btrfs-subvolume(8)

SYNOPSIS

btrfs subvolume <subcommand> [<args>]

DESCRIPTION

btrfs subvolume is used to create/delete/list/show btrfs subvolumes and snapshots.

A BTRFS subvolume is a part of filesystem with its own independent file/directory hierarchy and inode number namespace. Subvolumes can share file extents. A snapshot is also subvolume, but with a given initial content of the original subvolume. A subvolume has always inode number 256 (see more in Inode numbers (in Subvolumes)).

Note

A subvolume in BTRFS is not like an LVM logical volume, which is block-level snapshot while BTRFS subvolumes are file extent-based.

A subvolume looks like a normal directory, with some additional operations described below. Subvolumes can be renamed or moved, nesting subvolumes is not restricted but has some implications regarding snapshotting. The numeric id (called *subvolid* or *rootid*) of the subvolume is persistent and cannot be changed.

A subvolume in BTRFS can be accessed in two ways:

- like any other directory that is accessible to the user
- like a separately mounted filesystem (options *subvol* or *subvolid*)

In the latter case the parent directory is not visible and accessible. This is similar to a bind mount, and in fact the subvolume mount does exactly that.

A freshly created filesystem is also a subvolume, called *top-level*, internally has an id 5. This subvolume cannot be removed or replaced by another subvolume. This is also the subvolume that will be mounted by default, unless the default subvolume has been changed (see btrfs subvolume set-default).

A snapshot is a subvolume like any other, with given initial content. By default, snapshots are created read-write. File modifications in a snapshot do not affect the files in the original subvolume.

Subvolumes can be given capacity limits, through the qgroups/quota facility, but otherwise share the single storage pool of the whole btrfs filesystem. They may even share data between themselves (through deduplication or snapshotting).

Note

A snapshot is not a backup: snapshots work by use of BTRFS' copy-on-write behaviour. A snapshot and the original it was taken from initially share all of the same data blocks. If that data is damaged in some way (cosmic rays, bad disk sector, accident with dd to the disk), then the snapshot and the original will both be damaged. Snapshots are useful to have local online "copies" of the filesystem that can be referred back to, or to implement a form of deduplication, or to fix the state of a filesystem for making a full backup without anything changing underneath it. They do not in themselves make your data any safer.

Subvolume flags

The subvolume flag currently implemented is the *ro* property (read-only status). Read-write subvolumes have that set to *false*, snapshots as *true*. In addition to that, a plain snapshot will also have last change generation and creation generation equal.

Read-only snapshots are building blocks of incremental send (see btrfs-send(8)) and the whole use case relies on unmodified snapshots where the relative changes are generated from. Thus, changing the subvolume flags from read-only to read-write will break the assumptions and may lead to unexpected changes in the resulting incremental stream.

A snapshot that was created by send/receive will be read-only, with different last change generation, and with set *received_uuid* which identifies the subvolume on the filesystem that produced the stream. The use case relies on matching data on both sides. Changing the subvolume to read-write after it has been received requires to reset the *received_uuid*. As this is a notable change and could potentially break the incremental send use case, performing it by btrfs property set requires force if that is really desired by user.

Note

The safety checks have been implemented in 5.14.2, any subvolumes previously received (with a valid *received_uuid*) and read-write status may exist and could still lead to problems with send/receive. You can use btrfs subvolume show to identify them. Flipping the flags to read-only and back to read-write will reset the *received_uuid* manually. There may exist a convenience tool in the future.

Nested subvolumes

There are no restrictions for subvolume creation, so it's up to the user how to organize them, whether to have a flat layout (all subvolumes are direct descendants of the toplevel one), or nested.

