



TM

# FreeBSD

TM

## JOURNAL

March/April 2015

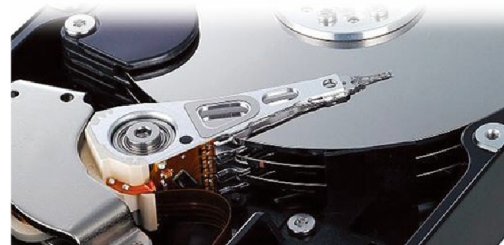
# ZFS

## BEST PRACTICES



## What's New in FreeNAS 9.3

### SHINGLED MAGNETIC RECORDING



## HOW TO BUILD A PORT



# 2.2

**FreeBSD 10.1-based Network Security  
from the pfSense® team.**

---

**Cloud Solutions:**



**Hardware Appliances:**

AES-NI Acceleration for IPSec  
Professional Support



Exclusively available from Netgate, the home of pfSense.



7212 McNeil Drive Suite 204 Austin, TX 78729 +1.512.646.4100

Netgate® is a registered trademark of Rubicon Communications, LLC. pfSense® is a registered trademark of Electric Sheep Fencing, LLC.  
Amazon Web Services and the "Powered by Amazon Web Services" logo are trademarks of Amazon.com, Inc. or its affiliates in the United States and/or other countries.  
VMware and the "VMware ready" logo are registered trademarks of VMware, Inc. in the United States and/or other jurisdictions.

# Table of Contents

# ZFS

## BEST PRACTICES

### 3 Foundation Letter

Pleased to present the eighth issue of FreeBSD Journal!

**32 Ports Report** In the first two months of 2015, 4,046 commits were applied to the ports tree, 1,182 PR were closed, and 1,002 emails were received by portmgr@ (without counting the spam...). And all this handled by an average of only 130 active ports developers! *By Frederic Culot*

### 34 This Month in FreeBSD

The FreeBSD Project has the distinction of being a mentoring organization for every Google Summer of Code since the program's inception in 2005. *By Dru Lavigne*

**36 Book Review** If you understand DNS and want to learn how to secure it, *DNSSEC Mastery* is for you. It's a clear and concise guide with a ton of hand-holding and plenty of examples. *By Joseph Kong*

### 38 Events Calendar

*By Dru Lavigne*

## INTERACTING with the FreeBSD Project

**24 How to Build a Port** A brief overview of how the ports system is structured and an introduction on how to build a simple port. *By Erwin Lansing*

**30 The FreeBSD Foundation at 15** We were not always so capable, but during our 15 years the vision to get us here has not changed. *By Justin Gibbs*

**4** Hopefully the tips and best practices in this article will help you avoid some of the common mistakes made by those new to ZFS.

*By Allan Jude*

## Shingled Magnetic Recording [SMR]

**12** While alternatives to PMR (perpendicular magnetic recording) are being developed, manufacturers are adopting a more compact track layout, termed shingled magnetic recording (SMR), to increase capacity. *By Justin Gibbs*

## What's New in FreeNAS 9.3

**18** Recently released FreeNAS 9.3 adds several new features, many of which take advantage of OpenZFS and recent FreeBSD optimizations.

*By Dru Lavigne*



# FreeNAS

## in an Enterprise Environment

By the time you're reading this, FreeNAS has been downloaded more than 5.5 million times. For home users, it's become an indispensable part of their daily lives, akin to the DVR. Meanwhile, all over the world, thousands of businesses, universities, and government departments use FreeNAS to build effective storage solutions in myriad applications.



**NEW RELEASE**

### What you will learn...

- How TrueNAS builds off the strong points of the FreeBSD and FreeNAS operating systems
- How TrueNAS meets modern storage challenges for enterprise

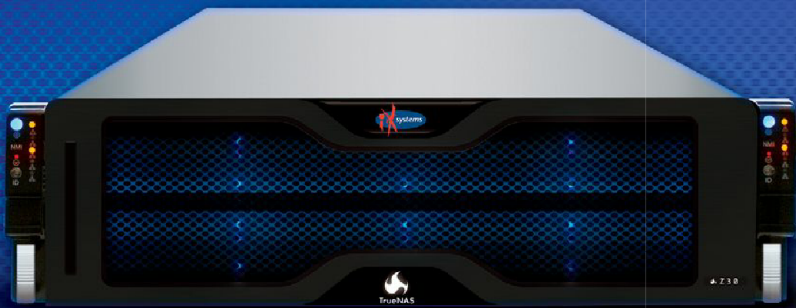
**DO NOT INTERRUPT THIS MAGAZINE TO BRING YOU THIS IMPORTANT ANNOUNCEMENT:**

THE PEOPLE WHO DEVELOP FREENAS, THE WORLD'S MOST POPULAR STORAGE OS, HAVE JUST REVAMPED TRUENAS.

The FreeNAS operating systems is free to the public and offers thorough documentation, an active community, and a feature-rich storage environment. Based on FreeBSD, it can share over a host of protocols (SMB, FTP, iSCSI, etc) and features an intuitive interface, the ZFS file system, a plug-in system for much more.

Despite the massive popularity of FreeNAS, many aren't aware of its big brother dutifully protecting data in some of the most demanding environments: the proven, enterprise-grade, professionally-supported line of TrueNAS.

But what makes TrueNAS different? Well, I'm glad you asked...



### Commercial Grade Support

When a mission critical storage system goes down, an organization's whole operation can halt. Whole community-based (and free), it can't always get an expert and running in a timely manner. Responsiveness and expert support from a dedicated support team can provide that safety.

Created by the same team that developed FreeNAS.

**POWER WITHOUT CONTROL MEANS NOTHING. TRUENAS STORAGE GIVES YOU BOTH.**

- |   |  |
|---|--|
| <input checked="" type="checkbox"/> Simple Management   | <input checked="" type="checkbox"/> Self-Healing Filesystem            |
| <input checked="" type="checkbox"/> Hybrid Flash Acceleration                                 | <input checked="" type="checkbox"/> High Availability                  |
| <input checked="" type="checkbox"/> Intelligent Compression                                   | <input checked="" type="checkbox"/> Qualified for VMware and HyperV    |
| <input checked="" type="checkbox"/> All Features Provided Up Front (no hidden licensing fees) | <input checked="" type="checkbox"/> Works Great With Citrix XenServer® |

To learn more, visit: [www.iXsystems.com/truenas](http://www.iXsystems.com/truenas)



### POWERED BY INTEL® XEON® PROCESSORS

Intel, the Intel logo, Intel Xeon and Intel Xeon Inside are trademarks of Intel Corporation in the U.S. and/or other countries.

VMware and VMware Ready are registered trademarks or trademarks of VMware, Inc. in the United States and other jurisdictions.

Citrix makes and you receive no representations or warranties of any kind with respect to the third party products, its functionality, the test(s) or the results therefrom, whether expressed, implied, statutory or otherwise, including without limitation those of fitness for a particular purpose, merchantability, non-infringement or title. To the extent permitted by applicable law. In no event shall Citrix be liable for any damages of any kind whatsoever arising out of your use of the third party product, whether direct, indirect, special, consequential, incidental, multiple, punitive or other damages.



# FreeBSD<sup>TM</sup> JOURNAL

## Editorial Board



- John Baldwin • Member of the FreeBSD Core Team
- Justin Gibbs • Founder and President of the FreeBSD Foundation and a senior software architect at Spectra Logic Corporation
- Daichi Goto • Director at BSD Consulting Inc. (Tokyo)
- Joseph Kong • Author of *FreeBSD Device Drivers*
- Dru Lavigne • Director of the FreeBSD Foundation and Chair of the BSD Certification Group
- Michael W. Lucas • Author of *Absolute FreeBSD*
- Kirk McKusick • Director of the FreeBSD Foundation and lead author of *The Design and Implementation* book series
- George Neville-Neil • Director of the FreeBSD Foundation and co-author of *The Design and Implementation of the FreeBSD Operating System*
- Hiroki Sato • Director of the FreeBSD Foundation, Chair of AsiaBSDCon, member of the FreeBSD Core Team and Assistant Professor at Tokyo Institute of Technology
- Robert Watson • Director of the FreeBSD Foundation, Founder of the TrustedBSD Project and Lecturer at the University of Cambridge

## S&W PUBLISHING LLC

PO BOX 408, BELFAST, MAINE 04915

- Publisher** • Walter Andrzejewski  
walter@freebsdjournal.com
- Editor-at-Large** • James Maurer  
jmaurer@freebsdjournal.com
- Art Director** • Dianne M. Kischitz  
dianne@freebsdjournal.com
- Office Administrator** • Michael Davis  
davism@freebsdjournal.com
- 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,  
PO Box 20247, Boulder, CO 80308  
ph: 720/207-5142 • fax: 720/222-2350  
email: board@freebsdoundation.org  
Copyright © 2015 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 Board

We're pleased to present you with the eighth issue of *FreeBSD Journal*. Subscription rolls continue to grow and subscriber retention remains high. Taken together, these are clear signs that the contributors, editorial board, and staff who spend so much of their time working on the *Journal* are doing a great job. Thank you!

Yet another cause for celebration is the 15-year anniversary of the FreeBSD Foundation, the good folks who help fund the *Journal* and who do a great many other things to keep the FreeBSD Project going strong. Justin Gibbs, the founder and current president of the Foundation, has written an article for this issue that lays out the history and future of the Foundation and also includes a mention of some exciting work coming out this year.



The editorial board decided to focus the technical articles in this issue on the topic of storage, one of the areas in which FreeBSD continues to excel. The adoption of ZFS from OpenSolaris has opened up huge new vistas for storage applications, all built around an open-source solution. Getting the most from a ZFS installation requires some tuning, and that's just what Allan Jude shows us in his article on ZFS Best Practices.

On an educational note, the FreeBSD Project is once again a mentoring organization for Google's Summer of Code (GSoC), and mentors and students are currently being matched up. You can find more on FreeBSD's involvement in GSoC at <http://www.google-melange.com/gsoc/homepage/google/gsoc2015> and in this issue in Dru Lavigne's This Month in FreeBSD column.

In closing we want to thank everyone who's reading this, those who have recently subscribed and those readers who have re-upped their subscriptions. We've got a lot of topics to cover in the *Journal* this year, including a few articles on how you can interact with the FreeBSD Project (see Erwin Lansing's article in this issue on How to Write a Port) as well as themed issues on networking, FreeBSD in the Cloud, performance tuning, and much more. Please stay tuned!

Sincerely,  
**FreeBSD Journal Editorial Board**



ZFS



# ■ ZFS BEST PRACTICES ■

BY ALLAN JUDE

ZFS is known for its reliability and for protecting data from the dreaded bitrot. However, a concerning number of users have run into trouble because they did not understand just how different ZFS is from every previous file system they have ever used. The goal of this article is to set out a number of the best practices for using ZFS to help you avoid running into these common misconceptions. This article covers hardware, configuration, tuning, and features, as well as a few other tips.

## **Beware Hardware RAID**


The most costly mistake people make is not giving ZFS control over their redundancy. It is preferable for ZFS to provide the redundancy (using ZFS mirroring or RAID-Z), rather than using a hardware RAID controller. Important data in ZFS is stored multiple times, in what are called ditto blocks. Pool-wide data has three ditto blocks (so is stored three times), and file system metadata has two ditto blocks. In addition, ZFS purposely stores the ditto blocks on different disks, so that the failure of any one disk cannot lead to the loss of all copies of the metadata. This gives ZFS better visibility into what's actually going on with the hardware, allowing more types of errors to be corrected.

When a ZFS pool is created on top of hardware RAID, the RAID controller presents a single logical volume to the operating system (and therefore to ZFS). When ZFS has only one disk to work with, it doesn't have anywhere to store the parity information required to rebuild the array in the event of a failure, and no way to ensure the ditto blocks end up on separate disks. While this doesn't seem like a very large issue because the underlying RAID controller can rebuild the missing disk from parity that exists at the hardware RAID level, using hardware RAID

means you do not get the benefit of ZFS' block level resilvering process. The hardware has to resilver every byte on the entire disk, rather than only the blocks that contain active data. Hardware RAID also doesn't help in the situations where ZFS excels: detecting and repairing flipped bits and other types of otherwise undetectable errors in your data or the file system itself. ZFS' checksumming is one of its most powerful features, but if there is no ZFS redundancy, it can only detect, not repair, the errors.

Hardware RAID controllers can also cause countless other issues, and you are much better off with a less expensive simple HBA controller instead. Hardware RAID controllers will sometimes mask certain errors and silently retry commands, hiding this information from ZFS. This is mostly a holdover from the era where the operating system could not be trusted to gracefully deal with the error. Without this information, ZFS cannot make informed decisions, nor can it self-heal a failing drive by reading the data from the parity location(s).

Even when trying to use a RAID controller as a simple HBA (in "IT", or "JBOD" mode), many controllers will require that you create a single disk RAID 0 (or a JBOD) out of each disk before you can use it. In these cases,



when it comes time to replace a failed disk, one cannot simply just swap the disk out of the chassis, but an operator must use software provided by the adapter manufacturer to create the new volume with the replaced disk. Such a tool may not exist, or may not be supported on FreeBSD, requiring a reboot to access the BIOS level tools provided by the adapter. Suddenly your hot-swap drives still require a reboot and are not providing the high availability you expect. One of the other advantages to ZFS is pool portability. All of the disks in a pool can be moved to another server, easily imported, and brought back into operation in short order. However, if the disks are labeled by manufacturer A's RAID utility, they won't be accessible by manufacturer B's RAID controller. This might make it impossible to read the disks if the controller fails, and cannot be replaced with an identical controller.

If some outside factor absolutely requires that you use hardware RAID, always present a pair of logical volumes to be mirrored in ZFS, or 3 or more volumes for RAID-Z. The added redundancy provided by the hardware RAID will cost capacity and performance, but not taking advantage of ZFS' better redundancy will eventually make you wish you had. Also be sure to disable "write back" mode on the controller. If write back is not disabled, the cache flush commands ZFS relies upon to ensure all the pending data is actually written to the disk are ignored, possibly causing data loss. Ultimately, ZFS on top of hardware RAID decreases performance while adding complexity, and that increases risk.

## ECC Ram

Many articles and posts will emphatically state that in order to use ZFS, you must have ECC RAM. To quote Matt Ahrens, cocreator of ZFS: "ZFS on a system without ECC is no more dangerous than any other file system on a system with ECC." The features of ZFS are extremely compelling and extraordinarily useful, no matter what role the machine is playing. The data protections provided by ZFS outclass that of every other file system, even without the benefit of ECC RAM. Don't let the FUD scare you away from running ZFS, even on a single disk on your laptop. Just be sure to take careful backups.

