

ZFS and Btrfs: a Quick Introduction to Modern Filesystems

In this article, I explain how to install a ZFS kernel module on Linux, create a filesystem and compare it to existing Btrfs utilities bundled with your Linux distro. HOWARD POWELL

I am a filesystem geek. I have to be—I run a small department and manage the backups of about ten different mission-critical servers. I buy hard drives by the 20-pack, several times a year. Disks are cheap, but not always reliable, so I've learned to adapt by using the latest generation of filesystems to overcome drive failures.

In the past, I relied on RAID hardware (and/or software) to handle drive redundancy and to increase performance, but that no longer is necessary. I also used LVM to make my storage more flexible and easier to manage. The newer

filesystems like ZFS and Btrfs (have or will soon) implement redundancy and internal checking for consistency without that extra layer of expensive RAID controllers or slow software. They also handle pools of storage in a fundamental way that makes an LVM layer less useful.

The biggest concept to grasp with ZFS and Btrfs is that ZFS and Btrfs expect disks to be disks. The usage of expensive hardware or slow software RAID systems is unnecessary and conflicts with how these filesystems expect to talk to storage disks. You still can use hardware RAID if you want, but you're removing

some of the built-in safeguards that help your filesystem prevent loss of data.

ZFS

First, I'm going to talk about ZFS. I start with a quick overview of how to download and install the source code for this system on your Red Hat-compatible OS, and help you set up a small ZFS storage pool. Then, I provide some code snippets you can use to create a rotating snapshot-like backup script using ZFS's native snapshot capability. Assuming you create your storage pool on multiple disks in a mirror or RAID-Z configuration, you will have built-in protection against bitrot—the natural propagation of errors in your data from misreads or writes to disk. If you then add to your system some off-site backups, you'll have a pretty good start to a robust backup solution.

The ZFS filesystem originally was created by Sun Microsystems for its Solaris operating system. ZFS has been something that many Linux users have desired for almost a decade because of its amazing features and simple-to-implement flexibility. Unfortunately, Sun decided to release ZFS under the CDDL license, which ultimately has caused the kernel gurus never to add support to the main Linux kernel. Some people have ported ZFS to FUSE so that it can run in userspace and not taint the kernel, but this is slower than native

kernel-space access. Recently, a group at Lawrence Livermore National Labs ported ZFS so that it's a separately compiled kernel module that can be installed on your system. (This, of course, taints your kernel's license.) This module is considered to be in release candidate state, but the core ZFS code itself is based on the fairly stable Solaris version.

ZFS is notable because it's a mature implementation of a copy-on-write filesystem. Copy-on-write (COW) means that when your data is read (for example, when you open a file) and then modified, a new copy is saved to disk, and the old copy is either re-allocated for future use or preserved as a "snapshot" of the current state of the filesystem. This means that no file can ever be "in the middle" of a write operation, so corruption leading to long fscks are greatly reduced.

To get started with ZFS, first you need to download the source from <http://zfsonlinux.com>. You need two packages: the Solaris Porting Layer (this provides the Solaris kernel APIs) and the ZFS source.

You also need kernel-devel and a few other packages—this should work on a Red Hat-compatible distribution:

```
yum groupinstall "Development Tools"  
yum install zlib-devel libuuid-devel libblkid-devel  
↳ libselinux-devel parted lsscsi
```

Once the dependencies are installed, first you should compile the Solaris Porting Layer. You have to unzip the SPL, cd into the directory, run `./configure` and make `rpms`. When done, you should have XYZ rpm files for your system, which you can next install with your package manager. For example, `rpm --Uvh *.x86_64.rpm` should work on a 64-bit system.

Next, you need to unzip and configure the ZFS source directory. Again, run `make rpms` and install the resulting rpm files it creates.

Once installed, you can begin creating a filesystem. Reboot or do `service zfs start` to load the modules and prepare your system for ZFS:

```
zpool create puddle /dev/sdb /dev/sdc
```

(Note that ZFS can use partitions, but if you give it a raw disk device, it automatically will create partitions and use the entire space effectively. This is the best practice for ZFS.)

The above example creates a concatenated storage pool called “puddle”. Storage on zfs is called “pools” or collections of disks and disk partitions. A storage pool is roughly analogous to the metadevices created by `mdadm` or `lvm`. The puddle filesystem you just created will be mounted at `/puddle` and can be browsed and used just like any other filesystem. Note the first big change—you do not need an entry in `fstab` or to mount the filesystem manually; ZFS takes care of this for you.

A mirror (equivalent in concept to a RAID-1) can be created with two or more devices by using the following command:

```
zpool create mirror puddle /dev/sdb /dev/sdc
```

Finally, a RAID-5 equivalent pool can be created with:

```
zpool create raidz puddle /dev/sdb /dev/sdc /dev/sdd
```

Oh, I almost forgot—if you prefer RAID-6 instead of RAID-5, and have the disks to throw at this, there’s a solution for you also:

```
zpool create raidz-2 puddle /dev/sdb /dev/sdc /dev/sdd /dev/sde
```

It should take only a moment or two to initialize the device.

