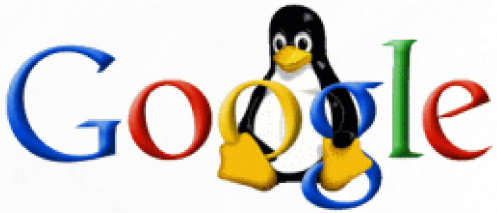
Linux.conf.au 2015



Why you should consider using Btrfs, real COW snapshots and file level incremental server OS upgrades.

http://marc.merlins.org/linux/talks/2015/Btrfs-LCA2015/

Marc MERLIN
marc_soft@merlins.org

So, why am I talking about Btrfs?

- I've been using linux and its different filesystems since 1993
- I've have been using ext2/ext3/ext4 for 20 years.
- But I worked at Network Appliance in 1997, and I've grown hooked to snapshots.
- LVM snapshots never worked well on linux, and have great performance problems (I've seen I/O speed go down to 2MB/s).
- I also like using partitions to separate my filesystems, but LVM partitions are not ideal and add the overhead of using LVM.
- I wanted this badly enough that I switched my laptop to btrfs 3 years ago, and more machines since then.

Why Should You Consider Btrfs?

- Copy On Write (COW) allows for atomic transactions without a separate journal
- Snapshots are built in the filesystem and cost little performance, especially compared to LVM.
- cp –reflink=always copies within and between subvolumes without duplicating data (ZFS doesn't support this).
- Metadata is redundant and checksummed. Data is checksummed too (ext4 only has experimental metadata checksums http://goo.gl/tmyAS3).
- If you use docker, Btrfs is your best underlying filesystem.
- Diffs after upgrades and rollbacks like Suse offers

Why Should You Consider Btrfs? (2)

- Raid 0, 1, 5, and 6 are also built in the filesystem
- You won't need multiple partitions or LVM Logical Volumes anymore, so you'll never have to resize a partition.
- File compression is also built in (lzo or zlib)
- Online background filesystem scrub (partial fsck)
- Block level filesystem diff backups (btrfs send/receive instead of slow rsync)
- Btrfs-convert can convert ext3 to btrfs while keeping your data, but it can fail, so backups are recommended and a normal copy is better.

But why not use ZFS?

- ZFS is more mature than Btrfs.
- ZFS offers almost all the features that Btrfs offers, and a few more.
- But it was licensed by SUN to be incompatible with the linux kernel license (you can put both together yourself, but you cannot redistribute a kernel with both).
- ZFS is fairly memory hungry, it's recommended to have 16GB of RAM and give 8GB or more to ZFS (it wasn't designed to use the linux memory filesystem, so it uses its own memory that can't be shared with the rest of linux).

ZFS licensing

- Oracle bought Sun which had licensing right and patents to the original ZFS code
- Therefore Oracle could relicense the original code from CDDL to GPL-2 and replace/get rights to the patches submitted to open Solaris.
- It would seem a like a lot less work than writing a new filesystem from scratch

Oracle's position on btrfs and ZFS

- Oracle's official position is

  Oracle began btrfs development years before the Sun acquisition and we currently have no interest in an “official” port of ZFS from Solaris into Linux which would require a relicensing effort. We’d rather focus on improving btrfs which was developed from scratch as a mainline kernel (GPLv2) next-generation filesystem. Oracle has several developers dedicated to ongoing btrfs development, and we support btrfs on Oracle Linux for production purposes.

- http://goo.gl/3JVHQe says:

  "According to Coekaerts, porting ZFS to Linux involves a non-optimal approach that is not native. As such, there is likely not a need to attempt to bring ZFS to Linux since Btrfs is now around to fit the bill. "

Were patents a problem with ZFS?

- Netapp sued Sun saying ZFS infringed 7 WAFL patents http://goo.gl/PlzByI
- That said Sun attacked Netapp first http://goo.gl/L5gyX5

Were patents a problem with ZFS?

- Apple was going to use ZFS and later dropped the idea http://goo.gl/C5b8M5
- Around the same time Oracle starts writing Btrfs
- Chris Mason hints around 2008 that Btrfs has newer design elements than ZFS (and WAFL), and isn't known to violate any patents http://goo.gl/Rfzi5D http://goo.gl/qntsNq
- Netapp and Oracle agreed to end the suit privately http://en.swpat.org/wiki/NetApp's_filesystem_patents
- Oracle may have stopped further work on ZFS as a result. Or it could be another reason entirely...