If your goal is high availability, the extra cost of ECC is easily justifiable. Server grade hardware uses ECC RAM, because it mitigates the

risk of memory errors, and the application crashes those errors can cause. Using ECC RAM reduces (but does not entirely eliminate) the risk of corruption on in-flight data, after it leaves the application, but before ZFS has calculated the checksum and written it to stable storage. If you care about the reliability and availability of your system, you should use ECC RAM, regardless of what file system you use.

## Backups Are Still Required

While the reliability and durability of ZFS make us feel much safer about our data, too few people still maintain proper backups of the data they store on their ZFS pools. No amount of RAID redundancy is as good as a backup. Accidentally destroy the wrong pool? Where is your backup? Pull the wrong disk while replacing a failed one? Where is your backup? Add an extra disk as a new vdev instead of attaching it to the existing mirror? Where is your backup?

ZFS provides a number of features that make taking backups much easier and safer. The first is instantaneous snapshots. Instead of backing up a live system, where files can be changing constantly, take a backup of a recursive snapshot of the system. This way, everything contained in the backup is saved as it was at an exact moment in time, even if the backup takes multiple days to complete. The other powerful backup feature is the built-in incremental replication system. The first advantage to this method is that, compared to walking the file system with Bacula or rsync, ZFS walks its internal representation of the data, resulting in a fast contiguous read of the blocks. The output of "zfs send" can be incremental or full. In incremental mode, the output maintains the copy-on-write aspects of the data stream, including all the snapshots. This output can either be stored as massive binary block (a ZFS data stream), or piped to "zfs receive", where a replica of the dataset can be recreated on another machine, effectively creating a "warm spare" of the dataset. ZFS also has another feature that is great for taking live backups: pool splitting. When your pool consists of mirror vdevs, with ideally at least 3 devices in each mirror set so that the pool is not at risk from a single device failure during this operation, one device from each mirror vdev can be detached by the "zpool split" command, creating a new pool. This new pool contains one drive from





each vdev and all of the data from the current pool. These disks can then be ejected and moved to an off-site location, similar to a tape backup. New disks are then swapped in, and rejoin the mirror set and resilver. Repeated on an ongoing basis, this process provides a complete offsite backup that takes only a few seconds to create. Drives can be reused once they have exceeded your backup retention threshold. Compared to a regular backup, the pool split method can be less performance impacting because the backup time is amortized over the period after the new drives are added to the pool. As new data is written to the pool, it is mirrored to the regular mirror devices, but also to the device that will be split off when the next backup is taken. In this setup, it may be wise to adjust the resilver throttling (`vfs.zfs.resilver_delay`), to avoid unduly impacting performance when the new drives are added after the split.

## Disk Labeling

There are a number of ways to handle labeling disks and creating the logical connection between the device exposed by the operating system and the physical disk in its physical location in the chassis. The one that has worked best for me in production has been the physical slot number followed by the disk's serial number. For example, our one chassis has 24 drives in the front, and 12 more in the rear. The drive in slot 6 in the front is `f06-WMC1F125320`, and a drive in the rear might be `r02-9WM7HATN`. There are a number of reasons to use the serial number, mainly inventory and warranty management; the serial number is required to check if the drive is still under warranty and to get it replaced. It is a convenient unique identifier, although depending on the model and drive manufacturer, it can be a bit unwieldy. The serial number for one of my Intel SSDs is `CVDA333604282403GN`, which one would struggle to fit on a label on the front of a 2.5 inch drive carrier, and doesn't fit within the 15 character GPT label. It is best to truncate and keep the most significant digits, so if the serial numbers share a common beginning, use the last characters of the serial number. Adding the slot number as a prefix helps operators and data center technicians more quickly identify the correct physical drive, without having to compare the entire serial number, especially when a series

of drives may have a common prefix in the serial number. The next step is to change the operating system's representation of the drive to match. The recommended approach is to use GPT partition labels, like so:

```
gpart modify -i 2 -l f01-9WM6T60L da0
```

This will create an alias for `da0` called `/dev/gpt/f01-9WM6T60L`

Note: The maximum length of the label is 15 characters.

This way, even if the order in which the operating system recognizes the devices changes, the name of the device stays the same. It also means that the output of "zpool status" will show each disk, with its location and serial number. With this technique, a missing disk will be obvious, and it will be easy to communicate to the operator or data center technician which disk needs to be replaced.

FreeBSD offers a number of different ways to label disks, and it may be helpful to disable the unused ones, to avoid ZFS picking up those device names instead:

```
/boot/loader.conf:
```

```
kern.geom.label.disk_ident.enable=1
kern.geom.label.gptid.enable=0
```

Disk Ident example:

```
/dev/diskid/DISK-07013121E6B2FA14
/dev/diskid/DISK-%20%20%20%20%20WD-WCC131365642
/dev/diskid/diskid/DISK-%20%20%20%20%20%20%20%20%20%20%20Z300HTCE
```

GPT ID example:

```
/dev/gptid/b829bf8c-46ad-11e3-ae0f-002590721162
/dev/gptid/b88eeff5-46ad-11e3-ae0f-002590721162
```

Notice how at first glance, the two appear to be identical. The difference is at the beginning of the string, rather than the end.

Another disadvantage to Disk Ident and GPTID is that the partition identifiers get tacked on the end, so the 2nd partition on the disk is

```
/dev/diskid/DISK-07013121E6B2FA14p2,
```

which blends into the unique ID of the disk

In more advanced setups with SAS expanders, dual ported disks, and multiple controllers, each disk may be presented to the operating system multiple times, once for each unique path.

FreeBSD's GEOM storage management layer has a system for this, `gmultipath`. This writes a unique label to the disk, and then when 2 or more devices appear with the same label, they are classified as multiple paths to the same physical disk.

In the end, it looks something like this:

```
# gmultipath status
multipath/f01-WMC1F125320  OPTIMAL      da0 (ACTIVE)
                               da36 (PASSIVE)
multipath/f02-WMC1F125298  OPTIMAL      da1 (ACTIVE)
                               da37 (PASSIVE)
multipath/f03-WMC1F125506  OPTIMAL      da2 (ACTIVE)
                               da38 (PASSIVE)
```

## Setting Up the Disks

Before creating the pool, the disks need to be prepared. There are a number of guides, blogs, and other resources that state that ZFS should always be used on an entire disk, not a partition. While this is true in Solaris, because of the way the disk cache works, it is not true under FreeBSD. There are a few considerations at this point. If the system will boot from the pool, then all of the disks should contain the ZFS boot code. In case of a failure, it may not be possible to predict which disk the system will try to boot from. The `freebsd-boot` partition should be 512kb; this is just shy of the maximum imposed by the FreeBSD ZFS boot blocks. The purpose behind using the maximum size is to ensure that there will be enough room for the boot blocks to grow over time. All of the partitions created on the disk should be aligned to 4k boundaries, to keep the partition layout consistent across all the disks. This can be accomplished by using `-a 4k` with each `gpart` command when creating the partitions.

```
gpart create -s gpt ada0
gpart add -t freebsd-boot -l bootfs0 -s 512k -a 4k ada0
gpart add -t freebsd-swap -l swap0 -s 2g -a 4k ada0
gpart add -t freebsd-zfs -l f01-9WM6T60L -a 4k ada0
gpart show -l ada0
=>  34 7814037100  ada0  GPT  (3.7T)
    34         6          - free - (3.0k)
    40        1024        1      bootfs0 (512k)
    1064       984        - free - (492k)
    2048      4194304     2      swap0 (2.0G)
    4196352   7809839104  3      f01-9WM6T60L (3.7T)
    7814035456 1678          - free - (839k)
```

Even if the current disks use 512 byte sectors, in the future it may not be easy to obtain 512 byte sector disks to replace these, so setting up the entire pool based on 4k sectors ensures that complications will not arise when a disk needs to be replaced in the future. With that same goal in mind, the partition that will hold the ZFS data should be created slightly smaller than the available size of the disk. This extra space can be used as a swap partition. This slack will offer some wiggle room in the case where the replacement disk does not have the exact same sector count as the original disk. Lastly, the ZFS pool itself should use 4k sectors. Again, even if your drives are 512 byte sectors, their future replacements may not be, and it is not possible to mix sector sizes, nor to change the ZFS sector size after the pool is created. To force ZFS to use a 4k sector size, set the `sysctl vfs.zfs.min_auto_ashift=12` ( $2^{12} = 4k$ ) before creating the pool. The downside to 4k sectors is slightly worse space efficiency and possibly worse performance for very small reads or writes (less than 4k). In the case of a database type workload and sub 1 TB disks, 512 byte sectors may be desirable.

Some disk models include an "XP Jumper," which offsets each LBA address by 1, so that the default starting location of the first MBR partition (63rd sector) becomes the 64th 512 byte sector and is therefore 4k aligned. However, if this jumper is set and we ask the partitioning tool to align the partitions to 4k, they will all be off by 1 sector, and will cause the performance penalties we were trying to avoid in the first place.

## Pool Layout

Determining the best way to lay out the disks and `vdevs` in your pool is one of the hardest questions facing a user creating a new pool.

There are many factors to consider, and once the decision is made, it generally cannot be changed. The biggest factors are random I/O performance, streaming performance, space efficiency, and fault tolerance. Each different configuration provides different benefits. For the best IOPS performance for random reads, the best solution is always more `vdevs`. Sets of mirrors (equivalent to RAID 10) provide the best performance because the IOPS of each `vdev` is effectively limited to that of the slowest device, so 12 disks in 6 mirror sets provides 6x the IOPS of a single disk,

whereas all 12 disks in a single RAID-Z (1, 2, or 3) provides only 1x the IOPS of a single disk.





More performance can be gained by running the 12 disks as 2 RAID-Z2 vdevs, or even 3 or 4 RAID-Z1 vdevs, at the cost of less usable space, but the performance never reaches that of the mirror sets. In the case of streaming performance, where IOPS make much less of a difference, spindle count is all that matters.

Assume a modest set of commodity spinning disks, 1 TB in size and capable of 250 IOPS and streaming read/writes at 100 MB/s:

## Avoiding Single Points of Failure

Avoiding single points of failure will increase the availability of your pool. With some planning and informed design decisions, the same hardware can be organized in a more fault-tolerant configuration. In larger installations, where all the disks may not reside in the same physical chassis, consideration should be given to which disks belong head with 3 external JBOD chassis with 36 disks


Disks	Configuration	Read IOPS	Write IOPS	Read MB/s	Write MB/s	Usable Space	Fault Tolerance
2	1x 2 disk Mirror	500	250	200	100	1 TB	1
3	1x 3 disk Mirror	750	250	300	100	1 TB	2
	1x 3 disk RAID-Z1	250	250	200	200	2 TB	1
4	2x 2 disk Mirror	1000	500	400	200	2 TB	1 (2*)
	1x 4 disk RAID-Z1	250	250	300	300	3 TB	1
5	1x 5 disk RAID-Z1	250	250	400	400	4 TB	1
	1x 5 disk RAID-Z2	250	250	300	300	3 TB	2
6	3x 2 disk Mirror	1500	750	600	300	3 TB	1 (3*)
	2x 3 disk Mirror	1500	500	600	200	2 TB	2 (4**)
	1x 6 disk RAID-Z1	250	250	500	500	5 TB	1
	1x 6 disk RAID-Z2	250	250	400	400	4 TB	2
12	6x 2 disk Mirror	3000	1500	1200	600	6 TB	1 (6*)
	4x 3 disk Mirror	3000	1000	1200	400	4 TB	2 (8**)
	2x 6 disk RAID-Z1	500	500	1000	1000	10 TB	1 (2*)
	2x 6 disk RAID-Z2	500	500	800	800	8 TB	2 (4**)
36	18x 2 disk Mirror	9000	4500	3600	1800	18 TB	1 (18*)
	12x 3 disk Mirror	9000	3000	3600	1200	12 TB	2 (24**)
	1x 36 disk RAID-Z2	250	250	3400	3400	34 TB	2
	2x 18 disk RAID-Z2	500	500	3200	3200	32 TB	2 (4**)
	4x 9 disk RAID-Z2	1000	1000	2800	2800	28 TB	2 (8**)
	6x 6 disk RAID-Z2	1500	1500	2400	2400	24 TB	2 (12**)

\* Provided that the failures are limited to 1 per vdev

\*\* Provided that the failures are limited to 2 per vdev

Using a larger number of smaller groups of disks increases performance at the cost of reduced usable space (more parity). Streaming read and write performance is constrained by the number of non-parity spindles. This leads to another consideration for both random and streaming performance: spindle count. An array of 12x 1 TB drives will usually outperform 6x 2 TB drives, because the greater spindle count increases both IOPS and streaming performance.

each should be configured such that each RAID-Z2 vdev consists of 2 disks from each JBOD (18 vdevs of 6 disks each). In this configuration, even if one JBOD's power supply, HBA, or cabling fails, each vdev is still functional. Whereas if the configuration consisted of vdevs made up of disks all in the same JBOD, a number of vdevs would be faulted, and the system would not be able to continue. The same approach can be taken in smaller systems that might not have multipath to tolerate an HBA failure. If constructing mirror



pairs, ensure that each disk is paired with another disk that is not on the same controller, so the failure of one controller only degrades, rather than faults, each affected vdev.

## Compression

ZFS supports transparent compression where data is compressed as it is written to the disk and decompressed as it is read back, without the user or application needing to be aware. In addition to the obvious decrease in storage utilization this provides, it also increases performance. Even with the small CPU usage penalty, compressed data can be read from the disks at the same speed as uncompressed data, but once decompressed, provides a much higher effective throughput. If a disk can read 100 MB/s, and data is compressed 50%, then that data can now be read at effectively 150 MB/s. The same applies to writes, where there is increased throughput and decreased latency because a smaller amount of data takes less time to write. This makes compressed datasets very useful for databases, which often contain highly compressible text and always benefit from higher throughput and lower latency. The newer LZ4 compression algorithm used in ZFS also has an “early abort” feature, which will store a block uncompressed if the compression ratio on the first bit of the block is less than 12.5%. This further reduces the performance impact of using compression, since incompressible files are quickly skipped. With this in mind, you can consider using LZ4 compression on the entire pool.

