Btrfs

7 languages

Page Discussion Read View source View history Tools

From the **Btrfs Documentation**:

> Btrfs is a modern copy on write (COW) file system for Linux aimed at implementing advanced features while also focusing on fault tolerance, repair and easy administration.

Note: Like some other file systems, Btrfs is under continuous development. This means that specific features may not be ready for everyday use yet. To see if your use case might be affected, check the Btrfs Documentation's **Status** and the **#Known issues** section of this article.

Related articles

- **File systems**
- **Snapper**
- **Timeshift**
- **Yabsnap**

Appearance hide

Text

- Small
- Standard
- Large

Width

- Standard
- Wide

1 Preparation

For user space utilities, **install** the `btrfs-progs` package that is required for basic operations.

If you need to boot from a Btrfs file system (i.e., your kernel and initramfs reside on a Btrfs partition), check if your **boot loader** supports Btrfs.

2 File system creation

The following shows how to create a new Btrfs **file system**. To convert an Ext3/4 partition to Btrfs, see **#Ext3/4 to Btrfs conversion**. To use a partitionless setup, see **#Partitionless Btrfs disk**.

See `mkfs.btrfs(8)` for more information.

2.1 File system on a single device

To create a Btrfs file system on partition `/dev/partition`:

```
# mkfs.btrfs -L mylabel /dev/partition
```

The Btrfs default nodesize for metadata is 16 KiB, while the default sectorsize for data is equal to page size and autodetected. To use a larger nodesize for metadata (must be a multiple of sectorsize, up to 64 KiB is allowed), specify a value for the `nodesize` via the `-n` switch as shown in this example using 32 KiB blocks:

```
# mkfs.btrfs -L mylabel -n 32k /dev/partition
```

Note: According to **`mkfs.btrfs(8)` § OPTIONS**, "[a] smaller node size increases fragmentation but leads to taller b-trees which in turn leads to lower locking contention. Higher node sizes give better packing and less fragmentation at the cost of more expensive memory operations while updating the metadata blocks".

2.2 Multi-device file system

Warning:

- The RAID 5 and RAID 6 modes of Btrfs are fatally flawed, and should not be used for "anything but testing with throw-away data." **List of known problems and partial workarounds**. See **`btrfs(5)` § RAID56 STATUS AND RECOMMENDED PRACTICES** for status updates.
- By default, **systemd** disables **CoW** for `/var/log/journal`, which can cause data corruption on RAID 1 (see **#Disabling CoW**). To prevent this, create an empty file `/etc/tmpfiles.d/journal-nocow.conf` to override `/usr/lib/tmpfiles.d/journal-nocow.conf` (see **`tmpfiles.d(5)` § CONFIGURATION DIRECTORIES AND PRECEDENCE**).

Multiple devices can be used to create a RAID. Supported RAID levels include RAID 0, RAID 1, RAID 10, RAID 5 and RAID 6. Starting from kernel 5.5 RAID1c3 and RAID1c4 for 3- and 4- copies of RAID 1 level. The RAID levels can be configured separately for data and metadata using the `-d` and `-m` options respectively. By default, the data has one copy (`single`) and the metadata is mirrored (`raid1`). This is similar to creating a **JBOD configuration**, where disks are seen as one file system, but files are not duplicated. See **Using Btrfs with Multiple Devices** for more information about how to create a Btrfs RAID volume.

```
# mkfs.btrfs -d single -m raid1 /dev/part1 /dev/part2 ...
```

You must include either the `udev` hook, `systemd` hook or the `btrfs` hook in `/etc/mkinitcpio.conf` in order to use multiple Btrfs devices in a pool. See the **Mkinitcpio#Common hooks** article for more information.

Once the file system is created, it is advised to run the following command to scan for multi-device Btrfs file systems and register them, allowing to mount the multi-device file system by specifying only one member:

```
# btrfs device scan
```

Note:

- It is possible to add devices to a multiple-device file system later on. See the **Btrfs wiki article** for more information.
- Devices can be of different sizes. However, if one drive in a RAID configuration is bigger than the others, this extra space will not be used.
- Some **boot loaders** such as **Syslinux** do not support multi-device file systems.
- Btrfs does not automatically read from the fastest device, so mixing different kinds of disks results in inconsistent performance. See **[1]** for details.

See **#RAID** for advice on maintenance specific to multi-device Btrfs file systems.

2.2.1 Profiles

Btrfs uses the concept of **profiles** to configure mirroring, parity, and striping. In standard **RAID** terminology, this is called *RAID level*. Profiles for metadata (the `-m` option of **`mkfs.btrfs(8)`**) and data (the `-d` option of **`mkfs.btrfs(8)`**) may be different for the same Btrfs file system.

Some notable profiles:

single
: No mirroring, no striping, no parity, enables mapping several devices to a single file system, called `LINEAR` in **mdadm** terminology.

raid0
: No mirroring, striping, no parity, enables parallel access between devices, but not limited to same-size devices like traditional **mdadm** RAID.

raid1
: Mirroring, no striping, no parity, enables recovering from one drive failure.

3 Configuring the file system

3.1 Copy-on-Write (CoW)

By default, Btrfs uses **copy-on-write** for all files all the time. Writes do not overwrite data in place; instead, a modified copy of the block is written to a new location, and metadata is updated to point at the new location. See the **Btrfs Sysadmin Guide section** for implementation details, as well as advantages and disadvantages.

