EndeavourOS

BTRFS (https://wiki.archlinux.org/title/Btrfs)

Displaying used/free space

```
# btrfs filesystem usage /
$ btrfs filesystem df /```
```

Defragmentation

> Warning: A major regression in the Linux kernel version 5.16.x < 5.16.5 causes Btrfs defragmentation to infinitely loop on some systems, affecting both manual and automatic defragmentation. This causes extreme I/O loads on the affected drives, which can heavily shorten their life time and affect their performance. As such, it is not recommended to use autodefrag on these versions, one should even use noautodefrag to make sure online defragmentation is disabled.

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.

Scrub

The Btrfs Wiki 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."

```
# btrfs scrub start /
# btrfs scrub status /
```

Note: A running scrub process will prevent the system from suspending

Scrub with systemd 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.

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."

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.

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

Subvolumes

Use when:

1. Nested subvolumes are not going to be part of snapshots created from their parent subvolume. So one typical reason is to exclude certain parts of the filesystem from being snapshot.
2. "Split" of areas which are "complete" and/or "consistent" in themselves.
3. Split of areas which need special properties / mount options.

Create

(/data is a btrfs partition in this example)

```
# btrfs subvolume create /data/photos
```

List subvolumes

```
# btrfs subvolume list /data
```

Show info

```
# btrfs subvolume show /data/photos
```

Mount

```
# mount /dev/sdb1 -o subvol=photos /tmp/photos
# mount /dev/sdb1 -o subvolid=PHOTOS_ID /tmp/photos
```

Snapshots

> A snapshot is simply a subvolume that shares its data (and metadata) with some other subvolume, using Btrfs's COW capabilities.

You can make a snapshot writable and use it as an evolving clone of the original subvolume. Or you can use the snapshot as a stable image of a subvolume for backup purposes or for migration to other systems. Snapshots can be created quickly and they initially consume very little disk space.

Once a [writable] snapshot is made, there is no difference in status between the original subvolume, and the new snapshot subvolume. To roll back to a snapshot, unmount the modified original subvolume, use mv to rename the old subvolume to a temporary location, and then again to rename the snapshot to the original name. You can then remount the subvolume.

Create

```
# btrfs subvolume snapshot /btrfs/SV1 /btrfs/SV1/SV1-snap
```

Create read-only

```
# btrfs subvolume snapshot -r /btrfs/SV1 /btrfs/SV1-rosnap
```

Snapshot of a file

```
# cp --reflink /btrfs/SV1/vmlinuz-3.10.0-693.17.1.el7.x86_64 /btrfs/SV1/copy_of_vmlinuz
```

```
# = run as root
$ = run as user```
```

Generic

Get mounts:

```
cat /proc/mounts
cat /proc/self/mounts
mount -l
df -aTh```
```

You should backup:

- /etc
- /home
- /root
- /usr/local/bin
- /srv
- /opt

Filesystem Hierarchy Standard link

- **/bin** is a place for most commonly used terminal commands, like ls, mount, rm, etc.
- **/boot** contains files needed to start up the system, including the Linux kernel, a RAM disk image and bootloader configuration files.
- **/dev** contains all device files, which are not regular files but instead refer to various hardware devices on the system, including hard drives.
- **/etc** contains system-global configuration files, which affect the system's behavior for all users.
- **/home** home sweet home, this is the place for users' home directories.
- **/lib** contains very important dynamic libraries and kernel modules
- **/media** is intended as a mount point for external devices, such as hard drives or removable media (floppies, CDs, DVDs).
- **/mnt** is also a place for mount points, but dedicated specifically to "temporarily mounted" devices, such as network filesystems.
- **/opt** can be used to store additional software for your system, which is not handled by the package manager.
- **/proc** is a virtual filesystem that provides a mechanism for kernel to send information to processes.
- **/root** is the superuser's home directory, not in /home/ to allow for booting the system even if /home/ is not available.
- **/run** is a tmpfs (temporary file system) available early in the boot process where ephemeral run-time data is stored. Files under this directory are removed or truncated at the beginning of the boot process.

(It deprecates various legacy locations such as /var/run, /var/lock, /lib/init/rw in otherwise non-ephemeral directory trees as well as /dev/.* and /dev/shm which are not device files.)

- **/sbin** contains important administrative commands that should generally only be employed by the superuser.
- **/srv** can contain data directories of services such as HTTP (/srv/www/) or FTP.
- **/sys** is a virtual filesystem that can be accessed to set or obtain information about the kernel's view of the system.
- **/tmp** is a place for temporary files used by applications.
- **/usr** contains the majority of user utilities and applications, and partly replicates the root directory structure, containing for instance, among others, /usr/bin/ and /usr/lib.
- **/var** is dedicated to variable data, such as logs, databases, websites, and temporary spool (e-mail etc.) files that persist from one boot to the next. A notable directory it contains is /var/log where system log files are kept.

Neofetch/Onefetch

`neofetch` displays system information. `onefetch` displays git repo information.

```
# = run as root
$ = run as user```
```

Helper

Commands

```
$ cd
$ ls
$ lscpu
$ cat file
$ cat file1 file2 > file3 (joins two files and pipes into file3)
$ mv file "new file path"
$ rm filename
$ mkdir
$ rmdir
$ uname -a (Show system and kernel)
```

Environment Variables

```
$ env
$ $PATH
```

Add to path:

```
$ export PATH="$HOME/bin:$PATH"
```

Here, $HOME/bin is a path pointing to /home/myusername/bin

To make the change permanent, you need to define the $PATH variable in the shell configuration files. In most Linux distributions when you start a new session, environment variables are read from the following files:

- Global shell specific configuration files such as `/etc/environment` and `/etc/profile`. Use this file if you want the new directory to be added to all system users $PATH.
- Per-user shell specific configuration files. For example, if you are using Bash, you can set the $PATH variable in the `~/.bashrc` file. If you are using Zsh the file name is `~/.zshrc`.

Shortcuts

Bash

```
CTRL-a Go to start of line
CTRL-e Go to end of line
```