## Deduplication

ZFS supports online deduplication, meaning data is deduplicated as it is written. While this feature appears attractive, you will see it is quite expensive. Here is why. In order to deduplicate the data as it is written, ZFS builds a hash table of the SHA256 checksum of each block (called the deduplication table, or DDT). The DDT is stored in main memory as part of the ARC (Adaptive Replacement Cache). As new blocks are queued to be written, they are first compared to the DDT, and if a match is found, the ref count of the existing block just needs to be increased, instead of writing out the block. If no match is found, the new data is written and the new checksum is added to the DDT. Each entry in the DDT takes 320 bytes of memory or more. If there is not enough room in the metadata portion of the ARC, the DDT entries are written to the L2ARC (usually an SSD or other fast storage device used as a second level cache) where they take even

more space. In the typical case of a zvol backed iSCSI target, the block size is 8 KB, meaning 1 TB of unique data would generate nearly 48 GB of DDT. If that won't fit in ram, it is instead stored in the L2ARC, but that has a memory cost of its own. In addition, DDT entries take more space on disk than they do in memory, and have worse performance. In the event of a problem with the pool, the DDT needs to be able to fit in memory when the pool is imported, and if it cannot, the pool may not be able to be imported. DDT and L2ARC mappings also count as metadata, which by default is restricted to only 1/4 of the available ARC memory. If you are going to use deduplication, the metadata limit will need to be adjusted. The benefit of deduplication has a very high cost—if your dataset is only going to get a moderate deduplication ratio, you are most likely much better off just using LZ4 compression. If I still have not dissuaded you from using deduplication, you should conduct some tests on your data to ensure you are getting a very good deduplication ratio, and calculate how much ram will be required to store the DDT for all of your data. Be sure to leave room in the ARC for actual metadata, in addition to the DDT and L2ARC index, and, of course, the cache of your actual data. You will need a lot of ram.

## Reservations

The pooled nature of storage in ZFS means that all the available free space is available to every dataset. Compared to the traditional way of doing things, partitioning a RAID volume or creating separate volumes, the free space does not become fragmented. The downside to this is that one workload or user can consume all the available space. While this can be addressed with quotas, that doesn't always solve the problem. ZFS also offers a reservation system where a specific dataset for a critical database or for security logs can be guaranteed a minimum amount of space unavailable to any other dataset. To ensure that the e-commerce database never runs out of room because of HTTP logs, give its dataset a reservation.

Although recent improvements to ZFS have improved the situation greatly, a ZFS pool that is nearly full will perform badly. If the pool becomes completely full, the administrative commands to resolve the situation can take an exceedingly long time. One way around this is to create a new dataset, called “reserved,” with a reservation of 20% to 25% of the total capacity of the pool. This will prevent that critical last bit of space from being used up. The reservation can be relaxed to

allow the administrator to perform needed operations or to tide the system over until the pool can be expanded.

## Tuning

The biggest variable in ZFS is the size of the ARC. The ARC is where ZFS stores recently-used and frequently-used data as well as its metadata. The ARC provides most of the amazing performance that comes with ZFS. The maximum size of the ARC defaults to 1 GB less than all memory in the system (or 1/2 of all memory in machines with little memory). For a dedicated file server, this makes sense; however, if there are going to be other applications that require memory, you might want to tune this to something more modest, leaving room for other applications like a web server or database server. The limit is set with a tunable in `/boot/loader.conf`:

`vfs.zfs.arc_max`. Generally it is best to leave at least a few gigabytes of memory for the OS and for applications, but maximizing the memory available to the ARC will increase performance. The ARC will release memory back to the OS when it detects memory pressure, but this is not instant and may cause heavy swapping. As dis-

cussed earlier, the default limit on metadata is 1/4 of the ARC. If your dataset contains a very large number of small files, it might be advantageous to increase this value.

Hopefully these tips and best practices guide you well and help you avoid some of the common mistakes made by those new to ZFS. We hope to see you join the thriving community of OpenZFS users and developers. Special thanks to Dan Langille, the backup guru, and Michael Dexter, who has dressed many self-inflicted foot wounds for various clients and shared what he learned. •

---

**Allan Jude** is VP of operations at ScaleEngine Inc., a global HTTP and Video Streaming CDN (Content Distribution Network), where he makes extensive use of ZFS on FreeBSD. He is also the host of the video podcasts *BSD Now* (with Kris Moore) and *TechSNAP* on *JupiterBroadcasting.com*. Allan is a FreeBSD doc committer, focused on improving the handbook and documenting ZFS. He taught FreeBSD and NetBSD at Mohawk College in Hamilton, Canada, from 2007 to 2010 and has 12 years of BSD UNIX sysadmin experience.

**ISILON** The industry leader in Scale-Out Network Attached Storage (NAS)

Isilon is deeply invested in advancing FreeBSD performance and scalability. We are looking to hire and develop FreeBSD committers for kernel product development and to improve the Open Source Community.



We're Hiring!

With offices around the world, we likely have a job for you! Please visit our website at <http://www.emc.com/careers> or send direct inquiries to [karl.augustine@isilon.com](mailto:karl.augustine@isilon.com).

EMC<sup>2</sup>

ISILON



# Shingled [SMR] Magnetic Recording

**[This Is Not Your Father's Disk Drive]**

Existing hard disk products are nearing the estimated 1Tb/in<sup>2</sup> limit in areal density achievable using current perpendicular magnetic recording (PMR) technology. While alternatives to PMR are being developed, manufacturers are adopting a more compact track layout, termed shingled magnetic recording (SMR), to increase capacity. SMR currently provides a 20% improvement in density, but removes the ability to independently update sectors on the media. In this article we explore the characteristics of SMR and its impact on traditional storage software.



# by Justin Gibbs

**Since the introduction** of the hard drive by IBM in 1956, the disk industry has been tracking Moore's Law, doubling storage density every two years. But that trend may now be in jeopardy. Disk platters are nearing the 1Tb/in<sup>2</sup> areal density limit of the currently used technology, perpendicular magnetic recording (PMR), and higher density schemes such as heat-assisted magnetic recording (HAMR) and bit patterned media (BPM) are still two years or more away.

To fill this roadmap gap, manufacturers are turning to shingled magnetic recording (SMR). SMR achieves a higher density by shrinking the effective track size. Doing this without any changes to the media or head design takes advantage of the existing PMR technology's ability to read smaller tracks than it can write. Instead of leaving a gap between two adjoining tracks to ensure there is no interference (Figures 1 and 2), SMR purposely overwrites a portion of the previously written, neighboring track when laying down a new stripe of data.

The old track remains, but is trimmed to a narrower width. The new track is the same width as on a non-SMR drive. Writing in this way across a platter, with an occasional gap band so writes don't have to be continuous across the entire media, currently yields a 20% increase in capacity. (Figure 3) This advantage may increase in the future.

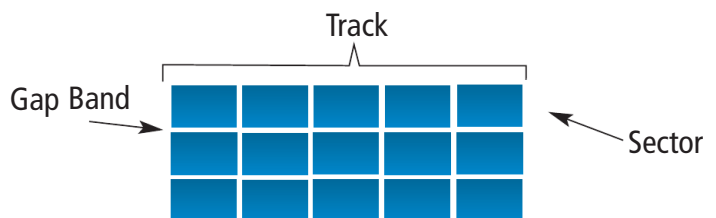
SMR is so called because writes are performed in a pattern similar to the shingles on a roof. Just as it is impossible on a roof to replace a single shingle without perturbing those surrounding it, SMR drives do not allow operations that replace the contents of individual sectors. Write operations must start at a gap band on the media and proceed sequentially to the next gap band in order to not lose previously written data.

Since this model is so different from the behavior of disk drives recorded with independent tracks, drive manufacturers are adding firmware features to ease the transition for file systems and applications. In products shipping today, the drive manages all the complexity of shingling and emulates a standard direct access device. Models available within the next year will provide additional commands to expose the SMR traits of the drive.

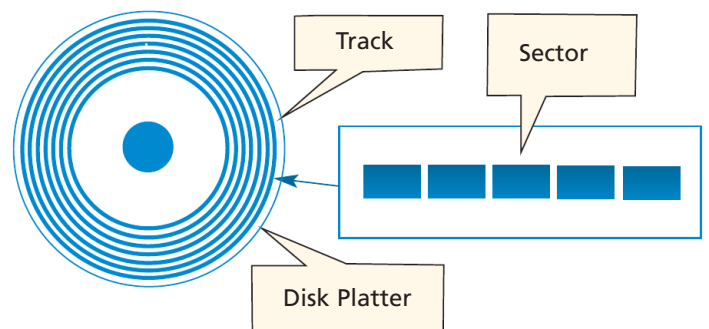
SMR-aware software can use this information to avoid operations that must be emulated, but the drive will still accept any direct access command. For environments where the whole software stack is SMR aware, models will eventually be offered with no emulation at all. With the extra CPU, memory, and media over-provisioning required for emulation removed, these drive models should provide the best price per terabyte.

The emulation provided by current SMR models allows all existing software to function, but often at a significant performance penalty. Writing randomly to a shingled device will force the firmware to perform garbage collec-

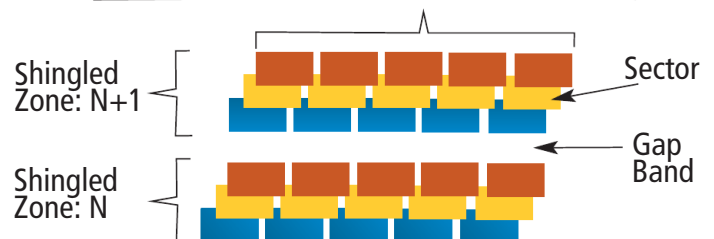
**Fig. 1 Conventional Track/Sector Layout**



**Fig. 2 Conventional Media Layout**



**Fig. 3 Shingled Zone/Track/Sector Layout**



tion, and in most cases, a single write operation from the host will be converted into two or more write operations to the media. Given enough media over-provisioning, RAM, or flash, it should be possible to provide performance that meets or exceeds that of a traditionally formatted hard drive. But along with increasing density, the market expects decreasing per-terabyte cost.

The additional component cost would break this trend. So while emulation will improve over time, the storage systems looking to achieve optimal performance will need to become SMR aware.

Adapting existing storage stacks to SMR will not be easy—the ability to overwrite any LBA is an inherent assumption of most implementations. Given this difficulty, many have hoped that SMR will be phased out once new technology like HAMR and BPR are available. Currently, the chance that this will occur seems remote. The roadmaps of the major players in the spinning drive market show SMR as a contributing technology far into the future for their march to achieve Moore’s Law.

## SMR Devices

Current industry roadmaps show three distinct types of SMR devices: drive managed, host aware, and host managed. Drive-managed devices provide the full direct access command set of a standard drive and use internal mapping techniques to give the illusion that individual sectors can be updated randomly. Host-aware devices include all the emulation support of a drive-managed device, but also support commands that SMR-aware software can use to write optimally and avoid emulation penalties. Host-managed devices remove all emulation support, forcing the host to manage tasks like

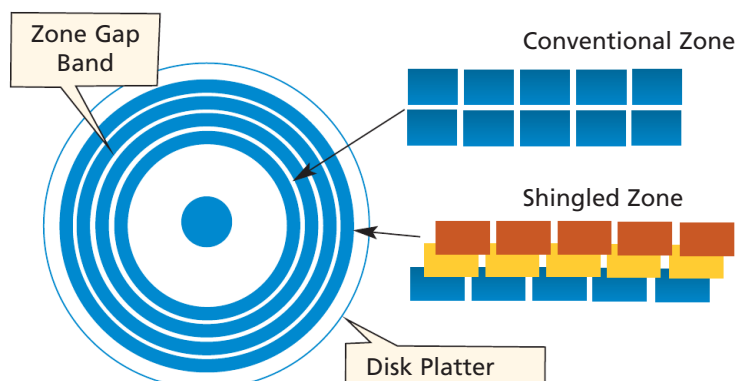
garbage collection directly. In all cases, the physical characteristics of an SMR drive are the same.

An SMR hard drive is divided up into zones, collections of sequentially addressed sectors. Zones can be managed independently since they are separated from each other by unwritten space. These “gap bands” ensure that write activity from one zone cannot impact another. Zones can be either conventionally formatted or shingled. A conventional zone allows updates to individual sectors, just like on a conventional, non-SMR, disk drive. A shingled zone must be written sequentially from lowest to highest logical block address (LBA-sector address) so that already in-place data isn’t corrupted by subsequent writes. In order to achieve the density improvements SMR promises, SMR zones are large—tens if not hundreds of megabytes—and the majority of zones in a device must be shingled. Typical configurations may also include a small number of conventional zones for storing rapidly changing blocks and/or metadata for managing the SMR zones, but this is not required (Figure 4).

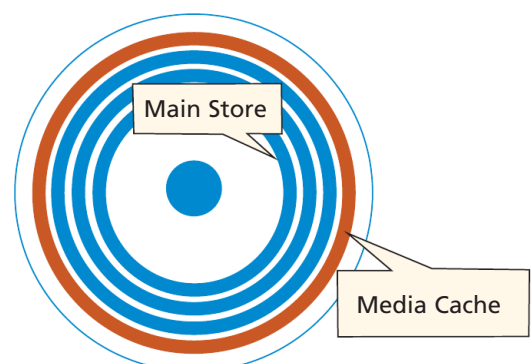
## Drive-Managed Devices

Providing conventional hard drive semantics while using SMR media adds tremendously to the complexity of the device’s firmware. One possible strategy for performing this emulation is to completely decouple the LBA presented to the host from the physical location of data on the media. This technique is used to great effect on solid state drives, and SMR zones have many of the same traits as the flash memory used in SSDs: a shingled zone is effectively the erase block size, and writing in ascending address order is highly preferred. However, the economics do not support this approach. The market

**Fig. 4 SMR Media Layout**



**Fig. 5 Layout with “Media Cache”**





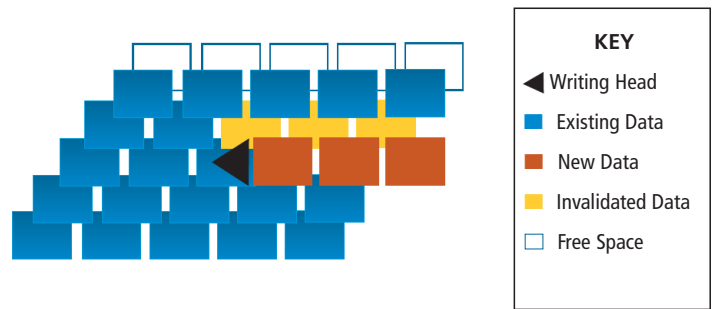
**Fig. 6 Mid-Zone Overwrite**

expects spinning drives to be significantly cheaper than flash storage. Currently an 8-TB SMR drive retails for under \$.04 per GB, compared to almost \$.40 per GB for the lowest performance 1-TB flash drive. There isn't enough price margin to allow manufacturers to add the large RAM, flash, or CPU resources needed to manage a fully dynamic LBA map on an SMR device. Even if the market supported higher prices, the latency cost for random-access reads on spinning media gives good incentive to make most requests that are sequential to the host (sequential in LBA space) also sequential on the media.

Instead, current generation drives use a "cache and clean" approach. Writes that would violate the sequential write requirements of an SMR zone are directed to a media cache. This media cache is composed of zones reserved for use by the firmware. These zones typically occupy space at the outer diameter of the platters where transfer rates are the greatest (Figure 5). An implementation may also reserve zones throughout the media in order to avoid large seek penalties for diverting to the media cache when the head is already positioned at the inner diameter. The media cache may utilize either conventionally formatted zones, shingled zones, or a combination of both.

