Btrfs

Related articles

File systems

Snapper

dm-crypt/Encrypting an entire system#Btrfs subvolumes with swap

From **Wikipedia:Btrfs**:

> Btrfs (B-tree file system, pronounced as "butter F S", "better F S", "b-tree F S", or simply by spelling it out) is a file system based on the copy-on-write (COW) principle, initially designed at Oracle Corporation for use in Linux. The development of Btrfs began in 2007, and by August 2014, the file system's on-disk format has been marked as stable.

From **Btrfs Wiki (https://btrfs.wiki.kernel.org/index.php/Main_Page)**:

> Btrfs is a new copy on write (CoW) filesystem for Linux aimed at implementing advanced features while focusing on fault tolerance,

> repair and easy administration. Jointly developed at Oracle, Red Hat, Fujitsu, Intel, SUSE, STRATO and many others, Btrfs is licensed under the GPL and open for contribution from anyone.

Warning: Btrfs has some features that are considered experimental. See the Btrfs Wiki's **Status (https://btrfs.wiki.kernel.org/index.php/Status)**, **Is Btrfs stable? (https://btrfs.wiki.kernel.org/index.php/FAQ#Is_btrfs_stable.3F)** and **Getting started (https://btrfs.wiki.kernel.org/index.php/Getting_started)** for more detailed information. See the **#Known issues** section.

Contents

Preparation

The official kernels `linux (https://www.archlinux.org/packages/?name=linux)` and `linux-lts (https://www.archlinux.org/packages/?name=linux-lts)` include support for Btrfs. If you want to boot from a Btrfs file system, check if your **boot loader** supports Btrfs.

User space utilities are available by **installing** the `btrfs-progs (https://www.archlinux.org/packages/?name=btrfs-progs)` package.

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**.

File system on a single device

To format a partition do:

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

The Btrfs default blocksize is 16KB. To use a larger blocksize for data/metadata, specify a value for the `nodesize` via the `-n` switch as shown in this example using 16KB blocks:

```
# mkfs.btrfs -L mylabel -n 16k /dev/partition
```

Multi-device file system

Warning:

- The RAID 5, RAID 6 mode of Btrfs is considered *fatally flawed*, and should not be used for "anything but testing with throw-away data." See **the Btrfs page on RAID5 and RAID6 (https://btrfs.wiki.kernel.org/index.php/RAID56)** for status updates.
- Some **boot loaders** such as **Syslinux** do not support multi-device file systems.

Multiple devices can be entered to create a RAID. Supported RAID levels include RAID 0, RAID 1, RAID 10, RAID 5 and RAID 6. The RAID levels can be configured separately for data and metadata using the `-d` and `-m` options respectively. By default the data is striped (`raid0`) and the metadata is mirrored (`raid1`). See **Using Btrfs with Multiple Devices (https://btrfs.wiki.kernel.org/index.php/Using_Btrfs_with_Multiple_Devices)** for more information about how to create a Btrfs RAID volume as well as the manpage for `mkfs.btrfs`.

```
# mkfs.btrfs -d raid0 -m raid1 /dev/part1 /dev/part2 ...
```

You **must** include either the `udev` 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.

Note: If the disks in your multi-disk array have different sizes, this may not use the full capacity of all drives. In order to utilize the full capacity of all disks, use `-d single` instead of `-d raid0 -m raid1` (metadata mirrored, data not mirrored and not striped)

Note: Mounting such a filesystem may result in all but one of the according *.device*-jobs getting stuck and systemd never finishing startup due to a **bug (https://github.com/systemd/systemd/issues/1921)** in handling this type of filesystem.

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

Configuring the file system

Copy-on-Write (CoW)

By default, Btrfs uses **Wikipedia:copy-on-write** for all files all the time. See the **Btrfs Sysadmin Guide section (https://btrfs.wiki.kernel.org/index.php/SysadminGuide#Copy_on_Write_.28CoW.29)** for implementation details, as well as advantages and disadvantages.

Disabling CoW

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) (https://jlk.fjfi.cvut.cz/arch/manpages/man/btrfs.5)`** for details.

