(/USERS/CSCHRODER) CARLA SCHRODER (/USERS/CSCHRODER) | MARCH 20, 2014

How to Manage Btrfs Storage Pools, Subvolumes And Snapshots on Linux (part 1)

Before we dive into using Btrfs, how is it pronounced? Lots of ways, like Bee Tree Eff Ess and Bee Tee Arr Eff Ess. That's too many syllables, so I favor Butter Eff Ess. It sounds nice, and everyone likes butter. In this two-part series we'll build a three-node Btrfs storage pool and learn all about managing snapshots, rollbacks, and subvolumes. Part 1 covers installing Btrfs, creating a simple test lab, creating a storage volume, and what commands to use to see what's in it. In Part 2 we'll create and manage subvolumes, snapshots and rollbacks.

What's the Big Deal about Btrfs?

Btrfs is the next-generation Linux filesystem all cram-full of advanced features designed for maximum data protection and massive scalability such as copy-on-write, storage pools, checksums, support for 16, count 'em, 16-exabyte filesystems, journaling, online grow and shrink, and space-efficient live snapshots. If you're accustomed to using LVM and RAID to manage your data storage, Btrfs can replace these.

A snapshot is a copy of a Btrfs *subvolume* at a particular point in time. It's many times faster than making a traditional backup, and incurs no downtime. You can make snapshots of a filesystem whenever you want, and then quickly roll back to any of them.

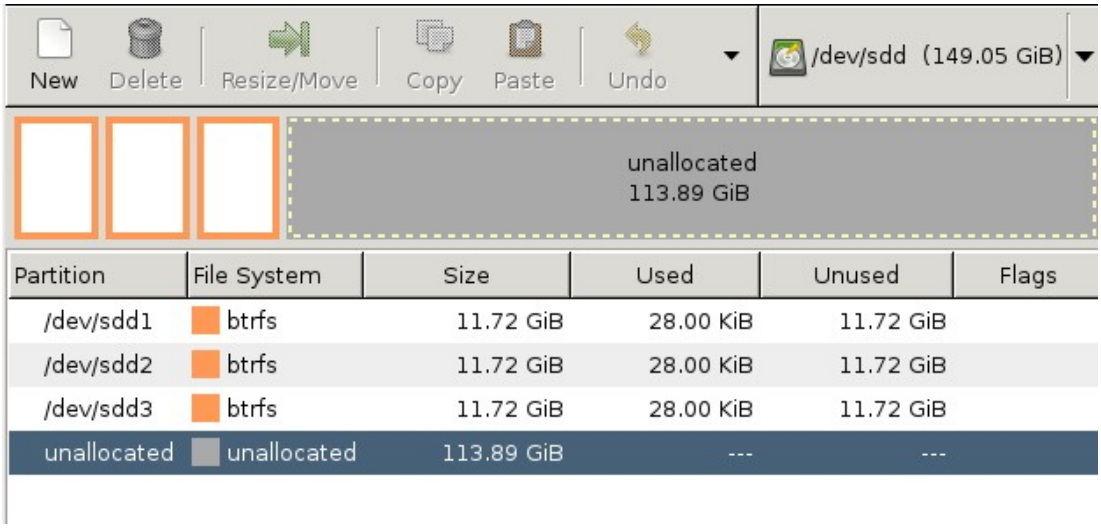
Prerequisites

To use Btrfs you need a recent version of Debian, Arch Linux, Ubuntu, OpenSUSE, SUSE Enterprise Linux, Red Hat Enterprise Linux, or Fedora Linux, and an extra empty hard disk to play with, or ~50GB of free space on a hard disk. Btrfs is already supported in the kernels of these distros (run `cat /proc/filesystems` to check), so you only need to install the user-space tools `btrfs-progs`, which is `btrfs-tools` on Debian/Ubuntu/Mint/etc.

You'll see a lot of warnings in the Btrfs documentation, and even in the output of some commands, that it is not ready for production systems and to not trust it for anything important. However, the good people at SUSE Enterprise Linux (/news/enterprise/systems-management/677226-suse-linux-says-btrfs-is-ready-to-rock) claim the opposite, and have supported it for production systems since SLES 11 SP2. I use it on my OpenSUSE and Ubuntu systems without drama. But, as they say, your mileage may vary and you should do your own testing. Meanwhile, it's free to test and learn, so let's get cracking.

Creating a Btrfs Storage Pool

First create three partitions of equal size to create a simple testing environment. GParted is a great graphical app to do this, and it partitions and creates the filesystem at the same time (figure 1). The Btrfs documentation recommends a mininum partition size of one gigabyte. In the examples for this tutorial they are 12 gigabytes each. I'm using a blank 150GB SATA hard disk for this article (`/dev/sdd`) because it makes me feel a little safer using a separate hard drive for filesystem testing. You can use any hard disk on your PC that has enough free space to play with, and 50GB gives you plenty of room to do mad Btrfs experiments. Do be careful to not destroy stuff you want to keep, like your root filesystem and data.



