btrfs-quota(8)

SYNOPSIS

btrfs quota <subcommand> <args>

DESCRIPTION

The commands under **btrfs quota** are used to affect the global status of quotas of a btrfs filesystem. The quota groups (qgroups) are managed by the subcommand btrfs-qgroup(8).

Note

Qgroups are different than the traditional user quotas and designed to track shared and exclusive data per-subvolume. Please refer to the section HIERARCHICAL QUOTA GROUP CONCEPTS for a detailed description.

PERFORMANCE IMPLICATIONS

When quotas are activated, they affect all extent processing, which takes a performance hit. Activation of qgroups is not recommended unless the user intends to actually use them.

STABILITY STATUS

The qgroup implementation has turned out to be quite difficult as it affects the core of the filesystem operation. Qgroup users have hit various corner cases over time, such as incorrect accounting or system instability. The situation is gradually improving and issues found and fixed.

HIERARCHICAL QUOTA GROUP CONCEPTS

The concept of quota has a long-standing tradition in the Unix world. Ever since computers allow multiple users to work simultaneously in one filesystem, there is the need to prevent one user from using up the entire space. Every user should get his fair share of the available resources.

In case of files, the solution is quite straightforward. Each file has an *owner* recorded along with it, and it has a size. Traditional quota just restricts the total size of all files that are owned by a user. The concept is quite flexible: if a user hits his quota limit, the administrator can raise it on the fly.

On the other hand, the traditional approach has only a poor solution to restrict directories. At installation time, the harddisk can be partitioned so that every directory (e.g. `/usr`, `/var`, ...) that needs a limit gets its own partition. The obvious problem is that those limits cannot be changed without a reinstallation. The btrfs subvolume feature builds a bridge. Subvolumes correspond in many ways to partitions, as every subvolume looks like its own filesystem. With subvolume quota, it is now possible to restrict each subvolume like a partition, but keep the flexibility of quota. The space for each subvolume can be expanded or restricted on the fly.

As subvolumes are the basis for snapshots, interesting questions arise as to how to account used space in the presence of snapshots. If you have a file shared between a subvolume and a snapshot, whom to account the file to? The creator? Both? What if the file gets modified in the snapshot, should only these changes be accounted to it? But wait, both the snapshot and the subvolume belong to the same user home. I just want to limit the total space used by both! But somebody else might not want to charge the snapshots to the users.

Btrfs subvolume quota solves these problems by introducing groups of subvolumes and let the user put limits on them. It is even possible to have groups of groups. In the following, we refer to them as *qgroups*.

Each qgroup primarily tracks two numbers, the amount of total referenced space and the amount of exclusively referenced space.

referenced

space is the amount of data that can be reached from any of the subvolumes contained in the qgroup, while

exclusive

is the amount of data where all references to this data can be reached from within this qgroup.

Subvolume quota groups

The basic notion of the Subvolume Quota feature is the quota group, short qgroup. Qgroups are notated as *level/id*, e.g. the qgroup 3/2 is a qgroup of level 3. For level 0, the leading 0/ can be omitted. Qgroups of level 0 get created automatically when a subvolume/snapshot gets created. The ID of the qgroup corresponds to the ID of the subvolume, so 0/5 is the qgroup for the root subvolume. For the **btrfs qgroup** command, the path to the subvolume can also be used instead of *0/ID*. For all higher levels, the ID can be chosen freely.

Each qgroup can contain a set of lower level qgroups, thus creating a hierarchy of qgroups. Figure 1 shows an example qgroup tree.

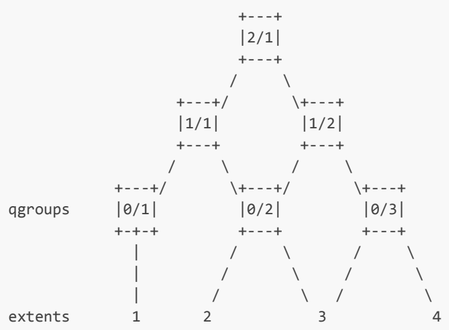

Figure 1: Sample qgroup hierarchy

At the bottom, some extents are depicted showing which qgroups reference which extents. It is important to understand the notion of *referenced* vs *exclusive*. In the example, qgroup 0/2 references extents 2 and 3, while 1/2 references extents 2-4, 2/1 references all extents.