> **Note:** From Btrfs Wiki **Mount options (https://btrfs.wiki.kernel.org/index.php/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
```

This will disable copy-on-write for those operation in which there is only one reference to the file. If there is more than one reference (e.g. through `cp --reflink=always` or because of a filesystem snapshot), copy-on-write still occurs.

> **Note:** From chattr man page: "For btrfs, the 'C' flag should be set on new or empty files. If it is set on a file which already has data blocks, it is undefined when the blocks assigned to the file will be fully stable. If the

'C' flag is set on a directory, it will have no effect on the directory, but new files created in that directory will have the No_COW attribute."

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 /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 --reflink` as described below will not work.

Creating lightweight copies

By default, when copying files on a Btrfs filesystem with `cp`, actual copies are created. To create a lightweight copy referencing to the original data, use the *reflink* option:

```
$ cp --reflink source dest
```

See the man page on cp(1) (https://jlk.fjfi.cvut.cz/arch/manpages/man/cp.1) for more details on the `--reflink` flag.

Compression

Btrfs supports transparent compression, meaning every file on the partition is automatically compressed. This not only reduces the size of files, but also **improves performance (http://www.phoronix.com/scan.php?page=article&item=btrfs_compress_2635&num=1)**, in particular if using the **lzo algorithm (http://www.phoronix.com/scan.php?page=article&item=btrfs_lzo_2638&num=1)**, in some specific use cases (e.g. single thread with heavy file IO), while obviously harming performance on other cases (e.g. multithreaded and/or cpu intensive tasks with large file IO).

Compression is enabled using the `compress` mount option, which can be set to `zlib`, `lzo`, `zstd`, or `no` (for no compression). Only files created or modified after the mount option is added will be compressed.

Note: Systems using older kernels or btrfs-progs (https://www.archlinux.org/packages/?name=btrfs-progs) without `zstd` support may be unable to read or repair your filesystem if you use this option.

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 lzo (https://www.archlinux.org/packages/?name=lzo), run the following command:

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

Tip: Compression can also be enabled per-file without using the `compress` mount option; to do so apply `chattr +c` to the file. When applied to directories, it will cause new files to be automatically compressed as they come.

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

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 filesystem, or it can be mounted in its own right." **[1] (https://btrfs.wiki.kernel.org/index.php/SysadminGuide#Subvolumes)**

Each Btrfs file system has a top-level subvolume with ID 5. It can be mounted as `/` (by default), or another subvolume can be **mounted** instead. Subvolumes can be moved around in the filesystem and are rather identified by their id than their path.

See the following links for more details:

- **Btrfs Wiki SysadminGuide#Subvolumes (https://btrfs.wiki.kernel.org/index.php/SysadminGuide#Subvolumes)**
- **Btrfs Wiki Getting started#Basic Filesystem Commands (https://btrfs.wiki.kernel.org/index.php/Getting_started#Basic_Filesystem_Commands)**
- **Btrfs Wiki Trees (https://btrfs.wiki.kernel.org/index.php/Trees)**

Creating a subvolume

To create a subvolume:

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

Listing subvolumes

To see a list of current subvolumes under `path`:

```
# btrfs subvolume list -p path
```

Deleting a subvolume

To delete a subvolume:

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

Attempting to remove the directory `/path/to/subvolume` without using the above command will not delete the subvolume.

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. Thus one can easily restore a file system (or part of it) to a previous state using **#Snapshots**.

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

Note: "Most mount options apply to the **whole filesystem**, 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." **[2] (https://btrfs.wiki.kernel.org/index.php/Mount_options)** See the **Btrfs Wiki FAQ (https://btrfs.wiki.kernel.org/index.php/FAQ#Can_I_mount_subvolumes_with_different_mount_options.3F)** for which mount options can be used per subvolume.

See **Snapper#Suggested filesystem layout**, **Btrfs SysadminGuide#Managing Snapshots (https://btrfs.wiki.kernel.org/index.php/SysadminGuide#Managing_Snapshots)**, and **Btrfs SysadminGuide#Layout (https://btrfs.wiki.kernel.org/index.php/SysadminGuide#Layout)** for example file system layouts using subvolumes.

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 (https://bbs.archlinux.org/viewtopic.php?pid=1615373)**.

Changing the default subvolume with `btrfs subvolume set-default` will make the top level of the filesystem inaccessible, except by use of the `subvol=/` or `subvolid=5` mount options **[3] (https://btrfs.wiki.kernel.org/index.php/SysadminGuide)**.

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 (https://btrfs.wiki.kernel.org/index.php/Quota_support#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, unshared 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 filesystem use

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

Commit Interval

The resolution at which data are written to the filesystem 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 filesystem. This can be changed by appending the `commit` mount option in `/etc/fstab` for the btrfs partition.

