Btrfs: Subvolumes and snapshots

By **Jonathan Corbet**
January 6, 2014

LWN's guide to Btrfs

The previous installment in LWN's ongoing series on the Btrfs filesystem covered multiple device handling: various ways of setting up a single filesystem on a set of physical devices. Another interesting aspect of Btrfs can be thought of as working in the opposite manner: subvolumes allow the creation of multiple filesystems on a single device (or array of devices). Subvolumes create a number of interesting possibilities not supported by other Linux filesystems. This article will discuss how to use the subvolume feature and the associated snapshot mechanism.

Subvolume basics

A typical Unix-style filesystem contains a single directory tree with a single root. By default, a Btrfs filesystem is organized in the same way. Subvolumes change that picture by creating alternative roots that function as independent filesystems in their own right. This can be illustrated with a simple example:

```
    # mkfs.btrfs /dev/sdb5
    # mount /dev/sdb5 /mnt/1
    # cd /mnt/1
    # touch a
```

Thus far, we have a mundane btrfs filesystem with a single empty file (called "`a`") on it. To create a subvolume and create a file within it, one can type:

```
    # btrfs subvolume create subv
    # touch subv/b
    # tree
    .
    ├── a
    └── subv
        └── b

    1 directory, 2 files
```

The subvolume has been created with the name `subv`; thus far, the operation looks nearly indistinguishable from having simply created a directory by that name. But there are some differences that pop up if one looks for them. For example:

```
    # ln a subv/
    ln: failed to create hard link ‘subv/a’ => ‘a’: Invalid cross-device link
```

So, even though `subv` looks like an ordinary subdirectory, the filesystem treats it as if it were on a separate physical device; moving into `subv` is like crossing an ordinary Unix mount point, even though it's still housed within the original btrfs filesystem. The subvolume can also be mounted independently:

```
    # btrfs subvolume list /mnt/1
    ID 257 gen 8 top level 5 path subv
    # mount -o subvolid=257 /dev/sdb5 /mnt/2
    # tree /mnt/2
    /mnt/2
    └── b

    0 directories, 1 file
```

The end result is that each subvolume can be treated as its own filesystem. It is entirely possible to create a whole series of subvolumes and mount each separately, ending up with a set of independent filesystems all sharing the underlying storage device. Once the subvolumes have been created, there is no need to mount the "root" device at all if only the subvolumes are of interest.

Btrfs will normally mount the root volume unless explicitly told to do otherwise with the `subvolid=` mount option. But that is simply a default; if one wanted the new subvolume to be mounted by default instead, one could run:

```
    btrfs subvolume set-default 257 /mnt/1
```

Thereafter, mounting `/dev/sdb5` with no `subvolid=` option will mount the subvolume `subv`. The root volume has a subvolume ID of zero, so mounting with `subvolid=0` will mount the root.

Subvolumes can be made to go away with:

```
    btrfs subvolume delete path
```

For ordinary subvolumes (as opposed to snapshots, described below), the subvolume indicated by *`path`* must be empty before it can be deleted.

Snapshots

A snapshot in Btrfs is a special type of subvolume — one which contains a copy of the current state of some other subvolume. If we return to our simple filesystem created above:

```
    # btrfs subvolume snapshot /mnt/1 /mnt/1/snapshot
    # tree /mnt/1
    /mnt/1
    ├── a
    ├── snapshot
    │   ├── a
    │   └── subv
    │
    └── subv
        └── b

    3 directories, 3 files
```

The `snapshot` subcommand creates a snapshot of the given subvolume (the `/mnt/1` root volume in this case), placing that snapshot under the requested name (`/mnt/1/snapshot`) in that subvolume. As a result, we now have a new subvolume called `snapshot` which appears to contain a full copy of everything that was in the filesystem previously. But, of course, Btrfs is a copy-on-write filesystem, so there is no need to actually copy all of that data; the snapshot simply has a reference to the current root of the filesystem. If anything is changed — in *either* the main volume or the snapshot — a copy of the relevant data will be made, so the other copy will remain unchanged.

Note also that the contents of the existing subvolume (`subv`) do not appear in the snapshot. If a snapshot of a subvolume is desired, that must be created separately.

