

---

# **CernVM-FS Documentation**

*Release 2.4.0*

**CernVM Team**

**Jul 04, 2017**



---

# Contents

---

<b>1</b>	<b>What is CernVM-FS?</b>	<b>1</b>
<b>2</b>	<b>Contents</b>	<b>3</b>
2.1	Release Notes for CernVM-FS 2.4 . . . . .	3
2.2	Overview . . . . .	3
2.3	Getting Started . . . . .	4
2.4	Client Configuration . . . . .	7
2.5	Setting up a Local Squid Proxy . . . . .	16
2.6	CernVM-FS on Supercomputers . . . . .	17
2.7	Creating a Repository (Stratum 0) . . . . .	19
2.8	Setting up a Replica Server (Stratum 1) . . . . .	34
2.9	CernVM-FS Gateway Services and Release Managers . . . . .	36
2.10	CernVM-FS Server Meta Information . . . . .	38
2.11	Client Plug-Ins . . . . .	40
2.12	Implementation Notes . . . . .	41
<b>3</b>	<b>Additional Information</b>	<b>51</b>
3.1	Security Considerations . . . . .	51
3.2	CernVM-FS Parameters . . . . .	52
3.3	CernVM-FS Server Infrastructure . . . . .	55
3.4	Available RPMs . . . . .	58
3.5	References . . . . .	59
<b>4</b>	<b>Contact and Authors</b>	<b>61</b>
	<b>Bibliography</b>	<b>63</b>



---

## What is CernVM-FS?

---

The CernVM-File System (CernVM-FS) provides a scalable, reliable and low- maintenance software distribution service. It was developed to assist High Energy Physics (HEP) collaborations to deploy software on the worldwide-distributed computing infrastructure used to run data processing applications. CernVM-FS is implemented as a POSIX read-only file system in user space (a FUSE module). Files and directories are hosted on standard web servers and mounted in the universal namespace `/cvmfs`. Internally, CernVM-FS uses content-addressable storage and Merkle trees in order to maintain file data and meta-data. CernVM-FS uses outgoing HTTP connections only, thereby it avoids most of the firewall issues of other network file systems. It transfers data and meta-data on demand and verifies data integrity by cryptographic hashes.

By means of aggressive caching and reduction of latency, CernVM-FS focuses specifically on the software use case. Software usually comprises many small files that are frequently opened and read as a whole. Furthermore, the software use case includes frequent look-ups for files in multiple directories when search paths are examined.

CernVM-FS is actively used by small and large HEP collaborations. In many cases, it replaces package managers and shared software areas on cluster file systems as means to distribute the software used to process experiment data.



## Release Notes for CernVM-FS 2.4

### Overview

The CernVM File System (CernVM-FS) is a read-only file system designed to deliver scientific software onto virtual machines and physical worker nodes in a fast, scalable, and reliable way. Files and file metadata are downloaded on demand and aggressively cached. For the distribution of files, CernVM-FS uses a standard HTTP [\[BernersLee96\]](#) [\[Fielding99\]](#) transport, which allows exploitation of a variety of web caches, including commercial content delivery networks. CernVM-FS ensures data authenticity and integrity over these possibly untrusted caches and connections. The CernVM-FS software comprises client-side software to mount “CernVM-FS repositories” (similar to AFS volumes) as well as a server-side toolkit to create such distributable CernVM-FS repositories.

Fig. 2.1: A CernVM-FS client provides a virtual file system that loads data only on access. In this example, all releases of a software package (such as an HEP experiment framework) are hosted as a CernVM-FS repository on a web server.

The first implementation of CernVM-FS was based on `grow-fs` [\[Compostella10\]](#) [\[Thain05\]](#), which was originally provided as one of the private file system options available in Parrot. Ever since the design evolved and diverged, taking into account the works on HTTP- Fuse [\[Suzaki06\]](#) and content-delivery networks [\[Freedman03\]](#) [\[Nygren10\]](#) [\[Tolia03\]](#). Its current implementation provides the following key features:

- Use of the the `Fuse kernel module` that comes with in-kernel caching of file attributes
- Cache quota management
- Use of a content addressable storage format resulting in immutable files and automatic file de-duplication
- Possibility to split a directory hierarchy into sub catalogs at user-defined levels
- Automatic updates of file catalogs controlled by a time to live stored inside file catalogs
- Digitally signed repositories
- Transparent file compression/decompression and transparent file chunking

- Capability to work in offline mode providing that all required files are cached
- File system versioning
- File system hotpatching
- Dynamic expansion of environment variables embedded in symbolic links
- Support for extended attributes, such as file capabilities and SELinux attributes
- Automatic mirror server selection based on geographic proximity
- Automatic load-balancing of proxy servers
- Support for WPAD/PAC auto-configuration of proxy servers
- Efficient replication of repositories
- Possibility to use S3 compatible storage instead of a file system as repository storage

In contrast to general purpose network file systems such as `nfs` or `afs`, CernVM-FS is particularly crafted for fast and scalable software distribution. Running and compiling software is a use case general purpose distributed file systems are not optimized for. In contrast to virtual machine images or Docker images, software installed in CernVM-FS does not need to be further packaged. Instead it is distributed and versioned file-by-file. In order to create and update a CernVM-FS repository, a distinguished machine, the so-called *Release Manager Machine*, is used. On such a release manager machine, a CernVM-FS repository is mounted in read/write mode by means of a union file system [Wright04]. The union file system overlays the CernVM-FS read-only mount point by a writable scratch area. The CernVM-FS server tool kit merges changes written to the scratch area into the CernVM-FS repository. Merging and publishing changes can be triggered at user-defined points in time; it is an atomic operation. As such, a CernVM-FS repository is similar to a repository in the sense of a versioning system.

On the client, only data and metadata of the software releases that are actually used are downloaded and cached.

Fig. 2.2: Opening a file on CernVM-FS. CernVM-FS resolves the name by means of an SQLite catalog. Downloaded files are verified against the cryptographic hash of the corresponding catalog entry. The `stat()` system call can be entirely served from the in-kernel file system buffers.

## Getting Started

This section describes how to install the CernVM-FS client. The CernVM-FS client is supported on x86, x86\_64, and ARMv7 architectures running Scientific Linux 4-7, Ubuntu  $\geq 12.04$ , SLES 11 and openSUSE 13.1, Fedora 22 and 23, or Mac OS X  $\geq 10.10$ . There is experimental support for AArch64 and Power 8.

## Getting the Software

The CernVM-FS source code and binary packages are available [on our website](#). Binary packages are produced for rpm, dpkg, and Mac OS X (.pkg). Packages for 64 bit and 32 bit Scientific Linux 5 and 6 and 64 bit Scientific Linux 7 are available as a [yum repository](#). The `cvmfs-release` packages can be used to add a these yum/apt repositories to the local yum installation. The `cvmfs-release` packages are available on [our download page](#).

The CernVM-FS client is not relocatable and needs to be installed under `/usr`. On Intel architectures, it needs a `gcc`  $\geq 4.2$  compiler, on ARMv7 a `gcc`  $\geq 4.7$  compiler. In order to compile and install from sources, use the following cmake command:



```
cmake .
make
sudo make install
```

## Installation

### Linux

To install, proceed according to the following steps:

**Step 1** Install the CernVM-FS packages. With yum, run

```
sudo yum install cvmfs cvmfs-config-default
```

If yum does not show the latest packages, clean the yum cache by `sudo yum clean all`. Packages can be also installed with rpm instead with the command `rpm -vi`. On Ubuntu, use

```
sudo apt-get install cvmfs cvmfs-config-default
```

If apt does not show the latest packages, run `sudo apt-get update` before. Packages can be also installed with dpkg instead with the command `dpkg -i`.

**Step 2** For the base setup, run `cvmfs_config setup`. Alternatively, you can do the base setup by hand: ensure that `user_allow_other` is set in `/etc/fuse.conf`, ensure that `/cvmfs /etc/auto.cvmfs` is set in `/etc/auto.master` and that the autofs service is running. If the autofs service is systemd managed, ensure that the unit's "kill mode" is set to `process`. If you migrate from a previous version of CernVM-FS, check the release notes if there is anything special to do for migration.

**Step 3** Create `/etc/cvmfs/default.local` and open the file for editing.

**Step 4** Select the desired repositories by setting `CVMFS_REPOSITORIES=repo1,repo2,...`. For ATLAS, for instance, set

```
CVMFS_REPOSITORIES=atlas.cern.ch,atlas-condb.cern.ch,grid.cern.ch
```

Specify the HTTP proxy servers on your site with

```
CVMFS_HTTP_PROXY="http://myproxy1:port|http://myproxy2:port"
```

For the syntax of more complex HTTP proxy settings, see *Network Settings*. Make sure your local proxy servers allow access to all the Stratum 1 web servers (more on *proxy configuration here*). For Cern repositories, the Stratum 1 web servers are listed in `/etc/cvmfs/domain.d/cern.ch.conf`.

**Step 5** Check if CernVM-FS mounts the specified repositories by `cvmfs_config probe`. If the probe fails, try to restart autofs with `sudo service autofs restart`.

### Mac OS X

On Mac OS X, CernVM-FS is based on [OSXFuse](#). It is not integrated with autofs. In order to install, proceed according to the following steps:

**Step 1** Install the CernVM-FS package by opening the .pkg file.

**Step 2** Create `/etc/cvmfs/default.local` and open the file for editing.

**Step 3** Select the desired repositories by setting `CVMFS_REPOSITORIES=repo1,repo2,...`. For CMS, for instance, set

```
CVMFS_REPOSITORIES=cms.cern.ch
```

Specify the HTTP proxy servers on your site with

```
CVMFS_HTTP_PROXY="http://myproxy1:port|http://myproxy2:port"
```

If you're unsure about the proxy names, set `CVMFS_HTTP_PROXY=DIRECT`.

**Step 4** Mount your repositories like

```
sudo mkdir -p /cvmfs/cms.cern.ch
sudo mount -t cvmfs cms.cern.ch /cvmfs/cms.cern.ch
```

## Usage

The CernVM-FS repositories are located under `/cvmfs`. Each repository is identified by a *fully qualified repository name*. The fully qualified repository name consists of a repository identifier and a domain name, similar to DNS records [Mockapetris87]. The domain part of the fully qualified repository name indicates the location of repository creation and maintenance. For the ATLAS experiment software, for instance, the fully qualified repository name is `atlas.cern.ch` although the hosting web servers are spread around the world.

Mounting and un-mounting of the CernVM-FS is controlled by `autofs` and `automount`. That is, starting from the base directory `/cvmfs` different repositories are mounted automatically just by accessing them. For instance, running the command `ls /cvmfs/atlas.cern.ch` will mount the ATLAS software repository on the fly. This directory gets automatically unmounted after some `automount`-defined idle time.

## Debugging Hints

In order to check for common misconfigurations in the base setup, run

```
cvmfs_config chksetup
```

CernVM-FS gathers its configuration parameter from various configuration files that can overwrite each others settings (default configuration, domain specific configuration, local setup, ...). To show the effective configuration for `repository.cern.ch`, run

```
cvmfs_config showconfig repository.cern.ch
```

In order to exclude `autofs`/`automounter` as a source of problems, you can try to mount `repository.cern.ch` manually by

```
mkdir -p /mnt/cvmfs
mount -t cvmfs repository.cern.ch /mnt/cvmfs
```

In order to exclude SELinux as a source of problems, you can try mounting after SELinux has been disabled by

```
/usr/sbin/setenforce 0
```

Once you sorted out a problem, make sure that you do not get the original error served from the file system buffers by

```
service autofs restart
```

In case you need additional assistance, please don't hesitate to contact us at [cernvm.support@cern.ch](mailto:cernvm.support@cern.ch). Together with the problem description, please send the system information tarball created by `cvmfs_config bugreport`.

## Client Configuration

### Structure of `/etc/cvmfs`

The local configuration of CernVM-FS is controlled by several files in `/etc/cvmfs` listed in the table below. For every `.conf` file except for the files in `/etc/cvmfs/default.d` you can create a corresponding `.local` file having the same prefix in order to customize the configuration. The `.local` file will be sourced after the corresponding `.conf` file.

In a typical installation, a handful of parameters need to be set in `/etc/cvmfs/default.local`. Most likely, this is the list of repositories (`CVMFS_REPOSITORIES`), HTTP proxies (see `:ref:sct_network'`), and perhaps the cache directory and the cache quota (see `:ref:sct_cache'`). In a few cases, one might change a parameter for a specific domain or a specific repository, provide an exclusive cache for a specific repository (see `:ref:sct_cache'`). For a list of all parameters, see Appendix “*Client parameters*”.

The `.conf` and `.local` configuration files are key-value pairs in the form `PARAMETER=value`. They are sourced by `/bin/sh`. Hence, a limited set of shell commands can be used inside these files including comments, `if` clauses, parameter evaluation, and shell math (`$((...))`). Special characters have to be quoted. For instance, instead of `CVMFS_HTTP_PROXY=p1;p2`, write `CVMFS_HTTP_PROXY='p1;p2'` in order to avoid parsing errors. The shell commands in the configuration files can use the `CVMFS_FQFN` parameter, which contains the fully qualified repository names that is being mounted. The current working directory is set to the parent directory of the configuration file at hand.

File	Purpose
<code>config.sh</code>	Set of internal helper functions.
<code>default.conf</code>	Set of base parameters.
<code>default.d/ \$config.conf</code>	Adjustments to the <code>default.conf</code> configuration, usually installed by a <code>cvmfs-config-...</code> package. Read before <code>default.local</code> .
<code>domain.d/\$domain. conf</code>	Domain-specific parameters and implementations of the functions in <code>config.sh</code>
<code>config.d/ \$repository.conf</code>	Repository-specific parameters and implementations of the functions in <code>config.sh</code>
<code>keys/</code>	Contains domain-specific sub directories with public keys used to verify the digital signature of file catalogs

### The “Config Repository”

In addition to the local system configuration, a client can configure a dedicated “config repository”. A config repository is a standard mountable CernVM-FS repository that resembles the directory structure of `/etc/cvmfs`. It can be used to centrally maintain the public keys and configuration of repositories that should not be distributed with rather static packages. Configuration from the config repository is overwritten by the local configuration in case of conflicts. The config repository is set by the `CVMFS_CONFIG_REPOSITORY` parameter. The default configuration sets this parameter to `cvmfs-config.cern.ch`.

The `CVMFS_CONFIG_REPO_REQUIRED` parameter can be used to force availability of the config repository in order for other repositories to get mounted.

## Mounting

Mounting of CernVM-FS repositories is typically handled by `autofs`. Just by accessing a repository directory under `/cvmfs` (`/cvmfs/atlas.cern.ch`), `autofs` will take care of mounting. `autofs` will also automatically unmount a repository if it is not used for a while.

Instead of using `autofs`, CernVM-FS repositories can be mounted manually with the system’s `mount` command. In order to do so, use the `cvmfs` file system type, like

```
mount -t cvmfs atlas.cern.ch /cvmfs/atlas.cern.ch
```

Likewise, CernVM-FS repositories can be mounted through entries in `/etc/fstab`. A sample entry in `/etc/fstab`:

```
atlas.cern.ch /mnt/test cvmfs defaults,_netdev,nodev 0 0
```

Every mount point corresponds to a CernVM-FS process. Using `autofs` or the system's `mount` command, every repository can only be mounted once. Otherwise multiple CernVM-FS processes would collide in the same cache location. If a repository is needed under several paths, use a *bind mount* or use a *private file system mount point*.

### Private Mount Points

In contrast to the system's `mount` command which requires root privileges, CernVM-FS can also be mounted like other Fuse file systems by normal users. In this case, CernVM-FS uses parameters from one or several user-provided config files instead of using the files under `/etc/cvmfs`. CernVM-FS private mount points do not appear as `cvmfs2` file systems but as `fuse` file systems. The `cvmfs_config` and `cvmfs_talk` commands ignore privately mounted CernVM-FS repositories. On an interactive machine, private mount points are for instance unaffected by an administrator unmounting all system's CernVM-FS mount points by `cvmfs_config umount`.

In order to mount CernVM-FS privately, use the `cvmfs2` command like

```
cvmfs2 -o config=myparams.conf atlas.cern.ch /home/user/myatlas
```

A minimal sample `myparams.conf` file could look like this:

```
CVMFS_CACHE_BASE=/home/user/mycache
CVMFS_CLAIM_OWNERSHIP=yes
CVMFS_RELOAD_SOCKETS=/home/user/mycache
CVMFS_SERVER_URL=http://cvmfs-stratum-one.cern.ch/cvmfs/atlas.cern.ch
CVMFS_HTTP_PROXY=DIRECT
```

Make sure to use absolute path names for the mount point and for the cache directory. Use `fusermount -u` in order to unmount a privately mounted CernVM-FS repository.

The private mount points can also be used to use the CernVM-FS Fuse module in case it has not been installed under `/usr` and `/etc`. If the public keys are not installed under `/etc/cvmfs/keys`, the directory of the keys needs to be specified in the config file by `CVMFS_KEYS_DIR=<directory>`. If the `libcvmfs_fuse.so` library is not installed in one of the standard search paths, the `LD_LIBRARY_PATH` variable has to be set accordingly for the `cvmfs2` command.

### Docker Containers

There are two options to mount CernVM-FS in docker containers. The first option is to bind mount a mounted repository as a volume into the container. This has the advantage that the CernVM-FS cache is shared among multiple containers. The second option is to mount a repository inside a container, which requires a *privileged* container.

### Volume Driver

There is an [external package](#) that provides a Docker Volume Driver for CernVM-FS. This package provides management of repositories in Docker and Kubernetes. It provides a convenient interface to handle CernVM-FS volume definitions.

## Bind mount from the host

On Docker  $\geq$  1.10, the autofs managed area /cvmfs can be directly mounted into the container as a shared mount point like

```
docker run -it -v /cvmfs:/cvmfs:shared centos /bin/bash
```

In order to bind mount an individual repository from the host, turn off autofs on the host and mount the repository manually, like:

```
service autofs stop # systemd: systemctl stop autofs
chkconfig autofs off # systemd: systemctl disable autofs
mkdir -p /cvmfs/sft.cern.ch
mount -t cvmfs sft.cern.ch /cvmfs/sft.cern.ch
```

Start the docker container with the `-v` option to mount the CernVM-FS repository inside, like

```
docker run -it -v /cvmfs/sft.cern.ch:/cvmfs/sft.cern.ch centos /bin/bash
```

The `-v` option can be used multiple times with different repositories.

## Mount inside a container

In order to use mount inside a container, the container must be started in privileged mode, like

```
docker run --privileged -i -t centos /bin/bash
```

In such a container, CernVM-FS can be installed and used the usual way provided that autofs is turned off.

## Parrot Connector to CernVM-FS

In case Fuse cannot be installed, the `parrot toolkit` provides a means to “mount” CernVM-FS on Linux in pure user space. Parrot sandboxes an application in a similar way `gdb` sandboxes an application. But instead of debugging the application, parrot transparently rewrites file system calls and can effectively provide /cvmfs to an application. We recommend to use the `latest precompiled parrot`, which has CernVM-FS support built-in.

In order to sandbox a command `<CMD>` with options `<OPTIONS>` in parrot, use

```
export PARROT_ALLOW_SWITCHING_CVMFS_REPOSITORIES=yes
export PARROT_CVMFS_REPO="<default-repositories>"
export HTTP_PROXY='<SITE HTTP PROXY>' # or 'DIRECT;' if not on a cluster or grid site
parrot_run <PARROT_OPTIONS> <CMD> <OPTIONS>
```

Repositories that are not available by default from the builtin `<default-repositories>` list can be explicitly added to `PARROT_CVMFS_REPO`. The repository name, a stratum 1 URL, and the public key of the repository need to be provided. For instance, in order to add `alice-ocdb.cern.ch` and `ilc.desy.de` to the list of repositories, one can write

```
export CERN_S1="http://cvmfs-stratum-one.cern.ch/cvmfs"
export DESY_S1="http://grid-cvmfs-one.desy.de:8000/cvmfs"
export PARROT_CVMFS_REPO="<default-repositories> \
  alice-ocdb.cern.ch:url=${CERN_S1}/alice-ocdb.cern.ch,pubkey=<PATH/key.pub> \
  ilc.desy.de:url=${DESY_S1}/ilc.desy.de,pubkey=<PATH/key.pub>"
```

given that the repository public keys are in the provided paths.