Using shingled zones to manage other shingled zones may seem counterproductive, but doing so increases the density of the cache, resulting in less raw capacity being lost to emulation. Drives that chose this approach apply the same log-structured, journaling techniques used in databases, flash devices, and even some conventional file systems to always write sequentially to the cache.

The steps taken to service a read request help to illustrate how the media cache and the remaining SMR zones interact in this emulated environment. The LBAs requested by the read fall into two categories: those residing in the main SMR store and those that were redirected to the media cache and may be migrated back to the main store sometime in the future. To make this classification, the firmware maintains enough information in RAM to quickly determine if an LBA has been redirected to the media cache. If that lookup succeeds, the physical location in the media cache is known. If the lookup fails, the physical location is also known. Bad block remapping aside, there is a linear mapping from LBA to physical address in the main store, allowing the zone and zone offset to be quickly derived from the LBA. Once all physical locations are known,



the disk drive can perform seek optimization, read the blocks, and stitch them together for transmission to the host.

By maintaining a map of only those sectors that reside in the media cache, the RAM and CPU resources required for emulation can be greatly reduced. The media cache is a small fraction of the device's total capacity and so is the map. But this also means that it is possible to exhaust the media cache. This is where the clean step of "cache and clean" comes into play. In parallel with servicing host activity that causes reads and writes to the media cache, the drive must also promote data from the media cache to the main store, freeing space to service future write activity.

In its most basic form, cleaning entails preserving any data to be retained from a main SMR zone in the media cache, overwriting the SMR zone with both new and retained data, and then marking portions of the media cache as free for reuse. There are, however, strategies the firmware can employ to make the process less expensive. To reduce the impact of cleaning operations on host command latency, the cleaning can be done in stages. If host I/O arrives, those LBAs that have been transferred to the main store can be marked as free, and the cleaning operation suspended. After servicing higher-priority operations, cleaning can resume where it left off. Another savings is possible given that writes to an LBA in the middle of a zone can impact LBAs at higher addresses, but not at lower addresses. The firmware can thus avoid preserving data at LBAs below the first LBA of data being promoted from the media cache. Additionally, the range of LBAs disturbed by a mid-zone write is relatively small (a track or two) and well known to the firmware. Rather than clean all the way to the end of the zone, the firmware may stop early and just leave the LBAs for the invalidated sectors in the media cache (Figure 6). You can think of this approach as

“hole filling.” If the range of contiguous LBAs to be promoted to the media cache is greater than the number of LBAs invalidated by a write, hole filling results in a net reduction in the amount of media cache in use.

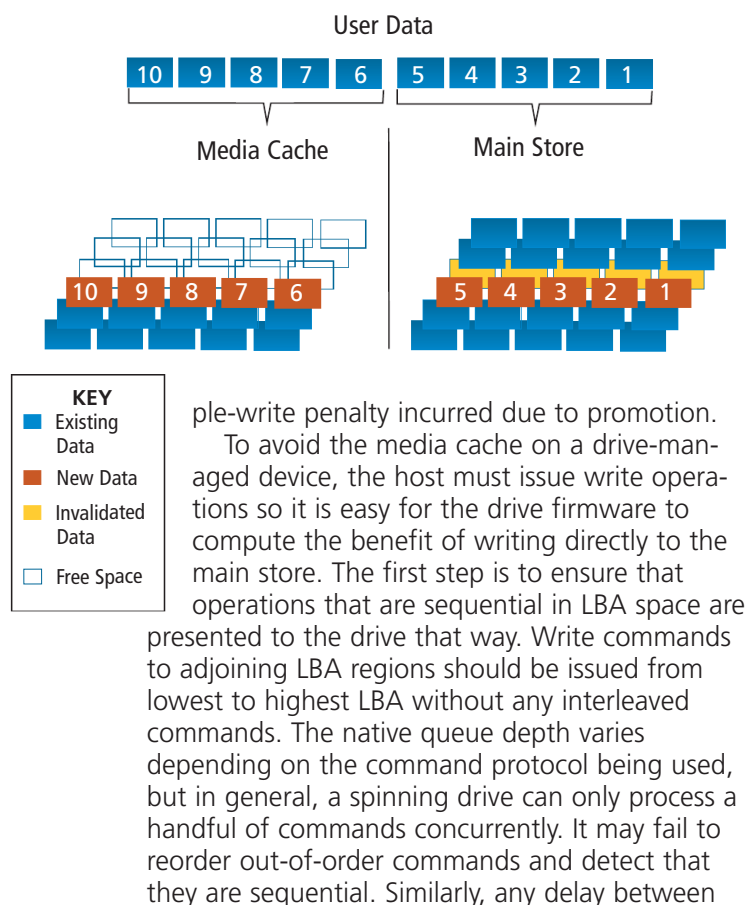
There are workloads where having a media cache increases performance. If, for example, the host is rapidly issuing random writes to a small number of LBAs, these sectors can simply reside in the media cache without ever being promoted. Since most media cache implementations are journaled, the LBAs from these writes are written sequentially to the media. The net result is near sequential I/O performance for bursts of random writes. Even when promotion must eventually occur, if the total load on the device is low enough, the drive may have sufficient idle time to do promotion without any host-visible penalty. However, random write workloads are migrating to solid-state devices. For the more common spinning-disk workload of storing larger objects that are rarely overwritten, the goal should be to avoid the media cache entirely and eliminate the multi-

sequential commands may cause the firmware to timeout and assume that the stream has ended.

Issuing commands in the right order is required, but not sufficient to bypass the media cache. The goal of the firmware is to retire a write without having to read any data. It can only do that and also bypass the media cache if the write operation is large. At today’s media densities, a track holds on the order of 4MB, and at least a track is invalidated by a mid-zone write. Any write operation (or group of sequential write operations) that consumes less than two tracks of space will go to the media cache. For writes larger than this, the drive will write the first portion directly to the main store and direct the last track’s worth of data to the media cache (Figure 7). In this way, the drive can fulfill the write without having to read any data—the write to the main store is stopped before invalidating any of the sectors that need to be retained.

The need to meet data integrity guarantees can also prevent writing directly to the main store. It is possible for the host to disable a drive’s write cache, forcing all write commands to be reported as complete only after the data is committed to stable storage (persists across power loss). Given the limited queue depth and RAM resources of the drive, this will almost certainly cause the firmware to prematurely terminate a direct media write so that it can finalize commands and return them to the host. Selectively issuing write commands that bypass the cache (e.g., setting the “force unit access” bit in a SCSI write) or a command to flush the full cache will also force the drive to end any current optimization. For these reasons, the write cache should always be enabled on drive-managed SMR devices and the frequency of synchronizing commands heavily limited.

**Fig. 7 Hole Filling/Redirection to Media Cache**



mand allows modification of the write pointer in a zone. Currently, this command is specified to only allow the write pointer to be set to the beginning of a zone, effectively erasing a zone in one shot. Based on the reserved byte space in this command, it appears that future versions of the ATA and SCSI specifications may allow the write pointer offset to be explicitly set—for example, to just erase then overwrite the tail end of a zone.

With this geometry information in hand, the host can intelligently assign LBAs to the data it is writing. Data structures for tracking free and allocated space can be aligned and sized to match a drive's native zone layout. Rapidly overwritten data such as a file system's free space maps can be directed to conventional zones. Large writes of user data can be allocated from contiguous space starting at the write pointer in SMR zones. As LBAs are freed from SMR zones, the free space fragmentation this causes can be tracked so that this space is only reused when the system is relatively idle or when there is a high return when compared to the cost of reclaiming a zone. When all the data in an SMR zone is no longer needed, a RESET WRITE POINTER will tell the drive it needn't preserve any sectors for subsequent writes to that zone. These are just a few of the possible optimizations for avoiding nonsequential writes and the penalties for having the drive emulate them.

## Host-Managed Devices

With sufficient sophistication in the host software, we can take these optimizations to their logical conclusion: perform all zone management within the host. This is where the host-managed devices in the SMR roadmap come into play. While host-aware devices report shingled zones that are "sequential write preferred," on a host-managed device, these are "sequential-write mandatory." All emulation features are absent on this drive type and any write that doesn't start at the write pointer for a zone is rejected.

There are a few unexpected complications with host-managed devices. While the firmware of a drive-managed or host-aware device is free to invalidate LBAs in the middle of a shingled zone so long as access to those LBAs is properly redirected to the media cache, the current specification for host-managed devices is much more restrictive. To ensure the drive can report which LBAs are valid utilizing the least amount of drive-maintained metadata, writes are only allowed at the write pointer, and only those

LBAs below the write pointer are readable. Any attempt to read outside the range from the beginning of the zone to the write pointer will fail, even if those sectors were written previously and are still valid on the media. For this reason, host-aware software is somewhat more constrained in the approaches it can take to effectively utilize a host-managed device.

The specifications for host access to SMR drives are in their infancy, and we can expect some aspects of this restriction to eventually be addressed. Already there is a proposal to allow a circular zone type. In this type of zone, only LBAs in the interference region just past the write pointer are considered invalid. All other sectors can be read. While still not providing the freedom to implement some of the redirection schemes of drive-managed and host-aware devices, circular zones would naturally lend themselves to journaling. And journaling may be sufficient for getting optimum performance from these devices.

## Conclusion

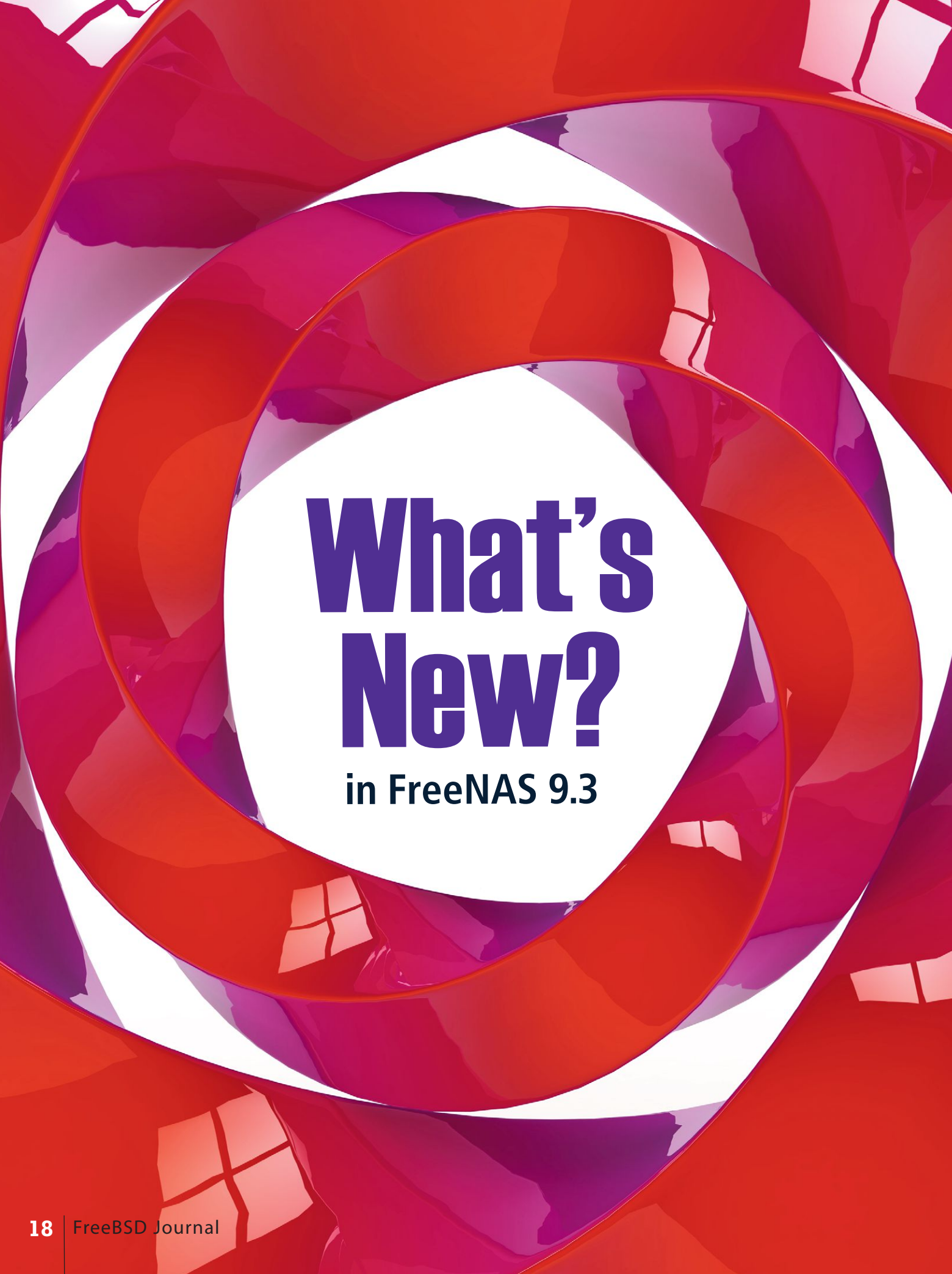
After almost 60 years, the fundamental way that software interacts with spinning magnetic media is changing. Given the trajectory of the main providers of hard disks, storage systems developers have little choice but to adapt. Until they do, consumers should expect reduced performance from SMR drives when deployed in traditional workloads.

The stage is now set to see how storage appliances, operating systems, and file systems tackle the challenge of SMR. With an understanding of how drive-managed devices perform their emulation, some basic optimization for SMR drives is possible. Going beyond this and taking full advantage of host-aware and host-managed devices will take more time. Storage stacks are complicated—often taking tens to hundreds of man-years to develop. Because of this, the impact of SMR will proceed in slow motion through the storage industry. Five years from now, there will be definite winners and losers: those who have adapted to SMR and those who have not. For now, it's just too early to predict who the casualties will be. ●

---

**Justin Gibbs** is the founder and president of the FreeBSD Foundation, and has been working on the storage-related subsystems of FreeBSD since 1993. He currently works at Spectra Logic Corporation building petabyte-scale, archive storage systems using FreeBSD, flash, disk, and tape.





# **What's New?**

**in FreeNAS 9.3**



# FreeNAS

**FreeNAS is an open-source, BSD-licensed, network-attached storage (NAS) operating system based on FreeBSD. It uses OpenZFS, the open-source version of ZFS, a self-healing file system which is particularly suited for storage and for maintaining the integrity of the data being stored.**

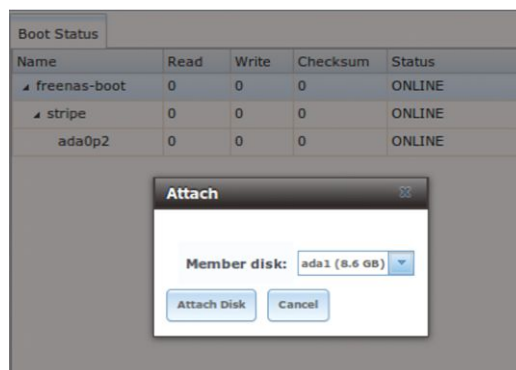
FreeNAS 9.3, which was released in December 2014, adds several new features, many of which take advantage of OpenZFS and recent FreeBSD optimizations. This article provides an overview of some of these features. Refer to the FreeNAS 9.3 User Guide ([doc.freenas.org/9.3](http://doc.freenas.org/9.3) <<http://doc.freenas.org/9.3>> <<http://doc.freenas.org/9.3>>) for more information about FreeNAS, its features, and available configuration options.

