CernVM-FS Documentation

Release 2.5.2

CernVM Team

Dec 19, 2018

Contents

CHAPTER 1

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.

CHAPTER 2

Contents

2.1 Release Notes for CernVM-FS 2.5.2

CernVM-FS 2.5.2 is a patch release. It contains bugfixes and improvements for clients, stratum 0 and stratum 1 servers.

As with previous releases, upgrading clients should be seamless just by installing the new package from the repository. As usual, we recommend to update only a few worker nodes first and gradually ramp up once the new version proves to work correctly. Please take special care when upgrading a cvmfs client in NFS mode.

For Stratum 1 servers, there should be no running snapshots during the upgrade. For Release Manager Machines, all transactions must be closed before upgrading. Together with CernVM-FS 2.5.2 we also release the CernVM-FS Gateway Services version 0.3.1.

Note for upgrades from versions prior to 2.5.1: please also see the specific instructions in the release notes for version 2.5.1 and earlier.

2.1.1 Bug Fixes and Improvements

- Client: fix cache cleanup logic for chunks >25M CVM-1625
- Client: fix busy waiting in cache manager communication under heavy load CVM-1618
- Client: fix stale authz session cache when repository membership changes CVM-1653
- Client, macOS: add support for `CVMFS_FUSE_NOTIFY_INVALIDATION`, disable by default on macOS to fix stability issues CVM-1638
- Client: add support for `CVMFS_MAX_[EXTERNAL_]SERVERS` to restrict the effective number of stratum 1 servers CVM-1631
- Client: add support for `CVMFS_DNS_{MIN,MAX}_TTL` for proxy name resolution CVM-1659
- Client: add support for `CVMFS_SUID` config parameter CVM-1591
- Client: log to syslog when session authorization disappears CVM-1658
- Client: change `cvmfs_talk` implementation from perl to C++ to improve speed

- Server: more fine-grained syncfs control through `CVMFS_SYNCFS_LEVEL=[none,default,cautious]`, change default back to 2.5.0 behavior CVM-1646
- Server: update Apache config to ignore If-Modified-Since on stratum 1s CVM-1655
- Server: add retry support for `cvmfs_server transaction` CVM-1611
- Server: restrict lazy downloads of geodb to once a day CVM-1647
- Server: print the start of mark and sweep phases during garbage collection
- Server, replication/preloader: improve error reporting of network failures CVM-1624
- Server: print warnings during publish to stdout instead of stderr CVM-1630
- Server, S3: fix cache control headers for objects pushed to S3 CVM-1606
- Server, S3: fix content type header for objects pushed to S3
- Server, S3: add support for URL subpaths and DNS-style bucket URLs CVM-1641
- Server, S3: add support for `CVMFS_S3_DNS_BUCKETS=false` to disable DNS-style bucket URLs when S3 backend doesn't support them (e.g. Minio) CVM-1641
- Server, gateway: delete session tokens for transactions after use CVM-1643
- Server, gateway: fix permissions on temporary directory used by the receiver
- Server, gateway: automatically apply lease path to abort and publish commands CVM-1601
- Server, gateway: bump gateway protocol version to 2 CVM-1626
- Fix potential memory corruption in catalog traversal
- Fix building on macOS 10.14
- Gateway: fixed bug allowing conflicting leases to be acquired CVM-1642
- Gateway: various improvements to the RPM and DEB packages; the cvmfs-gateway package now depends on `cvmfs-server >= 2.5.2` CVM-1615

2.1.2 Manual Migration from 2.5.1 Stratum 1 Servers

If you do not want to use `cvmfs_server migrate` to automatically upgrade, stratum 1 servers that maintain repositories replicas using the Apache backend can be migrated from version 2.5.1 with the following steps:

1. Ensure that there are no active replication or garbage collection processes before updating the server software and during the repository layout migration.
2. Install the `cvmfs-server` 2.5.2 package.
3. Edit the Apache configuration for the repositories and add to the `<Directory>` section of the repositories `RequestHeader unset If-Modified-Since`
4. Update /etc/cvmfs/repositories.d/<REPOSITORY>/server.conf and set `CVMFS_CREATOR_VERSION=140`

2.1.3 Updating from cvmfs-gateway 0.3.0

The RPM package for `cvmfs-gateway` version 0.3.0 has a bug in the post uninstall scripts which interferes with the setup of the new package, when updating. The old package needs to be manually uninstalled, before the new one is installed:

```
systemctl stop cvmfs-gateway
yum remove cvmfs-gateway
yum install cvmfs-gateway-0.3.1-1.el7.x86_64.rpm
```

2.2 Release Notes for CernVM-FS 2.5.1

CernVM-FS 2.5.1 is a patch release. It contains bugfixes and improvements for clients, stratum 0 and stratum 1 servers. This release also adds support for Ubuntu 18.04 ("bionic").

As with previous releases, upgrading clients should be seamless just by installing the new package from the repository. As usual, we recommend to update only a few worker nodes first and gradually ramp up once the new version proves to work correctly. Please take special care when upgrading a cvmfs client in NFS mode.

