jails/jail1/tmp/cbsdpuppet (nullfs, local, noa
time, read-only, nfsv4acls)
/usr/jails/jails-data/cbsdpuppet1-data/usr/local on /usr/jails/jails/jail1/tmp/puppet (nullfs
, local, noatime, read-only, nfsv4acls)
root@cbsd:/ #
```

Upon completion and reconfiguration of the service, temporary nullfs file systems are dismounted as superfluous. Therefore, if we look at the installed software in the jail1 container from the example above, there would not be the puppet7 package or the files it depends on.

```
~ # cbsd jexec jname=jail1 pkg info
pkg-1.17.4                      Package manager
redis-6.2.6                     Persistent key-value database
```

Stating the matter another way, with the use of the cbsdpuppet1 container, there is no need to install puppet7 and the necessary dependencies in each container--the configuration application is not dependent upon the presence (or absence) of any system software in a finite container.

Epilogue

- cbsd forms is one of the most powerful capabilities of the CBSD framework when working with jail-based containers, forming a bridge between CBSD and configuration managers. Accordingly, instead of the sed/awk scripts support and static templates, the CBSD project contributes FreeBSD support to the appropriate Puppet modules, if it is out of support: if a module can operate in a FreeBSD environment, you can use it via cbsd forms transparently. There are benefits for particular CBSD users and all the Puppet users on FreeBSD in general. If you use another configuration system, you can set the call of other systems by using the cbsd forms script.

The CBSD project sustains oscillation and distribution of ready-to-use images based on existing templates, which you can use through the CBSD repository using cbsd repo and cbsd images. See inline documentation and examples:

```
~ # cbsd repo --help
~ # cbsd images --help
```

In the next of the series of articles, we'll look at the virtual machine management capabilities of CBSD.

OLEG GINZBURG lives in Russia and works as a DevOps engineer in the X5 Retail group. He has been fond of computer science since the ZX Spectrum and is a Unix fan and FreeBSD enthusiast. He is a FreeBSD promoter and member of several projects: CBSD, ClonOS, MyB, K8S-bhyve, and AdvanceBSD group.

Porting OpenBSD's pf syncookie Code to FreeBSD's pf

BY KRISTOF PROVOST

The internet can be a lawless place, and our on-line services can be attacked in a variety of ways. The firewalls we run can be part of the way we protect our systems, but it turns out they can also be an avenue for attack themselves.

One way of attacking services is to exhaust system resources by pretending to open connections. This is commonly called a SYN flood, and it's a fairly insidious attack. Let's start by refreshing our memories about how TCP connections work. Opening a TCP connection is a three-step process:

1) SYN: the client sends a SYN packets to indicate it wishes to open a connection. It sets an initial sequence number in the SYN packet.
2) SYN+ACK: the server responds, acknowledging the client sequence number, and settings its own.
3) ACK: the client accepts that the connection is open, and acknowledges the server's sequence number.

When the server receives the initial SYN it will set up internal data structures to support the new connection. This takes CPU time and memory.

This means that malicious clients can generate SYN packets (which are small, and easy to generate), consuming much of the server's available memory and CPU resources. Even worse, it only requires the single SYN packet. There's no need for the attacker to receive the server's SYN+ACK reply. That means that the source IP address can be faked, making these attacks difficult to filter out. They can also be targeted at the service's main TCP port (e.g., 443 for a web server), making the attack indistinguishable from real client requests.

SYN cookies

In 1996 Daniel J. Bernstein and Eric Schenk came up with a method to resist such attacks, called SYN cookies. Simply put, SYN cookies ensure the server does not run out of memory by not allocating any memory for the new connection when we receive a SYN packet.

We still generate a SYN+ACK reply, but wait to create server-side state until the client has responded to our SYN+ACK with its own ACK. This ensures that the client really exists (i.e., the source IP address is not spoofed).

The obvious issue with this is that we still need the information we'd normally save when the SYN first arrives. This information includes things like Maximum Segment Size (MSS) and Window Scale (WSCALE). While these options are … well optional, they are important for TCP performance, and we don't want to refuse them.

Furthermore, we also need some way of ensuring that the acknowledgement number in the client's ACK matches the sequence number we used in the SYN+ACK message. If we didn't check this, malicious clients could simply send a SYN, wait a little while and send an ACK blind, that is, without having to actually receive the server's SYN+ACK using a random acknowledgement number.

So, how do we accomplish this? We do so by encoding all the options into the sequence number of the SYN+ACK packet.

In the pf implementation, this is done in `pf_syncookie_generate()`:

```
uint32_t
pf_syncookie_generate(struct mbuf *m, int off, struct pf_pdesc *pd,
    uint16_t mss)
{
        uint8_t                 i, wscale;
        uint32_t                iss, hash;
        union pf_syncookie      cookie;

        PF_RULES_RASSERT();

        cookie.cookie = 0;

        /* map MSS */
        for (i = nitems(pf_syncookie_msstab) - 1;
            pf_syncookie_msstab[i] > mss && i > 0; i--)
                /* nada */;
        cookie.flags.mss_idx = i;

        /* map WSCALE */
        wscale = pf_get_wscale(m, off, pd->hdr.tcp.th_off, pd->af);
        for (i = nitems(pf_syncookie_wstab) - 1;
            pf_syncookie_wstab[i] > wscale && i > 0; i--)
                /* nada */;
        cookie.flags.wscale_idx = i;
        cookie.flags.sack_ok = 0;       /* XXX */

        cookie.flags.oddeven = V_pf_syncookie_status.oddeven;
        hash = pf_syncookie_mac(pd, cookie, ntohl(pd->hdr.tcp.th_seq));

        /*
         * Put the flags into the hash and XOR them to get better ISS number
         * variance.  This doesn't enhance the cryptographic strength and is
         * done to prevent the 8 cookie bits from showing up directly on the
         * wire.
         */
        iss = hash & ~0xff;
        iss |= cookie.cookie ^ (hash >> 24);

        return (iss);
}
```

The eagle-eyed reader will note that for both MSS and WSCALE we don't actually encode the correct value, but instead find the closest match in a lookup table. This reduces the number of bits needed to encode the information, but still gets us a good approximation of the real value. Having a slightly smaller maximum segment size or window scale will cost us a little performance, but not significantly so. The values are chosen so that the most frequently used MSS or WSCALE values are represented, so for most clients there will be no performance loss at all.

This information is encoded into an authenticated hash. That is, to recreate the hash you need both the input information (MSS, WSCALE, ...) and a secret key. In other words: attackers cannot predict the result of the hash, and consequently cannot predict the sequence number the server will choose. That's handled by the `pf_syncookie_mac()` function:

```
uint32_t
pf_syncookie_mac(struct pf_pdesc *pd, union pf_syncookie cookie, uint32_t seq)
{
	SIPHASH_CTX	ctx;
	uint32_t	siphash[2];

	PF_RULES_RASSERT();
	MPASS(pd->proto == IPPROTO_TCP);