By default, parrot uses a shared CernVM-FS cache for all parrot instances of the same user stored under a temporary directory that is derived from the user id. In order to place the CernVM-FS cache into a different directory, use

```
export PARROT_CVMFS_ALIEN_CACHE=</path/to/cache>
```

In order to share this directory among multiple users, the users have to belong to the same UNIX group.

## Network Settings

CernVM-FS uses HTTP for the data transfer. Repository data can be replicated to multiple web servers and cached by standard web proxies such as Squid [Guerrero99]. In a typical setup, repositories are replicated to a handful of web servers in different locations. These replicas form the CernVM-FS Stratum 1 service, whereas the replication source server is the CernVM-FS Stratum 0 server. In every cluster of client machines, there should be two or more web proxy servers that CernVM-FS can use (see *Setting up a Local Squid Proxy*). These site-local web proxies reduce the network latency for the CernVM-FS clients and they reduce the load for the Stratum 1 service. CernVM-FS supports WPAD/PAC proxy auto configuration [Gauthier99], choosing a random proxy for load-balancing, and automatic fail-over to other hosts and proxies in case of network errors. Roaming clients can connect directly to the Stratum 1 service.

### IP Protocol Version

CernVM-FS can use both IPv4 and IPv6. For dual-stack stratum 1 hosts it will use the system default settings when connecting directly to the host. When connecting to a proxy, by default it will try on the IPv4 address unless the proxy only has IPv6 addresses configured. The `CVMFS_IPFAMILY_PREFER=[4|6]` parameter can be used to select the preferred IP protocol for dual-stack proxies.

### Stratum 1 List

To specify the Stratum 1 servers, set `CVMFS_SERVER_URL` to a semicolon-separated list of known replica servers (enclose in quotes). The so defined URLs are organized as a ring buffer. Whenever download of files fails from a server, CernVM-FS automatically switches to the next mirror server. For repositories under the `cern.ch` domain, the Stratum 1 servers are specified in `/etc/cvmfs/domain.d/cern.ch.conf`.

It is recommended to adjust the order of Stratum 1 servers so that the closest servers are used with priority. This can be done automatically by *using geographic ordering*. Alternatively, for roaming clients (clients not using a proxy server), the Stratum 1 servers can be automatically sorted according to round trip time by `cvmfs_talk host probe` (see *Auxiliary Tools*). Otherwise, the proxy server would invalidate round trip time measurement.

The special sequence `@fqrn@` in the `CVMFS_SERVER_URL` string is replaced by fully qualified repository name (`atlas.cern.cn`, `cms.cern.ch`, ...). That allows to use the same parameter for many repositories hosted under the same domain. For instance, `http://cvmfs-stratum-one.cern.ch/cvmfs/@fqrn@` can resolve to `http://cvmfs-stratum-one.cern.ch/cvmfs/atlas.cern.ch`, `http://cvmfs-stratum-one.cern.ch/cvmfs/cms.cern.ch`, and so on depending on the repository that is being mounted. The same works for the sequence `@org@` which is replaced by the unqualified repository name (`atlas`, `cms`, ...).

### Proxy Lists

CernVM-FS uses a dedicated HTTP proxy configuration, independent from system-wide settings. Instead of a single proxy, CernVM-FS uses a *chain of load-balanced proxy groups*. The CernVM-FS proxies are set by the `CVMFS_HTTP_PROXY` parameter.

Proxies within the same proxy group are considered as a load-balance group and a proxy is selected randomly. If a proxy fails, CernVM-FS automatically switches to another proxy from the current group. If all proxies from a group

have failed, CernVM-FS switches to the next proxy group. After probing the last proxy group in the chain, the first proxy is probed again. To avoid endless loops, for each file download the number of switches is restricted by the total number of proxies.

The chain of proxy groups is specified by a string of semicolon separated entries, each group is a list of pipe separated hostnames<sup>1</sup>. Multiple IP addresses behind a single proxy host name (DNS *round-robin* entry) are automatically transformed into a load-balanced group. In order to limit the number of proxy servers used from a round-robin DNS entry, set `CVMFS_MAX_IPADDR_PER_PROXY`. This also limits the perceived “hang duration” while CernVM-FS performs fail-overs.

The `DIRECT` keyword for a hostname avoids using proxies. Note that the `CVMFS_HTTP_PROXY` parameter is necessary in order to mount. If you don’t use proxies, set the parameter to `DIRECT`.

Multiple proxy groups are often organized as a primary proxy group at the local site and backup proxy groups at remote sites. In order to avoid CernVM-FS being stuck with proxies at a remote site after a fail-over, CernVM-FS will automatically retry to use proxies from the primary group after some time. The delay for re-trying a proxies from the primary group is set in seconds by `CVMFS_PROXY_RESET_AFTER`. The distinction of primary and backup proxy groups can be turned off by setting this parameter to 0.

## Automatic Proxy Configuration

The proxy settings can be automatically gathered through WPAD. The special proxy server “auto” in `CVMFS_HTTP_PROXY` is resolved according to the proxy server specification loaded from a PAC file. PAC files can be on a file system or accessible via HTTP. CernVM-FS looks for PAC files in the order given by the semicolon separated URLs in the `CVMFS_PAC_URLS` environment variable. This variable defaults to <http://wpad/wpad.dat>. The `auto` keyword used as a URL in `CVMFS_PAC_URLS` is resolved to <http://wpad/wpad.dat>, too, in order to be compatible with Frontier [*Blumenfeld08*].

## Fallback Proxy List

In addition to the regular proxy list set by `CVMFS_HTTP_PROXY`, a fallback proxy list is supported in `CVMFS_FALLBACK_PROXY`. The syntax of both lists is the same. The fallback proxy list is appended to the regular proxy list, and if the fallback proxy list is set, any `DIRECT` is removed from both lists. The automatic proxy configuration of the previous section only sets the regular proxy list, not the fallback proxy list. Also the fallback proxy list can be automatically reordered; see the next section.

## Ordering of Servers according to Geographic Proximity

CernVM-FS Stratum 1 servers provide a RESTful service for geographic ordering. Clients can request `http://<HOST>/cvmfs/<FQDN>/api/v1.0/geo/<proxy_address>/<server_list>` The proxy address can be replaced by a UUID if no proxies are used, and the CernVM-FS client does that if there are no regular proxies. The server list is comma-separated. The result is an ordered list of indexes of the input host names. Use of this API can be enabled in a CernVM-FS client with `CVMFS_USE_GEOAPI=yes`. That will geographically sort both the servers set by `CVMFS_SERVER_URL` and the fallback proxies set by `CVMFS_FALLBACK_PROXY`.

## Timeouts

CernVM-FS tries to gracefully recover from broken network links and temporarily overloaded paths. The timeout for connection attempts and for very slow downloads can be set by `CVMFS_TIMEOUT` and `CVMFS_TIMEOUT_DIRECT`. The two timeout parameters apply to a connection with a proxy server and to a direct connection to a Stratum 1 server, respectively. A download is considered to be “very slow” if the transfer rate is below for more than the timeout interval.

<sup>1</sup> The usual proxy notation rules apply, like `http://proxy1:8080|http://proxy2:8080;DIRECT`

The threshold can be adjusted with the `CVMFS_LOW_SPEED_LIMIT` parameter. A very slow download is treated like a broken connection.

On timeout errors and on connection failures (but not on name resolving failures), CernVM-FS will retry the path using an exponential backoff. This introduces a jitter in case there are many concurrent requests by a cluster of nodes, allowing a proxy server or web server to serve all the nodes consecutively. `CVMFS_MAX_RETRIES` sets the number of retries on a given path before CernVM-FS tries to switch to another proxy or host. The overall number of requests with a given proxy/host combination is `$CVMFS_MAX_RETRIES+1`. `CVMFS_BACKOFF_INIT` sets the maximum initial backoff in seconds. The actual initial backoff is picked with milliseconds precision randomly in the interval `[1, $CVMFS_BACKOFF_INIT · 1000]`. With every retry, the backoff is then doubled.

## Cache Settings

Downloaded files will be stored in a local cache directory. The CernVM-FS cache has a soft quota; as a safety margin, the partition hosting the cache should provide more space than the soft quota limit. Once the quota limit is reached, CernVM-FS will automatically remove files from the cache according to the least recently used policy [*Panagiotou06*]. Removal of files is performed bunch-wise until half of the maximum cache size has been freed. The quota limit can be set in Megabytes by `CVMFS_QUOTA_LIMIT`. For typical repositories, a few Gigabytes make a good quota limit. For repositories hosted at cern, quota recommendations can be found under <http://cernvm.cern.ch/portal/cvmfs/examples>.

The cache directory needs to be on a local file system in order to allow each host the accurate accounting of the cache contents; on a network file system, the cache can potentially be modified by other hosts. Furthermore, the cache directory is used to create (transient) sockets and pipes, which is usually only supported by a local file system. The location of the cache directory can be set by `CVMFS_CACHE_BASE`.

On SELinux enabled systems, the cache directory and its content need to be labeled as `cvmfs_cache_t`. During the installation of CernVM-FS RPMs, this label is set for the default cache directory `/var/lib/cvmfs`. For other directories, the label needs to be set manually by `chcon -Rv --type=cvmfs_cache_t $CVMFS_CACHE_BASE`.

Each repository can either have an exclusive cache or join the CernVM-FS shared cache. The shared cache enforces a common quota for all repositories used on the host. File duplicates across repositories are stored only once in the shared cache. The quota limit of the shared directory should be at least the maximum of the recommended limits of its participating repositories. In order to have a repository not join the shared cache but use an exclusive cache, set `CVMFS_SHARED_CACHE=no`.

## Alien Cache

An “alien cache” provides the possibility to use a data cache outside the control of CernVM-FS. This can be necessary, for instance, in HPC environments where local disk space is not available or scarce but powerful cluster file systems are available. The alien cache directory is a directory in addition to the ordinary cache directory. The ordinary cache directory is still used to store control files.

The alien cache directory is set by the `CVMFS_ALIEN_CACHE` option. It can be located anywhere including cluster and network file systems. If configured, all data chunks are stored there. CernVM-FS ensures atomic access to the cache directory. It is safe to have the alien directory shared by multiple CernVM-FS processes and it is safe to unlink files from the alien cache directory anytime. The contents of files, however, must not be touched by third-party programs.

In contrast to normal cache mode where files are store in mode 0600, in the alien cache files are stored in mode 0660. So all users being part of the alien cache directory’s owner group can use it.

The skeleton of the alien cache directory should be created upfront. Otherwise, the first CernVM-FS process accessing the alien cache determines the ownership. The `cvmfs2` binary can create such a skeleton using

```
cvmfs2 __MK_ALIEN_CACHE__ $alien_cachedir $owner_uid $owner_gid
```



Since the alien cache is unmanaged, there is no automatic quota management provided by CernVM-FS; the alien cache directory is ever-growing. The `CVMFS_ALIEN_CACHE` requires `CVMFS_QUOTA_LIMIT=-1` and `CVMFS_SHARED_CACHE=no`.

The alien cache might be used in combination with a special repository replication mode that preloads a cache directory (Section *Setting up a Replica Server (Stratum 1)*). This allows to propagate an entire repository into the cache of a cluster file system for HPC setups that do not allow outgoing connectivity.

## NFS Server Mode

In case there is no local hard disk space available on a cluster of worker nodes, a single CernVM-FS client can be exported via nfs [*Callaghan95*] [*Shepler03*] to these worker nodes. This mode of deployment will inevitably introduce a performance bottleneck and a single point of failure and should be only used if necessary.

NSF export requires Linux kernel  $\geq 2.6.27$  on the NFS server. For instance, exporting works for Scientific Linux 6 but not for Scientific Linux 5. The NFS server should run a lock server as well. For proper NFS support, set `CVMFS_NFS_SOURCE=yes`. On the client side, all available nfs implementations should work.

In the NFS mode, upon mount an additionally directory `nfs_maps.$repository_name` appears in the CernVM-FS cache directory. These *NFS maps* use `leveldb` to store the virtual inode CernVM-FS issues for any accessed path. The virtual inode may be requested by NFS clients anytime later. As the NFS server has no control over the lifetime of client caches, entries in the NFS maps cannot be removed.

Typically, every entry in the NFS maps requires some 150-200 Bytes. A recursive `find` on `/cvmfs/atlas.cern.ch` with 50 million entries, for instance, would add up 8GB in the cache directory. For a CernVM-FS instance that is exported via NFS, the safety margin for the NFS maps needs be taken into account. It also might be necessary to monitor the actual space consumption.

## Tuning

The default settings in CernVM-FS are tailored to the normal, non-NFS use case. For decent performance in the NFS deployment, the amount of memory given to the meta-data cache should be increased. By default, this is 16M. It can be increased, for instance, to 256M by setting `CVMFS_MEMCACHE_SIZE` to 256. Furthermore, the maximum number of download retries should be increased to at least 2.

The number of NFS daemons should be increased as well. A value of 128 NFS daemons has shown perform well. In Scientific Linux, the number of NFS daemons is set by the `RPCNFSDCOUNT` parameter in `/etc/sysconfig/nfs`.

The performance will benefit from large RAM on the NFS server ( $\geq 16GB$ ) and CernVM-FS caches hosted on an SSD hard drive.

## Shared NFS Maps (HA-NFS)

As an alternative to the existing, `leveldb` managed NFS maps, the NFS maps can optionally be managed out of the CernVM-FS cache directory by `SQLite`. This allows the NFS maps to be placed on shared storage and accessed by multiple CernVM-FS NFS export nodes simultaneously for clustering and active high-availability setups. In order to enable shared NFS maps, set `CVMFS_NFS_SHARED` to the path that should be used to host the `SQLite` database. If the path is on shared storage, the shared storage has to support `POSIX` file locks. The drawback of the `SQLite` managed NFS maps is a significant performance penalty which in practice can be covered by the memory caches.

## Example

An example entry `/etc/exports` (note: the `fsid` needs to be different for every exported CernVM-FS repository)

```
/cvmfs/atlas.cern.ch 172.16.192.0/24 (ro, sync, no_root_squash, \
no_subtree_check, fsid=101)
```

A sample entry `/etc/fstab` entry on a client:

```
172.16.192.210:/cvmfs/atlas.cern.ch /cvmfs/atlas.cern.ch nfs4 \
ro,ac,actimeo=60,lookupcache=all,nolock,rsize=1048576,wsiz=1048576 0 0
```

## Hotpatching and Reloading

By hotpatching a running CernVM-FS instance, most of the code can be reloaded without unmounting the file system. The current active code is unloaded and the code from the currently installed binaries is loaded. Hotpatching is logged to `syslog`. Since CernVM-FS is re-initialized during hotpatching and configuration parameters are re-read, hotpatching can be also seen as a “reload”.

Hotpatching has to be done for all repositories concurrently by

```
cvmfs_config [-c] reload
```

The optional parameter `-c` specifies if the CernVM-FS cache should be wiped out during the hotpatch. Reloading of the parameters of a specific repository can be done like

```
cvmfs_config reload atlas.cern.ch
```

In order to see the history of loaded CernVM-FS Fuse modules, run

```
cvmfs_talk hotpatch history
```

The currently loaded set of parameters can be shown by

```
cvmfs_talk parameters
```

The CernVM-FS packages use hotpatching in the package upgrade process.

## Auxiliary Tools

### `cvmfs_fsck`

CernVM-FS assumes that the local cache directory is trustworthy. However, it might happen that files get corrupted in the cache directory caused by errors outside the scope of CernVM-FS. CernVM-FS stores files in the local disk cache with their cryptographic content hash key as name, which makes it easy to verify file integrity. CernVM-FS contains the `cvmfs_fsck` utility to do so for a specific cache directory. Its return value is comparable to the system’s `fsck`. For example,

```
cvmfs_fsck -j 8 /var/lib/cvmfs/shared
```

checks all the data files and catalogs in `/var/lib/cvmfs/shared` using 8 concurrent threads. Supported options are:

<code>-v</code>	Produce more verbose output.
<code>-j</code> <code>#threads</code>	Sets the number of concurrent threads that check files in the cache directory. Defaults to 4.
<code>-p</code>	Tries to automatically fix problems.
<code>-f</code>	Unlinks the cache database. The database will be automatically rebuilt by CernVM-FS on next mount.

The `cvmfs_config fsck` command can be used to verify all configured repositories.

## cvmfs\_config

The `cvmfs_config` utility provides commands in order to setup the system for use with CernVM-FS.

**setup** The `setup` command takes care of basic setup tasks, such as creating the `cvmfs` user and allowing access to CernVM-FS mount points by all users.

**chksetup** The `chksetup` command inspects the system and the CernVM-FS configuration in `/etc/cvmfs` for common problems.

**showconfig** The `showconfig` command prints the CernVM-FS parameters for all repositories or for the specific repository given as argument.

**stat** The `stat` command prints file system and network statistics for currently mounted repositories.

**status** The `status` command shows all currently mounted repositories and the process id (PID) of the CernVM-FS processes managing a mount point.

**probe** The `probe` command tries to access `/cvmfs/$repository` for all repositories specified in `CVMFS_REPOSITORIES` or the ones specified as a space separated list on the command line, respectively.

**fsck** Run `cvmfs_fsck` on all repositories specified in `CVMFS_REPOSITORIES`.

**reload** The `reload` command is used to *reload or hotpatch CernVM-FS instances*.

**umount** The `umount` command unmounts all currently mounted CernVM-FS repositories, which will only succeed if there are no open file handles on the repositories.

**wipecache** The `wipecache` command is an alias for `reload -c`.

**killall** The `killall` command immediately unmounts all repositories under `/cvmfs` and terminates the associated processes. It is meant to escape from a hung state without the need to reboot a machine. However, all processes that use CernVM-FS at the time will be terminated, too. The need to use this command very likely points to a network problem or a bug in `cvmfs`.

**bugreport** The `bugreport` command creates a tarball with collected system information which helps to *debug a problem*.

## cvmfs\_talk

The `cvmfs_talk` command provides a way to control a currently running CernVM-FS process and to extract information about the status of the corresponding mount point. Most of the commands are for special purposes only or covered by more convenient commands, such as `cvmfs_config showconfig` or `cvmfs_config stat`. Three commands might be of particular interest though.

```
cvmfs_talk cleanup 0
```

will, without interruption of service, immediately cleanup the cache from all files that are not currently pinned in the cache.

```
cvmfs_talk cleanup rate 120
```

shows the number of cache cleanups in the last two hours (120 minutes). If this value is larger than one or two, the cache size is probably too small and the client experiences cache thrashing.

```
cvmfs_talk internal affairs
```

prints the internal status information and performance counters. It can be helpful for performance engineering.

### Other

Information about the current cache usage can be gathered using the `df` utility. For repositories created with the CernVM-FS 2.1 toolchain, information about the overall number of file system entries in the repository as well as the number of entries covered by currently loaded meta-data can be gathered by `df -i`.

For the Nagios monitoring system [Schubert08], a checker plugin is available on our website.

### Debug Logs

The `cvmfs2` binary forks a watchdog process on start. Using this watchdog, CernVM-FS is able to create a stack trace in case certain signals (such as a segmentation fault) are received. The watchdog writes the stack trace into `syslog` as well as into a file `stacktrace` in the cache directory.