You can check your new storage pool’s status by using:

```
zpool status
```

Next, you may decide you want to move your mounted filesystem to a different mountpoint. You can use the ZFS command-line utility to do this:

```
zfs setmount puddle /mnt/puddle
```

You also can use the ZFS utility to create new sub-pools of storage that can be handled independently or together with the main pool:

```
zfs create puddle/droplet
```

Snapshots are a very effective way to make instantaneous same-disk backups of your files:

```
zfs snapshot puddle/droplet@today
```

You can mount these snapshots to copy or recover data:

```
mount -t zfs puddle/droplet@today /mnt/
```

You also can list the filesystems and snapshots, and you can see how much disk space each is consumed by using the `zfs list` command:

```
zfs list puddle # this will list all subvolumes below puddle
```

```
zfs list -t snapshot puddle # this will list all snapshots
# of puddle and subvolumes
```

Finally, maintenance of your filesystem is vital. With ext3/4 and other filesystems, you use `fsck` to make sure your system is coherent and healthy, but this requires you to take the filesystem and/or machine off-line, and it could take hours to check a large multi-terabyte disk. With ZFS, file “scrubbing” happens on-line while the system is active and available for use. Scrubbing scans through every file and makes sure that the internal checksums are still valid and correct. If you have redundant storage (a RAID-Z or a mirror), the filesystem will be self-healing and your data will be fixed automatically if any

filesystem problems are detected:

```
zpool scrub puddle
```


Much like `fsck`, a scrub operation can take a few hours, but the big difference is that your system remains on-line and ready for use! Go ahead and work with your system; you will not cause the scrub operation any problems.

Technically, a scrub of each file happens automatically every time that file is opened. The above scrub command will check every file on that storage pool,

Open Frame PPC

New - PPC-E7+

- ARM9 400Mhz Fanless Processor
- Up to 1 GB Flash & 256 MB RAM
- 7" 800 x 480 TFT LED Backlit LCD
- Analog Resistive Touchscreen
- 10/100 Base-T Ethernet
- 3 RS232 & 1 RS232/422/485 Port
- 1 USB 2.0 (High Speed) Host port
- 1 USB 2.0 (High Speed) OTG port
- 2 Micro SD Flash Card Sockets
- SPI & I2C ports
- I2S Audio Interface w/ Line-in/out
- Operating Voltage of 12 to 26 Vdc
- Optional 2D Accelerated Video & Decoder
- Pricing starts at \$550 for Qty 1



2.6 KERNEL

The PPC-E7+ Compact Panel PC comes ready to run with the Operating System installed on Flash Disk. Apply power and watch either the Linux X Windows or the Windows CE User Interface appear on the vivid 7" color LCD. Interact with the PPC-E7+ using the responsive integrated touch-screen. Everything works out of the box, allowing you to concentrate on your application, rather than building and configuring device drivers. Just Write-It and Run-It. For additional information please contact EMAC.

www.emacinc.com/panel_pc/ppc_e7+.htm

Since 1985
OVER
27
YEARS OF
SINGLE BOARD
SOLUTIONS

EMAC, inc.

EQUIPMENT MONITOR AND CONTROL

Phone: (618) 529-4525 • Fax: (618) 457-0110 • Web: www.emacinc.com

and best practices suggest the scrub command should be run periodically to check files that are rarely accessed.

Okay, now for a few code snippets to show you how you can use ZFS to your advantage for backups:

```
zfs list -H -r -d1 -t snapshot puddle
```

This snippet lists, in chronological order, all of the snapshots for your pool.

You easily can modify it to show just a sub-pool:

```
zfs list -H -r -d1 -t snapshot puddle/droplet
```

If you feed that to a `head -1` command, you instantly have the oldest snapshot on your system. You can use your favorite shell-fu commands to decide whether to keep or delete the old snapshot by:

```
zfs destroy puddle/droplet@today
```

Let's say you're copying `/home` to `/puddle` once a day, and you now have five days' worth of snapshots. You want only the last five days of backups, so let's write a script to delete the oldest snapshot, `rsync /home` to `/puddle`, and create a new snapshot:

```
OLDEST=`zfs list -H -r -d1 -t snapshot puddle|head -1`
zfs destroy ${OLDEST}
rsync -av --delete /home /puddle
zfs snapshot puddle@"date +%Y%m%d-%H.%M.%S"
```

And there you go—with as few as four lines of code, you have a rotating backup script for your data.

For those of you really interested in learning about ZFS, remember that it's a mature filesystem that's been around for almost a decade and heavily developed and invested in by many organizations, notably Sun and Oracle. Only the Linux kernel module implementation is new, so if you already have a ZFS filesystem created on a BSD UNIX or on Solaris, you easily can import it into your Linux system or vice versa. The Linux module is being actively maintained and updated periodically.

There is a lot of great documentation on-line about ZFS, but one of the best documents for people investigating using ZFS in a real environment is the ZFS Best Practices Guide, which references Solaris heavily, but don't let that scare you away (see Resources).