## ZFS on the Boot Device(s)

FreeNAS provides a separation between the boot device, to which the operating system is installed, and the disk(s) used for storage. This means that a problem with the boot device or the operating system itself does not affect the data stored on the storage disk(s), and that data becomes available once the problem with the boot device or operating system is resolved. Traditionally, FreeNAS formatted the boot device with UFS and did not support mirroring of the boot device. While it was easy to recover from a failed boot device, the operation did result in some downtime in order to realize the failure, prepare a new boot device, and boot into it.

Beginning with version 9.3, FreeNAS formats the boot device with ZFS and supports mirrored boot devices, allowing for the system to continue to operate if one of the boot devices in a mirror fails. The current sta-

tus and number of boot device(s) can be viewed in System -> Boot -> Status. Another boot device can be added as a mirror at any time by clicking the "Attach" button in the same screen and selecting the device to add, as seen in the example in Figure 1.



**Figure 1** Adding Another Boot Device

The use of ZFS provides an additional feature: boot environments, or the ability to take a snapshot of the operating system itself and to boot into a previous version of the operating system. This feature is integrated into the new update mechanism, described in the next section.

## Managing Updates

Beginning with version 9.3, FreeNAS no longer uses point releases to provide security patches, bug fixes, new drivers, or other types of updates. Instead, cryptographically

# FreeNAS 9.3

signed updates are provided as they become available, providing the administrator flexibility in determining when to apply the available updates. In addition, the update mechanism allows the administrator to track different “trains” or release/development branches. This allows the administrator to “test drive” upcoming releases or to apply a needed feature that is not yet available in the current STABLE branch, yet still roll back to the previous version of the operating system as needed.

Figure 2 provides an example of the System -> Update screen. In this example, the system is currently running the STABLE version of 9.3 and the following trains are available: FreeNAS-10-Nightlies (tracks the upcoming, not-yet-released 10 alpha version), FreeNAS-9.3-Nightlies (tracks nightly, possibly untested changes), and FreeNAS-9.3-STABLE (tracks tested updates).

Once a train is selected, the administrator can click the “Check Now” button to see if any updates are available for that train. Figure 3 shows an example of the results of an update check where several updates are available.

In this example, the numbers in the

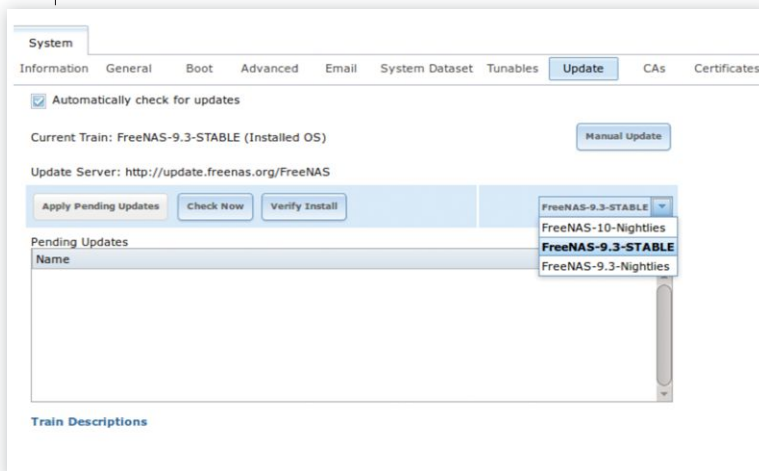


Figure 2 Selecting a Train from Update Manager



Figure 3 Reviewing Available Updates

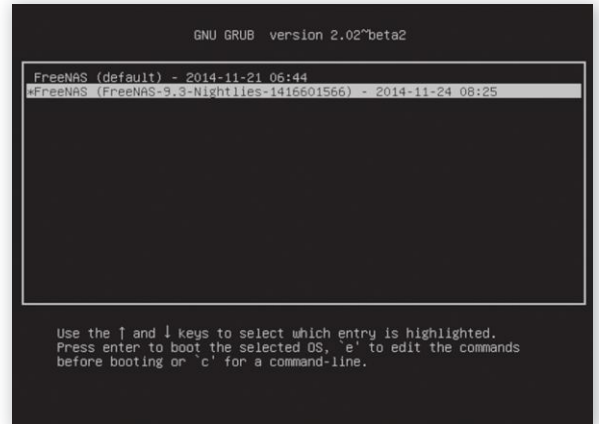


Figure 4 Selecting a Boot Environment from Boot Menu

Changelog that begin with a # represent the bug report number from bugs.freenas.org. Click the “ChangeLog” hyperlink to open the log of changes in a web browser. Click the “ReleaseNotes” hyperlink to open the 9.3 Release Notes in a web browser.

To download and apply the updates, click the “OK” button. Alternately, most updates require a reboot after they are applied. Uncheck the box “Apply updates after downloading.” This will instruct the system to only download the updates. The updates can then be applied at a time that least impacts users by clicking the “Apply Pending Updates” button shown in Figure 2.

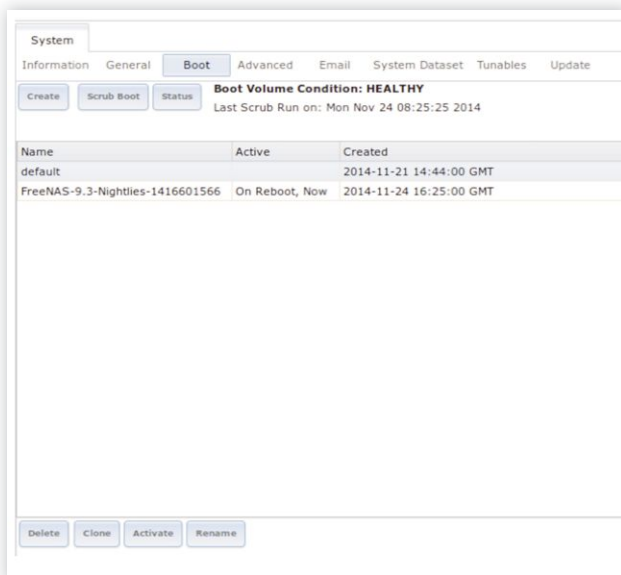
Whenever an update is applied, the system automatically creates a boot environment of the newly updated operating system and adds it as the default entry in the boot menu. In the example shown in Figure 4, the default version of the operating system was installed on November 21 and an update using the 9.3-Nightlies train was applied on November 24. Should an update fail, or the administrator wish to return to a previous version of the operating system, simply reboot and select the desired boot environment to boot into.

## Boot Manager

In addition to the automated boot environments created by the system updater, FreeNAS 9.3 includes a boot manager for creating manual boot environments and pruning old boot environments. This configuration screen, shown in Figure 5, can be accessed from System -> Boot.

This screen displays the status of the boot volume and the time and results





**Figure 5** Managing Boot Environments

of the last ZFS scrub of the boot volume. By default, the boot device is scrubbed every 35 days. Click the “Scrub Boot” button to initiate a scrub of the boot volume now. To review the number of devices in the boot volume and the status of each device, click the “Status” button.

If you highlight a boot environment, you can delete, clone, activate (if it is not currently set as the boot default), or rename it. If the boot volume is getting low on space or you have applied many updates, you can delete multiple older boot environments that you do not plan to boot into in the future. To instead create a new environment, click the “Create” button and input the name to appear in the boot menu.

## Configuration Wizard

Beginning with 9.3, FreeNAS provides a configuration wizard which runs automatically either after the initial upgrade to or installation of version 9.3. This wizard provides an efficient mechanism for quickly configuring the system in order to reduce the amount of time from initial bootup to serving data over the network. It allows the administrator to:

- Configure the system’s localization, keyboard mapping, and time zone.
- Import an existing ZFS pool or create a new ZFS pool.
- Select a directory service to attach to (Active Directory, LDAP, or NIS) and provide the required credentials.
- Configure CIFS (including guest access), AFP (including Time Machine), NFS, and iSCSI shares. Each share configuration should work “out of

the box” and can be further fine-tuned for more complex scenarios using the FreeNAS graphical administrative interface.

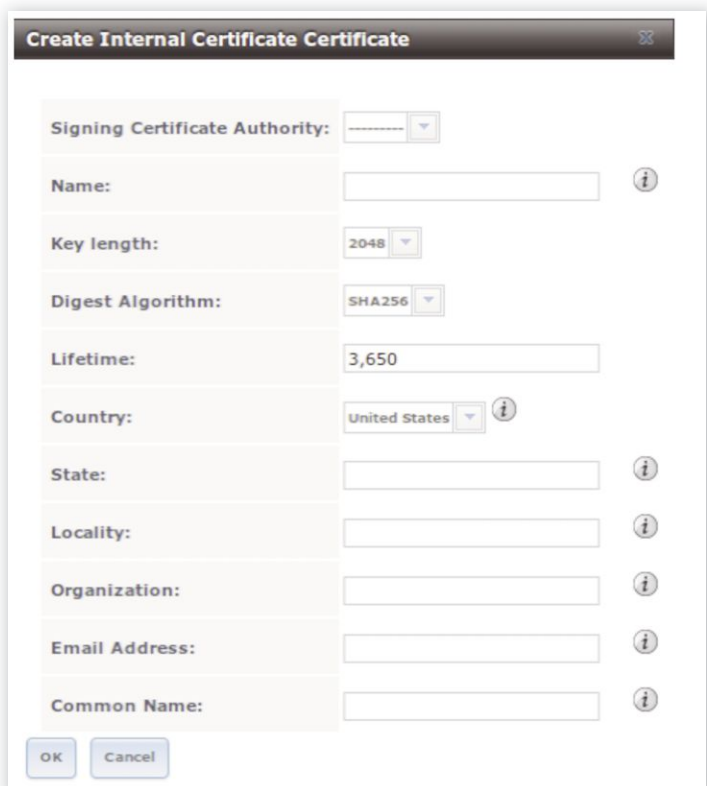
- Set the administrative email address which will receive security run outputs and administrative alerts.
- Configure the system to optionally display console messages at the bottom of the screen to ease troubleshooting.

The wizard can be rerun at any time, making it trivial to add additional shares or directory services. Any configuration performed by the wizard can still be viewed and edited using the configuration screens provided by the FreeNAS GUI.

## Certificate Manager

Many of the FreeNAS services support encryption using a certificate. FreeNAS 9.3 provides a graphical certificate manager for creating a certificate authority, importing and creating certificates, self-signing certificates, and creating certificate signing requests. This certificate manager is integrated into the system, meaning that all added certificates are available for use within the configuration screens of the services which support encryption.

Creating a self-signed certificate is as simple as filling in the information in the screen shown in Figure 6.



**Figure 6** Using Certificate Manager to Create a Self-Signed Certificate

# FreeNAS 9.3

Figure 7 provides an example of configuring the graphical administrative interface to use HTTPS. In this example, two certificates have been created, one for the web service and one for the ftp service. See Figure 7.

## Improved iSCSI/Virtualization Integration

The move to kernel iSCSI, with many performance improvements, and support for all of the VAAI (vStorage APIs for Array Integration) storage primitives greatly enhance FreeNAS integration with virtual datastores. VAAI is an API framework that enables certain storage tasks, such as thin provisioning, to be offloaded from the virtualization hardware to the storage array. The following VAAI primitives are supported in 9.3:

- **unmap:** tells ZFS that the space created by deleted files should be freed. Without unmap, ZFS is unaware of freed space made using a virtualization technology such as VMware or Hyper-V.
- **atomic test and set:** allows a virtual machine to only lock the part of the virtual machine it is using rather than locking the whole LUN, which would prevent other hosts from accessing the same LUN simultaneously.
- **write same:** when allocating virtual machines with thick provisioning, the necessary write of

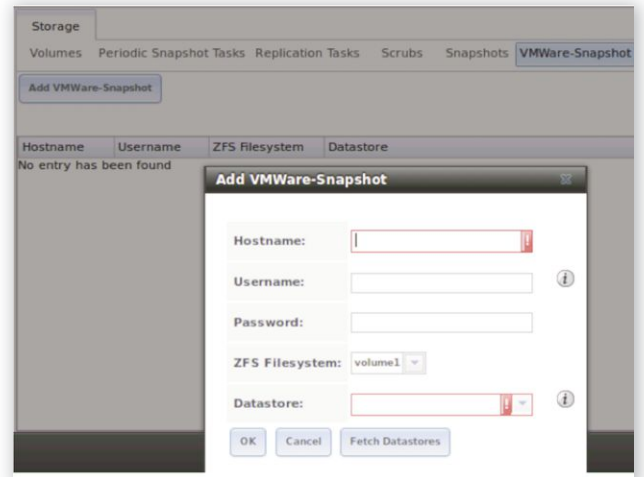


Figure 8 Configuring a VMware Datastore

zeros is done locally, rather than over the network, so virtual machine creation is much quicker.

- **xcopy:** similar to Microsoft ODX, copies happen locally rather than over the network.
- **stun:** if a volume runs out of space, this feature pauses any running virtual machines so that the space issue can be fixed before any corruption occurs.
- **threshold warning:** the system reports an error when a configurable capacity is reached. In FreeNAS, this threshold can be configured both at the pool level and the zvol (device extent) level.
- **LUN reporting:** the LUN reports that it is thin provisioned.

In addition, ZFS snapshots work correctly when VMware is configured as a datastore. FreeNAS will automatically snapshot any running VMware virtual machines before taking a ZFS snapshot of the dataset or zvol backing that VMware datastore. This means that the resulting ZFS snapshots will contain coherent VMware snapshots. A new configuration screen, found in Storage -> VMWare-Snapshot and shown in Figure 8, can be used to configure the datastore.

## Miscellaneous Features

Several other features have been added in 9.3, including:

- A new sharing type, WebDAV, provides authenticated access to the specified volume or dataset from a web browser or webdav client. Encryption or forced encryption can optionally be configured on these shares.
- Kerberized NFSv4 support is now available.
- The LLDP service has been added for providing Ethernet device discovery using IEEE 802.1AB.