	SipHash24_Init(&ctx);
	SipHash_SetKey(&ctx, V_pf_syncookie_status.key[cookie.flags.oddeven]);

	switch (pd->af) {
	case AF_INET:
		SipHash_Update(&ctx, pd->src, sizeof(pd->src->v4));
		SipHash_Update(&ctx, pd->dst, sizeof(pd->dst->v4));
		break;
	case AF_INET6:
		SipHash_Update(&ctx, pd->src, sizeof(pd->src->v6));
		SipHash_Update(&ctx, pd->dst, sizeof(pd->dst->v6));
		break;
	default:
		panic("unknown address family");
	}

	SipHash_Update(&ctx, pd->sport, sizeof(*pd->sport));
	SipHash_Update(&ctx, pd->dport, sizeof(*pd->dport));
	SipHash_Update(&ctx, &seq, sizeof(seq));
	SipHash_Update(&ctx, &cookie, sizeof(cookie));
	SipHash_Final((uint8_t *)&siphash, &ctx);

	return (siphash[0] ^ siphash[1]);
}
```

With the resulting hash post-processed we have enough information to send the server's SYN+ACK response.

At this point pf processing stops. We do not create state, we do not perform any further examination of the packet. This also means that if the firewall protects a different host (i.e., it's running on a router between the client and server) the server will not even be aware that the client has attempted to initiate a new connection. We want that because it means the server is protected from SYN floods, without needing any code or configuration changes.

If the client never responds, nothing happens. The server has remembered nothing about this specific SYN message and has no memory allocated to it. If on the other hand the client does respond (i.e., is a legitimate client, at least for the purpose of this discussion), we must reconstruct the information we've not retained when we received the original SYN message.

Upon receiving a SYN+ACK message we first validate it in `pf_syncookie_validate()`:

```
uint8_t
pf_syncookie_validate(struct pf_pdesc *pd)
{
	uint32_t		hash, ack, seq;
	union pf_syncookie	cookie;

	MPASS(pd->proto == IPPROTO_TCP);
	PF_RULES_RASSERT();

	seq = ntohl(pd->hdr.tcp.th_seq) - 1;
	ack = ntohl(pd->hdr.tcp.th_ack) - 1;
	cookie.cookie = (ack & 0xff) ^ (ack >> 24);

	/* we don't know oddeven before setting the cookie (union) */
	if (atomic_load_64(&V_pf_status.syncookies_inflight[cookie.flags.oddeven])
	    == 0)
		return (0);

	hash = pf_syncookie_mac(pd, cookie, seq);
	if ((ack & ~0xff) != (hash & ~0xff))
		return (0);

	counter_u64_add(V_pf_status.lcounters[KLCNT_SYNCOOKIES_VALID], 1);
	atomic_add_64(&V_pf_status.syncookies_inflight[cookie.flags.oddeven], -1);

	return (1);
}
```

We check that the cookie contains the correct authentication string. If it does, we continue into `pf_syncookie_recreate()`, where we reconstruct the original SYN packet. This isn't strictly required for the syncookie system itself, but we need to tell pf about the SYN packet we'd originally discarded so it can create the relevant state entries.

This also allows pf to continue processing, and potentially forwards the reconstituted SYN packet to the remote server. The remote server would then reply with its own SYN+ACK packet, with a different sequence number from ours. pf will have to modify the sequence and ac-

knowledgement numbers on all traffic between client and server. Happily, this is standard functionality for pf.

At this point the connection is fully established on both sides, and it does not meaningfully differ from a connection set up without syncookies. No special action is taken on connection shutdown because this does not present new opportunities for a malicious client to generate memory pressure.

Downsides

So far, we've discussed how syncookies help us, but we've not spent much time any drawbacks. Does that mean that there are none? Sadly, no.

We've already talked about MSS and WSCALE. With syncookies we are unable to reflect the proposed value from the client with full fidelity. This may mean that for some clients we leave some TCP performance on the table. In most cases this is not something to worry about.

Another downside is implicit in how syncookies work: we unconditionally reply SYN+ACK to the SYN packet. Even if the port is actually closed. That means that the client may think opening the connection is working, until it's fully established, only to receive an RST afterwards. That's not ideal and may provoke unexpected client behavior. That is, this may look different from a "normal" failure to connect to users.

This can be mostly mitigated by ensuring that the firewall immediately rejects packets to closed ports. That's generally a good idea anyway, and it goes doubly so if syncookies are enabled.

With syncookies we are unable to reflect the proposed value from the client with full fidelity.

Another downside is that there's no retransmit mechanism for lost SYN+ACK packets. There couldn't be, because as soon as we send the SYN+ACK we forget everything about it. This isn't too much of a concern because the client will just assume its SYN packet got lost and retransmit that. That will lead the server to generate a new SYN+ACK, which will hopefully not get lost this time.

A final thing to bear in mind is that syncookies are not magic. They work well against SYN-flood attacks, but they cannot protect against other attacks. For example, if a specially crafted HTTP request consumes excessive system resources in the web server, this will not be stopped by syncookies.

There's also nothing to stop a motivated attacker from initiating connections from many different client IP addresses without spoofing the source address. In that case, the attacker can still potentially open enough connections to exhaust the server's resources. However, syncookies make this much more expensive for the attacker. A SYN flood can be performed from a single attacking host, with moderate bandwidth requirements. An attack that has the same effect using non-spoofed, TCP connections will require many more attacking hosts.

History

The FreeBSD pf syncookie code was adapted from the pf syncookie code in OpenBSD's pf. This code was originally written by Henning Brauer in 2018 with help from Alexandr Nedvedicky.

The OpenBSD pf syncookie code was based on syncookie code in FreeBSD's TCP stack, originally developed by Jonathan Lemon in 2001 (a9c96841638186f2e8d10962b80e8e9f683d0cbc).

It looks like the OpenBSD commit message is incorrect in its attribution to Andre Oppermann. Andre did make significant improvements to the syncookie code in 2013 (81d392a09de0f2eeabaf68787896863eb9c370a8), which is probably where the misunderstanding came from.

Implementation Notes

While OpenBSD and FreeBSD's pf versions have diverged a bit over the years the similarities still greatly outweigh the differences. As such, porting OpenBSD pf features to FreeBSD is often relatively straightforward. The main stumbling block is the different approaches in locking strategy. OpenBSD's pf, like OpenBSD's network stack, is protected by a single lock (NET_LOCK). This has the advantage of great simplicity but does come with some performance drawbacks.

FreeBSD's pf takes a much more complex approach to locking, but does get better performance in return.

This turned out to be relevant for the adaptive mode. Other aspects of the syncookie code fit neatly into the existing locking approach. However, OpenBSD's approach of incrementing and decrementing a single counter value to track the number of half-open states and in-flight syncookie packets. This required the use of atomic operations in FreeBSD because there's no equivalent NET_LOCK and multiple cores can be processing TCP SYN or other packets at the same time.

While this ensures we do not under or over count the number of half-open states or in-flight syncookie packets, it is still imperfect. The retrieval of the values is atomic, but as we retrieve multiple values, they do not always reflect a perfect snapshot. Happily, there is no requirement for strict correctness here. The worst case is that we enable or disable syncookies slightly early or late. As both syncookie-mediated and normal connections can be established at the same time, this is not a noticeable concern for users.

While OpenBSD and FreeBSD's pf versions have diverged a bit over the years the similarities still greatly outweigh the differences.

Configuration

After all of that, readers could be forgiven for assuming that the configuration of syncookies is a complicated affair, but this is not the case. There's only one required line, in the options section of pf.conf:

```
set syncookies always
```

or

```
set syncookies adaptive
```

The first will always respond to SYN packets with a syncookie SYN+ACK. In adaptive mode pf will only do so when a lot of connections are in half-open state. That is, we've replied SYN-+ACK to an ACK message and are waiting for the ACK in response. This ideally combines the best of both worlds: we get all the advantages of normal TCP connection processing (i.e., full-option negotiation, immediate feedback when the connection cannot be opened) but with some protection against SYN floods.

Adaptive mode, low and high water marks (i.e. where we disable and enable syncookies respectively) can be configured as well:

```
set syncookies adaptive (start 25%, end 12%)
```

The values are expressed as percentages of the state table size. If no syncookie configuration line is present the feature will default to being disabled. This means there is no change in behavior unless users explicitly enable syncookies.

Conclusion

Are syncookies right for you? They may be if your systems are attacked by SYN floods. If they are not you may want to leave them disabled, but even so, it's good to know they exist. SYN floods are a very old type of attack, but as long as they work, even occasionally, attackers may decide to use them. Defenders must be ready with appropriate tools.

The new pf syncookie feature is already present in the recent 12.3 release, and will also be present in the upcoming 13.1 release.

The effort to port the OpenBSD pf syncookie code to FreeBSD's pf was sponsored by Modirum MDPay.

KRISTOF PROVOST is a freelance embedded software engineer specializing in network and video applications. He's a FreeBSD committer, maintainer of the pf firewall in FreeBSD and a board member of the EuroBSDCon foundation. Kristof has an unfortunate tendency to stumble into uClibc bugs, and a burning hatred for FTP. Do not talk to him about IPv6 fragmentation.

FreeBSD® JOURNAL

The FreeBSD Journal is Now Free!

Yep, that's right Free.

The voice of the FreeBSD Community and the BEST way to keep up with the latest releases and new developments in FreeBSD is now openly available to everyone.

DON'T MISS A SINGLE ISSUE!

2022 Editorial Calendar

- Software and System Management (January-February)
- ARM64 is Tier 1 (March-April)
- Disaster Recovery (May-June)
- Science, Systems, and FreeBSD (July-August)
- Performance (September-October)
- Topic to be decided (November-December)

Find out more at: freebsdfoundation/journal

WIP/CFT: mkjail

BY TOM JONES

Jails are one of FreeBSDs most famous features. They enable a single server to offer many services while maintaining clear separation between the applications the server runs. While jails were one of the first uses of containerization, FreeBSD didn't end up with the majority of the mindshare and, instead, tools such as docker won the day.

Part of the reason for this can probably be attributed to the high-quality tools in a normal FreeBSD installation being more than enough to manage a fleet of services on a single host. Over time, tooling has gotten friendlier all round, while in FreeBSD, there is a lack of a single, straight-forward management and automation framework for jail management tasks. This can be seen through the sheer number of projects that aim to add an extra layer on top of the FreeBSD base tools to manage jails and their life cycles.

mkjail is not one of these jail management layers, it is, instead, a simple interface to the creation, updating and upgrading of jails. mkjail only does these three things. Management of the rest of the jail's life cycle is handled by the normal FreeBSD jail infrastructure.

mkjail eschews a lot of the complexity that come with other fuller-featured jail frameworks. Other frameworks allow complex hierarchies of configuration and overlay, using thin jails on top of base jails to reduce foot print a little. Instead, mkjail is a simple tool for creating jails — it creates FreeBSD systems in bottles. The first line of its source describes it well: "Lazy, dirty tool for creating fat jails."

mkjail eschews a lot of the complexity that come with other fuller-featured jail frameworks.

mkjail can be simple by reducing the situations in which it works. It only offers the creation of what are called "fat" jails and every fat jail has an vidual jails.

mkjail uses a small configuration file that describes the location of the zfs datasets for the jails mkjail will create, their location in the file system and finally the install sets which should be installed in the jails. Once a jail has been created with mkjail, it must be integrated into /etc/jails.conf in the same way as another jail managed with the base system tools.

Jail creation can be straightforward with base system tools — bsdinstall has the jail command that can install a jail into any specified directory. mkjail expands on this functionality and handles integration with zfs tools and a clean interface to performing updates and upgrades.

mkjail shines when it comes to updating and upgrading jails, turning these operations into single commands. Because mkjail only runs with fat jails, none of the management issues with thin jails appear when doing updates and upgrades.

WIP/CFT: mkjail

mkjail[1] was initially developed by Mark Felder, was put on github by BSDCan's Dan Lagille and has had contributions from Andrew Fyfe. The project is developed on github (https://github.com/mkjail/mkjail) and is still very young — the github import of the source tree to github goes back to mid 2021.

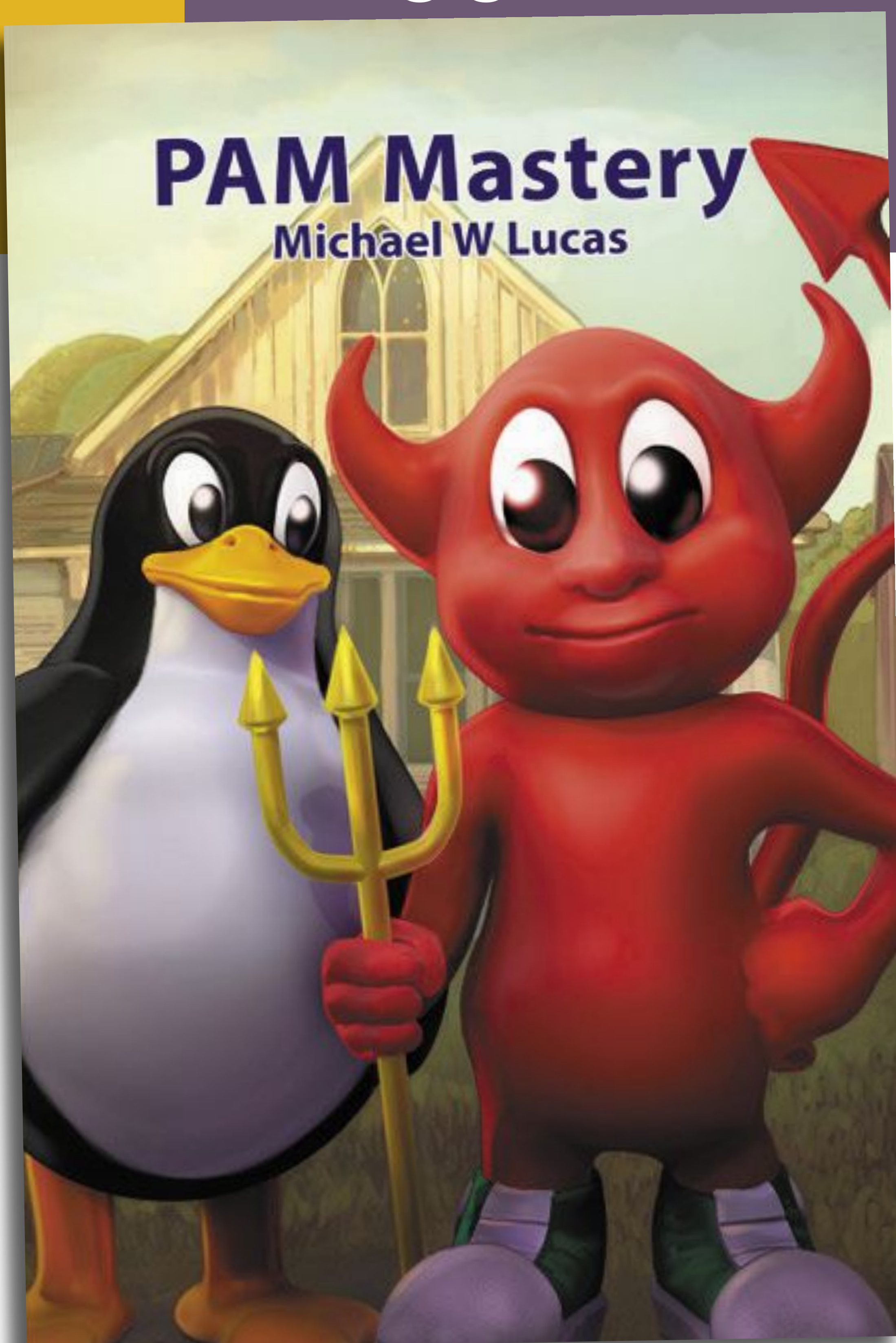
mkjail is intended to be a small tool to enhance and ease jail creation, updates, and upgrades. It has been written as a couple of small shell scripts and is small enough that it could be integrated into the FreeBSD base system.

mkjail is happy to accept contributions through its github in the form of pull requests for code or documentation or with issues there to discuss bugs and new features. mkjail is entirely written in sh which makes contributing code relatively straight forward.

As mkjail is a young project, it can benefit a lot from testing, both of its sub commands as implemented, but also through exposure to your workflow. The best way to contribute to mkjail is to try it out and feedback any issues you had with the documentation, the code or any pain points where small additions would make sense to enable your workflow.

TOM JONES wants FreeBSD-based projects to get the attention they deserve. He lives in the North East of Scotland and offers FreeBSD consulting..

Pluggable Authentication Modules: Threat or Menace?



PAM is one of the most misunderstood parts of systems administration. Many sysadmins live with authentication problems rather than risk making them worse. PAM's very nature makes it unlike any other Unix access control system.