Be wary of ZFS for production use

- You can use ZFS and patch it against kernels on your own, but the code needs to be maintained out of tree and patched forever.
- Vmware workstation mostly died and was replaced by virtualbox because the vmware drivers never worked with newer kernels, and it stopped working when you upgraded.
- Due to the CDDL being incompatible with GPLv2, a linux vendor or hardware vendor will never be able to ship a linux distribution or hardware device using ZFS
- As a result, you shouldn't plan on using ZFS for any product that you might ever want to ship one day.
- It is only safe to use ZFS for internal use on something that will never ship to others. Yet, many things that weren't meant to ship become products one day (google search appliance for instance)

Btrfs: Wait, is it stable/safe yet?

- Oracle and Suse support Btrfs in their commercial distribution
- Basic Btrfs is mostly stable: Snapshots, raid 0, raid 1.
- It typically doesn't just corrupt itself in recent kernels (>3.1x), but it could. Always have backups.
- 3.14.x works ok, avoid 3.15 to 3.16.1, stay one stable kernel behind.
- It changes quickly though, so use recent kernels if you can, but consider staying a kernel or two behind for stability.
- It can get out of balance and require manual re-balancing
- Auto defrag has performance problems with journal and virtual disk image files
- Btrfs send/receive mostly works reliably as of 3.14.x
- Raid 5 and 6 are still experimental as of 3.18

What's not there yet?

- Fsck.btrfs aka btrfsck or btrfs check –repair is still a bit incomplete but much better in later btrfs-tools.
- But thankfully it's mostly not needed and there are other recovery options.
- File encryption is not supported yet (can be done via dm-crypt)
- Dedup is experimental via a userland tool, and online real time dedup hasn't been written yet
- More testing and polish, as well as brave users like you :)

Who contributes to Btrfs?

Incomplete list: https://btrfs.wiki.kernel.org/index.php/Contributors

- Facebook
- Fujitsu
- Fusion-IO
- Intel
- Linux Foundation
- Netgear
- Oracle
- Red Hat
- Strato
- Suse / Novell
- <Your company name here> :)

Who uses Btrfs in production?

- https://btrfs.wiki.kernel.org/index.php/Production_Users
- http://www.phoronix.com/scan.php?page=news_item&px=MTY0NDk

It looks like in 2014 might finally be the year we see more real-world deployments of Btrfs in place of EXT4 or XFS. This year openSUSE 13.2 is switching to Btrfs by default for new installations as the first tier-one Linux distribution relying upon the next-generation open-source file-system.

- http://lwn.net/Articles/577728/ (Jon Corbet's predictions)

Btrfs will start seeing wider production use in 2014, finally, though users will learn to pick and choose between the various available features.

Ok, great, so how do I use BTRFS?

We will look at best practices, namely:

- When things go wrong: filesystem recovery
- Btrfs scrub/log parsing
- Dmcrypt, Raid, and Compression
- Pool directory
- Historical Snapshots and backups
- What to do with out of space problems (real and not real)
- Btrfs send/receive
- Tips and tricks: cp –reflink, defragmenting, nocow with chattr
- How btrfs raid 1 works. Raid 5/6

Filesystem Recovery

https://btrfs.wiki.kernel.org/index.php/Btrfsck explains:

- btrfs scrub to detect issues on live filesystems (but it is not a full online fsck).
- look at btrfs detected errors in syslog
- mount -o ro,recovery to mount a filesystem with issues
- btrfs-zero-log might help in specific cases.
- btrfs restore will help you copy data off a broken btrfs filesystem. https://btrfs.wiki.kernel.org/index.php/Restore
- btrfs check --repair, aka btrfsck is your last option if the ones above have not worked.