3.1.1 Disabling CoW

Warning: Disabling CoW in Btrfs also disables checksums. Btrfs will not be able to detect corrupted `nodatacow` files. When combined with RAID 1, power outages or other sources of corruption can cause the data to become out of sync.

To disable copy-on-write for newly created files in a mounted subvolume, use the `nodatacow` mount option. This will only affect newly created files. Copy-on-write will still happen for existing files. The `nodatacow` option also disables compression. See **`btrfs(5)`** for details.

Note: From **`btrfs(5)` § MOUNT OPTIONS**: "within a single file system, it is not possible to mount some subvolumes with `nodatacow` and others with `datacow`. The mount option of the first mounted subvolume applies to any other subvolumes."

To disable copy-on-write for single files/directories, do:

```
$ chattr +C /dir/file
```

Note: From **Btrfs documentation: File attributes**: "When set on a directory, all newly created files will inherit this attribute. Due to implementation limitations, this flag can be set/unset only on empty files."

Tip: In accordance with the note above, you can use the following trick to disable copy-on-write on existing files in a directory:

```
$ mv /path/to/dir /path/to/dir_old
$ mkdir /path/to/dir
$ chattr +C /path/to/dir
```

```
$ cp -a --reflink=never /path/to/dir_old/. /path/to/dir
$ rm -rf /path/to/dir_old
```

Make sure that the data are not used during this process. Also note that `mv` or `cp` without `--reflink=never` as described below will not work.

3.1.1.1 Effect on snapshots

If a file has copy-on-write disabled (NOCOW) and a **snapshot** is taken, the first write to a file block after the snapshot **will be a COW operation** because the snapshot locks the old file blocks in place. However, the file will retain the NOCOW attribute and any subsequent writes to the same file block will be in-place until the next snapshot.

Frequent snapshots can reduce the effectiveness of NOCOW, as COW is still required on the first write. To avoid copy-on-write for such files altogether, put them in a separate subvolume and do not take snapshots of that subvolume.

3.2 Compression

Btrfs supports **transparent and automatic compression**. This reduces the size of files as well as significantly increases the lifespan of flash-based media by reducing write amplification. See **Fedora:Changes/BtrfsByDefault#Compression**, **[2]**, and **[3]**. It can also **improve performance**, in some cases (e.g. single thread with heavy file I/O), while obviously harming performance in other cases (e.g. multi-threaded and/or CPU intensive tasks with large file I/O). Better performance is generally achieved with the fastest compress algorithms, *zstd* and *lzo*, and some **benchmarks** provide detailed comparisons.

LZO has a fixed compression level, whereas zlib and zstd have a range of levels from 1 (low compression) to 9 (zlib) or 15 (zstd); see **btrfs(5) § COMPRESSION**. Changing the levels will affect CPU and I/O throughput differently, so they should be checked / benchmarked before and after changing.

The `compress=alg[:level]` mount option enables automatically considering every file for compression, where `alg` is either `zlib`, `lzo`, `zstd`, or `no` (for no compression). Using this option, Btrfs will check if compressing the first portion of the data shrinks it. If it does, the entire write to that file will be compressed. If it does not, none of it is compressed. With this option, if the first portion of the write does not shrink, no compression will be applied to the write even if the rest of the data would shrink tremendously. **[4]** This is done to prevent making the disk wait to start writing until all of the data to be written is fully given to Btrfs and compressed.

This article or section needs expansion.

Reason: Missing reference for the "empirical testing" mentioned in the following paragraph. (Discuss in **Talk:Btrfs**)

The `compress-force=alg[:level]` mount option can be used instead, which makes Btrfs skip checking if compression shrinks the first portion, and enables automatic compression try for every file. In a worst-case scenario, this can cause (slightly) more CPU usage for no purpose. However, empirical testing on multiple mixed-use systems showed a significant improvement of about 10% disk compression from using `compress-force=zstd` over just `compress=zstd`, which also had 10% disk compression. However, keep in mind that forcing compression is against **official Btrfs guidelines**.

Only files created or modified after the mount option is added will be compressed.

To apply compression to existing files, use the `btrfs filesystem defragment -calg` command, where `alg` is either `zlib`, `lzo` or `zstd`. For example, in order to re-compress the whole file system with **zstd**, run the following command:

```
# btrfs filesystem defragment -r -v -czstd /
```

Warning: Defragmenting a file which has a COW copy (either a snapshot copy or one made with `cp` or bcp) plus using the `-c` switch with a compression algorithm may result in two unrelated files effectively increasing the disk usage.

To enable compression when installing Arch to an empty Btrfs partition, use the `compress` option when **mounting** the file system: `mount -o compress=zstd /dev/sdxY /mnt/`. During configuration, add `compress=zstd` to the mount options of the root file system in **fstab**.

Tip: Compression can also be enabled per-file without using the `compress` mount option; to do so, apply `chattr +c` **or** `btrfs property set file compression alg` to the file. The first command is using legacy interface of file attributes inherited from ext2 filesystem and is not flexible, so by default the `zlib` compression is set. The other command sets a property on the file with the given algorithm. (Note: setting level that way is not yet implemented.) When applied to directories, it will cause new files to be automatically compressed as they come.

