Red Hat Enterprise Linux 8

Managing file systems

Creating, modifying, and administering file systems in Red Hat Enterprise Linux 8

Last Updated: 2019-09-19

Red Hat Enterprise Linux 8 Managing file systems

Creating, modifying, and administering file systems in Red Hat Enterprise Linux 8

Legal Notice

Abstract

This documentation collection provides instructions on how to effectively manage file systems in Red Hat Enterprise Linux 8.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Please let us know how we could make it better. To do so:

- For simple comments on specific passages, make sure you are viewing the documentation in the Multi-page HTML format. Highlight the part of text that you want to comment on. Then, click the **Add Feedback** pop-up that appears below the highlighted text, and follow the displayed instructions.
- For submitting more complex feedback, create a Bugzilla ticket:
    1. Go to the Bugzilla website.
    2. As the Component, use **Documentation**.
    3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
    4. Click **Submit Bug**.

CHAPTER 1. OVERVIEW OF AVAILABLE FILE SYSTEMS

Choosing the file system that is appropriate for your application is an important decision due to the large number of options available and the trade-offs involved. This chapter describes some of the file systems that ship with Red Hat Enterprise Linux 8 and provides historical background and recommendations on the right file system to suit your application.

1.1. TYPES OF FILE SYSTEMS

Red Hat Enterprise Linux 8 supports a variety of file systems (FS). Different types of file systems solve different kinds of problems, and their usage is application specific. At the most general level, available file systems can be grouped into the following major types:

Table 1.1. Types of file systems and their use cases

Type	File system	Attributes and use cases
Disk or local FS	XFS	XFS is the default file system in RHEL. Because it lays out files as extents, it is less vulnerable to fragmentation than ext4. Red Hat recommends deploying XFS as your local file system unless there are specific reasons to do otherwise: for example, compatibility or corner cases around performance.
	ext4	ext4 has the benefit of longevity in Linux. Therefore, it is supported by almost all Linux applications. In most cases, it rivals XFS on performance. ext4 is commonly used for home directories.
Network or client-and-server FS	NFS	Use NFS to share files between multiple systems on the same network.
	SMB	Use SMB for file sharing with Microsoft Windows systems.
Shared storage or shared disk FS	GFS2	GFS2 provides shared write access to members of a compute cluster. The emphasis is on stability and reliability, with the functional experience of a local file system as possible. SAS Grid, Tibco MQ, IBM Websphere MQ, and Red Hat Active MQ have been deployed successfully on GFS2.
Volume-managing FS	Stratis (Technology Preview)	Stratis is a volume manager built on a combination of XFS and LVM. The purpose of Stratis is to emulate capabilities offered by volume-managing file systems like Btrfs and ZFS. It is possible to build this stack manually, but Stratis reduces configuration complexity, implements best practices, and consolidates error information.

1.2. LOCAL FILE SYSTEMS

Local file systems are file systems that run on a single, local server and are directly attached to storage.

For example, a local file system is the only choice for internal SATA or SAS disks, and is used when your server has internal hardware RAID controllers with local drives. Local file systems are also the most common file systems used on SAN attached storage when the device exported on the SAN is not shared.

All local file systems are POSIX-compliant and are fully compatible with all supported Red Hat Enterprise Linux releases. POSIX-compliant file systems provide support for a well-defined set of system calls, such as **read()**, **write()**, and **seek()**.

From the application programmer's point of view, there are relatively few differences between local file systems. The most notable differences from a user's perspective are related to scalability and performance. When considering a file system choice, consider how large the file system needs to be, what unique features it should have, and how it performs under your workload.

Available local file systems

- XFS
- ext4

1.3. THE XFS FILE SYSTEM

XFS is a highly scalable, high-performance, robust, and mature 64-bit journaling file system that supports very large files and file systems on a single host. It is the default file system in Red Hat Enterprise Linux 8. XFS was originally developed in the early 1990s by SGI and has a long history of running on extremely large servers and storage arrays.

The features of XFS include:

Reliability

- Metadata journaling, which ensures file system integrity after a system crash by keeping a record of file system operations that can be replayed when the system is restarted and the file system remounted
- Extensive run-time metadata consistency checking
- Scalable and fast repair utilities
- Quota journaling. This avoids the need for lengthy quota consistency checks after a crash.

Scalability and performance

- Supported file system size up to 1024 TiB
- Ability to support a large number of concurrent operations
- B-tree indexing for scalability of free space management
- Sophisticated metadata read-ahead algorithms
- Optimizations for streaming video workloads

Allocation schemes

- Extent-based allocation
- Stripe-aware allocation policies
- Delayed allocation
- Space pre-allocation
- Dynamically allocated inodes

Other features

- Reflink-based file copies (new in Red Hat Enterprise Linux 8)
- Tightly integrated backup and restore utilities
- Online defragmentation
- Online file system growing
- Comprehensive diagnostics capabilities
- Extended attributes (**xattr**). This allows the system to associate several additional name/value pairs per file.
- Project or directory quotas. This allows quota restrictions over a directory tree.
- Subsecond timestamps

Performance characteristics

XFS has a high performance on large systems with enterprise workloads. A large system is one with a relatively high number of CPUs, multiple HBAs, and connections to external disk arrays. XFS also performs well on smaller systems that have a multi-threaded, parallel I/O workload.

XFS has a relatively low performance for single threaded, metadata-intensive workloads: for example, a workload that creates or deletes large numbers of small files in a single thread.

1.4. THE EXT4 FILE SYSTEM

The ext4 file system is the fourth generation of the ext file system family. It was the default file system in Red Hat Enterprise Linux 6.

The ext4 driver can read and write to ext2 and ext3 file systems, but the ext4 file system format is not compatible with ext2 and ext3 drivers.

ext4 adds several new and improved features, such as:

- Supported file system size up to 50 TiB
- Extent-based metadata
- Delayed allocation
- Journal checksumming
- Large storage support

The extent-based metadata and the delayed allocation features provide a more compact and efficient way to track utilized space in a file system. These features improve file system performance and reduce the space consumed by metadata. Delayed allocation allows the file system to postpone selection of the permanent location for newly written user data until the data is flushed to disk. This enables higher performance since it can allow for larger, more contiguous allocations, allowing the file system to make decisions with much better information.

File system repair time using the **fsck** utility in ext4 is much faster than in ext2 and ext3. Some file system repairs have demonstrated up to a six-fold increase in performance.

1.5. CHOOSING A LOCAL FILE SYSTEM

To choose a file system that meets your application requirements, you need to understand the target system on which you are going to deploy the file system. You can use the following questions to inform your decision:

- Do you have a large server?
- Do you have large storage requirements or have a local, slow SATA drive?
- What kind of I/O workload do you expect your application to present?
- What are your throughput and latency requirements?
- How stable is your server and storage hardware?
- What is the typical size of your files and data set?
- If the system fails, how much downtime can you suffer?

If both your server and your storage device are large, XFS is the best choice. Even with smaller storage arrays, XFS performs very well when the average file sizes are large (for example, hundreds of megabytes in size).

If your existing workload has performed well with ext4, staying with ext4 should provide you and your applications with a very familiar environment.

The ext4 file system tends to perform better on systems that have limited I/O capability. It performs better on limited bandwidth (less than 200MB/s) and up to around 1000 IOPS capability. For anything with higher capability, XFS tends to be faster.

XFS consumes about twice the CPU-per-metadata operation compared to ext4, so if you have a CPU-bound workload with little concurrency, then ext4 will be faster. In general, ext4 is better if an application uses a single read/write thread and small files, while XFS shines when an application uses multiple read/write threads and bigger files.

You cannot shrink an XFS file system. If you need to be able to shrink the file system, consider using ext4, which supports offline shrinking.

In general, Red Hat recommends that you use XFS unless you have a specific use case for ext4. You should also measure the performance of your specific application on your target server and storage system to make sure that you choose the appropriate type of file system.

Table 1.2. Summary of local file system recommendations

Scenario	Recommended file system
No special use case	XFS
Large server	XFS
Large storage devices	XFS
Large files	XFS
Multi-threaded I/O	XFS
Single-threaded I/O	ext4
Limited I/O capability (under 1000 IOPS)	ext4
Limited bandwidth (under 200MB/s)	ext4
CPU-bound workload	ext4
Support for offline shrinking	ext4

1.6. NETWORK FILE SYSTEMS

Network file systems, also referred to as client/server file systems, enable client systems to access files that are stored on a shared server. This makes it possible for multiple users on multiple systems to share files and storage resources.

Such file systems are built from one or more servers that export a set of file systems to one or more clients. The client nodes do not have access to the underlying block storage, but rather interact with the storage using a protocol that allows for better access control.

Available network file systems

- The most common client/server file system for RHEL customers is the NFS file system. RHEL provides both an NFS server component to export a local file system over the network and an NFS client to import these file systems.
- RHEL also includes a CIFS client that supports the popular Microsoft SMB file servers for Windows interoperability. The userspace Samba server provides Windows clients with a Microsoft SMB service from a RHEL server.

1.7. SHARED STORAGE FILE SYSTEMS

Shared storage file systems, sometimes referred to as cluster file systems, give each server in the cluster direct access to a shared block device over a local storage area network (SAN).

Comparison with network file systems

Like client/server file systems, shared storage file systems work on a set of servers that are all members of a cluster. Unlike NFS, however, no single server provides access to data or metadata to other

members: each member of the cluster has direct access to the same storage device (the *shared storage*), and all cluster member nodes access the same set of files.

Concurrency

Cache coherency is key in a clustered file system to ensure data consistency and integrity. There must be a single version of all files in a cluster visible to all nodes within a cluster. The file system must prevent members of the cluster from updating the same storage block at the same time and causing data corruption. In order to do that, shared storage file systems use a cluster wide-locking mechanism to arbitrate access to the storage as a concurrency control mechanism. For example, before creating a new file or writing to a file that is opened on multiple servers, the file system component on the server must obtain the correct lock.

The requirement of cluster file systems is to provide a highly available service like an Apache web server. Any member of the cluster will see a fully coherent view of the data stored in their shared disk file system, and all updates will be arbitrated correctly by the locking mechanisms.

Performance characteristics

Performance of shared disk file systems is normally less than that of a local file system running on the same system since it has to account for the cost of the locking overhead. Shared disk file systems perform well with workloads where each node writes almost exclusively to a particular set of files that are not shared with other nodes or where a set of files is shared in an almost exclusively read-only manner across a set of nodes. This results in a minimum of cross-node cache invalidation and can maximize performance.

Setting up a shared disk file system is complex, and tuning an application to perform well on a shared disk file system can be challenging.

Available shared storage file systems

- Red Hat Enterprise Linux provides the GFS2 file system. GFS2 comes tightly integrated with the Red Hat Enterprise Linux High Availability Add-On and the Resilient Storage Add-On. Red Hat Enterprise Linux supports GFS2 on clusters that range in size from 2 to 16 nodes.

1.8. CHOOSING BETWEEN NETWORK AND SHARED STORAGE FILE SYSTEMS

When choosing between network and shared storage file systems, consider the following points:

- NFS-based network file systems are an extremely common and popular choice for environments that provide NFS servers.

- Network file systems can be deployed using very high-performance networking technologies like Infiniband or 10 Gigabit Ethernet. This means that you should not turn to shared storage file systems just to get raw bandwidth to your storage. If the speed of access is of prime importance, then use NFS to export a local file system like XFS.

- Shared storage file systems are not easy to set up or to maintain, so you should deploy them only when you cannot provide your required availability with either local or network file systems.

- A shared storage file system in a clustered environment helps reduce downtime by eliminating the steps needed for unmounting and mounting that need to be done during a typical fail-over scenario involving the relocation of a high-availability service.

Red Hat recommends that you use network file systems unless you have a specific use case for shared storage file systems. Use shared storage file systems primarily for deployments that need to provide high-availability services with minimum downtime and have stringent service-level requirements.

1.9. VOLUME-MANAGING FILE SYSTEMS

Volume-managing file systems integrate the entire storage stack for the purposes of simplicity and in-stack optimization.

Available volume-managing file systems

- Red Hat Enterprise Linux 8 provides the Stratis volume manager as a Technology Preview. Stratis uses XFS for the file system layer and integrates it with LVM, Device Mapper, and other components.
  Stratis was first released in Red Hat Enterprise Linux 8.0. It is conceived to fill the gap created when Red Hat deprecated Btrfs. Stratis 1.0 is an intuitive, command line-based volume manager that can perform significant storage management operations while hiding the complexity from the user:

  - Volume management
  - Pool creation
  - Thin storage pools
  - Snapshots
  - Automated read cache

  Stratis offers powerful features, but currently lacks certain capabilities of other offerings that it might be compared to, such as Btrfs or ZFS. Most notably, it does not support CRCs with self healing.

CHAPTER 2. MOUNTING NFS SHARES

As a system administrator, you can mount remote NFS shares on your system to access shared data.

2.1. INTRODUCTION TO NFS

This section explains the basic concepts of the NFS service.

A Network File System (NFS) allows remote hosts to mount file systems over a network and interact with those file systems as though they are mounted locally. This enables you to consolidate resources onto centralized servers on the network.

The NFS server refers to the **/etc/exports** configuration file to determine whether the client is allowed to access any exported file systems. Once verified, all file and directory operations are available to the user.

2.2. SUPPORTED NFS VERSIONS

This section lists versions of NFS supported in Red Hat Enterprise Linux and their features.

Currently, Red Hat Enterprise Linux 8 supports the following major versions of NFS:

- NFS version 3 (NFSv3) supports safe asynchronous writes and is more robust at error handling than the previous NFSv2; it also supports 64-bit file sizes and offsets, allowing clients to access more than 2 GB of file data.
- NFS version 4 (NFSv4) works through firewalls and on the Internet, no longer requires an **rpcbind** service, supports Access Control Lists (ACLs), and utilizes stateful operations.

NFS version 2 (NFSv2) is no longer supported by Red Hat.

Default NFS version

The default NFS version in Red Hat Enterprise Linux 8 is 4.2. NFS clients attempt to mount using NFSv4.2 by default, and fall back to NFSv4.1 when the server does not support NFSv4.2. The mount later falls back to NFSv4.0 and then to NFSv3.

Features of minor NFS versions

Following are the features of NFSv4.2 in Red Hat Enterprise Linux 8:

Server-side copy

Enables the NFS client to efficiently copy data without wasting network resources using the **copy_file_range()** system call.

Sparse files

Enables files to have one or more *holes*, which are unallocated or uninitialized data blocks consisting only of zeroes. The **lseek()** operation in NFSv4.2 supports **seek_hole()** and **seek_data()**, which enables applications to map out the location of holes in the sparse file.

Space reservation

Permits storage servers to reserve free space, which prohibits servers to run out of space. NFSv4.2 supports the **allocate()** operation to reserve space, the **deallocate()** operation to unreserve space, and the **fallocate()** operation to preallocate or deallocate space in a file.

Labeled NFS

Enforces data access rights and enables SELinux labels between a client and a server for individual files on an NFS file system.

Layout enhancements

Provides the **layoutstats()** operation, which enables some Parallel NFS (pNFS) servers to collect better performance statistics.

Following are the features of NFSv4.1:

- Enhances performance and security of network, and also includes client-side support for pNFS.
- No longer requires a separate TCP connection for callbacks, which allows an NFS server to grant delegations even when it cannot contact the client: for example, when NAT or a firewall interferes.
- Provides exactly once semantics (except for reboot operations), preventing a previous issue whereby certain operations sometimes returned an inaccurate result if a reply was lost and the operation was sent twice.

2.3. SERVICES REQUIRED BY NFS

This section lists system services that are required for running an NFS server or mounting NFS shares. Red Hat Enterprise Linux starts these services automatically.

Red Hat Enterprise Linux uses a combination of kernel-level support and service processes to provide NFS file sharing. All NFS versions rely on Remote Procedure Calls (RPC) between clients and servers. To share or mount NFS file systems, the following services work together depending on which version of NFS is implemented:

nfsd

The NFS server to service requests for shared NFS file systems.

rpcbind

Accepts port reservations from local RPC services. These ports are then made available (or advertised) so the corresponding remote RPC services can access them. The **rpcbind** service responds to requests for RPC services and sets up connections to the requested RPC service. This is not used with NFSv4.

rpc.mountd

This process is used by an NFS server to process **MOUNT** requests from NFSv3 clients. It checks that the requested NFS share is currently exported by the NFS server, and that the client is allowed to access it. If the mount request is allowed, the **nfs-mountd** service replies with a Success status and provides the File-Handle for this NFS share back to the NFS client.

rpc.nfsd

This process enables explicit NFS versions and protocols the server advertises to be defined. It works with the Linux kernel to meet the dynamic demands of NFS clients, such as providing server threads each time an NFS client connects. This process corresponds to the **nfs-server** service.

lockd

This is a kernel thread that runs on both clients and servers. It implements the Network Lock Manager (NLM) protocol, which enables NFSv3 clients to lock files on the server. It is started automatically whenever the NFS server is run and whenever an NFS file system is mounted.

rpc.statd

This process implements the Network Status Monitor (NSM) RPC protocol, which notifies NFS clients when an NFS server is restarted without being gracefully brought down. The **rpc-statd** service is started automatically by the **nfs-server** service, and does not require user configuration. This is not used with NFSv4.

rpc.rquotad

This process provides user quota information for remote users. The **rpc-rquotad** service is started automatically by the **nfs-server** service and does not require user configuration.

rpc.idmapd

This process provides NFSv4 client and server upcalls, which map between on-the-wire NFSv4 names (strings in the form of ***user@domain***) and local UIDs and GIDs. For **idmapd** to function with NFSv4, the **/etc/idmapd.conf** file must be configured. At a minimum, the **Domain** parameter should be specified, which defines the NFSv4 mapping domain. If the NFSv4 mapping domain is the same as the DNS domain name, this parameter can be skipped. The client and server must agree on the NFSv4 mapping domain for ID mapping to function properly.
Only the NFSv4 server uses **rpc.idmapd**, which is started by the **nfs-idmapd** service. The NFSv4 client uses the keyring-based **nfsidmap** utility, which is called by the kernel on-demand to perform ID mapping. If there is a problem with **nfsidmap**, the client falls back to using **rpc.idmapd**.

The RPC services with NFSv4

The mounting and locking protocols have been incorporated into the NFSv4 protocol. The server also listens on the well-known TCP port 2049. As such, NFSv4 does not need to interact with **rpcbind**, **lockd**, and **rpc-statd** services. The **nfs-mountd** service is still required on the NFS server to set up the exports, but is not involved in any over-the-wire operations.

Additional resources

- To configure an NFSv4-only server, which does not require **rpcbind**, see Section 3.14, “Configuring an NFSv4-only server”.

2.4. NFS HOST NAME FORMATS

This section describes different formats that you can use to specify a host when mounting or exporting an NFS share.

You can specify the host in the following formats:

Single machine

Either of the following:

- A fully-qualified domain name (that can be resolved by the server)
- Host name (that can be resolved by the server)
- An IP address.

Series of machines specified with wildcards

You can use the * or **?** characters to specify a string match.
Wildcards are not to be used with IP addresses; however, they might accidentally work if reverse DNS lookups fail. When specifying wildcards in fully qualified domain names, dots (**.**) are not included in the wildcard. For example, ***.example.com** includes **one.example.com** but does not include **one.two.example.com**.

IP networks

Either of the following formats is valid:

- ***a.b.c.d/z***, where ***a.b.c.d*** is the network and ***z*** is the number of bits in the netmask; for example **192.168.0.0/24**.

- ***a.b.c.d/netmask***, where ***a.b.c.d*** is the network and ***netmask*** is the netmask; for example, **192.168.100.8/255.255.255.0**.

Netgroups

The ***@group-name*** format , where ***group-name*** is the NIS netgroup name.

2.5. INSTALLING NFS

This procedure installs all packages necessary to mount or export NFS shares.

Procedure

- Install the **nfs-utils** package:

```
# yum install nfs-utils
```

2.6. DISCOVERING NFS EXPORTS

This procedure discovers which file systems a given NFSv3 or NFSv4 server exports.

Procedure

- With any server that supports NFSv3, use the **showmount** utility:

```
$ showmount --exports my-server

Export list for my-server
/exports/foo
/exports/bar
```

- With any server that supports NFSv4, mount the root directory and look around:

```
# mount my-server:/ /mnt/
# ls /mnt/

exports

# ls /mnt/exports/

foo
bar
```

On servers that support both NFSv4 and NFSv3, both methods work and give the same results.

Additional resources

- The **showmount(8)** man page.

2.7. MOUNTING AN NFS SHARE WITH MOUNT

This procedure mounts an NFS share exported from a server using the **mount** utility.

Procedure

- To mount an NFS share, use the following command:

```
# mount -t nfs -o options host:/remote/export /local/directory
```

This command uses the following variables:

options

A comma-delimited list of mount options.

host

The host name, IP address, or fully qualified domain name of the server exporting the file system you wish to mount.

/remote/export

The file system or directory being exported from the server, that is, the directory you wish to mount.

/local/directory

The client location where */remote/export* is mounted.

Additional resources

- Section 2.8, "Common NFS mount options"
- Section 2.4, "NFS host name formats"
- Section 10.3, "Mounting a file system with mount"
- The **mount(8)** man page

2.8. COMMON NFS MOUNT OPTIONS

This section lists options commonly used when mounting NFS shares. These options can be used with manual mount commands, **/etc/fstab** settings, and **autofs**.

Common NFS mount options

lookupcache=*mode*

Specifies how the kernel should manage its cache of directory entries for a given mount point. Valid arguments for *mode* are **all**, **none**, or **positive**.

nfsvers=*version*

Specifies which version of the NFS protocol to use, where *version* is **3**, **4**, **4.0**, **4.1**, or **4.2**. This is useful for hosts that run multiple NFS servers, or to disable retrying a mount with lower versions. If no version is specified, NFS uses the highest version supported by the kernel and the **mount** utility. The option **vers** is identical to **nfsvers**, and is included in this release for compatibility reasons.

noacl

Turns off all ACL processing. This may be needed when interfacing with older versions of Red Hat Enterprise Linux, Red Hat Linux, or Solaris, because the most recent ACL technology is not compatible with older systems.

nolock

Disables file locking. This setting is sometimes required when connecting to very old NFS servers.

noexec

Prevents execution of binaries on mounted file systems. This is useful if the system is mounting a non-Linux file system containing incompatible binaries.

nosuid

Disables the **set-user-identifier** and **set-group-identifier** bits. This prevents remote users from gaining higher privileges by running a **setuid** program.

port=*num*

Specifies the numeric value of the NFS server port. If *num* is **0** (the default value), then **mount** queries the **rpcbind** service on the remote host for the port number to use. If the NFS service on the remote host is not registered with its **rpcbind** service, the standard NFS port number of TCP 2049 is used instead.

rsize=*num* and **wsize=*num***

These options set the maximum number of bytes to be transferred in a single NFS read or write operation.
There is no fixed default value for **rsize** and **wsize**. By default, NFS uses the largest possible value that both the server and the client support. In Red Hat Enterprise Linux 8, the client and server maximum is 1,048,576 bytes. For more details, see the What are the default and maximum values for rsize and wsize with NFS mounts? KBase article.

sec=*mode*

Security flavors to use for accessing files on the mounted export.
The default setting is **sec=sys**, which uses local UNIX UIDs and GIDs. These use **AUTH_SYS** to authenticate NFS operations.