Snapshots clearly have a useful backup function. If, for example, one has a Linux system using Btrfs, one can create a snapshot prior to installing a set of distribution updates. If the updates go well, the snapshot can simply be deleted. (Deletion is done with "`btrfs subvolume delete`" as above, but snapshots are not expected to be empty before being deleted). Should the update go badly, instead, the snapshot can be made the default subvolume and, after a reboot, everything is as it was before.

Snapshots can also be used to implement a simple "time machine" functionality. While working on this article series, your editor set aside a Btrfs partition to contain a copy of `/home`. On occasion, a simple script runs:

```
    rsync -aix --delete /home /home-backup
    btrfs subvolume snapshot /home-backup /home-backup/ss/`date +%y-%m-%d_%H-%M`
```

The `rsync` command makes `/home-backup` look identical to `/home`; a snapshot is then made of that state of affairs. Over time, the result is the creation of a directory full of timestamped snapshots; returning to the state of `/home` at any given time is a simple matter of going into the proper snapshot. Of course, if `/home` is also on a Btrfs filesystem, one could make regular snapshots without the `rsync` step, but the redundancy that comes with a backup drive would be lost.

One can quickly get used to having this kind of resource available. This also seems like an area that is just waiting for the development of some higher-level tools. Some projects are already underway; see Snapper or btrfs-time-machine, for example. There is also an "autosnap" feature that has been posted in the past, though it does not seem to have seen any development recently. For now, most snapshot users are most likely achieving the desired functionality through their own sets of *ad hoc* scripts.

Subvolume quotas

It typically will not take long before one starts to wonder how much disk space is used by each subvolume. A naive use of a tool like `du` may or may not produce a useful answer; it is slow and unable to take into account the sharing of data between subvolumes (snapshots in particular). Beyond that, in many situations, it would be nice to be able to divide a volume into subvolumes but not to allow any given subvolume to soak up all of the available storage space. These needs can be met through the Btrfs subvolume quota group mechanism.

Before getting into quotas, though, a couple of caveats are worth mentioning. One is that "quotas" in this sense are not normal, per-user disk quotas; those can be managed on Btrfs just like with any other filesystem. Btrfs subvolume quotas, instead, track and regulate usage by subvolumes, with no regard for the ownership of the files that actually take up the space. The other thing worth bearing in mind is that the quota mechanism is relatively new. The management tools are on the rudimentary side, there seem to be some performance issues associated with quotas, and there's still a sharp edge or two in there waiting for unlucky users.

By default, Btrfs filesystems do not have quotas enabled. To turn this feature on, run:

```
    # btrfs quota enable path
```

A bit more work is required to retrofit quotas into an older Btrfs filesystem; see this wiki page for details. Once quotas are established, one can look at actual usage with:

```
    # btrfs qgroup show /home-backup
    qgroupid rfer        excl
    -------- ----        ----
    0/5      21184458752 49152
    0/277    21146079232 2872635392
    0/281    20667858944 598929408
    0/282    20731035648 499802112
    0/284    20733419520 416395264
    0/286    20765806592 661327872
    0/288    20492754944 807755776
    0/290    20672286720 427991040
    0/292    20718280704 466567168
    0/294    21184458752 49152
```

This command was run in the time-machine partition described above, where all of the subvolumes are snapshots. The **qgroupid** is the ID number (actually a pair of numbers — see below) associated with the quota group governing each subvolume, **rfer** is the total amount of data referred to in the subvolume, and **excl** is the amount of data that is not shared with any other subvolume. In short, "rfer" approximates what "`du`" would indicate for the amount of space used in a subvolume, while "excl" tells how much space would be freed by deleting the subvolume.

...or, something approximately like that. In this case, the subvolume marked `0/5` is the root volume, which cannot be deleted. "`0/294`" is the most recently created snapshot; it differs little from the current state of the filesystem, so there is not much data that is unique to the snapshot itself. If one were to delete a number of files from the main filesystem, the amount of "excl" data in that last snapshot would increase (since those files still exist in the snapshot) while the amount of free space in the filesystem as a whole would not increase.