```
LABEL=arch64 / btrfs defaults,noatime,compress=lzo,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 resides in `Documentation/sysctl/vm.txt`.

SSD TRIM

A Btrfs filesystem is able to free unused blocks from an SSD drive supporting the TRIM command.

More information about enabling and using TRIM can be found in **Solid State Drives#TRIM**.

Usage

Displaying used/free space

General linux userspace tools such as `df` will inaccurately report free space on a Btrfs partition. It is recommended to use `btrfs filesystem usage` to query Btrfs partitions. For example:

```
# btrfs filesystem usage /
```

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

See **[4] (https://btrfs.wiki.kernel.org/index.php/FAQ#How_much_free_space_do_I_have.3F)** for more information.

Defragmentation

Btrfs supports online defragmentation through the **mount option (https://btrfs.wiki.kernel.org/index.php/Mount_options)** `autodefrag`. 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.

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

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 (https://btrfs.wiki.kernel.org/index.php/Using_Btrfs_with_Multiple_Devices)** for more information. The Btrfs sysadmin page also **has a section (https://btrfs.wiki.kernel.org/index.php/SysadminGuide#RAID_and_data_replication)** 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 (https://btrfs.wiki.kernel.org/index.php/RAID56)** and a bug report on **linux-btrfs mailing list (https://www.mail-archive.com/linux-btrfs@vger.kernel.org/msg55161.html)** for more detailed information.

Scrub

The **Btrfs Wiki Glossary (https://btrfs.wiki.kernel.org/index.php/Glossary)** says that Btrfs scrub is "[a]n online filesystem checking tool. Reads all the data and metadata on the filesystem, and uses checksums and the duplicate copies from RAID storage to identify and repair any corrupt data."

Warning: A running scrub process will prevent the system from suspending, see **this thread (http://comments.gmane.org/gmane.comp.file-systems.btrfs/33106)** for details.

Start manually

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

```
# btrfs scrub start /
```

To check the status of a running scrub:

```
# btrfs scrub status /
```

Start with a service or timer

The **`btrfs-progs (https://www.archlinux.org/packages/?name=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) (https://jlk.fjfi.cvut.cz/arch/manpages/man/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` is that the results of the scrub will be logged in the **systemd journal**.

Balance

"A balance passes all data in the filesystem through the allocator again. It is primarily intended to rebalance the data in the filesystem 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." **[5] (https://btrfs.wiki.kernel.org/index.php/Glossary)** See **Upstream FAQ page (https://btrfs.wiki.kernel.org/index.php/FAQ#What_does_.22balance.22_do.3F)**.

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 **"filesystem full" issues (https://btrfs.wiki.kernel.org/index.php/FAQ#Help.21_Btrfs_claims_I.27m_out_of_space.2C_but_it_looks_like_I_should_have_lots_left.21)**.

```
# btrfs balance start /
# btrfs balance status /
```

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 (https://btrfs.wiki.kernel.org/index.php/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 writable version of a readonly snapshot, simply create a snapshot of it.

Note: Snapshots are not recursive. Every nested subvolume will be an empty directory inside the snapshot.

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).

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 (https://btrfs.wiki.kernel.org/index.php/Incremental_Backup)** on how to use this for incremental backups and for tools that automate the process.

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 (https://aur.archlinux.org/packages/duperemove/)AUR, bedup (https://aur.archlinux.org/packages/bedup/)AUR and *btrfs-dedup*. One may also want to merely deduplicate data on a file based level instead using e.g. rmlint (https://www.archlinux.org/packages/?name=rmlint) or jdupes (https://aur.archlinux.org/packages/jdupes/)AUR. For an overview of available features of those programs and additional information have a look at the **upstream Wiki entry (https://btrfs.wiki.kernel.org/index.php/Deduplication#Batch)**.

