Application Development Framework
Application Express
Big Data
Business Intelligence
Cloud Computing
Communications
Database Performance & Availability
Data Warehousing
Database
.NET
Dynamic Scripting Languages
Embedded
Digital Experience
Enterprise Architecture
Enterprise Management
Identity & Security
Java
Linux
Mobile
Service-Oriented Architecture
Solaris
SQL & PL/SQL
Systems - All Articles
Virtualization

# How I Use the Advanced Capabilities of Btrfs

*by Margaret Bierman with Lenz Grimmer*

**This article continues an exploration of Btrfs, looking into the more interesting—and sometimes less obvious—features of Btrfs, such as redundant configurations, data integrity options, compression, snapshots, and performance enhancements.**

Published August 2012

## Introduction

In my last article, "How I Got Started with the Btrfs File System for Oracle Linux," I provided an overview of the file system and illustrated how to start using its features. In this article, I continue the exploration and delve into some of the interesting—and sometimes less obvious—features of Btrfs. While Btrfs includes a number of advanced capabilities, this article focuses on those that can be used easily and have immediate benefit to users, such as redundant configurations, data integrity options, compression, snapshots, and performance enhancements.

My research and the examples provided throughout this article are based on the version of Btrfs available in Oracle Linux 6 with the Unbreakable Enterprise Kernel Release 2 (Version 2.6.39).

## A Quick Review

Before diving into some of the advanced capabilities of Btrfs, let's review the basic creation and deletion mechanisms. The example in Listing 1 creates a file system on the device `/dev/sdb` and mounts it on `/mnt`.

```
/* Create a Btrfs file system on a device using default options */

# mkfs.btrfs /dev/sdb
adding device /dev/sdb id 2
fs created label (null) on /dev/sdb
nodesize 4096 leafsize 4096 sectorsize 4096 size 10.00GB
Btrfs Btrfs v0.19

/* Mount the newly created file system */

# mount /dev/sdb /mnt
```

**Listing 1. Creating and Mounting a File System**

To copy files to and delete files from the newly created file system, use the `cp` and `rm` commands, as shown in Listing 2.

```
/* Create a subvolume named MYFILES */

# cd /mnt
# btrfs subvolume create MYFILES

/* Copy files to the new subvolume */
```

```
# cp myfile* /mnt/MYFILES

/* List the files */

# ls /mnt/MYFILES
myfile1
myfile2
myfile3

/* Delete myfile2 */

# rm /mnt/MYFILES/myfile2

/* List the files */

# ls /mnt/MYFILES
myfile1
myfile3
```

**Listing 2. Copying and Deleting Files**

It is important to note that recursively removing files with the `rm -rf` command can be very time consuming, particularly if millions of files reside on scores of disk drives. Instead, use the `btrfs subvolume delete` command, which only needs to "walk" the metadata structures to execute the deletion.

```
/* Remove subvolume MYFILES */

# btrfs subvolume delete MYFILES
```

Note that the default subvolume cannot be deleted, because that would result in the destruction of the file system. The only clean way to destroy the default subvolume is to rerun the `mkfs.btrfs` command, which would destroy existing data. As a result, it is important to think ahead when creating the initial Btrfs file system and default subvolume.

## Redundant Configuration

With Btrfs, you no longer need to use `mdadm` to create mirrored volumes or complex RAID configurations. These capabilities are built into the file system. To start, a Btrfs file system can be created on one or more devices. Additional disk drives can be added at any time to expand capacity, and they do not need to be the same size or have similar geometry. However, performance can be impacted if the drives have radically different performance characteristics.

By default, Btrfs mirrors metadata across two devices and stripes data across all devices underlying the file system. If only one device is in use, metadata is duplicated on that device and is commingled with the data store. Different RAID modes are supported for data and metadata, even on the same disks.