What should be mentioned early is that a snapshotting is not recursive, so a subvolume or a snapshot is effectively a barrier and no files in the nested subvolumes appear in the snapshot. Instead, there's a stub subvolume, also sometimes called *empty subvolume*, with the same name as original subvolume and with inode number 2. This can be used intentionally but could be confusing in case of nested layouts.

```
$ btrfs subvolume create subvol1
$ btrfs subvolume create subvol1/subvol2
$ btrfs subvolume snapshot subvol1 snap1
$ find -ls
121093  0  drwxr-xr-x  1  user  users    24  Jul 30  12:34  .
   256  0  drwxr-xr-x  1  user  users    14  Jul 30  12:34  ./subvol1
   256  0  drwxr-xr-x  1  user  users     0  Jul 30  12:34  ./subvol1/subvol2
   257  0  -rw-r--r--  1  user  users     0  Jul 30  12:34  ./subvol1/subvol2/file
   256  0  drwxr-xr-x  1  user  users    14  Jul 30  12:34  ./snap1
     2  0  drwxr-xr-x  1  user  users     0  Jul 30  12:34  ./snap1/subvol2
```

The numbers in the first columns are inode numbers, 256 is for a regular subvolume (or snapshot), 2 is the *empty subvolume*. The snapshotted directory representing *subvol2* does not contain the *file*.

Note

The *empty subvolume* will not be sent (btrfs-send(8)) and thus will not be created on the receive side (btrfs-receive(8)).

Case study: system root layouts

There are two ways how the system root directory and subvolume layout could be organized. The interesting use case for root is to allow rollbacks to previous version, as one atomic step. If the entire filesystem hierarchy starting in `/` is in one subvolume, taking snapshot will encompass all files. This is easy for the snapshotting part but has undesirable consequences for rollback. For example, log files would get rolled back too, or any data that are stored on the root filesystem but are not meant to be rolled back either (database files, VM images, ...).

Here we could utilize the snapshotting barrier mentioned above, making each directory that stores data to be preserved across rollbacks its own subvolume. This could be e.g. `/var`. Further more fine-grained partitioning could be done, e.g. adding separate subvolumes for `/var/log`, `/var/cache` etc.

The fact that there are separate subvolumes requires separate actions to take the snapshots (here, it gets disconnected from the system root snapshots). This needs to be taken care of by system tools, installers, together with selection of which directories are highly recommended to be separate subvolumes.

Mount options

Mount options are of two kinds, generic (that are handled by VFS layer) and specific, handled by the filesystem. The following list shows which are applicable to individual subvolume mounts, while there are more options that always affect the whole filesystem:

- Generic: noatime/relatime/..., nodev, nosuid, ro, rw, dirsync
- Filesystem-specific: compress, autodefrag, nodatacow, nodatasum

Examples of whole filesystem options are e.g. *space_cache*, *rescue*, *device*, *skip_balance*, etc. The exceptional options are *subvol* and *subvolid* that are actually used for mounting a given subvolume and can be specified only once for the mount.

Subvolumes belong to a single filesystem and, as implemented now, all share the same specific mount options. Also, changes done by remount have immediate effect. This may change in the future.

Mounting a read-write snapshot as read-only is possible and will not change the *ro* property and flag of the subvolume.

The name of the mounted subvolume is stored in file `/proc/self/mountinfo` in the 4th column:

```
27 21 0:19 /subv1 /mnt rw,relatime - btrfs /dev/sda rw,space_cache
          ^^^^^^
```

Inode numbers

A directory representing a subvolume has always inode number 256 (sometimes also called a root of the subvolume):

```
$ ls -lis
total 0
389111 0 drwxr-xr-x 1 user users 0 Jan 20 12:13 dir
389110 0 -rw-r--r-- 1 user users 0 Jan 20 12:13 file
   256 0 drwxr-xr-x 1 user users 0 Jan 20 12:13 snap1
   256 0 drwxr-xr-x 1 user users 0 Jan 20 12:13 subv1
```

If a subvolume is nested and then a snapshot is taken, then the cloned directory entry representing the subvolume becomes empty and the inode has number 2. All other files and directories in the target snapshot preserve their original inode numbers.

Note

Inode number is not a filesystem-wide unique identifier, some applications assume that. Please use the *subvolumeid:inodenumber* pair for that purpose. The subvolume id can be read by btrfs inspect-internal rootid or by the ioctl BTRFS_IOC_INO_LOOKUP.

Performance

Subvolume creation needs to flush dirty data that belong to the subvolume and this step may take some time. Otherwise, once there's nothing else to do, the snapshot is instantaneous and only creates a new tree root copy in the metadata.

Snapshot deletion has two phases: first its directory is deleted and the subvolume is added to a queuing list, then the list is processed one by one and the data related to the subvolume get deleted. This is usually called *cleaning* and can take some time depending on the amount of shared blocks (can be a lot of metadata updates), and the number of currently queued deleted subvolumes.