Limits can be applied to subvolumes with a command like:

```
    # btrfs qgroup limit 30M /mnt/1/subv
```

One can then test the limit with:

```
# dd if=/dev/zero of=/mnt/1/subv/junk bs=10k
dd: error writing ‘junk’: Disk quota exceeded
2271+0 records in
2270+0 records out
23244800 bytes (23 MB) copied, 0.0334957 s, 694 MB/s
```

One immediate conclusion that can be drawn is that the limits are somewhat approximate at best; in this case, a limit of 30MB was requested, but the enforcement kicked in rather sooner than that. This happens even though the system appears to have a clear understanding of both the limit and current usage:

```
# btrfs qgroup show -r /mnt/1
qgroupid rfer     excl     max_rfer
-------- ----     ----     --------
0/5      16384    16384    0
0/257    23261184 23261184 31457280
```

The `0/257` line corresponds to the subvolume of interest; the current usage is shown as being rather less than the limit, but writes were limited anyway.

There is another interesting complication with subvolume quotas, as demonstrated by:

```
# rm /mnt/1/subv/junk
rm: cannot remove ‘/mnt/1/subv/junk’: Disk quota exceeded
```

In a copy-on-write world, even deleting data requires allocating space, for a while at least. A user in this situation would appear to be stuck; little can be done until somebody raises the limit for at least as long as it takes to remove some files. This particular problem has been known to the Btrfs developers since 2012, but there does not yet appear to be a fix in the works.

The quota group is somewhat more flexible than has been shown so far; it can, for example, organize quotas in hierarchies that apply limits at multiple levels. Imagine one had a Btrfs filesystem to be used for home directories, among other things. Each user's home could be set up as a separate subvolume with something like this:

```
# cd /mnt/1
# btrfs subvolume create home
# btrfs subvolume create home/user1
# btrfs subvolume create home/user2
# btrfs subvolume create home/user3
```

By default, each subvolume is in its own quota group, so each user's usage can be limited easily enough. But if there are other hierarchies in the same Btrfs filesystem, it might be nice to limit the usage of `home` as a whole. One would start by creating a new quota group:

```
# btrfs qgroup create 1/1 home
```

Quota group IDs are, as we have seen, a pair of numbers; the first of those numbers corresponds to the group's level in the hierarchy. At the leaf level, that number is zero; IDs at that level have the subvolume ID as the second number of the pair. All higher levels are created by the administrator, with the second number being arbitrary.

The assembly of the hierarchy is done by assigning the bottom-level groups to the new higher-level groups. In this case, the subvolumes created for the user-level directories have IDs 258, 259, and 260 (as seen with `btrfs subvolume list`), so the assignment is done with:

```
    # btrfs qgroup assign 0/258 1/1 .
    # btrfs qgroup assign 0/259 1/1 .
    # btrfs qgroup assign 0/260 1/1 .
```

Limits can then be applied with:

```
    # btrfs qgroup limit 5M 0/258 .
    # btrfs qgroup limit 5M 0/259 .
    # btrfs qgroup limit 5M 0/260 .
    # btrfs qgroup limit 10M 1/1 .
```

With this setup, any individual user can use up to 5MB of space within their own subvolume. But users as a whole will be limited to 10MB of space within the `home` subvolume, so if `user1` and `user2` use their full quotas, `user3` will be entirely out of luck. After creating exactly such a situation, querying the quota status on the filesystem shows:

```
    # btrfs qgroup show -r .
    qgroupid rfer     excl     max_rfer
    -------- ----     ----     --------
    0/5      16384    16384    0
    0/257    16384    16384    0
    0/258    5189632  5189632  5242880
    0/259    5189632  5189632  5242880
    0/260    16384    16384    5242880
    1/1      10346496 10346496 10485760
```

We see that the first two user subvolumes have exhausted their quotas; that is also true of the upper-level quota group (`1/1`) that we created for `home` as a whole. As far as your editor can tell, there is no way to query the shape of the hierarchy; one simply needs to know how that hierarchy was built to work with it effectively.

As can be seen, subvolume quota support still shows signs of being relatively new code; there is still a fair amount of work to be done before it is truly ready for production use. Subvolume and snapshot support in general, though, has been around for years and is in relatively good shape. All told, subvolumes offer a highly useful feature set; in the future, we may well wonder how we ran our systems without them.

At this point, our survey of the major features of the Btrfs filesystem is complete. The next (and final) installment in this series will cover a number of loose ends, the send/receive feature, and more.

(Log in to post comments)

Deleting subvolumes

Posted Jan 6, 2014 19:48 UTC (Mon) by **fishface60** (subscriber, #88700) [Link]

> For ordinary subvolumes (as opposed to snapshots, described below),
> the subvolume indicated by path must be empty before it can be deleted.