If you have PAM misery or PAM mysteries, you need PAM Mastery!

"Once again Michael W Lucas nailed it." — nixCraft

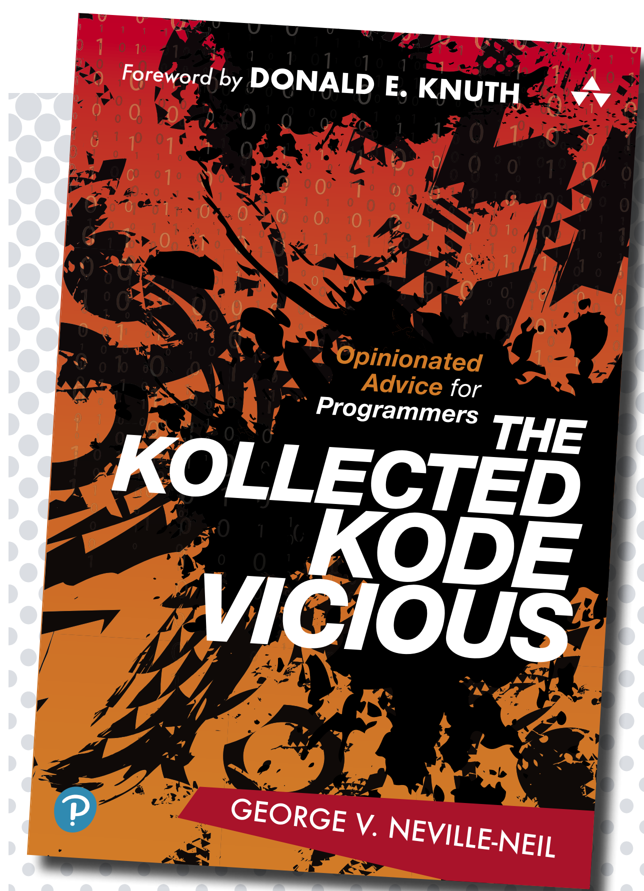
PAM Mastery by Michael W Lucas

https://mwl.io

PRACTICAL PORTS

A review of *The Kollected Kode Vicious*

BY BENEDICT REUSCHLING



Dear Practical Ports,
Should I set aside some time to read *The Kollected Kode Vicious* by George V. Neville-Neil instead of a book about a new programming language — one that I need for my job? I don't have much time for reading, so I need to choose carefully. I also don't see how someone using a pseudonym like KV instead of their real name can be taken seriously, especially when giving such opinionated advice to other people.

— A Skeptical Reader Without Much Time These Days

Dear Skeptical,
Have you ever come across the name Alice Addertongue or Silence Dogood? Or maybe Poor Richard rings a bell? These were pseudonyms used by none other than Benjamin Franklin. Pseudonyms and caricatures were very commonly used during Franklin's time, even though it was often clear who was behind the writing. Not only did the different caricatures allow Franklin to offer readers (of his own newspaper) a different perspective on a wide variety of subjects, but they also permitted him to use hyperbole, sarcasm, and whimsy to entertain readers and shape their opinions. So, writing under an assumed name and persona is not new, and, actually, as you're writing to Practical Ports, you're also participating in the scenario.

Before we get to the book, let's address your problem of setting aside time for reading books.

"Everybody has time. Stop watching fscking Lost!"
— Gary Vaynerchuk, in 2008 at Web 2.0 Expo NY
https://www.youtube.com/watch?v=EhqZ0RU95d4

But, ok, that aside, I can understand that time can be hard to come by these days. Plus, there are only so many books you can read (bonus points if they are good books!). Doomscrolling the news or social media does not count as reading, nor does getting caught in the endless Youtube loop. Books are still relevant sources of knowledge and entertainment — perhaps now more than ever — which is why I've devoted time to reviewing this one. Even with

the sea of electronic information that is the Internet, there is timeless wisdom to be absorbed and pleasure to be had when turning physical pages.

Now, to your question of whether the *Kode Vicious* book is worth some of your apportioned time? For the uninitiated KV reader, I should explain that this book is a collection and grouping by theme of KV's columns from the *ACM Queue* column of the same name. The author wrote the columns over a number of years and *kollected* the best ones for the book, organizing them by topic and adding short intro pieces. How he got one of the giants of Computer Science, Donald E. Knuth, to write the foreword, will perhaps forever remain the author's secret. Talk about being knighted!

I'm a huge fan of quotes at the beginning of chapters and there are plenty of them here that encapsulate what you're about to read in the columns. (Example: *Oh Bullwinkle, that trick never works!* – Rocky J. Squirrel) If you have never read a KV column, know that they follow the revered question-and-answer/advice column format — a format so old that it goes back to Ben Franklin's deconstruction and reconstruction approach, and then, of course, to Socrates. The letters/questions in *The Kollected Kode Vicious* are framed as coming from a confused or puzzled individual seeking KV's advice. The questions are realistic and relevant and often resemble what I hear from students. Some of the later columns clearly address actual correspondence while some of the earlier columns are ghostwritten. In all cases, the answer/advice picks up the question and explains, defuses, deconstructs, rearranges, or discourages assumptions while shedding new light on the topic. Opinions are offered that are generally well intentioned and provide the sought-after advice. Many of these letters (if not all) could end with the words often used in Zen stories: "and thus the student was enlightened."

Checking out the table of contents reveals the following groupings:

- **The Kode at Hand** — deals with everyday annoyances and musings that programmers (like you) must deal with — from allocating too much memory, exception handling (or the lack of it) to proper logging and coding style discussions. This chapter will appeal mostly to programmers and is not for occasional computer users who wonder why the Internet is not working.
- **Koding Konundrums** — considers more philosophical questions like what makes a good programming language, avoiding spaghetti code that endlessly includes one file after the other, why tests are important, and meta-topics like code scanners and debugging strategies. All are valuable to read, and if you've programmed for several years, you'll definitely find a familiar story in there. If you ask yourself why the things are the way they are now, then the next chapter will enlighten you.
- **Systems Design** — looks at the choices (mostly bad ones) that people have made in designing systems we use every day or to program the ones we'll have to shake our fist at in the future. Evergreens like authentication vs. encryption, cross-site scripting, phishing and infections (the latter dealing with computer systems, mind you), and UI design will have you frequently nodding your head in agreement and shaking it at the prime examples of bad design. I will leave it up to you to find out why Java is listed in that chapter.
- **Machine to Machine** — discusses latency, failures to scale, protocol design, and the ever-growing list standards. if you know the author or know of him, it will not surprise you that this is not about networks connecting these machines. And if you think this is upsetting, then wait until you've read the next section...

- **Human to Human** — ponders how to name your hosts, leaving your pride out of it, code interview questions, and bikeshed coloring. People bring a lot of interesting traits to computing — if only they were good ones.

In each of these chapters, using stories that reveal insight and experience, the author cautions and educates. Although often presented with a grumpy undertone (perhaps from having seen too many of these instances), there is not one page where you get the feeling things are hopeless. Instead, there is a lot of practical advice mixed with personal preferences and recommendations and it's always imparted with wit and good humor.

I can imagine two excellent uses for this book. One is for light entertainment reading, and the other is for use as a reference book. The columns are brief enough to read during a short break (with a beverage of choice) or before going to sleep (nightmares notwithstanding).