On the other hand, extent 1 is exclusive to 0/1, extent 2 is exclusive to 0/2, while extent 3 is neither exclusive to 0/2 nor to 0/3. But because both references can be reached from 1/2, extent 3 is exclusive to 1/2. All extents are exclusive to 2/1.

So exclusive does not mean there is no other way to reach the extent, but it does mean that if you delete all subvolumes contained in a qgroup, the extent will get deleted.

Exclusive of a qgroup conveys the useful information how much space will be freed in case all subvolumes of the qgroup get deleted.

All data extents are accounted this way. Metadata that belongs to a specific subvolume (i.e. its filesystem tree) is also accounted. Checksums and extent allocation information are not accounted.

In turn, the referenced count of a qgroup can be limited. All writes beyond this limit will lead to a 'Quota Exceeded' error.

Inheritance

Things get a bit more complicated when new subvolumes or snapshots are created. The case of (empty) subvolumes is still quite easy. If a subvolume should be part of a qgroup, it has to be added to the qgroup at creation time. To add it at a later time, it would be necessary to at least rescan the full subvolume for a proper accounting.

Creation of a snapshot is the hard case. Obviously, the snapshot will reference the exact amount of space as its source, and both source and destination now have an exclusive count of 0 (the filesystem nodesize to be precise, as the roots of the trees are not shared). But what about qgroups of higher levels? If the qgroup contains both the source and the destination, nothing changes. If the qgroup contains only the source, it might lose some exclusive.

But how much? The tempting answer is, subtract all exclusive of the source from the qgroup, but that is wrong, or at least not enough. There could have been an extent that is referenced from the source and another subvolume from that qgroup. This extent would have been exclusive to the qgroup, but not to the source subvolume. With the creation of the snapshot, the qgroup would also lose this extent from its exclusive set.

So how can this problem be solved? In the instant the snapshot gets created, we already have to know the correct exclusive count. We need to have a second qgroup that contains all the subvolumes as the first qgroup, except the subvolume we want to snapshot. The moment we create the snapshot, the exclusive count from the second qgroup needs to be copied to the first qgroup, as it represents the correct value. The second qgroup is called a tracking qgroup. It is only there in case a snapshot is needed.

Use cases

Below are some use cases that do not mean to be extensive. You can find your own way how to integrate qgroups.

Single-user machine

Replacement for partitions. The simplest use case is to use qgroups as simple replacement for partitions. Btrfs takes the disk as a whole, and `/`, `/usr`, `/var`, etc. are created as subvolumes. As each subvolume gets it own qgroup automatically, they can simply be restricted. No hierarchy is needed for that.

Track usage of snapshots. When a snapshot is taken, a qgroup for it will automatically be created with the correct values. *Referenced* will show how much is in it, possibly shared with other subvolumes. *Exclusive* will be the amount of space that gets freed when the subvolume is deleted.

Multi-user machine

Restricting homes. When you have several users on a machine, with home directories probably under `/home`, you might want to restrict `/home` as a whole, while restricting every user to an individual limit as well. This is easily accomplished by creating a qgroup for `/home`, e.g. 1/1, and assigning all user subvolumes to it. Restricting this qgroup will limit /home, while every user subvolume can get its own (lower) limit.

Accounting snapshots to the user. Let's say the user is allowed to create snapshots via some mechanism. It would only be fair to account space used by the snapshots to the user. This does not mean the user doubles his usage as soon as he takes a snapshot. Of course, files that are present in his home and the snapshot should only be accounted once. This can be accomplished by creating a qgroup for each user, say *1/UID*. The user home and all snapshots are assigned to this qgroup. Limiting it will extend the limit to all snapshots, counting files only once. To limit `/home` as a whole, a higher level group 2/1 replacing 1/1 from the previous example is needed, with all user qgroups assigned to it.