SUBVOLUME AND SNAPSHOT

A subvolume is a part of filesystem with its own independent file/directory hierarchy. Subvolumes can share file extents. A snapshot is also subvolume, but with a given initial content of the original subvolume.

Note

A subvolume in btrfs is not like an LVM logical volume, which is block-level snapshot while btrfs subvolumes are file extent-based.

A subvolume looks like a normal directory, with some additional operations described below. Subvolumes can be renamed or moved, nesting subvolumes is not restricted but has some implications regarding snapshotting.

A subvolume in btrfs can be accessed in two ways:

- like any other directory that is accessible to the user
- like a separately mounted filesystem (options *subvol* or *subvolid*)

In the latter case the parent directory is not visible and accessible. This is similar to a bind mount, and in fact the subvolume mount does exactly that.

A freshly created filesystem is also a subvolume, called *top-level*, internally has an id 5. This subvolume cannot be removed or replaced by another subvolume. This is also the subvolume that will be mounted by default, unless the default subvolume has been changed (see subcommand set-default).

A snapshot is a subvolume like any other, with given initial content. By default, snapshots are created read-write. File modifications in a snapshot do not affect the files in the original subvolume.

SUBCOMMAND

create [options] [<dest>/]<name> [[<dest2>/]<name2> ...]

Create subvolume(s) at the destination(s).

If *dest* part of the path is not given, subvolume *name* will be created in the current directory.

If multiple destinations are given, then the given options are applied to all subvolumes.

If failure happens for any of the destinations, the command would still retry the remaining destinations, but would return 1 to indicate the failure (similar to what **mkdir** would do.

`Options`

-i *<qgroupid>*

Add the newly created subvolume to a qgroup. This option can be given multiple times.

-p|--parents

Create any missing parent directories for each argument (like **mkdir -p**).

delete [options] [<subvolume> [<subvolume>...]], delete -i|--subvolid <subvolid> <path>

Delete the subvolume(s) from the filesystem.

If *subvolume* is not a subvolume, btrfs returns an error but continues if there are more arguments to process.

If *--subvolid* is used, *path* must point to a btrfs filesystem. See btrfs subvolume list or btrfs inspect-internal rootid how to get the subvolume id.

The corresponding directory is removed instantly but the data blocks are removed later in the background. The command returns immediately. See btrfs subvolume sync how to wait until the subvolume gets completely removed.

The deletion does not involve full transaction commit by default due to performance reasons. As a consequence, the subvolume may appear again after a crash. Use one of the *--commit* options to wait until the operation is safely stored on the device.

Deleting subvolume needs sufficient permissions, by default the owner cannot delete it unless it's enabled by a mount option *user_subvol_rm_allowed*, or deletion is run as root. The default subvolume (see btrfs subvolume set-default) cannot be deleted and returns error (EPERM) and this is logged to the system log. A subvolume that's currently involved in send (see btrfs-send(8)) also cannot be deleted until the send is finished. This is also logged in the system log.

`Options`

-c|--commit-after

wait for transaction commit at the end of the operation.

-C|--commit-each

wait for transaction commit after deleting each subvolume.

-i|--subvolid <subvolid>

subvolume id to be removed instead of the <path> that should point to the filesystem with the subvolume

-R|--recursive

delete subvolumes beneath each subvolume recursively

This requires either *CAP_SYS_ADMIN* or the filesystem must be mounted with *user_subvol_rm_allowed* mount option. In the unprivileged case, subvolumes which cannot be accessed are skipped. The deletion is not atomic.

-v|--verbose

(deprecated) alias for global *-v* option

find-new <subvolume> <last_gen>

List the recently modified files in a subvolume, after *last_gen* generation.

get-default <path>

Get the default subvolume of the filesystem *path*.

The output format is similar to **subvolume list** command.

list [options] [-G [+|-]<value>] [-C [+|-]<value>] [--sort=rootid,gen,ogen,path] <path>

List the subvolumes present in the filesystem *path*.