Continuing with the previous example, the command in Listing 3 adds device `/dev/sdc` to the Btrfs file system. Until new files are stored, or a rebalance is triggered, data is not migrated to the new device.

```
/* Add a new device to the existing Btrfs file system */

# btrfs device add /dev/sdc /mnt/btrfs

/* Verify the addition of the device to the file system */

# btrfs filesystem show
```

```
Label: none  uuid: b4f5c9a8-d8ec-4a5b-84f0-2b8c8d18b257
        Total devices 2 FS bytes used 200.33MB
        devid    1 size 5.00GB used 5.00GB path /dev/sdb
        devid    2 size 5.00GB used 4.98GB path /dev/sdc
```

**Listing 3. Adding a New Device to the File System**

**RAID Levels**
I discovered that Btrfs uses the term *RAID* differently than drive controller RAID implementations use the term. While Btrfs and traditional RAID are similar in concept, Btrfs redundancy is implemented at the chunk level rather than at the drive level. For example, traditional hardware and software RAID implementations aggregate two or more disks into a RAID1 or RAID5 logical volume and stripe data across all disks. In Btrfs, portions of disks, called chunks, can be used to create RAID logical volumes.

In Btrfs, chunks are at least 256 MB and can be mirrored or striped across multiple devices. Chunk and device trees link device items to underlying physical chunks as chunk map items. Every chunk and device is assigned a universally unique identifier (UUID). Data and metadata can be stored with different RAID levels to maximize availability. Critical information, such as metadata and device and extent trees, always is mirrored to improve the survivability of data.

You can control metadata and data RAID levels when creating the file system. For example, the commands in Listing 4 can be used to create a RAID1 (mirrored) configuration and a RAID10 (striped and mirrored) configuration. The metadata and data profiles do not need to match. In fact, metadata, data access patterns, and integrity requirements tend to be different enough that the profiles should be different.

```
/* Example that creates a RAID1 mirror for both data and metadata */

# mkfs.btrfs -m raid1 -d raid1 /dev/sdb /dev/sdc
mkfs.btrfs -m raid1 -d raid1 /dev/sdb /dev/sdc
WARNING! - Btrfs Btrfs v0.19 IS EXPERIMENTAL
WARNING! - see http://btrfs.wiki.kernel.org before using
adding device /dev/sdc id 2
fs created label (null) on /dev/sdb
        nodesize 4096 leafsize 4096 sectorsize 4096 size 10.00GB
Btrfs Btrfs v0.19

/* Example that creates a RAID10 striped mirror */

# mkfs.btrfs -m raid10 -d raid10 /dev/sdd /dev/sde
WARNING! - Btrfs Btrfs v0.19 IS EXPERIMENTAL
WARNING! - see http://btrfs.wiki.kernel.org before using
adding device /dev/sde id 2
fs created label (null) on /dev/sdd
        nodesize 4096 leafsize 4096 sectorsize 4096 size 10.00GB
Btrfs Btrfs v0.19

/* Example that mixes RAID levels for data and metadata */

/* Mirror the metadata, and stripe and mirror user data */

# mkfs.btrfs -m raid1 -d raid10 /dev/sdf /dev/sdg /dev/sdh /dev/sdi
```

**Listing 4. Creating a RAID1 and RAID10 Configuration**

## Data Integrity
Btrfs includes a number of built-in data integrity mechanisms:

**Fault isolation and checksum algorithms**. In order to preserve the integrity of data against corruption, Btrfs generates checksums for data and metadata blocks. Fault isolation is provided by storing metadata separately from user data and by protecting information through cyclical redundancy checks (CRCs) that are stored in a btree that is separate from the data.

**Corruption detection and correction**. In Btrfs, checksums are verified each time a data block is read from disk. If the file system detects a checksum mismatch while reading a block, it first tries to obtain (or create) a good copy of this block from another device—if mirroring or RAID techniques are in use. If a good copy is found, it is returned instead and the bad block is corrected. This self-healing mechanism does not appear to introduce significant overhead, and it provides a huge benefit: File systems always are consistent. Administrators are notified of repair events, and checksum failures are logged to the `syslog` facility.