In addition to *these debugging hints*, CernVM-FS can be started in debug mode. In the debug mode, CernVM-FS will log with high verbosity which makes the debug mode unsuitable for production use. In order to turn on the debug mode, set `CVMFS_DEBUGFILE=/tmp/cvmfs.log`.

## Setting up a Local Squid Proxy

For clusters of nodes with CernVM-FS clients, we strongly recommend to setup two or more Squid forward proxy servers as well. The forward proxies will reduce the latency for the local worker nodes, which is critical for cold cache performance. They also reduce the load on the Stratum 1 servers.

From what we have seen, a Squid server on commodity hardware scales well for at least a couple of hundred worker nodes. The more RAM and hard disk you can devote for caching the better. We have good experience with of memory cache and of hard disk cache. We suggest to setup two identical Squid servers for reliability and load-balancing. Assuming the two servers are A and B, set

```
CVMFS_HTTP_PROXY="http://A:3128|http://B:3128"
```

Squid is very powerful and has lots of configuration and tuning options. For CernVM-FS we require only the very basic static content caching. If you already have a Frontier Squid [Blumenfeld08] [Dykstra10] installed you can use it as well for CernVM-FS.

Otherwise, cache sizes and access control needs to be configured in order to use the Squid server with CernVM-FS. In order to do so, browse through your `/etc/squid/squid.conf` and make sure the following lines appear accordingly:

```
minimum_expiry_time 0

max_filedesc 8192
maximum_object_size 1024 MB

cache_mem 128 MB
maximum_object_size_in_memory 128 KB
# 50 GB disk cache
cache_dir ufs /var/spool/squid 50000 16 256
```

Furthermore, Squid needs to allow access to all Stratum 1 servers. This is controlled through Squid ACLs. For the Stratum 1 servers for the cern.ch, egi.eu, and opensciencegrid.org domains, add the following lines to you Squid configuration:

```
acl cvmfs dst cvmfs-stratum-one.cern.ch
acl cvmfs dst cernvmfs.gridpp.rl.ac.uk
acl cvmfs dst cvmfs.racf.bnl.gov
acl cvmfs dst cvmfs02.grid.sinica.edu.tw
acl cvmfs dst cvmfs.fnal.gov
acl cvmfs dst cvmfs-atlas-nightlies.cern.ch
acl cvmfs dst cvmfs-egi.gridpp.rl.ac.uk
acl cvmfs dst klei.nikhef.nl
acl cvmfs dst cvmfsrepo.lcg.triumf.ca
acl cvmfs dst cvmfsrep.grid.sinica.edu.tw
acl cvmfs dst cvmfs-s1bnl.opensciencegrid.org
acl cvmfs dst cvmfs-s1fnal.opensciencegrid.org
http_access allow cvmfs
```

The Squid configuration can be verified by `squid -k parse`. Before the first service start, the cache space on the hard disk needs to be prepared by `squid -z`. In order to make the increased number of file descriptors effective for Squid, execute `ulimit -n 8192` prior to starting the squid service.

## CernVM-FS on Supercomputers

There are several characteristics in which supercomputers can differ from other nodes with respect to CernVM-FS

1. Fuse is not allowed on the individual nodes
2. Individual nodes do not have Internet connectivity
3. Nodes have no local hard disk to store the CernVM-FS cache

These problems can be overcome as described in the following sections.

### Parrot-Mounted CernVM-FS in lieu of Fuse Module

Instead of accessing `/cvmfs` through a Fuse module, processes can use the [Parrot connector](#). The parrot connector works on x86\_64 Linux if the `ptrace` system call is not disabled. In contrast to a plain copy of a CernVM-FS repository to a shared file system, this approach has the following advantages:

- Millions of synchronized meta-data operations per node (path lookups, in particular) will not drown the shared cluster file system but resolve locally in the parrot-cvmfs clients.
- The file system is always consistent; applications never see half-synchronized directories.
- After initial preloading, only change sets need to be transferred to the shared file system. This is much faster than `rsync`, which always has to browse the entire name space.
- Identical files are internally de-duplicated. While space of the order of terabytes is usually not an issue for HPC shared file systems, file system caches benefit from deduplication. It is also possible to preload only specific parts of a repository namespace.
- Support for extra functionality implemented by CernVM-FS such as versioning and variant symlinks (symlinks resolved according to environment variables).

## Preloading the CernVM-FS Cache

The `cvmfs_preload` utility can be used to preload a CernVM-FS cache onto the shared cluster file system. Internally it uses the same code that is used to replicate between CernVM-FS stratum 0 and stratum 1. The `cvmfs_preload` command is a self-extracting binary with no further dependencies and should work on a majority of x86\_64 Linux hosts.

The `cvmfs_preload` command replicates from a stratum 0 (not from a stratum 1). Because this induces significant load on the source server, stratum 0 administrators should be informed before using their server as a source. As an example, in order to preload the ALICE repository into `/shared/cache`, one could run from a login node

```
cvmfs_preload -u http://cvmfs-stratum-zero-hpc.cern.ch:8000/cvmfs/alice.cern.ch -r /  
↳shared/cache
```

This will preload the entire repository. In order to preload only specific parts of the namespace, you can create a `_dirtab_` file with path prefixes. The path prefixes must not involve symbolic links. An example dirtab file for ALICE could look like

```
/example/etc  
/example/x86_64-2.6-gnu-4.8.3/Modules  
/example/x86_64-2.6-gnu-4.8.3/Packages/GEANT3  
/example/x86_64-2.6-gnu-4.8.3/Packages/ROOT  
/example/x86_64-2.6-gnu-4.8.3/Packages/gcc  
/example/x86_64-2.6-gnu-4.8.3/Packages/AliRoot/v5*
```

The corresponding invocation of `cvmfs_preload` is

```
cvmfs_preload -u http://cvmfs-stratum-zero-hpc.cern.ch:8000/cvmfs/alice.cern.ch -r /  
↳shared/cache \  
-d </path/to/dirtab>
```

The initial preloading can take several hours to a few days. Subsequent invocations of the same command only transfer a change set and typically finish within seconds or minutes. These subsequent invocations need to be either done manually when necessary or scheduled for instance with a cron job.

The `cvmfs_preload` command can preload files from multiple repositories into the same cache directory.

## Access from the Nodes

In order to access a preloaded cache from the nodes, set the path to the directory as an *Alien Cache*. Since there won't be cache misses, `parrot` or `fuse` clients do not need to download additional files from the network.

If clients do have network access, they might find a repository version online that is newer than the preloaded version in the cache. This results in conflicts with `cvmfs_preload` or in errors if the cache directory is read-only. Therefore, we recommend to explicitly disable network access for the `parrot` process on the nodes, for instance by setting

```
HTTP_PROXY='INVALID-PROXY'
```

before the invocation of `parrot_run`.

## Compiling `cvmfs_preload` from Sources

In order to compile `cvmfs_preload` from sources, use the `-DBUILD_PRELOADER=yes` `cmake` option.

## Loopback File Systems for Nodes' Caches

If nodes have Internet access but no local hard disk, it is preferable to provide the CernVM-FS caches as loopback file systems on the cluster file system. This way, CernVM-FS automatically populates the cache with the latest upstream content. A Fuse mounted CernVM-FS will also automatically manage the cache quota.

This approach requires a separate file for every node (not every mountpoint) on the cluster file system. The file should be 15% larger than the configured CernVM-FS cache size on the nodes, and it should be formatted with an ext3/4 or an xfs file system. These files can be created with the `dd` and `mkfs` utilities. Nodes can mount these files as loopback file systems from the shared file system.

Because there is only a single file for every node, the parallelism of the cluster file system can be exploited and all the requests from CernVM-FS circumvent the cluster file system's meta-data server(s).

## Creating a Repository (Stratum 0)

CernVM-FS is a file system with a single source of (new) data. This single source, the repository *Stratum 0*, is maintained by a dedicated *release manager machine* or *installation box*. A read-writable copy of the repository is accessible on the release manager machine. The CernVM-FS server tool kit is used to *publish* the current state of the repository on the release manager machine. Publishing is an atomic operation.

All data stored in CernVM-FS have to be converted into a CernVM-FS *repository* during the process of publishing. The CernVM-FS repository is a form of content-addressable storage. Conversion includes creating the file catalog(s), compressing new and updated files and calculating content hashes. Storing the data in a content-addressable format results in automatic file de-duplication. It furthermore simplifies data verification and it allows for file system snapshots.

In order to provide a writable CernVM-FS repository, CernVM-FS uses a union file system that combines a read-only CernVM-FS mount point with a writable scratch area [*Wright04*]. *This figure below* outlines the process of publishing a repository.

## CernVM-FS Server Quick-Start Guide

### System Requirements

- Apache HTTP server *or* S3 compatible storage service
- union file system in the kernel
  - AUFS (see *Installing the AUFS-enabled Kernel on Scientific Linux 6*)
  - OverlayFS (as of kernel version 4.2.x)
- Officially supported platforms
  - Scientific Linux 5 (64 bit)
  - Scientific Linux 6 (64 bit - with custom AUFS enabled kernel - Appendix “*Available RPMs*”)
  - CentOS/SL  $\geq 7.3$ , provided that `/var/spool/cvmfs` is served by an ext4 file system.
  - Fedora 22 and above (with kernel  $\geq 4.2.x$ )
  - Ubuntu 12.04 64 bit and above
    - \* Ubuntu  $< 15.10$ : with installed AUFS kernel module (cf. *linux-image-extra* package)
    - \* Ubuntu 15.10 and later (using upstream OverlayFS)

### Installation

1. Install `cvmfs` and `cvmfs-server` packages
2. Ensure enough disk space in `/var/spool/cvmfs` (>50GiB)
3. For local storage: Ensure enough disk space in `/srv/cvmfs`
4. Create a repository with `cvmfs_server mkfs` (See *Repository Creation*)

### Content Publishing

1. `cvmfs_server transaction <repository name>`
2. Install content into `/cvmfs/<repository name>`
3. Create nested catalogs at proper locations
  - Create `.cvmfscatalog` files (See *Managing Nested Catalogs*) or
  - Consider using a `.cvmfsdirtab` file (See *Managing Nested Catalogs with .cvmfsdirtab*)
4. `cvmfs_server publish <repository name>`

### Backup Policy

- Create backups of signing key files in `/etc/cvmfs/keys`
- Entire repository content
  - For local storage: `/srv/cvmfs`
  - Stratum 1s can serve as last-ressort backup of repository content

## Installing the AUFS-enabled Kernel on Scientific Linux 6

CernVM-FS uses the union file-system `aufs` to efficiently determine file-system tree updates while publishing repository transactions on the server (see Figure *below*). Note that this is *only* required on a CernVM-FS server and *not* on the client machines.

We provide customised kernel packages for Scientific Linux 6 (see Appendix “*Available RPMs*”) and keep them up-to-date with upstream kernel updates. The kernel RPMs are published in the `cernvm-kernel` yum repository. Please follow these steps to install the provided customised kernel:

1. Download the latest `cvmfs-release` package from [the CernVM website](#)
2. Install the `cvmfs-release` package: `yum install cvmfs-release*.rpm`  
This adds the CernVM yum repositories to your machine’s configuration.
3. Install the `aufs` enabled kernel from `cernvm-kernel`:  
`yum --disablerepo=* --enablerepo=cernvm-kernel install kernel`
4. Install the `aufs` user utilities:  
`yum --enablerepo=cernvm-kernel install aufs2-util`
5. Reboot the machine



Once a new kernel version is released `yum update` will *not* pick the upstream version but it will wait until the patched kernel with aufs support is published by the CernVM team. We always try to follow the kernel updates as quickly as possible.

## Publishing a new Repository Revision

Fig. 2.3: Updating a mounted CernVM-FS repository by overlaying it with a copy-on-write union file system volume. Any changes will be accumulated in a writable volume (yellow) and can be synchronized into the CernVM-FS repository afterwards. The file catalog contains the directory structure as well as file metadata, symbolic links, and secure hash keys of regular files. Regular files are compressed and renamed to their cryptographic content hash before copied into the data store.

Since the repositories may contain many file system objects (i.e. ATLAS contains  $70 * 10^6$  file system objects – February 2016), we cannot afford to generate an entire repository from scratch for every update. Instead, we add a writable file system layer on top of a mounted read-only CernVM-FS repository using a union file system. This renders a read-only CernVM-FS mount point writable to the user, while all performed changes are stored in a special writable scratch area managed by the union file system. A similar approach is used by Linux Live Distributions that are shipped on read-only media, but allow *virtual* editing of files where changes are stored on a RAM disk.

If a file in the CernVM-FS repository gets changed, the union file system first copies it to the writable volume and applies any changes to this copy (copy-on-write semantics). Also newly created files or directories will be stored in the writable volume. Additionally the union file system creates special hidden files (called *white-outs*) to keep track of file deletions in the CernVM-FS repository.

Eventually, all changes applied to the repository are stored in this scratch area and can be merged into the actual CernVM-FS repository by a subsequent synchronization step. Up until the actual synchronization step takes place, no changes are applied to the CernVM-FS repository. Therefore, any unsuccessful updates to a repository can be rolled back by simply clearing the writable file system layer of the union file system.

## Requirements for a new Repository

In order to create a repository, the server and client part of CernVM-FS must be installed on the release manager machine. Furthermore you will need a kernel containing a union file system implementation as well as a running Apache2 web server. Currently we support Scientific Linux 6, Ubuntu 12.04+ and Fedora 22+ distributions. Please note, that Scientific Linux 6 *does not* ship with an aufs enabled kernel, therefore we provide a compatible patched kernel as RPMs (see *Installing the AUFS-enabled Kernel on Scientific Linux 6* for details).

Historically CernVM-FS solely used `aufs` as a union file system. However, the Linux kernel community favoured `OverlayFS`, a competing union file system implementation that was merged upstream.

Since CernVM-FS 2.2.0 we support the usage of both `OverlayFS` and `aufs`. Note however, that the first versions of `OverlayFS` were broken and will not work properly with CernVM-FS. At least a 4.2.x kernel is needed to use CernVM-FS with `OverlayFS`. (Red Hat) Enterprise Linux  $\geq 7.3$  works, too, provided that `/var/spool/cvmfs` is served by an ext4 file system. Furthermore note that `OverlayFS` cannot fully comply with POSIX semantics, in particular hard links must be broken into individual files. That is usually not a problem but should be kept in mind when installing certain software distributions into a CernVM-FS repository.

## Notable CernVM-FS Server Locations and Files

There are a number of possible customisations in the CernVM-FS server installation. The following table provides an overview of important configuration files and intrinsical paths together with some customisation hints. For an exhaustive description of the CernVM-FS server infrastructure please consult Appendix “*CernVM-FS Server Infrastructure*”.

File Path	Description
/cvmfs	<b>Repository mount points</b> Contains read-only union file system mountpoints that become writable during repository updates. Do not symlink or manually mount anything here.
/srv/cvmfs	<b>Central repository storage location</b> Can be mounted or symlinked to another location <i>before</i> creating the first repository.
/srv/cvmfs/ <fqrn>	<b>Storage location of a repository</b> Can be symlinked to another location <i>before</i> creating the repository <fqrn>.
/var/spool/cvmfs	<b>Internal states of repositories</b> Can be mounted or symlinked to another location <i>before</i> creating the first repository. Hosts the scratch area described <a href="#">here</a> , thus might consume notable disk space during repository updates.
/etc/cvmfs	<b>Configuration files and keychains</b> Similar to the structure described in <a href="#">this table</a> . Do not symlink this directory.
/etc/cvmfs/ cvmfs_server_hooks sh	<b>Customisable server behaviour</b> See “ <a href="#">Customizable Actions Using Server Hooks</a> ” for further details
/etc/cvmfs/ repositories.d	<b>Repository configuration location</b> Contains repository server specific configuration files.

## CernVM-FS Repository Creation and Updating

The CernVM-FS server tool kit provides the `cvmfs_server` utility in order to perform all operations related to repository creation, updating, deletion, replication and inspection. Without any parameters it prints a short documentation of its commands.

### Repository Creation

A new repository is created by `cvmfs_server mkfs`:

```
cvmfs_server mkfs my.repo.name
```

The utility will ask for a user that should act as the owner of the repository and afterwards create all the infrastructure for the new CernVM-FS repository. Additionally it will create a reasonable default configuration and generate a new release manager certificate and software signing key. The public key in `/etc/cvmfs/keys/my.repo.name.pub` needs to be distributed to all client machines.

The `cvmfs_server` utility will use `/srv/cvmfs` as storage location by default. In case a separate hard disk should be used, a partition can be mounted on `/srv/cvmfs` or `/srv/cvmfs` can be symlinked to another location (see [Notable CernVM-FS Server Locations and Files](#)). Besides local storage it is possible to use an *S3 compatible storage service* as data backend.

Once created, the repository is mounted under `/cvmfs/my.repo.name` containing only a single file called `new_repository`. The next steps describe how to change the repository content.

The repository name resembles a DNS scheme but it does not need to reflect any real server name. It is supposed to be a globally unique name that indicates where/who the publishing of content takes place. A repository name must only contain alphanumeric characters plus `-`, `_`, and `.` and it is limited to a length of 60 characters.

### Repositories for Volatile Files

Repositories can be flagged as containing *volatile* files using the `-v` option:

```
cvmfs_server mkfs -v my.repo.name
```

When CernVM-FS clients perform a cache cleanup, they treat files from volatile repositories with priority. Such volatile repositories can be useful, for instance, for experiment conditions data.

## Confidential Repositories

Repositories can be created with the `-V` options or republished with the `-F` option with a `membership` requirement. Clients that mount repositories with a membership requirement will grant or deny access to the repository based on the decision made by an authorization helper. See Section [Authorization Helpers](#) for details on authorization helpers.

For instance, a repository can be configured to grant access to a repository only to those users that have a X.509 certificate with a certain DN. Note that the corresponding client-side X.509 authorization helper is not part of CernVM-FS but is provided as a third-party plugin by the Open Science Grid.

A membership requirement makes most sense if the repository is served by an HTTPS server that requires client-side authentication. Note that such repositories cannot be replicated to Stratum 1 servers. Such repositories also cannot benefit from site proxies. Instead, such repositories are either part of a (non CernVM-FS) HTTPS content distribution network or they might be installed for a small number of users that, for example, require access to licensed software.

## S3 Compatible Storage Systems

CernVM-FS can store files directly to S3 compatible storage systems, such as Amazon S3, Huawei UDS and Open-Stack SWIFT. The S3 storage settings are given as parameters to `cvmfs_server mkfs` or `cvmfs_server add-replica`:

```
cvmfs_server mkfs -s /etc/cvmfs/.../mys3.conf \
-w http://s3.amazonaws.com/mybucket-1-1 my.repo.name
```

The file “mys3.conf” contains the S3 settings (see [:ref: table below <tab\\_s3confparameters>](#)). The “-w” option is used to define the S3 server URL, e.g. <http://localhost:3128>, which is used for accessing the repository’s backend storage on S3. Note that this URL can be different than the S3 server address that is used for uploads, e.g. if a proxy server is deployed in front of the server. Note that the buckets need to exist before the repository is created. In the example above, a single bucket `mybucket-1-1` needs to be created beforehand.