Keep the book where you'll be able to grab it quickly so that, when necessary, you can re-read the relevant section about the problem you're facing. Another thought is to give it to your colleagues and peers when an argument ensues or when you have a pending design decision. "Let's turn to the Vicious book to remind ourselves of what not to do," can be uttered when discussion seems to be stuck and could use a fresh perspective. Both seasoned and new colleagues will find something interesting and relevant in these columns as they nod in agreement (been there, done that) or smile about an amusing anecdote.

Now, returning to your question of whether you should fill your limited reading time with The Kollected Kode Vicious over a book on a new programming language, I would say, "yes." I think you should read KV's book, but don't expect to find much advice like "use this clever code snippet to make your code faster" or "using Emacs as your editor will increase your productivity thousandfold." Another programming language may look good on a resume, but at the end of the day, these things are mere tools of the trade. There is so much more to what we do than writing code! This is what the book will explain to you as it offers a broad view of the joys and sorrows of the industry. Your mileage may vary, but I think it will be worth your while to read this book and internalize its teachings.

Let me close with another quote:

"Last year, a foolish monk.
This year, no change."
— Japanese Zen poet Ryokan Taigu

PP

BENEDICT REUSCHLING is a documentation committer in the FreeBSD project and member of the documentation engineering team. He serves on the board of directors of the FreeBSD Foundation as vice president. In the past, he served on the FreeBSD core team for two terms. He administers a big data cluster at the University of Applied Sciences, Darmstadt, Germany. He's also teaching a course "Unix for Developers" for undergraduates. Benedict is one of the hosts of the weekly bsdnow.tv podcast.

WeGet letters

by Michael W Lucas

Dear Geezer,

You've been around a while. I know, because one of your books had a terrible accident when I was learning not to eat the cat and my dad was upset because you had signed it, meaning you had touched it, and he was afraid I might catch something from the pages. I know you won't be upset, because he had to buy a second copy, so you came out all right.

For almost that long, people have been babbling about this thing called "packaged base." It's supposed to solve everything, but it never happens. What's *really* going on with it?

—A Young, Smart Person Tired of Waiting

Dearest Useless Punk,

Just because my current age matches the freeway speed limit in oppressive regimes, like Salt Lake City, doesn't mean I'm a geezer. My experience and astute realism, which are often misperceived as undying impenetrable cynicism, make me the geezer. And packaged base has only fed that.

There is only one true way to upgrade any BSD system. You get the source code, preferably by checking it out via SCCS although floppies will do in a pinch and install it under /usr/src. You build the software. You can do this because real operating systems ship with fully functional compilers in the default install. Your phone is not a computer. It is an appliance. So is your doorbell, your pacemaker, anything running Windows, and most Linux installs. (Yes, with enough hacking you can get a compiler on your pacemaker, upgrading it to a computer, but that demands special skills and an impressive degree of reckless self-disregard.) You're running BSD, so you have a computer. Having built it, you install it on that same system, reboot, and voilà! You're upgraded. Yes, some optimizations are permissible — you can use NFS to share the source code and your hand-compiled binaries across your server farm, or better yet assign some flunky who has annoyed you to perform all the upgrades without disturbing you. This is the only way to be certain that the code you install is intended for your systems.

This is the natural state of any BSD system. Deviations from it are unnatural.

Unfortunately, a certain well-meaning but flawed person who I'm not going to specifically point out but whose name rhymes with *Polin Cercival* thought that FreeBSD needed an upgrade system usable by people with a morbid fear of compilers. (Never mind that such people should

not be allowed near a computer, an appliance, or an abacus with three or more beads.) That's where freebsd-update(8) came from. It "conveniently" downloads the smallest possible binary diffs and applies them to a system, so that you can trivially upgrade thousands of systems without even working late. Working late, alone in the office, illuminated by only flickering emergency lights and with no sound but the hum of the air conditioner, is one of the fringe benefits of being a systems administrator. It gives you an excuse to be cantankerous the rest of the time. Why be a sysadmin if you can't surl at the lesser mortals? Fortunately, sysadmins still have developers, network administrators, and the entire sales department to provide an excuse.

And that's where we are today. FreeBSD can be upgraded by anyone who can weasel, wander, or whimsy their way into a root prompt.

Packaged base is the dread dragon of FreeBSD, devouring every developer who sets out to conquer it.

This deplorable state of affairs is somehow insufficiently welcoming for certain members of the community, however. They look at less magnificent operating systems and see that their so-called "base systems" are broken up into packages. User management software is a package. Network software? A package. Every little bitty piece of the system becomes its own package, with its own files and metadata and installation scripts and — worst of all — *dependencies*. Rule of System Administration #32 is very correct in that "Dependencies are the root of all suffering." We've all been trapped managing some barbarian system composed of dozens or hundreds of packages and discovering that essential programs like traceroute and ifconfig are not installed. You have to hunt around to figure out in which package this particular operating system imprisons those vital programs and try to install it, only to discover that the package management system itself needs updating and the package repository version has changed and a currently installed package isn't compatible with the new package and law enforcement officers show up to discuss what your boss keeps insisting was a "bit of an overreaction" when you know perfectly well the entire spree was justified and that the janitor will have no trouble getting the stains out of the carpet, ceiling, and driveway.