**File system scrubbing**. Btrfs can initiate a check of the entire file system by triggering a file system scrub job that is performed in the background. The scrub job scans the entire file system for integrity and automatically attempts to report and repair any bad blocks it finds along the way. The file system only checks and repairs the portions of disks that are in use—this is much faster than scanning all the disks in a logical volume or storage pool. You can trigger a file system scrub with the following command:

```
/* Initiate a check of the file system */

# btrfs scrub start /mnt/MYFILES
```

**Rebuild times**. As aptly noted by Mason, Btrfs rebuilds only involve only the blocks actively used by the file system. As drive capacities increase, this is a considerable advantage over traditional file system and RAID protection mechanisms. In traditional approaches, the time to rebuild high-capacity drives can be measured in days, during which time there is reduced or no protection.

**Integration**. While Btrfs can be layered upon RAID hardware and software, using it on top of plain disks enables the built-in RAID functionality to be utilized fully. I find this results in easy integration and provides good protection from common failures.

**Encryption**. At this time, Btrfs does not provide built-in encryption functionality. An encrypted Btrfs file system can be created on top of the `dm_crypt` disk encryption subsystem and Linux Unified Key Setup (LUKS) layer, which supports a variety of encryption standards. However, this approach disables some of the capabilities and advantages of using Btrfs on raw block devices, such as automatic solid-state disk support and detection.

## Capacity Utilization

Btrfs offers compression functionality designed to optimize storage capacity utilization. Compression is supported on a per mount basis, and can be enabled after the subvolume is created. Only files created after the file system is mounted with the compression option are compressed. Once enabled, Btrfs automatically tries to compress files using Lempel-Ziv-Oberhumer (LZO) or `zlib` compression. (Other compression algorithms, such as Snappy and LZ4, are in development.) If a file does not compress well, it is marked as not compressible and written to disk uncompressed. In this case, Btrfs does not make additional compression attempts. A `force-compress` option is available in case newly added file content can be compressed.

The following command illustrates how to enable compression for a file system at mount time:

```
/* Compression can be set at the file system-level by mounting */

/* The file system with compression enabled */

# mount -o compress=lzo /dev/sdb /mnt/MYFILES
```

The `subvol` option can be used to enable compression on a subvolume. The following commands create a subvolume and mount it with compression enabled:

```
/* Create a subvolume named mysubvol */

# btrfs subvolume create /mnt/MYFILES/mysubvol

/* Mount the subvolume and enable compression */

# mount -o compress=lzo,subvol=mysubvol /dev/sdb /mnt/MYSUBVOL
```

## Snapshots, Clones, and Seed Devices

The copy-on-write nature of Btrfs makes it easy for the file system to provide several features that facilitate the replication, migration, backup, and restoration of information.

**Snapshots**. In Btrfs, a snapshot is a copy of an entire subvolume taken at a given point in time. Snapshots let you save the state of a file system at a particular point in time and recreate it on another machine to simplify data migration. You can create snapshots almost instantly, and initially they consume virtually no additional disk space (any modest impact results from additional metadata). This capability is useful when it is important to keep copies of older versions of a file system around or move them to other systems for backup or restore operations. (Since backing up a live file system often is impractical, the ability to back up a snapshot is handy.) The commands in Listing 5 show how to create a snapshot of a subvolume.

```
/* Copy two files to the MYFILES subvolume */

# cp myfile* /mnt/MYFILES

/* List the contents of the source subvolume */

# ls /mnt/MYFILES
myfile1
myfile2

/* Create a snapshot of the MYFILES subvolume and put it in /mnt/SNAPSHOT */

# btrfs subvolume snapshot /mnt/MYFILES /mnt/SNAPSHOT

/* List the contents of the snapshot subvolume */

# ls /mnt/SNAPSHOT
myfile1
myfile2
```