Now that we have three Btrfs partitions to play with, we will combine them into a Btrfs storage pool with the `mkfs.btrfs`command:

```
# mkfs.btrfs -f -L testbtrfs  /dev/sdd1 /dev/sdd2 /dev/sdd3
WARNING! - Btrfs v0.20-rc1 IS EXPERIMENTAL
WARNING! - see  (http://btrfs.wiki.kernel.org/)http://btrfs.wiki.kernel.
adding device /dev/sdd2 id 2
adding device /dev/sdd3 id 3
fs created label testbtrfs on /dev/sdd1
        nodesize 4096 leafsize 4096 sectorsize 4096 size 35.16GB
Btrfs v0.20-rc1
```

The `-f` option forces an overwrite of any existing filesystems. `-L` creates a filesystem label, which is any name you want to give it. With no other options this command creates a three-node RAID array, using RAID0 for data and RAID1 for metadata. The RAID in Btrfs has some differences from the old-fashioned RAID we're used to. In Btrfs RAID0 stripes your data across all available devices with no redundancy. RAID1 mirrors your data in pairs, round-robin across all available devices, so there are always two copies of your metadata regardless of how many devices are in the storage pool.

Seeing Your Partitions and UUIDs

You can use the familiar old `blkid` command to see your new Btrfs filesystems (the UUIDs are abbreviated in this example):

```
# blkid  /dev/sdd*
/dev/sdd: UUID="e9b11649" UUID_SUB="af7ce22c" TYPE="btrfs"
/dev/sdd1: LABEL="testbtrfs" UUID="b6a05243" UUID_SUB="4770cbfb" TYPE="bt
```

```
/dev/sdd2: LABEL="testbtrfs" UUID="b6a05243" UUID_SUB="b4524e3d" TYPE="bt
/dev/sdd3: LABEL="testbtrfs" UUID="b6a05243" UUID_SUB="7e279107" TYPE="bt
```

Mounting the Btrfs Storage Volume

Notice that the UUIDs on the three partitions in our storage volume are the same, but the UUID_SUBs are unique. If you run the `blkid` command before creating the storage pool, the UUIDs will also be unique. I like to create a special testing directory-- in this example, `/btrfs` -- so I don't accidently gum up something important. Mounting any single device mounts the whole works, like this:

```
# mkdir /btrfs
# mount /dev/sdd3 /btrfs
```

You can create an `/etc/fstab` entry in the same way as for any filesystem. Use your label or the UUID (not the UUID_SUB) like one of these examples:

```
LABEL=testbtrfs  /btrfs  btrfs  defaults  0 0
UUID=b6a05243  /btrfs  btrfs  defaults  0 0
```

What are my RAID Levels?

You can check your RAID levels with the `btrfs` command:

```
# btrfs filesystem df /btrfs
Data, RAID0: total=3.00GB, used=0.00
Data: total=8.00MB, used=0.00
System, RAID1: total=8.00MB, used=4.00KB
System: total=4.00MB, used=0.00
Metadata, RAID1: total=1.00GB, used=24.00KB
Metadata: total=8.00MB, used=0.00
```

Measuring Available Space

You can't use our good ole `du` and `df` commands to measure used and free space on the mounted Btrfs filesystem, because they don't understand Btrfs metadata, RAID, and how it manages storage. Measuring available space on a Btrfs volume is tricky because of these factors. I copied 7GB of files into my little test volume, and this is what it looks like with the`btrfs` command:

```
# btrfs filesystem df btrfs/
Data, RAID0: total=9.00GB, used=6.90GB
Data: total=8.00MB, used=0.00
System, RAID1: total=8.00MB, used=4.00KB
System: total=4.00MB, used=0.00
```

```
Metadata, RAID1: total=1.00GB, used=46.01MB
Metadata: total=8.00MB, used=0.00
```

You could also try it on any raw device in the storage pool:

```
# btrfs filesystem show /dev/sdd1
failed to open /dev/sr0: No medium found
Label: 'testbtrfs'  uuid: b6a05243
        Total devices 3 FS bytes used 6.95GB
        devid    3 size 11.72GB used 4.01GB path /dev/sdd3
        devid    2 size 11.72GB used 3.01GB path /dev/sdd2
        devid    1 size 11.72GB used 4.02GB path /dev/sdd1
```

Allrighty then, we have a nice Btrfs storage pool to play with, and know how to poke around in it. Come back for part 2 to learn how to create, remove, and manage snapshots and subvolumes.