Warning:

- Systems using older kernels or **btrfs-progs** without `zstd` support may be unable to read or repair your file system if you use this option.
- **GRUB** introduced *zstd* support in 2.04. Make sure you have actually upgraded the bootloader installed in your MBR/ESP since then, by running `grub-install` with the appropriate options for your BIOS/UEFI setup, since that is not done automatically. See **FS#63235**.

3.2.1 View compression types and ratios

compsize takes a list of files (or an entire Btrfs file system) and measures compression types used and effective compression ratios. Uncompressed size may not match the number given by other programs such as **du(1)**, because every extent is counted once, even if it is reflinked several times, and even if a part of it is no longer used anywhere but has not been garbage collected. The `-x` option keeps it on a single file system, which is useful in situations like `compsize -x /` to avoid it from attempting to look in non-Btrfs subdirectories and fail the entire run.

3.3 Subvolumes

"A Btrfs subvolume is not a block device (and cannot be treated as one) instead, a Btrfs subvolume can be thought of as a POSIX file namespace. This namespace can be accessed via the top-level subvolume of the file system, or it can be mounted in its own right." **[5]**

Each Btrfs file system has a top-level subvolume with ID 5. This subvolume cannot be removed or replaced by another subvolume. The top-level subvolume has path `/` on the file system and other subvolumes are *nested* below the top-level subvolume. However, subvolumes can be moved around in the file system and their path may change, whereas their ID cannot.

By default, the top-level subvolume is mounted when mounting the file system. Options allow to **mount a specific subvolume** instead.

A major use case for subvolumes are **snapshots**.

See the following links for more details:

- **Btrfs Documentation**
- **Btrfs Wiki SysadminGuide#Subvolumes**
- **Btrfs Wiki Trees**

3.3.1 Creating a subvolume

To create a subvolume, the Btrfs file system must be mounted. The subvolume's name is set using the last argument.

```
# btrfs subvolume create /path/to/subvolume
```

Note: You can use `--parents` to automatically create parent directories if they do not exist.

3.3.2 Listing subvolumes

To see a list of all subvolumes of the filesystem that `path` belongs to:

```
# btrfs subvolume list -t path
```

`-t` triggers a more readable table view.

3.3.3 Deleting a subvolume

To delete a subvolume:

```
# btrfs subvolume delete /path/to/subvolume
```

If the subvolume contains other subvolumes that should be deleted, add the `-R`/`--recursive` option. Alternatively, a subvolume can be deleted like a regular directory (`rm -r`, `rmdir`).

Warning: Make sure that a subvolume is not mounted before deleting it. Deleting a mounted subvolume can lead to filesystem inconsistencies.

3.3.4 Mounting subvolumes

Subvolumes can be mounted like file system partitions using the `subvol=/path/to/subvolume` or `subvolid=objectid` mount flags. For example, you could have a subvolume named `subvol_root` and mount it as `/`. One can mimic traditional file system partitions by creating various subvolumes under the top level of the file system and then mounting them at the appropriate mount points. It is preferable to mount using `subvol=/path/to/subvolume`, rather than the subvolid, as the subvolid may change when restoring **#Snapshots**, requiring a change of mount configuration.

Tip: Changing subvolume layouts is made simpler by not using the top-level subvolume (ID=5) as `/` (which is done by default). Instead, consider creating a subvolume to house your actual data and mounting it as `/`.

Note: From **btrfs(5) § MOUNT OPTIONS**: "Most mount options apply to the **whole file system**, and only the options for the first subvolume to be

mounted will take effect. This is due to lack of implementation and may change in the future.

See the **Btrfs Wiki FAQ** for which mount options can be used per subvolume.

See **Snapper#Suggested filesystem layout**, **Btrfs SysadminGuide#Managing Snapshots**, and **SysadminGuide#Layout** for example file system layouts using subvolumes.

See `btrfs(5)` for a full list of Btrfs-specific mount options.

3.3.5 Mounting subvolume as root

To use a subvolume as the root mountpoint, either make it the **default subvolume**, or specify the subvolume via a **kernel parameter** using `rootflags=subvol=/path/to/subvolume`. Edit the root mountpoint in `/etc/fstab` and specify the mount option `subvol=`. Alternatively, the subvolume can be specified with its id, `rootflags=subvolid=objectid` as kernel parameter and `subvolid=objectid` as mount option in `/etc/fstab`. It is preferable to mount using `subvol=/path/to/subvolume`, rather than the subvolid, as the subvolid may change when restoring **#Snapshots**, requiring a change of mount configuration, or else the system will not boot.

3.3.6 Changing the default sub-volume

The default sub-volume is mounted if no `subvol=` mount option is provided. To change the default subvolume, do:

```
# btrfs subvolume set-default subvolume-id /
```

where *subvolume-id* can be found by **listing**.

Note: After changing the default subvolume on a system with **GRUB**, you should run `grub-install` again to notify the bootloader of the changes. See **this forum thread**.

Changing the default subvolume with `btrfs subvolume set-default` will make the top level of the file system inaccessible, except by use of the `subvol=/` or `subvolid=5` mount options **[6]**.

3.4 Quota

Warning: Qgroup is not stable yet and combining quota with (too many) snapshots of subvolumes can cause performance problems, for example when deleting snapshots. Plus there are several more **known issues**.