For Stratum 1 servers, there should be no running snapshots during the upgrade. For Release Manager Machines, all transactions must be closed before upgrading. Together with CernVM-FS 2.5.1 we also release the CernVM-FS Gateway Services version 0.3.

Note for upgrades from versions prior to 2.5.0: please also see the specific instructions in the release notes for version 2.5.0 and earlier.

2.2.1 Bug Fixes and Improvements

- Client, macOS: Improved check for OSXFUSE in `cvmfs_config chksetup` (CVM-1550)
- Client: avoid mount helper crash if required config repository is missing (CVM-1512)
- Client: Apply catalog updates from updated alien cache (CVM-1515)
- Client: Fix occasional false error of `cvmfs_config probe` on Fedora 28
- Client: add support for CA bundle files through new parameter `X509_CERT_BUNDLE` (CVM-1421)
- Libcvmfs: fix workspace default location
- Server: fix broken repository manifest after catalog migration operations (CVM-1534)
- Server: fix locking bug in cvmfs_server snapshot (CVM-1598)
- Server: fix Yubikey signature handling (CVM-1604)
- Server: flush file system buffers after snapshot, gc, resign, and publish (CVM-1552)
- Server: fix `cvmfs_suid_helper` on Ubuntu 18.04 for symlinked spool directory
- Server: replace deprecated geolite free database by geolite2 (CVM-1496)
- Server, S3: parallelize object removal during garbage collection (CVM-1593)
- Server, S3: Make S3 network parameters adjustable, new parameters `CVMFS_S3_MAX_RETRIES` and `CVMFS_S3_TIMEOUT`
- Gateway: handle spooler failures gracefully in the gateway receiver (CVM-1545)
- Gateway: fix publishing with `CVMFS_AUTO_TAGS=false` (CVM-1559)
- Gateway: fix potential deadlock when uploading files to the repository storage (CVM-1555)
- Gateway: fix hard link handling (CVM-1542)
- Gateway: terminate gracefully when reflog is missing (CVM-1560)
- Gateway: performance improvements for publishing

- Fix potential memory corruption on gcc7+ in tiered cache manager and cvmfs_swissknife
- Fix compilation with libattr >= 2.4.48
- Fix compiler optimization flags for gcc8+ and macOS clang9+

2.3 Release Notes for CernVM-FS 2.5.0

CernVM-FS 2.5 is a feature release that comes with performance improvements, new functionality, and bugfixes. We would like to thank Dave Dykstra (FNAL), Brian Bockelman (U. Nebraska) and Ben Tovar (U. Notre Dame) for their contributions to this release!

This release comes with the new Repository Gateway Services that allow for multiple release managers operating concurrently on different subtrees of a repository.

This release also comes with rewritten code for the processing of new files. This was necessary to address several lurking deadlocks. This change should be transparent to users.

Other notable changes include

- Support for AWSv4 authorization protocol in the S3 backend
- Removal of the "multi-bucket" support in the S3 backend (this feature was aimed at a specific, now outdated hardware product)
- Allow for automatic but infrequent garbage collection
- Support for publishing special files (named pipes, sockets, device files)
- Client can adjust itself to a change of the DNS servers
- New platforms: Fedora 26 and 27 on x86_64, macOS 10.11+

As with previous releases, upgrading should be seamless just by installing the new package from the repository. As usual, we recommend to update only a few worker nodes first and gradually ramp up once the new version proves to work correctly. Please take special care when upgrading a client in NFS mode.

For Stratum 0 servers, all transactions must be closed before upgrading. For Stratum 1 servers, there should be no running snapshots during the upgrade. After the software upgrade, both stratum 0 and 1 servers require doing `cvmfs_server migrate` for each repository.

Note: if the configuration of the Stratum 0/1 server is handled by a configuration management system (Puppet, Chef, . . .), please see Section *Manual Migration from 2.4.4 Release Manager Machines*.

2.3.1 Gateway Services

The new CernVM-FS Gateway Services allow for distributed server deployments. This can be used for multi-tenant repositories, where every tenant takes ownership of a specific repository subtree. It can also be used to parallelize publishing of content if the different change sets are limited to a specific subtree.

The gateway services come as separate packages. They control the access to the storage and they need to be installed on a central machine. Multiple release manager machines can then be installed that use the gateway service to operate on the same repository.

Detailed documentation is available in Chapter *CernVM-FS Repository Gateway and Release Managers*.

2.3.2 Automatic, Infrequent Garbage Collection

The new parameter `CVMFS_AUTO_GC_LAPSE` can be used on stratum 0 and stratum 1 to specify how often the garbage collection should run (CVM-1400).

It works like the existing `CVMFS_..._TIMESPAN` parameters with a string that is parsed by the `date` utility. The default setting is `1 day ago`, meaning that garbage collection runs on publish if the last garbage collection (manual or automatic) was more that one day ago.