Plan for recovery before you need it

- A bug could cause your root filesystem not to be mountable
- Try passing mountopts=recovery,ro to grub/lilo
- If you use initrd, make sure it includes /sbin/btrfs and dependencies
- Also make sure initrd includes /sbin/btrfs-zero-log (debian does)
- If you use btrfs for root, have a 2nd root partition to boot from
- Test that your 2nd root partition works
- Generally on a laptop, consider having 2 drives to boot from, especially if you use an SSD (SSDs do die unexpectedly)

```
legolas:~# cat /etc/initramfs-tools/hooks/btrfs-recover
#!/bin/sh

. /usr/share/initramfs-tools/hook-functions

[ -x "/sbin/btrfs-zero-log" ] && copy_exec /sbin/btrfs-zero-log /sbin
[ -x "/sbin/btrfs" ] && copy_exec /sbin/btrfs /sbin
```

Btrfs scrub

- **Run scrub nightly or weekly on all btrfs filesystems**
- Even on a non RAID filesystem, btrfs usually has two copies of metadata which are both checksummed (-m dup for mkfs.btrfs).
- Data blocks are not duplicated unless you have RAID1 or higher, but they are checksummed
- Scrub will therefore know if your metadata is corrupted and typically correct it on its own
- It can also tell you if your data blocks got corrupted, auto fix them if RAID allows, or report them to you in syslog otherwise.
- Knowing that your data is corrupted is valuable, since you know you can restore from backup (many filesystems do not give you this information).
- More repair strategies and watching btrfs-scrub logs on my blog: http://goo.gl/knHpM6

Btrfs scrub issue

How to fix a scrub that stopped half way:

```
gargamel:~# btrfs scrub start -d /dev/mapper/dshelf1
ERROR: scrub is already running.
To cancel use 'btrfs scrub cancel /dev/mapper/dshelf1'.
gargamel:~# btrfs scrub status  /dev/mapper/dshelf1
scrub status for 6358304a-2234-4243-b02d-4944c9af47d7
        scrub started at Tue Apr  8 08:36:18 2014, running for 46347 seconds
        total bytes scrubbed: 5.70TiB with 0 errors
gargamel:~# btrfs scrub cancel  /dev/mapper/dshelf1
ERROR: scrub cancel failed on /dev/mapper/dshelf1: not running
gargamel:~# perl -pi -e 's/finished:0/finished:1/' /var/lib/btrfs/*  << FIX
gargamel:~# btrfs scrub status  /dev/mapper/dshelf1
scrub status for 6358304a-2234-4243-b02d-4944c9af47d7
        scrub started at Tue Apr  8 08:36:18 2014 and finished after 46347
seconds
        total bytes scrubbed: 5.70TiB with 0 errors
gargamel:~# btrfs scrub start -d /dev/mapper/dshelf1
scrub started on /dev/mapper/dshelf1, fsid 6358304a-2234-4243-b02d-
4944c9af47d7 (pid=24196)
```

Dmcrypt, dm-raid, and btrfs, which one comes first?

- If you use software raid, it's less work to setup decryption of a single md raid5 device than decrypting X underlying devices first.
- With dmcrypt on top of dm-raid5, you only need to crypt n-1 disks of data.
- Recent kernels thread dmcrypt well enough across CPUs
- So, you have btrfs on top of dmcrypt on top of dm-raid.
- But if you use btrfs built in raid 0, 1, 5, 6, you have to setup a decryption of each device at boot.

Multi-device dmcrypt and btrfs

- Mounting btrfs from dmcrypted devices is more work
- First you need to decrypt all devices
- Run `btrfs scan` and then you can run btrfs mount
- You either use `mount LABEL=` or `mount /dev/mapper/cryptdev1` (give any device and the others get auto-detected):

  mount -v -t btrfs -o compress=zlib,noatime LABEL=$LABEL /mnt/btrfs_pool
  mount -v -t btrfs -o compress=lzo,noatime /dev/mapper/cryptdev1 /mnt/btrfs_pool
- You can use my script to automate this: start-btrfs-dmcrypt available at http://goo.gl/0FI94W

Btrfs pool, subvolumes

- With btrfs, you don't need to create partitions anymore
- You typically create a storage pool
- In which you create subvolumes
- Subvolumes can get quotas and get snapshotted
- Think of subvolumes as resizeable partitions that can share data blocks

  mount -o compress=lzo,noatime LABEL=btrfs_pool /mnt/btrfs_pool
  btrfs subvolume create /mnt/btrfs_pool/root
  btrfs subvolume create /mnt/btrfs_pool/usr
- Mount with:

  boot: root=/dev/sda1 rootflags=subvol=root
  mount -o subvol=usr /dev/sda1 /usr
  or mount -o bind /mnt/btrfs_pool/usr /usr