**Listing 5. Creating a Snapshot of a Subvolume**

**Clones**. Btrfs also supports the creation of clones for individual files within a file system or subvolume using the `cp --reflink` command. Clones are lightweight copies—only an inode is created, and it shares the same disk blocks as the original file. The following example clones the file `myfile1`, naming the cloned version `myfile3`.

```
/* Clone the file named myfile1, creating the clone myfile3 */

# cp --reflink /mnt/MYFILES/myfile1 /mnt/MYFILES/myfile3

/* List the contents of the source subvolume */

# ls /mnt/MYFILES
myfile1
myfile2
myfile3
```

**Seed devices**. Btrfs *seed devices* provide a read-only foundation to which multiple read/write file systems can point. All local updates go to these descendants. When the bulk of the data remains unchanged from the original seed file, there is considerable space savings. This can be considered another form of cloning. Note that you cannot change the seed once it is created. I found this is not a serious limitation, because necessary changes can be made and a new seed can be created.
**Provisioning**. While it is not obvious at first, Btrfs snapshots can be used to move entire software stacks onto different servers to get new servers up and running quickly—a handy feature. Using the Linux container feature, I discovered that you can replicate an entire environment pretty easily. You

simply create the container and tailor its contents, and then shut it down, clone it, and migrate it to another machine. The process is fast, reliable, and identical over as many hosts as needed. A good example of how to do this can be found at this "Containers on Linux" blog. This is a nice melding of Linux Containers (LXC) and Btrfs functionality. Unlike running multiple virtual machines, containment and isolation are provided at the container or file system level for a lightweight environment.

**Backup and restore**. Btrfs does not provide built-in support for creating backups. A best practice is to create a snapshot of a volume and use traditional backup utilities to copy data off the file system. To help, a Btrfs feature is available (`btrfs` *`subvolume`* `find-new`) that identifies which files have changed on a given subvolume. I find this feature faster than traversing the entire file system with the `find -mtime` command to locate changed files. Obviously, you can use commercial backup applications, simply using a snapshot as the source. For other ideas of how to go about backing up Btrfs file systems, see "Do-It-Yourself Backup System Using Rsync and Btrfs" and "Incremental Backups with Btrfs ."

I find that there are many different backup styles and disciplines that match different customer use cases. When Btrfs snapshots are available, all the methods work better since you do not have to quiesce active operations to the files being backed up.

**Ext file system conversion**. Btrfs supports the in-place conversion of existing ext3 and ext4 file systems. The original ext3 or ext4 file system metadata is kept in a snapshot, so the conversion can be reversed if necessary. Obviously, if a converted file system is modified heavily or modified over a protracted period, the ability to go back could have limited practical value. If it is reverted after only a short time, this can be a very useful feature. Once you have determined that a rollback is no longer likely to be helpful, the original snapshot can be deleted to save space.

An advanced use of this facility is the conversion of the root (/) file system to Btrfs. Using file system conversion with the yum-plugin-fs-snapshot facility permit rollbacks, such as undoing a software installation.

## Performance Enhancements

These days, no matter what your type of work, system performance matters. Btrfs provides functionality and device support designed to improve file system performance characteristics.

### Solid-State Disk Support

Flash memory is low-cost, nonvolatile computer memory that can be electrically erased and reprogrammed. Most of us use Flash technology on a regular basis in the form of memory cards we put in our digital cameras and the removable USB drives we use to back up and transport data from one machine to another. In the enterprise, Flash technology is used in solid-state disk drives (SSDs) to increase application performance. Wear-leveling is performed in the hardware to foster data integrity.

Btrfs is SSD-aware and exploits TRIM/Discard to allow the file system to report unused blocks to the storage device for reuse. On SSDs, Btrfs avoids unnecessary seek optimization and aggressively sends writes in clusters, even if they are from unrelated files. This results in larger write operations and faster write throughput, albeit at the expense of more seeks later. This article has some dated, but still very meaningful performance examples.