2.3.3 Bug Fixes

- Client: fix crash in `cvmfs_talk remount` with fixed repository snapshot
- Client: fix retry of repository manifest download in "offline mode"
- Client: fix statvfs for cache size >4G on macOS (CVM-1474)
- Client: use lazy unmount as a last resort in `cvmfs_config killall` (CVM-1465)
- Client: Fix storage location of the catalog checksum destination in certain rare cache configurations (CVM-962)
- Client: fix error message when trying to mount an already mounted repo (CVM-1477)
- Server: fix garbage collection of idle repositories (CVM-1460)
- Server: use `systemd start <mount unit>` in suid helper if applicable (CVM-1398)
- Server: fix transaction abort with many temporary files (CVM-1390)
- Server: place bootstrapping symlinks on replica storage (CVM-1366)
- Server: sanitize repository names in cvmfs_server (CVM-1389)
- Server: check for autofs in `cvmfs_server rmfs` only for stratum 0s (CVM-1490)
- Server: fix warnings with bash >= 4.4 (CVM-1401)

2.3.4 Other Improvements

- Client: don't enforce `user_allow_other` fuse option (CVM-1379)
- Client: use /etc/auto.master.d/cvmfs.autofs if applicable (CVM-675)
- Client: improve CPU utilization when downloading with limited bandwidth (CVM-1480)
- Client: send "offline mode" enter/recover events to syslog (CVM-1497)
- Client: implement `CVMFS_DNS_ROAMING` on Linux (CVM-496)
- Client: increase default cache limit to 20G on macOS
- Client: use `CVMFS_MAX_IPADDR_PER_PROXY=2` by default on macOS
- Client: automatically restart failed authz helper after cool-off period
- Client: create libcvmfs.a and libcvmfs_cache.a on macOS (CVM-1489)
- Server: use AWSv4 S3 authorization if `CVMFS_S3_REGION` is set (CVM-988)
- Server: add `CAP_DAC_READ_SEARCH` to swissknife to publish locked-down files
- Server: add support for diff snapshots based on root hash (CVM-1452)
- Server: add `cvmfs_server tag -b` to print the hierarchy of branches (CVM-1392)

- Server: make `CVMFS_GENERATE_LEGACY_BULK_CHUNKS=false` the default (CVM-1429)
- Server: add CloudFlare support to GeoAPI (CVM-1468)
- Server: set httpd selinux label for GeoIP database (CVM-1454)
- Server: new server parameter `CVMFS_IGNORE_SPECIAL_FILES`

2.3.5 Manual Migration from 2.4.4 Release Manager Machines

If you do not want to use `cvmfs_server migrate` to automatically upgrade, release manager machines that maintain Stratum 0 repositories as well as web servers serving stratum 0/1 repositories can be migrated from version 2.4.4 with the following steps:

1. Ensure that there are no open transactions and no active replication or garbage collection processes before updating the server software and during the repository layout migration.
2. Install the `cvmfs-server` 2.5 package.
3. *Only on release manager machines*: Adjust the /etc/fstab entries for union file system mount (/cvmfs/. . .) of the repositories: add the `nodev` mount option after the `noauto` mount option.
4. *Only on systemd managed release manager machines*: Ensure that the mount units for all the repositories exist by running

```
/usr/lib/systemd/system-generators/systemd-fstab-generator \
  /run/systemd/generator '' '' 2>/dev/null
systemctl daemon-reload
```

On both stratum 0 and stratum 1 servers

5. Update /etc/cvmfs/repositories.d/<REPOSITORY>/server.conf and set `CVMFS_CREATOR_VERSION=139`

On release manager machines, in agreement with the repository owner it's recommended to make a test publish

```
cvmfs_server transaction <REPOSITORY>
cvmfs_server publish <REPOSITORY>
```

before resuming normal operation.

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. 1: A CernVM-FS client provides a virtual file system that loads data only on access. In this example, all releases of a sofware 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: 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.

2.5 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 ≥ 12.04, SLES 11, 12 and openSuSE 13.1, Fedora 25 and 26, or Mac OS X ≥ 10.12. There is experimental support for AArch64 and Power 8.

2.5.1 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. Ubuntu and Debian packages are available through our apt repository. The `cvmfs-release` packages can be used to add these yum/apt repositories to the local package sources list. 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
```

2.5.2 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 `/cvmfs /etc/auto.cvmfs` is set in /etc/auto.master and that the autofs service is running. Reload the autofs service in order to apply the updated configuration. 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.

If you're unsure about the proxy names, set `CVMFS_HTTP_PROXY=DIRECT`. This should *only* be done for a small number of clients (< 5), because large numbers can put a heavy load on the Stratum 1 servers and result, amongst others, in poorer performance for the client.

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
```

2.5.3 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.

2.5.4 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. Together with the problem description, please send the system information tarball created by `cvmfs_config bugreport`.