Who could possibly want to inflict this upon millions of FreeBSD users? Advocates say that packaging the base system would make it very easy to install minimal FreeBSD systems that contained only the programs needed to perform their assigned tasks. That sounds great, but it's like "exercise" and "eating healthy" and "not petting the adorable Sumatran tiger even though it's *right there*." It's not going to happen. Designing operating system installs that contain only what you need requires predicting infinite capacity to predict the future, or planning, neither of which is likely. You know perfectly well that the tiny system intended for use only for a nameserver is gonna wind up running CRM suites and video editing software for the CEO's nephew's girlfriend's glitterpunk band. That's the natural server lifecycle.

The correct way to get an uncomfortably sparse FreeBSD system is to build it from source. The FreeBSD build system includes options to include and exclude components. Michael Dexter has organized and tested all these options in his Build Options Survey (https://callfortesting.org/results/). You could even proceed directly to OccamBSD (https://github.com/michaeldexter/occambsd), a minimum viable FreeBSD build intended to host jails, bhyve, and Xen clients. OccamBSD is a good place to start, as re-enabling features is much simpler than tearing them out.

Fortunately, FreeBSD itself strongly resists being packaged. It is designed as a single cohesive system and does not like being teased apart into independent components. Sorting out what parts of the system truly depend on one another, and which are merely close personal friends, is a seriously hard problem that many developers have beaten their heads against for years. Many approaches have been attempted and failed. Packaged base is the dread dragon of FreeBSD, devouring every developer who sets out to conquer it. The world has an endless supply of optimistic developers, however, and I have no doubt that one day one of them will succeed and further weaken the moral fiber necessary to run FreeBSD.

With any luck, I'll be dead by then. Or at least not answering your letters.

Have a question for Michael?
Send it to letters@freebsdjournal.org

MICHAEL W LUCAS' head is stuffed full of obsolete knowledge, much of it about FreeBSD, other BSDs, and a few other, lesser operating systems. To learn anything new, he'll have to throw out some of that junk. His latest books include *$ git sync murder*, *TLS Mastery*, and *SNMP Mastery*. *DNSSEC Mastery* should be out by the time you read this, but he's lazy so it probably won't be. Learn more at https://mwl.io.

The FreeBSD Project is looking for

- Programmers
- Testers
- Researchers
- Tech writers
- Anyone who wants to get involved

Find out more by

Checking out our website
freebsd.org/projects/newbies.html

Downloading the Software
freebsd.org/where.html

We're a welcoming community looking for people like you to help continue developing this robust operating system. Join us!

Already involved?

Don't forget to check out the latest grant opportunities at **freebsdfoundation.org**

Help Create the Future. Join the FreeBSD Project!

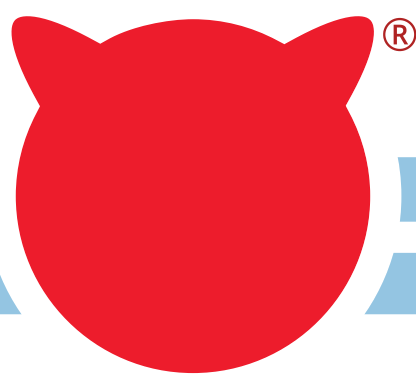
FreeBSD is internationally recognized as an innovative leader in providing a high-performance, secure, and stable operating system.

Not only is FreeBSD easy to install, but it runs a huge number of applications, offers powerful solutions, and cutting edge features. The best part? It's FREE of charge and comes with full source code.

Did you know that working with a mature, open source project is an excellent way to gain new skills, network with other professionals, and differentiate yourself in a competitive job market? Don't miss this opportunity to work with a diverse and committed community bringing about a better world powered by FreeBSD.

The FreeBSD Community is proudly supported by

Support FreeBSD®



Donate to the Foundation!

You already know that FreeBSD is an internationally recognized leader in providing a high-performance, secure, and stable operating system. It's because of you. Your donations have a direct impact on the Project.

Please consider making a gift to support FreeBSD for the coming year. It's only with your help that we can continue and increase our support to make FreeBSD the high-performance, secure, and reliable OS you know and love!

Your investment will help:

- Funding Projects to Advance FreeBSD
- Increasing Our FreeBSD Advocacy and Marketing Efforts
- Providing Additional Conference Resources and Travel Grants
- Continued Development of the FreeBSD Journal
- Protecting FreeBSD IP and Providing Legal Support to the Project
- Purchasing Hardware to Build and Improve FreeBSD Project Infrastructure

Making a donation is quick and easy.

freebsdfoundation.org/donate

My First EuroBSDCon: A New Organizer's Perspective

BY KATIE MCMILLAN

I became involved in EuroBSDCon during the Covid pandemic. It was September 2020, and I didn't know the Board of Directors for the EuroBSDCon Foundation (henceforth referred to as "the Foundation"), and only knew of some of the members of the organizing committee and selection committee from other conferences like BSDCan. EuroBSDCon 2020 had been cancelled as had BSDCan 2021, and EuroBSDCon 2021 was looking dicey too. There wasn't much enthusiasm for a virtual conference (or "online" conference, which sounded slightly better than "virtual", but still bad).

I will be specific about how I got involved, because I think it's important for people to know what to do if they want to get involved in conferences and administrative aspects of open source projects, especially internationally. I am a Canadian woman and would encourage anyone who wants to get involved in open source projects to reach out to key people on the projects, starting with Boards of Directors. We are all humans, and volunteering/donating (whether it be lines of code, bug fixes, porting, vulnerability discoveries, monetary donations, hosting booths at conferences, giving talks, administrative support, documentation, project management, community building, advocacy, etc.) makes a great impression. There are many ways to give back to open source projects, which are bringing humankind some of the best hardware and software available today.

In this case, I emailed the Foundation that runs EuroBSDCon to ask how to get involved. I kept it short, and said:

Hello,
I would like to volunteer for EuroBSDCon 2021 and am wondering what the best way is to get involved? I live in Ottawa, Canada, and normally volunteer for BSDCan, but as it has been cancelled for 2021 I would like to divert my energy to EuroBSDCon.
Thank you,
Katie