Quota support in Btrfs is implemented at a subvolume level by the use of quota groups or qgroup: Each subvolume is assigned a quota groups in the form of *0/subvolume_id* by default. However, it is possible to create a quota group using any number if desired.

To use qgroups, you need to enable quota first using

```
# btrfs quota enable path
```

From this point onwards, newly created subvolumes will be controlled by those groups. In order to retrospectively enable them for already existing subvolumes, enable quota normally, then create a qgroup (quota group) for each of those subvolume using their *subvolume_id* and rescan them:

```
# btrfs subvolume list path | cut -d' ' -f2 | xargs -I{} -n1 btrfs qgroup create 0/{} path
# btrfs quota rescan path
```

Quota groups in Btrfs form a tree hierarchy, whereby qgroups are attached to subvolumes. The size limits are set per qgroup and apply when any limit is reached in tree that contains a given subvolume.

Limits on quota groups can be applied either to the total data usage, un-shared data usage, compressed data usage or both. File copy and file deletion may both affect limits since the unshared limit of another qgroup can change if the original volume's files are deleted and only one copy is remaining. For example, a fresh snapshot shares almost all the blocks with the original subvolume, new writes to either subvolume will raise towards the exclusive limit, deletions of common data in one volume raises towards the exclusive limit in the other one.

To apply a limit to a qgroup, use the command `btrfs qgroup limit`. Depending on your usage, either use a total limit, unshared limit (`-e`) or compressed limit (`-c`). To show usage and limits for a given path within a file system, use

```
# btrfs qgroup show -reF path
```

3.5 Commit interval

The resolution at which data are written to the file system is dictated by Btrfs itself and by system-wide settings. Btrfs defaults to a 30 seconds checkpoint interval in which new data are committed to the file system. This can be changed by appending the `commit` mount option in `/etc/fstab` for the Btrfs partition.

```
LABEL=arch64 / btrfs defaults,compress=zstd,commit=120 0 0
```

System-wide settings also affect commit intervals. They include the files under `/proc/sys/vm/*` and are out-of-scope of this wiki article. The kernel documentation for them is available at **https://docs.kernel.org/admin-guide/sysctl/vm.html**.

3.6 SSD TRIM

A Btrfs file system is able to free unused blocks from an SSD drive supporting the TRIM command. Asynchronous discard support is available with mount option `discard=async`, and is enabled by default as of `linux` 6.2. Freed extents are not discarded immediately, but grouped together and trimmed later by a separate worker thread, improving commit latency.

Asynchronous discard can safely be used alongside periodic trim **[7]**.

More information about enabling and using TRIM can be found in **Solid state drive#TRIM**.

4 Usage

4.1 Swap file

Note: Swap files on file systems spanning multiple devices are not supported. See `btrfs(5) § SWAPFILE SUPPORT` for all limitations.

To properly initialize a swap file, first create a *non-snapshotted* subvolume to host the file, e.g.

```
# btrfs subvolume create /swap
```

Tip: Consider creating the subvolume directly below the top-level subvolume, e.g. `@swap`. Then, make sure the subvolume is **mounted** to `/swap` (or any other accessible location).

Create the swap file:

```
# btrfs filesystem mkswapfile --size 4g --uuid clear /swap/swapfile
```

If `--size` is omitted, the default of 2GiB is used.

Activate the swap file:

```
# swapon /swap/swapfile
```

Finally, edit the **fstab** configuration to add an entry for the swap file:

```
/etc/fstab
```

```
/swap/swapfile none swap defaults 0 0
```

For additional information, see **fstab#Usage**.

Note: You can also create the swap file manually, by **setting the** `No_COW` attribute on the whole subvolume with `chattr`, then following the steps in **Swap#Swap file creation**. See `btrfs(5) § SWAPFILE SUPPORT` for an alternative method.

To use a swap file for hibernation, you might need to additionally follow the procedures described in **Hibernation**.

4.2 Displaying used/free space

General linux userspace tools such as `df(1)` will inaccurately report free space on a Btrfs partition. It is recommended to use `btrfs filesystem usage` to query Btrfs partitions. For example, for a full breakdown of device allocation and usage stats:

```
# btrfs filesystem usage /
```

Note: The `btrfs filesystem usage` command does not currently work correctly with `RAID5/RAID6` RAID levels.

Alternatively, `btrfs filesystem df` allows a quick check on usage of allocated space without the requirement to run as root:

```
$ btrfs filesystem df /
```

See [8] for more information.

The same limitations apply to tools which analyze space usage for some subset of the file system, such as `du(1)` or `ncdu(1)`, as they do not take into account reflinks, snapshots and compression. Instead, see `btdu`[AUR] and **compsize** for Btrfs-aware alternatives.

4.3 Defragmentation

Btrfs supports online defragmentation through the mount option `autodefrag`; see `btrfs(5) § MOUNT OPTIONS`. To manually defragment your root, use:

```
# btrfs filesystem defragment -r /
```

Using the above command without the `-r` switch will result in only the metadata held by the subvolume containing the directory being defragmented. This allows for single file defragmentation by simply specifying the path.

Warning: Defragmenting a file which has a COW copy (either a snapshot copy or one made with `cp` or bcp) plus using the `-c` switch with a compression algorithm may result in two unrelated files effectively increasing the disk usage.