Btrfs subvolume snapshots

- The most appealing feature in btrfs is snapshots.
- Snapshots work on subvolumes
- You can snapshot a snapshot
- You can make a read only snapshot, or a read-write snapshot of a read-only snapshot

```
legolas:/mnt/btrfs_pool1# btrfs subvolume create test
Create subvolume './test'
legolas:/mnt/btrfs_pool1# touch test/foo
legolas:/mnt/btrfs_pool1# btrfs subvolume snapshot test test-snap
Create a snapshot of 'test' in './test-snap'
legolas:/mnt/btrfs_pool1# touch test-snap/bar
legolas:/mnt/btrfs_pool1# rm test/foo
legolas:/mnt/btrfs_pool1# ls test-snap/
bar  foo
```

Btrfs subvolume snapshots

```
legolas:/mnt/btrfs_pool1# btrfs subvolume show test
/mnt/btrfs_pool1/test
Name:             test
uuid:             2cd10c93-31a3-ed42-bac9-84f6f86215b3
Parent uuid:          -
Creation time:    2014-05-03 18:41:15
Object ID:        25576
Generation (Gen):406949
Gen at creation: 406947
Snapshot(s):
                    Test-snap <<<<<<<<<<<<<

legolas:/mnt/btrfs_pool1# btrfs subvolume show test-snap/
/mnt/btrfs_pool1/test-snap
Name:             test-snap
uuid:             9529d4e3-1c64-ec4e-89a0-7fa010a115fe
Parent uuid:      2cd10c93-31a3-ed42-bac9-84f6f86215b3 <<<<<<<<<<<<<
Creation time:    2014-05-03 18:41:29
Object ID:        25577
Generation (Gen):406949
Gen at creation: 406948
Snapshot(s):
```

Snapshots are not real backups

This is probably the wrong backup strategy :)

Historical snapshots to go back in time

- You can use the script I wrote that will look at any btrfs pool, find all the subvolumes and snapshot them (optionally) by hour/day/week.
- See my blog post at http://goo.gl/On7CZR

```
drwxr-xr-x 1 root root  370 Feb 24 10:38 root
drwxr-xr-x 1 root root  370 Feb 24 10:38 root_daily_20140316_00:05:01
drwxr-xr-x 1 root root  370 Feb 24 10:38 root_daily_20140318_00:05:01
drwxr-xr-x 1 root root  370 Feb 24 10:38 root_daily_20140319_00:05:01
drwxr-xr-x 1 root root  370 Feb 24 10:38 root_daily_20140320_00:05:00
drwxr-xr-x 1 root root  370 Feb 24 10:38 root_hourly_20140316_22:33:00
drwxr-xr-x 1 root root  370 Feb 24 10:38 root_hourly_20140318_00:05:01
drwxr-xr-x 1 root root  370 Feb 24 10:38 root_hourly_20140319_00:05:01
drwxr-xr-x 1 root root  370 Feb 24 10:38 root_hourly_20140320_00:05:00
drwxr-xr-x 1 root root  336 Feb 19 21:40 root_weekly_20140223_00:06:01
drwxr-xr-x 1 root root  370 Feb 24 10:38 root_weekly_20140302_00:06:01
drwxr-xr-x 1 root root  370 Feb 24 10:38 root_weekly_20140309_00:06:01
drwxr-xr-x 1 root root  370 Feb 24 10:38 root_weekly_20140316_00:06:01
```

Atime, relatime vs snapshots

- Atime forces the system to write to each inode every time you update a file
- Relatime (now the default with recent kernels) does this more intelligently and less often, but at least once a day, that's too much still.
- Unfortunately if you use snapshots, this creates a lot of writes and differing data each time you access files and cause an atime update
- As a result, if you use snapshots, noatime is strongly recommended.

Help, I really ran out of space

```
gandalfthegreat:~# btrfs fi show
Label: 'btrfs_pool1'  uuid: 873d526c-e911-4234-af1b-239889cd143d
        Total devices 1 FS bytes used 214.44GB
        devid    1 size 231.02GB used 231.02GB path /dev/dm-0
```

- Btrfs fi show will tell you if your device is really full.
- Here the device is full (needs a rebalance) and the FS almost full.
- First thing to do if you have historical snapshots is to delete the oldest ones
- If you just copied 100G without –reflink and deleted the copy, you may have to delete all historical snapshots
- **If you delete snapshots, it may take minutes for btrfs to do garbage collection and free space to come back.**
- Unfortunately it's currently difficult to know how much space each snapshot takes (since the space is shared and deleting a single snapshot may not reclaim it)

Help, btrfs says I ran out of space, but I didn't.