### Online Defragmentation

Over the years, I have noticed that file systems that experience a great deal of churn that fragments available capacity tend to deliver lower performance. Btrfs provides a mount option (`-o autodefrag`) that enables an auto-defragmentation helper. When a block is copied and written to disk, the auto-defragmentation helper marks that portion of the file for defragmentation and hands it off to another thread, enabling fragmentation to be reduced automatically in the background. This capability can provide significant benefit to small database workloads, browser caches, and similar workloads. The great thing is that defragmentation can take place while the file system is mounted and actively performing operations.

The following command shows how to initiate file system defragmentation for the Btrfs file system.

```
/* Initiate a defragmentation operation on the mounted Btrfs file system */

# btrfs filesystem defrag /mnt
```

## Final Thoughts

While it is a young file system, Btrfs has matured at a fast pace. Today, it has a wide range of built-in capabilities—redundant configurations, data integrity options, compression, snapshots, and performance enhancements—that elevate it to an enterprise-class file system. If you use Oracle Linux, Btrfs is a natural choice for deploying high-performance, robust platforms.

## See Also

The following resources provide more information on the capabilities of Btrfs:

Btrfs Project Wiki: https://btrfs.wiki.kernel.org/
"Btrfs: The Swiss Army Knife of Storage," by Josef Bacik: https://c59951.ssl.cf2.rackcdn.com/4376-bacik_0.pdf
"I Can't Believe This Is Butter! A Tour of Btrfs," by Avi Miller (January 2012): http://www.youtube.com/watch?v=hxWuaozpe2I
"A Btrfs Update at LinuxCon Europe," by Jonathan Corbet (November 2011): http://lwn.net/Articles/465160/
"The Btrfs File System," a presentation by Chris Mason at LinuxCon Japan, 2011: http://events.linuxfoundation.org/slides/2011/linuxcon-japan/lcj2011_mason.pdf
"Weekend Project: Get Started with Btrfs," a Linux tutorial: https://www.linux.com/learn/tutorials/371623-weekend-project-get-started-with-btrfs
Wim Coekaerts' Btrfs blog entries: https://blogs.oracle.com/wim/tags/btrfs

## About the Authors

Lenz Grimmer is a member of the Oracle Linux product management team. He has been involved in Linux and Open Source Software since 1995.

Margaret Bierman is a senior writer and trainer specializing in the research and development of technical marketing collateral for high-tech companies. Prior to writing, she worked as a software engineer on optical storage systems, specializing in the development of cross-platform file systems, hierarchical storage management systems, device drivers, and controller firmware. Margaret was also heavily involved in related standards committees, as well as training ISVs and helping them implement solutions. She received a B.S. in computational mathematics from Rensselaer Polytechnic Institute.

Revision 1.0, 08/11/2012

E-mail this page Printer View

Contact Us
US Sales: +1.800.633.0738
Have Oracle Call You
Global Contacts
Support Directory

About Oracle
Company Information
Communities
Careers
Customer Successes

Cloud
Overview of Cloud Solutions
Software (SaaS)
Platform (PaaS)
Infrastructure (IaaS)
Data (DaaS)
Free Cloud Trial

Events
Oracle OpenWorld
Oracle Code
JavaOne
All Oracle Events

Top Actions
Download Java
Download Java for Developers
Try Oracle Cloud
Subscribe to Emails

News
Newsroom
Magazines
Acquisitions
Blogs

Key Topics
ERP, EPM (Finance)
NetSuite
HCM (HR, Talent)
Marketing Cloud
CX (Sales, Service, Commerce)
Supply Chain
Industry Solutions
Database
MySQL
Middleware
Java
Engineered Systems

ORACLE® Integrated Cloud
Applications & Platform Services