2.6 Client Configuration

2.6.1 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_FQRN` 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
`config.sh`	Set of internal helper functions.
`default.conf`	Set of base parameters.
`default.d/$config.conf`	Adjustments to the default.conf configuration, usually installed by a cvmfs-config-… package. Read before default.local.
`domain.d/$domain.conf`	Domain-specific parameters and implementations of the functions in `config.sh`
`config.d/$repository.conf`	Repository-specific parameters and implementations of the functions in `config.sh`
`keys/`	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, and also to centrally *blacklist* compromised repository keys. Configuration from the config repository is overwritten by the local configuration in case of conflicts; see the comments in /etc/cvmfs/default.conf for the precise ordering of processing the config files. The config repository is set by the `CVMFS_CONFIG_REPOSITORY` parameter. The default configuration rpm cvmfs-config-default 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.

2.6.2 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_RELOAD_SOCKETS=/home/user/mycache
CVMFS_USYSLOG=/home/user/cvmfs.log
CVMFS_CLAIM_OWNERSHIP=yes
CVMFS_SERVER_URL=http://cvmfs-stratum-one.cern.ch/cvmfs/atlas.cern.ch
CVMFS_KEYS_DIR=/etc/cvmfs/keys/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 >= 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 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.

2.6.3 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 failover 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[1]. 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.

[1] The usual proxy notation rules apply, like `http://proxy1:8080|http://proxy2:8080;DIRECT`

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/<FQRN>/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. 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 $[1, \$\text{CVMFS\_BACKOFF\_INIT} \cdot 1000]$. With every retry, the backoff is then doubled.

DNS Nameserver Changes

CernVM-FS can watch /etc/resolv.conf and automatically follow changes to the DNS servers. This behavior is controlled by the `CVMFS_DNS_ROAMING` client configuration. It is by default turned on on macOS and turned off on Linux.

2.6.4 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. 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.

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.

Advanced Cache Configuration

For exotic cache configurations, CernVM-FS supports specifying multiple, independent “cache manager instances” of different types. Such cache manager instances replace the local cache directory. Since the local cache directory is also used to store transient special files, CVMFS_WORKSPACE=$local_path must be used when advanced cache configuration is used.

A concrete cache manager instance has a user-defined name and it is specified like

```
CVMFS_CACHE_PRIMARY=myInstanceName
CVMFS_CACHE_myInstanceName_TYPE=posix
```

Multiple instances can thus be safely defined with different names but only one is selected when the client boots. The following table lists the valid cache manager instance types.

** Type**	**Behavior**
posix	Uses a cache directory with the standard cache implementation
tiered	Uses two other cache manager instances in a layered configuration
external	Uses an external cache plugin process (see Section *Client Plug-Ins*)

The instance name “default” is blocked because the regular cache configuration syntax is automatically mapped to CVMFS_CACHE_default_... parameters. The command sudo cvmfs_talk cache instance can be used to show the currently used cache manager instance.

Tiered Cache

The tiered cache manager combines two other cache manager instances as an upper layer and a lower layer into a single functional cache manager. Usually, a small and fast upper layer (SSD, memory) is combined with a larger and slower lower layer (HDD, network drive). The upper layer needs to be large enough to serve all currently open files. On an upper layer cache miss, CernVM-FS tries to copy the missing object from the lower into the upper layer. On a lower layer cache miss, CernVM-FS download and stores objects either in both layers or in the upper layer only, depending on the configuration.

The parameters `CVMFS_CACHE_$tieredInstanceName_UPPER` and `CVMFS_CACHE_$tieredInstanceName_LOWER` set the names of the upper and the lower instances. The parameter `CVMFS_CACHE_$tieredInstanceName_LOWER_READONLY=[yes|no]` controls whether the lower layer can be populated by the client or not.

External Cache Plugin

A CernVM-FS cache manager instance can be provided by an external process. The cache manager process and the CernVM-FS client are connected through a socket, whose address is called “locator”. The locator can either address a UNIX domain socket on the local file system, or a TCP socket, as in the following examples

```
CVMFS_CACHE_instanceName_LOCATOR=unix=/var/lib/cvmfs/cache.socket
# or
CVMFS_CACHE_instanceName_LOCATOR=tcp=192.168.0.24:4242
```

If a UNIX domain socket is used, both the CernVM-FS client and the cache manager need to be able to access the socket file. Usually that means they have to run under the same user.

Instead of manually starting the cache manager, the CernVM-FS client can optionally automatically start and stop the cache manager process. This is called a “supervised cache manager”. The first booting CernVM-FS client starts the cache manager process, the last terminating client stops the cache manager process. In order to start the cache manager in supervised mode, use `CVMFS_CACHE_instanceName_CMDLINE=<executable and arguments>`, using a comma (`,`) instead of a space to separate the command line parameters.