Other options include:

- **sec=krb5** uses Kerberos V5 instead of local UNIX UIDs and GIDs to authenticate users.
- **sec=krb5i** uses Kerberos V5 for user authentication and performs integrity checking of NFS operations using secure checksums to prevent data tampering.
- **sec=krb5p** uses Kerberos V5 for user authentication, integrity checking, and encrypts NFS traffic to prevent traffic sniffing. This is the most secure setting, but it also involves the most performance overhead.

tcp

Instructs the NFS mount to use the TCP protocol.

Additional resources

- The **mount(8)** man page
- The **nfs(5)** man page

2.9. RELATED INFORMATION

- The Linux NFS wiki: https://linux-nfs.org
- To mount NFS shares persistently, see Section 10.8, “Persistently mounting file systems”.
- To mount NFS shares on demand, see Section 10.9, “Mounting file systems on demand”.

CHAPTER 3. EXPORTING NFS SHARES

As a system administrator, you can use the NFS server to share a directory on your system over network.

3.1. INTRODUCTION TO NFS

This section explains the basic concepts of the NFS service.

A Network File System (NFS) allows remote hosts to mount file systems over a network and interact with those file systems as though they are mounted locally. This enables you to consolidate resources onto centralized servers on the network.

The NFS server refers to the **/etc/exports** configuration file to determine whether the client is allowed to access any exported file systems. Once verified, all file and directory operations are available to the user.

3.2. SUPPORTED NFS VERSIONS

This section lists versions of NFS supported in Red Hat Enterprise Linux and their features.

Currently, Red Hat Enterprise Linux 8 supports the following major versions of NFS:

- NFS version 3 (NFSv3) supports safe asynchronous writes and is more robust at error handling than the previous NFSv2; it also supports 64-bit file sizes and offsets, allowing clients to access more than 2 GB of file data.
- NFS version 4 (NFSv4) works through firewalls and on the Internet, no longer requires an **rpcbind** service, supports Access Control Lists (ACLs), and utilizes stateful operations.

NFS version 2 (NFSv2) is no longer supported by Red Hat.

Default NFS version

The default NFS version in Red Hat Enterprise Linux 8 is 4.2. NFS clients attempt to mount using NFSv4.2 by default, and fall back to NFSv4.1 when the server does not support NFSv4.2. The mount later falls back to NFSv4.0 and then to NFSv3.

Features of minor NFS versions

Following are the features of NFSv4.2 in Red Hat Enterprise Linux 8:

Server-side copy

Enables the NFS client to efficiently copy data without wasting network resources using the **copy_file_range()** system call.

Sparse files

Enables files to have one or more *holes*, which are unallocated or uninitialized data blocks consisting only of zeroes. The **lseek()** operation in NFSv4.2 supports **seek_hole()** and **seek_data()**, which enables applications to map out the location of holes in the sparse file.

Space reservation

Permits storage servers to reserve free space, which prohibits servers to run out of space. NFSv4.2 supports the **allocate()** operation to reserve space, the **deallocate()** operation to unreserve space, and the **fallocate()** operation to preallocate or deallocate space in a file.

Labeled NFS

Enforces data access rights and enables SELinux labels between a client and a server for individual files on an NFS file system.

Layout enhancements

Provides the **layoutstats()** operation, which enables some Parallel NFS (pNFS) servers to collect better performance statistics.

Following are the features of NFSv4.1:

- Enhances performance and security of network, and also includes client-side support for pNFS.
- No longer requires a separate TCP connection for callbacks, which allows an NFS server to grant delegations even when it cannot contact the client: for example, when NAT or a firewall interferes.
- Provides exactly once semantics (except for reboot operations), preventing a previous issue whereby certain operations sometimes returned an inaccurate result if a reply was lost and the operation was sent twice.

3.3. THE TCP AND UDP PROTOCOLS IN NFSV3 AND NFSV4

NFSv4 requires the Transmission Control Protocol (TCP) running over an IP network.

NFSv3 could also use the User Datagram Protocol (UDP) in earlier Red Hat Enterprise Linux versions. In Red Hat Enterprise Linux 8, NFS over UDP is no longer supported. By default, UDP is disabled in the NFS server.

3.4. SERVICES REQUIRED BY NFS

This section lists system services that are required for running an NFS server or mounting NFS shares. Red Hat Enterprise Linux starts these services automatically.

Red Hat Enterprise Linux uses a combination of kernel-level support and service processes to provide NFS file sharing. All NFS versions rely on Remote Procedure Calls (RPC) between clients and servers. To share or mount NFS file systems, the following services work together depending on which version of NFS is implemented:

nfsd

The NFS server to service requests for shared NFS file systems.

rpcbind

Accepts port reservations from local RPC services. These ports are then made available (or advertised) so the corresponding remote RPC services can access them. The **rpcbind** service responds to requests for RPC services and sets up connections to the requested RPC service. This is not used with NFSv4.

rpc.mountd

This process is used by an NFS server to process **MOUNT** requests from NFSv3 clients. It checks that the requested NFS share is currently exported by the NFS server, and that the client is allowed to access it. If the mount request is allowed, the **nfs-mountd** service replies with a Success status and provides the File-Handle for this NFS share back to the NFS client.

rpc.nfsd

This process enables explicit NFS versions and protocols the server advertises to be defined. It works with the Linux kernel to meet the dynamic demands of NFS clients, such as providing server threads each time an NFS client connects. This process corresponds to the **nfs-server** service.

lockd

This is a kernel thread that runs on both clients and servers. It implements the Network Lock Manager (NLM) protocol, which enables NFSv3 clients to lock files on the server. It is started automatically whenever the NFS server is run and whenever an NFS file system is mounted.

rpc.statd

This process implements the Network Status Monitor (NSM) RPC protocol, which notifies NFS clients when an NFS server is restarted without being gracefully brought down. The **rpc-statd** service is started automatically by the **nfs-server** service, and does not require user configuration. This is not used with NFSv4.

rpc.rquotad

This process provides user quota information for remote users. The **rpc-rquotad** service is started automatically by the **nfs-server** service and does not require user configuration.

rpc.idmapd

This process provides NFSv4 client and server upcalls, which map between on-the-wire NFSv4 names (strings in the form of ***user@domain***) and local UIDs and GIDs. For **idmapd** to function with NFSv4, the **/etc/idmapd.conf** file must be configured. At a minimum, the **Domain** parameter should be specified, which defines the NFSv4 mapping domain. If the NFSv4 mapping domain is the same as the DNS domain name, this parameter can be skipped. The client and server must agree on the NFSv4 mapping domain for ID mapping to function properly.

Only the NFSv4 server uses **rpc.idmapd**, which is started by the **nfs-idmapd** service. The NFSv4 client uses the keyring-based **nfsidmap** utility, which is called by the kernel on-demand to perform ID mapping. If there is a problem with **nfsidmap**, the client falls back to using **rpc.idmapd**.

The RPC services with NFSv4

The mounting and locking protocols have been incorporated into the NFSv4 protocol. The server also listens on the well-known TCP port 2049. As such, NFSv4 does not need to interact with **rpcbind**, **lockd**, and **rpc-statd** services. The **nfs-mountd** service is still required on the NFS server to set up the exports, but is not involved in any over-the-wire operations.

Additional resources

- To configure an NFSv4-only server, which does not require **rpcbind**, see Section 3.14, “Configuring an NFSv4-only server”.

3.5. NFS HOST NAME FORMATS

This section describes different formats that you can use to specify a host when mounting or exporting an NFS share.

You can specify the host in the following formats:

Single machine

Either of the following:

- A fully-qualified domain name (that can be resolved by the server)
- Host name (that can be resolved by the server)
- An IP address.

Series of machines specified with wildcards

You can use the ***** or **?** characters to specify a string match.

Wildcards are not to be used with IP addresses; however, they might accidentally work if reverse DNS

lookups fail. When specifying wildcards in fully qualified domain names, dots (**.**) are not included in the wildcard. For example, ***.example.com** includes **one.example.com** but does not include **one.two.example.com**.

IP networks

Either of the following formats is valid:

- ***a.b.c.d/z***, where ***a.b.c.d*** is the network and ***z*** is the number of bits in the netmask; for example **192.168.0.0/24**.
- ***a.b.c.d/netmask***, where ***a.b.c.d*** is the network and ***netmask*** is the netmask; for example, **192.168.100.8/255.255.255.0**.

Netgroups

The ***@group-name*** format , where ***group-name*** is the NIS netgroup name.

3.6. NFS SERVER CONFIGURATION

This section describes the syntax and options of two ways to configure exports on an NFS server:

- Manually editing the **/etc/exports** configuration file
- Using the **exportfs** utility on the command line

3.6.1. The /etc/exports configuration file

The **/etc/exports** file controls which file systems are exported to remote hosts and specifies options. It follows the following syntax rules:

- Blank lines are ignored.
- To add a comment, start a line with the hash mark (**#**).
- You can wrap long lines with a backslash (\).
- Each exported file system should be on its own individual line.
- Any lists of authorized hosts placed after an exported file system must be separated by space characters.
- Options for each of the hosts must be placed in parentheses directly after the host identifier, without any spaces separating the host and the first parenthesis.

Export entry

Each entry for an exported file system has the following structure:

```
export host(options)
```

It is also possible to specify multiple hosts, along with specific options for each host. To do so, list them on the same line as a space-delimited list, with each host name followed by its respective options (in parentheses), as in:

```
export host1(options1) host2(options2) host3(options3)
```

In this structure:

export

The directory being exported

host

The host or network to which the export is being shared

options

The options to be used for host

Example 3.1. A simple /etc/exports file

In its simplest form, the **/etc/exports** file only specifies the exported directory and the hosts permitted to access it:

```
/exported/directory bob.example.com
```

Here, **bob.example.com** can mount **/exported/directory/** from the NFS server. Because no options are specified in this example, NFS uses default options.

IMPORTANT

The format of the **/etc/exports** file is very precise, particularly in regards to use of the space character. Remember to always separate exported file systems from hosts and hosts from one another with a space character. However, there should be no other space characters in the file except on comment lines.

For example, the following two lines do not mean the same thing:

```
/home bob.example.com(rw)
/home bob.example.com (rw)
```

The first line allows only users from **bob.example.com** read and write access to the **/home** directory. The second line allows users from **bob.example.com** to mount the directory as read-only (the default), while the rest of the world can mount it read/write.

Default options

The default options for an export entry are:

ro

The exported file system is read-only. Remote hosts cannot change the data shared on the file system. To allow hosts to make changes to the file system (that is, read and write), specify the rw option.

sync

The NFS server will not reply to requests before changes made by previous requests are written to disk. To enable asynchronous writes instead, specify the option **async**.

wdelay

The NFS server will delay writing to the disk if it suspects another write request is imminent. This can improve performance as it reduces the number of times the disk must be accessed by separate write commands, thereby reducing write overhead. To disable this, specify the **no_wdelay** option, which is available only if the default sync option is also specified.

root_squash

This prevents root users connected remotely (as opposed to locally) from having root privileges; instead, the NFS server assigns them the user ID **nfsnobody**. This effectively "squashes" the power of the remote root user to the lowest local user, preventing possible unauthorized writes on the remote server. To disable root squashing, specify the **no_root_squash** option.
To squash every remote user (including root), use the **all_squash** option. To specify the user and group IDs that the NFS server should assign to remote users from a particular host, use the **anonuid** and **anongid** options, respectively, as in:

```
export host(anonuid=uid,anongid=gid)
```

Here, *uid* and *gid* are user ID number and group ID number, respectively. The **anonuid** and **anongid** options enable you to create a special user and group account for remote NFS users to share.

By default, access control lists (ACLs) are supported by NFS under Red Hat Enterprise Linux. To disable this feature, specify the **no_acl** option when exporting the file system.

Default and overridden options
Each default for every exported file system must be explicitly overridden. For example, if the **rw** option is not specified, then the exported file system is shared as read-only. The following is a sample line from **/etc/exports** which overrides two default options:

```
/another/exported/directory 192.168.0.3(rw,async)
```

In this example, **192.168.0.3** can mount **/another/exported/directory/** read and write, and all writes to disk are asynchronous.

3.6.2. The exportfs utility

The **exportfs** utility enables the root user to selectively export or unexport directories without restarting the NFS service. When given the proper options, the **exportfs** utility writes the exported file systems to **/var/lib/nfs/xtab**. Because the **nfs-mountd** service refers to the **xtab** file when deciding access privileges to a file system, changes to the list of exported file systems take effect immediately.

Common exportfs options
The following is a list of commonly-used options available for **exportfs**:

-r

Causes all directories listed in **/etc/exports** to be exported by constructing a new export list in **/etc/lib/nfs/xtab**. This option effectively refreshes the export list with any changes made to **/etc/exports**.

-a

Causes all directories to be exported or unexported, depending on what other options are passed to **exportfs**. If no other options are specified, **exportfs** exports all file systems specified in **/etc/exports**.

-o *file-systems*

Specifies directories to be exported that are not listed in **/etc/exports**. Replace *file-systems* with additional file systems to be exported. These file systems must be formatted in the same way they are specified in **/etc/exports**. This option is often used to test an exported file system before adding it permanently to the list of exported file systems.

-i

Ignores **/etc/exports**; only options given from the command line are used to define exported file systems.

-u

Unexports all shared directories. The command **exportfs -ua** suspends NFS file sharing while keeping all NFS services up. To re-enable NFS sharing, use **exportfs -r**.

-v

Verbose operation, where the file systems being exported or unexported are displayed in greater detail when the **exportfs** command is executed.

If no options are passed to the **exportfs** utility, it displays a list of currently exported file systems.

Additional resources

- For information on different methods for specifying host names, see Section 3.5, "NFS host name formats".
- For a complete list of export options, see the **exports(5)** man page.
- For more information about the **exportfs** utility, see the **exportfs(8)** man page.

3.7. NFS AND RPCBIND

This section explains the purpose of the **rpcbind** service, which is required by NFSv3.

The **rpcbind** service maps Remote Procedure Call (RPC) services to the ports on which they listen. RPC processes notify **rpcbind** when they start, registering the ports they are listening on and the RPC program numbers they expect to serve. The client system then contacts **rpcbind** on the server with a particular RPC program number. The **rpcbind** service redirects the client to the proper port number so it can communicate with the requested service.

Because RPC-based services rely on **rpcbind** to make all connections with incoming client requests, **rpcbind** must be available before any of these services start.

Access control rules for **rpcbind** affect all RPC-based services. Alternatively, it is possible to specify access control rules for each of the NFS RPC daemons.

Additional resources

- For the precise syntax of access control rules, see the **rpc.mountd(8)** and **rpc.statd(8)** man pages.

3.8. INSTALLING NFS

This procedure installs all packages necessary to mount or export NFS shares.

Procedure

- Install the **nfs-utils** package:

```
# yum install nfs-utils
```

3.9. STARTING THE NFS SERVER

This procedure describes how to start the NFS server, which is required to export NFS shares.

Prerequisites

- For servers that support NFSv2 or NFSv3 connections, the **rpcbind** service must be running. To verify that **rpcbind** is active, use the following command:

```
$ systemctl status rpcbind
```

If the service is stopped, start and enable it:

```
$ systemctl enable --now rpcbind
```

Procedure

- To start the NFS server and enable it to start automatically at boot, use the following command:

```
# systemctl enable --now nfs-server
```

Additional resources

- To configure an NFSv4-only server, which does not require **rpcbind**, see Section 3.14, "Configuring an NFSv4-only server".

3.10. TROUBLESHOOTING NFS AND RPCBIND

Because the **rpcbind** service provides coordination between RPC services and the port numbers used to communicate with them, it is useful to view the status of current RPC services using **rpcbind** when troubleshooting. The **rpcinfo** utility shows each RPC-based service with port numbers, an RPC program number, a version number, and an IP protocol type (TCP or UDP).

Procedure

1. To make sure the proper NFS RPC-based services are enabled for **rpcbind**, use the following command:

```
# rpcinfo -p
```

Example 3.2. rpcinfo -p command output

The following is sample output from this command:

```
program vers proto   port  service
 100000    4   tcp    111  portmapper
 100000    3   tcp    111  portmapper
 100000    2   tcp    111  portmapper
 100000    4   udp    111  portmapper
 100000    3   udp    111  portmapper
 100000    2   udp    111  portmapper
 100005    1   udp  20048  mountd
 100005    1   tcp  20048  mountd
 100005    2   udp  20048  mountd
 100005    2   tcp  20048  mountd
 100005    3   udp  20048  mountd
 100005    3   tcp  20048  mountd
 100024    1   udp  37769  status
```

```
100024    1   tcp  49349  status
100003    3   tcp   2049  nfs
100003    4   tcp   2049  nfs
100227    3   tcp   2049  nfs_acl
100021    1   udp  56691  nlockmgr
100021    3   udp  56691  nlockmgr
100021    4   udp  56691  nlockmgr
100021    1   tcp  46193  nlockmgr
100021    3   tcp  46193  nlockmgr
100021    4   tcp  46193  nlockmgr
```

If one of the NFS services does not start up correctly, **rpcbind** will be unable to map RPC requests from clients for that service to the correct port.

2. In many cases, if NFS is not present in **rpcinfo** output, restarting NFS causes the service to correctly register with **rpcbind** and begin working:

```
# systemctl restart nfs-server
```

Additional resources

- For more information and a list of **rpcinfo** options, see the **rpcinfo(8)** man page.
- To configure an NFSv4-only server, which does not require **rpcbind**, see Section 3.14, “Configuring an NFSv4-only server”.

3.11. CONFIGURING THE NFS SERVER TO RUN BEHIND A FIREWALL

NFS requires the **rpcbind** service, which dynamically assigns ports for RPC services and can cause issues for configuring firewall rules. This procedure describes how to configure the NFS server to work behind a firewall.

Procedure

1. To allow clients to access NFS shares behind a firewall, set which ports the RPC services run on in the **[mountd]** section of the **/etc/nfs.conf** file:

```
[mountd]

port=port-number
```

This adds the **-p *port-number*** option to the **rpc.mount** command line: **rpc.mount -p *port-number***.

2. To allow NFSv4.0 callbacks to pass through firewalls, set **/proc/sys/fs/nfs/nfs_callback_tcpport** and allow the server to connect to that port on the client.
This step is not needed for NFSv4.1 or higher, and the other ports for **mountd**, **statd**, and **lockd** are not required in a pure NFSv4 environment.

3. To specify the ports to be used by the RPC service **nlockmgr**, set the port number for the **nlm_tcpport** and **nlm_udpport** options in the **/etc/modprobe.d/lockd.conf** file.

4. Restart the NFS server:

```
# systemctl restart nfs-server
```

If NFS fails to start, check **/var/log/messages**. Commonly, NFS fails to start if you specify a port number that is already in use.

5. Confirm the changes have taken effect:

```
# rpcinfo -p
```

Additional resources

- To configure an NFSv4-only server, which does not require **rpcbind**, see Section 3.14, "Configuring an NFSv4-only server".

3.12. EXPORTING RPC QUOTA THROUGH A FIREWALL

If you export a file system that uses disk quotas, you can use the quota Remote Procedure Call (RPC) service to provide disk quota data to NFS clients.

Procedure

1. Enable and start the **rpc-rquotad** service:

```
# systemctl enable --now rpc-rquotad
```

NOTE

The **rpc-rquotad** service is, if enabled, started automatically after starting the nfs-server service.

2. To make the quota RPC service accessible behind a firewall, the TCP (or UDP, if UDP is enabled) port 875 need to be open. The default port number is defined in the **/etc/services** file. You can override the default port number by appending **-p port-number** to the **RPCRQUOTADOPTS** variable in the **/etc/sysconfig/rpc-rquotad** file.

3. By default, remote hosts can only read quotas. If you want to allow clients to set quotas, append the **-S** option to the **RPCRQUOTADOPTS** variable in the **/etc/sysconfig/rpc-rquotad** file.

4. Restart **rpc-rquotad** for the changes in the **/etc/sysconfig/rpc-rquotad** file to take effect:

```
# systemctl restart rpc-rquotad
```

3.13. ENABLING NFS OVER RDMA (NFSORDMA)

The remote direct memory access (RDMA) service works automatically in Red Hat Enterprise Linux 8 if there is RDMA-capable hardware present.

Procedure

1. Install the **rdma** and **rdma-core** packages:

```
# yum install rdma rdma-core
```

2. To enable automatic loading of NFSoRDMA *server* modules, add the **SVCRDMA_LOAD=yes** option on a new line in the **/etc/rdma/rdma.conf** configuration file.
   The **rdma=20049** option in the **[nfsd]** section of the **/etc/nfs.conf** file specifies the port number on which the NFSoRDMA service listens for clients. The RFC 5667 standard specifies that servers must listen on port **20049** when providing NFSv4 services over RDMA.

   The **/etc/rdma/rdma.conf** file contains a line that sets the **XPRTRDMA_LOAD=yes** option by default, which requests the **rdma** service to load the NFSoRDMA *client* module.

3. Restart the **nfs-server** service:

```
# systemctl restart nfs-server
```

Additional resources

- The RFC 5667 standard: https://tools.ietf.org/html/rfc5667.

3.14. CONFIGURING AN NFSV4-ONLY SERVER

As an NFS server administrator, you can configure the NFS server to support only NFSv4, which minimizes the number of open ports and running services on the system.

3.14.1. Benefits and drawbacks of an NFSv4-only server

This section explains the benefits and drawbacks of configuring the NFS server to only support NFSv4.

By default, the NFS server supports NFSv2, NFSv3, and NFSv4 connections in Red Hat Enterprise Linux 8. However, you can also configure NFS to support only NFS version 4.0 and later. This minimizes the number of open ports and running services on the system, because NFSv4 does not require the **rpcbind** service to listen on the network.