Parameter	Meaning
CVMFS_S3_ACCOUNTS	Number of S3 accounts to be used, e.g. 1. With some S3 servers use of multiple accounts can increase the upload speed significantly
CVMFS_S3_ACCESS_KEY	S3 account access key(s) separated with <code>:</code> , e.g. KEY-A:KEY-B:...
CVMFS_S3_SECRET_KEY	S3 account secret key(s) separated with <code>:</code> , e.g. KEY-A:KEY-B:...
CVMFS_S3_BUCKETS_PER_ACCOUNT	S3 buckets used per account, e.g. 1. With some S3 servers use of multiple buckets can increase the upload speed significantly
CVMFS_S3_HOST	S3 server hostname, e.g. s3.amazonaws.com
CVMFS_S3_BUCKET	S3 bucket base name. Account and bucket index are appended to the bucket base name. If you use just one account and one bucket, e.g. named <code>mybucket</code> , then you need to create only one bucket called <code>mybucket-1-1</code>
CVMFS_S3_MAX_NUMBER_OF_UPLOADS	Number of parallel uploads to the S3 server, e.g. 400

In addition, if the S3 backend is configured to use multiple accounts or buckets, a proxy server is needed to map HTTP requests to correct buckets. This mapping is needed because CernVM-FS does not support buckets but assumes that all files are stored in a flat namespace. The recommendation is to use a Squid proxy server (version  $\geq 3.1.10$ ). The `squid.conf` can look like this:

```
http_access allow all
http_port 127.0.0.1:3128 intercept
cache_peer swift.cern.ch parent 80 0 no-query originserver
url_rewrite_program /usr/bin/s3_squid_rewrite.py
cache deny all
```

The bucket mapping logic is implemented in `s3_squid_rewrite.py` file. This script is not provided by CernVM-FS but needs to be written by the repository owner (the CernVM-FS Git repository contains an example). The script needs to read requests from stdin and write mapped URLs to stdout, for instance:

```
in: http://localhost:3128/data/.cvmfswhitelist
out: http://swift.cern.ch/cernbucket-9-91/data/.cvmfswhitelist
```

### Repository Update

Typically a repository publisher does the following steps in order to create a new revision of a repository:

1. Run `cvmfs_server transaction` to switch to a copy-on-write enabled CernVM-FS volume
2. Make the necessary changes to the repository, add new directories, patch certain binaries, ...
3. Test the software installation
4. Do one of the following:
  - Run `cvmfs_server publish` to finalize the new repository revision *or*
  - Run `cvmfs_server abort` to clear all changes and start over again

CernVM-FS supports having more than one repository on a single server machine. In case of a multi-repository host, the target repository of a command needs to be given as a parameter when running the `cvmfs_server` utility. The `cvmfs_server resign` command should run every 30 days to update the signatures of the repository. Most `cvmfs_server` commands allow for wildcards to do manipulations on more than one repository at once, `cvmfs_server migrate *.cern.ch` would migrate all present repositories ending with `.cern.ch`.

### Variant Symlinks

It may be convenient to have a symlink in the repository resolve based on the CVMFS client configuration; this is called a *variant symlink*. For example, in the `oasis.opensciencegrid.org` repository, the OSG provides a default set of CAs at `/cvmfs/oasis.opensciencegrid.org/mis/certificates` but would like to give the sysadmin the ability to override this with their own set of CA certificates.

To setup a variant symlink in your repository, create a symlink as follows inside a repository transaction:

```
ln -s '${OSG_CERTIFICATES}' /cvmfs/oasis.opensciencegrid.org/mis/certificates
```

Here, the `certificates` symlink will evaluate to the value of the `OSG_CERTIFICATES` configuration variable in the client. If `OSG_CERTIFICATES` is not provided, the symlink resolution will be an empty string. To provide a server-side default value, you can instead do:

```
ln -s '${OSG_CERTIFICATES:-/cvmfs/oasis.opensciencegrid.org/mis/certificates-real}' /
↪cvmfs/oasis.opensciencegrid.org/mis/certificates
```

Here, the symlink will evaluate to `/cvmfs/oasis.opensciencegrid.org/mis/certificates-real` by default unless the sysadmin sets `OSG_CERTIFICATES` in a configuration file (such as `/etc/cvmfs/config.d/oasis.opensciencegrid.org.local`).

## Repository Import

The CernVM-FS server tools support the import of a CernVM-FS file storage together with its corresponding signing keychain. The import functionality is useful to bootstrap a release manager machine for a given file storage.

`cvmfs_server import` works similar to `cvmfs_server mkfs` (described in [Repository Creation](#)) except it uses the provided data storage instead of creating a fresh (and empty) storage. In case of a CernVM-FS 2.0 file storage `cvmfs_server import` also takes care of the file catalog migration into the latest catalog schema (see [Legacy Repository Import](#) for details).

During the import it might be necessary to resign the repository's whitelist. Usually because the whitelist's expiry date has exceeded. This operations requires the corresponding masterkey to be available in `/etc/cvmfs/keys`. Resigning is enabled by adding `-r` to `cvmfs_server import`.

An import can either use a provided repository keychain placed into `/etc/cvmfs/keys` or generate a fresh repository key and certificate for the imported repository. The latter case requires an update of the repository's whitelist to incorporate the newly generated repository key. To generate a fresh repository key add `-t -r` to `cvmfs_server import`.

Refer to Section [Repository Signature](#) for a comprehensive description of the repository signature mechanics.

## Legacy Repository Import

We strongly recommend to install CernVM-FS 2.1 on a fresh or at least a properly cleaned machine without any traces of the CernVM-FS 2.0 installation before installing CernVM-FS 2.1 server tools.

The command `cvmfs_server import` requires the full CernVM-FS 2.0 data storage which is located at `/srv/cvmfs` by default as well as the repository's signing keys. Since the CernVM-FS 2.1 server backend supports multiple repositories in contrast to its 2.0 counterpart, we recommend to move the repository's data storage to `/srv/cvmfs/<FQRN>` upfront to avoid later inconsistencies.

The following steps describe the transformation of a repository from CernVM-FS 2.0 into 2.1. As an example we are using a repository called **legacy.cern.ch**.

1. Make sure that you have backups of both the repository's backend storage and its signing keys
2. Install and test the CernVM-FS 2.1 server tools on the machine that is going to be used as new Stratum 0 maintenance machine
3. Place the repository's backend storage data in `/srv/cvmfs/legacy.cern.ch` (default storage location)
4. Transfer the repository's signing keychain to the machine (f.e. to `/legacy_keys/`)
5. Run `cvmfs_server import` like this:

```
cvmfs_server import
-o <username of repo maintainer> \
-k ~/legacy_keys \
-l          \ # for 2.0.x file catalog migration
-s          \ # for further repository statistics
legacy.cern.ch
```

6. Check the imported repository with `cvmfs_server check legacy.cern.ch` for integrity (see [Integrity Check](#))

## Customizable Actions Using Server Hooks

The `cvmfs_server` utility allows release managers to trigger custom actions before and after crucial repository manipulation steps. This can be useful for example for logging purposes, establishing backend storage connections automatically or other workflow triggers, depending on the application.

There are six designated server hooks that are potentially invoked during the *repository update procedure*:

- When running `cvmfs_server transaction`:
  - *before* the given repository is transitioned into transaction mode
  - *after* the transition was successful
- When running `cvmfs_server publish`:
  - *before* the publish procedure for the given repository is started
  - *after* it was published and remounted successfully
- When running `cvmfs_server abort`:
  - *before* the unpublished changes will be erased for the given repository
  - *after* the repository was successfully reverted to the last published state

All server hooks must be defined in a single shell script file called:

```
/etc/cvmfs/cvmfs_server_hooks.sh
```

The `cvmfs_server` utility will check the existence of this script and source it. To subscribe to the described hooks one needs to define one or more of the following shell script functions:

- `transaction_before_hook()`
- `transaction_after_hook()`
- `publish_before_hook()`
- `publish_after_hook()`
- `abort_before_hook()`
- `abort_after_hook()`

The defined functions get called at the specified positions in the repository update process and are provided with the fully qualified repository name as their only parameter (`$1`). Undefined functions automatically default to a NO-OP. An example script is located at `cvmfs/cvmfs_server_hooks.sh.demo` in the CernVM-FS sources.

## Maintaining a CernVM-FS Repository

CernVM-FS is a versioning, snapshot-based file system. Similar to versioning systems, changes to `/cvmfs/...` are temporary until they are committed (`cvmfs_server publish`) or discarded (`cvmfs_server abort`). That allows you to test and verify changes, for instance to test a newly installed release before publishing it to clients. Whenever changes are published (committed), a new file system snapshot of the current state is created. These file system snapshots can be tagged with a name, which makes them *named snapshots*. A named snapshot is meant to stay in the file system. One can rollback to named snapshots and it is possible, on the client side, to mount any of the named snapshots in lieu of the newest available snapshot.

Two named snapshots are managed automatically by CernVM-FS, `trunk` and `trunk-previous`. This allows for easy unpublishing of a mistake, by rolling back to the `trunk-previous` tag.

## Integrity Check

CernVM-FS provides an integrity checker for repositories. It is invoked by

```
cvmfs_server check
```

The integrity checker verifies the sanity of file catalogs and verifies that referenced data chunks are present. Ideally, the integrity checker is used after every publish operation. Where this is not affordable due to the size of the repositories, the integrity checker should run regularly.

The checker can also run on a nested catalog subtree. This is useful to follow up a specific issue where a check on the full tree would take a lot of time:

```
cvmfs_server check -s <path to nested catalog mountpoint>
```

Optionally `cvmfs_server check` can also verify the data integrity (command line flag `-i`) of each data object in the repository. This is a time consuming process and we recommend it only for diagnostic purposes.

## Named Snapshots

Named snapshots or *tags* are an easy way to organise checkpoints in the file system history. CernVM-FS clients can explicitly mount a repository at a specific named snapshot to expose the file system content published with this tag. It also allows for rollbacks to previously created and tagged file system revisions. Tag names need to be unique for each repository and are not allowed to contain spaces or special characters. Besides the actual tag's name they can also contain a free descriptive text and store a creation timestamp.

Named snapshots are best to use for larger modifications to the repository, for instance when a new major software release is installed. Named snapshots provide the ability to easily undo modifications and to preserve the state of the file system for the future. Nevertheless, named snapshots should not be used excessively. Less than 50 named snapshots are a good number of named snapshots in many cases.

## Automatically Generated Tags

By default, new repositories will automatically create a generic tag if no explicit tag is given during publish. The automatic tagging can be turned off using the `-g` option during repository creation or by setting `CVMFS_AUTO_TAG=false` in the `/etc/cvmfs/repositories.d/$repository/server.conf` file.

The life time of automatic tags can be restricted by the `CVMFS_AUTO_TAG_TIMESPAN` parameter or by the `-G` option to `cvmfs_server mkfs`. The parameter takes a string that the `date` utility can parse, for instance `"4 weeks ago"`. On every publish, automatically generated tags older than the defined threshold are removed.

## Creating a Named Snapshot

Tags can be added while publishing a new file system revision. To do so, the `-a` and `-m` options for `cvmfs_server publish` are used. The following command publishes a CernVM-FS revision with a new revision that is tagged as "release-1.0":

```
cvmfs_server transaction
# Changes
cvmfs_server publish -a release-1.0 -m "first stable release"
```

## Managing Existing Named Snapshots

Management of existing tags is done by using the `cvmfs_server tag` command. Without any command line parameters, it will print all currently available named snapshots. Snapshots can be inspected (`-i <tag name>`), removed (`-r <tag name>`) or created (`-a <tag name> -m <tag description> -h <catalog root hash>`). Furthermore machine readable modes for both listing (`-l -x`) as well as inspection (`-i <tag name> -x`) is available.

## Rollbacks

A repository can be rolled back to any of the named snapshots. Rolling back is achieved through the command `cvmfs_server rollback -t release-1.0`. A rollback is, like restoring from backups, not something one would do often. Use caution, a rollback is irreversible.

## Managing Nested Catalogs

CernVM-FS stores meta-data (path names, file sizes, ...) in file catalogs. When a client accesses a repository, it has to download the file catalog first and then it downloads the files as they are opened. A single file catalog for an entire repository can quickly become large and impractical. Also, clients typically do not need all of the repository's meta-data at the same time. For instance, clients using software release 1.0 do not need to know about the contents of software release 2.0.

With nested catalogs, CernVM-FS has a mechanism to partition the directory tree of a repository into many catalogs. Repository maintainers are responsible for sensible cutting of the directory trees into nested catalogs. They can do so by creating and removing magic files named `.cvmfscatalog`.

For example, in order to create a nested catalog for software release 1.0 in the hypothetical repository `experiment.cern.ch`, one would invoke

```
cvmfs_server transaction
touch /cvmfs/experiment.cern.ch/software/1.0/.cvmfscatalog
cvmfs_server publish
```

In order to merge a nested catalog with its parent catalog, the corresponding `.cvmfscatalog` file needs to be removed. Nested catalogs can be nested on arbitrary many levels.

## Recommendations for Nested Catalogs

Nested catalogs should be created having in mind which files and directories are accessed together. This is typically the case for software releases, but can be also on the directory level that separates platforms. For instance, for a directory layout like

```
/cvmfs/experiment.cern.ch
|- /software
|  |- /i686
|  |  |- 1.0
|  |  |- 2.0
|  |  `-- common
|  |- /x86_64
|  |  |- 1.0
|  |  `-- common
|- /grid-certificates
|- /scripts
```



it makes sense to have nested catalogs at

```
/cvmfs/experiment.cern.ch/software/i686
/cvmfs/experiment.cern.ch/software/x86_64
/cvmfs/experiment.cern.ch/software/i686/1.0
/cvmfs/experiment.cern.ch/software/i686/2.0
/cvmfs/experiment.cern.ch/software/x86_64/1.0
```

A nested catalog at the top level of each software package release is generally the best approach because once package releases are installed they tend to never change, which reduces churn and garbage generated in the repository from old catalogs that have changed. In addition, each run only tends to access one version of any package so having a separate catalog per version avoids loading catalog information that will not be used. A nested catalog at the top level of each platform may make sense if there is a significant number of platform-specific files that aren't included in other catalogs.

It could also make sense to have a nested catalog under grid-certificates, if the certificates are updated much more frequently than the other directories. It would not make sense to create a nested catalog under `/cvmfs/experiment.cern.ch/software/i686/common`, because this directory needs to be accessed anyway whenever its parent directory is needed. As a rule of thumb, a single file catalog should contain more than 1000 files and directories but not contain more than  $\approx 200000$  files. See *Inspecting Nested Catalog Structure* how to find catalogs that do not satisfy this recommendation.

Restructuring the repository's directory tree is an expensive operation in CernVM-FS. Moreover, it can easily break client applications when they switch to a restructured file system snapshot. Therefore, the software directory tree layout should be relatively stable before filling the CernVM-FS repository.

### Managing Nested Catalogs with `.cvmfsdirtab`

Rather than managing `.cvmfscatalog` files by hand, a repository administrator may create a file called `.cvmfsdirtab`, in the top directory of the repository, which contains a list of paths relative to the top of the repository where `.cvmfscatalog` files will be created. Those paths may contain shell wildcards such as asterisk (\*) and question mark (?). This is useful for specifying patterns for creating nested catalogs as new files are installed. A very good use of the patterns is to identify directories where software releases will be installed.

In addition, lines in `.cvmfsdirtab` that begin with an exclamation point (!) are shell patterns that will be excluded from those matched by lines without an exclamation point. For example a `.cvmfsdirtab` might contain these lines for the repository of the previous subsection:

```
/software/*
/software/**
! */common
/grid-certificates
```

This will create nested catalogs at

```
/cvmfs/experiment.cern.ch/software/i686
/cvmfs/experiment.cern.ch/software/i686/1.0
/cvmfs/experiment.cern.ch/software/i686/2.0
/cvmfs/experiment.cern.ch/software/x86_64
/cvmfs/experiment.cern.ch/software/x86_64/1.0
/cvmfs/experiment.cern.ch/grid-certificates
```

Note that unlike the regular lines that add catalogs, asterisks in the exclamation point exclusion lines can span the slashes separating directory levels.

### Automatic Management of Nested Catalogs

An alternative to `.cvmfsdirtab` is the automatic catalog generation. This feature automatically generates nested catalogs based on their weight (number of entries). It can be enabled by setting `CVMFS_AUTOCATALOGS=true` in the server configuration file.

Catalogs are split when their weight is greater than a specified maximum threshold, or removed if their weight is less than a minimum threshold. Automatically generated catalogs contain a `.cvmfsautocatalog` file (along with the `.cvmfscatalog` file) in its root directory. User-defined catalogs (containing only a `.cvmfscatalog` file) always remain untouched. Hence one can mix both manual and automatically managed directory sub-trees.

The following conditions are applied when processing a nested catalog:

- If the weight is greater than `CVMFS_AUTOCATALOGS_MAX_WEIGHT`, this catalog will be split in smaller catalogs that meet the maximum and minimum thresholds.
- If the weight is less than `CVMFS_AUTOCATALOGS_MIN_WEIGHT`, this catalog will be merged into its parent.

Both `CVMFS_AUTOCATALOGS_MAX_WEIGHT` and `CVMFS_AUTOCATALOGS_MIN_WEIGHT` have reasonable defaults and usually do not need to be defined by the user.

### Inspecting Nested Catalog Structure

The following command visualizes the current nested file catalog layout of a repository.

```
cvmfs_server list-catalogs
```

Additionally this command allows to spot degenerated nested catalogs. As stated [here](#) the recommended maximal file entry count of a single catalog should not exceed  $\approx 200000$ . One can use the switch `list-catalogs -e` to inspect the current nested catalog entry counts in the repository. Furthermore `list-catalogs -s` will print the file sizes of the catalogs in bytes.

### Repository Mount Point Management

CernVM-FS server maintains two mount points for each repository (see *CernVM-FS Server Infrastructure* for details) and needs to keep them in sync with *transactional operations* on the repository.

In rare occasions (for example at reboot of a release manager machine) CernVM-FS might need to perform repair operations on those mount points. As of *CernVM-FS 2.2.0* those mount points are not automatically mounted on reboot of the release manager machine anymore. Usually the mount point handling happens automatically and transparently to the user when invoking arbitrary `cvmfs_server` commands.

Nevertheless `cvmfs_server mount <repo name>` allows users to explicitly trigger this repair operation any-time for individual repositories. Mounting all hosted repositories is possible with the `-a` parameter but requires root privileges. If you want to have all hosted repositories mounted after reboot then put `cvmfs_server mount -a` in a boot script, for example in `/etc/rc.local`.

```
# properly mount a specific repository
cvmfs_server mount test.cern.ch

# properly mount all hosted repositories (as root)
sudo cvmfs_server mount -a
```

## Syncing files into a repository with `cvmfs_rsync`

A common method of publishing into CernVM-FS is to first install all the files into a convenient shared filesystem, mount the shared filesystem on the publishing machine, and then sync the files into the repository during a transaction. The most common tool to do the syncing is `rsync`, but `rsync` by itself doesn't have a convenient mechanism for avoiding generated `.cvmfscatalog` and `.cvmfsautocatalog` files in the CernVM-FS repository. Actually the `--exclude` option is good for avoiding the extra files, but the problem is that if a source directory tree is removed, then `rsync` will not remove the corresponding copy of the directory tree in the repository if it contains a catalog, because the extra file remains in the repository. For this reason, a tool called `cvmfs_rsync` is included in the `cvmfs-server` package. This is a small wrapper around `rsync` that adds the `--exclude` options and removes `.cvmfscatalog` and `.cvmfsautocatalog` files from a repository when the corresponding source directory is removed. This is the usage:

```
cvmfs_rsync [rsync_options] srcdir /cvmfs/reponame[/destsubdir]
```

This is an example use case:

```
$ cvmfs_rsync -av --delete /data/lhapdf /cvmfs/cms.cern.ch
```

## Migrate File Catalogs

In rare cases the further development of CernVM-FS makes it necessary to change the internal structure of file catalogs. Updating the CernVM-FS installation on a Stratum 0 machine might require a migration of the file catalogs.

It is recommended that `cvmfs_server list` is issued after any CernVM-FS update to review if any of the maintained repositories need a migration. Outdated repositories will be marked as “INCOMPATIBLE” and `cvmfs_server` refuses all actions on these repositories until the file catalogs have been updated.