Example

The following example configures a tiered cache with an external cache plugin as an upper layer and a read-only, network drive as a lower layer. The cache plugin uses memory to cache data and is part of the CernVM-FS client. This configuration could be used in a data center with diskless nodes and a preloaded cache on a network drive (see Chapter *CernVM-FS on Supercomputers*)

```
CVMFS_WORKSPACE=/var/lib/cvmfs
CVMFS_CACHE_PRIMARY=hpc

CVMFS_CACHE_hpc_TYPE=tiered
CVMFS_CACHE_hpc_UPPER=memory
CVMFS_CACHE_hpc_LOWER=preloaded
CVMFS_CACHE_hpc_LOWER_READONLY=yes

CVMFS_CACHE_memory_TYPE=external
CVMFS_CACHE_memory_CMDLINE=/usr/libexec/cvmfs/cache/cvmfs_cache_ram,/etc/cvmfs/cache-
↪mem.conf
CVMFS_CACHE_memory_LOCATOR=unix=/var/lib/cvmfs/cvmfs-cache.socket

CVMFS_CACHE_preloaded_TYPE=posix
```

(continues on next page)

(continued from previous page)

```
CVMFS_CACHE_preloaded_ALIEN=/gpfs/cvmfs/alien
CVMFS_CACHE_preloaded_SHARED=no
CVMFS_CACHE_preloaded_QUOTA_LIMIT=-1
```

The example configuration for the in-memory cache plugin in /etc/cvmfs/cache-mem.conf is

```
CVMFS_CACHE_PLUGIN_LOCATOR=unix=/var/lib/cvmfs/cvmfs-cache.socket
# 2G RAM
CVMFS_CACHE_PLUGIN_SIZE=2000
```

2.6.5 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 >= 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-availablity 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,wsize=1048576 0 0
```

2.6.6 File Ownership

By default, cvmfs presents all files and directories as belonging to the mounting user, which for system mounts under /cvmfs is the user `cvmfs`. Alternatively, CernVM-FS can present the uid and gid of file owners as they have been at the time of publication by setting `CVMFS_CLAIM_OWNERSHIP=no`.

If the real uid and gid values are shown, stable uid and gid values across nodes are recommended; otherwise the owners shown on clients can be confusing. The client can also dynamically remap uid and gid values. To do so, the parameters `CVMFS_UID_MAP` and `CVMFS_GID_MAP` should provide the path to text files that specify the mapping. The format of the map files is identical to the map files used for *bulk changes of ownership on release manager machines*.

2.6.7 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.

2.6.8 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:

`-v`	Produce more verbose output.
`-j #threads`	Sets the number of concurrent threads that check files in the cache directory. Defaults to 4.
`-p`	Tries to automatically fix problems.
`-f`	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. With the *-s* option, only non-empty parameters are shown.

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 two 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.

2.6.9 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`.

2.7 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.

2.8 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. *This figure below* outlines the process of publishing a repository.

2.8.1 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 or RHEL7.3)
- Officially supported platforms
  - Scientific Linux 6 (64 bit - with custom AUFS enabled kernel - Appendix "*Available Packages*")
  - CentOS/SL >= 7.3, provided that /var/spool/cvmfs is served by an ext4 file system.
  - Fedora 25 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

2.8.2 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 Packages*") 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.

2.8.3 Publishing a new Repository Revision

Fig. 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.

2.8.4 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 25+ 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).

CernVM-FS 2.2.0 supports both OverlayFS and aufs as a union file system. At least a 4.2.x kernel is needed to use CernVM-FS with OverlayFS. (Red Hat) Enterprise Linux >= 7.3 works, too, provided that /var/spool/cvmfs is served by an ext3 or 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.

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

2.8.6 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 by default a new master key and corresponding public key (see more about that in the next section).

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.

Master keys

Each cvmfs repository uses two sets of keys, one for the individual repository and another called the "masterkey" which signs the repository key. The pub key that corresponds to the masterkey is what needs to be distributed to clients to verify the authenticity of the repository. It is usually most convenient to share the masterkey between all repositories in a domain so new repositories can be added without updating the client configurations. If the clients are maintained by multiple organizations it can be very difficult to quickly update the distributed pub key, so in that case it is important to keep the masterkey especially safe from being stolen. If only repository keys are stolen, they can be replaced without having to update client configurations.

By default, `cvmfs_server mkfs my.repo.name` creates a new `/etc/cvmfs/keys/my.repo.name.masterkey` and corresponding `/etc/cvmfs/keys/my.repo.name.pub` for every new repository. Additional user-written procedures can then be applied to replace those files with a common masterkey/pub pair, and then `cvmfs_server resign` must be run to update the corresponding signature (in `/srv/cvmfs/my.repo.name/.cvmfswhitelist`). Signatures are only good for 30 days by default, so `cvmfs_server resign` must be run again before they expire.

`cvmfs_server` also supports the ability to store the masterkey in a separate inexpensive smartcard, so that even if the computer hosting the repositories is compromised, the masterkey cannot be stolen. Smartcards allow writing keys into them and signing files but they never allow reading the keys back. Currently the supported hardware are the Yubikey 4 or Nano USB devices.

If one of those devices is plugged in to a release manager machine, this is how to use it:

1. Create a repository with `cvmfs_server mkfs my.repo.name`
2. Store its masterkey and pub into the smartcard with `cvmfs_server masterkeycard -s my.repo.name`
3. **Make a backup copy of `/etc/cvmfs/keys/my.repo.name.masterkey` on** at least one USB flash drive because the next step will irretrievably delete the file. Keep the flash drive offline in a safe place in case something happens to the smartcard.
4. Convert the repository to use the smartcard with `cvmfs_server masterkeycard -c my.repo.name`. This will delete the masterkey file. This command can also be applied to other repositories on the same machine; their pub file will be updated with what is stored in the card and they will be resigned.

From then on, every newly created repository on the same machine will automatically use the shared masterkey stored on the smartcard.

When using a masterkeycard, the default signature expiration reduces from 30 days to 7 days. `cvmfs_server resign` needs to be run to renew the signature. It is recommended to run that daily from cron.

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.

Compression and Hash Algorithms

Files in the CernVM-FS repository data store are compressed and named according to their compressed content hash. The default settings use DEFLATE (zlib) for compression and SHA-1 for hashing.

CernVM-FS can optionally skip compression of files. This can be beneficial, for instance, if the repository is known to contain already compressed content, such as JPG images or compressed ROOT files. In order to disable compression, set `CVMFS_COMPRESSION_ALGORITHM=none` in the `/etc/cvmfs/repositories.d/$repository/server.conf` file. Client version >= 2.2 is required in order to read uncompressed files.

Instead of SHA-1, CernVM-FS can use RIPEMD-160 or SHAKE-128 (a variant of SHA-3 with 160 output bits) as hash algorithm. In general, we advise not to change the default. In future versions, the default might change from SHA-1 to SHAKE-128. In order to enforce the use of a specific hash algorithm, set `CVMFS_HASH_ALGORITHM=sha1`, `CVMFS_HASH_ALGORITHM=rmd160`, or `CVMFS_HASH_ALGORITHM=shake128` in the `server.conf` file. Client version >= 2.1.18 is required for accessing repositories that use RIPEMD-160. Client version >= 2.2 is required for accessing repositories that use SHAKE-128.

Both compression and hash algorithm can be changed at any point during the repository life time. Existing content will remain untouched, new content will be processed with the new settings.

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 data directly to S3 compatible storage systems, such as Amazon S3, or Ceph. The S3 target bucket needs to be created beforehand, for example with `s3cmd`. The bucket needs to be public for reading and require authorization for writing:

```
# The --configure is optional. For the CERN Ceph S3 instance, for example, use host s3.cern.ch and the %(bucket).s3.cern.ch URL template.
s3cmd --configure
export AWS_ACCESS_KEY_ID=<ACCESS KEY>
export AWS_SECRET_ACCESS_KEY=<SECRET KEY>
s3cmd mb s3://<BUCKET NAME>
s3cmd --acl-public setacl s3://<BUCKET NAME>
```

Note: if you use the Minio client, the `download` bucket policy won't work as a bucket policy.

Once the bucket is available, 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://mybucket.s3.amazonaws.com my.repo.name
```

The file "mys3.conf" contains the S3 settings (see :ref: *table below <tab_s3confparameters>*). The "-w" option is used define the S3 server URL, e.g. http://localhost:3128, which is used for accessing the repository's backend storage on S3.

Parameter	Meaning
`CVMFS_S3_ACCESS_KEY`	S3 account access key
`CVMFS_S3_SECRET_KEY`	S3 account secret key
`CVMFS_S3_HOST`	S3 server hostname, e.g. s3.amazonaws.com. The hostname should NOT be prefixed by "http://"
`CVMFS_S3_REGION`	The S3 region, e.g. eu-central-1. If specified, AWSv4 authorization protocol is used.
`CVMFS_S3_PORT`	The port on which the S3 instance is running
`CVMFS_S3_BUCKET`	S3 bucket name. The repository name is used as a subdirectory inside the bucket.
`CVMFS_S3_TIMEOUT`	Timeout in seconds for the connection to the S3 server.
`CVMFS_S3_MAX_RETRIES`	Number of retries for the connection to the S3 server.
`CVMFS_S3_MAX_NUMBER_OF_PARALLEL_CONNECTIONS`	Number of parallel uploads to the S3 server, e.g. 400
`CVMFS_S3_DNS_BUCKETS`	Set to false to disable DNS-style bucket URLs (http://<bucket>.<host>/<object>). Enabled by default.
`CVMFS_S3_PEEK_BEFORE_PUT`	Make PUT requests conditional to a prior HEAD request. Enabled by default.

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.

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.

Repository Update Propagation

Updates to repositories won't immediately appear on the clients. For scalability reasons, clients will only regularly check for updates. The frequency of update checks is stored in the repository itself and defaults to 4 minutes. The default can be changed by setting CVMFS_REPOSITORY_TTL in the /etc/cvmfs/repositories.d/$repository/server.conf file to a new value given in seconds. The value should not fall below 1 minute.