4.4 RAID

Btrfs offers native "RAID" for **#Multi-device file systems**. Notable features which set Btrfs RAID apart from **mdadm** are self-healing redundant arrays and online balancing. See **the Btrfs wiki page** for more information. The Btrfs sysadmin page also **has a section** with some more technical background.

Warning: Parity RAID (RAID 5/6) code has multiple serious data-loss bugs in it. See the Btrfs Wiki's **RAID5/6 page** and a bug report on **linux-btrfs mailing list** for more detailed information. In June 2020, somebody posted a **comprehensive list of current issues** and a **helpful recovery guide**.

4.4.1 Scrub

This article or section needs expansion.

Reason: Does Btrfs scrub examine subvolume or the device that subvolume resides on? (Discuss in Talk:Btrfs)

The **Btrfs Wiki Glossary** says that Btrfs scrub is "[a]n online file system checking tool. Reads all the data and metadata on the file system and uses checksums and the duplicate copies from RAID storage to identify and repair any corrupt data."

Note: A running scrub process will prevent the system from suspending, see **this thread** for details.

4.4.1.1 Start manually

To start a (background) scrub on the file system which contains `/`:

```
# btrfs scrub start /
```

To check the status of a running scrub:

```
# btrfs scrub status /
```

4.4.1.2 Start with a service or timer

The `btrfs-progs` package brings the `btrfs-scrub@.timer` unit for monthly scrubbing the specified mountpoint. **Enable** the timer with an escaped path, e.g. `btrfs-scrub@-.timer` for `/` and `btrfs-scrub@home.timer` for `/home`. You can use `systemd-escape -p /path/to/mountpoint` to escape the path; see `systemd-escape(1)` for details.

You can also run the scrub by **starting** `btrfs-scrub@.service` (with the same encoded path). The advantage of this over `btrfs scrub` (as the root user) is that the results of the scrub will be logged in the **systemd journal**.

On large NVMe drives with insufficient cooling (e.g. in a laptop), scrubbing can read the drive fast enough and long enough to get it very hot. If you are running scrubs with systemd, you can easily limit the rate of scrubbing with the `IOReadBandwidthMax` option described in `systemd.resource-control(5)` by using a **drop-in file**.

4.4.2 Balance

"A balance passes all data in the file system through the allocator again. It is primarily intended to rebalance the data in the file system across the devices when a device is added or removed. A balance will regenerate missing copies for the redundant RAID levels, if a device has failed." [9] See **Upstream FAQ page**.

On a single-device filesystem, a balance may be also useful for (temporarily) reducing the amount of allocated but unused (meta)data chunks. Sometimes this is needed for fixing **"file system full" issues**.

```
# btrfs balance start --bg /
# btrfs balance status /
```

4.5 Snapshots

"A snapshot is simply a subvolume that shares its data (and metadata) with some other subvolume, using Btrfs's COW capabilities." See **Btrfs Wiki SysadminGuide#Snapshots** for details.

To create a snapshot:

```
# btrfs subvolume snapshot source [dest/]name
```

To create a readonly snapshot, add the `-r` flag. To create a writable version of a readonly snapshot, simply create a snapshot of it.

Note:

- It is possible for a snapshot to be converted in-place from readonly to writeable with `btrfs property set -f -ts '/path/to/snapshot' ro false`. However, this is not recommended because it **causes issues** with any future incremental send/receive. Making a new writeable snapshot prevents such issues.
- Snapshots are not recursive. Every nested subvolume will be an empty directory inside the snapshot.

4.6 Send/receive

A subvolume can be sent to stdout or a file using the `send` command. This is usually most useful when piped to a Btrfs `receive` command. For example, to send a snapshot named `/root_backup` (perhaps of a snapshot you made of `/` earlier) to `/backup`, you would do the following:

```
# btrfs send /root_backup | btrfs receive /backup
```

The snapshot that is sent *must* be readonly. The above command is useful for copying a subvolume to an external device (e.g. a USB disk mounted at `/backup` above).

The subvolume will be created on the receiving end. It does not need to be created manually.

Another example which creates: `/mnt/arch-v2/subvolumes/@var`:

```
# btrfs send --proto 2 --compressed-data '/mnt/arch/snapshots/@var' | btrfs receive '/mnt/arch-v2/subvolume
s/'
```

The parameters `--proto 2` and `--compressed-data` used in the example might be useful for more efficient sending (assumes compressed data).

You can also send only the difference between two snapshots. For example, if you have already sent a copy of `root_backup` above and have made a new readonly snapshot on your system named `root_backup_new`, then to send only the incremental difference to `/backup`, do:

```
# btrfs send -p /root_backup /root_backup_new | btrfs receive /backup
```

Now, a new subvolume named `root_backup_new` will be present in `/backup`.

See **Btrfs Wiki's Incremental Backup page** and **#Incremental backup to external drive** on how to use this for incremental backups and for tools that automate the process.

4.7 Deduplication

Using copy-on-write, Btrfs is able to copy files or whole subvolumes without actually copying the data. However, whenever a file is altered, a new *proper* copy is created. Deduplication takes this a step further by actively identifying blocks of data which share common sequences and combining them into an extent with the same copy-on-write semantics.

Tools dedicated to deduplicate a Btrfs formatted partition include `duperemove` and `bees`. One may also want to merely deduplicate data on a file based level instead using e.g. `rmlint-git`[AUR], `rdfind`, `jdupes`[AUR] or `dduper-git`[AUR]. For an overview of available features of those programs and additional information, have a look at the **upstream Wiki entry**.