Furthermore Btrfs developers are working on inband (also known as synchronous or inline) deduplication, meaning deduplication done when writing new data to the filesystem. Currently it is still an experiment which is developed out-of-tree. Users willing to test the new feature should read the **appropriate kernel wiki page (https://btrfs.wiki.kernel.org/index.php/User_notes_on_dedupe)**.

Known issues

A few limitations should be known before trying.

Encryption

Btrfs has no built-in encryption support, but this **may (https://lwn.net/Articles/700487/)** come in the future. Users can encrypt the partition before running `mkfs.btrfs`. See **dm-crypt/Encrypting an entire system#Btrfs subvolumes with swap**.

Existing Btrfs file systems can use something like **EncFS** or **TrueCrypt**, though perhaps without some of Btrfs' features.

Swap file

Btrfs does not yet support **swap files**. This is due to swap files requiring a function that Btrfs intentionally does not have for possibility of file system corruption **[6] (https://btrfs.wiki.kernel.org/index.php/FAQ#Does_btrfs_support_swap_files.3F)**. Patches for swapfile support are already available **[7] (https://lkml.org/lkml/2014/12/9/718)** and may be included in an upcoming kernel release. As an alternative a swap file can be mounted on a loop device with poorer performance but will not be able to hibernate. Install the package `systemd-swap (https://www.archlinux.org/packages/?name=systemd-swap)` to automate this.

TLP

Using TLP requires special precautions in order to avoid filesystem corruption. Refer to the **according TLP section** for more information.

Tips and tricks

Partitionless Btrfs disk

Warning: Most users do not want this type of setup and instead should install Btrfs on a regular 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 filesystem** on an existing **partition** that was created using another method. There are some limitations to partitionless single disk setups:

- Cannot use different **file systems** for different **mount points**.
- Cannot use **swap area** as Btrfs does not support **swap files** and there is no place to create **swap partition**. This also limits the use of hibernation/resume, which needs a swap area to store the hibernation image.
- Cannot use **UEFI** to 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.

Install the **boot loader** like you would for a data storage device with a **Master Boot Record**. See **Syslinux#Manual install** or **GRUB/Tips and tricks#Install to partition or partitionless disk**.

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 **Conversion from**

Ext3 (https://btrfs.wiki.kernel.org/index.php/Conversion_from_Ext3) on the btrfs wiki for more information.

Boot from an install CD, then convert by doing:

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

Mount the partion 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 the GRUB menu list (see **Install from existing Linux** and **GRUB** articles). If converting a root filesystem, while still chrooted run `mkinitcpio -p linux` to regenerate the initramfs or the system will not successfully boot. If you get stuck in grub with 'unknown filesystem' try reinstalling grub with `grub-install /dev/partition` and regenerate the config as well `grub-mkconfig -o /boot/grub/grub.cfg`.

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. Notably **TLP#Btrfs** needs special care to avoid filesystem corruption but other applications may profit from certain features as well.

Checksum hardware acceleration

To verify if Btrfs checksum is hardware accelerated:

```
$ dmesg | grep crc32c
```

```
Btrfs loaded, crc32c=crc32c-intel
```

If you see `crc32c=crc32c-generic`, it is probably because your root partition is Btrfs, and you will have to compile `crc32c-intel` into the kernel to make it work. Putting `crc32c-intel` into **mkinitcpio.conf** does *not* work.