If the repository is replicated to a stratum 1 server (see Chapter *Setting up a Replica Server (Stratum 1)*), replication of the changes needs to finish before the repository time-to-live applies. The status of the replication can be checked by the cvmfs_info utility, like

```
cvmfs_info http://cvmfs-stratum-zero.cern.ch/cvmfs/cernvm-prod.cern.ch
```

The cvmfs_info utility can be downloaded as a stand-alone Perl script from the linked github repository.

The cvmfs_info utility relies on the repository meta-data as described in Chapter *CernVM-FS Server Meta Information*. It shows timestamp and revision number of the repository on the stratum 0 master server and all replicas, as well as the remaining life time of the repository whitelist and the catalog time-to-live.

Note: The cvmfs_info utility queries stratum servers without passing through web proxies. It is not meant to be used on a large-scale by all clients. On clients, the extended attribute revision can be used to check for the currently active repository state, like

```
attr -g revision /cvmfs/cernvm-prod.cern.ch
```

Grafting Files

When a repository is updated, new files are checksummed and copied / uploaded to a directory exported to the web. There are situations where this is not optimal - particularly, when *"large-scale" repositories* are used, it may not be pragmatic to copy every file to a single host. In these cases, it is possible to "graft" files by creating a special file containing the necessary publication data. When a graft is encountered, the file is published as if it was present on the repository machine: the repository admin is responsible for making sure the file's data is distributed accordingly.

To graft a file, foo to a directory, one must: - Create an empty, zero-length file named foo in the directory. - Create a separate graft-file named .cvmfsgraft-foo in the same directory.

The .cvmfsgraft file must have the following format:

```
size=$SIZE
checksum=$CHECKSUM
chunk_offsets=$OFFSET_1,$OFFSET_2,$OFFSET_3,...
chunk_checksums=$CHECKSUM_1,$CHECKSUM_2,$CHECKSUM_3,...
```

Here, $SIZE is the entire file size and $CHECKSUM is the file's checksum; the checksums used by this file are assumed to correspond to the algorithm selected at publication time. The offsets $OFFSET_X and checksums $CHECKSUM_X correspond to the checksum and beginning offset of each chunk in the file. $OFFSET_1 is always at 0. Implicitly, the last chunk ends at the end of the file.

To help generate checksum files, the cvmfs_swissknife graft command is provided. The graft command takes the following options:

Option	Description
`-i`	Input file to process (`-` for reading from stdin)
`-o`	Output location for graft file (optional)
`-v`	Verbose output (optional)
`-Z`	Compression algorithm (default: none) (optional)
`-c`	Chunk size (in MB; default: 32) (optional)
`-a`	hash algorithm (default: `SHA-1`) (optional)

This command outputs both the `.cvmfsgraft` file and and zero-length "real" file if `-o` is used; otherwise, it prints the contents of the `.cvmfsgraft` file to `stdout`. A typical invocation would look like this:

```
cat /path/to/some/file | cvmfs_swissknife graft -i - -o /cvmfs/repo.example.com/my_
↪file
```

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.

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* or in a masterkeycard. 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.

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.

2.8.7 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 spacial 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 restriced 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.

Named Snapshot Diffs

The command `cvmfs_server diff` shows the difference in terms of added, deleted, and modified files and directories between any two named snapshots. It also shows the difference in total number of files and nested catalogs.

Unless named snapshots are provided by the `-s` and `-d` flags, the command shows the difference from the last snapshot ("trunk-previous") to the current one ("trunk").

Instant Access to Named Snapshots

CernVM-FS can maintain a special directory

```
/cvmfs/${repository_name}/.cvmfs/snapshots
```

through which the contents of all named snapshots is accessible by clients. The directory is enabled and disabled by setting `CVMFS_VIRTUAL_DIR=[true,false]`. If enabled, for every named snapshot `$tag_name` a directory `/cvmfs/${repository_name}/.cvmfs/snapshots/${tag_name}` is maintained, which contains the contents of the repository in the state referenced by the snapshot.

To prevent accidental recursion, the top-level directory `.cvmfs` is hidden by CernVM-FS clients >= 2.4 even for operations that show dot-files like `ls -a`. Clients before version 2.4 will show the `.cvmfs` directory but they cannot recurse into the named snapshot directories.

Branching

In certain cases, one might need to publish a named snapshot based not on the latest revision but based on a previous named snapshot. This can be useful, for instance, if versioned data sets are stored in CernVM-FS and certain files in a past data set needs to be fixed.

In order to publish a branch, use `cvmfs_server checkout` in order to switch to the desired parent branch before starting a transaction. The following example publishes based on the existing snapshot "data-v201708" the new named snapshot "data-v201708-fix01" in the branch "fixes_data-v201708".

```
cvmfs_server checkout -b fixes_data-v201708 -t data-v201708
cvmfs_server transaction
# show that the repository is in a checked-out state
cvmfs_server list
```

(continues on next page)

(continued from previous page)

```
# make changes to /cvmfs/${repository_name}
cvmfs_server publish -a data-v201708-fix01
# show all named snapshots and their branches
cvmfs_server tag -l
# verify that the repository is back on the trunk revision
cvmfs_server list
```

When publishing a checked out state, it is mandatory to specify a tag name. Later, it might be necessary to publish another set of fixes in the same branch. To do so, the command `cvmfs_server checkout -b fixes_data-v201708` checks out the latest named snapshot from the given branch. The command `cvmfs_server checkout` jumps back to the trunk of the repository.