And a couple of weeks later I was managing the website. Well, no, it wasn't quite that simple. Turns out I was signing on to learn Wordpress and inherit a website for a conference which may or may not happen. Some plugins, content, assets, different users, SEO configurations, a theme…, but while inheriting the site would also seem scary in the sense of entering into a "decision-by-committee" process with the Foundation, who were mysterious (to someone who

doesn't use social media), it ended up not being like that at all. In fact, they wanted someone to help with the website, and were happy that I wanted to contribute in this way. So, I started learning to use Wordpress as a CMS, picking up HTML and CSS, and with Lighthouse audits and SEO tools, realized how powerful open source tools were for web development. The board members acted as experienced guides; I felt accompanied, assisted, and appreciated., I developed at my own pace, and that worked really well. When I screwed up and crashed the website (which only happened once or twice) everyone was fine with that too. What it highlighted to me, is that learning is a continuous process, and I wasn't just going to learn web development, get my gold star, and be finished. With their support I'm continuing to develop and learn new things all the time.

The Conference

Some of us attended the conference online and others in person; it really ended up being a hybrid event. In all, we had approximately 500 registrations, reaching 100 simultaneous attendees. It was amazing to see so many great talks, questions, and to put faces to names. For me, it was particularly well-timed as PGDay Austria was being held in-person at the same time, so I was able to attend that while I was in Vienna. It was so nice to attend in-person, as, due to the pandemic, I hadn't travelled for a long time before that. I really wanted to be there to help support the community and tools. I was pleasantly surprised when Austrian locals showed up to attend and support in person!

My favorite part was using the tool I supported from the beginning of the conference organization: Big Blue Button.

For the conference infrastructure we leveraged a combination of tools, some open source some closed source. Kristof Provost brought key insights to ensuring that the tools were going to work harmoniously and provide a streamlined experience. We aimed to somewhat resemble (or be inspired by) the in-person conference experience with moderated talk rooms, allowing for engagement with speakers and Q/A sessions, with social coffee periods and sponsor booths in a separate area. My favorite part was using the tool I supported from the beginning of the conference organization: Big Blue Button. We used it as our video conference and recording software: this open source software is awesome! I use it all the time to whip up a quick meeting room, and it is one of the favorite open source tools in my toolbox, especially after it worked so well for an online conference of this size.

The other neat part about using Big Blue Button was our ability to find a fantastic provider of hosting, support, and video editing services. Since it's open source, we were able to be highly selective, and look everywhere, and we ended up choosing a company called RiAdvice, based in Tunisia. The CEO, Ghazi Triki, and his team were fantastic to work with and it really felt like having a strategic partner who was part of the team. We were grateful to the team at RiAdvice for all of their help with planning, execution, support, and post conference activities

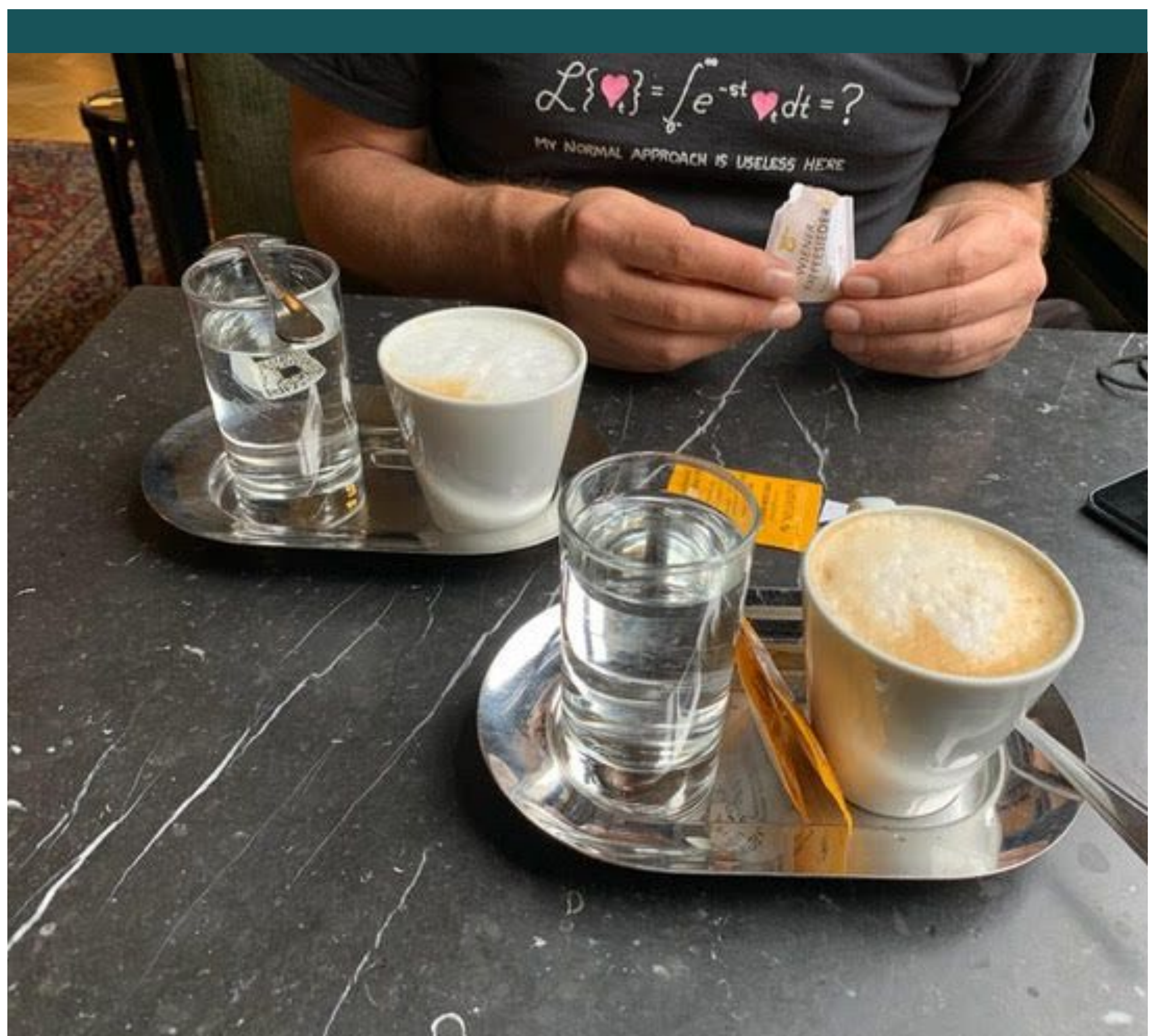
including video editing. Having this level of friendly, professional support really added to the fun and reduced the stress, of the whole event. It was great to be a part of such an innovative and collaborative conference experience and to be able to leverage open source software for an open source software conference.

I appreciate how included I felt as a woman at the conference. Not only did the bathroom doors have inclusive signs on them, but everyone made me feel welcome and part of the team. So much so, in fact, that I have since formally joined the Board of Directors for the Foundation. I do ask—especially if you're a person identifying as any gender other than male and reading this—that you consider submitting a talk abstract or volunteering at [next year's conference!]

Why I'll Do it Again Next Year

The coffee was spectacular. Okay, that's not the only reason, but you absolutely must add "drink a melange in Vienna" to your Bucket List, it is a wonderful experience. Vienna is a truly breathtaking city, with its fabulous architecture, food, monuments, shopping, theatre, palaces, public transportation — I encourage everyone to travel to this gorgeous city.

I'll definitely do it again next year because of the people. I loved the eclectic group of people and was touched by the hospitality and inclusion. I laughed so hard, and not just when Henning Brauer tasted a wild plant that was growing on the SBA Research terrace. He was fine, but I have decided to update my First Aid and CPR certificate for next year's conference.



At the end of the day, conferences and other types of community building, networking, professional development, and engagement activities are critical to the successx- of open source projects of all kinds. These are community projects leveraging transparent community-driven development. This approach fosters the sustainability, advanced security, vendor neutrality, social positivity, inclusivity, innovation, and interoperability of the projects. It also encourages international human connections and relationships. Let's not lose sight of how important human connections are in this digital world. Sometimes it is easy to feel like

an island when it feels like those around you don't share your interests.

I thank the EuroBSDCon Foundation Board, Conference Sponsors, Organizing Committee, Selection Committee, and Attendees for allowing me to be part of something so important to foster the diversity, creativity, comradery, sense of humor, and endurance of the BSD international community.

My friends, see you next year and I wish you a "Gute Fahrt!"

KATIE MCMILLAN has worked in Canadian healthcare since 2004; she began her career at Health Canada assisting with the development of the Air Quality Health Index (AQHI). She has continued in health roles, building her career around her love for standards, innovation and digital excellence. With a focus on empowering digital strategies with security and interoperability, which has led her to open-source solutions, she has had job titles such as GIS Analyst, Application Services Consultant, and Digital Strategy and Excellence Lead. Since 2021 she has broadened her vision of the healthcare system, and now works for an established Canadian marketing, applications, and software vendor, Snap360. She continues to participate in the advancement of digital health through use, development, and promotion of open-source technologies such as OSCAR EMR, R/R Studio, PostgreSQL, Mirth Connect, *BSD, WordPress, and HL7. Katie enjoys hiking, coffee, cross-country skiing, sudokus, horse-back riding, and the Bay of Quinte.