In order to run a file catalog migration use `cvmfs_server migrate` for each of the outdated repositories. This will essentially create a new repository revision that contains the exact same file structure as the current revision. However, all file catalogs will be recreated from scratch using the updated internal structure. Note that historic file catalogs of all previous repository revisions stay untouched and are not migrated.

After `cvmfs_server migrate` has successfully updated all file catalogs repository maintenance can continue as usual.

## Change File Ownership on File Catalog Level

CernVM-FS tracks the UID and GID of all contained files and exposes them through the client to all using machines. Repository maintainers should keep this in mind and plan their UID and GID assignments accordingly.

Repository operation might occasionally require to bulk-change many or all UIDs/GIDs. While this is of course possible via `chmod -R` in a normal repository transaction, it is cumbersome for large repositories. We provide a tool to quickly do such adaption on *CernVM-FS catalog level* using UID and GID mapping files:

```
cvmfs_server catalog-chown -u <uid map> -g <gid map> <repo name>
```

Both the UID and GID map contain a list of rules to apply to each file meta data record in the CernVM-FS catalogs. This is an example of such a rules list:

```
# map root UID/GID to 1001
0 1001

# swap UID/GID 1002 and 1003
1002 1003
```

```
1003 1002
# map everything else to 1004
* 1004
```

Note that running `cvmfs_server catalog-chown` produces a new repository revision containing *CernVM-FS catalogs* with updated UIDs and GIDs according to the provided rules. Thus, previous revisions of the CernVM-FS repository will *not* be affected by this update.

## Repository Garbage Collection

Since CernVM-FS is a versioning file system it is following an insert-only policy regarding its backend storage. When files are deleted from a CernVM-FS repository, they are not automatically deleted from the underlying storage. Therefore legacy revisions stay intact and usable forever (cf. *Named Snapshots*) at the expense of an ever-growing storage volume both on the Stratum 0 and the Stratum 1s.

For this reason, applications that frequently install files into a repository and delete older ones - for example the output from nightly software builds - might quickly fill up the repository's backend storage. Furthermore these applications might actually never make use of the aforementioned long-term revision preservation rendering most of the stored objects "garbage".

CernVM-FS supports garbage-collected repositories that automatically remove unreferenced data objects and free storage space. This feature needs to be enabled on the Stratum 0 and automatically scans the repository's catalog structure for unreferenced objects both on the Stratum 0 and the Stratum 1 installations on every publish respectively snapshot operation.

### Garbage Sweeping Policy

The garbage collector of CernVM-FS is using a mark-and-sweep algorithm to detect unused files in the internal catalog graph. Revisions that are referenced by named snapshots (cf. *Named Snapshots*) or that are recent enough are preserved while all other revisions are condemned to be removed. By default this time-based threshold is *three days* but can be changed using the configuration variable `CVMFS_AUTO_GC_TIMESPAN` both on Stratum 0 and Stratum 1. The value of this variable is expected to be parseable by the `date` command, for example `3 days ago` or `1 week ago`.

### Enabling Garbage Collection

#### Creating a Garbage Collectable Repository

Repositories can be created as *garbage-collectable* from the start by adding `-z` to the `cvmfs_server mkfs` command (cf. *Repository Creation*). It is generally recommended to also add `-g` to switch off automatic tagging in a garbage collectable repository. For debugging or bookkeeping it is possible to log deleted objects into a file by setting `CVMFS_GC_DELETION_LOG` to a writable file path.

#### Enabling Garbage Collection on an Existing Repository (Stratum 0)

Existing repositories can be reconfigured to be garbage collectable by adding `CVMFS_GARBAGE_COLLECTION=true` and `CVMFS_AUTO_GC=true` to the `server.conf` of the repository. Furthermore it is recommended to switch off automatic tagging by setting `CVMFS_AUTO_TAG=false` for a garbage collectable repository. The garbage collection will be enabled with the next published transaction.

## Enabling Garbage Collection on an Existing Replication (Stratum 1)

In order to use automatic garbage collection on a stratum 1 replica `CVMFS_AUTO_GC=true` needs to be added in the `server.conf` file of the stratum 1 installation. This will only work if the upstream stratum 0 repository has garbage collection enabled.

## Limitations on Repository Content

Because CernVM-FS provides what appears to be a POSIX filesystem to clients, it is easy to think that it is a general purpose filesystem and that it will work well with all kinds of files. That is not the case, however, because CernVM-FS is optimized for particular types of files and usage. This section contains guidelines for limitations on the content of repositories for best operation.

### Data files

First and foremost, CernVM-FS is designed to distribute executable code that is shared between a large number of jobs that run together at grid sites, clouds, or clusters. Worker node cache sizes and web proxy bandwidth are generally engineered to accommodate that application. The total amount read per job is expected to be roughly limited by the amount of RAM per job slot. The same files are also expected to be read from the worker node cache multiple times for the same type of job, and read from a caching web proxy by multiple worker nodes.

If there are data files distributed by CernVM-FS that follow similar access patterns and size limits as executable code, it will probably work fine. In addition, if there are files that are larger but read slowly throughout long jobs, as opposed to all at once at the beginning, that can also work well if the same files are read by many jobs. That is because web proxies have to be engineered for handling bursts at the beginning of jobs and so they tend to be lightly loaded a majority of the time.

In general, a good rule of thumb is to calculate the maximum rate at which jobs typically start and limit the amount of data that might be read from a web proxy to per thousand jobs, assuming a reasonable amount of overlap of jobs onto the same worker nodes. Also, limit the amount of data that will be put into any one worker node cache to . Of course, if you have a special arrangement with particular sites to have large caches and bandwidths available, these limits can be made higher at those sites. Web proxies may also need to be engineered with faster disks if the data causes their cache hit ratios to be reduced.

Also, keep in mind that the total amount of data distributed is not unlimited. The files are stored and distributed compressed, and files with the same content stored in multiple places in the same repository are collapsed to the same file in storage, but the storage space is used not only on the original repository server, it is also replicated onto multiple Stratum 1 servers. Generally if only executable code is distributed, there is no problem with the space taken on Stratum 1s, but if many large data files are distributed they may exceed the Stratum 1 storage capacity. Data files also tend to not compress as well, and that is especially the case of course if they are already compressed before installation.

### Tarballs, zip files, and other archive files

If the contents of a tarball, zip file, or some other type of archive file is desired to be distributed by CernVM-FS, it is usually better to first unpack it into its separate pieces first. This is because it allows better sharing of content between multiple releases of the file; some pieces inside the archive file might change and other pieces might not in the next release, and pieces that don't change will be stored as the same file in the repository. CernVM-FS will compress the content of the individual pieces, so even if there's no sharing between releases it shouldn't take much more space.

### File permissions

Care should be taken to make all the files in a repository readable by “other”. This is because permissions on files in the original repository are generally the same as those seen by end clients, except the files are owned by the “cvmfs” user and group. The write permissions are ignored by the client since it is a read-only filesystem. However, unless the client has set

```
CVMFS_CHECK_PERMISSIONS=no
```

(and most do not), unprivileged users will not be able to read files unless they are readable by “other” and all their parent directories have at least “execute” permissions. It makes little sense to publish files in CernVM-FS if they won’t be able to be read by anyone.

### Hardlinks

By default CernVM-FS does not allow hardlinks of a file to be in different directories. If there might be any such hardlinks in a repository, set the option

```
CVMFS_IGNORE_XDIR_HARDLINKS=true
```

in the repository’s `server.conf`. The file will not appear to be hardlinked to the client, but it will still be stored as only one file in the repository just like any other files that have identical content. Note that if, in a subsequent publish operation, only one of these cross-directory hardlinks gets changed, the other hardlinks remain unchanged (the hardlink got “broken”).

## Setting up a Replica Server (Stratum 1)

While a CernVM-FS Stratum 0 repository server is able to serve clients directly, a large number of clients is better be served by a set of Stratum 1 replica servers. Multiple Stratum 1 servers improve the reliability, reduce the load, and protect the Stratum 0 master copy of the repository from direct accesses. Stratum 0 server, Stratum 1 servers and the site-local proxy servers can be seen as content distribution network. The *figure below* shows the situation for the repositories hosted in the `cern.ch` domain.

A Stratum 1 server is a standard web server that uses the CernVM-FS server toolkit to create and maintain a mirror of a CernVM-FS repository served by a Stratum 0 server. To this end, the `cvmfs_server` utility provides the `add-replica` command. This command will register the Stratum 0 URL and prepare the local web server. Periodical synchronization has to be scheduled, for instance with `cron`, using the `cvmfs_server snapshot` command. The advantage over general purpose mirroring tools such as `rSync` is that all CernVM-FS file integrity verifications mechanisms from the Fuse client are reused. Additionally, by the aid of the CernVM-FS file catalogs, the `cvmfs_server` utility knows beforehand (without remote listing) which files to transfer.

In order to prevent accidental synchronization from a repository, the Stratum 0 repository maintainer has to create a `.cvmfs_master_replica` file in the HTTP root directory. This file is created by default when a new repository is created. Note that replication can thrash caches that might exist between Stratum 1 and Stratum 0. A direct connection is therefore preferable.

### Recommended Setup

The vast majority of HTTP requests will be served by the site’s local proxy servers. Being a publicly available service, however, we recommend to install a Squid frontend in front of the Stratum 1 web server.

We suggest the following key parameters:

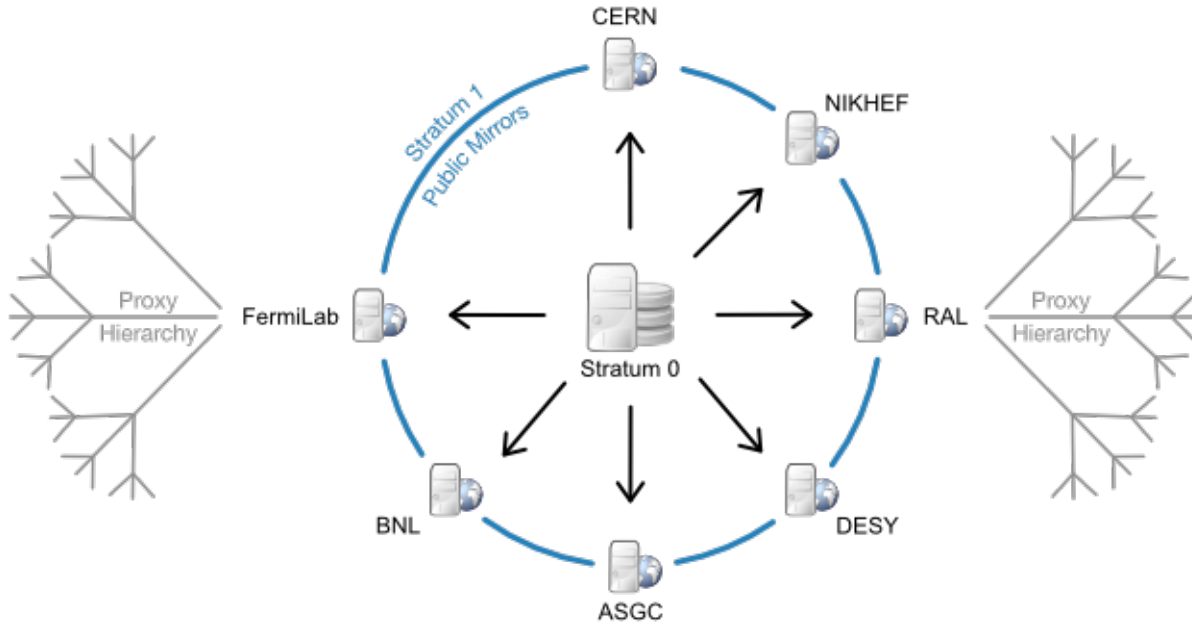


Fig. 2.4: CernVM-FS content distribution network for the cern.ch domain: Stratum1 replica servers are located in Europe, the U.S. and Asia. One protected read/write instance (Stratum 0) is feeding up the public, distributed mirror servers. A distributed hierarchy of proxy servers fetches content from the closest public mirror server.

**Storage** RAID-protected storage. The `cvmfs_server` utility should have low latency to the storage because it runs a large number of system calls (`stat()`) against it. For the local storage backends `ext3/4` filesystems are preferred (rather than XFS).

**Web server** A standard Apache server. Directory listing is not required. In addition, it is a good practice to exclude search engines from the replica web server by an appropriate `robots.txt`. The webserver should be close to the storage in terms of latency.

**Squid frontend** Squid should be used as a frontend to Apache, configured as a reverse proxy. It is recommended to run it on the same machine as Apache to reduce the number of points of failure. Alternatively, separate Squid server machines may be configured in load-balance mode forwarding to the Apache server, but note that if any of them are down the entire service will be considered down by CernVM-FS clients. The Squid frontend should listen on ports 80 and 8000. The more RAM that the operating system can use for file system caching, the better.

**DNS cache** A Stratum 1 does a lot of DNS lookups, so we recommend installing a DNS caching mechanism on the machine such as `dnsmasq` or `bind`. We do not recommend `nsd` since it does not honor the DNS Time-To-Live protocol.

## Squid Configuration

The Squid configuration differs from the site-local Squids because the Stratum 1 Squid servers are transparent to the clients (*reverse proxy*). As the expiry rules are set by the web server, Squid cache expiry rules remain unchanged.

The following lines should appear accordingly in `/etc/squid/squid.conf`:

```
http_port 80 accel
http_port 8000 accel
http_access allow all
cache_peer <APACHE_HOSTNAME> parent <APACHE_PORT> 0 no-query originserver
```

```
cache_mem <MEM_CACHE_SIZE> MB
cache_dir ufs /var/spool/squid <DISK_CACHE_SIZE in MB> 16 256
maximum_object_size 1024 MB
maximum_object_size_in_memory 128 KB
```

Note that `http_access allow all` has to be inserted before (or instead of) the line `http_access deny all`. If Apache is running on the same host, the `APACHE_HOSTNAME` will be `localhost`. Also, in that case there is not a performance advantage for squid to cache files that came from the same machine, so you can configure squid to not cache files. Do that with the following lines:

```
acl CVMFSAPI urlpath_regex ^/cvmfs/[^/]*api/
cache deny !CVMFSAPI
```

Then the squid will only cache API calls. You can then set `MEM_CACHE_SIZE` and `DISK_CACHE_SIZE` quite small.

Check the configuration syntax by `squid -k parse`. Create the hard disk cache area with `squid -z`. In order to make the increased number of file descriptors effective for Squid, execute `ulimit -n 8192` prior to starting the squid service.

## Monitoring

The `cvmfs_server` utility reports status and problems to `stdout` and `stderr`.

For the web server infrastructure, we recommend standard Nagios HTTP checks. They should be configured with the URL `http://protect\T1\textdollarreplica-server/cvmfs/protect\T1\textdollarrepository_name/cvmfspublished`. This file can also be used to monitor if the same repository revision is served by the Stratum 0 server and all the Stratum 1 servers. In order to tune the hardware and cache sizes, keep an eye on the Squid server's CPU and I/O load.

Keep an eye on HTTP 404 errors. For normal CernVM-FS traffic, such failures should not occur. Traffic from CernVM-FS clients is marked by an `X-CVMFS2` header.

## CernVM-FS Gateway Services and Release Managers

This page details the installation and configuration of a repository setup involving a gateway machine and separate release manager machines.

## Glossary

**Gateway (GW)** The machine running an instance of the [CVMFS gateway services](#) and which has access to the authoritative storage of the managed repositories. There is a current limitation in the implementation of the gateway services that the authoritative storage must be a locally mounted partition on GW, however S3 backends should be supported in the future. The purpose of the GW is to manage access to a set of repositories by assigning exclusive leases to specific repository sub-paths to different release manager (RM) machines. The RM can publish files to the sub-path for which it currently holds a lease but sending object packs to the GW. Having received the published payload from the RM, the final task of the GW in the publication lifecycle is to rebuild the catalogs and repository manifests for the modified repositories.



**Release manager (RM)** A machine running the CVMFS server tools which can request leases from a GW and publish change to different repositories where it currently holds a valid lease.

It is on the RM that the computationally heavy tasks of compressing and hashing the files which are to be added or modified as part of a publish operation. The processed files are finally packed together and send to the GW to be inserted into the repository and made available to clients.

## Gateway Services Configuration

Gateway services installation instructions will be added after the deployment strategy for the gateway services application is clarified.

## Release Manager Configuration

This section describes the steps needed to set up a release manager for a specific CVMFS repository. The precondition is a working gateway machine where the repository has been created as a Stratum 0.

1. Create another instance of the repository on the release manager using `cvmfs_server mkfs`.
2. Retrieve the public key of the repository and the GW API key from the GW and place them into `/etc/cvmfs/keys`.
3. In the `client.conf` file for the repository, modify the `CVMFS_SERVER_URL` variable to point to the GW URL.
4. In the `server.conf` file for the repository, modify the `CVMFS_STRATUM0` variable to point to the GW URL. Set `CVMFS_UPSTREAM_STORAGE` to `gw,<UNUSED_PATH>,<GW_API_URL>`. Set `CVMFS_GATEWAY_KEYS` to point to the gateway key file.

### Example:

- The gateway machine is `gateway.cern.ch`.
- The release manager is `rm.cern.ch`.
- The new repository's fully qualified name is `test.cern.ch`.
- The repository's public key is `test.cern.ch.pub`.
- The GW API key is `gateway_key`.
- The GW services application is running on port 8080 at the URL `http://gateway.cern.ch:8080/api/v1`.

### Steps:

1. `cvmfs_server mkfs test.cern.ch`
2. Copy `test.cern.ch.pub` and `gateway_key` to `/etc/cvmfs/keys/`.
3. In `client.conf`, set `CVMFS_SERVER_URL` to `http://gateway.cern.ch/cvmfs/test.cern.ch`.
4. In `server.conf`, set `CVMFS_STRATUM0` to `http://gateway.cern.ch/cvmfs/test.cern.ch`, `CVMFS_UPSTREAM_STORAGE` to `gw,/srv/cvmfs/test.cern.ch/data/txt,http://gateway.cern.ch:8080/api/v1` and `CVMFS_GATEWAY_KEYS` to `/etc/cvmfs/keys/gateway_key`

At this point, from the RM we can publish to the repository:

```
$ cvmfs_server transaction -e test.cern.ch  
  
... make changes to the repository ...  
  
$ cvmfs_server publish -e test.cern.ch
```

## CernVM-FS Server Meta Information

The CernVM-FS server automatically maintains both global and repository specific meta information as JSON data. Release manager machines keep a list of hosted Stratum0 and Stratum1 repositories and user-defined administrative meta information.

Furthermore each repository contains user-maintained and signed meta information gets replicated to Stratum1 servers automatically.

### Global Meta Information

This JSON data provides information about the CernVM-FS server itself. A list of all repositories (both Stratum0 and Stratum1) hosted at this specific server is automatically generated and can be accessed here:

```
http://<server base URL>/cvmfs/info/v1/repositories.json
```

Furthermore there might be user-defined information like the administrator's name, contact information and an arbitrary user-defined JSON portion here:

```
http://<server base URL>/cvmfs/info/v1/meta.json
```

Using the `cvmfs_server` utility, an administrator can edit the user-defined portion of the data with a text editor (cf. `$EDITOR`):

```
cvmfs_server update-info
```

Note that the `cvmfs_server` package requires the `jq` utility, which validates CVMFS JSON data.

Below are *examples* of both the repository list and user-defined JSON files.

### Repository Specific Meta Information

Each repository contains a JSON object with repository specific meta data. The information is maintained by the repository's owner on the Stratum0 release manager machine. It contains the maintainer's contact information, a description of the repository's content, the recommended Stratum 0 URL and a list of recommended Stratum 1 replica URLs. Furthermore it provides a custom JSON region for arbitrary information.

