SYSADMIN

System administration technologies brought to you from the coalface of Linux.

Jonathan Roberts dropped out of an MA in Theology to work with Linux. A Fedora advocate and systems administrator, we hear his calming tones whenever we're stuck with something hard.

Among developers, 'test driven development' has become trendy once more. It's certainly not a new idea, as similar practices are described in *The Mythical Man Month*, which was originally published in 1975, but it is as popular now as it has ever been.

The idea is simple. Developers write automated tests to check that functions and features they've implemented work. Usually, these tests take the form of simple functions that call the code to be tested, and compare the output to an expected value, using an assert statement or similar. If the expected and actual value match, the test passes; if they're different, the test fails.

In TDD, this is taken one step further, and advocates argue that the developer should first write a failing test for the function they're about to implement, and then they can keep working until it passes.

What's this got to do with system administration? Well, I would argue that operations teams should take a similar approach when building out the infrastructure for a new product.

Tests can take the form of checks in monitoring software such as Check_MK or Nagios. Ensure that, after your provisioning servers, your monitoring server is the first thing you install. Then, for each subsequent server to be installed, first add it and all necessary checks (process checks for Apache, MySQL server status checks etc) to your monitoring software.

Then you can begin building it. At first, all the checks will be red. But as you boot it, and then run your configuration management recipes, you'll see check after check turn green. If any stay red by the time Puppet etc has finished, you know you need to tweak your recipes.

Btrfs

The filesystem that's better (or butter) in so many ways.

In our third and final look at new technologies making their way to Linux, we're going to explore Btrfs (which in my head I'm pronouncing butter-eff-ess). Btrfs is a new copy-on-write filesystem for Linux, which aims to deliver advanced features such as volume management, snapshots, checksums and send/receive of subvolumes.

If you're not already a filesystem expert, many of those terms might sound alien to you, but continue reading and we'll do our best to explain what these features do, why you want them, and when you'll get them.

Stability

Let's start with that final question, as any one who's paid even a little attention to news about Btrfs has heard horror stories about it destroying data and may well think it's a long way from production.

As it stands now, OpenSUSE plans to be the first major distribution to use it by default in its November 13.2 release, indicating that they believe it's stable enough for daily use. Facebook, too, which has recently hired many Btrfs developers, has announced plans to begin using Btrfs in its production web tier, where it can test performance and stability on real, albeit easily recoverable, workloads.

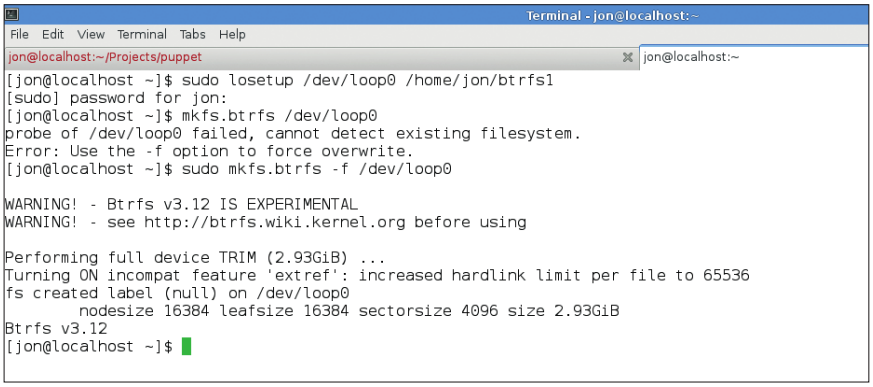
This anecdotal support from distributions and large production environments, along with the official wiki claiming that the on-disk format is now stable, suggests that if you want to start testing, now is the time to do so. It might not be making its way in to the next round of enterprise distribution releases as default, but it will be there for those users who really need it.

So, if you want to try some of the features we'll describe in the rest of this article, make sure you have automatic and tested backups running. Do that, and even if the 'experimental' status of Btrfs does lead to data loss, you won't be left cursing.

Getting started

With that word on stability out of the way, let's get to work and create a new Btrfs filesystem. Once we have the filesystem in place, we'll start working through some of its core features, showing what they do, why they're great and how to use them.

For our simple experiments, we're going to use some plain files mounted as loop devices. So, to start, first create an empty 3GB file, use **losetup** to create a new loop



Creating a btrfs filesystem is just like any other – easy. In this tutorial, we've used loopback mounts to experiment, but this would all work just as well on a real disk .

device, and then create a new Btrfs filesystem:

```
dd if=/dev/zero of=/home/jon/btrfs1 bs=1024 count=3072000
losetup /dev/loop0 /home/jon/btrfs1
mkfs.btrfs /dev/loop0
```

That's all there is to it. You can then mount **/dev/loop0** as you would any other filesystem, examine it with tools like **df** etc.

As with any filesystem, there are a host of options you can specify at mount time to change the way that it works. With Btrfs, one useful option is **compress**, which enables you to turn on compression using either **zlib** or **lzo**:

```
mount -o compress=lzo /dev/loop0 /mnt/btrfs
```

While compression brings the obvious advantage of letting you store more data on disk, in some circumstances it can also bring a performance benefit too. On most systems without solid state storage, there are often CPU cycles to spare, while disk I/O can be a real bottleneck. By asking the disks to pull back less data, but asking the CPU to do some more work uncompressing that data, you can improve your performance.