When your NFS server is configured as NFSv4-only, clients attempting to mount shares using NFSv2 or NFSv3 fail with an error like the following:

```
Requested NFS version or transport protocol is not supported.
```

Optionally, you can also disable listening for the **RPCBIND**, **MOUNT**, and **NSM** protocol calls, which are not necessary in the NFSv4-only case.

The effects of disabling these additional options are:

- Clients that attempt to mount shares from your server using NFSv2 or NFSv3 become unresponsive.
- The NFS server itself is unable to mount NFSv2 and NFSv3 file systems.

3.14.2. NFS and rpcbind

This section explains the purpose of the **rpcbind** service, which is required by NFSv3.

The **rpcbind** service maps Remote Procedure Call (RPC) services to the ports on which they listen. RPC processes notify **rpcbind** when they start, registering the ports they are listening on and the RPC program numbers they expect to serve. The client system then contacts **rpcbind** on the server with a particular RPC program number. The **rpcbind** service redirects the client to the proper port number so it can communicate with the requested service.

Because RPC-based services rely on **rpcbind** to make all connections with incoming client requests, **rpcbind** must be available before any of these services start.

Access control rules for **rpcbind** affect all RPC-based services. Alternatively, it is possible to specify access control rules for each of the NFS RPC daemons.

Additional resources

- For the precise syntax of access control rules, see the **rpc.mountd(8)** and **rpc.statd(8)** man pages.

3.14.3. Configuring the NFS server to support only NFSv4

This procedure describes how to configure your NFS server to support only NFS version 4.0 and later.

Procedure

1. Disable NFSv2 and NFSv3 by adding the following lines to the **[nfsd]** section of the **/etc/nfs.conf** configuration file:

```
[nfsd]

vers2=no
vers3=no
```

2. Optionally, disable listening for the **RPCBIND**, **MOUNT**, and **NSM** protocol calls, which are not necessary in the NFSv4-only case. Disable related services:

```
# systemctl mask --now rpc-statd.service rpcbind.service rpcbind.socket
```

3. Restart the NFS server:

```
# systemctl restart nfs-server
```

The changes take effect as soon as you start or restart the NFS server.

3.14.4. Verifying the NFSv4-only configuration

This procedure describes how to verify that your NFS server is configured in the NFSv4-only mode by using the **netstat** utility.

Procedure

- Use the **netstat** utility to list services listening on the TCP and UDP protocols:

```
# netstat --listening --tcp --udp
```

Example 3.3. Output on an NFSv4-only server

The following is an example **netstat** output on an NFSv4-only server; listening for **RPCBIND**, **MOUNT**, and **NSM** is also disabled. Here, **nfs** is the only listening NFS service:

```
# netstat --listening --tcp --udp
```

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:ssh             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:nfs             0.0.0.0:*               LISTEN
tcp6       0      0 [::]:ssh                [::]:*                  LISTEN
tcp6       0      0 [::]:nfs                [::]:*                  LISTEN
udp        0      0 localhost.locald:bootpc 0.0.0.0:*
```

Example 3.4. Output before configuring an NFSv4-only server

In comparison, the **netstat** output before configuring an NFSv4-only server includes the **sunrpc** and **mountd** services:

```
# netstat --listening --tcp --udp

Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address State
tcp        0      0 0.0.0.0:ssh             0.0.0.0:*       LISTEN
tcp        0      0 0.0.0.0:40189           0.0.0.0:*       LISTEN
tcp        0      0 0.0.0.0:46813           0.0.0.0:*       LISTEN
tcp        0      0 0.0.0.0:nfs             0.0.0.0:*       LISTEN
tcp        0      0 0.0.0.0:sunrpc          0.0.0.0:*       LISTEN
tcp        0      0 0.0.0.0:mountd          0.0.0.0:*       LISTEN
tcp6       0      0 [::]:ssh                [::]:*          LISTEN
tcp6       0      0 [::]:51227              [::]:*          LISTEN
tcp6       0      0 [::]:nfs                [::]:*          LISTEN
tcp6       0      0 [::]:sunrpc             [::]:*          LISTEN
tcp6       0      0 [::]:mountd             [::]:*          LISTEN
tcp6       0      0 [::]:45043              [::]:*          LISTEN
udp        0      0 localhost:1018          0.0.0.0:*
udp        0      0 localhost.locald:bootpc 0.0.0.0:*
udp        0      0 0.0.0.0:mountd          0.0.0.0:*
udp        0      0 0.0.0.0:46672           0.0.0.0:*
udp        0      0 0.0.0.0:sunrpc          0.0.0.0:*
udp        0      0 0.0.0.0:33494           0.0.0.0:*
udp6       0      0 [::]:33734              [::]:*
udp6       0      0 [::]:mountd             [::]:*
udp6       0      0 [::]:sunrpc             [::]:*
udp6       0      0 [::]:40243              [::]:*
```

3.15. RELATED INFORMATION

- The Linux NFS wiki: https://linux-nfs.org

CHAPTER 4. SECURING NFS

To minimize NFS security risks and protect data on the server, consider the following sections when exporting NFS file systems on a server or mounting them on a client.

4.1. NFS SECURITY WITH AUTH_SYS AND EXPORT CONTROLS

NFS provides the following traditional options in order to control access to exported files:

- The server restricts which hosts are allowed to mount which file systems either by IP address or by host name.
- The server enforces file system permissions for users on NFS clients in the same way it does for local users. Traditionally, NFS does this using the **AUTH_SYS** call message (also called **AUTH_UNIX**), which relies on the client to state the UID and GIDs of the user. Be aware that this means that a malicious or misconfigured client might easily get this wrong and allow a user access to files that it should not.

To limit the potential risks, administrators often limits the access to read-only or squash user permissions to a common user and group ID. Unfortunately, these solutions prevent the NFS share from being used in the way it was originally intended.

Additionally, if an attacker gains control of the DNS server used by the system exporting the NFS file system, they can point the system associated with a particular hostname or fully qualified domain name to an unauthorized machine. At this point, the unauthorized machine *is* the system permitted to mount the NFS share, because no username or password information is exchanged to provide additional security for the NFS mount.

Wildcards should be used sparingly when exporting directories through NFS, as it is possible for the scope of the wildcard to encompass more systems than intended.

Additional resources

- For more information on securing NFS and **rpcbind**, see the **iptables** man page.

4.2. NFS SECURITY WITH AUTH_GSS

All version of NFS support RPCSEC_GSS and the Kerberos mechanism.

Unlike AUTH_SYS, with the RPCSEC_GSS Kerberos mechanism, the server does not depend on the client to correctly represent which user is accessing the file. Instead, cryptography is used to authenticate users to the server, which prevents a malicious client from impersonating a user without having that user's Kerberos credentials. Using the RPCSEC_GSS Kerberos mechanism is the most straightforward way to secure mounts because after configuring Kerberos, no additional setup is needed.

4.3. CONFIGURING AN NFS SERVER AND CLIENT TO USE KERBEROS

Kerberos is a network authentication system that allows clients and servers to authenticate to each other by using symmetric encryption and a trusted third party, the KDC. Red Hat recommends using Identity Management (IdM) for setting up Kerberos.

Prerequisites

- The Kerberos Key Distribution Centre (**KDC**) is installed and configured.

Procedure

1. - Create the **nfs/hostname.*domain@REALM*** principal on the NFS server side.
   - Create the **host/hostname.*domain@REALM*** principal on both the server and the client side.
   - Add the corresponding keys to keytabs for the client and server.
2. On the server side, use the **sec=** option to enable the wanted security flavors. To enable all security flavors as well as non-cryptographic mounts:

   ```
   /export *(sec=sys:krb5:krb5i:krb5p)
   ```

   Valid security flavors to use with the **sec=** option are:

   - **sys**: no cryptographic protection, the default
   - **krb5**: authentication only
   - **krb5i**: integrity protection
   - **krb5p**: privacy protection
3. On the client side, add **sec=krb5** (or **sec=krb5i**, or **sec=krb5p**, depending on the setup) to the mount options:

   ```
   # mount -o sec=krb5 server:/export /mnt
   ```

Additional resources

- If you need to write files as root on the Kerberos-secured NFS share and keep root ownership on these files, see https://access.redhat.com/articles/4040141. Note that this configuration is not recommended.
- For more information on NFS configuration, see the **exports**(5) and **nfs**(5) man pages.
- For further information on the **RPCSEC_GSS** framework, including how **gssproxy** and **rpc.gssd** inter-operate, see the GSSD flow description.

4.4. NFSV4 SECURITY OPTIONS

NFSv4 includes ACL support based on the Microsoft Windows NT model, not the POSIX model, because of the Microsoft Windows NT model's features and wide deployment.

Another important security feature of NFSv4 is the removal of the use of the **MOUNT** protocol for mounting file systems. The **MOUNT** protocol presented a security risk because of the way the protocol processed file handles.

4.5. FILE PERMISSIONS ON MOUNTED NFS EXPORTS

Once the NFS file system is mounted as either read or read and write by a remote host, the only protection each shared file has is its permissions. If two users that share the same user ID value mount the same NFS file system on different client systems, they can modify each others' files. Additionally,

anyone logged in as root on the client system can use the **su -** command to access any files with the NFS share.

By default, access control lists (ACLs) are supported by NFS under Red Hat Enterprise Linux. Red Hat recommends to keep this feature enabled.

By default, NFS uses *root squashing* when exporting a file system. This sets the user ID of anyone accessing the NFS share as the root user on their local machine to **nobody**. Root squashing is controlled by the default option **root_squash**; for more information about this option, see Section 3.6, "NFS server configuration".

When exporting an NFS share as read-only, consider using the **all_squash** option. This option makes every user accessing the exported file system take the user ID of the **nfsnobody** user.

CHAPTER 5. ENABLING PNFS SCSI LAYOUTS IN NFS

You can configure the NFS server and client to use the pNFS SCSI layout for accessing data. pNFS SCSI is beneficial in use cases that involve longer-duration single-client access to a file.

5.1. PREREQUISITES

- Both the client and the server must be able to send SCSI commands to the same block device. That is, the block device must be on a shared SCSI bus.
- The block device must contain an XFS file system.
- The SCSI device must support SCSI Persistent Reservations as described in the SCSI-3 Primary Commands specification.

5.2. THE PNFS TECHNOLOGY

The pNFS architecture improves the scalability of NFS. When a server implements pNFS, the client is able to access data through multiple servers concurrently. This can lead to performance improvements.

pNFS supports the following storage protocols or layouts on RHEL:

- Files
- Flexfiles
- SCSI

5.3. PNFS SCSI LAYOUTS

The SCSI layout builds on the work of pNFS block layouts. The layout is defined across SCSI devices. It contains a sequential series of fixed-size blocks as logical units (LUs) that must be capable of supporting SCSI persistent reservations. The LU devices are identified by their SCSI device identification.

pNFS SCSI performs well in use cases that involve longer-duration single-client access to a file. An example might be a mail server or a virtual machine housing a cluster.

Operations between the client and the server

When an NFS client reads from a file or writes to it, the client performs a **LAYOUTGET** operation. The server responds with the location of the file on the SCSI device. The client might need to perform an additional operation of **GETDEVICEINFO** to determine which SCSI device to use. If these operations work correctly, the client can issue I/O requests directly to the SCSI device instead of sending **READ** and **WRITE** operations to the server.

Errors or contention between clients might cause the server to recall layouts or not issue them to the clients. In those cases, the clients fall back to issuing **READ** and **WRITE** operations to the server instead of sending I/O requests directly to the SCSI device.

To monitor the operations, see Section 5.8, “Monitoring pNFS SCSI layouts functionality”.

Device reservations

pNFS SCSI handles fencing through the assignment of reservations. Before the server issues layouts to clients, it reserves the SCSI device to ensure that only registered clients may access the device. If a client can issue commands to that SCSI device but is not registered with the device, many operations

from the client on that device fail. For example, the **blkid** command on the client fails to show the UUID of the XFS file system if the server has not given a layout for that device to the client.

The server does not remove its own persistent reservation. This protects the data within the file system on the device across restarts of clients and servers. In order to repurpose the SCSI device, you might need to manually remove the persistent reservation on the NFS server.

5.4. CHECKING FOR A SCSI DEVICE COMPATIBLE WITH PNFS

This procedure checks if a SCSI device supports the pNFS SCSI layout.

Prerequisites

- Install the **sg3_utils** package:

```
# yum install sg3_utils
```

Procedure

- On both the server and client, check for the proper SCSI device support:

```
# sg_persist --in --report-capabilities --verbose path-to-scsi-device
```

Ensure that the *Persist Through Power Loss Active* (**PTPL_A**) bit is set.

Example 5.1. A SCSI device that supports pNFS SCSI

The following is an example of **sg_persist** output for a SCSI device that supports pNFS SCSI. The **PTPL_A** bit reports **1**.

```
  inquiry cdb: 12 00 00 00 24 00
  Persistent Reservation In cmd: 5e 02 00 00 00 00 00 20 00 00
LIO-ORG   block11          4.0
  Peripheral device type: disk
Report capabilities response:
  Compatible Reservation Handling(CRH): 1
  Specify Initiator Ports Capable(SIP_C): 1
  All Target Ports Capable(ATP_C): 1
  Persist Through Power Loss Capable(PTPL_C): 1
  Type Mask Valid(TMV): 1
  Allow Commands: 1
  Persist Through Power Loss Active(PTPL_A): 1
    Support indicated in Type mask:
      Write Exclusive, all registrants: 1
      Exclusive Access, registrants only: 1
      Write Exclusive, registrants only: 1
      Exclusive Access: 1
      Write Exclusive: 1
      Exclusive Access, all registrants: 1
```

Additional resources

- The **sg_persist(8)** man page

5.5. SETTING UP PNFS SCSI ON THE SERVER

This procedure configures an NFS server to export a pNFS SCSI layout.

Procedure

1. On the server, mount the XFS file system created on the SCSI device.
2. Configure the NFS server to export NFS version 4.1 or higher. Set the following option in the **[nfsd]** section of the **/etc/nfs.conf** file:

```
[nfsd]

vers4.1=y
```

3. Configure the NFS server to export the XFS file system over NFS with the **pnfs** option:

Example 5.2. An entry in /etc/exports to export pNFS SCSI

The following entry in the **/etc/exports** configuration file exports the file system mounted at **/exported/directory/** to the **allowed.example.com** client as a pNFS SCSI layout:

```
/exported/directory allowed.example.com(pnfs)
```

Additional resources

- For more information on configuring an NFS server, see Chapter 3, *Exporting NFS shares*.

5.6. SETTING UP PNFS SCSI ON THE CLIENT

This procedure configures an NFS client to mount a pNFS SCSI layout.

Prerequisites

- The NFS server is configured to export an XFS file system over pNFS SCSI. See Section 5.5, "Setting up pNFS SCSI on the server".

Procedure

- On the client, mount the exported XFS file system using NFS version 4.1 or higher:

```
# mount -t nfs -o nfsvers=4.1 host:/remote/export /local/directory
```

Do not mount the XFS file system directly without NFS.

Additional resources

- For more information on mounting NFS shares, see Chapter 2, *Mounting NFS shares*.

5.7. RELEASING THE PNFS SCSI RESERVATION ON THE SERVER

This procedure releases the persistent reservation that an NFS server holds on a SCSI device. This enables you to repurpose the SCSI device when you no longer need to export pNFS SCSI.

You must remove the reservation from the server. It cannot be removed from a different IT Nexus.

Prerequisites

- Install the **sg3_utils** package:

```
# yum install sg3_utils
```

Procedure

1. Query an existing reservation on the server:

```
# sg_persist --read-reservation path-to-scsi-device
```

Example 5.3. Querying a reservation on /dev/sda

```
# sg_persist --read-reservation /dev/sda

  LIO-ORG   block_1           4.0
  Peripheral device type: disk
  PR generation=0x8, Reservation follows:
    Key=0x100000000000000
    scope: LU_SCOPE,  type: Exclusive Access, registrants only
```

2. Remove the existing registration on the server:

```
# sg_persist --out \
            --release \
            --param-rk=reservation-key \
            --prout-type=6 \
            path-to-scsi-device
```

Example 5.4. Removing a reservation on /dev/sda

```
# sg_persist --out \
              --release \
              --param-rk=0x100000000000000 \
              --prout-type=6 \
              /dev/sda

  LIO-ORG   block_1           4.0
  Peripheral device type: disk
```

Additional resources

- The **sg_persist(8)** man page

5.8. MONITORING PNFS SCSI LAYOUTS FUNCTIONALITY

You can monitor if the pNFS client and server exchange proper pNFS SCSI operations or if they fall back on regular NFS operations.

5.8.1. Prerequisites

- A pNFS SCSI client and server are configured.

5.8.2. Checking pNFS SCSI operations from the server using nfsstat

This procedure uses the **nfsstat** utility to monitor pNFS SCSI operations from the server.

Procedure

1. Monitor the operations serviced from the server:

```
# watch --differences \
      "nfsstat --server | egrep --after-context=1 read\|write\|layout"

Every 2.0s: nfsstat --server | egrep --after-context=1 read\|write\|layout

putrootfh    read         readdir      readlink     remove  rename
2         0% 0         0% 1         0% 0         0% 0         0% 0         0%
--
setcltidconf verify   write        rellockowner bc_ctl  bind_conn
0         0% 0         0% 0         0% 0         0% 0         0% 0         0%
--
getdevlist   layoutcommit layoutget    layoutreturn secinfononam sequence
0         0% 29        1% 49        1% 5         0% 0         0% 2435     86%
```

2. The client and server use pNFS SCSI operations when:

   - The **layoutget**, **layoutreturn**, and **layoutcommit** counters increment. This means that the server is serving layouts.
   - The server **read** and **write** counters do not increment. This means that the clients are performing I/O requests directly to the SCSI devices.

5.8.3. Checking pNFS SCSI operations from the client using mountstats

This procedure uses the **/proc/self/mountstats** file to monitor pNFS SCSI operations from the client.

Procedure

1. List the per-mount operation counters:

```
# cat /proc/self/mountstats \
     | awk /scsi_lun_0/,/^$/ \
     | egrep device\|READ\|WRITE\|LAYOUT

device 192.168.122.73:/exports/scsi_lun_0 mounted on /mnt/rhel7/scsi_lun_0 with fstype
nfs4 statvers=1.1
   nfsv4:
bm0=0xfdffbfff,bm1=0x40f9be3e,bm2=0x803,acl=0x3,sessions,pnfs=LAYOUT_SCSI
          READ: 0 0 0 0 0 0 0 0
         WRITE: 0 0 0 0 0 0 0 0
      READLINK: 0 0 0 0 0 0 0 0
       READDIR: 0 0 0 0 0 0 0 0
     LAYOUTGET: 49 49 0 11172 9604 2 19448 19454
```

```
LAYOUTCOMMIT: 28 28 0 7776 4808 0 24719 24722
LAYOUTRETURN: 0 0 0 0 0 0 0 0
 LAYOUTSTATS: 0 0 0 0 0 0 0 0
```

2. In the results:

   - The **LAYOUT** statistics indicate requests where the client and server use pNFS SCSI operations.
   - The **READ** and **WRITE** statistics indicate requests where the client and server fall back to NFS operations.

CHAPTER 6. MOUNTING AN SMB SHARE ON RED HAT ENTERPRISE LINUX

The Server Message Block (SMB) protocol implements an application-layer network protocol used to access resources on a server, such as file shares and shared printers.

NOTE

In the context of SMB, you can find mentions about the Common Internet File System (CIFS) protocol, which is a dialect of SMB. Both the SMB and CIFS protocol are supported, and the kernel module and utilities involved in mounting SMB and CIFS shares both use the name **cifs**.

This section describes how to mount shares from an SMB server. For details about setting up an SMB server on Red Hat Enterprise Linux using Samba, see the section about using Samba in the Configuring and deploying different types of servers guide.

6.1. PREREQUISITES

On Microsoft Windows, SMB is implemented by default. On Red Hat Enterprise Linux, the **cifs.ko** file system module of the kernel provides support for mounting SMB shares. Therefor install the **cifs-utils** package:

```
# yum install cifs-utils
```

The **cifs-utils** package provides utilities to:

- Mount SMB and CIFS shares
- Manage NT Lan Manager (NTLM) credentials in the kernel's keyring
- Set and display Access Control Lists (ACL) in a security descriptor on SMB and CIFS shares

6.2. SUPPORTED SMB PROTOCOL VERSIONS

The **cifs.ko** kernel module supports the following SMB protocol versions:

- SMB 1
- SMB 2.0
- SMB 2.1
- SMB 3.0

NOTE

Depending on the protocol version, not all SMB features are implemented.

6.3. UNIX EXTENSIONS SUPPORT

Samba uses the **CAP_UNIX** capability bit in the SMB protocol to provide the UNIX extensions feature. These extensions are also supported by the **cifs.ko** kernel module. However, both Samba and the kernel module support UNIX extensions only in the SMB 1 protocol.

To use UNIX extensions:

1. Set the **server min protocol** parameter in the **[global]** section in the **/etc/samba/smb.conf** file to **NT1**. This is the default on Samba servers.
2. Mount the share using the SMB 1 protocol by providing the **-o vers=1.0** option to the mount command. For example:

   ```
   # mount -t cifs -o vers=1.0,username=user_name //server_name/share_name /mnt/
   ```

   By default, the kernel module uses SMB 2 or the highest later protocol version supported by the server. Passing the **-o vers=1.0** option to the **mount** command forces that the kernel module uses the SMB 1 protocol that is required for using UNIX extensions.

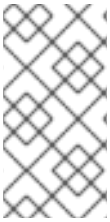
To verify if UNIX extensions are enabled, display the options of the mounted share:

```
# mount
...
//server/share on /mnt type cifs (...,unix,...)
```

If the **unix** entry is displayed in the list of mount options, UNIX extensions are enabled.

6.4. MANUALLY MOUNTING AN SMB SHARE

If you only require an SMB share to be temporary mounted, you can mount it manually using the **mount** utility.



NOTE

Manually mounted shares are not mounted automatically again when you reboot the system. To configure that Red Hat Enterprise Linux automatically mounts the share when the system boots, see Section 6.5, “Mounting an SMB share automatically when the system boots”.

Prerequisites

- The **cifs-utils** package is installed.

Procedure

To manually mount an SMB share, use the **mount** utility with the **-t cifs** parameter:

```
# mount -t cifs -o username=user_name //server_name/share_name /mnt/
Password for user_name@//server_name/share_name:  password
```

In the **-o** parameter, you can specify options that are used to mount the share. For details, see Section 6.8, “Frequently used mount options” and the **OPTIONS** section in the **mount.cifs(8)** man page.

Example 6.1. Mounting a share using an encrypted SMB 3.0 connection

To mount the **server****example**\ share as the ***DOMAIN*****Administrator** user over an encrypted SMB 3.0 connection into the /**mnt**/ directory:

```
# mount -t cifs -o username=DOMAIN\Administrator,seal,vers=3.0 //server/example /mnt/
Password for DOMAIN\Administrator@//server_name/share_name:  password
```

6.5. MOUNTING AN SMB SHARE AUTOMATICALLY WHEN THE SYSTEM BOOTS

If access to a mounted SMB share is permanently required on a server, mount the share automatically at boot time.

Prerequisites

- The **cifs-utils** package is installed.

Procedure

To mount an SMB share automatically when the system boots, add an entry for the share to the **/etc/fstab** file. For example:

```
//server_name/share_name  /mnt  cifs  credentials=/root/smb.cred  0 0
```

IMPORTANT

To enable the system to mount a share automatically, you must store the user name, password, and domain name in a credentials file. For details, see Section 6.6, "Authenticating to an SMB share using a credentials file".

In the fourth field of the row in the **/etc/fstab**, specify mount options, such as the path to the credentials file. For details, see Section 6.8, "Frequently used mount options" and the **OPTIONS** section in the **mount.cifs(8)** man page.

To verify that the share mounts successfully, enter:

```
# mount /mnt/
```

6.6. AUTHENTICATING TO AN SMB SHARE USING A CREDENTIALS FILE

In certain situations, such as when mounting a share automatically at boot time, a share should be mounted without entering the user name and password. To implement this, create a credentials file.

Prerequisites

- The **cifs-utils** package is installed.

Procedure

1. Create a file, such as **/root/smb.cred**, and specify the user name, password, and domain name that file:

```
username=user_name
password=password
domain=domain_name
```

2. Set the permissions to only allow the owner to access the file:

```
# chown user_name /root/smb.cred
# chmod 600 /root/smb.cred
```

You can now pass the **credentials=*file_name*** mount option to the **mount** utility or use it in the **/etc/fstab** file to mount the share without being prompted for the user name and password.

6.7. PERFORMING A MULTI-USER SMB MOUNT

The credentials you provide to mount a share determine the access permissions on the mount point by default. For example, if you use the ***DOMAIN*\example** user when you mount a share, all operations on the share will be executed as this user, regardless which local user performs the operation.

However, in certain situations, the administrator wants to mount a share automatically when the system boots, but users should perform actions on the share's content using their own credentials. The **multiuser** mount options lets you configure this scenario.

IMPORTANT

To use the **multiuser** mount option, you must additionally set the **sec** mount option to a security type that supports providing credentials in a non-interactive way, such as **krb5** or the **ntlmssp** option with a credentials file. For details, see Section 6.7.4, "Accessing a share as a user".

The **root** user mounts the share using the **multiuser** option and an account that has minimal access to the contents of the share. Regular users can then provide their user name and password to the current session's kernel keyring using the **cifscreds** utility. If the user accesses the content of the mounted share, the kernel uses the credentials from the kernel keyring instead of the one initially used to mount the share.

Using this feature consists of the following steps:

- Mount a share with the **multiuser** option.
- Optionally, verify if the share was successfully mounted with the **multiuser** option.
- Access the share as a user .

6.7.1. Prerequisites

- The **cifs-utils** package is installed.

6.7.2. Mounting a share with the multiuser option

Before users can access the share with their own credentials, mount the share as the **root** user using an account with limited permissions.

Procedure

To mount a share automatically with the **multiuser** option when the system boots:

1. Create the entry for the share in the **/etc/fstab** file. For example:

```
//server_name/share_name  /mnt  cifs  multiuser,sec=ntlmssp,credentials=/root/smb.cred
0 0
```

2. Mount the share:

```
# mount /mnt/
```

If you do not want to mount the share automatically when the system boots, mount it manually by passing **-o multiuser,sec=security_type** to the **mount** command. For details about mounting an SMB share manually, see Section 6.4, “Manually mounting an SMB share” .

6.7.3. Verifying if an SMB share is mounted with the multiuser option

To verify if a share is mounted with the **multiuser** option, display the mount options.

Procedure

```
# mount
...
//server_name/share_name on /mnt type cifs (sec=ntlmssp,multiuser,...)
```

If the **multiuser** entry is displayed in the list of mount options, the feature is enabled.

6.7.4. Accessing a share as a user

If an SMB share is mounted with the **multiuser** option, users can provide their credentials for the server to the kernel’s keyring:

```
# cifscreds add -u SMB_user_name server_name
Password: password
```

When the user performs operations in the directory that contains the mounted SMB share, the server applies the file system permissions for this user, instead of the one initially used when the share was mounted.

NOTE

Multiple users can perform operations using their own credentials on the mounted share at the same time.

6.8. FREQUENTLY USED MOUNT OPTIONS

When you mount an SMB share, the mount options determine:

- How the connection will be established with the server. For example, which SMB protocol version is used when connecting to the server.

- How the share will be mounted into the local file system. For example, if the system overrides the remote file and directory permissions to enable multiple local users to access the content on the server.

To set multiple options in the fourth field of the **/etc/fstab** file or in the **-o** parameter of a mount command, separate them with commas. For example, see Section 6.7.2, "Mounting a share with the multiuser option".

The following list gives frequently used mount options:

Option	Description
credentials=*file_name*	Sets the path to the credentials file. See Section 6.6, "Authenticating to an SMB share using a credentials file"
dir_mode=*mode*	Sets the directory mode if the server does not support CIFS UNIX extensions.
file_mode=*mode*	Sets the file mode if the server does not support CIFS UNIX extensions.
password=*password*	Sets the password used to authenticate to the SMB server. Alternatively, specify a credentials file using the **credentials** option.
seal	Enables encryption support for connections using SMB 3.0 or a later protocol version. Therefore, use **seal** together with the **vers** mount option set to **3.0** or later. See Example 6.1, "Mounting a share using an encrypted SMB 3.0 connection".
sec=*security_mode*	Sets the security mode, such as **ntlmsspi**, to enable NTLMv2 password hashing and enabled packet signing. For a list of supported values, see the option's description in the **mount.cifs(8)** man page. If the server does not support the **ntlmv2** security mode, use **sec=ntlmssp**, which is the default. For security reasons, do not use the insecure **ntlm** security mode.
username=*user_name*	Sets the user name used to authenticate to the SMB server. Alternatively, specify a credentials file using the **credentials** option.
vers=*SMB_protocol_version*	Sets the SMB protocol version used for the communication with the server.

For a complete list, see the **OPTIONS** section in the **mount.cifs(8)** man page.

CHAPTER 7. OVERVIEW OF PERSISTENT NAMING ATTRIBUTES

As a system administrator, you need to refer to storage volumes using persistent naming attributes to build storage setups that are reliable over multiple system boots.

7.1. DISADVANTAGES OF NON-PERSISTENT NAMING ATTRIBUTES

Red Hat Enterprise Linux provides a number of ways to identify storage devices. It is important to use the correct option to identify each device when used in order to avoid inadvertently accessing the wrong device, particularly when installing to or reformatting drives.

Traditionally, non-persistent names in the form of /**dev**/**sd***(major number)(minor number)* are used on Linux to refer to storage devices. The major and minor number range and associated **sd** names are allocated for each device when it is detected. This means that the association between the major and minor number range and associated **sd** names can change if the order of device detection changes.

Such a change in the ordering might occur in the following situations:

- The parallelization of the system boot process detects storage devices in a different order with each system boot.
- A disk fails to power up or respond to the SCSI controller. This results in it not being detected by the normal device probe. The disk is not accessible to the system and subsequent devices will have their major and minor number range, including the associated **sd** names shifted down. For example, if a disk normally referred to as **sdb** is not detected, a disk that is normally referred to as **sdc** would instead appear as **sdb**.
- A SCSI controller (host bus adapter, or HBA) fails to initialize, causing all disks connected to that HBA to not be detected. Any disks connected to subsequently probed HBAs are assigned different major and minor number ranges, and different associated **sd** names.
- The order of driver initialization changes if different types of HBAs are present in the system. This causes the disks connected to those HBAs to be detected in a different order. This might also occur if HBAs are moved to different PCI slots on the system.
- Disks connected to the system with Fibre Channel, iSCSI, or FCoE adapters might be inaccessible at the time the storage devices are probed, due to a storage array or intervening switch being powered off, for example. This might occur when a system reboots after a power failure, if the storage array takes longer to come online than the system take to boot. Although some Fibre Channel drivers support a mechanism to specify a persistent SCSI target ID to WWPN mapping, this does not cause the major and minor number ranges, and the associated **sd** names to be reserved; it only provides consistent SCSI target ID numbers.

These reasons make it undesirable to use the major and minor number range or the associated **sd** names when referring to devices, such as in the **/etc/fstab** file. There is the possibility that the wrong device will be mounted and data corruption might result.

Occasionally, however, it is still necessary to refer to the **sd** names even when another mechanism is used, such as when errors are reported by a device. This is because the Linux kernel uses **sd** names (and also SCSI host/channel/target/LUN tuples) in kernel messages regarding the device.

7.2. FILE SYSTEM AND DEVICE IDENTIFIERS

This sections explains the difference between persistent attributes identifying file systems and block devices.

File system identifiers

File system identifiers are tied to a particular file system created on a block device. The identifier is also stored as part of the file system. If you copy the file system to a different device, it still carries the same file system identifier. On the other hand, if you rewrite the device, such as by formatting it with the **mkfs** utility, the device loses the attribute.

File system identifiers include:

- Unique identifier (UUID)
- Label

Device identifiers

Device identifiers are tied to a block device: for example, a disk or a partition. If you rewrite the device, such as by formatting it with the **mkfs** utility, the device keeps the attribute, because it is not stored in the file system.

Device identifiers include:

- World Wide Identifier (WWID)
- Partition UUID
- Serial number

Recommendations

- Some file systems, such as logical volumes, span multiple devices. Red Hat recommends accessing these file systems using file system identifiers rather than device identifiers.

7.3. DEVICE NAMES MANAGED BY THE UDEV MECHANISM IN /DEV/DISK/

This section lists different kinds of persistent naming attributes that the **udev** service provides in the /**dev**/**disk**/ directory.

The **udev** mechanism is used for all types of devices in Linux, not just for storage devices. In the case of storage devices, Red Hat Enterprise Linux contains **udev** rules that create symbolic links in the /**dev**/**disk**/ directory. This enables you to refer to storage devices by:

- Their content
- A unique identifier
- Their serial number.

Although **udev** naming attributes are persistent, in that they do not change on their own across system reboots, some are also configurable.

7.3.1. File system identifiers

The UUID attribute in /dev/disk/by-uuid/

Entries in this directory provide a symbolic name that refers to the storage device by a **unique identifier** (UUID) in the content (that is, the data) stored on the device. For example:

```
/dev/disk/by-uuid/3e6be9de-8139-11d1-9106-a43f08d823a6
```

You can use the UUID to refer to the device in the **/etc/fstab** file using the following syntax:

```
UUID=3e6be9de-8139-11d1-9106-a43f08d823a6
```

You can configure the UUID attribute when creating a file system, and you can also change it later on.

The Label attribute in /dev/disk/by-label/

Entries in this directory provide a symbolic name that refers to the storage device by a **label** in the content (that is, the data) stored on the device.

For example:

```
/dev/disk/by-label/Boot
```

You can use the label to refer to the device in the **/etc/fstab** file using the following syntax:

```
LABEL=Boot
```

You can configure the Label attribute when creating a file system, and you can also change it later on.

7.3.2. Device identifiers

The WWID attribute in /dev/disk/by-id/

The World Wide Identifier (WWID) is a persistent, **system-independent identifier** that the SCSI Standard requires from all SCSI devices. The WWID identifier is guaranteed to be unique for every storage device, and independent of the path that is used to access the device. The identifier is a property of the device but is not stored in the content (that is, the data) on the devices.

This identifier can be obtained by issuing a SCSI Inquiry to retrieve the Device Identification Vital Product Data (page **0x83**) or Unit Serial Number (page **0x80**).

Red Hat Enterprise Linux automatically maintains the proper mapping from the WWID-based device name to a current **/dev/sd** name on that system. Applications can use the **/dev/disk/by-id/** name to reference the data on the disk, even if the path to the device changes, and even when accessing the device from different systems.

Example 7.1. WWID mappings

WWID symlink	Non-persistent device	Note
/dev/disk/by-id/scsi-3600508b400105e210000900000490000	**/dev/sda**	A device with a page **0x83** identifier
/dev/disk/by-id/scsi-SSEAGATE_ST373453LW_3HW1RHM6	**/dev/sdb**	A device with a page **0x80** identifier

WWID symlink	Non-persistent device	Note
/dev/disk/by-id/ata-SAMSUNG_MZNLN256HMHQ-000L7_S2WDNX0J336519-part3	**/dev/sdc3**	A disk partition

In addition to these persistent names provided by the system, you can also use **udev** rules to implement persistent names of your own, mapped to the WWID of the storage.

The Partition UUID attribute in /dev/disk/by-partuuid

The Partition UUID (PARTUUID) attribute identifies partitions as defined by GPT partition table.

Example 7.2. Partition UUID mappings

PARTUUID symlink	Non-persistent device
/dev/disk/by-partuuid/4cd1448a-01	**/dev/sda1**
/dev/disk/by-partuuid/4cd1448a-02	**/dev/sda2**
/dev/disk/by-partuuid/4cd1448a-03	**/dev/sda3**

The Path attribute in /dev/disk/by-path/

This attribute provides a symbolic name that refers to the storage device by the **hardware path** used to access the device.

WARNING

The Path attribute is unreliable, and Red Hat does not recommend using it.

7.4. THE WORLD WIDE IDENTIFIER WITH DM MULTIPATH

This section describes the mapping between the World Wide Identifier (WWID) and non-persistent device names in a Device Mapper Multipath configuration.

If there are multiple paths from a system to a device, DM Multipath uses the WWID to detect this. DM Multipath then presents a single "pseudo-device" in the **/dev/mapper/wwid** directory, such as **/dev/mapper/3600508b400105df70000e00000ac0000**.

The command **multipath -l** shows the mapping to the non-persistent identifiers:

- ***Host*:*Channel*:*Target*:*LUN***
- /**dev**/**sd** name
- ***major*:*minor*** number

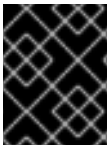
Example 7.3. WWID mappings in a multipath configuration

An example output of the **multipath -l** command:

```
3600508b400105df70000e00000ac0000 dm-2 vendor,product
[size=20G][features=1 queue_if_no_path][hwhandler=0][rw]
\_ round-robin 0 [prio=0][active]
 \_ 5:0:1:1 sdc 8:32  [active][undef]
 \_ 6:0:1:1 sdg 8:96  [active][undef]
\_ round-robin 0 [prio=0][enabled]
 \_ 5:0:0:1 sdb 8:16  [active][undef]
 \_ 6:0:0:1 sdf 8:80  [active][undef]
```

DM Multipath automatically maintains the proper mapping of each WWID-based device name to its corresponding /**dev**/**sd** name on the system. These names are persistent across path changes, and they are consistent when accessing the device from different systems.

When the **user_friendly_names** feature of DM Multipath is used, the WWID is mapped to a name of the form /**dev**/**mapper**/**mpath*N***. By default, this mapping is maintained in the file /**etc**/**multipath**/**bindings**. These **mpath*N*** names are persistent as long as that file is maintained.



IMPORTANT

If you use **user_friendly_names**, then additional steps are required to obtain consistent names in a cluster.

7.5. LIMITATIONS OF THE UDEV DEVICE NAMING CONVENTION

The following are some limitations of the **udev** naming convention:

- It is possible that the device might not be accessible at the time the query is performed because the **udev** mechanism might rely on the ability to query the storage device when the **udev** rules are processed for a **udev** event. This is more likely to occur with Fibre Channel, iSCSI or FCoE storage devices when the device is not located in the server chassis.
- The kernel might send **udev** events at any time, causing the rules to be processed and possibly causing the /**dev**/**disk**/**by-***/ links to be removed if the device is not accessible.
- There might be a delay between when the **udev** event is generated and when it is processed, such as when a large number of devices are detected and the user-space **udevd** service takes some amount of time to process the rules for each one. This might cause a delay between when the kernel detects the device and when the /**dev**/**disk**/**by-***/ names are available.
- External programs such as **blkid** invoked by the rules might open the device for a brief period of time, making the device inaccessible for other uses.

7.6. LISTING PERSISTENT NAMING ATTRIBUTES

This procedure describes how to find out the persistent naming attributes of non-persistent storage devices.

Procedure

- To list the UUID and Label attributes, use the **lsblk** utility:

```
$ lsblk --fs storage-device
```

For example:

Example 7.4. Viewing the UUID and Label of a file system

```
$ lsblk --fs /dev/sda1

NAME FSTYPE LABEL UUID                                 MOUNTPOINT
sda1 xfs    Boot  afa5d5e3-9050-48c3-acc1-bb30095f3dc4 /boot
```

- To list the PARTUUID attribute, use the **lsblk** utility with the **--output +PARTUUID** option:

```
$ lsblk --output +PARTUUID
```

For example:

Example 7.5. Viewing the PARTUUID attribute of a partition

```
$ lsblk --output +PARTUUID /dev/sda1

NAME MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT PARTUUID
sda1   8:1    0  512M  0 part /boot      4cd1448a-01
```

- To list the WWID attribute, examine the targets of symbolic links in the **/dev/disk/by-id/** directory. For example:

Example 7.6. Viewing the WWID of all storage devices on the system

```
$ file /dev/disk/by-id/*

/dev/disk/by-id/ata-QEMU_HARDDISK_QM00001:
symbolic link to ../../sda
/dev/disk/by-id/ata-QEMU_HARDDISK_QM00001-part1:
symbolic link to ../../sda1
/dev/disk/by-id/ata-QEMU_HARDDISK_QM00001-part2:
symbolic link to ../../sda2
/dev/disk/by-id/dm-name-rhel_rhel8-root:                                symbolic link to
../../dm-0
/dev/disk/by-id/dm-name-rhel_rhel8-swap:                                 symbolic link
to ../../dm-1
/dev/disk/by-id/dm-uuid-LVM-
QIWtEHtXGobe5bewlIUDivKOz5ofkgFhP0RMFsNyySVihqEl2cWWbR7MjXJolD6g:
symbolic link to ../../dm-1
/dev/disk/by-id/dm-uuid-LVM-
```

```
QIWtEHtXGobe5bewlIUDivKOz5ofkgFhXqH2M45hD2H9nAf2qfWSrlRLhzfMyOKd:
symbolic link to ../../dm-0
/dev/disk/by-id/lvm-pv-uuid-atlr2Y-vuMo-ueoH-CpMG-4JuH-AhEF-wu4QQm:
symbolic link to ../../sda2
```

7.7. MODIFYING PERSISTENT NAMING ATTRIBUTES

This procedure describes how to change the UUID or Label persistent naming attribute of a file system.

NOTE

Changing **udev** attributes happens in the background and might take a long time. The **udevadm settle** command waits until the change is fully registered, which ensures that your next command will be able to utilize the new attribute correctly.

In the following commands:

- Replace *new-uuid* with the UUID you want to set; for example, **1cdfbc07-1c90-4984-b5ec-f61943f5ea50**. You can generate a UUID using the **uuidgen** command.
- Replace *new-label* with a label; for example, **backup_data**.

Prerequisites

- If you are modifying the attributes of an XFS file system, unmount it first.

Procedure

- To change the UUID or Label attributes of an **XFS** file system, use the **xfs_admin** utility:

```
# xfs_admin -U new-uuid -L new-label storage-device
# udevadm settle
```

- To change the UUID or Label attributes of an **ext4**, **ext3**, or **ext2** file system, use the **tune2fs** utility:

```
# tune2fs -U new-uuid -L new-label storage-device
# udevadm settle
```

- To change the UUID or Label attributes of a swap volume, use the **swaplabel** utility:

```
# swaplabel --uuid new-uuid --label new-label swap-device
# udevadm settle
```

CHAPTER 8. GETTING STARTED WITH PARTITIONS

As a system administrator, you can use the following procedures to create, delete, and modify various types of disk partitions.

For an overview of the advantages and disadvantages to using partitions on block devices, see the following KBase article: https://access.redhat.com/solutions/163853.

8.1. VIEWING THE PARTITION TABLE

As a system administrator, you can display the partition table of a block device to see the partition layout and details about individual partitions.

8.1.1. Viewing the partition table with parted

This procedure describes how to view the partition table on a block device using the **parted** utility.

Procedure

1. Start the interactive **parted** shell:

```
# parted block-device
```

   - Replace *block-device* with the path to the device you want to examine: for example, **/dev/sda**.

2. View the partition table:

```
(parted) print
```

3. Optionally, use the following command to switch to another device you want to examine next:

```
(parted) select block-device
```

Additional resources

- The **parted(8)** man page.

8.1.2. Example output of parted print

This section provides an example output of the **print** command in the **parted** shell and describes fields in the output.

Example 8.1. Output of the print command

```
Model: ATA SAMSUNG MZNLN256 (scsi)
Disk /dev/sda: 256GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type     File system  Flags
 1      1049kB  269MB   268MB   primary  xfs          boot
```