Note that this JSON file is stored inside CernVM-FS's backend data structure and gets replicated to Stratum1 servers automatically.

Editing is done per repository using the `cvmfs_server` utility. As with the *global meta information* `cvmfs_server` uses `jq` to validate edited JSON information before storing it:

```
cvmfs_server update-repoinfo <repo name>
```

Besides the interactive editing (cf. `$EDITOR`) one can specify a file path that should be stored as the repository's meta information:

```
cvmfs_server update-repoinfo -f <path to JSON file> <repo name>
```

An example of a repository specific meta information file can be found in *the section below*.

## Examples

### /cvmfs/info/v1/meta.json

```
{
  "administrator" : "Your Name",
  "email"         : "you@organisation.org",
  "organisation"  : "Your Organisation",

  "custom" : {
    "_comment" : "Put arbitrary structured data here"
  }
}
```

### /cvmfs/info/v1/repositories.json

```
{
  "schema"      : 1,
  "repositories" : [
    {
      "name" : "atlas.cern.ch",
      "url"  : "/cvmfs/atlas.cern.ch"
    },
    {
      "name" : "cms.cern.ch",
      "url"  : "/cvmfs/cms.cern.ch"
    }
  ],
  "replicas" : [
    {
      "name" : "lhcb.cern.ch",
      "url"  : "/cvmfs/lhcb.cern.ch"
    }
  ]
}
```

## Repository Specific Meta Information

```
{
  "administrator" : "Your Name",
  "email"         : "you@organisation.org",
  "organisation"  : "Your Organisation",
  "description"   : "Repository content",
  "recommended-stratum0" : "stratum 0 url",
  "recommended-stratum1s" : [ "stratum1 url", "stratum1 url" ],

  "custom" : {
    "_comment" : "Put arbitrary structured data here"
  }
}
```

```
}  
}
```

## Client Plug-Ins

The CernVM-FS client's functionality can be extended through plug-ins. CernVM-FS plug-ins are binaries (processes) that communicate with the main client process through IPC. Currently the only plug-in interface is for authorization helpers.

### Authorization Helpers

Client authorization helpers (*authz helper*) can be used to grant or deny read access to a mounted repository. To do so, authorization helpers can verify the local UNIX user (uid/gid) and the process id (pid) that is issuing a file system request.

An authz helper is spawned by CernVM-FS if the root file catalog contains *membership requirement* (see below). The binary to be spawned is derived from the membership requirement but it can be overwritten with the `CVMFS_AUTHZ_HELPER` parameter. The authz helper listens for commands on `stdin` and it replies on `stdout`.

Grant/deny decisions are typically cached for a while by the client. Note that replies are cached for the entire session (session id) that contains the calling process id.

### Membership Requirement

The root file catalog of a repository determines if and which authz helper should be used by a client. The membership requirement (also called *VOMS authorization*) can be set, unset, and changed when creating a repository and on every publish operation. It has the form

```
<helper>%<membership string>
```

The `<helper>` component helps the client find an authz helper. The client searches for a binary `${CVMFS_AUTHZ_SEARCH_PATH}/cvmfs_helper_<helper>`. By default, the search path is `/usr/libexec/cvmfs/authz`. CernVM-FS comes with two helpers: `cvmfs_helper_allow` and `cvmfs_helper_deny`. Both helpers make static decisions and disregard the membership string. Other helpers can use the membership string to specify user groups that are allowed to access a repository.

### Authz Helper Protocol

The authz helper gets spawned by the CernVM-FS client with `stdin` and `stdout` connected. There is a command/reply style of messages. Messages have a 4 byte version (=1), a 4 byte length, and then a JSON text that needs to contain the top-level struct `cvmfs_authz_v1 { ... }`. Communication starts with a handshake where the client passes logging parameters to the authz helper. The client then sends zero or more authorization requests, each of which is answered by a positive or negative permit. A positive permit can include an access token that should be used to download data. The permits are cached by the client with a TTL that the helper can chose. On unmount, the client sends a quit command to the helper.

When spawned, the authz helper's environment is prepopulated with all `CVMFS_AUTHZ_...` environment variables that are in the CernVM-FS client's environment. Furthermore the parameter `CVMFS_AUTHZ_HELPER=yes` is set.

The JSON snippet of every message contains `msgid` and `revision` integer fields. The revision is currently 0 and unused. Message ids indicate certain other fields that can or should be present. Additional JSON text is ignored. The message id can be one of the following

Code	Meaning
0	Cvmfs: “Hello, helper, are you there?” (handshake)
1	Helper: “Yes, cvmfs, I’m here” (handshake reply)
2	Cvmfs: “Please verify, helper” (verification request)
3	Helper: “I verified, cvmfs, here’s the result” (permit)
4	Cvmfs: “Please shutdown, helper” (termination)

## Handshake and Termination

In the JSON snippet of the hand shake, the CernVM-FS client transmits the fully qualified repository name (`fqrn` string field) and the syslog facility and syslog level the helper is supposed to use (`syslog_facility`, `syslog_level` integer fields). The handshake reply as well as the termination have no additional payload.

## Verification Requests

A verification request contains the `uid`, `gid`, and `pid` of the calling process (`uid`, `gid`, `pid` integer fields). It furthermore contains the Base64 encoded membership string from the membership requirement (`membership` string field).

The permit has to contain a status indicating success or failure (`status` integer field) and a time to live for this reply in seconds (`ttl` integer field). The status can be one of the following

Code	Meaning
0	Success (allow access)
1	Authentication token of the user not found (deny access)
2	Invalid authentication token (deny access)
3	User is not member of the required groups (deny access)

On success, the permit can optionally contain a Base64 encoded version of an X.509 proxy certificate (`x509_proxy` string field). This certificate is used by the CernVM-FS client when downloading nested catalogs files as client-side HTTPS authentication certificate.

## Implementation Notes

CernVM-FS has a modular structure and relies on several open source libraries. Figure *below* shows the internal building blocks of CernVM-FS. Most of these libraries are shipped with the CernVM-FS sources and are linked statically in order to facilitate debugging and to keep the system dependencies minimal.

## File Catalog

A CernVM-FS repository is defined by its *file catalog*. The file catalog is an SQLite database [Allen10] having a single table that lists files and directories together with its metadata. The table layout is shown in the table below:

Field	Type
Path MD5	128Bit Integer
Parent Path MD5	128Bit Integer
Hardlinks	Integer
Content Hash	BLOB
Size	Integer
Mode	Integer
Last Modified	Timestamp
Flags	Integer
Name	String
Symlink	String
uid	Integer
gid	Integer
xattr	BLOB

In order to save space we do not store absolute paths. Instead we store MD5 [Rivest92], [Turner11] hash values of the absolute path names. Symbolic links are kept in the catalog. Symbolic links may contain environment variables in the form `$(VAR_NAME)` or `$(VAR_NAME:~/default/path)` that will be dynamically resolved by CernVM-FS on access. Hardlinks are emulated by CernVM-FS. The hardlink count is stored in the lower 32bit of the hardlinks field, a *hardlink group* is stored in the higher 32 bit. If the hardlink group is greater than zero, all files with the same hardlink group will get the same inode issued by the CernVM-FS Fuse client. The emulated hardlinks work within the same directory, only. The cryptographic content hash refers to the zlib-compressed [Deutsch96] version of the file. Flags indicate the type of an directory entry (see table *below*).

Extended attributes are either NULL or stored as a BLOB of key-value pairs. It starts with 8 bytes for the data structure's version (currently 1) followed by 8 bytes for the number of extended attributes. This is followed by the list of pairs, which start with two 8 byte values for the length of the key/value followed by the concatenated strings of the

Flags	Meaning
1	Directory
2	Transition point to a nested catalog
33	Root directory of a nested catalog
4	Regular file
8	Symbolic link
68	Chunked file
132	External file (stored under path name)

key and the value.

As of bit 8, the flags store the cryptographic content hash algorithm used to process the given file. Bit eleven is 1 if the file is stored uncompressed.

A file catalog contains a *time to live* (TTL), stored in seconds. The catalog TTL advises clients to check for a new version of the catalog, when expired. Checking for a new catalog version takes place with the first file system operation on a CernVM-FS volume after the TTL has expired. The default TTL is 15 minutes. If a new catalog is available, CernVM-FS delays the loading for the period of the CernVM-FS kernel cache life time (default: 1 minute). During this drain-out period, the kernel caching is turned off. The first file system operation on a CernVM-FS volume after that additional delay will apply a new file catalog and kernel caching is turned back on.

## Content Hashes

CernVM-FS can use SHA-1 [Jones01], RIPEMD-160 [Dobbertin96] and SHAKE-128 [Bertoni09] as cryptographic hash function. The hash function can be changed on the Stratum 0 during the lifetime of repositories. On a change, new and updated files will use the new cryptographic hash while existing files remain unchanged. This is transparent to the clients since the hash function is stored in the flags field of file catalogs for each and every file. The default hash function is SHA-1. New software versions might introduce support for further cryptographic hash functions.

## Nested Catalogs

In order to keep catalog sizes reasonable<sup>1</sup>, repository subtrees may be cut and stored as separate *nested catalogs*. There is no limit on the level of nesting. A reasonable approach is to store separate software versions as separate nested catalogs. The figure *below* shows the simplified directory structure which we use for the ATLAS repository.

Fig. 2.5: Directory structure used for the ATLAS repository (simplified).

When a subtree is moved into a nested catalog, its entry directory serves as *transition point* for nested catalogs. This directory appears as empty directory in the parent catalog with flags set to 2. The same path appears as root-directory in the nested catalog with flags set to 33. Because the MD5 hash values refer to full absolute paths, nested catalogs store the root path prefix. This prefix is prepended transparently by CernVM-FS. The cryptographic hash of nested catalogs is stored in the parent catalog. Therefore, the root catalog fully defines an entire repository.

Loading of nested catalogs happens on demand by CernVM-FS on the first attempt to access of anything inside, a user won't see the difference between a single large catalog and several nested catalogs. While this usually avoids unnecessary catalogs to be loaded, recursive operations like `find` can easily bypass this optimization.

## Catalog Statistics

A CernVM-FS file catalog maintains several counters about its contents and the contents of all of its nested catalogs. The idea is that the catalogs know how many entries there are in their sub catalogs even without opening them. This way, one can immediately tell how many entries, for instance, the entire ATLAS repository has. Some of the numbers are shown using the number of inodes in `statvfs`. So `df -i` shows the overall number of entries in the repository and (as number of used inodes) the number of entries of currently loaded catalogs. Nested catalogs create an additional entry (the transition directory is stored in both the parent and the child catalog). File hardlinks are still individual entries (inodes) in the `cvmfs` catalogs. The following counters are maintained for both a catalog itself and for the subtree this catalog is root of:

- Number of regular files
- Number of symbolic links
- Number of directories
- Number of nested catalogs
- Number of external files
- Number of chunked files
- Number of individual file chunks
- Overall file content size
- File content size stored in chunked files

## Repository Manifest (`.cvmfspublished`)

Every CernVM-FS repository contains a repository manifest file that serves as entry point into the repository's catalog structure. The repository manifest is the first file accessed by the CernVM-FS client at mount time and therefore must be accessible via HTTP on the repository root URL. It is always called `.cvmfspublished` and contains fundamental repository meta data like the root catalog's cryptographic hash and the repository revision number as a key-value list.

<sup>1</sup> As a rule of thumb, file catalogs up to (compressed) are reasonably small.

## Internal Manifest Structure

Below is an example of a typical manifest file. Each line starts with a capital letter specifying the meta data field, followed by the actual data string. The list of meta information is ended by a separator line (--) followed by signature information further described [here](#).

```
C64551dccf8be0a48de7618dd7deb290200b474759
B1442336
Rd41d8cd98f00b204e9800998ecf8427e
D900
S42
Nexample.cern.ch
X731cca9476eb882f5a3f24aaa38001105a0e35eb
T1390301299
--
edde5308e502dd5e8fe405c56f5700f7477dc319
[...]
```

Please refer to table below for detailed information about each of the meta data fields.

Field	Meta Data Description
C	Cryptographic hash of the repository's current root catalog
B	Size of the root file catalog in bytes
A	"yes" if the catalog should be fetched under its alternative name (outside servers /data directory)
R	MD5 hash of the repository's root path (usually always d41d8cd98f00b204e9800998ecf8427e)
B	File size of the root catalog in bytes
X	Cryptographic hash of the signing certificate
G	"yes" if the repository is garbage-collectable
H	Cryptographic hash of the repository's named tag history database
T	Unix timestamp of this particular revision
D	Time To Live (TTL) of the root catalog
S	Revision number of this published revision
N	The full name of the manifested repository
M	Cryptographic hash of the repository JSON metadata
L	currently unused (reserved for micro catalogs)

## Repository Signature

In order to provide authoritative information about a repository publisher, the repository manifest is signed by an X.509 certificate together with its private key.

## Signing a Repository

It is important to note that it is sufficient to sign just the manifest file itself to gain a secure chain of the whole repository. The manifest refers to the cryptographic content hash of the root catalog which in turn recursively references all sub-catalogs with their cryptographic content hashes. Each catalog lists its files along with their cryptographic content hashes. This concept is called a merkle tree and eventually provides a single hash that depends on the *complete* content of the repository.

The top level hash used for the repository signature can be found in the repository manifest right below the separator line (-- / *see above*). It is the cryptographic hash of the manifest's meta data lines excluding the separator line.



Following the top level hash is the actual signature produced by the X.509 certificate signing procedure in binary form.

## Signature Validation

In order to validate repository manifest signatures, CernVM-FS uses a white-list of valid publisher certificates. The white-list contains the cryptographic fingerprints of known publisher certificates and a timestamp. A white-list is valid for 30 days. It is signed by a private RSA key, which we refer to as *master key*. The public RSA key that corresponds to the master key is distributed with the `cvmfs-config-...` RPMs as well as with every instance of CernVM.

In addition, CernVM-FS checks certificate fingerprints against the local blacklist `/etc/cvmfs/blacklist`. The blacklisted fingerprints have to be in the same format than the fingerprints on the white-list. The blacklist has precedence over the white-list.

As crypto engine, CernVM-FS uses libcrypto from the [OpenSSL project](#).

## Use of HTTP

The particular way of using the HTTP protocol has significant impact on the performance and usability of CernVM-FS. If possible, CernVM-FS tries to benefit from the HTTP/1.1 features keep-alive and cache-control. Internally, CernVM-FS uses the [libcurl library](#).

The HTTP behaviour affects a system with cold caches only. As soon as all necessary files are cached, there is only network traffic when a catalog TTL expires. The CernVM-FS download manager runs as a separate thread that handles download requests asynchronously in parallel. Concurrent download requests for the same URL are collapsed into a single request.

## DoS Protection

A subtle denial of service attack (DoS) can occur when CernVM-FS is successfully able to download a file but fails to store it in the local cache. This situation escalates into a DoS when the application using CernVM-FS remains in an endless loop and tries to open a file over and over again. Such a situation is prevented by CernVM-FS by re-trying with an exponential backoff. The backoff is triggered by consecutive failures to cache a downloaded file within 10 seconds.

## Keep-Alive

Although the HTTP protocol overhead is small in terms of data volume, in high latency networks we suffer from the bare number of requests: Each request-response cycle has a penalty of at least the network round trip time. Using plain HTTP/1.0, this results in at least 3 · round trip time additional running time per file download for TCP handshake, HTTP GET, and TCP connection finalisation. By including the `Connection: Keep-Alive` header into HTTP requests, we advise the HTTP server end to keep the underlying TCP connection opened. This way, overhead ideally drops to just round trip time for a single HTTP GET. The impact of the keep-alive feature is shown in here.

This feature, of course, somewhat sabotages a server-side load-balancing. However, exploiting the HTTP keep-alive feature does not affect scalability per se. The servers and proxies may safely close idle connections anytime, in particular if they run out of resources.

### Cache Control

In a limited way, CernVM-FS advises intermediate web caches how to handle its requests. Therefor it uses the `Pragma: no-cache` and the `Cache-Control: no-cache` headers in certain cases. These cache control headers apply to both, forward proxies as well as reverse proxies. This is not a guarantee that intermediate proxies fetch a fresh original copy (though they should).

By including these headers, CernVM-FS tries to not fetch outdated cache copies. Only in case CernVM-FS downloads a corrupted file from a proxy server, it retries having the HTTP `no-cache` header set. This way, the corrupted file gets replaced in the proxy server by a fresh copy from the backend.

### Identification Header

CernVM-FS sends a custom header (`X-CVMFS2`) to be identified by the web server. If you have set the CernVM GUID, this GUID is also transmitted.

### Redirects

Normally, the Stratum-1 servers directly respond to HTTP requests so CernVM-FS has no need to support HTTP redirect response codes. However, there are some high-bandwidth applications where HTTP redirects are used to transfer requests to multiple data servers. To enable support for redirects in the CernVM-FS client, set `CVMFS_FOLLOW_REDIRECTS=yes`.

### Name Resolving

Round-robin DNS entries for proxy servers are treated specially by CernVM-FS. Multiple IP addresses for the same proxy name are automatically transformed into multiple proxy servers within the same load-balance group. So the usual rules for load-balancing and fail-over apply to the different servers in a round-robin entry. CernVM-FS resolves all the proxy servers at once (and in parallel) at mount time. From that point on, proxy server names are resolved on demand, when a download takes place and the TTL of the active proxy expired. CernVM-FS resolves using `/etc/host` (resp. the file referenced in the `HOST_ALIASES` environment variable) or, if a host name is not resolvable locally, it uses the `c-ares` resolver. Proxy servers given in IP notation remain unchanged.

CernVM-FS uses the TTLs that come from DNS servers. However, there is a cutoff at 1 minute minimum TTL and 1 day maximum TTL. Locally resolved host names get a TTL of 1 minute. The host alias file is re-read with every attempt to resolve a name. Failed attempts to resolve a name remain cached for 1 minute, too. If a name has been successfully resolved previously, this result stays active until another successful attempt is done. If the DNS entries change for a host name, CernVM-FS adjust the corresponding load-balance group and picks a new server from the group at random.

The name resolving silently ignores errors in individual records. Only if no valid IP address is returned at all it counts as an error. IPv4 addresses have precedence if available. If the `CVMFS_IPV4_ONLY` environment variable is set, CernVM-FS does not try to resolve IPv6 records.

The timeout for name resolving is hard-coded to 2 attempts with a timeout of 3 seconds each. This is independent from the `CVMFS_TIMEOUT` and `CVMFS_TIMEOUT(_DIRECT)` settings. The effective timeout can be a bit longer than 6 seconds because of a backoff.

The name server used by CernVM-FS is looked up only once on start. If the name server changes during the life time of a CernVM-FS mount point, this change needs to be manually advertised to CernVM-FS using `cvmfs_talk nameserver set`.

## Disk Cache

Each running CernVM-FS instance requires a local cache directory. Data are downloaded into a temporary files. Only at the very latest point they are renamed into their content-addressable names atomically by `rename()`.

The hard disk cache is managed, CernVM-FS maintains cache size restrictions and replaces files according to the least recently used (LRU) strategy [Panagiotou06]. In order to keep track of files sizes and relative file access times, CernVM-FS sets up another SQLite database in the cache directory, the *cache catalog*. The cache catalog contains a single table; its structure is shown here:

Field	Type
Hash	String (hex notation)
Size	Integer
Access Sequence	Integer
Pinned	Integer
File type (chunk or file catalog)	Integer

CernVM-FS does not strictly enforce the cache limit. Instead CernVM-FS works with two customizable soft limits, the *cache quota* and the *cache threshold*. When exceeding the cache quota, files are deleted until the overall cache size is less than or equal to the cache threshold. The cache threshold is currently hard-wired to half of the cache quota. The cache quota is for data files as well as file catalogs. Currently loaded catalogs are pinned in the cache, they will not be deleted until unmount or until a new repository revision is applied. On unmount, pinned file catalogs are updated with the highest sequence number. As a pre-caution against a cache that is blocked by pinned catalogs, all catalogs except the root catalog are unpinned when the volume of pinned catalogs exceeds of the overall cache volume.