Subvolumes

Now that you have a Btrfs filesystem available, let's look at the second (after transparent compression) feature of interest: subvolumes. A Btrfs filesystem can be divided into multiple roots that can each be treated as a filesystem in its own right (unlike logical volumes, these independent roots are not separate block devices):

```
btrfs subvolume create /mnt/btrfs/images
```

If you inspect the mounted filesystem at this point, you'll see what appears to be a new directory. You can **cd** in to it, you can create files within it etc. What happens, however, if you try to create a hard link between a file in this subvolume and the parent **btrfs** file volume?

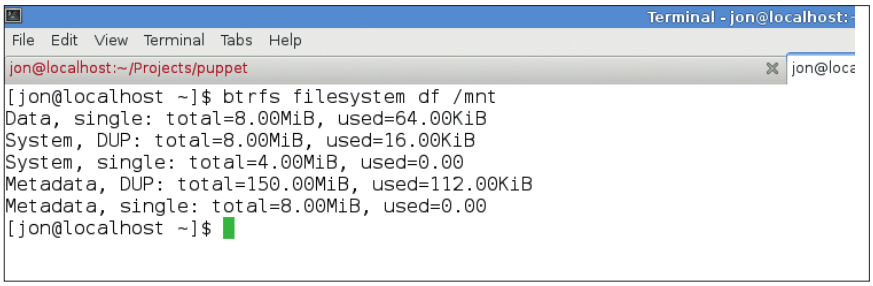
```
ln /mnt/btrfs/images/screen1.png /mnt/btrfs/screen1.png
```

That operation fails, just as if you'd tried to create a hard link between two different mount points.

OK, so what can you do with subvolumes? Well, when creating a subvolume, you can make it a snapshot of another Btrfs volume:

```
btrfs subvolume snapshot /mnt/btrfs/images /mnt/btrfs/ss-images
```

Because Btrfs is a copy-on-write filesystem, this snapshot could be an exact replica of a 300GB filesystem and it would still have been created instantly. Btrfs only needs to copy data when information in the snapshot or the original volume actually changes, making them fast to create and remove as well as extremely space efficient.



Some of the tools you've used in the past, such as df, won't take into account metadata and other features of Btrfs, so it has its own tools, such as btrfs filesystem df /path (the substitute for df).

What really makes subvolumes useful, however, is that you can mount individual subvolumes without mounting their parent. First, list all of the subvolumes in your Btrfs volumes to find out their 'subvolume IDs':

```
btrfs subvolume list /mnt/btrfs
```

Then, assuming the **ss-images** snapshot created above has volume id 258, **umount** the Btrfs filesystem before remounting with the following options:

```
mount -o subvolumeid=258 /dev/loop0 /mnt/btrfs
```

When you list the contents of **/mnt/btrfs**, you'll only see the contents of that subvolume. This feature is particularly important because it means, for example, you can snapshot your root volume before an upgrade, and if things go awry, remount the snapshot as your root and get back to a working state straight away.

> **"A Btrfs filesystem can be divided into roots that can each be treated as a filesystem in its own right."**

Multiple volumes

As well as having these LVM-like features, Btrfs also shares features with traditional RAID, too. A Btrfs filesystem can be spread across multiple devices, and you can configure it to distribute the data across the devices according to one of several common RAID levels:

- **RAID 0** Striping, in which data is striped across disks, leading to improved read and write speeds. Btrfs also supports an extension of RAID 0 in which disks do not have to be the same size, known as 'single'.
- **RAID 1** Mirroring, in which data is mirrored across two or more disks of the same size. Can be faster for reads, but will slow down writes, as data has to be written twice.
- **RAID 10** Mirrored striped, in which data is striped across a pair of drives, which is in turn mirrored to another pair of drives. Aims to give the benefits of RAID 0 and 1.
- **RAID 5 and 6** Stripe data, as in RAID 0, but sacrifice some space for 'parity' information. This parity information allows the array to lose one disk in RAID 5 or two disks in RAID 6.

To set up a multi-device Btrfs filesystem like this, first create a second loop device:

```
dd if=/dev/zero of=/home/jon/btrfs2 bs=1024 count=3072000
losetup /dev/loop1 /home/jon/btrfs2
```

Then use the **mkfs.btrfs** command again, but with the following options:

```
mkfs.btrfs -d raid0 /dev/loop0 /dev/loop1
```

You can check the man page for **mkfs.btrfs** to see other options for the **-d** switch.

Self healing

We're close to the end of this month's overview, and there's so much we haven't touched on – file cloning, filesystem mirroring with send/receive, online rebalancing (aka changing RAID levels) and much more. Before finishing this month's section, there's one other aspect of Btrfs that I'd like to draw to your attention: Btrfs aims to be self healing.

Btrfs records checksums for each block that it writes. When it reads the data, it compares the data to its checksum, and if there's a difference, it automatically tries to re-read the data from one of your redundant copies or parity information – eg if you're using RAID1/10, 5 or 6 and Btrfs reads a bad block, you'd never know it happened unless you were to take a look in the logs.

That's all we have space for, but I hope you'll start thinking about all you could do with Btrfs when it does eventually hit your favourite distribution.