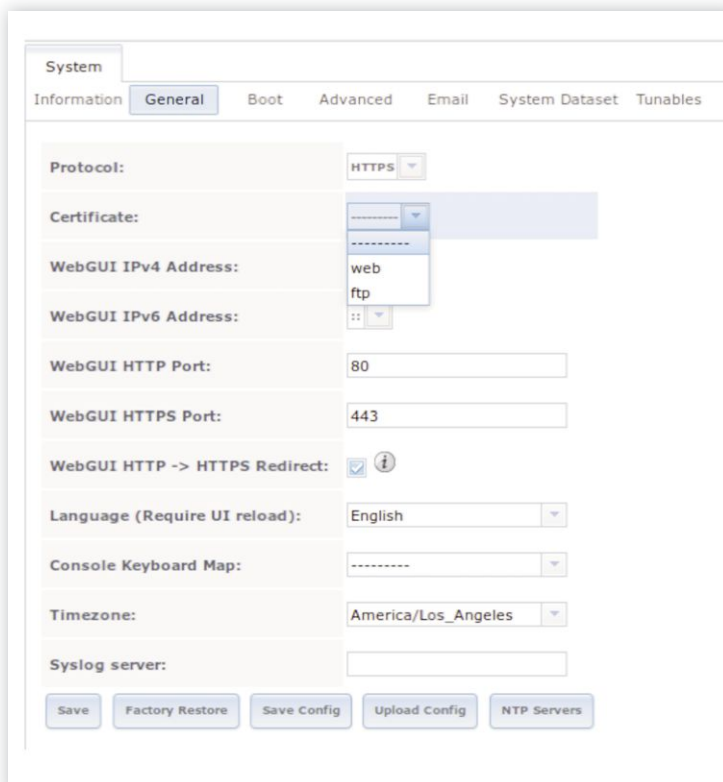


Figure 7 Selecting the Certificate to Use

- The addition of the SSSD service means that multiple directory services are supported.
- SNMP is now provided by Net-SNMP.
- The system logger has been replaced by syslog-ng.
- The ability to manage sysctls, loader.conf values, and rc.conf values has been integrated into one configuration screen (System -> Tunables).
- ZFS pool upgrades can now be performed using the graphical interface. The alert system will indicate when newer OpenZFS feature flags are available.
- Zvols can now be grown using the graphical interface. LUNs can now be grown without the need to first disconnect initiators or stop the iSCSI service.
- Kerberos realms and Kerberos keytabs can now be configured from the graphical interface and, once added, become available in the configuration screens for the directory services that support realms and keytabs.

## Summary

FreeNAS continues to add innovative features, making it easier than ever to configure this open-source operating system and integrate

it as a storage solution within any size network. Boot environments reduce the risk of updating the operating system, and the update mechanism provides flexibility when determining when to apply updates. The configuration wizard reduces the amount of time needed to deploy FreeNAS, and the certificate manager makes it easy to create and manage certificates.

This article did not cover all of the features and improvements added to FreeNAS 9.3. Refer to the 9.3 Release Notes and the “What’s New in 9.3” section of the FreeNAS 9.3 Users Guide for a more complete list of the new features. ●

---

**Dru Lavigne** has been using FreeBSD as her primary platform since 1997 and is the lead documentation writer for the FreeBSD-derived PC-BSD and FreeNAS projects. She is author of *BSD Hacks*, *The Best of FreeBSD Basics*, and *The Definitive Guide to PC-BSD*. She is founder and current Chair of the BSD Certification Group Inc., a nonprofit organization with a mission to create the standard for certifying BSD system administrators, and serves on the Board of the FreeBSD Foundation.

---

# RootBSD

## Premier VPS Hosting

RootBSD has multiple datacenter locations, and offers friendly, knowledgeable support staff. Starting at just \$20/mo you are granted access to the latest FreeBSD, full Root Access, and Private Cloud options.



[www.rootbsd.net](http://www.rootbsd.net)





**INTERACTING** with the FreeBSD Project

# HOW TO Build a Port

**by Erwin Lansing** / FreeBSD has long been known for its ports tree, where users had to start by compiling all the software they wanted to install, which in some cases can be slow and bothersome. Over the last few years, a lot of work has been done to build a first-class packaging system, where a user can simply download and install a pre-built package, with entirely new packaging tools and also a brand new package distribution infrastructure. For most users, this is now the recommended way to install software, and most users will no longer need to compile software themselves if using the new pkg(8) system (see *FreeBSD Journal*, March/April 2014). Underneath it all, however, is still the FreeBSD ports system, which has all the information on how to build and install third-party software on FreeBSD. With almost 25,000 packages available, it still may happen that a package does not exist, and so a new port will need to be written. Similarly, to change an existing package, the underlying ports need to be modified. In this article I give a brief overview of how the ports system is structured and an introduction on how to build a simple port.



First you need to get a fresh copy of the FreeBSD ports tree. The fastest way to get one is by using portsnap(8). This is a two-step process. First fetch the portsnap data file, which is about 70Mb and can take some time depending on your Internet connection. Then install the ports tree itself, which will extract a large number of small files, which also can take time depending on the speed of your disks.

```
portsnap fetch
portsnap extract
```

This will place a full copy of the ports tree in /usr/ports. Updating an earlier version of the ports tree installed by portsnap is a similar two-step process:

```
portsnap fetch
portsnap update
```

Looking in /usr/ports you can see a number of files and a large number of directories. One of the first files that should be noticeable is the Makefile. The FreeBSD ports tree is written in make(1), a language extensively used for building and installing software. Of course a single 200

line Makefile is not enough, and most of the ports infrastructure code is in the `/usr/ports/Mk` directory. Most of the other directories are the categories containing the actual ports.

A very simple port consists of four files: Makefile, `pkg-descr`, `pkg-plist`, and `distinfo`. The Makefile contains some basic information about the port and the instructions on how to fetch, build, and install it. `pkg-descr`, or package description, contains a short description of what this port is and does and also a link to the official project website of the software package. `pkg-plist`, or packaging list, is a list of files that this port installs, and finally, `distinfo` contains a checksum and size of any external files this port needs to fetch from the Internet. I'm going to use the Stand-Alone Shell (SASH) port, found in `/usr/ports/shells/sash`, as an example here. In Example 1 you can also see a directory named `files`, which can contain additional files useful for this port.

Looking at the SASH Makefile in Example 2, the first line has an attribution to the original author when the port first was added to the ports tree. Next is the subversion identification string, which will be expanded by subversion automatically and for a new port should only be:

```
# $FreeBSD$
```

Next is a section of variables with some essential information about the software this port will install. The name of the port in `PORTNAME` should be unique across the whole ports tree, although there are options available for adding a prefix or suffix to the name.

**PORTVERSION** is the version of this port and is usually the same as the version of the original software package. Note that the ports version can never decrease, as the tools for upgrading installed packages cannot handle versions going back in time.

**CATEGORIES** is a list of port categories starting with the primary category this port belongs to, which is also the directory in the ports tree it is in. A port can belong to more than one category listed after the primary category including virtual categories that do not have a directory themselves, but can still help users find the port. A full list of available categories is available in the Porters Handbook: [https://www.freebsd.org/doc/en\\_US.ISO8859-1/books/porters-handbook/makefile-categories.html#porting-categories](https://www.freebsd.org/doc/en_US.ISO8859-1/books/porters-handbook/makefile-categories.html#porting-categories).

### Example 1:

```
erwin@panda:/usr/ports/shells/sash % ls -l
total 20
-rw-r--r--  1 root  wheel  349 Apr 11  2014 Makefile
-rw-r--r--  1 root  wheel  123 Apr 11  2014 distinfo
drwxr-xr-x  2 root  wheel  512 Jun 24  2014 files
-rw-r--r--  1 root  wheel  458 Jan 22  2014 pkg-descr
-rw-r--r--  1 root  wheel   35 Jun 11  2014 pkg-plist
```

### Example 2:

```
erwin@panda:/usr/ports/shells/sash % cat Makefile
# Created by: Patrick Gardella <patrick@FreeBSD.org>
# $FreeBSD: head/shells/sash/Makefile 350947 2014-04-11 13:41:06Z miwi $

PORTNAME=          sash
PORTVERSION=       3.8
CATEGORIES=shells
MASTER_SITES=      http://members.tip.net.au/~dbell/programs/

MAINTAINER=        ports@FreeBSD.org
COMMENT=           Stand-Alone Shell combining many common utilities

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

The list of URLs in `MASTER_SITES` is used by the ports system to fetch the original sources of the software package. It is recommended to have more than one in case that site is temporarily unreachable. This is also where some of the magic starts. The actual URL of the file to be fetched is constructed by the ports system automatically, based on some of the earlier specified variables. In its simplest form it will default to

```
${MASTER_SITE}/${PORTNAME}-
${PORTVERSION}.tar.gz
```

In the example SASH port, this will be:  
`http://members.tip.net.au/~dbell/programs/sash-3.8.tar.gz`

Lots of software projects use their own versioning or naming schemes, and there are a large number of additional variables available to influence the final URL, e.g., setting `USE_BZIP2` will change the default suffix to `.tar.bz2`, and `USE_XZ` will change it to `.tar.xz`. There are also macros available for well-known download sites with a large number of mirrors, like SourceForge and CPAN. Another interesting and extensively used site is github, which does not have a concept of official released files, and so the port will need to depend on a given commit and tag to ensure it downloads a consistent and tested version of the software. These are all described in the Porters Handbook: [https://www.freebsd.org/doc/en\\_US.ISO8859-1/books/porters-handbook/makefile-distfiles.html](https://www.freebsd.org/doc/en_US.ISO8859-1/books/porters-handbook/makefile-distfiles.html).

---

The next section contains some basic metadata about the port. The FreeBSD ports system has a concept of maintainership, where a single person or multiple people behind a mailing list address are the primary contact for the port. Requesting changes to a port, like updating to a newer version, should be approved by the maintainer, and filing a bug report in the FreeBSD bugzilla instance will automatically generate an email to the address listed in the MAINTAINER field.

**COMMENT** is, as you may have guessed, a short, one-sentence description of the port. A longer description is available in the pkg-descr file.

Finally, the port needs to invoke the ports infrastructure itself by including the main file:

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

### Example 3:

```
erwin@panda:/usr/ports/shells/sash % cat pkg-descr  
SASH (Stand-Alone Shell)
```

It is a nice combination of bare-bones shell and a dozen or so most useful UNIX commands.

Shell includes: echo pwd cd mkdir mknod rmdir sync rm chmod  
chown chgrp touch mv ln cp cmp more exit  
setenv printenv umask kill where

Commands include: dd ed grep gzip ls tar file find mount chattr  
WWW: <http://members.tip.net.au/~dbell/>

Speaking of the pkg-descr file, Example 3 lists the contents of pkg-descr for the shells/sash port, which is quite explicit and detailed. A usual pkg-descr should only have a few paragraphs concisely describing the port so a user knows what the port does without having to read documentation or visit a website. The official website of the software project can be included on the last line, prepended by WWW.

### Example 4:

```
erwin@panda:/usr/ports/shells/sash % cat pkg-plist  
@shell bin/sash  
man/man1/sash.1.gz
```

The files installed by the port are listed in the pkg-plist file. This list is used at several stages in a port's lifetime. The tools that generate packages use this list to know what files to include, and a tool removing an installed port or package also needs to know which files it should delete. Files are installed in `${PREFIX}`, which defaults to `/usr/local`, is implicitly assumed in the pkg-plist file,

and should not be included. As you can see in Example 4, our small port only installs one executable file and its corresponding man-page. More strengths of the ports system turn up here. Keywords can be used in the packaging list to handle more advanced cases where just installing a given file in a given location needs more care, or where an additional file needs to be modified. The `@shell` keyword in our example not only installs the sash executable in `/usr/local/bin`, but also adds it to `/etc/shells` so that it becomes a recognized shell on the system. Another example is the `@sample` keyword. This is very useful for installing configuration files that should not be overwritten by the stock version in a later upgrade. Using the `@sample` keyword, the port or package will install the stock version as `filename.conf.sample`, but before that it will test whether `filename.conf` exists, and if so, whether it is the same as the stock file in the previous version, and if not, or if it doesn't exist, install the stock version as `filename.conf` as well. In other words, if the configuration file already exists and was modified by the user since its installation, those modifications will not be overwritten. More keywords can be found in the `/usr/ports/Keywords` directory.

---

Finally we get to the distinfo file. This is also the easiest file to create. If all the variables in the Makefile were correctly filled in, especially those that influence how to fetch the original sources, it is just a matter of running:

```
make makesum
```

This will fetch the files needed and generate the distinfo file with a SHA256 checksum and the size of the file. See Example 5.

The files directory in a port can hold any file the port needs at some point during the build, installation, or packaging. Filenames starting with the prefix `patch-` will automatically be applied between unpacking the original sources and compilation and can be very useful when porting software written without FreeBSD in mind. See Example 6.

For a new port, an entry in the category Makefile needs to be created. In our example, that would be in `/usr/ports/shells/Makefile`

```
SUBDIR += sash
```

## Dependencies

One important and often used feature our example port does not use is dependencies. A dependency is another port that the current port needs in some phase of its lifetime, for example, an executable or other file during run time, or while building, or a library. Libraries are specified in the `LIB_DEPENDS` variable as a list of library:port combinations.



### Example 5:

```
erwin@panda:/usr/ports/shells/sash % cat distinfo
SHA256 (sash-3.8.tar.gz) =
13c4f9a911526949096bf543c21a41149e6b037061193b15ba6b707eea7b6579
SIZE (sash-3.8.tar.gz) = 53049
```

### Example 6:

```
erwin@panda:/usr/ports/shells/sash % ls -l files/
total 20
-rw-r--r--  1 root  wheel  1356 Apr 11  2014 patch-Makefile
-rw-r--r--  1 root  wheel   442 Jan 22  2014 patch-cmd_ls.c
-rw-r--r--  1 root  wheel  2572 Apr 11  2014 patch-cmds.c
-rw-r--r--  1 root  wheel   551 Apr 11  2014 patch-sash.c
-rw-r--r--  1 root  wheel   219 Jan 22  2014 patch-sash.h
```

### Example 7:

```
erwin@panda:/usr/ports/shells/sash % make describe
sash-3.8/usr/ports/shells/sash/usr/localStand-Alone shell combining many common
utilities/usr/ports/shells/sash/pkg-descrports@FreeBSD.orgshellsllllllhttp://members.tip.net.au/~dbell/
```

```
LIB_DEPENDS=
libsqlite3.so:${PORTSDIR}/databases/sqlite3
Run time or build time dependencies are specified in a similar way in the RUN_DEPENDS and BUILD_DEPENDS variables, consisting of a list of file:port combinations.
```

```
RUN_DEPENDS=
${LOCALBASE}/bin/bash:${PORTSDIR}/shells/bash
The ${LOCALBASE} variable is usually the same as ${PREFIX} we saw earlier, but make the distinction between where already installed ports are (LOCALBASE) and where the current port will be installed (PREFIX). For an executable in the default path, the full path is not needed, so the above example could be shortened to:
```