The cache catalog can be re-constructed from scratch on mount. Re-constructing the cache catalog is necessary when the managed cache is used for the first time and every time when “unmanaged” changes occurred to the cache directory, when CernVM-FS was terminated unexpectedly.

In case of an exclusive cache, the cache manager runs as a separate thread of the `cvmfs2` process. This thread gets notified by the Fuse module whenever a file is opened or inserted. Notification is done through a pipe. The shared cache uses the very same code, except that the thread becomes a separate process (see Figure *below*). This cache manager process is not another binary but `cvmfs2` forks to itself with special arguments, indicating that it is supposed to run as a cache manager. The cache manager does not need to be started as a service. The first CernVM-FS instance that uses a shared cache will automatically spawn the cache manager process. Subsequent CernVM-FS instances will connect to the pipe of this cache manager. Once the last CernVM-FS instance that uses the shared cache is unmounted, the communication pipe is left without any writers and the cache manager automatically quits.

The CernVM-FS cache supports two classes of files with respect to the cache replacement strategy: *normal* files and *volatile* files. The sequence numbers of volatile files have bit 63 set. Hence they are interpreted as negative numbers and have precedence over normal files when it comes to cache cleanup. On automatic rebuild the volatile property of entries in the cache database is lost.

## NFS Maps

In normal mode, CernVM-FS issues inodes based on the row number of an entry in the file catalog. When exported via NFS, this scheme can result in inconsistencies because CernVM-FS does not control the cache lifetime of NFS clients. A once issued inode can be asked for anytime later by a client. To be able to reply to such client queries even after reloading catalogs or remounts of CernVM-FS, the CernVM-FS *NFS maps* implement a persistent store of the path names  $\mapsto$  inode mappings. Storing them on hard disk allows for control of the CernVM-FS memory consumption (currently  $\approx$  45 MB extra) and ensures consistency between remounts of CernVM-FS. The performance penalty for doing so is small. CernVM-FS uses *Google’s leveldb* <<https://github.com/google/leveldb>>, a fast, local key value store. Reads and writes are only performed when meta-data are looked up in SQLite, in which case the SQLite query supposedly dominates the running time.

A drawback of the NFS maps is that there is no easy way to account for them by the cache quota. They sum up to some 150-200 Bytes per path name that has been accessed. A recursive `find` on `/cvmfs/atlas.cern.ch` with 50 million entries, for instance, would add up 8GB in the cache directory. This is mitigated by the fact that the NFS mode will be only used on few servers that can be given large enough spare space on hard disk.

## Loader

The CernVM-FS Fuse module comprises a minimal *loader* loader process (the `cvmfs2` binary) and a shared library containing the actual Fuse module (`libcvmfs_fuse.so`). This structure makes it possible to reload CernVM-FS code and parameters without unmounting the file system. Loader and library don't share any symbols except for two global structs `cvmfs_exports` and `loader_exports` used to call each others functions. The loader process opens the Fuse channel and implements stub Fuse callbacks that redirect all calls to the CernVM-FS shared library. Hotpatch is implemented as unloading and reloading of the shared library, while the loader temporarily queues all file system calls in-between. Among file system calls, the Fuse module has to keep very little state. The kernel caches are drained out before reloading. Open file handles are just file descriptors that are held open by the process. Open directory listings are stored in a Google dense\_hash that is saved and restored.

## File System Interface

CernVM-FS implements the following read-only file system call-backs.

### mount

On mount, the file catalog has to be loaded. First, the file catalog `manifest.cvmfspublished` is loaded. The manifest is only accepted on successful validation of the signature. In order to validate the signature, the certificate and the white-list are downloaded in addition if not found in cache. If the download fails for whatever reason, CernVM-FS tries to load a local file catalog copy. As long as all requested files are in the disk cache as well, CernVM-FS continues to operate even without network access (*offline mode*). If there is no local copy of the manifest or the downloaded manifest and the cache copy differ, CernVM-FS downloads a fresh copy of the file catalog.

### getattr and lookup

Requests for file attributes are entirely served from the mounted catalogs, there is no network traffic involved. This function is called as pre-requisite to other file system operations and therefore the most frequently called Fuse callback. In order to minimize relatively expensive SQLite queries, CernVM-FS uses a hash table to store negative and positive query results. The default size of for this memory cache is determined according to benchmarks with LHC experiment software.

Additionally, the callback takes care of the catalog TTL. If the TTL is expired, the catalog is re-mounted on the fly. Note that a re-mount might possibly break running programs. We rely on careful repository publishers that produce more or less immutable directory trees, new repository versions just add files.

If a directory with a nested catalog is accessed for the first time, the respective catalog is mounted in addition to the already mounted catalogs. Loading nested catalogs is transparent to the user.

### readlink

A symbolic link is served from the file catalog. As a special extension, CernVM-FS detects environment variables in symlink strings written as `$(VARIABLE)` or `$(VARIABLE:~/default/path)`. These variables are expanded by CernVM-FS dynamically on access (in the context of the `cvmfs2` process). This way, a single symlink can point to

different locations depending on the environment. This is helpful, for instance, to dynamically select software package versions residing in different directories.

## readdir

A directory listing is served by a query on the file catalog. Although the “parent”-column is indexed (see *Catalog table schema*), this is a relatively slow function. We expect directory listing to happen rather seldom.

## open / read

The `open()` call has to provide a file descriptor for a given path name. In CernVM-FS file requests are always served from the disk cache. The Fuse file handle is a file descriptor valid in the context of the CernVM-FS process. It points into the disk cache directory. Read requests are translated into the `pread()` system call.

## getxattr

CernVM-FS uses extended attributes to display additional repository information. There are two supported attributes:

**chunks** Number of chunks of a regular file.

**compression** Compression algorithm, for regular files only. Either “zlib” or “none”.

**expires** Shows the remaining life time of the mounted root file catalog in minutes.

**external\_file** Indicates if a regular file is an external file or not. Either 0 or 1.

**external\_host** Like `host` but for the host settings to fetch external files.

**external\_timeout** Like `timeout` but for the host settings to fetch external files.

**fqrn** Shows the fully qualified repository name of the mounted repository.

**hash** Shows the cryptographic hash of a regular file as listed in the file catalog.

**host** Shows the currently active HTTP server.

**host\_list** Shows the ordered list of HTTP servers.

**inode\_max** Shows the highest possible inode with the current set of loaded catalogs.

**lhash** Shows the cryptographic hash of a regular file as stored in the local cache, if available.

**maxfd** Shows the maximum number of file descriptors available to file system clients.

**nclg** Shows the number of currently loaded nested catalogs.

**ndiropen** Shows the overall number of opened directories.

**ndownload** Shows the overall number of downloaded files since mounting.

**nioerr** Shows the total number of I/O errors encountered since mounting.

**nopen** Shows the overall number of `open()` calls since mounting.

**pid** Shows the process id of the CernVM-FS Fuse process.

**proxy** Shows the currently active HTTP proxy.

**pubkeys** The loaded public RSA keys used for repository whitelist verification.

**rawlink** Shows unresolved variant symbolic links; only accessible as root.

**revision** Shows the file catalog revision of the mounted root catalog, an auto-increment counter increased on every repository publish.

**root\_hash** Shows the cryptographic hash of the root file catalog.

**rx** Shows the overall amount of downloaded kilobytes.

**speed** Shows the average download speed.

**tag** The configured repository tag.

**timeout** Shows the timeout for proxied connections in seconds.

**timeout\_direct** Shows the timeout for direct connections in seconds.

**uptime** Shows the time passed since mounting in minutes.

**usedfd** Shows the number of file descriptors currently issued to file system clients.

**version** Shows the version of the loaded CernVM-FS binary.

Extended attributes can be queried using the `attr` command. For instance, `attr -g hash /cvmfs/atlas.cern.ch/ChangeLog` returns the cryptographic hash of the file at hand. The extended attributes are used by the `cvmfs_config stat` command in order to show a current overview of health and performance numbers.

## Repository Publishing

Repositories are not immutable, every now and then they get updated. This might be installation of a new release or a patch for an existing release. But, of course, each time only a small portion of the repository is touched, say out of . In order not to re-process an entire repository on every update, we create a read-write file system interface to a CernVM-FS repository where all changes are written into a distinct scratch area.

### Read-write Interface using a Union File System

Union file systems combine several directories into one virtual file system that provides the view of merging these directories. These underlying directories are often called *branches*. Branches are ordered; in the case of operations on paths that exist in multiple branches, the branch selection is well-defined. By stacking a read-write branch on top of a read-only branch, union file systems can provide the illusion of a read-write file system for a read-only file system. All changes are in fact written to the read-write branch.

Preserving POSIX semantics in union file systems is non-trivial; the first fully functional implementation has been presented by Wright et al. [*Wright04*]. By now, union file systems are well established for “Live CD” builders, which use a RAM disk overlay on top of the read-only system partition in order to provide the illusion of a fully read-writable system. CernVM-FS supports both aufs and OverlayFS union file systems.

Union file systems can be used to track changes on CernVM-FS repositories (Figure *below*). In this case, the read-only file system interface of CernVM-FS is used in conjunction with a writable scratch area for changes.

Fig. 2.6: A union file system combines a CernVM-FS read-only mount point and a writable scratch area. It provides the illusion of a writable CernVM-FS mount point, tracking changes on the scratch area.

Based on the read-write interface to CernVM-FS, we create a feed-back loop that represents the addition of new software releases to a CernVM-FS repository. A repository in base revision  $r$  is mounted in read-write mode on the publisher’s end. Changes are written to the scratch area and, once published, are re-mounted as repository revision  $r + 1$ . In this way, CernVM-FS provides snapshots. In case of errors, one can safely resume from a previously committed revision.

## Security Considerations

CernVM-FS provides end-to-end data integrity and authenticity using a signed Merkle Tree. CernVM-FS clients verify the signature and the content hashes of all downloaded data. Once a particular revision of a file system is stored in a client's local cache, the client will not apply an older revision anymore.

The public key used to ultimately verify a repository's signature needs to be distributed to clients through a channel different from CernVM-FS content distribution. In practice, these public keys are distributed as part of the source code or through `cvmfs-config-...` packages. One or multiple public keys can be configured for a repository (the *fully qualified repository name*), all repositories within a specific domain (like `*.cern.ch`) or all repositories (`*`). If multiple keys are configured, it is sufficient if any of them validates a signature.

Besides the client, data is also verified by the replication code (Stratum 1 or preloaded cache) and by the release manager machine in case the repository is stored in S3 and not on a local file system.

CernVM-FS does **not** provide data confidentiality out of the box. By default data is transferred through HTTP and thus only public data should be stored on CernVM-FS. However, CernVM-FS can be operated with HTTPS data transport. In combination with client-authentication using an authz helper (see Section *Authorization Helpers*), CernVM-FS can be configured for end-to-end data confidentiality.

Once downloaded and stored in a cache, the CernVM-FS client fully trusts the cache. Data in the cache can be checked for silent corruption but no integrity re-check takes place.

## Signature Details

Creating and validating a repository signature is a two-step process. The *repository manifest* (the file `.cvmfspublished`) is signed by a private RSA key whose public part is stored in the form of an X.509 certificate in the repository. The fingerprint of all certificates that are allowed to sign a repository is stored on a *repository whitelist* (the file `.cvmfswhitelist`). The whitelist is signed with a different RSA key, the *repository master key*. Only the public part of this master key needs to be distributed to clients.

The X.509 certificate currently only serves as an envelope for the public part of a repository key. No further certificate validation takes place.

The repository manifest contains, among other information, the content hash of the root file catalog, the content hash of the signing certificate, the fully qualified repository name, and a timestamp. In order to sign the manifest, the content of the manifest is hashed and encrypted with a private repository key. The timestamp and repository name are used prevent replay attacks.

The whitelist contains the fully qualified repository name, a creation timestamp, an expiry timestamp, and the certificate fingerprints. Since the whitelist expires, it needs to be regularly resigned. The default validity period of a whitelist is 30 days.

The private part of the repository key needs to be accessible on the release manager machine. The private key of the repository master key used to sign the whitelist *can* be maintained on the release manager machine. We recommend, however, to use a smart card to store this private key and to sign a repository whitelist on a dedicated key-signing machine.

## Content Hashes

CernVM-FS supports multiple content hash algorithms: SHA-1 (default), RIPEMD-160, and SHAKE-128 with 160 output bits. The content hash algorithm can be changed with every repository publish operation. Files and file catalogs hashed with different content hash algorithms can co-exist. On changing the algorithm, new and changed files are hashed with the new algorithm, existing data remains unchanged. That allows seamless migration from one algorithm to another.

## Local UNIX Permissions

Most parts of CernVM-FS do not require root privileges. On the server side, only creating and deleting a repository (or replica) requires root privileges. Repository transactions and snapshots can be performed with an unprivileged user account. In order to remount a new file system revision after publishing a transaction, the release manager machines uses a custom `suid` binary.

On client side, the CernVM-FS fuse module is normally started as root. It drops root privileges and changes the persona to the `cvmfs` user early in the file system initialization. The client RPM package installs SELinux rules for RHEL6 and RHEL7. The cache directory should be labeled as `cvmfs_cache_t`.

## CernVM-FS Software Distribution

CernVM-FS software is distributed through HTTPS in packages. There are yum and apt repositories for Linux and `pkg` packages for OS X. Software is available from HTTPS servers. The Linux packages and repositories are signed with a GPG key.

## CernVM-FS Parameters

### Client parameters

Parameters recognized in configuration files under `/etc/cvmfs`:

Parameter	Meaning
<code>CVMFS_ALIEN_CACHE</code>	If set, use an alien cache at the given location
<code>CVMFS_ALT_ROOT_PATH</code>	If set to <i>yes</i> , use alternative root catalog path. Only required for fixed catalogs (tag /
<code>CVMFS_AUTO_UPDATE</code>	If set to <i>no</i> , disables the automatic update of file catalogs.



Table 3.1 – continued from previous page

Parameter	Meaning
CVMFS_AUTHZ_HELPER	Full path to an authz helper, overwrites the helper hint in the catalog.
CVMFS_AUTHZ_SEARCH_PATH	Full path to the directory that contains the authz helpers.
CVMFS_BACKOFF_INIT	Seconds for the maximum initial backoff when retrying to download data.
CVMFS_BACKOFF_MAX	Maximum backoff in seconds when retrying to download data.
CVMFS_CACHE_BASE	Location (directory) of the CernVM-FS cache.
CVMFS_CHECK_PERMISSIONS	If set to <i>no</i> , disable checking of file ownership and permissions (open all files).
CVMFS_CLAIM_OWNERSHIP	If set to <i>yes</i> , allows CernVM-FS to claim ownership of files and directories.
CVMFS_DEBUGLOG	If set, run CernVM-FS in debug mode and write a verbose log to the specified file.
CVMFS_DEFAULT_DOMAIN	The default domain will be automatically appended to repository names when given.
CVMFS_DNS_RETRIES	Number of retries when resolving proxy names
CVMFS_DNS_TIMEOUT	Timeout in seconds when resolving proxy names
CVMFS_EXTERNAL_FALLBACK_PROXY	List of HTTP proxies similar to CVMFS_EXTERNAL_HTTP_PROXY. The fallback proxy is used if the main proxy fails.
CVMFS_EXTERNAL_HTTP_PROXY	Chain of HTTP proxy groups to be used when CernVM-FS is accessing external data.
CVMFS_EXTERNAL_TIMEOUT	Timeout in seconds for HTTP requests to an external-data server with a proxy server.
CVMFS_EXTERNAL_TIMEOUT_DIRECT	Timeout in seconds for HTTP requests to an external-data server without a proxy server.
CVMFS_EXTERNAL_URL	Semicolon-separated chain of web servers serving external data chunks.
CVMFS_FALLBACK_PROXY	List of HTTP proxies similar to CVMFS_HTTP_PROXY. The fallback proxies are used if the main proxy fails.
CVMFS_FOLLOW_REDIRECTS	When set to <i>yes</i> , follow up to 4 HTTP redirects in requests.
CVMFS_HIDE_MAGIC_XATTRS	If set to <i>yes</i> the client will not expose CernVM-FS specific extended attributes.
CVMFS_HOST_RESET_AFTER	See CVMFS_PROXY_RESET_AFTER.
CVMFS_HTTP_PROXY	Chain of HTTP proxy groups used by CernVM-FS. Necessary. Set to DIRECT if you don't want to use proxies.
CVMFS_IGNORE_SIGNATURE	When set to <i>yes</i> , don't verify CernVM-FS file catalog signatures.
CVMFS_INITIAL_GENERATION	Initial inode generation. Used for testing.
CVMFS_IPFAMILY_PREFER	Which IP protocol to prefer when connecting to proxies. Can be either 4 or 6.
CVMFS_KCACHE_TIMEOUT	Timeout for path names and file attributes in the kernel file system buffers.
CVMFS_KEYS_DIR	Directory containing *.pub files used as repository signing keys. If set, this parameter overrides the default location.
CVMFS_LOW_SPEED_LIMIT	Minimum transfer rate a server or proxy must provide.
CVMFS_MAX_IPADDR_PER_PROXY	Limit the number of IP addresses a proxy name resolves into. From all registered addresses.
CVMFS_MAX_RETRIES	Maximum number of retries for a given proxy/host combination.
CVMFS_MAX_TTL	Maximum file catalog TTL in minutes. Can overwrite the TTL stored in the catalog.
CVMFS_MEMCACHE_SIZE	Size of the CernVM-FS meta-data memory cache in Megabyte.
CVMFS_MOUNT_RW	Mount CernVM-FS as a read/write file system. Write operations will fail but this option is necessary for some applications.
CVMFS_NFILES	Maximum number of open file descriptors that can be used by the CernVM-FS process.
CVMFS_NFS_SOURCE	If set to <i>yes</i> , act as a source for the NFS daemon (NFS export).
CVMFS_NFS_SHARED	If set a path, used to store the NFS maps in an SQLite database, instead of the usual location.
CVMFS_PAC_URLS	Chain of URLs pointing to PAC files with HTTP proxy configuration information.
CVMFS_PROXY_RESET_AFTER	Delay in seconds after which CernVM-FS will retry the primary proxy group in case of failure.
CVMFS_PROXY_TEMPLATE	Overwrite the default proxy template in Geo-API calls. Only needed for debugging.
CVMFS_PUBLIC_KEY	Colon-separated list of repository signing keys.
CVMFS_QUOTA_LIMIT	Soft-limit of the cache in Megabyte.
CVMFS_RELOAD_SOCKETS	Directory of the sockets used by the CernVM-FS loader to trigger hotpatching/reloading.
CVMFS_REPOSITORIES	Comma-separated list of fully qualified repository names that shall be mountable under the current path.
CVMFS_REPOSITORY_DATE	A timestamp in ISO format (e.g. 2007-03-01T13:00:00Z). Selects the repository to use.
CVMFS_REPOSITORY_TAG	Select a named repository snapshot that should be mounted instead of <code>trunk</code> .
CVMFS_CONFIG_REPO_REQUIRED	If set to <i>yes</i> , no repository can be mounted unless the config repository is available.
CVMFS_ROOT_HASH	Hash of the root file catalog, implies CVMFS_AUTO_UPDATE=no.
CVMFS_SEND_INFO_HEADER	If set to <i>yes</i> , include the <code>cvmfs</code> path of downloaded data in HTTP headers.
CVMFS_SERVER_CACHE_MODE	Enable special cache semantics for a client used as a release manager repository base.
CVMFS_SERVER_URL	Semicolon-separated chain of Stratum-1 servers.