Do not account snapshots. On the other hand, when the snapshots get created automatically, the user has no chance to control them, so the space used by them should not be accounted to him. This is already the case when creating snapshots in the example from the previous section.

Snapshots for backup purposes. This scenario is a mixture of the previous two. The user can create snapshots, but some snapshots for backup purposes are being created by the system. The user's snapshots should be accounted to the user, not the system. The solution is similar to the one from section *Accounting snapshots to the user*, but do not assign system snapshots to user's qgroup.

Simple quotas (squota)

As detailed in this document, qgroups can handle many complex extent sharing and unsharing scenarios while maintaining an accurate count of exclusive and shared usage. However, this flexibility comes at a cost: many of the computations are global, in the sense that we must count up the number of trees referring to an extent after its references change. This can slow down transaction commits and lead to unacceptable latencies, especially in cases where snapshots scale up.

To work around this limitation of qgroups, btrfs also supports a second set of quota semantics: *simple quotas* or *squotas*. Squotas fully share the qgroups API and hierarchical model, but do not track shared vs. exclusive usage. Instead, they account all extents to the subvolume that first allocated it. With a bit of new bookkeeping, this allows all accounting decisions to be local to the allocation or freeing operation that deals with the extents themselves, and fully avoids the complex and costly back-reference resolutions.

`Example`

To illustrate the difference between squotas and qgroups, consider the following basic example assuming a nodesize of 16KiB.

1. create subvolume 256
2. rack up 1GiB of data and metadata usage in 256
3. snapshot 256, creating subvolume 257
4. COW 512MiB of the data and metadata in 257
5. delete everything in 256

At each step, qgroups would have the following accounting:

1. 0/256: 16KiB excl 0 shared
2. 0/256: 1GiB excl 0 shared
3. 0/256: 0 excl 1GiB shared; 0/257: 0 excl 1GiB shared
4. 0/256: 512MiB excl 512MiB shared; 0/257: 512MiB excl 512MiB shared
5. 0/256: 16KiB excl 0 shared; 0/257: 1GiB excl 0 shared

Whereas under squotas, the accounting would look like:

1. 0/256: 16KiB excl 16KiB shared
2. 0/256: 1GiB excl 1GiB shared
3. 0/256: 1GiB excl 1GiB shared; 0/257: 16KiB excl 16KiB shared
4. 0/256: 1GiB excl 1GiB shared; 0/257: 512MiB excl 512MiB shared
5. 0/256: 512MiB excl 512MiB shared; 0/257: 512MiB excl 512MiB shared

Note that since the original snapshotted 512MiB are still referenced by 257, they cannot be freed from 256, even after 256 is emptied, or even deleted.

`Summary`

If you want some of power and flexibility of quotas for tracking and limiting subvolume usage, but want to avoid the performance penalty of accurately tracking extent ownership life cycles, then squotas can be a useful option.

Furthermore, squotas is targeted at use cases where the original extent is immutable, like image snapshotting for container startup, in which case we avoid these awkward scenarios where a subvolume is empty or deleted but still has significant extents accounted to it. However, as long as you are aware of the accounting semantics, they can handle mutable original extents.

SUBCOMMAND

disable <path>

Disable subvolume quota support for a filesystem.

enable [options] <path>

Enable subvolume quota support for a filesystem. At this point it's possible the two modes of accounting. The *full* means that extent ownership by subvolumes will be tracked all the time, *simple* will account everything to the first owner. See the section for more details.

`Options`

-s|--simple

use simple quotas (squotas) instead of full qgroup accounting

rescan [options] <path>

Trash all qgroup numbers and scan the metadata again with the current config.

`Options`

-s|--status

show status of a running rescan operation.

-w|--wait

start rescan and wait for it to finish (can be already in progress)

-W|--wait-norescan

wait for rescan to finish without starting it

EXIT STATUS

btrfs quota returns a zero exit status if it succeeds. Non zero is returned in case of failure.

AVAILABILITY

btrfs is part of btrfs-progs. Please refer to the documentation at https://btrfs.readthedocs.io.

SEE ALSO

btrfs-qgroup(8), btrfs-subvolume(8), mkfs.btrfs(8)

Previous Next