- Currently btrfs has issues where it needs to have its chunks rewritten to rebalance space (btrfs filesystem df /mnt and btrfs fi show both agree there is free space)
- Counting free space is tricky, see https://btrfs.wiki.kernel.org/index.php/FAQ#Raw_disk_usage
- More generally I've written a page explaining how to deal with filesystems are full (but usually aren't really): http://goo.gl/NU42P0
- As of 3.18+ btrfs is doing more work to auto rebalance data/metadata chunks for you.

Btrfs built in compression

- mount -o compress=lzo is fast and best for ssd
- mount -o compress=zlib compresses better but is slower.
- Compression is a mount option that affects files written after the mount
- If you change your mind after the fact, you can use btrfs balance to rewrite data blocks which recompresses them in the process.

Defragmentation and NOCOW

- If you have a virtual disk image, or a database file that gets written randomly in the middle, Copy On Write is going to cause many fragments since each write is a new fragment.
- My virtualbox image grew 100,000 fragments quickly.
- You can turn off COW for specific files and directories with `chattr +C /path` (new files will inherit this).
- `btrfs filesystem defragment vbox.vdi` could take hours
- `cp --reflink=never vbox.vdi vbox.vdi.new; rm vbox.vdi` is much faster
- btrfs filesystem defragment can be used to recompress files while defragmenting, but can be very slow if you have snapshots.

Block deduplication and cp --reflink

- After the fact block deduplication is currently experimental: https://github.com/g2p/bedup
- Inline deduplication is not ready yet, but still planned.
- cp –reflink /mnt/src /mnt/dest duplicates a file while reusing the same data blocks, but allows you to modify dest (can't do this with hardlinks)
- cp –reflink /mnt/btrfs_pool/subvol1/src /mnt/btrfs_pool/subvol2/dest lets you copy/move a file across subvolumes without duplicating data blocks.

Btrfs send/receive

- This is a killer app for btrfs: rsync is inefficient and slow when you have many files to copy.
- This is where btrfs send/receive come in.
- `btrfs send vol | ssh host "btrfs receive /mnt/pool"`
- Later copies only send a diff between snapshots, this diff is computed instantly when rsync could take an hour or more to generate a list of diffs:

  ```
  btrfs send -p vol_new_ro vol_old_ro | btrfs receive /mnt/pool
  ```
- It's a bit hard to setup, so I wrote a script you can use: http://goo.gl/OLkbjT

Finale: btrfs send/receive to replicate server images

- Last year I gave a talk on how google does live server upgrades by doing file level updates http://goo.gl/CxedK
- We use a custom rsync like system that was tricky to write and generates a lot of I/O
- Instead, you can use btrfs send/receive to replicate a golden image in a subvolume to all your servers
- Local changes are symlinked to a local FS or bind mounted on a file by file basis.
- Once the new subvolume has the new image, you can reboot to it and/or use kexec for a faster reboot cycle.
- btrfs-diff shows most files changed, use this to sync changes with cp –reflink instead of reboot http://goo.gl/fkLxAu
- Improve btrfs send/receive to only show files changed.

Backing up my laptop SSD to internal HD hourly

- SSDs can die suddenly with no warning (3 times for me)
- Back them up at least daily, or even hourly
- Rsync backups are slow
- Perfect use case for btrfs send/receive for quick incremental backups
- Destination backup is read only snapshot, but my script makes a 2nd R/W snapshot with a symlink pointing to it
- That way if my SSD entirely dies, I can still boot from my backup HD and recovery instantly (but OMG, hard drives are slow).
- Script that does all the work for you is here: http://goo.gl/OLkbjT