Table 3.1 – continued from previous page

Parameter	Meaning
CVMFS_SHARED_CACHE	If set to <i>no</i> , makes a repository use an exclusive cache.
CVMFS_STRICT_MOUNT	If set to <i>yes</i> , mount only repositories that are listed in CVMFS_REPOSITORIES.
CVMFS_SYSLOG_FACILITY	If set to a number between 0 and 7, uses the corresponding LOCALn\$ facility for s
CVMFS_SYSLOG_LEVEL	If set to 1 or 2, sets the syslog level for CernVM-FS messages to LOG_DEBUG or L
CVMFS_SYSTEMD_NOKILL	If set to <i>yes</i> , modify the command line to @vmfs2 . . . in order to act as a systemd
CVMFS_TIMEOUT	Timeout in seconds for HTTP requests with a proxy server.
CVMFS_TIMEOUT_DIRECT	Timeout in seconds for HTTP requests without a proxy server.
CVMFS_TRACEFILE	If set, enables the tracer and trace file system calls to the given file.
CVMFS_USE_GEOAPI	Request order of Stratum 1 servers and fallback proxies via Geo-API.
CVMFS_USER	Sets the <code>gid</code> and <code>uid</code> mount options. Don't touch or overwrite.
CVMFS_USYSLOG	All messages that normally are logged to syslog are re-directed to the given file. This

## Server parameters

Parameter	Meaning
CVMFS_AUFS_WARNING	Set to <i>false</i> to silence AUFS kernel deadlock warning.
CVMFS_AUTO_GC	Enables the automatic garbage collection on <i>publish</i> and <i>snapshot</i>
CVMFS_AUTO_GC_TIMESPAN	Date-threshold for automatic garbage collection (For example: <i>3 days ago</i> ,
CVMFS_AUTO_REPAIR_MOUNTPOINT	Set to <i>true</i> to enable automatic recovery from bogus server mount states.
CVMFS_AUTO_TAG	Creates a generic revision tag for each published revision (if set to <i>true</i> ).
CVMFS_AUTO_TAG_TIMESPAN	Date-threshold for automatic tags, after which auto tags get removed (For e
CVMFS_AUTOCATALOGS	Enable/disable automatic catalog management using autocatalogs.
CVMFS_AUTOCATALOGS_MAX_WEIGHT	Maximum number of entries in an autocatalog to be considered overflowed.
CVMFS_AUTOCATALOGS_MIN_WEIGHT	Minimum number of entries in an autocatalog to be considered underflowed.
CVMFS_AVG_CHUNK_SIZE	Desired Average size of a file chunk in bytes (see also CVMFS_USE_FILE_CHUNK
CVMFS_CATALOG_ALT_PATHS	Enable/disable generation of catalog bootstrapping shortcuts during publish
CVMFS_CATALOG_ENTRY_WARN_THRESHOLD	Threshold of catalog entry count before triggering a warning message.
CVMFS_COMPRESSION_ALGORITHM	Compression algorithm to be used during publishing (currently either 'deflate'
CVMFS_CREATOR_VERSION	The CernVM-FS version that was used to create this repository (do not chan
CVMFS_DONT_CHECK_OVERLAYFS_VERSION	Disable checking of OverlayFS version before usage. Using OverlayFS in k
CVMFS_EXTERNAL_DATA	Set to <i>true</i> to mark repository to contain external data that is served from an
CVMFS_FORCE_REMOUNT_WARNING	Enable/disable warning through <code>wall</code> and grace period before forcefully re
CVMFS_GARBAGE_COLLECTION	Enables repository garbage collection (Stratum~0 only   if set to <i>true</i> )
CVMFS_GC_DELETION_LOG	Log file path to track all garbage collected objects during sweeping for boot
CVMFS_HASH_ALGORITHM	Define which secure hash algorithm should be used by CernVM-FS for CA
CVMFS_IGNORE_XDIR_HARDLINKS	If set to <i>yes</i> , do not abort the publish operation when cross-directory hardlin
CVMFS_INCLUDE_XATTRS	Set to <i>true</i> to process extended attributes
CVMFS_MAX_CHUNK_SIZE	Maximal size of a file chunk in bytes (see also CVMFS_USE_FILE_CHUNK
CVMFS_MAXIMAL_CONCURRENT_WRITES	Maximal number of concurrently processed files during publishing.
CVMFS_MIN_CHUNK_SIZE	Minimal size of a file chunk in bytes (see also CVMFS_USE_FILE_CHUNK
CVMFS_NUM_WORKERS	Maximal number of concurrently downloaded files during a Stratum1 pull o
CVMFS_PUBLIC_KEY	Path to the public key file of the repository to be replicated. (Stratum 1 only
CVMFS_REPLICA_ACTIVE	Stratum1-only: Set to <i>no</i> to skip this Stratum1 when executing <code>cv</code>
CVMFS_REPOSITORY_NAME	The fully qualified name of the specific repository.
CVMFS_REPOSITORY_TYPE	Defines if the repository is a master copy ( <i>stratum0</i> ) or a replica ( <i>stratum1</i> ).
CVMFS_SPOOL_DIR	Location of the upstream spooler scratch directories; the read-only CernVM
CVMFS_STRATUM0	URL of the master copy ( <i>stratum0</i> ) of this specific repository.
CVMFS_STRATUM1	URL of the Stratum1 HTTP server for this specific repository.

Table 3.2 – continued from previous page

Parameter	Meaning
CVMFS_UNION_DIR	Mount point of the union file system for copy-on-write semantics of CernVM-FS.
CVMFS_UNION_FS_TYPE	Defines the union file system to be used for the repository. (currently <i>aufs</i> and <i>overlayfs</i> )
CVMFS_UPSTREAM_STORAGE	Upstream spooler description defining the basic upstream storage type and options.
CVMFS_USE_FILE_CHUNKING	Allows backend to split big files into small chunks ( <i>true</i>   <i>false</i> )
CVMFS_USER	The user name that owns and manipulates the files inside the repository.
CVMFS_VOMS_AUTHZ	Membership requirement (e.g. VOMS authentication) to be added into the file system.

## CernVM-FS Server Infrastructure

This section provides technical details on the CernVM-FS server setup including the infrastructure necessary for an individual repository. It is highly recommended to first consult “*Notable CernVM-FS Server Locations and Files*” for a more general overview of the involved directory structure.

### Prerequisites

A CernVM-FS server installation depends on the following environment setup and tools to be in place:

- kernel 4.2.x or later (for OverlayFS) *or* aufs support built into the kernel (see Section *Installing the AUFS-enabled Kernel on Scientific Linux 6*)
- Backend storage location available through HTTP
- Backend storage accessible at `/srv/cvmfs/...` (unless stored on S3)
- `cvmfs` and `cvmfs-server` packages installed

### Local Backend Storage Infrastructure

CernVM-FS stores the entire repository content (file content and meta-data catalogs) into a content addressable storage (CAS). This storage can either be a file system at `/srv/cvmfs` or an S3 compatible object storage system (see “*Confidential Repositories*” for details). In the former case the contents of `/srv/cvmfs` are as follows:

File Path	Description
/srv/cvmfs	<b>Central repository storage location</b> Can be mounted or symlinked to another location <i>before</i> creating the first repository.
/srv/cvmfs/ <fqrn>	<b>Storage location of a specific repository</b> Can be symlinked to another location <i>before</i> creating the repository <fqrn>. This location needs to be both writable by the repository owner and accessible through an HTTP server.
/srv/cvmfs/ <fqrn>/. cvmfspublished	<b>Manifest file of the repository</b> The manifest provides the entry point into the repository. It is the only file that needs to be signed by the repository's private key.
/srv/cvmfs/ <fqrn>/. cvmfswitelist	<b>List of trusted repository certificates</b> Contains a list of certificate fingerprints that should be allowed to sign a repository manifest (see .cvmfspublished). The whitelist needs to be signed by a globally trusted private key.
/srv/cvmfs/ <fqrn>/data	<b>CAS location of the repository</b> Data storage of the repository. Contains catalogs, files, file chunks, certificates and history databases in a content addressable file format. This directory and all its contents need to be writable by the repository owner.
/srv/cvmfs/ <fqrn>/data/00.. ff	<b>Second CAS level directories</b> Splits the flat CAS namespace into multiple directories. First two digits of the file content hash defines the directory the remainder is used as file name inside the corresponding directory.
/srv/cvmfs/ <fqrn>/data/txn	<b>CAS transaction directory</b> Stores partial files during creation. Once writing has completed, the file is committed into the CAS using an atomic rename operation.

## Server Spool Area of a Repository (Stratum0)

The spool area of a repository contains transaction infrastructure and scratch area of a Stratum0 or specifically a release manager machine installation. It is always located inside `/var/spool/cvmfs` with directories for individual repositories. Note that the data volume of the spool area can grow very large for massive repository updates since it contains the writable union file system branch and a CernVM-FS client cache directory.

File Path	Description
<code>/var/spool/cvmfs</code>	<b>CernVM-FS server spool area</b> Contains administrative and scratch space for CernVM-FS repositories. This directory should only contain directories corresponding to individual CernVM-FS repositories.
<code>/var/spool/cvmfs/&lt;fqrn&gt;</code>	<b>Individual repository spool area</b> Contains the spool area of an individual repository and might temporarily contain large data volumes during massive repository updates. This location can be mounted or symlinked to other locations. Furthermore it must be writable by the repository owner.
<code>/var/spool/cvmfs/&lt;fqrn&gt;/cache</code>	<b>CernVM-FS client cache directory</b> Contains the cache of the CernVM-FS client mounting the r/o branch (i.e. <code>/var/spool/cvmfs/&lt;fqrn&gt;/rdonly</code> ) of the union file system mount point located at <code>/cvmfs/&lt;fqrn&gt;</code> . The content of this directory is fully managed by the CernVM-FS client and hence must be configured as a CernVM-FS cache and writable for the repository owner.
<code>/var/spool/cvmfs/&lt;fqrn&gt;/rdonly</code>	<b>CernVM-FS client mount point</b> Serves as the mount point of the CernVM-FS client exposing the latest published state of the CernVM-FS repository. It needs to be owned by the repository owner and should be empty if CernVM-FS is not mounted to it.
<code>/var/spool/cvmfs/&lt;fqrn&gt;/scratch</code>	<b>Writable union file system scratch area</b> All file system changes applied to <code>/cvmfs/&lt;fqrn&gt;</code> during a transaction will be stored in this directory. Hence, it potentially needs to accommodate a large data volume during massive repository updates. Furthermore it needs to be writable by the repository owner.
<code>/var/spool/cvmfs/&lt;fqrn&gt;/tmp</code>	<b>Temporary scratch location</b> Some CernVM-FS server operations like publishing store temporary data files here, hence it needs to be writable by the repository owner. If the repository is idle this directory should be empty.
<code>/var/spool/cvmfs/&lt;fqrn&gt;/client.config</code>	<b>CernVM-FS client configuration</b> This contains client configuration variables for the CernVM-FS client mounted to <code>/var/spool/cvmfs/&lt;fqrn&gt;/rdonly</code> . Most notably it needs to contain <code>CVMFS_ROOT_HASH</code> configured to the latest revision published in the corresponding repository. This file needs to be writable by the repository owner.

## Repository Configuration Directory

The authoritative configuration of a CernVM-FS repository is located in `/etc/cvmfs/repositories.d` and should only be writable by the administrator. Furthermore the repository's keychain is located in `/etc/cvmfs/keys` and follows the naming convention `<fqrn>.cert` for the certificate, `<fqrn>.key` for the repository's private key and `<fqrn>.pub` for the public key. All of those files can be symlinked somewhere else if necessary.

File Path	Description
/etc/cvmfs/repositories.d	<b>CernVM-FS server config directory</b> This contains the configuration directories for individual CernVM-FS repositories. Note that this path is shortened using <code>../repos.d/</code> in the rest of this table.
../repos.d/<fqrn>	<b>Config directory for specific repo</b> This contains the configuration files for one specific CernVM-FS repository server.
../repos.d/<fqrn>/server.conf	<b>Server configuration file</b> Authoritative configuration file for the CernVM-FS server tools. This file should only contain <i>valid server configuration variables</i> as it controls the behaviour of the CernVM-FS server operations like publishing, pulling and so forth.
../repos.d/<fqrn>/client.conf	<b>Client configuration file</b> Authoritative configuration file for the CernVM-FS client used to mount the latest revision of a Stratum 0 release manager machine. This file should only contain <i>valid client configuration variables</i> . This file must not exist for Stratum 1 repositories.
../repos.d/<fqrn>/replica.conf	<b>Replication configuration file</b> Contains configuration variables for Stratum 1 specific repositories. This file must not exist for Stratum 0 repositories.

## Environment Setup

Apart from file and directory locations a CernVM-FS server installation depends on a few environment configurations. Most notably the possibility to access the backend storage through HTTP and to allow for mounting of both the CernVM-FS client at `/var/spool/cvmfs/<fqrn>/rdonly` and a union file system on `/cvmfs/<fqrn>`.

Granting HTTP access can happen in various ways and depends on the chosen backend storage type. For an S3 hosted backend storage, the CernVM-FS client can usually be directly pointed to the S3 bucket used for storage (see “*Confidential Repositories*” for details). In case of a local file system backend any web server can be used for this purpose. By default CernVM-FS assumes Apache and uses that automatically.

Internally the CernVM-FS server uses a SUID binary (i.e. `cvmfs_suid_helper`) to manipulate its mount points. This is necessary since transactional CernVM-FS commands must be accessible to the repository owner that is usually different from root. Both the mount directives for `/var/spool/cvmfs/<fqrn>/rdonly` and `/cvmfs/<fqrn>` must be placed into `/etc/fstab` for this reason. By default CernVM-FS uses the following entries for these mount points:

```
cvmfs2#<fqrn> /var/spool/cvmfs/<fqrn>/rdonly fuse \
allow_other,config=/etc/cvmfs/repositories.d/<fqrn>/client.conf: \
/var/spool/cvmfs/<fqrn>/client.local,cvmfs_suid 0 0

aufs_<fqrn> /cvmfs/<fqrn> aufs br=/var/spool/cvmfs/<fqrn>/scratch=rw: \
/var/spool/cvmfs/<fqrn>/rdonly=rr,udba=none,ro 0 0
```

## Available RPMs

The CernVM-FS software is available in form of several RPM packages:

**cvmfs-release** Adds the CernVM-FS yum/apt repository.

**cvmfs-config-default** Contains a configuration and public keys suitable for nodes in the Worldwide LHC Computing Grid. Provides access to repositories in the `cern.ch`, `egi.eu`, and `opensciencegrid.org` domains.

**cvmfs-config-none** Empty package to satisfy the `cvmfs-config` requirement of the `cvmfs` package without actually installing any configuration.

**cvmfs** Contains the Fuse module and additional client tools. It has dependencies to at least one of the `cvmfs-config-...` packages.

**cvmfs-devel** Contains the `libcvmfs.a` static library and the `libcvmfs.h` header file for use of CernVM-FS with Parrot *[Thain05]*.

**cvmfs-auto-setup** Only available through yum. This is a wrapper for `cvmfs_config setup`. This is supposed to provide automatic configuration for the ATLAS Tier3s. Depends on `cvmfs`.

**cvmfs-server** Contains the CernVM-FS server tool kit for maintaining Stratum 0 and Stratum 1 servers.

**kernel-...-aufs21** Scientific Linux 6 kernel with aufs. Required for SL6 based Stratum 0 servers.

**cvmfs-unittests** Contains the `cvmfs_unittests` binary. Only required for testing.

## References





## CHAPTER 4

---

### Contact and Authors

---

Visit our website on [cernvm.cern.ch](http://cernvm.cern.ch).

Authors of this documentation:

- Jakob Blomer
- Predrag Buncic
- Dave Dykstra
- René Meusel
- José Molina Colmenero
- Radu Popescu



---

## Bibliography

---

- [Blumenfeld08] Blumenfeld, B. et al. 2008. CMS conditions data access using FroNTier. *Journal of Physics: Conference Series*. 119, (2008).
- [Callaghan95] Callaghan, B. et al. 1995. *NFS Version 3 Protocol Specification*. Technical Report #1813. Internet Engineering Task Force.
- [Gauthier99] Gauthier, P. et al. 1999. *Web proxy auto-discovery protocol*. IETF Secretariat.
- [Guerrero99] Guerrero, D. 1999. Caching the web, part 2. *Linux Journal*. 58 (February 1999).
- [Panagiotou06] Panagiotou, K. and Souza, A. 2006. On adequate performance measures for paging. *Annual ACM Symposium on Theory Of Computing*. 38, (2006), 487-496.
- [Schubert08] Schubert, M. et al. 2008. *Nagios 3 enterprise network monitoring*. Syngress.
- [Shepler03] Shepler, S. et al. 2003. *Network File System (NFS) version 4 Protocol*. Technical Report #3530. Internet Engineering Task Force.
- [Jones01] 3rd, D.E. and Jones, P. 2001. *US Secure Hash Algorithm 1 (SHA1)*. Technical Report #3174. Internet Engineering Task Force.
- [Dobbertin96] Dobbertin, H. et al. 1996. RIPEMD-160: A strengthened version of RIPEMD. Springer. 71-82.
- [Bertoni09] Bertoni, G., Daemen, J., Peeters, M. and Van Assche, G., 2009. Keccak sponge function family main document. Submission to NIST (Round 2), 3, p.30.
- [Rivest92] Rivest, R. 1992. *The MD5 Message-Digest Algorithm*. Technical Report #1321. Internet Engineering Task Force.
- [Turner11] Turner, S. and Chen, L. 2011. *Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms*. Technical Report #6151. Internet Engineering Task Force.
- [Deutsch96] Deutsch, P. and Gailly, J.-L. 1996. *ZLIB Compressed Data Format Specification version 3.3*. Technical Report #1950. Internet Engineering Task Force.
- [Allen10] Allen, G. and Owens, M. 2010. *The definitive guide to SQLite*. Apress.
- [Wright04] Wright, C.P. et al. 2004. *Versatility and unix semantics in a fan-out unification file system*. Technical Report #FSL-04-01b. Stony Brook University.
- [BernersLee96] Berners-Lee, T. et al. 1996. *Hypertext Transfer Protocol - HTTP/1.0*. Technical Report #1945. Internet Engineering Task Force.

- [Fielding99] Fielding, R. et al. 1999. *Hypertext Transfer Protocol - HTTP/1.1*. Technical Report #2616. Internet Engineering Task Force.
- [Compostella10] Compostella, G. et al. 2010. CDF software distribution on the Grid using Parrot. *Journal of Physics: Conference Series*. 219, (2010).
- [Thain05] Thain, D. and Livny, M. 2005. Parrot: an application environment for data-intensive computing. *Scalable Computing: Practice and Experience*. 6, 3 (18 2005), 9.
- [Suzaki06] Suzaki, K. et al. 2006. HTTP-FUSE Xenoppix. *Proc. of the 2006 linux symposium* (2006), 379-392.
- [Freedman03] Freedman, M.J. and Mazières, D. 2003. Sloppy hashing and self-organizing clusters. M.F. Kaashoek and I. Stoica, eds. Springer. 45-55.
- [Nygren10] Nygren, E. et al. 2010. The Akamai network: A platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review*. 44, 3 (2010), 2-19.
- [Tolia03] Tolia, N. et al. 2003. Opportunistic use of content addressable storage for distributed file systems. *Proc. of the uSENIX annual technical conference* (2003).
- [Mockapetris87] Mockapetris, P. 1987. *Domain names - implementation and specification*. Technical Report #1035. Internet Engineering Task Force.
- [Dykstra10] Dykstra, D. and Lueking, L. 2010. Greatly improved cache update times for conditions data with frontier/Squid. *Journal of Physics: Conference Series*. 219, (2010).