4.8 Resizing

Warning: To avoid data loss, ensure that you back up your data before you begin any resizing task.

You can grow a file system to the maximum space available on the device, or specify an exact size. Ensure that you grow the size of the device or logical volume before you attempt to increase the size of the file system. When specifying an exact size for the file system on a device, either increasing or decreasing, ensure that the new size satisfies the following conditions:

- The new size must be greater than the size of the existing data; otherwise, data loss occurs.
- The new size must be equal to or less than the current device size because the file system size cannot extend beyond the space available.

Note: If you plan to also decrease the size of the logical volume that holds the file system, ensure that you decrease the size of the file system before you attempt to decrease the size of the device or logical volume.

To extend the file system size to the maximum available size of the device:

```
# btrfs filesystem resize max /
```

To extend the file system to a specific size:

```
# btrfs filesystem resize size /
```

Replace `size` with the desired size in bytes. You can also specify units on the value, such as K (kibibytes), M (mebibytes), or G (gibibytes). Alternatively, you can specify an increase or decrease to the current size by prefixing the value with a plus (+) or a minus (-) sign, respectively:

```
# btrfs filesystem resize +size /
# btrfs filesystem resize -size /
```

5 Known issues

A few limitations should be known before trying.

5.1 Encryption

Btrfs has no built-in encryption support, but this **may** come in the future. Users can encrypt the partition before running `mkfs.btrfs`, see **dm-crypt/Encrypting an entire system**. Another approach is **stacked filesystem encryption**.

5.2 btrfs check issues

The tool `btrfs check` has known issues and should not be run without further reading; see section **#btrfs check**.

6 Tips and tricks

6.1 Partitionless Btrfs disk

Warning: This type of setup is not ideal for a boot device and it is recommended to instead set up a Btrfs partition along with a separate **EFI system partition**. Furthermore, GRUB strongly discourages installation to a partitionless disk.

Btrfs can occupy an entire data storage device, replacing the **MBR** or **GPT** partitioning schemes, using **subvolumes** to simulate partitions. However, using a partitionless setup is not required to simply **create a Btrfs file system** on an existing **partition** that was created using another method. There are some limitations to partitionless single disk setups:

- Cannot place other **file systems** on another partition on the same disk.
- Due to the previous point, having an **ESP** on this disk is not possible. Another device is necessary for **UEFI** boot.

To overwrite the existing partition table with Btrfs, run the following command:

```
# mkfs.btrfs /dev/sdX
```

For example, use `/dev/sda` rather than `/dev/sda1`. The latter would format an existing partition instead of replacing the entire partitioning scheme. Because the root partition is Btrfs, make sure `btrfs` is compiled into the kernel, or put `btrfs` into **mkinitcpio.conf#MODULES** and **regenerate the initramfs**.

Install the **boot loader** like you would for a data storage device with a **Master Boot Record**. See **Syslinux#Manually** or **GRUB/Tips and tricks#Install to partition or partitionless disk**. If your kernel does not boot due to `Failed to mount /sysroot.`, please add `GRUB_PRELOAD_MODULES="btrfs"` in `/etc/default/grub` and **generate** the grub configuration.

6.2 Ext3/4 to Btrfs conversion

Warning: There are many reports on the btrfs mailing list about incomplete/corrupt/broken conversions. Make sure you have *working* backups of any data you cannot afford to lose. See the **Convert** page on the Btrfs wiki for more information.

Boot from an install CD, then convert by doing:

```
# btrfs-convert /dev/partition
```

Mount the partition and test the conversion by checking the files. Be sure to change the `/etc/fstab` to reflect the change (**type** to `btrfs` and **fs_passno** [the last field] to `0` as Btrfs does not do a file system check on boot). Also note that the UUID of the partition will have changed, so update fstab accordingly when using UUIDs. `chroot` into the system and rebuild your boot loader's menu list (see **Install from existing Linux**). If converting a root file system, while still chrooted, run `mkinitcpio -p linux` to regenerate the initramfs or the system will not successfully boot.

Note: If there is anything wrong, either unable to mount or write files to the newly converted Btrfs; there is always the option to rollback as long as the backup subvolume `/ext2_saved` is still there. Use the `btrfs-convert -r /dev/partition` command to rollback; this will discard any modifications to the newly converted Btrfs file system.

After confirming that there are no problems, complete the conversion by deleting the backup `ext2_saved` sub-volume. Note that you cannot revert back to ext3/4 without it.

```
# btrfs subvolume delete /ext2_saved
```

Finally, **balance** the file system to reclaim the space.

Remember that some applications which were installed prior have to be adapted to Btrfs.

Note: Ext3/4 to Btrfs conversion is a time consuming operation. For a 4 TiB file system and a regular HDD, it can take up to 10 hours.

6.3 Corruption recovery

Warning: The tool `btrfs check` has known issues, see section **#btrfs check**

btrfs-check cannot be used on a mounted file system. To be able to use *btrfs-check* without booting from a live USB, add it to the initial ramdisk:

```
/etc/mkinitcpio.conf

BINARIES=(btrfs)
```

Regenerate the initramfs.

Then if there is a problem booting, the utility is available for repair.

