BTRFS

Pronouncation

- Butter-fs
- Better-fs
- Actually stands for B-tree file system
- Meant to replace ext4 but also include next-generation features including
    - Volume manager
    - RAID array manager
    - On-the-fly disk compression
    - Copy-on-write (CoW)
    - and more...

History

- Chris Mason is the founding developer
- Started working on btfrs in 2007 while working at Oracle
- btrfs 1.0 was accepted in mainline linux kernel in 2009
    - Was it production-ready? No
    - The following message was displayed until 2013:

Btrfs is a new filesystem with extents, writable snapshotting, support for multiple devices and many more features.

Btrfs is highly experimental, and THE DISK FORMAT IS NOT YET FINALIZED. You should say N here unless you are interested in testing Btrfs with non-critical data.

Current State of Affairs

- Good news
    - Perfectly cromulent single-disk ext-4 replacement
- Bad news
    - ZFS replacement for a more complex stack built on discrete RAID and volume management, and simple file system? Not so much!

Features

- Copy on write (CoW)
- Compression
- Subvolumes
- Snapshots
- Quota
- SSD trim
- Multi-device file system (RAID array, JBOD, etc.)

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."
- RAID 0, 1, and 10 are not perfect either, use at your own risk.
- Single-device use of btrfs seems to be the safest way to use it at the moment.

Copy on Write (CoW)

- Used for all files all the time unless turned off by
  1. mounting the file system via *nodatacow* option
     - only affects newly created files and CoW would still apply to existing files
     - also disables compression
     - also disables checksums and prevents detection of corrupted *nodatacow* files
  2. `chattr +C /dir/file` command
- Turning off CoW is recommended for heavily updated-in-place files such as VM images and database stores
- Create light-weight copies in btrfs by using `cp --reflink source destination` syntax

Compression

- Transparent and automatic compression
- Reduces the size of files as well as significantly increases the lifespan of flash-based media by reducing write amplification
- May improve or worsen performance based on the use case scenario
- `compress=alg` in mount option needs to be used, where alg is either *zlib, lzo, zstd* or *no*
- Faster algorithms like *zstd* or *lzo* seems to provide better performance
- `compsize -x` is used see the list of files with compression types and effective compression ratios
  - `du` is not reliable because it is blind to light-weight copies created by `cp --reflink`
  - `-x` option above keeps `compsize` in a single file system

Subvolumes

- An independently mountable POSIX filetree and not a block device
- 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
- Layout may be flat or nested
- Each one has advantages and disadvantages

```
toplevel         (volume root directory, not to be mounted by default)
  +-- root       (subvolume root directory, to be mounted at /)
  +-- home       (subvolume root directory, to be mounted at /home)
  +-- var        (directory)
  |   \-- www    (subvolume root directory, to be mounted at /var/www)
  \-- postgres   (subvolume root directory, to be mounted at /var/lib/postgresql)
```

Flat

```
toplevel                  (volume root directory, to be mounted at /)
+-- home                  (subvolume root directory)
+-- var                   (subvolume root directory)
    +-- www               (subvolume root directory)
    +-- lib               (directory)
         \-- postgresql   (subvolume root directory)
```

Nested

```
toplevel                          (volume root directory, not mounted)
  \-- root                        (subvolume root directory, to be mounted at /)
      +-- home                    (subvolume root directory)
      +-- var                     (subvolume root directory)
          +-- www                 (subvolume root directory)
          +-- lib                 (directory)
               \-- postgresql     (subvolume root directory)
```

Nested-better

More About Subvolume Layouts

Flat

- Management of snapshots (especially rolling them) may be considered easier as the effective layout is more directly visible
- All subvolumes need to be mounted manually (e.g. via fstab) to their desired locations

```
LABEL=the-btrfs-fs-device   /                    btrfs subvol=/root,defaults,noatime  0  0
LABEL=the-btrfs-fs-device   /home                btrfs subvol=/home,defaults,noatime  0  0
LABEL=the-btrfs-fs-device   /var/www             btrfs subvol=/var/www,noatime        0  0
LABEL=the-btrfs-fs-device   /var/lib/postgresql  btrfs subvol=/postgres,noatime       0  0
```

- Each of these subvolumes/mountpoints can be mounted with some options being different
- Everything in the volume that's not beneath a subvolume that has been mounted, is not accessible or even visible (beneficial for security, especially when used with snapshots)

Nested

- Management of snapshots (especially rolling them) may be considered more difficult as the effective layout isn't directly visible
- Subvolumes don't need to be mounted manually (or via fstab) to their desired locations, they "appear automatically" at their respective locations
- For each of these subvolumes the mount options of their mountpoint applies
- Everything is visible

When to Make Subvolumes

- Split of areas which are complete and/or consistent in themselves
  - `/var/www`
  - `/var/lib/postgresql`
  - `/home`
- Split of areas which need special properties / mount options
- Nested subvolumes are not going to be part of snapshots created from their parent subvolume, i.e. excluding parts of system from being snapshot

Snapshots

- Simply a subvolume that shares its data (and metadata) with some other subvolume, using btrfs's COW capabilities
- A writable snapshot has no difference in status as compared to the original subvolume
- `btrfs [-r] snapshot source target` r option would make the snapshot read-only
- read-only snapshots cannot be moved
- To roll back to a snapshot, unmount the modified original subvolume
  - `mv original-subvolume temporary-location`
  - `mv snapshot original-subvolume` *--exception: read-only snapshots cannot be moved*
  - `mount original-subvolume`
  - optionally `rm -rf temporary-location`
- OR
  - `btrfs subvolume delete original-subvolume`
  - `btrfs subvolume snapshot snapshot original-subvolume`
  - `mount original-subvolume`