Btrfs

Now, some people may prefer to avoid downloading source code and compiling their own kernel modules (even though it's trivial to do so on a standard distribution these days). For these people, Btrfs (B-tree File System) is a GPL-licensed copy-on-write filesystem already included in most Linux distributions. Btrfs still is undergoing heavy development, and many features, such as parity-based

RAID, aren't yet complete.

Btrfs modules already should be installed as part of modern kernels (I can confirm it's in RHEL 6.0 and above). You need to install the `btrfs-progs` package to create a filesystem and work with it:

```
yum install btrfs-progs
```

Now, you can create the filesystem on a second, unused partition (`sdf1`) and assign it a label:

```
mkfs.btrfs -L/local /dev/sdf1
```

To pool two or more devices together:

```
mkfs.btrfs -L/local /dev/sdf1 /dev/sdg1
```

To create a mirrored filesystem for your data on two or more devices:

```
mkfs.btrfs -L/local -d raid1 /dev/sdf1 /dev/sdg1
```

With Btrfs, you need to create an entry in `/etc/fstab`. You could use "LABEL=/local" with the above example, but because RHEL 6 prefers UUIDs instead of labels, you can discover the UUID and use it instead with `btrfs-show`:

```
btrfs filesystem show /local
```

Now you can add it to the `fstab`, such as:

```
UUID=[something here] /local btrfs defaults 1 2
```

Finally, to mount the disk, just run the `mount` command. If you want to relocate the filesystem to a different mountpoint, update `fstab` and remount the filesystem.

To create a snapshot of a Btrfs mount:

```
btrfs subvolume snapshot /local /local/snapshot
```

The snapshot is mounted automatically at `/local/snapshot`. You cannot mount these snapshots outside of the Btrfs tree—they have to be subvolumes of your main Btrfs mountpoint.

New on LinuxJournal.com, the White Paper Library



www.linuxjournal.com/whitepapers

Just like ZFS, the “scrub” equivalent of a file happens automatically each time that file is accessed or read.

To list all of the snapshots on your Btrfs mount:

```
btrfs subvolume list /local
```

Finally, to destroy a snapshot:

```
btrfs subvolume destroy /local/snapshot
```

Now for a few other notes on Btrfs. Just like ZFS, the “scrub” equivalent of a file happens automatically each time that file is accessed or read. However, no on-line scrub of the entire filesystem currently is available. You could, however, use the find command to simulate a scrub:

```
find /local -mount -type f -exec cat '{}' > /dev/null \;
```

Okay, and now to replicate the functionality of the ZFS rotating snapshot backup I demoed before:

```
OLDEST=`btrfs subvolume list /local|head -1| awk '{print $NF}'`
btrfs subvolume destroy /local/${OLDEST}
rsync -av --delete /home /local
btrfs subvolume snapshot /local /local/snap-`date +%Y%m%d-%H.%M.%S`
```

Now you can wander through your snapshot directory at will and copy data as necessary.

Other Similarities

I’ve demonstrated some simple things you can do with ZFS and Btrfs here. Both filesystems have other features implemented that I did not mention, such as on-line compression using GZIP or LZO algorithms. Both filesystems support user and group quotas.

Conclusions

Personally, I have worked with ZFS longer and consider it to be more stable than Btrfs due to its heritage and inherited code base from Solaris and the BSD UNIXes. However, Btrfs is being worked on heavily by multiple groups, and at some point, it’s likely that the features of Btrfs will advance further than those available in ZFS as more distributions add support for it and as more hackers get to play with it.

If you have pre-existing scripts or filesystems from other OSes that use ZFS, the ZFS on Linux Project is just what you need to get these filesystems working with your Linux OS efficiently and easily.

On the other hand, Btrfs offers the possibility to convert an ext3 or ext4 filesystem to Btrfs, which is perfect if you already have data in place on your disks. This is a powerful tool on large storage servers where downtime due to data migration must be minimized.

I hope these examples and this quick introduction inspire you to go out and look at the new filesystems available and help contribute feedback to the developers on features you need. With your help, we finally can break free from expensive RAID

hardware and start to think of disks as just pools of storage to be used. ■

Howard Powell has been working with Linux and Solaris systems for a decade at the University of Virginia Astronomy Department. He loves filesystems, and you can reach him at both@virginia.edu.

Resources

ZFS on Linux Project: <http://zfsonlinux.org>

Best Practices Guide: http://www.solarisinternals.com/wiki/index.php/ZFS_Best_Practices_Guide

Btrfs: <http://en.wikipedia.org/wiki/Btrfs>

BTRFS Fu: http://www.funtoo.org/wiki/BTRFS_Fun

LINUX JOURNAL

now available
for the **iPad** and
iPhone at the
App Store.



Available on the
App Store

linuxjournal.com/ios



For more information about advertising opportunities within *Linux Journal* iPhone, iPad and Android apps, contact Rebecca Cassity at +1-713-344-1956 x2 or ads@linuxjournal.com.