Backup your laptop on itself and boot the backups

- I then have historical backups on both my SSD and HD, and I can boot from my HD if my SSD fails while I'm on a trip (happened twice already with Crucial and OCZ) or if my btrfs filesystem crashes and can't boot.
- Make sure you can boot the 2nd drive whether it shows up as the first drive or second drive in the bios.
- Another option for your laptop if you don't need btrfs snapshots for your root filesystem, consider putting that on ext4 or having 2 copies you can boot from like I do (btrfs failures can still lead to a root filesystem that can't mount, making your computer sad).
- You can finally push backups of your laptop from hotel wireless since they're smaller and faster with btrfs send

Backup your backups, and historical backups

Historical backups with btrfs: snapshots + rsync

a) Create a subvolume: backup-$DATE
b) Rsync/copy the data to it
c) snapshot of backup-$DATE to backup-$NEWDATE
d) Rsync filesystem to backup-$NEWDATE
e) Repeat

This is easy to setup

- A bit of work to do after the fact with existing backups
- You cannot hardlink files between snapshots to save space (but you can use cp –reflink).
- A bit complicated to copy all these snapshots to a secondary server while keeping the snapshot relationship.

Historical backups with btrfs: cp -a --link + rsync

a) Create a directory backup-$DATE
b) Rsync/copy the data to it
c) cp -a --link backup-$DATE backup-$NEWDATE
d) Rsync filesystem to backup-$NEWDATE
e) Repeat

- It's not as efficient as cp –reflink but you can see the link relationship between files and keep that when copying to another filesystem
- Rsync on top of a copied file breaks the hardlink.
- Older versions of btrfs only allow a limited amount of hardlinks to a single inode.

Historical backups with btrfs: cp --reflink + rsync

a) Create a directory backup-$DATE
b) Rsync/copy the data to it
c) cp -a --reflink backup-$DATE backup-$NEWDATE
d) Rsync filesystem to backup-$NEWDATE
e) Repeat

This may work better than snapshots because:

- You can use hardlinks between backup directories to dedup identical files: http://code.google.com/p/hardlinkpy/
- Rsync on top of a copied file only modifies changed blocks
- However, unix tools can't keep track of this relationship
- Btrfs send/receive is the only way to keep this.

Btrfs mixed type filesystems

- mkfs.btrfs **-m raid0 -d raid0** /dev/sda1 /dev/sdb1
  => non redundant faster filesystem
- mkfs.btrfs **-m raid1 -d raid0** /dev/sda1 /dev/sdb1
  => metadata is redundant but data is still stripped and non recoverable is a drive is lost (files bigger than 16MB)
- mkfs.btrfs **-m raid1 -d single** /dev/sda1 /dev/sdb1
  => data for each file will usually be on a single drive and recoverable (up to 1GB).
- mkfs.btrfs **-m raid1 -d raid1** /dev/sda1 /dev/sdb1
  => data is fully redundant and stored on 2 drives
- mkfs.btrfs **-m raid1 -d raid5** /dev/sda1 /dev/sdb1 /dev/sdc1
  => medata is only stored on 2 drives, so not more redundant and slower than -m raid5 -d raid5.

Raid 1 vs Raid 5 or 6

- Btrfs lets you convert from a single drive to Raid 1, or Raid 5/6 on the fly (you need to rebalance after the change)
- Raid 1 only copies each piece of data twice, regardless of how many drives you have
- Raid 5, 6 are experimental still. Recovery doesn't work in some cases (code missing), and scrub cannot deal with rebuilding bad blocks yet (as of 3.16).
- However, you can shrink (remove a drive) while running: this causes a rebalance to remove data from the drive.
- Adding a drive is instant and you can ask btrfs to rebalance existing files on all drives (this takes much longer).
- I wrote a lot more about raid5 status here: http://goo.gl/OLkbjT

Now is time for you to evaluate Btrfs.

- If you pick the right btrfs release/kernel (you can let Oracle or Suse do this for you), you can look at Btrfs for production use.
- Btrfs send/receive is mostly production ready and you can look at it today.
- Raid 5/6 still needs work, but is so much faster that it is worth your time to contribute code to it.
- Bedup (block dedup) also needs contributors, please help if it's interesting to you.
- In a nutshell, while Btrfs is still experimental, it is usable for its core features, even if backups are recommended.
- 2015 is the year for you to evaluate it.

Questions?
Want a job at Google?

Talk slides for download:
http://marc.merlins.org/linux/talks/2015/Btrfs-LCA2015/

Marc MERLIN
marc_soft@merlins.org