For every subvolume the following information is shown by default:

ID *ID* gen *generation* top level *parent_ID* path *path*

where *ID* is subvolume's (root)id, *generation* is an internal counter which is updated every transaction, *parent_ID* is the same as the parent subvolume's id, and *path* is the relative path of the subvolume to the top level subvolume. The subvolume's ID may be used by the subvolume set-default command, or at mount time via the *subvolid=* option.

`Options`

Path filtering:

-o

Print only subvolumes below specified <path>. Note that this is not a recursive command, and won't show nested subvolumes under <path>.

-a

print all the subvolumes in the filesystem and distinguish between absolute and relative path with respect to the given *path*.

Field selection:

-p

print the parent ID (*parent* here means the subvolume which contains this subvolume).

-c

print the ogeneration of the subvolume, aliases: ogen or origin generation.

-g

print the generation of the subvolume (default).

-u

print the UUID of the subvolume.

-q

print the parent UUID of the subvolume (*parent* here means subvolume of which this subvolume is a snapshot).

-R

print the UUID of the sent subvolume, where the subvolume is the result of a receive operation.

Type filtering:

-s

only snapshot subvolumes in the filesystem will be listed.

-r

only readonly subvolumes in the filesystem will be listed.

-d

list deleted subvolumes that are not yet cleaned.

Other:

-t

print the result as a table.

Sorting:

By default the subvolumes will be sorted by subvolume ID ascending.

-G [+|-]<value>

list subvolumes in the filesystem that its generation is >=, <= or = value. '+' means >= value, '-' means <= value, If there is neither '+' nor '-', it means = value.

-C [+|-]<value>

list subvolumes in the filesystem that its ogeneration is >=, <= or = value. The usage is the same to *-G* option.

--sort=rootid,gen,ogen,path

list subvolumes in order by specified items. you can add + or - in front of each items, + means ascending, - means descending. The default is ascending.

for *--sort* you can combine some items together by ,, just like *--sort=+ogen,-gen,path,rootid*.

set-default [<subvolume>|<id> <path>]

Set the default subvolume for the (mounted) filesystem.

Set the default subvolume for the (mounted) filesystem at *path*. This will hide the top-level subvolume (i.e. the one mounted with *subvol=/* or *subvolid=5*). Takes action on next mount.

There are two ways how to specify the subvolume, by *id* or by the *subvolume* path. The id can be obtained from btrfs subvolume list btrfs subvolume show or btrfs inspect-internal rootid.

show [options] <path>

Show more information about a subvolume (UUIDs, generations, times, flags, related snapshots).

```
/mnt/btrfs/subvolume
        Name:                   subvolume
        UUID:                   5e076a14-4e42-254d-ac8e-55bebea982d1
        Parent UUID:            -
        Received UUID:          -
        Creation time:          2018-01-01 12:34:56 +0000
        Subvolume ID:           79
        Generation:             2844
        Gen at creation:        2844
        Parent ID:              5
        Top level ID:           5
        Flags:                  -
        Snapshot(s):
```

`Options`

-r|--rootid <ID>

show details about subvolume with root *ID*, looked up in *path*

-u|--uuid UUID

show details about subvolume with the given *UUID*, looked up in *path*

snapshot [-r] [-i <qgroupid>] <source> <dest>|[<dest>/]<name>

Create a snapshot of the subvolume *source* with the name *name* in the *dest* directory.

If only *dest* is given, the subvolume will be named the basename of *source*. If *source* is not a subvolume, btrfs returns an error.

`Options`

-r

Make the new snapshot read only.

-i *<qgroupid>*

Add the newly created subvolume to a qgroup. This option can be given multiple times.

sync <path> [subvolid...]

Wait until given subvolume(s) are completely removed from the filesystem after deletion. If no subvolume id is given, wait until all current deletion requests are completed, but do not wait for subvolumes deleted in the meantime.

If the filesystem status changes to read-only then the waiting is interrupted.

`Options`

-s *<N>*

sleep N seconds between checks (default: 1)

EXAMPLES

Deleting a subvolume

If we want to delete a subvolume called *foo* from a btrfs volume mounted at `/mnt/bar` we could run the following:

```
btrfs subvolume delete /mnt/bar/foo
```

EXIT STATUS

btrfs subvolume returns a zero exit status if it succeeds. A non-zero value 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-quota(8), btrfs-send(8), mkfs.btrfs(8), mount(8)

Previous Next