Note: If the fsck process has to invalidate the space cache (and/or other caches?), it is normal for a subsequent boot to hang up for a while (it may give console messages about btrfs-transaction being hung). The system should recover from this after a while.

See **btrfs-check(8)** for more information.

6.4 Booting into snapshots

In order to boot into a snapshot, the same procedure applies as for mounting a subvolume as your root partition, as given in section **mounting a subvolume as your root partition**, because snapshots can be mounted like subvolumes.

- If using **GRUB**, you can automatically populate your boot menu with Btrfs snapshots when regenerating the configuration file with the help of **grub-btrfs** or **grub-btrfs-git**[AUR].
- If using **rEFInd**, you can automatically populate your boot menu with Btrfs snapshots with the help of **refind-btrfs**[AUR], after **enabling** `refind-btrfs.service`.
- If using **Limine**, you can install **limine-snapper-sync**[AUR], which automatically generates snapshot entries in your boot menu whenever your **Snapper** list changes after enabling `limine-snapper-watcher.service`. See **Limine#Snapper snapshot integration for Btrfs** for more information.

6.5 Use Btrfs subvolumes with systemd-nspawn

See the **Systemd-nspawn#Use Btrfs subvolume as container root** and **Systemd-nspawn#Use temporary Btrfs snapshot of container** articles.

6.6 Reducing access time metadata updates

Because of the copy-on-write nature of Btrfs, simply accessing files can trigger the metadata copy and writing. Reducing the frequency of access time updates may eliminate this unexpected disk usage and increase performance. See **fstab#atime options** for the available options.

6.7 Incremental backup to external drive

The following packages use `btrfs send` and `btrfs receive` to send backups incrementally to an external drive. Refer to their documentation to see differences in implementation, features, and requirements.

- **btrbk** — Tool for creating snapshots and remote backups of Btrfs subvolumes.
  https://github.com/digint/btrbk || `btrbk`
- **snap-sync** — Use **Snapper** snapshots to backup to external drive or remote machine.
  https://github.com/wesbarnett/snap-sync || `snap-sync`
- **snapsync** — A synchronization tool for **Snapper**.
  https://github.com/doudou/snapsync || `ruby-snapsync`[AUR]

The following package allows backing up snapper snapshots to non-Btrfs file systems.

- **snapborg** — borgmatic-like tool which integrates snapper snapshots with `borg` backups.
  https://github.com/enzingerm/snapborg || `snapborg`[AUR]

6.8 Automatic snapshots

For the managing and automatic creation of Btrfs snapshot, one may use a snapshot manager such as **Snapper**, **Timeshift** or **Yabsnap**.

6.9 Automatic notifications

Desktop notifications help you notice serious Btrfs issues immediately, offering better awareness compared to having no notifications at all.

`btrfs-desktop-notification`[AUR] provides desktop notifications for the following events:

- Booting into any read-only snapshot or system.
- Btrfs warning, error, or fatal messages appearing in the dmesg log.

See **https://gitlab.com/Zesko/btrfs-desktop-notification** for more information and configuration.

7 Troubleshooting

See the **Btrfs Troubleshooting pages** and **Btrfs Problem FAQ** for general troubleshooting.

7.1 GRUB

7.1.1 Partition offset

The offset problem may happen when you try to embed `core.img` into a partitioned disk. It means that **it is OK** to embed GRUB's `core.img` into a Btrfs pool on a partitionless disk (e.g. `/dev/sdX`) directly.

GRUB can boot Btrfs partitions, however the module may be larger than other **file systems**. And the `core.img` file made by `grub-install` may not fit in the first 63 sectors (31.5KiB) of the drive between the MBR and the first partition. Up-to-date partitioning tools such as `fdisk` and `gdisk` avoid this issue by offsetting the first partition by roughly 1MiB or 2MiB.

7.1.2 Missing root

The factual accuracy of this article or section is disputed.

Reason: Suggests editing a non-configuration file manually. (Discuss in **Talk:Btrfs#Should not suggest to edit files in /usr/share**)

Users experiencing the following: `error no such device: root` when booting from a RAID style setup then edit /usr/share/grub/grub-mkconfig_lib and remove both quotes from the line `echo " search --no-floppy --fs-uuid --set=root ${hints} ${fs_uuid}"`. Regenerate the config for grub and the system should boot without an error.

Warning: Overwriting package files is discouraged and the file will be overwritten by next update.

7.2 Mounting timed out

Sometimes, especially with large RAID1 arrays, mounting might time out during boot with a journal message such as:

```
Jan 25 18:05:12 host systemd[1]: storage.mount: Mounting timed out. Terminating.
Jan 25 18:05:46 host systemd[1]: storage.mount: Mount process exited, code=killed, status=15/TERM
Jan 25 18:05:46 host systemd[1]: storage.mount: Failed with result 'timeout'.
Jan 25 18:05:46 host systemd[1]: Failed to mount /storage.
Jan 25 18:05:46 host systemd[1]: Startup finished in 32.943s (firmware) + 3.097s (loader) + 7.247s (kernel)>
Jan 25 18:05:46 host kernel: BTRFS error (device sda): open_ctree failed
```

This can easily be worked around by providing a longer timeout via the systemd-specific mount option `x-systemd.mount-timeout` in **fstab**. For example:

```
/dev/sda               /storage    btrfs       rw,relatime,x-systemd.mount-timeout=5min  0 0
```

7.3 BTRFS: open_ctree failed

The factual accuracy of this article or section is disputed.