Snapshots

- Snapshots of snapshots are possible since snapshots are subvolumes
- **Beware:** Snapshots of volumes that are visible to any user (e.g. when they are created in a nested layout) will remain visible to any user
- Snapshots has to be on the same device where the subvolume is
- Snapshots can be sent to other btrfs devices
    - `btrfs subvolume snapshot -r / snapshot`
    - `btrfs send snapshot | btrfs receive /mnt/externaldrive/snapshots/backup20211011` *initial snapshot*
    - work on projects whole day and start a new day
    - `btrfs subvolume snapshot -r / new-snapshot`
    - `btrfs send -p snapshot new-snapshot | btrfs receive /mnt/externaldrive/snapshots/backup20211012` *incremental snapshot*
- **Snapshots are NOT backups**

Quota

- Must be enabled before any subvolume is added
- `btrfs quota enable <path>`
- If quotas weren't enabled, they can be enabled and then a qgroup (quota group) is created for each subvolume using the subvolume ID and rescan them
- `btrfs subvolume list <path> | cut -d' ' -f2 | xargs -I{} -n1 btrfs qgroup create 0/{} <path>`
- `btrfs quota rescan <path>`
- `btrfs qgroup limit 100G <path>/<subvolume>`
- `btrfs qgroup show <path>`

SSD Trim

- A Btrfs filesystem is able to free unused blocks from an SSD drive supporting the TRIM command. Starting with kernel version 5.6 there is asynchronous discard support, enabled with mount option `discard=async`
- My fstab file as an example:
    - `UUID=e88e98e4-b123-4379-a81a-ca73c224b114 / btrfs rw,noatime,compress=zstd:3,ssd,discard=async,space_cache,subvolid=256,subvol=/@ 0 0`
    - `UUID=9682-61B5 /boot vfat rw,relatime,fmask=0022,dmask=0022,codepage=437,iocharset=ascii,shortname=mixed,utf8,errors=rem ro 0 2`
    - `UUID=e88e98e4-b123-4379-a81a-ca73c224b114 /home btrfs rw,noatime,compress=zstd:3,ssd,discard=async,space_cache,subvolid=257,subvol=/@home 0 0`
    - `UUID=e88e98e4-b123-4379-a81a-ca73c224b114 /var/cache btrfs rw,noatime,compress=zstd:3,ssd,discard=async,space_cache,subvolid=258,subvol=/@cache 0 0`
    - `/dev/mapper/swap none swap defaults 0 0`

Other usage of btrfs

- `# btrfs filesystem usage <path>` preferred to `df <path>` since the latter may be inaccurate on a btrfs partition
- `$ btrfs filesystem df <path>`
- `# btrfs scrub start <path>`
  - file system checking tool. Reads all the data and metadata on the file system and uses checksums and the duplicate copies from storage to identify and repair any corrupt data.
  - `# btrfs scrub status <path>`
  - can be started with a systemd timer and logged in the systemd journal
  - may need to limit the rate of scrubbing by `IOReadBandwithMax` option in NVMe drives in laptops to prevent overheating of the drive
- `# btrfs balance start --bg <path>`
  - Intended to rebalance the data in the file system across the devices when a device is added or removed
  - on a single-device file system, may be useful for (temporarily) reducing the amount of allocated but unused (meta)data chunks - `# btrfs balance status <path>`

Booting into Snapshots

- In Arch linux grub-btrfs package would automatically populate the boot menu with btrfs snapshots
- ? alternative in Debian ecosystem

Snapshot Creation Options

- Basic shell scripting
- GUI options
  - Timeshift
  - Snapper (with or without GUI)
  - Probably some others

My Primitive Setup

- `00 1 * * * btrfs subvolume snapshot -r / /snapshots/$(date +\%Y\%m\%d)_system >> /root/cronlogs/btrfsnapshots_system.log`
- `01 1 * * * btrfs subvolume snapshot -r /home /snapshots/$(date +\%Y\%m\%d)_home >> /root/cronlogs/btrfsnapshots_home.log`
- `02 1 * * * btrfs subvolume delete /snapshots/$(date -d "7 days ago" +\%Y\%m\%d)_system >> /root/cronlogs/btrfsnapshots_system.log`
- `03 1 * * * btrfs subvolume delete /snapshots/$(date -d "7 days ago" +\%Y\%m\%d)_home >> /root/cronlogs/btrfsnapshots_home.log`
- `04 1 * * * rsync -aAXvz --delete /snapshots/$(date +\%Y\%m\%d)_system rsync://nuc/dockerserver_system`
- `34 1 * * * rsync -aAXvz --delete /snapshots/$(date +\%Y\%m\%d)_home rsync://nuc/dockerserver_home`
- `30 1 * * 6 btrfs scrub start / >> /root/cronlogs/btrfscrub_system.log`
- `00 2 * * 6 btrfs scrub start /home >> /root/cronlogs/btrfscrub_home.log`
- `30 2 * * 6 btrfs scrub status / >> /root/cronlogs/btrfscrub_system.log`
- `32 2 * * 6 btrfs scrub status /home >> /root/cronlogs/btrfscrub_home.log`

Timeshift Demo Here

References

- BTRFS: Linux's Half-finished Filesystem
- Arch Linux Wiki - BTRFS
- BTRFS: SysAdmin Guide