```
2     269MB   34.6GB  34.4GB  primary
3     34.6GB  45.4GB  10.7GB  primary
4     45.4GB  256GB   211GB   extended
5     45.4GB  256GB   211GB   logical
```

Following is a description of the fields:

Model: ATA SAMSUNG MZNLN256 (scsi)

The disk type, manufacturer, model number, and interface.

Disk /dev/sda: 256GB

The file path to the block device and the storage capacity.

Partition Table: msdos

The disk label type.

Number

The partition number. For example, the partition with minor number 1 corresponds to **/dev/sda1**.

Start and **End**

The location on the device where the partition starts and ends.

Type

Valid types are metadata, free, primary, extended, or logical.

File system

The file system type. If the **File system** field of a device shows no value, this means that its file system type is unknown. The **parted** utility cannot recognize the file system on encrypted devices.

Flags

Lists the flags set for the partition. Available flags are **boot**, **root**, **swap**, **hidden**, **raid**, **lvm**, or **lba**.

8.2. CREATING A PARTITION TABLE ON A DISK

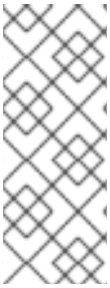
As a system administrator, you can format a block device with different types of partition tables to enable using partitions on the device.

WARNING

Formatting a block device with a partition table deletes all data stored on the device.

8.2.1. Considerations before modifying partitions on a disk

This section lists key points to consider before creating, removing, or resizing partitions.



NOTE

This section does not cover the DASD partition table, which is specific to the IBM Z architecture. For information on DASD, see:

- Configuring a Linux instance on IBM Z
- The What you should know about DASD article at the IBM Knowledge Center

The maximum number of partitions
The number of partitions on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, you can have either:
  - Up to four primary partitions, or
  - Up to three primary partitions, one extended partition, and multiple logical partitions within the extended.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum number of partitions is 128. While the GPT specification allows for more partitions by growing the area reserved for the partition table, common practice used by the **parted** utility is to limit it to enough area for 128 partitions.

The maximum size of a partition
The size of a partition on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, the maximum size is 2TiB.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum size is 8ZiB.

If you want to create a partition larger than 2TiB, the disk must be formatted with GPT.

Size alignment
The **parted** utility enables you to specify partition size using multiple different suffixes:

MiB, GiB, or TiB

Size expressed in powers of 2.

- The starting point of the partition is aligned to the exact sector specified by size.
- The ending point is aligned to the specified size minus 1 sector.

MB, GB, or TB

Size expressed in powers of 10.
The starting and ending point is aligned within one half of the specified unit: for example, ±500KB when using the MB suffix.

8.2.2. Comparison of partition table types

This section compares the properties of different types of partition tables that you can create on a block device.

Table 8.1. Partition table types

Partition table	Maximum number of partitions	Maximum partition size
Master Boot Record (MBR)	4 primary, or 3 primary and 12 logical inside an extended partition	2TiB
GUID Partition Table (GPT)	128	8ZiB

8.2.3. Creating a partition table on a disk with parted

This procedure describes how to format a block device with a partition table using the **parted** utility.

Procedure

1. Start the interactive **parted** shell:

```
# parted block-device
```

   - Replace *block-device* with the path to the device where you want to create a partition table: for example, /**dev**/**sda**.

2. Determine if there already is a partition table on the device:

```
(parted) print
```

   If the device already contains partitions, they will be deleted in the next steps.

3. Create the new partition table:

```
(parted) mklabel table-type
```

   - Replace *table-type* with with the intended partition table type:
     - **msdos** for MBR
     - **gpt** for GPT

   Example 8.2. Creating a GPT table

   For example, to create a GPT table on the disk, use:

```
(parted) mklabel gpt
```

   The changes start taking place as soon as you enter this command, so review it before executing it.

4. View the partition table to confirm that the partition table exists:

```
(parted) print
```

5. Exit the **parted** shell:

```
(parted) quit
```

Additional resources

- The **parted(8)** man page.

Next steps

- Create partitions on the device. See Section 8.3, "Creating a partition" for details.

8.3. CREATING A PARTITION

As a system administrator, you can create new partitions on a disk.

8.3.1. Considerations before modifying partitions on a disk

This section lists key points to consider before creating, removing, or resizing partitions.

NOTE

This section does not cover the DASD partition table, which is specific to the IBM Z architecture. For information on DASD, see:

- Configuring a Linux instance on IBM Z
- The What you should know about DASD article at the IBM Knowledge Center

The maximum number of partitions
The number of partitions on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, you can have either:
  - Up to four primary partitions, or
  - Up to three primary partitions, one extended partition, and multiple logical partitions within the extended.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum number of partitions is 128. While the GPT specification allows for more partitions by growing the area reserved for the partition table, common practice used by the **parted** utility is to limit it to enough area for 128 partitions.

The maximum size of a partition
The size of a partition on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, the maximum size is 2TiB.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum size is 8ZiB.

If you want to create a partition larger than 2TiB, the disk must be formatted with GPT.

Size alignment
The **parted** utility enables you to specify partition size using multiple different suffixes:

MiB, GiB, or TiB

Size expressed in powers of 2.

- The starting point of the partition is aligned to the exact sector specified by size.
- The ending point is aligned to the specified size minus 1 sector.

MB, GB, or TB

Size expressed in powers of 10.
The starting and ending point is aligned within one half of the specified unit: for example, ±500KB when using the MB suffix.

8.3.2. Partition types

This section describes different attributes that specify the type of a partition.

Partition types or flags
The partition type, or flag, is used by a running system only rarely. However, the partition type matters to on-the-fly generators, such as **systemd-gpt-auto-generator**, which use the partition type to, for example, automatically identify and mount devices.

- The **parted** utility provides some control of partition types by mapping the partition type to *flags*. The parted utility can handle only certain partition types: for example LVM, swap, or RAID.
- The **fdisk** utility supports the full range of partition types by specifying hexadecimal codes.

Partition file system type
The **parted** utility optionally accepts a file system type argument when creating a partition. The value is used to:

- Set the partition flags on MBR, or
- Set the partition UUID type on GPT. For example, the **swap**, **fat**, or **hfs** file system types set different GUIDs. The default value is the Linux Data GUID.

The argument does not modify the file system on the partition in any way. It only differentiates between the supported flags or GUIDs.

The following file system types are supported:

- **xfs**
- **ext2**
- **ext3**
- **ext4**
- **fat16**
- **fat32**
- **hfs**
- **hfs+**

- **linux-swap**
- **ntfs**
- **reiserfs**

8.3.3. Creating a partition with parted

This procedure describes how to create a new partition on a block device using the **parted** utility.

Prerequisites

- There is a partition table on the disk. For details on how to format the disk, see Section 8.2, "Creating a partition table on a disk".
- If the partition you want to create is larger than 2TiB, the disk must be formatted with the GUID Partition Table (GPT).

Procedure

1. Start the interactive **parted** shell:

```
# parted block-device
```

   - Replace *block-device* with the path to the device where you want to create a partition: for example, /**dev**/**sda**.

2. View the current partition table to determine if there is enough free space:

```
(parted) print
```

   - If there is not enough free space, you can resize an existing partition. For more information, see Section 8.5, "Resizing a partition" .
   - From the partition table, determine:
     - The start and end points of the new partition
     - On MBR, what partition type it should be.

3. Create the new partition:

```
(parted) mkpart part-type name fs-type start end
```

   - Replace *part-type* with with **primary**, **logical**, or **extended** based on what you decided from the partition table. This applies only to the MBR partition table.
   - Replace *name* with an arbitrary partition name. This is required for GPT partition tables.
   - Replace *fs-type* with any one of **xfs**, **ext2**, **ext3**, **ext4**, **fat16**, **fat32**, **hfs**, **hfs+**, **linux-swap**, **ntfs**, or **reiserfs**. The *fs-type* parameter is optional. Note that **parted** does not create the file system on the partition.
   - Replace *start* and *end* with the sizes that determine the starting and ending points of the partition, counting from the beginning of the disk. You can use size suffixes, such as **512MiB**, **20GiB**, or **1.5TiB**. The default size megabytes.

Example 8.3. Creating a small primary partition

For example, to create a primary partition from 1024MiB until 2048MiB on an MBR table, use:

```
(parted) mkpart primary 1024MiB 2048MiB
```

The changes start taking place as soon as you enter this command, so review it before executing it.

4. View the partition table to confirm that the created partition is in the partition table with the correct partition type, file system type, and size:

```
(parted) print
```

5. Exit the **parted** shell:

```
(parted) quit
```

6. Use the following command to wait for the system to register the new device node:

```
# udevadm settle
```

7. Verify that the kernel recognizes the new partition:

```
# cat /proc/partitions
```

Additional resources

- The **parted(8)** man page.

8.3.4. Setting a partition type with fdisk

This procedure describes how to set a partition type, or flag, using the **fdisk** utility.

Prerequisites

- There is a partition on the disk.

Procedure

1. Start the interactive **fdisk** shell:

```
# fdisk block-device
```

   - Replace *block-device* with the path to the device where you want to set a partition type: for example, /**dev**/**sda**.

2. View the current partition table to determine the minor partition number:

```
Command (m for help): print
```

You can see the current partition type in the **Type** column and its corresponding type ID in the **Id** column.

3. Enter the partition type command and select a partition using its minor number:

```
Command (m for help): type
Partition number (1,2,3 default 3): 2
```

4. Optionally, list the available hexadecimal codes:

```
Hex code (type L to list all codes): L
```

5. Set the partition type:

```
Hex code (type L to list all codes): 8e
```

6. Write your changes and exit the **fdisk** shell:

```
Command (m for help): write
The partition table has been altered.
Syncing disks.
```

7. Verify your changes:

```
# fdisk --list block-device
```

8.4. REMOVING A PARTITION

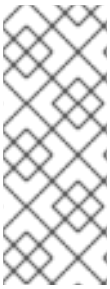
As a system administrator, you can remove a disk partition that is no longer used to free up disk space.

WARNING

Removing a partition deletes all data stored on the partition.

8.4.1. Considerations before modifying partitions on a disk

This section lists key points to consider before creating, removing, or resizing partitions.



NOTE

This section does not cover the DASD partition table, which is specific to the IBM Z architecture. For information on DASD, see:

- Configuring a Linux instance on IBM Z
- The What you should know about DASD article at the IBM Knowledge Center

The maximum number of partitions

The number of partitions on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, you can have either:
  - Up to four primary partitions, or
  - Up to three primary partitions, one extended partition, and multiple logical partitions within the extended.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum number of partitions is 128. While the GPT specification allows for more partitions by growing the area reserved for the partition table, common practice used by the **parted** utility is to limit it to enough area for 128 partitions.

The maximum size of a partition
The size of a partition on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, the maximum size is 2TiB.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum size is 8ZiB.

If you want to create a partition larger than 2TiB, the disk must be formatted with GPT.

Size alignment
The **parted** utility enables you to specify partition size using multiple different suffixes:

MiB, GiB, or TiB

Size expressed in powers of 2.

- The starting point of the partition is aligned to the exact sector specified by size.
- The ending point is aligned to the specified size minus 1 sector.

MB, GB, or TB

Size expressed in powers of 10.
The starting and ending point is aligned within one half of the specified unit: for example, ±500KB when using the MB suffix.

8.4.2. Removing a partition with parted

This procedure describes how to remove a disk partition using the **parted** utility.

Procedure

1. Start the interactive **parted** shell:

```
# parted block-device
```

   - Replace *block-device* with the path to the device where you want to remove a partition: for example, /**dev**/**sda**.

2. View the current partition table to determine the minor number of the partition to remove:

```
(parted) print
```

3. Remove the partition:

```
(parted) rm minor-number
```

   - Replace *minor-number* with the minor number of the partition you want to remove: for example, **3**.

   The changes start taking place as soon as you enter this command, so review it before executing it.

4. Confirm that the partition is removed from the partition table:

```
(parted) print
```

5. Exit the **parted** shell:

```
(parted) quit
```

6. Verify that the kernel knows the partition is removed:

```
# cat /proc/partitions
```

7. Remove the partition from the **/etc/fstab** file if it is present. Find the line that declares the removed partition, and remove it from the file.

8. Regenerate mount units so that your system registers the new **/etc/fstab** configuration:

```
# systemctl daemon-reload
```

9. If you have deleted a swap partition or removed pieces of LVM, remove all references to the partition from the kernel command line in the **/etc/default/grub** file and regenerate GRUB configuration:

   - On a BIOS-based system:

```
# grub2-mkconfig --output=/etc/grub2.cfg
```

   - On a UEFI-based system:

```
# grub2-mkconfig --output=/etc/grub2-efi.cfg
```

10. To register the changes in the early boot system, rebuild the **initramfs** file system:

```
# dracut --force --verbose
```

Additional resources

- The **parted(8)** man page

8.5. RESIZING A PARTITION

As a system administrator, you can extend a partition to utilize unused disk space, or shrink a partition to use its capacity for different purposes.

8.5.1. Considerations before modifying partitions on a disk

This section lists key points to consider before creating, removing, or resizing partitions.

NOTE

This section does not cover the DASD partition table, which is specific to the IBM Z architecture. For information on DASD, see:

- Configuring a Linux instance on IBM Z
- The What you should know about DASD article at the IBM Knowledge Center

The maximum number of partitions
The number of partitions on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, you can have either:
  - Up to four primary partitions, or
  - Up to three primary partitions, one extended partition, and multiple logical partitions within the extended.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum number of partitions is 128. While the GPT specification allows for more partitions by growing the area reserved for the partition table, common practice used by the **parted** utility is to limit it to enough area for 128 partitions.

The maximum size of a partition
The size of a partition on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, the maximum size is 2TiB.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum size is 8ZiB.

If you want to create a partition larger than 2TiB, the disk must be formatted with GPT.

Size alignment
The **parted** utility enables you to specify partition size using multiple different suffixes:

MiB, GiB, or TiB

Size expressed in powers of 2.

- The starting point of the partition is aligned to the exact sector specified by size.
- The ending point is aligned to the specified size minus 1 sector.

MB, GB, or TB

Size expressed in powers of 10.
The starting and ending point is aligned within one half of the specified unit: for example, ±500KB when using the MB suffix.

8.5.2. Resizing a partition with parted

This procedure resizes a disk partition using the **parted** utility.

Prerequisites

- If you want to shrink a partition, back up the data that are stored on it.

WARNING

Shrinking a partition might result in data loss on the partition.

- If you want to resize a partition to be larger than 2TiB, the disk must be formatted with the GUID Partition Table (GPT). For details on how to format the disk, see Section 8.2, "Creating a partition table on a disk".

Procedure

1. If you want to shrink the partition, shrink the file system on it first so that it is not larger than the resized partition. Note that XFS does not support shrinking.
2. Start the interactive **parted** shell:

```
# parted block-device
```

   - Replace *block-device* with the path to the device where you want to resize a partition: for example, /**dev**/**sda**.

3. View the current partition table:

```
(parted) print
```

   From the partition table, determine:

   - The minor number of the partition
   - The location of the existing partition and its new ending point after resizing

4. Resize the partition:

```
(parted) resizepart minor-number new-end
```

   - Replace *minor-number* with the minor number of the partition that you are resizing: for example, **3**.
   - Replace *new-end* with the size that determines the new ending point of the resized partition, counting from the beginning of the disk. You can use size suffixes, such as **512MiB**, **20GiB**, or **1.5TiB**. The default size megabytes.

Example 8.4. Extending a partition

For example, to extend a partition located at the beginning of the disk to be 2GiB in size, use:

```
(parted) resizepart 1 2GiB
```

The changes start taking place as soon as you enter this command, so review it before executing it.

5. View the partition table to confirm that the resized partition is in the partition table with the correct size:

```
(parted) print
```

6. Exit the **parted** shell:

```
(parted) quit
```

7. Verify that the kernel recognizes the new partition:

```
# cat /proc/partitions
```

8. If you extended the partition, extend the file system on it as well. See (reference) for details.

Additional resources

- The **parted(8)** man page.

CHAPTER 9. GETTING STARTED WITH XFS

This is an overview of how to create and maintain XFS file systems.

9.1. THE XFS FILE SYSTEM

XFS is a highly scalable, high-performance, robust, and mature 64-bit journaling file system that supports very large files and file systems on a single host. It is the default file system in Red Hat Enterprise Linux 8. XFS was originally developed in the early 1990s by SGI and has a long history of running on extremely large servers and storage arrays.

The features of XFS include:

Reliability

- Metadata journaling, which ensures file system integrity after a system crash by keeping a record of file system operations that can be replayed when the system is restarted and the file system remounted
- Extensive run-time metadata consistency checking
- Scalable and fast repair utilities
- Quota journaling. This avoids the need for lengthy quota consistency checks after a crash.

Scalability and performance

- Supported file system size up to 1024 TiB
- Ability to support a large number of concurrent operations
- B-tree indexing for scalability of free space management
- Sophisticated metadata read-ahead algorithms
- Optimizations for streaming video workloads

Allocation schemes

- Extent-based allocation
- Stripe-aware allocation policies
- Delayed allocation
- Space pre-allocation
- Dynamically allocated inodes

Other features

- Reflink-based file copies (new in Red Hat Enterprise Linux 8)
- Tightly integrated backup and restore utilities
- Online defragmentation

- Online file system growing
- Comprehensive diagnostics capabilities
- Extended attributes (**xattr**). This allows the system to associate several additional name/value pairs per file.
- Project or directory quotas. This allows quota restrictions over a directory tree.
- Subsecond timestamps

Performance characteristics

XFS has a high performance on large systems with enterprise workloads. A large system is one with a relatively high number of CPUs, multiple HBAs, and connections to external disk arrays. XFS also performs well on smaller systems that have a multi-threaded, parallel I/O workload.

XFS has a relatively low performance for single threaded, metadata-intensive workloads: for example, a workload that creates or deletes large numbers of small files in a single thread.

9.2. CREATING AN XFS FILE SYSTEM

As a system administrator, you can create an XFS file system on a block device to enable it to store files and directories.

9.2.1. Creating an XFS file system with mkfs.xfs

This procedure describes how to create an XFS file system on a block device.

Procedure

1. To create the file system:
   - If the device is a regular partition, an LVM volume, an MD volume, a disk, or a similar device, use the following command:

     ```
     # mkfs.xfs block-device
     ```

     - Replace *block-device* with the path to the block device. For example, **/dev/sdb1**, **/dev/disk/by-uuid/05e99ec8-def1-4a5e-8a9d-5945339ceb2a**, or **/dev/my-volgroup/my-lv**.
     - In general, the default options are optimal for common use.
     - When using **mkfs.xfs** on a block device containing an existing file system, add the **-f** option to overwrite that file system.
   - To create the file system on a hardware RAID device, check if the system correctly detects the stripe geometry of the device:
     - If the stripe geometry information is correct, no additional options are needed. Create the file system:

       ```
       # mkfs.xfs block-device
       ```

- If the information is incorrect, specify stripe geometry manually with the **su** and **sw** parameters of the **-d** option. The **su** parameter specifies the RAID chunk size, and the **sw** parameter specifies the number of data disks in the RAID device.
  For example:

```
# mkfs.xfs -d su=64k,sw=4 /dev/sda3
```

2. Use the following command to wait for the system to register the new device node:

```
# udevadm settle
```

Additional resources

- The **mkfs.xfs(8)** man page.

9.2.2. Additional resources

- The **mkfs.xfs(8)** man page.

9.3. BACKING UP AN XFS FILE SYSTEM

As a system administrator, you can use the **xfsdump** to back up an XFS file system into a file or on a tape. This provides a simple backup mechanism.

9.3.1. Features of XFS backup

This section describes key concepts and features of backing up an XFS file system with the **xfsdump** utility.

You can use the **xfsdump** utility to:

- Perform backups to regular file images.
  Only one backup can be written to a regular file.

- Perform backups to tape drives.
  The **xfsdump** utility also enables you to write multiple backups to the same tape. A backup can span multiple tapes.

  To back up multiple file systems to a single tape device, simply write the backup to a tape that already contains an XFS backup. This appends the new backup to the previous one. By default, **xfsdump** never overwrites existing backups.

- Create incremental backups.
  The **xfsdump** utility uses dump levels to determine a base backup to which other backups are relative. Numbers from 0 to 9 refer to increasing dump levels. An incremental backup only backs up files that have changed since the last dump of a lower level:

  - To perform a full backup, perform a level 0 dump on the file system.

  - A level 1 dump is the first incremental backup after a full backup. The next incremental backup would be level 2, which only backs up files that have changed since the last level 1 dump; and so on, to a maximum of level 9.

- Exclude files from a backup using size, subtree, or inode flags to filter them.

Additional resources

- The **xfsdump(8)** man page.

9.3.2. Backing up an XFS file system with xfsdump

This procedure describes how to back up the content of an XFS file system into a file or a tape.

Prerequisites

- An XFS file system that you can back up.
- Another file system or a tape drive where you can store the backup.

Procedure

- Use the following command to back up an XFS file system:

```
# xfsdump -l level [-L label] \
        -f backup-destination path-to-xfs-filesystem
```

  - Replace *level* with the dump level of your backup. Use **0** to perform a full backup or **1** to **9** to perform consequent incremental backups.
  - Replace *backup-destination* with the path where you want to store your backup. The destination can be a regular file, a tape drive, or a remote tape device. For example, **/backup-files/Data.xfsdump** for a file or **/dev/st0** for a tape drive.
  - Replace *path-to-xfs-filesystem* with the mount point of the XFS file system you want to back up. For example, **/mnt/data/**. The file system must be mounted.
  - When backing up multiple file systems and saving them on a single tape device, add a session label to each backup using the **-L *label*** option so that it is easier to identify them when restoring. Replace *label* with any name for your backup: for example, **backup_data**.

Example 9.1. Backing up multiple XFS file systems

- To back up the content of XFS file systems mounted on the **/boot/** and **/data/** directories and save them as files in the **/backup-files/** directory:

```
# xfsdump -l 0 -f /backup-files/boot.xfsdump /boot
# xfsdump -l 0 -f /backup-files/data.xfsdump /data
```

- To back up multiple file systems on a single tape device, add a session label to each backup using the **-L *label*** option:

```
# xfsdump -l 0 -L "backup_boot" -f /dev/st0 /boot
# xfsdump -l 0 -L "backup_data" -f /dev/st0 /data
```

Additional resources

- The **xfsdump(8)** man page.

9.3.3. Additional resources

- The **xfsdump(8)** man page.

9.4. RESTORING AN XFS FILE SYSTEM FROM BACKUP

As a system administrator, you can use the **xfsrestore** utility to restore XFS backup created with the **xfsdump** utility and stored in a file or on a tape.

9.4.1. Features of restoring XFS from backup

This section describes key concepts and features of restoring an XFS file system from backup with the **xfsrestore** utility.

The **xfsrestore** utility restores file systems from backups produced by **xfsdump**. The **xfsrestore** utility has two modes:

- The **simple** mode enables users to restore an entire file system from a level 0 dump. This is the default mode.
- The **cumulative** mode enables file system restoration from an incremental backup: that is, level 1 to level 9.

A unique *session ID* or *session label* identifies each backup. Restoring a backup from a tape containing multiple backups requires its corresponding session ID or label.

To extract, add, or delete specific files from a backup, enter the **xfsrestore** interactive mode. The interactive mode provides a set of commands to manipulate the backup files.

Additional resources

- The **xfsrestore(8)** man page.

9.4.2. Restoring an XFS file system from backup with xfsrestore

This procedure describes how to restore the content of an XFS file system from a file or tape backup.

Prerequisites

- A file or tape backup of XFS file systems, as described in Section 9.3, “Backing up an XFS file system”.
- A storage device where you can restore the backup.

Procedure

- The command to restore the backup varies depending on whether you are restoring from a full backup or an incremental one, or are restoring multiple backups from a single tape device:

```
# xfsrestore [-r] [-S session-id] [-L session-label] [-i]
           -f backup-location restoration-path
```

  - Replace *backup-location* with the location of the backup. This can be a regular file, a tape drive, or a remote tape device. For example, **/backup-files/Data.xfsdump** for a file or **/dev/st0** for a tape drive.
  - Replace *restoration-path* with the path to the directory where you want to restore the file system. For example, **/mnt/data/**.

- To restore a file system from an incremental (level 1 to level 9) backup, add the **-r** option.
- To restore a backup from a tape device that contains multiple backups, specify the backup using the **-S** or **-L** options.
  The **-S** option lets you choose a backup by its session ID, while the **-L** option lets you choose by the session label. To obtain the session ID and session labels, use the **xfsrestore -I** command.

  Replace *session-id* with the session ID of the backup. For example, **b74a3586-e52e-4a4a-8775-c3334fa8ea2c**. Replace *session-label* with the session label of the backup. For example, **my_backup_session_label**.
- To use **xfsrestore** interactively, use the **-i** option.
  The interactive dialog begins after **xfsrestore** finishes reading the specified device. Available commands in the interactive **xfsrestore** shell include **cd**, **ls**, **add**, **delete**, and **extract**; for a complete list of commands, use the **help** command.

Example 9.2. Restoring Multiple XFS File Systems

- To restore the XFS backup files and save their content into directories under **/mnt/**:

```
# xfsrestore -f /backup-files/boot.xfsdump /mnt/boot/
# xfsrestore -f /backup-files/data.xfsdump /mnt/data/
```

- To restore from a tape device containing multiple backups, specify each backup by its session label or session ID:

```
# xfsrestore -L "backup_boot" -f /dev/st0 /mnt/boot/
# xfsrestore -S "45e9af35-efd2-4244-87bc-4762e476cbab" \
        -f /dev/st0 /mnt/data/
```

Additional resources

- The **xfsrestore(8)** man page.

9.4.3. Informational messages when restoring an XFS backup from a tape

When restoring a backup from a tape with backups from multiple file systems, the **xfsrestore** utility might issue messages. The messages inform you whether a match of the requested backup has been found when **xfsrestore** examines each backup on the tape in sequential order. For example:

```
xfsrestore: preparing drive
xfsrestore: examining media file 0
xfsrestore: inventory session uuid (8590224e-3c93-469c-a311-fc8f23029b2a) does not match the
media header's session uuid (7eda9f86-f1e9-4dfd-b1d4-c50467912408)
xfsrestore: examining media file 1
xfsrestore: inventory session uuid (8590224e-3c93-469c-a311-fc8f23029b2a) does not match the
media header's session uuid (7eda9f86-f1e9-4dfd-b1d4-c50467912408)
[...]
```

The informational messages keep appearing until the matching backup is found.

9.4.4. Additional resources

- The **xfsrestore(8)** man page.

9.5. REPAIRING AN XFS FILE SYSTEM

As a system administrator, you can repair a corrupted XFS file system.

9.5.1. Error-handling mechanisms in XFS

This section describes how XFS handles various kinds of errors in the file system.

Unclean unmounts
Journalling maintains a transactional record of metadata changes that happen on the file system.

In the event of a system crash, power failure, or other unclean unmount, XFS uses the journal (also called log) to recover the file system. The kernel performs journal recovery when mounting the XFS file system.

Corruption
In this context, *corruption* means errors on the file system caused by, for example:

- Hardware faults
- Bugs in storage firmware, device drivers, the software stack, or the file system itself
- Problems that cause parts of the file system to be overwritten by something outside of the file system

When XFS detects corruption in the file system or the file-system metadata, it shuts down the file system and reports the incident in the system log. Note that if the corruption occurred on the file system hosting the /**var** directory, these logs will not be available after a reboot.

Example 9.3. System log entry reporting an XFS corruption

```
# dmesg --notime | tail -15

XFS (loop0): Mounting V5 Filesystem
XFS (loop0): Metadata CRC error detected at xfs_agi_read_verify+0xcb/0xf0 [xfs], xfs_agi block 0x2
XFS (loop0): Unmount and run xfs_repair
XFS (loop0): First 128 bytes of corrupted metadata buffer:
00000000027b3b56: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000005f9abc7a: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000005b0aef35: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000da9d2ded: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000001e265b07: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000006a40df69: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000000b272907: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000e484aac5: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
XFS (loop0): metadata I/O error in "xfs_trans_read_buf_map" at daddr 0x2 len 1 error 74
XFS (loop0): xfs_imap_lookup: xfs_ialloc_read_agi() returned error -117, agno 0
XFS (loop0): Failed to read root inode 0x80, error 11
```

User-space utilities usually report the *Input/output error* message when trying to access a corrupted XFS file system. Mounting an XFS file system with a corrupted log results in the following error message:

```
mount: /mount-point: mount(2) system call failed: Structure needs cleaning.
```

You must manually use the **xfs_repair** utility to repair the corruption. Unlike other file system repair utilities, **xfs_repair** does not run at boot time, even when an XFS file system was not cleanly unmounted. In the event of an unclean unmount, XFS simply replays the log at mount time, ensuring a consistent file system; **xfs_repair** cannot repair an XFS file system with a dirty log without remounting it first.

Additional resources

- The **xfs_repair(8)** man page provides a detailed list of XFS corruption checks.

9.5.2. Repairing an XFS file system with xfs_repair

This procedure repairs a corrupted XFS file system using the **xfs_repair** utility.

Procedure

1. Clear the log by remounting the file system:

```
# mount file-system
# umount file-system
```

2. Use the **xfs_repair** utility to repair the unmounted file system:

   - If the mount succeeded, no additional options are required:

```
# xfs_repair block-device
```

   - If the mount failed with the *Structure needs cleaning* error, the log is corrupted and cannot be replayed. Use the **-L** option (*force log zeroing*) to clear the log:

WARNING

This command causes all metadata updates in progress at the time of the crash to be lost, which might cause significant file system damage and data loss. This should be used only as a last resort.

```
# xfs_repair -L block-device
```

3. Mount the file system:

```
# mount file-system
```

Additional resources

- The **xfs_repair(8)** man page.

9.6. INCREASING THE SIZE OF AN XFS FILE SYSTEM

As a system administrator, you can increase the size of an XFS file system to utilize larger storage capacity.

IMPORTANT

It is not currently possible to decrease the size of XFS file systems.

9.6.1. Increasing the size of an XFS file system with xfs_growfs

This procedure describes how to grow an XFS file system using the **xfs_growfs** utility.

Prerequisites

- Ensure that the underlying block device is of an appropriate size to hold the resized file system later. Use the appropriate resizing methods for the affected block device.
- Mount the XFS file system.

Procedure

- While the XFS file system is mounted, use the **xfs_growfs** utility to increase its size:

```
# xfs_growfs file-system -D new-size
```

  - Replace *file-system* with the mount point of the XFS file system.
  - With the **-D** option, replace *new-size* with the desired new size of the file system specified in the number of file system blocks.
    To find out the block size in kB of a given XFS file system, use the **xfs_info** utility:

```
# xfs_info block-device

...
data     =               bsize=4096
...
```

  - Without the **-D** option, **xfs_growfs** grows the file system to the maximum size supported by the underlying device.

Additional resources

- The **xfs_growfs(8)** man page.

CHAPTER 10. MOUNTING FILE SYSTEMS

As a system administrator, you can mount file systems on your system to access data on them.

10.1. THE LINUX MOUNT MECHANISM

This section explains basic concepts of mounting file systems on Linux.

On Linux, UNIX, and similar operating systems, file systems on different partitions and removable devices (CDs, DVDs, or USB flash drives for example) can be attached to a certain point (the mount point) in the directory tree, and then detached again. While a file system is mounted on a directory, the original content of the directory is not accessible.

Note that Linux does not prevent you from mounting a file system to a directory with a file system already attached to it.

When mounting, you can identify the device by:

- a universally unique identifier (UUID): for example, **UUID=34795a28-ca6d-4fd8-a347-73671d0c19cb**
- a volume label: for example, **LABEL=home**
- a full path to a non-persistent block device: for example, **/dev/sda3**

When you mount a file system using the **mount** command without all required information, that is without the device name, the target directory, or the file system type, the **mount** utility reads the content of the **/etc/fstab** file to check if the given file system is listed there. The **/etc/fstab** file contains a list of device names and the directories in which the selected file systems are set to be mounted as well as the file system type and mount options. Therefore, when mounting a file system that is specified in **/etc/fstab**, the following command syntax is sufficient:

- Mounting by the mount point:

```
# mount directory
```

- Mounting by the block device:

```
# mount device
```

Additional resources

- The **mount(8)** man page.
- For information on how to list persistent naming attributes such as the UUID, see Section 7.6, “Listing persistent naming attributes”.

10.2. LISTING CURRENTLY MOUNTED FILE SYSTEMS

This procedure describes how to list all currently mounted file systems on the command line.

Procedure

- To list all mounted file systems, use the **findmnt** utility:

```
$ findmnt
```

- To limit the listed file systems only to a certain file system type, add the **--types** option:

```
$ findmnt --types fs-type
```

For example:

Example 10.1. Listing only XFS file systems

```
$ findmnt --types xfs

TARGET  SOURCE                                          FSTYPE OPTIONS
/       /dev/mapper/luks-5564ed00-6aac-4406-bfb4-c59bf5de48b5 xfs    rw,relatime
├─/boot /dev/sda1                                       xfs    rw,relatime
└─/home /dev/mapper/luks-9d185660-7537-414d-b727-d92ea036051e xfs    rw,relatime
```

Additional resources

- The **findmnt(8)** man page.

10.3. MOUNTING A FILE SYSTEM WITH MOUNT

This procedure describes how to mount a file system using the **mount** utility.

Prerequisites

- Make sure that no file system is already mounted on your chosen mount point:

```
$ findmnt mount-point
```

Procedure

1. To attach a certain file system, use the **mount** utility:

```
# mount device mount-point
```

Example 10.2. Mounting an XFS file system

For example, to mount a local XFS file system identified by UUID:

```
# mount UUID=ea74bbec-536d-490c-b8d9-5b40bbd7545b /mnt/data
```

2. If **mount** cannot recognize the file system type automatically, specify it using the **--types** option:

```
# mount --types type device mount-point
```

Example 10.3. Mounting an NFS file system

For example, to mount a remote NFS file system:

```
# mount --types nfs4 host:/remote-export /mnt/nfs
```

Additional resources

- The **mount(8)** man page.

10.4. MOVING A MOUNT POINT

This procedure describes how to change the mount point of a mounted file system to a different directory.

Procedure

1. To change the directory in which a file system is mounted:

```
# mount --move old-directory new-directory
```

Example 10.4. Moving a home file system

For example, to move the file system mounted in the **/mnt/userdirs/** directory to the **/home/** mount point:

```
# mount --move /mnt/userdirs /home
```

2. Verify that the file system has been moved as expected:

```
$ findmnt
$ ls old-directory
$ ls new-directory
```

Additional resources

- The **mount(8)** man page.

10.5. UNMOUNTING A FILE SYSTEM WITH UMOUNT

This procedure describes how to unmount a file system using the **umount** utility.

Procedure

1. Try unmounting the file system using either of the following commands:
   - By mount point:

```
# umount mount-point
```

   - By device:

```
# umount device
```

-

If the command fails with an error similar to the following, it means that the file system is in use because of a process is using resources on it:

```
umount: /run/media/user/FlashDrive: target is busy.
```

2. If the file system is in use, use the **fuser** utility to determine which processes are accessing it. For example:

```
$ fuser --mount /run/media/user/FlashDrive

/run/media/user/FlashDrive: 18351
```

Afterwards, terminate the processes using the file system and try unmounting it again.

10.6. COMMON MOUNT OPTIONS

This section lists some commonly used options of the **mount** utility.

You can use these options in the following syntax:

```
# mount --options option1,option2,option3 device mount-point
```

Table 10.1. Common mount options

Option	Description
async	Enables asynchronous input and output operations on the file system.
auto	Enables the file system to be mounted automatically using the **mount -a** command.
defaults	Provides an alias for the **async,auto,dev,exec,nouser,rw,suid** options.
exec	Allows the execution of binary files on the particular file system.
loop	Mounts an image as a loop device.
noauto	Default behavior disables the automatic mount of the file system using the **mount -a** command.
noexec	Disallows the execution of binary files on the particular file system.
nouser	Disallows an ordinary user (that is, other than root) to mount and unmount the file system.
remount	Remounts the file system in case it is already mounted.
ro	Mounts the file system for reading only.

Option	Description
rw	Mounts the file system for both reading and writing.
user	Allows an ordinary user (that is, other than root) to mount and unmount the file system.

10.7. SHARING A MOUNT ON MULTIPLE MOUNT POINTS

As a system administrator, you can duplicate mount points to make the file systems accessible from multiple directories.

10.7.1. Types of shared mounts

There are multiple types of shared mounts that you can use. The difference between them is what happens when you mount another file system under one of the shared mount points. The shared mounts are implemented using the *shared subtrees* functionality.

The types are:

Private mount

This type does not receive or forward any propagation events.
When you mount another file system under either the duplicate or the original mount point, it is not reflected in the other.

Shared mount

This type creates an exact replica of a given mount point.
When a mount point is marked as a shared mount, any mount within the original mount point is reflected in it, and vice versa.

This is the default mount type of the root file system.

Slave mount

This type creates a limited duplicate of a given mount point.
When a mount point is marked as a slave mount, any mount within the original mount point is reflected in it, but no mount within a slave mount is reflected in its original.

Unbindable mount

This type prevents the given mount point from being duplicated whatsoever.

10.7.2. Creating a private mount point duplicate

This procedure duplicates a mount point as a private mount. File systems that you later mount under the duplicate or the original mount point are not reflected in the other.

Procedure

1. Create a virtual file system (VFS) node from the original mount point:

```
# mount --bind original-dir original-dir
```

2. Mark the original mount point as private:

```
# mount --make-private original-dir
```

Alternatively, to change the mount type for the selected mount point and all mount points under it, use the **--make-rprivate** option instead of **--make-private**.

3. Create the duplicate:

```
# mount --bind original-dir duplicate-dir
```

Example 10.5. Duplicating /media into /mnt as a private mount point

1. Create a VFS node from the **/media** directory:

```
# mount --bind /media /media
```

2. Mark the **/media** directory as private:

```
# mount --make-private /media
```

3. Create its duplicate in **/mnt**:

```
# mount --bind /media /mnt
```

4. It is now possible to verify that **/media** and **/mnt** share content but none of the mounts within **/media** appear in **/mnt**. For example, if the CD-ROM drive contains non-empty media and the **/media/cdrom/** directory exists, use:

```
# mount /dev/cdrom /media/cdrom
# ls /media/cdrom
EFI  GPL  isolinux  LiveOS
# ls /mnt/cdrom
#
```

5. It is also possible to verify that file systems mounted in the **/mnt** directory are not reflected in **/media**. For instance, if a non-empty USB flash drive that uses the **/dev/sdc1** device is plugged in and the **/mnt/flashdisk/** directory is present, use:

```
# mount /dev/sdc1 /mnt/flashdisk
# ls /media/flashdisk
# ls /mnt/flashdisk
en-US  publican.cfg
```

Additional resources

- The **mount(8)** man page.

10.7.3. Creating a shared mount point duplicate

This procedure duplicates a mount point as a shared mount. File systems that you later mount under the original directory or the duplicate are always reflected in the other.

Procedure

1. Create a virtual file system (VFS) node from the original mount point:

```
# mount --bind original-dir original-dir
```

2. Mark the original mount point as shared:

```
# mount --make-shared original-dir
```

Alternatively, to change the mount type for the selected mount point and all mount points under it, use the **--make-rshared** option instead of **--make-shared**.

3. Create the duplicate:

```
# mount --bind original-dir duplicate-dir
```

Example 10.6. Duplicating /media into /mnt as a shared mount point

To make the **/media** and **/mnt** directories share the same content:

1. Create a VFS node from the **/media** directory:

```
# mount --bind /media /media
```

2. Mark the **/media** directory as shared:

```
# mount --make-shared /media
```

3. Create its duplicate in **/mnt**:

```
# mount --bind /media /mnt
```

4. It is now possible to verify that a mount within **/media** also appears in **/mnt**. For example, if the CD-ROM drive contains non-empty media and the **/media/cdrom/** directory exists, use:

```
# mount /dev/cdrom /media/cdrom
# ls /media/cdrom
EFI  GPL  isolinux  LiveOS
# ls /mnt/cdrom
EFI  GPL  isolinux  LiveOS
```

5. Similarly, it is possible to verify that any file system mounted in the **/mnt** directory is reflected in **/media**. For instance, if a non-empty USB flash drive that uses the **/dev/sdc1** device is plugged in and the **/mnt/flashdisk/** directory is present, use:

```
# mount /dev/sdc1 /mnt/flashdisk
# ls /media/flashdisk
en-US  publican.cfg
# ls /mnt/flashdisk
en-US  publican.cfg
```

Additional resources

- The **mount(8)** man page.

10.7.4. Creating a slave mount point duplicate

This procedure duplicates a mount point as a slave mount. File systems that you later mount under the original mount point are reflected in the duplicate but not the other way around.

Procedure

1. Create a virtual file system (VFS) node from the original mount point:

```
# mount --bind original-dir original-dir
```

2. Mark the original mount point as shared:

```
# mount --make-shared original-dir
```

   Alternatively, to change the mount type for the selected mount point and all mount points under it, use the **--make-rshared** option instead of **--make-shared**.

3. Create the duplicate and mark it as slave:

```
# mount --bind original-dir duplicate-dir
# mount --make-slave duplicate-dir
```

Example 10.7. Duplicating /media into /mnt as a slave mount point

This example shows how to get the content of the **/media** directory to appear in **/mnt** as well, but without any mounts in the **/mnt** directory to be reflected in **/media**.

1. Create a VFS node from the **/media** directory:

```
# mount --bind /media /media
```

2. Mark the **/media** directory as shared:

```
# mount --make-shared /media
```

3. Create its duplicate in **/mnt** and mark it as slave:

```
# mount --bind /media /mnt
# mount --make-slave /mnt
```

4. Verify that a mount within **/media** also appears in **/mnt**. For example, if the CD-ROM drive contains non-empty media and the **/media/cdrom/** directory exists, use:

```
# mount /dev/cdrom /media/cdrom
# ls /media/cdrom
EFI  GPL  isolinux  LiveOS
# ls /mnt/cdrom
EFI  GPL  isolinux  LiveOS
```

5. Also verify that file systems mounted in the **/mnt** directory are not reflected in **/media**. For instance, if a non-empty USB flash drive that uses the **/dev/sdc1** device is plugged in and the **/mnt/flashdisk/** directory is present, use:

```
# mount /dev/sdc1 /mnt/flashdisk
# ls /media/flashdisk
# ls /mnt/flashdisk
en-US  publican.cfg
```

Additional resources

- The **mount(8)** man page.

10.7.5. Preventing a mount point from being duplicated

This procedure marks a mount point as unbindable so that it is not possible to duplicate it in another mount point.

Procedure

- To change the type of a mount point to an unbindable mount, use:

```
# mount --bind mount-point mount-point
# mount --make-unbindable mount-point
```

Alternatively, to change the mount type for the selected mount point and all mount points under it, use the **--make-runbindable** option instead of **--make-unbindable**.

Any subsequent attempt to make a duplicate of this mount fails with the following error:

```
# mount --bind mount-point duplicate-dir

mount: wrong fs type, bad option, bad superblock on mount-point,
missing codepage or helper program, or other error
In some cases useful info is found in syslog - try
dmesg | tail  or so
```

Example 10.8. Preventing /media from being duplicated

- To prevent the **/media** directory from being shared, use:

```
# mount --bind /media /media
# mount --make-unbindable /media
```

Additional resources

- The **mount(8)** man page.

10.7.6. Related information

- The *Shared subtrees* article on Linux Weekly News: https://lwn.net/Articles/159077/.

10.8. PERSISTENTLY MOUNTING FILE SYSTEMS

As a system administrator, you can persistently mount file systems to configure non-removable storage.

10.8.1. The /etc/fstab file

This section describes the **/etc/fstab** configuration file, which controls persistent mount points of file systems. Using **/etc/fstab** is the recommended way to persistently mount file systems.

Each line in the **/etc/fstab** file defines a mount point of a file system. It includes six fields separated by white space:

1. The block device identified by a persistent attribute or a path it the **/dev** directory.
2. The directory where the device will be mounted.
3. The file system on the device.
4. Mount options for the file system. The option **defaults** means that the partition is mounted at boot time with default options. This section also recognizes **systemd** mount unit options in the **x-systemd.*option*** format.
5. Backup option for the **dump** utility.
6. Check order for the **fsck** utility.

Example 10.9. The /boot file system in/etc/fstab

Block device	Mount point	File system	Options	Backup	Check
UUID=ea74bbec-536d-490c-b8d9-5b40bbd7545b	**/boot**	**xfs**	**defaults**	**0**	**0**

The **systemd** service automatically generates mount units from entries in **/etc/fstab**.

Additional resources

- The **fstab(5)** man page.
- The *fstab* section of the **systemd.mount(5)** man page.

10.8.2. Adding a file system to /etc/fstab

This procedure describes how to configure persistent mount point for a file system in the **/etc/fstab** configuration file.

Procedure

1. Find out the UUID attribute of the file system:

```
$ lsblk --fs storage-device
```

For example:

Example 10.10. Viewing the UUID of a partition

```
$ lsblk --fs /dev/sda1

NAME FSTYPE LABEL UUID                                 MOUNTPOINT
sda1 xfs    Boot  ea74bbec-536d-490c-b8d9-5b40bbd7545b /boot
```

2. If the mount point directory does not exist, create it:

```
# mkdir --parents mount-point
```

3. As root, edit the **/etc/fstab** file and add a line for the file system, identified by the UUID. For example:

Example 10.11. The /boot mount point in /etc/fstab

```
UUID=ea74bbec-536d-490c-b8d9-5b40bbd7545b /boot xfs defaults 0 0
```

4. Regenerate mount units so that your system registers the new configuration:

```
# systemctl daemon-reload
```

5. Try mounting the file system to verify that the configuration works:

```
# mount mount-point
```

Additional resources

- Other persistent attributes that you can use to identify the file system: Section 7.3, "Device names managed by the udev mechanism in /dev/disk/"

10.9. MOUNTING FILE SYSTEMS ON DEMAND

As a system administrator, you can configure file systems, such as NFS, to mount automatically on demand.

10.9.1. The autofs service

This section explains the benefits and basic concepts of the **autofs** service, used to mount file systems on demand.

One drawback of permanent mounting using the **/etc/fstab** configuration is that, regardless of how infrequently a user accesses the mounted file system, the system must dedicate resources to keep the mounted file system in place. This might affect system performance when, for example, the system is maintaining NFS mounts to many systems at one time.

An alternative to **/etc/fstab** is to use the kernel-based **autofs** service. It consists of the following components:

- A kernel module that implements a file system, and
- A user-space service that performs all of the other functions.

The **autofs** service can mount and unmount file systems automatically (on-demand), therefore saving system resources. It can be used to mount file systems such as NFS, AFS, SMBFS, CIFS, and local file systems.

Additional resources

- The **autofs(8)** man page.

10.9.2. The autofs configuration files

This section describes the usage and syntax of configuration files used by the **autofs** service.

The master map file
The **autofs** service uses **/etc/auto.master** (master map) as its default primary configuration file. This can be changed to use another supported network source and name using the **autofs** configuration in the **/etc/autofs.conf** configuration file in conjunction with the Name Service Switch (NSS) mechanism.

All on-demand mount points must be configured in the master map. Mount point, host name, exported directory, and options can all be specified in a set of files (or other supported network sources) rather than configuring them manually for each host.

The master map file lists mount points controlled by **autofs**, and their corresponding configuration files or network sources known as automount maps. The format of the master map is as follows:

```
mount-point  map-name  options
```

The variables used in this format are:

mount-point
The **autofs** mount point; for example, **/mnt/data/**.

map-file
The map source file, which contains a list of mount points and the file system location from which those mount points should be mounted.

options
If supplied, these apply to all entries in the given map, if they do not themselves have options specified.

Example 10.12. The /etc/auto.master file

The following is a sample line from **/etc/auto.master** file:

```
/mnt/data  /etc/auto.data
```

Map files
Map files configure the properties of individual on-demand mount points.

The automounter creates the directories if they do not exist. If the directories exist before the automounter was started, the automounter will not remove them when it exits. If a timeout is specified, the directory is automatically unmounted if the directory is not accessed for the timeout period.

The general format of maps is similar to the master map. However, the options field appears between the mount point and the location instead of at the end of the entry as in the master map:

```
mount-point  options  location
```

The variables used in this format are:

mount-point

This refers to the **autofs** mount point. This can be a single directory name for an indirect mount or the full path of the mount point for direct mounts. Each direct and indirect map entry key (*mount-point*) can be followed by a space separated list of offset directories (subdirectory names each beginning with /) making them what is known as a multi-mount entry.

options

When supplied, these are the mount options for the map entries that do not specify their own options. This field is optional.

location

This refers to the file system location such as a local file system path (preceded with the Sun map format escape character **:** for map names beginning with /), an NFS file system or other valid file system location.

Example 10.13. A map file

The following is a sample from a map file; for example, **/etc/auto.misc**:

```
payroll  -fstype=nfs4  personnel:/dev/disk/by-uuid/52b94495-e106-4f29-b868-fe6f6c2789b1
sales    -fstype=xfs   :/dev/disk/by-uuid/5564ed00-6aac-4406-bfb4-c59bf5de48b5
```

The first column in the map file indicates the **autofs** mount point: **sales** and **payroll** from the server called **personnel**. The second column indicates the options for the **autofs** mount. The third column indicates the source of the mount.

Following the given configuration, the **autofs** mount points will be **/home/payroll** and **/home/sales**. The **-fstype=** option is often omitted and is generally not needed for correct operation.

Using the given configuration, if a process requires access to an **autofs** unmounted directory such as **/home/payroll/2006/July.sxc**, the **autofs** service automatically mounts the directory.

The amd map format

The **autofs** service recognizes map configuration in the **amd** format as well. This is useful if you want to reuse existing automounter configuration written for the **am-utils** service, which has been removed from Red Hat Enterprise Linux.

However, Red Hat recommends using the simpler **autofs** format described in the previous sections.

Additional resources

- The **autofs(5)**, **autofs.conf(5)**, and **auto.master(5)** man pages.

- For details on the **amd** map format, see the **/usr/share/doc/autofs/README.amd-maps** file, which is provided by the **autofs** package.

10.9.3. Configuring autofs mount points

This procedure describes how to configure on-demand mount points using the **autofs** service.

Prerequisites

- Install the **autofs** package:

```
# yum install autofs
```

- Start and enable the **autofs** service:

```
# systemctl enable --now autofs
```

Procedure

1. Create a map file for the on-demand mount point, located at **/etc/auto.*identifier***. Replace *identifier* with a name that identifies the mount point.
2. In the map file, fill in the mount point, options, and location fields as described in Section 10.9.2, "The autofs configuration files".
3. Register the map file in the master map file, as described in Section 10.9.2, "The autofs configuration files".
4. Try accessing content in the on-demand directory:

```
$ ls automounted-directory
```

10.9.4. Overriding or augmenting autofs site configuration files

It is sometimes useful to override site defaults for a specific mount point on a client system.

Example 10.14. Initial conditions

For example, consider the following conditions:

- Automounter maps are stored in NIS and the **/etc/nsswitch.conf** file has the following directive:

```
automount:    files nis
```

- The **auto.master** file contains:

```
+auto.master
```

- The NIS **auto.master** map file contains:

```
/home auto.home
```

- The NIS **auto.home** map contains:

```
beth    fileserver.example.com:/export/home/beth
joe     fileserver.example.com:/export/home/joe
*       fileserver.example.com:/export/home/&
```

- The file map **/etc/auto.home** does not exist.

Procedure

Example 10.15. Mounting home directories from a different server

Given the preceding conditions, let's assume that the client system needs to override the NIS map **auto.home** and mount home directories from a different server.

- In this case, the client needs to use the following **/etc/auto.master** map:

```
/home /etc/auto.home
+auto.master
```

- The **/etc/auto.home** map contains the entry:

```
*    labserver.example.com:/export/home/&
```

Because the automounter only processes the first occurrence of a mount point, the **/home** directory contains the content of **/etc/auto.home** instead of the NIS **auto.home** map.

Example 10.16. Augmenting auto.home with only selected entries

Alternatively, to augment the site-wide **auto.home** map with just a few entries:

1. Create an **/etc/auto.home** file map, and in it put the new entries. At the end, include the NIS **auto.home** map. Then the **/etc/auto.home** file map looks similar to:

```
mydir someserver:/export/mydir
+auto.home
```

2. With these NIS **auto.home** map conditions, listing the content of the **/home** directory outputs:

```
$ ls /home

beth joe mydir
```

This last example works as expected because **autofs** does not include the contents of a file map of the same name as the one it is reading. As such, **autofs** moves on to the next map source in the **nsswitch** configuration.

10.9.5. Using LDAP to store automounter maps

This procedure configures **autofs** to store automounter maps in LDAP configuration rather than in **autofs** map files.

Prerequisites

- LDAP client libraries must be installed on all systems configured to retrieve automounter maps from LDAP. On Red Hat Enterprise Linux, the **openldap** package should be installed automatically as a dependency of the **autofs** package.

Procedure

1. To configure LDAP access, modify the **/etc/openldap/ldap.conf** file. Ensure that the **BASE**, **URI**, and **schema** options are set appropriately for your site.

2. The most recently established schema for storing automount maps in LDAP is described by the **rfc2307bis** draft. To use this schema, set it in the **/etc/autofs.conf** configuration file by removing the comment characters from the schema definition. For example:

   Example 10.17. Setting autofs configuration

   ```
   DEFAULT_MAP_OBJECT_CLASS="automountMap"
   DEFAULT_ENTRY_OBJECT_CLASS="automount"
   DEFAULT_MAP_ATTRIBUTE="automountMapName"
   DEFAULT_ENTRY_ATTRIBUTE="automountKey"
   DEFAULT_VALUE_ATTRIBUTE="automountInformation"
   ```

3. Ensure that all other schema entries are commented in the configuration. The **automountKey** attribute replaces the **cn** attribute in the **rfc2307bis** schema. Following is an example of an LDAP Data Interchange Format (LDIF) configuration:

   Example 10.18. LDF Configuration

   ```
   # extended LDIF
   #
   # LDAPv3
   # base <> with scope subtree
   # filter: (&(objectclass=automountMap)(automountMapName=auto.master))
   # requesting: ALL
   #

   # auto.master, example.com
   dn: automountMapName=auto.master,dc=example,dc=com
   objectClass: top
   objectClass: automountMap
   automountMapName: auto.master

   # extended LDIF
   #
   # LDAPv3
   # base <automountMapName=auto.master,dc=example,dc=com> with scope subtree
   # filter: (objectclass=automount)
   # requesting: ALL
   #

   # /home, auto.master, example.com
   dn: automountMapName=auto.master,dc=example,dc=com
   objectClass: automount
   cn: /home
   ```

```
automountKey: /home
automountInformation: auto.home

# extended LDIF
#
# LDAPv3
# base <> with scope subtree
# filter: (&(objectclass=automountMap)(automountMapName=auto.home))
# requesting: ALL
#

# auto.home, example.com
dn: automountMapName=auto.home,dc=example,dc=com
objectClass: automountMap
automountMapName: auto.home

# extended LDIF
#
# LDAPv3
# base <automountMapName=auto.home,dc=example,dc=com> with scope subtree
# filter: (objectclass=automount)
# requesting: ALL
#

# foo, auto.home, example.com
dn: automountKey=foo,automountMapName=auto.home,dc=example,dc=com
objectClass: automount
automountKey: foo
automountInformation: filer.example.com:/export/foo

# /, auto.home, example.com
dn: automountKey=/,automountMapName=auto.home,dc=example,dc=com
objectClass: automount
automountKey: /
automountInformation: filer.example.com:/export/&
```

Additional resources

- The **rfc2307bis** draft: https://tools.ietf.org/html/draft-howard-rfc2307bis.

10.10. SETTING READ-ONLY PERMISSIONS FOR THE ROOT FILE SYSTEM

Sometimes, you need to mount the root file system (/) with read-only permissions. Example use cases include enhancing security or ensuring data integrity after an unexpected system power-off.

10.10.1. Files and directories that always retain write permissions

For the system to function properly, some files and directories need to retain write permissions. When the root file system is mounted in read-only mode, these files are mounted in RAM using the **tmpfs** temporary file system.

The default set of such files and directories is read from the **/etc/rwtab** file, which contains:

-

```
dirs /var/cache/man
dirs /var/gdm
<content truncated>

empty /tmp
empty /var/cache/foomatic
<content truncated>

files /etc/adjtime
files /etc/ntp.conf
<content truncated>
```

Entries in the **/etc/rwtab** file follow this format:

```
copy-method    path
```

In this syntax:

- Replace *copy-method* with one of the keywords specifying how the file or directory is copied to tmpfs.
- Replace *path* with the path to the file or directory.

The **/etc/rwtab** file recognizes the following ways in which a file or directory can be copied to **tmpfs**:

empty

An empty path is copied to **tmpfs**. For example:

```
empty /tmp
```

dirs

A directory tree is copied to **tmpfs**, empty. For example:

```
dirs /var/run
```

files

A file or a directory tree is copied to **tmpfs** intact. For example:

```
files /etc/resolv.conf
```

The same format applies when adding custom paths to **/etc/rwtab.d/**.

10.10.2. Configuring the root file system to mount with read-only permissions on boot

With this procedure, the root file system is mounted read-only on all following boots.

Procedure

1. In the **/etc/sysconfig/readonly-root** file, set the **READONLY** option to **yes**:

```
# Set to 'yes' to mount the file systems as read-only.
READONLY=yes
```

2. Add the **ro** option in the root entry (/) in the **/etc/fstab** file:

```
/dev/mapper/luks-c376919e... / xfs x-systemd.device-timeout=0,ro 1 1
```

3. Add the **ro** option to the **GRUB_CMDLINE_LINUX** directive in the **/etc/default/grub** file and ensure that the directive does not contain **rw**:

```
GRUB_CMDLINE_LINUX="rhgb quiet... ro"
```

4. Recreate the GRUB2 configuration file:

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

5. If you need to add files and directories to be mounted with write permissions in the **tmpfs** file system, create a text file in the **/etc/rwtab.d/** directory and put the configuration there. For example, to mount the **/etc/example/file** file with write permissions, add this line to the **/etc/rwtab.d/example** file:

```
files /etc/example/file
```

IMPORTANT

Changes made to files and directories in **tmpfs** do not persist across boots.

6. Reboot the system to apply the changes.

Troubleshooting

- If you mount the root file system with read-only permissions by mistake, you can remount it with read-and-write permissions again using the following command:

```
# mount -o remount,rw /
```

CHAPTER 11. DISCARDING UNUSED BLOCKS

You can perform or schedule discard operations on block devices that support them.

11.1. BLOCK DISCARD OPERATIONS

Block discard operations discard blocks that are no longer in use by a mounted file system. They are useful on:

- Solid-state drives (SSDs)
- Thinly-provisioned storage

Requirements

The block device underlying the file system must support physical discard operations.

Physical discard operations are supported if the value in the **/sys/block/*device*/queue/discard_max_bytes** file is not zero.

11.2. TYPES OF BLOCK DISCARD OPERATIONS

You can run discard operations using different methods:

Batch discard

Are run explicitly by the user. They discard all unused blocks in the selected file systems.

Online discard

Are specified at mount time. They run in real time without user intervention. Online discard operations discard only the blocks that are transitioning from used to free.

Periodic discard

Are batch operations that are run regularly by a **systemd** service.

All types are supported by the XFS and ext4 file systems and by VDO.

Recommendations

Red Hat recommends that you use batch or periodic discard.

Use online discard only if:

- the system's workload is such that batch discard is not feasible, or
- online discard operations are necessary to maintain performance.

11.3. PERFORMING BATCH BLOCK DISCARD

This procedure performs a batch block discard operation to discard unused blocks on a mounted file system.

Prerequisites

- The file system is mounted.
- The block device underlying the file system supports physical discard operations.

Procedure

- Use the **fstrim** utility:
  - To perform discard only on a selected file system, use:

```
# fstrim mount-point
```

  - To perform discard on all mounted file systems, use:

```
# fstrim --all
```

If you execute the **fstrim** command on:

- a device that does not support discard operations, or
- a logical device (LVM or MD) composed of multiple devices, where any one of the device does not support discard operations,

the following message displays:

```
# fstrim /mnt/non_discard

fstrim: /mnt/non_discard: the discard operation is not supported
```

Additional resources

- The **fstrim(8)** man page

11.4. ENABLING ONLINE BLOCK DISCARD

This procedure enables online block discard operations that automatically discard unused blocks on all supported file systems.

Procedure

- Enable online discard at mount time:
  - When mounting a file system manually, add the **-o discard** mount option:

```
# mount -o discard device mount-point
```

  - When mounting a file system persistently, add the **discard** option to the mount entry in the **/etc/fstab** file.

Additional resources

- The **mount(8)** man page
- The **fstab(5)** man page

11.5. ENABLING PERIODIC BLOCK DISCARD

This procedure enables a **systemd** timer that regularly discards unused blocks on all supported file systems.

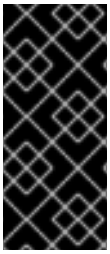
Procedure

- Enable and start the **systemd** timer:

```
# systemctl enable --now fstrim.timer
```

CHAPTER 12. MANAGING LAYERED LOCAL STORAGE WITH STRATIS

You can easily set up and manage complex storage configurations integrated by the Stratis high-level system.



IMPORTANT

Stratis is available as a Technology Preview. For information on Red Hat scope of support for Technology Preview features, see the Technology Preview Features Support Scope document.

Customers deploying Stratis are encouraged to provide feedback to Red Hat.

12.1. SETTING UP STRATIS FILE SYSTEMS

As a system administrator, you can enable and set up the Stratis volume-managing file system on your system to easily manage layered storage.

12.1.1. The purpose and features of Stratis

Stratis is a local storage-management solution for Linux. It is focused on simplicity and ease of use, and gives you access to advanced storage features.

Stratis makes the following activities easier:

- Initial configuration of storage
- Making changes later
- Using advanced storage features

Stratis is a hybrid user-and-kernel local storage management system that supports advanced storage features. The central concept of Stratis is a storage *pool*. This pool is created from one or more local disks or partitions, and volumes are created from the pool.

The pool enables many useful features, such as:

- File system snapshots
- Thin provisioning
- Tiering

12.1.2. Components of a Stratis volume

Externally, Stratis presents the following volume components in the command-line interface and the API:

blockdev

Block devices, such as a disk or a disk partition.

pool

Composed of one or more block devices.

A pool has a fixed total size, equal to the size of the block devices.

The pool contains most Stratis layers, such as the non-volatile data cache using the **dm-cache** target.

Stratis creates a /**stratis**/***my-pool***/ directory for each pool. This directory contains links to devices that represent Stratis file systems in the pool.

filesystem

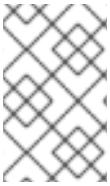
Each pool can contain one or more file systems, which store files.
File systems are thinly provisioned and do not have a fixed total size. The actual size of a file system grows with the data stored on it. If the size of the data approaches the virtual size of the file system, Stratis grows the thin volume and the file system automatically.

The file systems are formatted with XFS.

IMPORTANT

Stratis tracks information about file systems created using Stratis that XFS is not aware of, and changes made using XFS do not automatically create updates in Stratis. Users must not reformat or reconfigure XFS file systems that are managed by Stratis.

Stratis creates links to file systems at the /**stratis**/***my-pool***/***my-fs*** path.



NOTE

Stratis uses many Device Mapper devices, which show up in **dmsetup** listings and the /**proc**/**partitions** file. Similarly, the **lsblk** command output reflects the internal workings and layers of Stratis.

12.1.3. Block devices usable with Stratis

This section lists storage devices that you can use for Stratis.

Supported devices
Stratis pools have been tested to work on these types of block devices:

- LUKS
- LVM logical volumes
- MD RAID
- DM Multipath
- iSCSI
- HDDs and SSDs
- NVMe devices

WARNING

In the current version, Stratis does not handle failures in hard drives or other hardware. If you create a Stratis pool over multiple hardware devices, you increase the risk of data loss because multiple devices must be operational to access the data.

Unsupported devices
Because Stratis contains a thin-provisioning layer, Red Hat does not recommend placing a Stratis pool on block devices that are already thinly-provisioned.

Additional resources

- For iSCSI and other block devices requiring network, see the **systemd.mount(5)** man page for information on the **_netdev** mount option.

12.1.4. Installing Stratis

This procedure installs all packages necessary to use Stratis.

Procedure

1. Install packages that provide the Stratis service and command-line utilities:

```
# yum install stratisd stratis-cli
```

2. Make sure that the **stratisd** service is enabled:

```
# systemctl enable --now stratisd
```

12.1.5. Creating a Stratis pool

This procedure creates a Stratis pool from one or more block devices.

Prerequisites

- Stratis is installed. See Section 12.1.4, “Installing Stratis”.
- The **stratisd** service is running.
- The block devices on which you are creating a Stratis pool are not in use and not mounted.
- The block devices on which you are creating a Stratis pool are at least 1 GiB in size each.
- On the IBM Z architecture, the **/dev/dasd*** block devices must to be partitioned. Use the partition in the Stratis pool.
  For information on partitioning DASD devices, see Configuring a Linux instance on IBM Z .

Procedure

1. If the selected block device contains file system, partition table, or RAID signatures, erase them:

```
# wipefs --all block-device
```

Replace *block-device* with the path to a block device, such as **/dev/sdb**.

2. To create a Stratis pool on the block device, use:

```
# stratis pool create my-pool block-device
```

- Replace *my-pool* with an arbitrary name for the pool.
- Replace *block-device* with the path to the empty or wiped block device, such as **/dev/sdb**.

To create a pool from more than one block device, list them all on the command line:

```
# stratis pool create my-pool device-1 device-2 device-n
```

3. To verify, list all pools on your system:

```
# stratis pool list
```

Additional resources

- The **stratis(8)** man page

Next steps

- Create a Stratis file system on the pool. See Section 12.1.6, "Creating a Stratis file system".

12.1.6. Creating a Stratis file system

This procedure creates a Stratis file system on an existing Stratis pool.

Prerequisites

- Stratis is installed. See Section 12.1.4, "Installing Stratis".
- The **stratisd** service is running.
- You have created a Stratis pool. See Section 12.1.5, "Creating a Stratis pool".

Procedure

1. To create a Stratis file system on a pool, use:

```
# stratis fs create my-pool my-fs
```

- Replace *my-pool* with the name of your existing Stratis pool.
- Replace *my-fs* with an arbitrary name for the file system.

2. To verify, list file systems within the pool:

```
# stratis fs list my-pool
```

Additional resources

- The **stratis(8)** man page

Next steps

- Mount the Stratis file system. See Section 12.1.7, “Mounting a Stratis file system” .

12.1.7. Mounting a Stratis file system

This procedure mounts an existing Stratis file system to access the content.

Prerequisites

- Stratis is installed. See Section 12.1.4, “Installing Stratis” .
- The **stratisd** service is running.
- You have created a Stratis file system. See Section 12.1.6, “Creating a Stratis file system” .

Procedure

- To mount the file system, use the entries that Stratis maintains in the **/stratis/** directory:

```
# mount /stratis/my-pool/my-fs mount-point
```

The file system is now mounted on the *mount-point* directory and ready to use.

Additional resources

- The **mount(8)** man page

12.1.8. Persistently mounting a Stratis file system

This procedure persistently mounts a Stratis file system so that it is available automatically after booting the system.

Prerequisites

- Stratis is installed. See Section 12.1.4, “Installing Stratis” .
- The **stratisd** service is running.
- You have created a Stratis file system. See Section 12.1.6, “Creating a Stratis file system” .

Procedure

1. Determine the UUID attribute of the file system:

```
$ lsblk --output=UUID /stratis/my-pool/my-fs
```

For example:

Example 12.1. Viewing the UUID of Stratis file system

```
$ lsblk --output=UUID /stratis/my-pool/fs1

UUID
```

```
a1f0b64a-4ebb-4d4e-9543-b1d79f600283
```

2. If the mount point directory does not exist, create it:

```
# mkdir --parents mount-point
```

3. As root, edit the **/etc/fstab** file and add a line for the file system, identified by the UUID. Use **xfs** as the file system type and add the **x-systemd.requires=stratisd.service** option.
For example:

Example 12.2. The /fs1 mount point in /etc/fstab

```
UUID=a1f0b64a-4ebb-4d4e-9543-b1d79f600283 /fs1 xfs defaults,x-
systemd.requires=stratisd.service 0 0
```

4. Regenerate mount units so that your system registers the new configuration:

```
# systemctl daemon-reload
```

5. Try mounting the file system to verify that the configuration works:

```
# mount mount-point
```

Additional resources

- Section 10.8, “Persistently mounting file systems”

12.1.9. Related information

- The *Stratis Storage* website: https://stratis-storage.github.io/

12.2. EXTENDING A STRATIS VOLUME WITH ADDITIONAL BLOCK DEVICES

You can attach additional block devices to a Stratis pool to provide more storage capacity for Stratis file systems.

12.2.1. Components of a Stratis volume

Externally, Stratis presents the following volume components in the command-line interface and the API:

blockdev

Block devices, such as a disk or a disk partition.

pool

Composed of one or more block devices.
A pool has a fixed total size, equal to the size of the block devices.

The pool contains most Stratis layers, such as the non-volatile data cache using the **dm-cache** target.

Stratis creates a /**stratis**/***my-pool***/ directory for each pool. This directory contains links to devices that represent Stratis file systems in the pool.

filesystem

Each pool can contain one or more file systems, which store files.
File systems are thinly provisioned and do not have a fixed total size. The actual size of a file system grows with the data stored on it. If the size of the data approaches the virtual size of the file system, Stratis grows the thin volume and the file system automatically.

The file systems are formatted with XFS.

IMPORTANT

Stratis tracks information about file systems created using Stratis that XFS is not aware of, and changes made using XFS do not automatically create updates in Stratis. Users must not reformat or reconfigure XFS file systems that are managed by Stratis.

Stratis creates links to file systems at the /**stratis**/***my-pool***/***my-fs*** path.

NOTE

Stratis uses many Device Mapper devices, which show up in **dmsetup** listings and the /**proc**/**partitions** file. Similarly, the **lsblk** command output reflects the internal workings and layers of Stratis.

12.2.2. Adding block devices to a Stratis pool

This procedure adds one or more block devices to a Stratis pool to be usable by Stratis file systems.

Prerequisites

- Stratis is installed. See Section 12.1.4, “Installing Stratis”.
- The **stratisd** service is running.
- The block devices that you are adding to the Stratis pool are not in use and not mounted.
- The block devices that you are adding to the Stratis pool are at least 1 GiB in size each.

Procedure

- To add one or more block devices to the pool, use:

```
# stratis pool add-data my-pool device-1 device-2 device-n
```

Additional resources

- The **stratis(8)** man page

12.2.3. Related information

- The *Stratis Storage* website: https://stratis-storage.github.io/

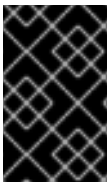
12.3. MONITORING STRATIS FILE SYSTEMS

As a Stratis user, you can view information about Stratis volumes on your system to monitor their state and free space.

12.3.1. Stratis sizes reported by different utilities

This section explains the difference between Stratis sizes reported by standard utilities such as **df** and the **stratis** utility.

Standard Linux utilities such as **df** report the size of the XFS file system layer on Stratis, which is 1 TiB. This is not useful information, because the actual storage usage of Stratis is less due to thin provisioning, and also because Stratis automatically grows the file system when the XFS layer is close to full.



IMPORTANT

Regularly monitor the amount of data written to your Stratis file systems, which is reported as the *Total Physical Used* value. Make sure it does not exceed the *Total Physical Size* value.

Additional resources

- The **stratis(8)** man page

12.3.2. Displaying information about Stratis volumes

This procedure lists statistics about your Stratis volumes, such as the total, used, and free size or file systems and block devices belonging to a pool.

Prerequisites

- Stratis is installed. See Section 12.1.4, "Installing Stratis".
- The **stratisd** service is running.

Procedure

- To display information about all **block devices** used for Stratis on your system:

```
# stratis blockdev

Pool Name  Device Node    Physical Size   State  Tier
my-pool    /dev/sdb           9.10 TiB  In-use  Data
```

- To display information about all Stratis **pools** on your system:

```
# stratis pool

Name    Total Physical Size  Total Physical Used
my-pool          9.10 TiB             598 MiB
```

- To display information about all Stratis **file systems** on your system:

```
# stratis filesystem

Pool Name  Name  Used     Created            Device
my-pool    my-fs 546 MiB  Nov 08 2018 08:03  /stratis/my-pool/my-fs
```

Additional resources

- The **stratis(8)** man page

12.3.3. Related information

- The *Stratis Storage* website: https://stratis-storage.github.io/

12.4. USING SNAPSHOTS ON STRATIS FILE SYSTEMS

You can use snapshots on Stratis file systems to capture file system state at arbitrary times and restore it in the future.

12.4.1. Characteristics of Stratis snapshots

This section describes the properties and limitations of file system snapshots on Stratis.

In Stratis, a snapshot is a regular Stratis file system created as a copy of another Stratis file system. The snapshot initially contains the same file content as the original file system, but can change as the snapshot is modified. Whatever changes you make to the snapshot will not be reflected in the original file system.

The current snapshot implementation in Stratis is characterized by the following:

- A snapshot of a file system is another file system.
- A snapshot and its origin are not linked in lifetime. A snapshotted file system can live longer than the file system it was created from.
- A file system does not have to be mounted to create a snapshot from it.
- Each snapshot uses around half a gigabyte of actual backing storage, which is needed for the XFS log.

12.4.2. Creating a Stratis snapshot

This procedure creates a Stratis file system as a snapshot of an existing Stratis file system.

Prerequisites

- Stratis is installed. See Section 12.1.4, “Installing Stratis”.
- The **stratisd** service is running.
- You have created a Stratis file system. See Section 12.1.6, “Creating a Stratis file system” .

Procedure

- To create a Stratis snapshot, use:

```
# stratis fs snapshot my-pool my-fs my-fs-snapshot
```

Additional resources

- The **stratis(8)** man page

12.4.3. Accessing the content of a Stratis snapshot

This procedure mounts a snapshot of a Stratis file system to make it accessible for read and write operations.

Prerequisites

- Stratis is installed. See Section 12.1.4, "Installing Stratis".
- The **stratisd** service is running.
- You have created a Stratis snapshot. See Section 12.4.2, "Creating a Stratis snapshot" .

Procedure

- To access the snapshot, mount it as a regular file system from the **/stratis/*my-pool*/** directory:

```
# mount /stratis/my-pool/my-fs-snapshot mount-point
```

Additional resources

- Section 12.1.7, "Mounting a Stratis file system"
- The **mount(8)** man page

12.4.4. Reverting a Stratis file system to a previous snapshot

This procedure reverts the content of a Stratis file system to the state captured in a Stratis snapshot.

Prerequisites

- Stratis is installed. See Section 12.1.4, "Installing Stratis".
- The **stratisd** service is running.
- You have created a Stratis snapshot. See Section 12.4.2, "Creating a Stratis snapshot" .

Procedure

1. Optionally, back up the current state of the file system to be able to access it later:

```
# stratis filesystem snapshot my-pool my-fs my-fs-backup
```

2. Unmount and remove the original file system:

```
# umount /stratis/my-pool/my-fs
# stratis filesystem destroy my-pool my-fs
```

3. Create a copy of the snapshot under the name of the original file system:

```
# stratis filesystem snapshot my-pool my-fs-snapshot my-fs
```

4. Mount the snapshot, which is now accessible with the same name as the original file system:

```
# mount /stratis/my-pool/my-fs mount-point
```

The content of the file system named *my-fs* is now identical to the snapshot *my-fs-snapshot*.

Additional resources

- The **stratis(8)** man page

12.4.5. Removing a Stratis snapshot

This procedure removes a Stratis snapshot from a pool. Data on the snapshot are lost.

Prerequisites

- Stratis is installed. See Section 12.1.4, "Installing Stratis".
- The **stratisd** service is running.
- You have created a Stratis snapshot. See Section 12.4.2, "Creating a Stratis snapshot" .

Procedure

1. Unmount the snapshot:

```
# umount /stratis/my-pool/my-fs-snapshot
```

2. Destroy the snapshot:

```
# stratis filesystem destroy my-pool my-fs-snapshot
```

Additional resources

- The **stratis(8)** man page

12.4.6. Related information

- The *Stratis Storage* website: https://stratis-storage.github.io/

12.5. REMOVING STRATIS FILE SYSTEMS

You can remove an existing Stratis file system or a Stratis pool, destroying data on them.

12.5.1. Components of a Stratis volume

Externally, Stratis presents the following volume components in the command-line interface and the API:

blockdev

Block devices, such as a disk or a disk partition.

pool

Composed of one or more block devices.
A pool has a fixed total size, equal to the size of the block devices.

The pool contains most Stratis layers, such as the non-volatile data cache using the **dm-cache** target.

Stratis creates a /**stratis**/***my-pool***/ directory for each pool. This directory contains links to devices that represent Stratis file systems in the pool.

filesystem

Each pool can contain one or more file systems, which store files.
File systems are thinly provisioned and do not have a fixed total size. The actual size of a file system grows with the data stored on it. If the size of the data approaches the virtual size of the file system, Stratis grows the thin volume and the file system automatically.

The file systems are formatted with XFS.

IMPORTANT

Stratis tracks information about file systems created using Stratis that XFS is not aware of, and changes made using XFS do not automatically create updates in Stratis. Users must not reformat or reconfigure XFS file systems that are managed by Stratis.

Stratis creates links to file systems at the /**stratis**/***my-pool***/***my-fs*** path.

NOTE

Stratis uses many Device Mapper devices, which show up in **dmsetup** listings and the /**proc**/**partitions** file. Similarly, the **lsblk** command output reflects the internal workings and layers of Stratis.

12.5.2. Removing a Stratis file system

This procedure removes an existing Stratis file system. Data stored on it are lost.

Prerequisites

- Stratis is installed. See Section 12.1.4, “Installing Stratis”.
- The **stratisd** service is running.
- You have created a Stratis file system. See Section 12.1.6, “Creating a Stratis file system” .

Procedure

1. Unmount the file system:

```
# umount /stratis/my-pool/my-fs
```

2. Destroy the file system:

```
# stratis filesystem destroy my-pool my-fs
```

3. Verify that the file system no longer exists:

```
# stratis filesystem list my-pool
```

Additional resources

- The **stratis(8)** man page

12.5.3. Removing a Stratis pool

This procedure removes an existing Stratis pool. Data stored on it are lost.

Prerequisites

- Stratis is installed. See Section 12.1.4, "Installing Stratis".
- The **stratisd** service is running.
- You have created a Stratis pool. See Section 12.1.5, "Creating a Stratis pool".

Procedure

1. List file systems on the pool:

```
# stratis filesystem list my-pool
```

2. Unmount all file systems on the pool:

```
# umount /stratis/my-pool/my-fs-1 \
        /stratis/my-pool/my-fs-2 \
        /stratis/my-pool/my-fs-n
```

3. Destroy the file systems:

```
# stratis filesystem destroy my-pool my-fs-1 my-fs-2
```

4. Destroy the pool:

```
# stratis pool destroy my-pool
```

5. Verify that the pool no longer exists:

```
# stratis pool list
```

Additional resources

- The **stratis(8)** man page

12.5.4. Related information

- The *Stratis Storage* website: https://stratis-storage.github.io/

CHAPTER 13. GETTING STARTED WITH AN EXT3 FILE SYSTEM

As a system administrator, you can create, mount, resize, backup, and restore an ext3 file system. The ext3 file system is essentially an enhanced version of the ext2 file system.

13.1. FEATURES OF AN EXT3 FILE SYSTEM

Following are the features of an ext3 file system:

- Availability: After an unexpected power failure or system crash, file system check is not required due to the journaling provided. The default journal size takes about a second to recover, depending on the speed of the hardware

NOTE

The only supported journaling mode in ext3 is **data=ordered** (default). For more information, see Is the EXT journaling option "data=writeback" supported in RHEL? Knowledgebase article.

- Data Integrity: The ext3 file system prevents loss of data integrity during an unexpected power failure or system crash.
- Speed: Despite writing some data more than once, ext3 has a higher throughput in most cases than ext2 because ext3's journaling optimizes hard drive head motion.
- Easy Transition: It is easy to migrate from ext2 to ext3 and gain the benefits of a robust journaling file system without reformatting.

Additional resources

- The **ext3** man page.

13.2. CREATING AN EXT3 FILE SYSTEM

As a system administrator, you can create an ext3 file system on a block device using **mkfs.ext3** command.

Prerequisites

- A partition on your disk. For information on creating MBR or GPT partitions, see Section 8.2, "Creating a partition table on a disk".
  Alternatively, use an LVM or MD volume.

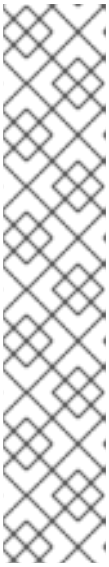
Procedure

1. To create an ext3 file system:
   - For a regular-partition device, an LVM volume, an MD volume, or a similar device, use the following command:

     ```
     # mkfs.ext3 /dev/block_device
     ```

     Replace /dev/*block_device* with the path to a block device.

For example, **/dev/sdb1**, **/dev/disk/by-uuid/05e99ec8-def1-4a5e-8a9d-5945339ceb2a**, or **/dev/my-volgroup/my-lv**. In general, the default options are optimal for most usage scenarios.



NOTE

- To specify a UUID when creating a file system:

```
# mkfs.ext3 -U UUID /dev/block_device
```

Replace *UUID* with the UUID you want to set: for example, **7cd65de3-e0be-41d9-b66d-96d749c02da7**.

Replace */dev/block_device* with the path to an ext3 file system to have the UUID added to it: for example, **/dev/sda8**.

- To specify a label when creating a file system:

```
# mkfs.ext3 -L label-name /dev/block_device
```

2. To view the created ext3 file system:

```
# blkid
```

Additional resources

- The **ext3** man page.
- The **mkfs.ext3** man page.

13.3. MOUNTING AN EXT3 FILE SYSTEM

As a system administrator, you can mount an ext3 file system using the **mount** utility.

Prerequisites

- An ext3 file system. For information on creating an ext3 file system, see Section 13.2, “Creating an ext3 file system”.

Procedure

1. To create a mount point to mount the file system:

```
# mkdir /mount/point
```

Replace */mount/point* with the directory name where mount point of the partition must be created.

2. To mount an ext3 file system:
   - To mount an ext3 file system with no extra options:

```
# mount /dev/block_device /mount/point
```

- To mount the file system persistently, see Section 10.8, "Persistently mounting file systems".

3. To view the mounted file system:

```
# df -h
```

Additional resources

- The **mount** man page.
- The **ext3** man page.
- The **fstab** man page.
- Chapter 10, *Mounting file systems*

13.4. RESIZING AN EXT3 FILE SYSTEM

As a system administrator, you can resize an ext3 file system using the **resize2fs** utility. The **resize2fs** utility reads the size in units of file system block size, unless a suffix indicating a specific unit is used. The following suffixes indicate specific units:

- s (sectors) - **512** byte sectors
- K (kilobytes) - **1,024** bytes
- M (megabytes) - **1,048,576** bytes
- G (gigabytes) - **1,073,741,824** bytes
- T (terabytes) - **1,099,511,627,776** bytes

Prerequisites

- An ext3 file system. For information on creating an ext3 file system, see Section 13.2, "Creating an ext3 file system".
- An underlying block device of an appropriate size to hold the file system after resizing.

Procedure

1. To resize an ext3 file system, take the following steps:
   - To shrink and grow the size of an unmounted ext3 file system:

     ```
     # umount /dev/block_device
     # e2fsck -f /dev/block_device
     # resize2fs /dev/block_device size
     ```

     Replace */dev/block_device* with the path to the block device, for example **/dev/sdb1**.

     Replace *size* with the required resize value using **s**, **K**, **M**, **G**, and **T** suffixes.

   - An ext3 file system may be grown while mounted using the **resize2fs** command:

```
# resize2fs /mount/device size
```

NOTE

The size parameter is optional (and often redundant) when expanding. The **resize2fs** automatically expands to fill the available space of the container, usually a logical volume or partition.

2. To view the resized file system:

```
# df -h
```

Additional resources

- The **resize2fs** man page.
- The **e2fsck** man page.
- The **ext3** man page.

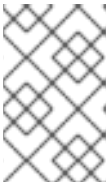
CHAPTER 14. GETTING STARTED WITH AN EXT4 FILE SYSTEM

As a system administrator, you can create, mount, resize, backup, and restore an ext4 file system. The ext4 file system is a scalable extension of the ext3 file system. With Red Hat Enterprise Linux 8, it can support a maximum individual file size of **16** terabytes, and file system to a maximum of **50** terabytes.

14.1. FEATURES OF AN EXT4 FILE SYSTEM

Following are the features of an ext4 file system:

- Using extents: The ext4 file system uses extents, which improves performance when using large files and reduces metadata overhead for large files.
- Ext4 labels unallocated block groups and inode table sections accordingly, which allows the block groups and table sections to be skipped during a file system check. It leads to a quick file system check, which becomes more beneficial as the file system grows in size.
- Metadata checksum: By default, this feature is enabled in Red Hat Enterprise Linux 8.
- Allocation features of an ext4 file system:
  - Persistent pre-allocation
  - Delayed allocation
  - Multi-block allocation
  - Stripe-aware allocation
- Extended attributes (**xattr**): This allows the system to associate several additional name and value pairs per file.
- Quota journaling: This avoids the need for lengthy quota consistency checks after a crash.



NOTE

The only supported journaling mode in ext4 is **data=ordered** (default). For more information, see Is the EXT journaling option "data=writeback" supported in RHEL? Knowledgebase article.

- Subsecond timestamps – This gives timestamps to the subsecond.

Additional resources

- The **ext4** man page.

14.2. CREATING AN EXT4 FILE SYSTEM

As a system administrator, you can create an ext4 file system on a block device using **mkfs.ext4** command.

Prerequisites

- A partition on your disk. For information on creating MBR or GPT partitions, see Section 8.2,

"Creating a partition table on a disk".
Alternatively, use an LVM or MD volume.

Procedure

1. To create an ext4 file system:

   - For a regular-partition device, an LVM volume, an MD volume, or a similar device, use the following command:

     ```
     # mkfs.ext4 /dev/block_device
     ```

     Replace /dev/*block_device* with the path to a block device.

     For example, **/dev/sdb1**, **/dev/disk/by-uuid/05e99ec8-def1-4a5e-8a9d-5945339ceb2a**, or **/dev/my-volgroup/my-lv**. In general, the default options are optimal for most usage scenarios.

   - For striped block devices (for example, RAID5 arrays), the stripe geometry can be specified at the time of file system creation. Using proper stripe geometry enhances the performance of an ext4 file system. For example, to create a file system with a 64k stride (that is, 16 x 4096) on a 4k-block file system, use the following command:

     ```
     # mkfs.ext4 -E stride=16,stripe-width=64 /dev/block_device
     ```

     In the given example:

     - stride=value: Specifies the RAID chunk size
     - stripe-width=value: Specifies the number of data disks in a RAID device, or the number of stripe units in the stripe.

NOTE

- To specify a UUID when creating a file system:

  ```
  # mkfs.ext4 -U UUID /dev/block_device
  ```

  Replace *UUID* with the UUID you want to set: for example, **7cd65de3-e0be-41d9-b66d-96d749c02da7**.

  Replace /dev/*block_device* with the path to an ext4 file system to have the UUID added to it: for example, **/dev/sda8**.

- To specify a label when creating a file system:

  ```
  # mkfs.ext4 -L label-name /dev/block_device
  ```

2. To view the created ext4 file system:

   ```
   # blkid
   ```

Additional resources

- The **ext4** man page.

- The **mkfs.ext4** man page.

14.3. MOUNTING AN EXT4 FILE SYSTEM

As a system administrator, you can mount an ext4 file system using the **mount** utility.

Prerequisites

- An ext4 file system. For information on creating an ext4 file system, see Section 14.2, "Creating an ext4 file system".

Procedure

1. To create a mount point to mount the file system:

   ```
   # mkdir /mount/point
   ```

   Replace */mount/point* with the directory name where mount point of the partition must be created.

2. To mount an ext4 file system:

   - To mount an ext4 file system with no extra options:

     ```
     # mount /dev/block_device /mount/point
     ```

   - To mount the file system persistently, see Section 10.8, "Persistently mounting file systems".

3. To view the mounted file system:

   ```
   # df -h
   ```

Additional resources

- The **mount** man page.
- The **ext4** man page.
- The **fstab** man page.
- Chapter 10, *Mounting file systems*

14.4. RESIZING AN EXT4 FILE SYSTEM

As a system administrator, you can resize an ext4 file system using the **resize2fs** utility. The **resize2fs** utility reads the size in units of file system block size, unless a suffix indicating a specific unit is used. The following suffixes indicate specific units:

- s (sectors) - **512** byte sectors
- K (kilobytes) - **1,024** bytes
- M (megabytes) - **1,048,576** bytes

- G (gigabytes) - **1,073,741,824** bytes
- T (terabytes) - **1,099,511,627,776** bytes

Prerequisites

- An ext4 file system. For information on creating an ext4 file system, see Section 14.2, "Creating an ext4 file system".
- An underlying block device of an appropriate size to hold the file system after resizing.

Procedure

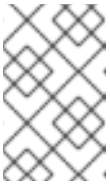
1. To resize an ext4 file system, take the following steps:
   - To shrink and grow the size of an unmounted ext4 file system:

     ```
     # umount /dev/block_device
     # e2fsck -f /dev/block_device
     # resize2fs /dev/block_device size
     ```

     Replace */dev/block_device* with the path to the block device, for example **/dev/sdb1**.

     Replace *size* with the required resize value using **s**, **K**, **M**, **G**, and **T** suffixes.
   - An ext4 file system may be grown while mounted using the **resize2fs** command:

     ```
     # resize2fs /mount/device size
     ```

     NOTE

     The size parameter is optional (and often redundant) when expanding. The **resize2fs** automatically expands to fill the available space of the container, usually a logical volume or partition.
2. To view the resized file system:

   ```
   # df -h
   ```

Additional resources

- The **resize2fs** man page.
- The **e2fsck** man page.
- The **ext4** man page.