EuroBSDCon 2021: Report by René Ladan

Q = imaginary interviewer

R = René

Q: How did you get involved with the EuroBSDCon Foundation?

R: I have been using FreeBSD since 2003 after dabbling for a bit with some Linux distributions. I started contributing to the (now defunct) Dutch translation of the FreeBSD Handbook and was consequently rewarded with a doc commit bit in 2008. I made the same mistake with ports land and was faced with a ports commit bit in 2010. I have been on the Ports Management Team (portmgr@) since 2016 where I mostly assume the role of secretary.

Q: OK, but what about the EuroBSDCon Foundation?

R: Ah, yes, so because I was following some mailing lists, I read about this conference in Canada called BSDCan. I figured at that moment that developers actually meet in person from time to time, which was a nice surprise. So, I ended up debuting my BSD conference visits with BSDCan in 2010. Unfortunately, I couldn't visit EuroBSDCon in 2010 because of work obligations, but I attended it in 2011 and haven't skipped a year of EuroBSDCon since.

This non-stop attendance also drew the attention of the EuroBSDCon Foundation in 2016, which resulted in me being pulled over during dinner to join a board meeting the next day. During that meeting, it turned out that they were looking for some more Dutch members because of (by)laws, as the Foundation is Dutch and I happen to be Dutch too. I agreed to join the board and have been doing secretary work ever since—things like proposing agendas and minuting, but also helping out attendees with embassy paperwork.

Normally, the EuroBSDCon is an in-person event, but the COVID pandemic changed the rules.

Q: What do the conferences that you help organize look like?

R: So, normally, the EuroBSDCon is an in-person event, but the COVID pandemic changed the rules. We decided to completely cancel the 2020 edition because BSDCan already went virtual and we felt a second online conference that year wouldn't add much. We had some hope that this year's conference could be held in person again, probably on a smaller scale—with prob-

ably mostly European attendees. But we had to organize an online component to cater to our overseas attendees anyway, so we figured it was easier to just keep it online.

Q: Did you ever organize an online conference before?
R: While we didn't have to organize any local component, we did have to learn how to organize an online conference from scratch. We visited some other online conferences just to take a peek at how they did things. We settled on using BigBlueButton for the talks and Spatial.chat for the hallway track.

We chose two services because BigBlueButton had more mature recording capabilities at the time and an API that we could use to schedule the recordings, while Spatial.chat provided a real nice online hallway experience.

I think the conference was a success despite being online. We did feel the interaction with people was missing, giving a talk or making announcements in front of a screen just isn't the same as with a crowd sitting in front of you. We held an unofficial, in-person mini-conference in Vienna which attracted some people from the organization and some local people—so that gave us the opportunity to still socialize a bit before, during, and after the conference.

Q: Would you organize another conference?
R: Yes, for multiple reasons.

First of all, it is lots of fun to attend these conferences, with a new city to visit each year. So, these conferences are also a kind of mini holiday for me. The talks provide insight into the current state of affairs, but more importantly, the hallway track allows me to catch up with people which I might not have seen for a year or longer. While I mostly do invisible work for most people, the work in itself does enable several people to attend each year and it also keeps the machinery going.

Let's hope that work is for a full, in-person conference in Vienna (again) later this year!

RENÉ LADAN studied computing science at the Eindhoven University of Technology where he graduated in 2006. After that he has worked at various companies, including the university itself. He currently works as a software engineer at Carapax IT.

René started his open source shadow career with some small projects on Sourceforge but it really took off when he started to work on FreeBSD in 2004. Meanwhile, he has been awarded both a documentation and a ports commit bit and is now part of the Ports Management Team (aka portmgr@). After visiting too many instances of EuroBSDCon, he was drawn into the accompanying Foundation in 2016 and assumes the role of secretary ever since.

When not doing BSD stuff and still in nerd mode, he likes to tinker with his homebrew time station receivers. Outside of technical things, René likes to hike, puzzle, and work in his parents' garden.

Events Calendar

BSD Events taking place through July 2022

BY ANNE DICKISON

Please send details of any FreeBSD related events or events that are of interest for FreeBSD users which are not listed here to freebsd-doc@FreeBSD.org.

Open Source 101

March 29, 2022
VIRTUAL
https://opensource101.com/

Open Source 101 is a one-day conference focusing on the "basics" of open source. Join the AllThingsOpen team and some amazing sponsors, speakers and attendees for a full day of live open source programming on Tuesday, March 29. FREE to attend. The Foundation is pleased to be a Media Sponsor.

SCaLE

SCALE 19x

July 28-31, 2022
Los Angeles, CA
https://www.socallinuxexpo.org/scale/19x

SCaLE 19X – the 19th annual Southern California Linux Expo – will take place July 28-31, 2022 in Los Angeles, CA. SCaLE is the largest community-run open-source and free software conference in North America. It is held annually in the greater Los Angeles area. Roller Angel will also be hosting a FreeBSD workshop during the conference.

FreeBSD Fridays

https://freebsdfoundation.org/freebsd-fridays/

Stay tuned for new episodes in early 2022.
Past FreeBSD Fridays sessions are available at: https://freebsdfoundation.org/freebsd-fridays/

FreeBSD Office Hours

https://wiki.freebsd.org/OfficeHours

Join members of the FreeBSD community for FreeBSD Office Hours. From general Q&A to topic-based demos and tutorials, Office Hours is a great way to get answers to your FreeBSD-related questions.

Past episodes can be found at the FreeBSD YouTube Channel.
https://www.youtube.com/c/FreeBSDProject.