Reason: The status is old. Replacing udev hook with systemd makes no sense since they both use systemd-udevd. (Discuss in **Talk:Btrfs**)

As of November 2014, there seems to be a bug in **systemd** or **mkinitcpio** causing the following error on systems with multi-device Btrfs file system using the `btrfs` hook in `mkinitcpio.conf`:

```
BTRFS: open_ctree failed
mount: wrong fs type, bad option, bad superblock on /dev/sdb2, missing codepage or helper program, or other e
rror

In some cases, useful info is found in syslog - try dmesg|tail or so.

You are now being dropped into an emergency shell.
```

A workaround is to remove `btrfs` from the `HOOKS` array in `/etc/mkinitcpio.conf` and instead add `btrfs` to the `MODULES` array. Then **regenerate the initramfs** and reboot.

You will get the same error if you try to mount a raid array without one of the devices. In that case, you must add the `degraded` mount option to `/etc/fstab`. If your root resides on the array, you must also add `rootflags=degraded` to your **kernel parameters**.

As of August 2016, a potential workaround for this bug is to mount the array by a single drive only in `/etc/fstab`, and allow Btrfs to discover and append the other drives automatically. Group-based identifiers such as UUID and LABEL appear to contribute to the failure. For example, a two-device RAID1 array consisting of 'disk1' and disk2' will have a UUID allocated to it, but instead of using the UUID, use only `/dev/mapper/disk1` in `/etc/fstab`. For a more detailed explanation, see the following **blog post**.

Another possible workaround is to remove the `udev` hook in **mkinitcpio.conf** and replace it with the `systemd` hook. In this case, `btrfs` should *not* be in the `HOOKS` or `MODULES` arrays.

See the **original forums thread** and **FS#42884** for further information and discussion.

7.4 btrfs check

This article or section is out of date.

Reason: The "heavy development" status is old. (Discuss in **Talk:Btrfs**)

Warning: Since Btrfs is under heavy development, especially the `btrfs check` command, it is highly recommended to create a **backup** and consult `btrfs-check(8)` before executing `btrfs check` with the `--repair` switch.

The `btrfs-check(8)` command can be used to check or repair an unmounted Btrfs file system. However, this repair tool is still immature and not able to repair certain file system errors even those that do not render the file system unmountable.

7.5 Constant drive activity

Since the **kernel** version 6.2, `discard=async` `mount(8)` option is set by default. This **has been reported** to cause constant drive activity on some drives even while idle, as the discard queue is filled faster than it is processed. This can cause increased power usage, especially on NVMe-based drives.

As of kernel version 6.3, the default discard `iops_limit` has been changed from 100 to 1000 to address this issue. You can manually set it to a desired value on an old kernel version, e.g.

```
# echo 1000 > /sys/fs/btrfs/uuid/discard/iops_limit
```

Where `uuid` is the UUID of the Btrfs file system. The limit of `1000` will need to be tuned experimentally.

To set the parameter on boot, **systemd-tmpfiles** may be used, e.g. by creating the following file:

```
/etc/tmpfiles.d/btrfs-discard.conf
```

```
w /sys/fs/btrfs/uuid/discard/iops_limit - - - - 1000
```

Alternatively, one can disable asynchronous discard altogether by mounting using the `nodiscard` mount option in **fstab**, and instead use **Periodic TRIM**.

7.6 Device total_bytes should be at most X but found Y

If a drive has been moved from another computer or the order of devices has changed, and the difference in sizes reported is tiny (at most megabytes), it is possible **HPA (Host Protected Area)** is in effect.

To verify if HPA is enabled, use **hdparm**:

```
# hdparm -N DEVICE
```

The output provides two numbers: the number of visible sectors and the actual number of sectors. If they differ, HPA is enabled.

If motherboard sets this forcefully and the firmware provides no option to turn that off, the only option is to shrink the affected file system. This is most easily done on the original computer, or on any machine that does not apply HPA.

7.7 No space left on device

The blog post **Fixing Btrfs Filesystem Full Problems** suggests and explains the following checks/steps:

1. Clear space now (remove historical snapshots)
2. Is your filesystem really full? Mis-balanced metadata and/or data chunks (run `btrfs balance`)
3. Is your filesystem really full? Mis-balanced data chunks
4. Is your filesystem really full? Mis-balanced metadata
5. Balance cannot run because the filesystem is full (temporarily add a device such as USB key or loop device to the btrfs filesystem using `btrfs device add` before running `btrfs balance`)

8 See also

- **Official site**
  - **Btrfs Documentation**
  - **Archived Btrfs Wiki**
- **Performance related**
  - **Btrfs on raw disks?**
  - **Varying leafsize and nodesize in Btrfs**
  - **Btrfs support for efficient SSD operation (data blocks alignment)**
  - **Is Btrfs optimized for SSDs?**
  - **Lzo vs. zLib**
- **Miscellaneous**
  - **Funtoo:BTRFS Fun**
  - **Avi Miller presenting Btrfs** at SCALE 10x, January 2012.
  - **Summary of Chris Mason's talk** from LFCS 2012
  - **Btrfs: stop providing a bmap operation to avoid swapfile corruptions** 2009-01-21
  - **Doing Fast Incremental Backups With Btrfs Send and Receive**

Category: File systems