Corruption recovery

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=("/usr/bin/btrfs")
```

Regenerate the initial ramdisk using **mkinitcpio**.

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?) then 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 the **Btrfs Wiki page (https://btrfs.wiki.kernel.org/index.php/Btrfsck)** for more information.

Booting into snapshots

In order to boot into a snapshot you must specify the subvolume via a **kernel parameter** using `rootflags=subvol=/path/to/subvolume` and alter your `/etc/fstab` to point to the same subvolume using `subvol=`. Alternatively the subvolume can be specified with its id - retrievable with e.g. `btrfs subvolume list /root/path` - and `rootflags=subvolid=objectid` as kernel parameter respectively `subvolid=objectid` as mount option in `/etc/fstab`.

If using GRUB you can automatically populate your boot menu with btrfs snapshots when regenerating the configuration file with the help of **`grub-btrfs (https://aur.archlinux.org/packages/grub-btrfs/)`**$^{\text{AUR}}$ or `grub-btrfs-git (https://aur.archlinux.org/packages/grub-btrfs-git/)`AUR.

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.

Troubleshooting

See the **Btrfs Problem FAQ (https://btrfs.wiki.kernel.org/index.php/Problem_FAQ)** for general troubleshooting.

GRUB

Partition offset

The offset problem may happen when you try to embed `core.img` into a partitioned disk. It means that **it is OK (https://wiki.archlinux.org/index.php?title=Talk:Btrfs&diff=319474&oldid=292530)** 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.

Missing root

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.

BTRFS: open_ctree failed

As of November 2014 there seems to be a bug in **systemd** or **mkinitcpio** causing the following error on systems with multi-device Btrfs filesystem 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 error

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 with `mkinitcpio -p linux` (adjust the preset if needed) 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 (https://blog.samcater.com/fix-for-btrfs-open_ctree-failed-when-running-root-fs-on-raid-1-or-raid10-arch-linux/)**.

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 (https://bbs.archlinux.org/viewtopic.php?id=189845)** and **FS#42884 (https://bugs.archlinux.org/task/42884)** for further information and discussion.

btrfs check

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

The ***btrfs check (https://btrfs.wiki.kernel.org/index.php/Manpage/btrfs-check)*** command can be used to check or repair an unmounted Btrfs filesystem. However, this repair tool is still immature and not able to repair certain filesystem errors even those that do not render the filesystem unmountable.

See **Btrfsck (https://btrfs.wiki.kernel.org/index.php/Btrfsck)** for more information.

See also

- Official site
  - Btrfs Wiki (https://btrfs.wiki.kernel.org/)
- Performance related
  - Btrfs on raw disks? (http://superuser.com/questions/432188/should-i-put-my-multi-device-btrfs-filesystem-on-disk-partitions-or-raw-devices)
  - Varying leafsize and nodesize in Btrfs (https://www.spinics.net/lists/linux-btrfs/msg18652.html)
  - Btrfs support for efficient SSD operation (data blocks alignment) (http://comments.gmane.org/gmane.comp.file-systems.btrfs/15646)
  - Is Btrfs optimized for SSDs? (https://btrfs.wiki.kernel.org/index.php/FAQ#Is_Btrfs_optimized_for_SSD.3F)
  - Phoronix mount option benchmarking
    - Linux 4.9 (http://www.phoronix.com/scan.php?page=article&item=btrfs-mount-linux49)

        - Linux 3.14 (http://www.phoronix.com/scan.php?page=article&item=linux_314_btrfs)
        - Linux 3.11 (http://www.phoronix.com/scan.php?page=article&item=linux_btrfs_311&num=1)
        - Linux 3.9 (http://www.phoronix.com/scan.php?page=news_item&px=MTM0OTU)
        - Linux 3.7 (http://www.phoronix.com/scan.php?page=article&item=btrfs_linux37_mounts&num=1)
        - Linux 3.2 (http://www.phoronix.com/scan.php?page=article&item=linux_btrfs_options&num=1)
    - Lzo vs. zLib (http://blog.erdemagaoglu.com/post/4605524309/lzo-vs-snappy-vs-lzf-vs-zlib-a-comparison-of)
- **Miscellaneous**
    - Funtoo Wiki Btrfs Fun (http://www.funtoo.org/wiki/BTRFS_Fun)
    - Avi Miller presenting Btrfs (http://www.phoronix.com/scan.php?page=news_item&px=MTA0ODU) at SCALE 10x, January 2012.

- **Summary of Chris Mason's talk (http://www.phoronix.com/scan.php?page=news_item&px=MTA4Mzc)** from LFCS 2012
- **Btrfs: stop providing a bmap operation to avoid swapfile corruptions (http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commit;h=35054394c4b3cecd52577c2662c84da1f3e73525)** 2009-01-21
- **Doing Fast Incremental Backups With Btrfs Send and Receive (http://marc.merlins.org/perso/btrfs/post_2014-03-22_Btrfs-Tips_-Doing-Fast-Incremental-Backups-With-Btrfs-Send-and-Receive.html)**

Retrieved from "https://wiki.archlinux.org/index.php?title=Btrfs&oldid=501486"

- This page was last edited on 9 December 2017, at 12:04.
-