```
RUN_DEPENDS=
bash:${PORTSDIR}/shells/bash
```

## Testing the Port

It is highly recommended to turn on the developer mode while testing. This is easily done by setting `DEVELOPER=yes` in `/etc/make.conf`:

```
# echo DEVELOPER=yes >> /etc/make.conf
```

This turns on extra quality checks and displays more warnings, like use of deprecated features.

A first and simple test that can be done is running “make describe.” This should produce a string composed of a lot of the metadata we talked about earlier. This information is used by several tools for keeping track of dependencies and searching for port names or descriptions. See Example 7.

The `portlint` utility, found in `ports-mgmt/portlint` or just “pkg install portlint,” evaluates whether the port is syntactically correct and also has some rec-

### Example 8:

```
erwin@panda:/usr/ports/shells/sash % portlint -C
WARN: Makefile: Consider defining LICENSE.
0 fatal errors and 1 warning found.
```

### Example 9:

```
LICENSE=      GPLv2
LICENSE_FILE=  ${WRKSRC}/COPYING
```

ommendations for best practices that have developed over the years. For a new port, `portlint -A` should be run; for an existing port `portlint -C` is sufficient. See Example 8.

This is a valid warning as our example port does not set the `${LICENSE}` variable. The `${LICENSE}` variable can be set to a known license abbreviation, listed in `/usr/ports/Mk/bsd.licenses.db.mk`. If a software project includes a file with its license, it should also be installed by using the `${LICENSE_FILE}` variable. See Example 9.

## Testing a Port

The easiest way to test is, of course, to install the newly hatched port on your local system. I would recommend against that, as it can easily break the existing system, accidentally overwriting an existing file or exhibit other unexpected behavior, and already installed software can also make it harder to check what the new port adds. For example, with plenty of software already installed, it is easy to miss a newly installed file and leave it out of the

pkg-plist.

Using a clean installation or a jail might help avoid most of those pitfalls, but tools are available to do all the hard work for you. The poudriere system (<https://github.com/freebsd/poudriere>) is not only useful for building your own in-house package repository, but is also an excellent port testing framework. It can be used to test the entire ports tree, a subset of the tree, or even a single port and its dependencies. Using jails and the ZFS file system, it ensures that it is self-contained and does not influence the host system in any way. It generates very useful log files on a per-port basis, that can be used to see how the port was built, packages installed and deinstalled, and is very useful for checking whether the pkg-plist file was complete. It can be a bit hardware hungry due to its heavy reliance on ZFS, and of course compiling does require a lot of processing.

Install poudriere from a package “pkg install poudriere” or from the port in ports-mgmt/poudriere, and review the settings in /usr/local/etc/poudriere.conf, especially those dealing with the ZFS file system. Next, create a jail with the FreeBSD version you want to test:

```
# poudriere jail -c -j 93Ramd64 -v 9.3-RELEASE -a amd64
```

If you plan to submit your port to be included in the FreeBSD ports tree, you should test on all supported major releases. The ports team supports the same FreeBSD releases as the FreeBSD security officer (<https://www.freebsd.org/security/>), which currently includes FreeBSD 8, 9, and 10, plus the development version in HEAD, 11.

Of course you need a ports tree as well, which can be created by:

```
# poudriere ports -c
```

This will create a ports tree named default.

You’re now ready to test your port:

```
# poudriere testport -j 93Ramd64 -p default -o shells/sash
```

You will find the log file in /poudriere/data/logs/bulk/93Ramd64-default/latest/logs/sash-3.8.log. A more extensive intro-

duction for using poudriere as a testing tool can be found in the Porters Handbook:

<https://www.freebsd.org/doc/en/books/porters-handbook/testing-poudriere.html>.

At the time this article was written, the redports (<https://redports.org>) system was down for maintenance. Hopefully, the site will be back, so keep checking. It provides all the tests above and a large number of extended checks, all via a convenient web interface.

## Submitting a Port

Once you are satisfied the port works the way you want, you are ready to submit it for inclusion in the official FreeBSD ports tree, which will also make packages available for easy installation with pkg. Make sure the port directory is clean of any unnecessary files. Removing the work/ directory can easily be done with the “make clean” command. From the category directory, one level up from the port directory, use the shar(1) utility to bundle the files in a shar archive:

```
shar `find sash`> sash.shar
```

The resulting sash.shar can be submitted to the FreeBSD bugzilla database.

We have only brushed the surface in this introduction to the ports tree, so be sure to check the Porters Handbook (<https://www.freebsd.org/doc/en/books/porters-handbook/index.html>), which is constantly updated and contains comprehensive information on all the features of the ports tree. There is also an active porters community that will be happy to answer any questions. You can find them on the freebsd-ports mailing list (<http://lists.freebsd.org/mailman/listinfo/freebsd-ports>) or on the IRC channel #bsdports on EFnet. Happy porting! ●

---

**Erwin Lansing lives in Copenhagen with his wife and son. He works for DK Hostmaster trying to keep the lights on on the Danish Internet. He is also Vice President of the Board**



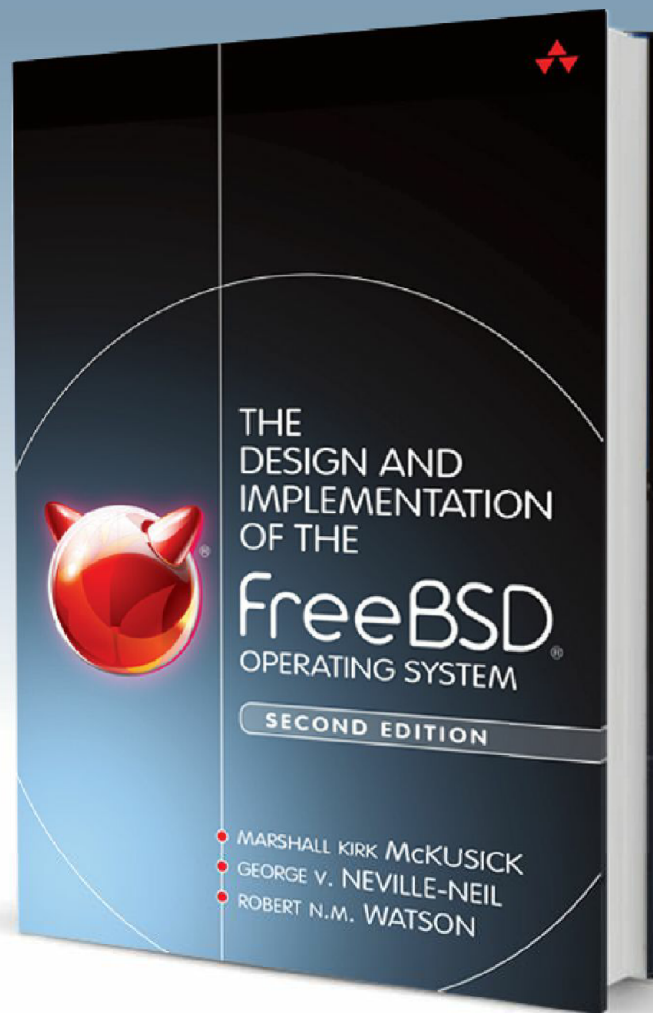
# We Can't Do This Without YOU!

Your contribution makes a real difference! Help the FreeBSD Foundation Support: ●Project Development ●FreeBSD Advocacy ●Growth of the FreeBSD Journal ●And More!



**PLEASE DO YOUR PART & DONATE TODAY**

NEW EDITION – Now Available!



The most complete, authoritative technical guide to the FreeBSD kernel's internal structure has now been extensively updated to cover all major improvements between Versions 5 and 11.



**SAVE 35%**

When you order from [informit.com/freebsd](http://informit.com/freebsd)  
Use discount code **FREEBSD35** during checkout

EBOOK FORMATS INCLUDE EPUB, MOBI, AND PDF – ALL FOR ONE PRICE • FREE SHIPPING WITHIN THE U.S.

Terms & Conditions: Discount code FREEBSD35 is applied to list price of Design and Implementation of FreeBSD, Second Edition print or eBook and cannot be combined with any other offers. Offer is only good at informit.com.

**informit.com**  
the trusted technology learning source

  
Addison  
Wesley





# The FreeBSD Foundation at 15

**by Justin Gibbs** / Some 15 years later, the FreeBSD Foundation continues to grow the ways it supports the FreeBSD community. From funded development projects and promotion of FreeBSD, to providing legal support and bringing our community together at conferences and summits, the FreeBSD Foundation compliments the hard work of FreeBSD's contributors, focusing on areas that are underserved in an all volunteer project.



We were not always so capable, but during our 15 years the vision to get us here has not changed. By telling the story of our origin—why I felt the FreeBSD Foundation had to exist—I hope to better explain our mission and how we'll continue to move forward in the next 15 years. The story begins during the early days of the FreeBSD Project.

I started my involvement with FreeBSD on a purely technical level. Filled with curiosity and an appetite for learning, I saw FreeBSD as an incomplete puzzle. Creating new pieces and putting them in place was addictive; finding out that my solutions worked for others, immensely satisfying. I had stumbled upon a technical playground where I could share my ideas and build great things without the impractical schedules and other constraints I found when writing software for school or work.

The founders of this "FreeBSD playground" may not have realized exactly what they were creating. The goal was to carry on the BSD tradition, challenge ourselves to create the "best UNIX," and to have fun. But as the project grew, it became clear that someone had to be responsible for the playground: the grass needed to be cut, the play equipment maintained, and the occasional fight broken up. To fill this role, without much desire or inclination to be managers, the founders and a few of the early technical contributors to the project came together to form the FreeBSD Core Team.

Late in 1993, I was invited to join the

FreeBSD Core Team. In those days, this was how the project acknowledged technical expertise—being on the Core Team lent weight to your arguments on technical matters. It was a tremendous honor and one I readily accepted. What I didn't fully grasp was that I was accepting a management position. Lacking the skill set or natural talent for "cat herding," I was now one of a handful of volunteers—hackers really—responsible for guiding FreeBSD forward.

The next seven years that I served on the FreeBSD Core Team was an amazing experience. Reluctant volunteer managers attempting to manage a volunteer community may sound like a recipe for disaster. However, it worked surprisingly well. We were young, fiercely loyal, and had enviously large amounts of spare time to dedicate to the project. With enough effort and sheer will, there is almost nothing you cannot do. But it became clear to me that as a group of technophiles, many of the things we had to do were things we didn't enjoy doing. Making the project function and grow took time away from the coding projects I loved. For me, this wasn't sustainable.

In 1999, two years into a serious relationship, my thoughts turned to marriage and raising a family. It was obvious that my time commitment to the FreeBSD Project would have to change, but I didn't want to give any less to a community that had given me so much in return. I also wanted to really enjoy the time that I chose to give to the project.

That same year, the FreeBSD Core Team was looking for a way to accept the transfer of the

FreeBSD trademark from Walnut Creek CDROM. FreeBSD is a collection of individuals, not a formal legal entity. We needed a structure for safeguarding the intellectual property of the project. In researching how a nonprofit company might fit this role, I found a potential solution to both problems. A “FreeBSD Foundation” could certainly hold the trademark. It could also be a sustainable way for me to give back to the project in ways I could never achieve as a single volunteer.

On March 15, 2000, the FreeBSD Foundation was born. The combination of a tiny budget, piles of paperwork, filing and legal fees, and letters to and from the IRS made for a slow and inauspicious beginning. During some particularly frustrating times, it was hard to believe my original goals were attainable. The FreeBSD Foundation still needed me for constant care and feeding. I rarely wrote code for FreeBSD in my spare time. I looked forward to the day when the Foundation would be able to bring in expertise from outside the Project to help with marketing and legal issues, while freeing volunteers to work on the things they loved.

In December 2000, the Foundation received provisional, 501(c)(3) nonprofit status. The first major hurdle had been cleared. Jumping those that followed became progressively easier: learning the rules of maintaining nonprofit status, setting up to legally receive donations, figuring out basic accounting. Finally, on June 27, 2001, the FreeBSD Foundation was publicly announced to the world.

<http://docs.freebsd.org/cgi/getmsg.cgi?fetch=6629+0+archive/2001/freebsd-announce/20010701.freebsd-announce>

Now the real work began to transform the Foundation from a mere paper entity into an asset to the FreeBSD community. We took ownership of and started to protect the FreeBSD trademark. We worked with Sun to license FreeBSD Java binaries. We funded the early work on network scalability for SMP systems. We fostered nascent BSD conferences. These early accomplishments proved the value of the FreeBSD Foundation. But it wasn't yet sustainable.

Prior to 2006, all of the Foundation's activities were managed by volunteers. Jonathan Bresler and John Polstra served as officers and provided tremendous amounts of support during the early life of the FreeBSD Foundation. Over time, more volunteers agreed to participate on the Foundation's board and help grow its capabilities. However, the Foundation's address was still my house. The Foundation's phone number still rang my phone. I had a day job and family to care for. The Foundation needed someone

to manage it on a daily basis or it would never reach its full potential.

Having no idea where to start, I asked the HR director at my work for advice on hiring the FreeBSD Foundation's first employee. Kathy Stoltz introduced me to her longtime friend, Deb Goodkin. Deb has been serving the FreeBSD Foundation, now as its executive director, ever since. I feel very lucky to have found such a capable steward for the FreeBSD Foundation. More importantly, she has worked with the Foundation board to create policies and a structure to ensure, as volunteers, board members, and staff come or go, that the FreeBSD Foundation will survive and continue its work supporting the FreeBSD Project.

This journey has taught me several things. FreeBSD is about much more than writing documentation or code and building a system. As a community, we need to understand that, and seek creative ways to achieve the goals we aren't best equipped to tackle as volunteers. The work we enjoy doing can have a tremendous impact on our world. But for that to happen, we must promote it, while making it easier to use and more accessible to researchers, educators, and builders of commercial products. The FreeBSD Foundation is a vehicle for performing this necessary work—for meeting challenges, in cooperation with the FreeBSD community, that seem impossible for the FreeBSD Project to conquer on its own. This is why I have invested so much to make the FreeBSD Foundation a reality.

The journey is not over. The work is not done. I want to thank the FreeBSD Foundation staff, former and current board members, and our donors for making our achievements possible. To the many users of FreeBSD we've talked to, thank you for your words of encouragement and your challenges to do even more.

Finally, as founder and president of the FreeBSD Foundation, I want to thank the FreeBSD community for your trust. Holding the FreeBSD trademark and supporting the FreeBSD Project is a privilege. The FreeBSD Foundation will continue to grow and the faces of those running it may change, but our commitment to this community and keeping your trust will always remain the same. ●

---

**Justin Gibbs is the founder and president of the FreeBSD Foundation, and has been working on the storage-related subsystems of FreeBSD since 1993. He currently works at Spectra Logic Corporation building petabyte-scale, archive storage systems using FreeBSD, flash, disk, and tape.**

# PORTSreport

by Frederic Culot

The new year started with sustained activity on the PR front, culminating in February with more than 600 problem reports closed! Thanks to all contributors and developers who helped make this happen. Regarding the ports tree, January and February saw a bit more than 4,000 commits in total. That's not bad, but at this rate last year's record with more than 37,500 commits is still standing, so any volunteer interested in working on ports is more than welcome to join our ranks in 2015!

the creator of redports.org, QAT, and the developer who ported and maintained VirtualBox for FreeBSD. Many thanks to him for all his hard work. For anyone who would like to know more about Bernhard you can find his interview at <http://blogs.freebsdish.org/portmgr/2013/10/29/getting-to-know-your-portmgr-bernhard-froehlich/>.

## IMPORTANT PORTS UPDATES

Several exp-runs were performed (23 actually) to check whether major ports updates are safe or not. From those important updates we can mention the following highlights:

- default python3 version set to 3.4
- default ruby version set to 2.1
- gcc updated to 4.9
- clang updated to 3.6.0
- cmake updated to 3.1.3
- ruby-gems updated to 2.4.5

As usual, please read the `/usr/ports/UPDATING` file carefully before updating your ports, as manual steps may be involved.

## NEW PORTS COMMITTERS AND SAFEKEEPING

In January and February two new committers were granted ports commit bits: Jan Beich, who is mentored by bap@ and flo@, and Brad Davis, who already owned a doc commit bit and who is mentored by bdrewery@, swills@, and zi@.

Only one commit bit was taken in for safekeeping during the last two months (rafan@), but we also had some sad news with decke@, who decided to step down from his duties at FreeBSD to focus on his family and professional life. decke@ has been invaluable to the FreeBSD Project, being

And to close this issue's column, we would like to give a few stats so that people can get a better idea of the outstanding amount of work performed by our volunteers. In the first two months of 2015, 4,046 commits were applied to the ports tree, 1,182 PR were closed, and 1,002 emails were received by portmgr@ (without counting the spam...). And all this handled by an average of only 130 active ports developers!

Frederic Culot has worked in the IT industry for the past 10 years. During his spare time he studies business and management and just completed an MBA. Frederic joined FreeBSD in 2010 as a ports committer, and since then has made around 2,000 commits, mentored six new committers, and now assumes the responsibilities of portmgr-secretary.

## EVENTS

There was an important event (FOSDEM) in Europe that brought together major players in the open-source world. During this event bap@ delivered an interesting talk about the history of pkg(8), which has become the main tool for installing third-party software on FreeBSD. Slides can be found online at <https://fosdem.org/2015/schedule/event/4yearofpkg/>.





We're excited to announce that March 15, 2015 marks the **15th Anniversary** of the FreeBSD Foundation!

You've helped up accomplish so much during the last 15 years and we look forward to continuing that progress through out the rest of 2015. The areas we'd like to invest in include the following:

- 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
- And More!

Thank you for all of your continued support. We can't do it without you!



# Support FreeBSD®

## Donate to the Foundation!

FreeBSD is internationally recognized as an innovative leader in providing a high-performance, secure, and stable operating system. Our mission is to continue and increase our support and funding to keep FreeBSD at the forefront of operating system technology.

For 15 years, the FreeBSD Foundation has been proudly supporting the FreeBSD Project and community thanks to people like you. We are incredibly grateful for all the support we receive from you and so many individuals and organizations that value FreeBSD.

Make a gift to support our work in 2015. Help us continue and increase our support of the FreeBSD Project and community worldwide!

Making a donation is quick and easy.  
Go to [freebsd.foundation.org/donate](http://freebsd.foundation.org/donate)



The  
**FreeBSD**

FOUNDATION

[freebsd.foundation.org](http://freebsd.foundation.org)

# this month

In FreeBSD

BY DRU LAVIGNE

This is the time of year when open-source mentoring organizations review proposals from students looking to participate in **Google's Summer of Code**. This month, we take a look at the FreeBSD Project's participation in the program.



Since 2005, Google has offered their annual Summer of Code (<http://www.google-melange.com/gsoc/homepage/google/gsoc2015>), a program that provides student developers from across the globe a three-month stipend to write code for a mentoring, open-source software project. This program provides several benefits to both the student and the mentoring organization. In addition to being paid, students learn how to write a proposal of work and to achieve timed milestones for that work. Their project is mentored by an experienced developer who can assist in code review, time organization, and learning that project's tools, communication channels, and coding standards. The mentoring organization has the opportunity to nurture new ideas and integrate new features into their codeset with the possibility of training new developers for the project.

The FreeBSD Project has the distinction of being a mentoring organization for every Google Summer of Code since the program's inception in

2005. This means that 2015 marks the 11th year of participation for the FreeBSD Project. As of this writing, student proposals are still being reviewed so we do not yet know the names of the 2015 student participants.

We can, however, take a look at the last 10 years of participation. Each Summer of Code student, mentor, project, and a summary of that project's results can be found at <https://www.freebsd.org/projects/summerofcode.html>. In addition to an overview of the process, that page contains a hyperlink for each year of participation.

Over the 10-year period, the FreeBSD Project has mentored a total of 136 students. Google Summer of Code allows students who are interested and still eligible for the program to reapply. For FreeBSD, the breakdown of student participation is presented below:

#### 136 students (total)

111 students .....	participated for 1 year
18 students .....	participated for 2 years
3 students .....	participated for 3 years
3 students .....	participated for 4 years
1 student.....	participated for 6 years

This level of participation, an average of over 13 students per year, reflects the importance to the FreeBSD Project of cultivating new ideas and attracting new developers as each student requires a time commitment of three months from a mentor. It is also encouraging that 25 students were both eligible and chose to reapply for subsequent years.

Google Summer of Code has received some bad press about students who do not complete their projects, do not have their code integrated into the project's codebase, or who never participate again with the project once their Summer of Code finishes. While not every Summer of Code student works out, the FreeBSD Project has had good success in cultivating relationships, integrating student code, and gaining new developers.

The FreeBSD Project uses a "commit bit" process to denote contributors who have the ability to commit directly to a FreeBSD source repository. Commit bits are granted to active contributors, and new committers have their commits reviewed by several mentors before they are committed. Commit bits are also retired should the committer no longer have time to contribute, typically after a year of inactivity. The FreeBSD Project defines three types of commit bits, indicating to which source repository the

contributor has commit access: src, ports, and doc.

Out of the 136 Google Summer of Code participants, 20, or nearly 15%, were granted commit bits. Some were granted multiple commit bits, which reflects on their continued level of participation within the Project. Out of the 20 committers over that 10 year period, 17 are still active committers. The breakdown is as follows:

**20 committers (total)**  
16 committers with 1 commit bit  
3 committers with 2 commit bits  
1 committer with 3 commit bits

The granted commit bits were for the source repositories below:

17 src committers, with 3 retired (out of a total of 314 src committers)  
6 ports committers (out of a total of 192 ports committers)  
2 doc committers (out of a total of 41 doc committers)

In addition to commit bits, some participants also became active in other areas. Several former Summer of Code students became mentors for later Summer of Code students. One participant became a member of the FreeBSD Doc Engineering Team and another is secretary for the FreeBSD Core Team.

While it is too late to apply for the 2015 Google Summer of Code, interested students should follow the Google Open Source blog (<http://google-opensource.blogspot.com/>) to keep up-to-date on the 2016 proposal period, which should take place next March. The

FreeBSD Project provides several idea lists to help students narrow down their project ideas:

- <https://wiki.freebsd.org/IdeasPage>
- <https://wiki.freebsd.org/SummerOfCodeIdeas>
- <https://wiki.freebsd.org/JuniorJobs>

Note that these ideas are not limited to Google Summer of Code students, which is good news for those of us who do not meet the age or student requirements of that program, but who are also interested in contribution ideas. Each idea provides a technical contact who is willing to act as a mentor for that idea. •

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



Ottawa, Canada

**THE TECHNICAL BSD CONFERENCE.**  
The BSD Conference held in Ottawa, Canada, has quickly established itself as the technical conference for people working on and with 4.BSD based operating systems and related projects. The organizers have found a fantastic formula that appeals to a wide range of people from extreme novices to advanced developers.



**BSDCAN 2015**  
**COME JOIN US AT THE 12TH ANNUAL BSDCAN!**  
**WHERE** .....Ottawa, Canada  
 University of Ottawa  
**WHEN** .....Thurs. & Fri. June 10-11 (tutorials)  
 Fri. & Sat. June 12-14 (conference)

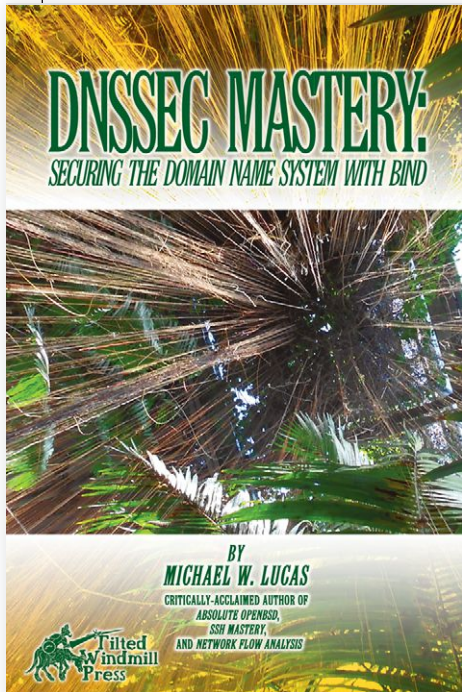
**BSD Certification • Courseware DVD • Register for an Exam**  
Join the growing ranks of people taking the BSDA exam. If you missed it in the past, now is your chance to catch up.

**GO ONLINE FOR MORE DETAILS**  
<http://www.bsdcn.org>



# BOOKreview

by Joseph Kong



## ***DNSSEC Mastery***

Michael W. Lucas

2013 Tilted Windmill Press • ISBN 978-1484924471 • 130 pages

I'll admit right away that I'm probably not the target audience for this book, because I never have and, hopefully, never will have to secure the Domain Name System (DNS) with Domain Name System Security Extensions (DNSSEC). However, being a security professional, I read Michael W. Lucas's *DNSSEC Mastery* to gain some familiarity with DNSSEC.

This book is right on target and does not waste your time. If you understand DNS and want to learn how to secure it, *DNSSEC Mastery* is for you. It's a clear and concise guide with a ton of handholding and plenty of examples. Also, at a little over 100 pages in length, it can be read in one sitting (I know, because I did just that).

Using his own domains, Lucas describes in detail how DNSSEC works, how to set it up, how to debug it when things go wrong (as they inevitably do), and how to maintain it. There's even a chapter on how to use DNSSEC as a cryptographically verified distribution method, so that you can, for example, verify SSL certificates without contacting a certificate authority (CA).

If I had one gripe with this book (and it's a minor one), it's that the topic is kind of boring. This isn't Lucas's fault. He does a great job of keeping your attention by injecting humor throughout. For example, on page 27 he states:

*A mob with torches and pitchforks is impractical these days, but should you have the urge to organize one, an obsolete registrar merits it.*

And on page 44 he writes:

*I abbreviate the hostnames in the key name, because I'm not that much of a masochist.*

Bits of humor like this are sprinkled throughout the book, which I greatly appreciated (however, a case might be made for others not liking it).

Overall, read this book if you're looking to deploy DNSSEC. It'll save you time and spare you a headache.

---

**Joseph Kong** is a self-taught computer enthusiast who dabbles in the fields of exploit development, reverse code engineering, rootkit development, and systems programming (FreeBSD, Linux, and Windows). He is the author of the critically acclaimed *Designing BSD Rootkits* and *FreeBSD Device Drivers*. For more information about Joseph Kong, visit [www.thestackframe.org](http://www.thestackframe.org).

**20% OFF**

Coupon code **FBJDNS** is good for 20% off ***DNSSEC Mastery*** at <https://www.tiltedwindmillpress.com/?product=dnssec-mastery-securing-the-domain-name-service-with-bind-ebook>.

This special offer to *FreeBSD Journal* subscribers expires August 31, 2015.

# THE INTERNET NEEDS YOU

**GET CERTIFIED AND GET IN THERE!**

Go to the next level with



Getting the most out of BSD operating systems requires a serious level of knowledge and expertise

## NEED AN EDGE?

• BSD Certification can make all the difference.

• Today's Internet is complex. Companies need individuals with proven skills to work on some of the most advanced systems on the Net. With BSD Certification

**YOU'LL HAVE  
WHAT IT TAKES!**

## SHOW YOUR STUFF!

Your commitment and dedication to achieving the **BSD ASSOCIATE CERTIFICATION**

can bring you to the attention of companies that need your skills.

# BSDCERTIFICATION.ORG

Providing psychometrically valid, globally affordable exams in BSD Systems Administration



# Events Calendar

The following BSD-related conferences will take place in the next few months. More information about these events, as well as local user group meetings, can be found at [www.bsdevents.org](http://www.bsdevents.org).



## LinuxFest Northwest • April 25 & 26 • Bellingham, WA

<http://linuxfestnorthwest.org/2015> • The 16th annual LFNW open-source conference will take place at Bellingham Technical College and is free to attend. There will be a FreeBSD booth in the expo area and the BSD Now crew will be filming interviews for future episodes.



## BSDCan • June 10–14 • Ottawa, Canada

<http://www.bsdcn.org> • The 12th annual BSDCan will take place in Ottawa, Canada. This popular conference appeals to a wide range of people from extreme novices to advanced developers of BSD operating systems. The conference includes a Developer Summit, Vendor Summit, Doc Sprints, tutorials, and presentations. The BSDA certification exam will be available during the lunch break on June 12 and 13 and the beta of the BSDP lab exam will be available on June 11 and 14.



## OSCON • July 21–23 • Portland, OR

<http://www.oscon.com> • There will be a FreeBSD booth in the Expo Hall of OSCON. Registration is required for this event.



## Texas LinuxFest • Aug. 21 & 22 • San Marcos, TX

<http://2015.texaslinuxfest.org/> • There will be a FreeBSD booth and several FreeBSD-related talks at TLF, to be held in the San Marcos Convention Center. The BSD certification exam will also be available during this event. There is a nominal registration fee for this conference.

Are you aware of a conference, event, or happening that might be of interest to *FreeBSD Journal* readers?

Submit calendar entries to [editor@freebsdjournal.com](mailto:editor@freebsdjournal.com).







**Testers, Systems Administrators,  
Authors, Advocates, and of course  
Programmers** to join any of our diverse teams.

**COME JOIN THE  
PROJECT THAT MAKES  
THE INTERNET GO!**

**★ DOWNLOAD OUR SOFTWARE ★**

*<http://www.freebsd.org/where.html>*

**★ JOIN OUR MAILING LISTS ★**

*<http://www.freebsd.org/community/maillinglists.html?>*

**★ ATTEND A CONFERENCE ★**

• *<http://www.bsdcan.org/2015/>* • *<http://2015.eurobsdcon.org/>*