The comomand `cvmfs_server tag -b` displays the tree of branches and their respective initial revisions. The `-x` switch triggers displaying of the tree in a machines-readable format.

Branching makes most sense for repositories that use the instant snapshot access (see Section *Branching*).

Please note that while CernVM-FS supports branching, it does not support merging of repository snapshots.

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
```

(continues on next page)

(continued from previous page)

```
  |     `     |- 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 ≈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. Empty lines and comment lines starting with a pound sign (`#`) are ignored. For example a `.cvmfsdirtab` might contain these lines for the repository of the previous subsection:

```
# Nested catalogs for every platform
/software/*
# Nested catalogs for every version
/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
```

(continues on next page)

(continued from previous page)

```
/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 ≈200000. One can use the switch `list-catalogs -e` to inspect the current nested catalog entry counts in the repository. Furthermore `list-catalgos -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 anytime 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.

2.8.8 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 and will run every once in a while after a publish operation. Alternatively, `CVMFS_AUTO_GC=false` may be set and `cvmfs_server gc` run from cron at a time when no publish operations will be happening; garbage collection and publish operations cannot happen at the same time.

Enabling Garbage Collection on an Existing Replication (Stratum 1)

In order to use automatic garbage collection on a stratum 1 replica, set `CVMFS_AUTO_GC=true` in the `server.conf` file of the stratum 1 installation. This will run the garbage collection every once in a while after a snapshot. It will only work if the upstream stratum 0 repository has garbage collection enabled.

Alternatively, `cvmfs_server gc -af` can be run from cron periodically (e.g. daily) to run garbage collection on all repositories that have garbage collection enabled on the stratum 0. Logs will go into /var/log/cvmfs/gc.log.

Frequency of the Automatic Garbage Collection

If `CVMFS_AUTO_GC=true` is set, the parameter `CVMFS_AUTO_GC_LAPSE` controls how frequently automatic garbage collection is executed. By default, `CVMFS_AUTO_GC_LAPSE` is set to `1 day ago`. If, on publish or snapshot, the last manual or automatic garbage collection is farther in the past then the given threshold, garbage collection will run. Otherwise it is skipped.

2.8.9 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.

If you need to publish files with much larger working set sizes than a typical software environment, refer to *"large-scale" repositories* document.

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").

2.8.10 Configuration Recommendation by Use Case

The default configuration of a fresh CernVM-FS repository are tuned for production software repositories and maximum compatibility and safety. For other typical use cases, the configuration should be adapted.

General Recommendations

NOTE: Do _not_ use `CVMFS_GENERATE_LEGACY_BULK_CHUNKS=false` together with a hash algorithm other than SHA-1 in CernVM-FS <= 2.4.3.

Unless an older client base needs to be supported, we recommend to the following configuration changes:

```
CVMFS_AUTO_TAG_TIMESPAN="2 weeks ago"
CVMFS_IGNORE_XDIR_HARDLINKS=true
CVMFS_GENERATE_LEGACY_BULK_CHUNKS=false
CVMFS_HASH_ALGORITHM=shake128
```

These changes make unreferenced objects older than two weeks subject to garbage collection (without enabling garbage collection). It tolerates that hard links among different directories (see *Hardlinks*), creates only chunked versions of large files and uses the more future-proof SHA-3 derived content hash algorithm.

Multi-Tenant Repositories

For repositories that are edited by several, possibly unexperienced users, we suggest the following configuration settings:

```
CVMFS_AUTOCATALOGS=true
CVMFS_ENFORCE_LIMITS=true
CVMFS_FORCE_REMOUNT_WARNING=false
```

This will, in addition to manually created nested catalogs, keep the maximum file catalog size small and enforce the limit on maximum file sizes. It will also prevent forced remounts from sending a broadcast message to all users.

Repositories for Software “Nightly Builds”

Repositories containing the result of “nightly builds” are usually subject to a lot of churn and accumulate unreferenced objects quickly. We recommend to set

```
CVMFS_AUTO_TAG=false
CVMFS_GARBAGE_COLLECTION=true
CVMFS_AUTO_GC=true
```

in order to activate garbage collection and to turn off CernVM-FS’ versioning (provided that the content on such repositories is ephemeral). Instead of autmatic garbage collection, one can also install a regular cron job running `cvmfs_server gc -af`, or the nightly build script should be updated to invoke `cvmfs_server gc <repo name>`.

Repositories for (Conditions) Data

Repositories containing data sets (cf. *Large-Scale Data CernVM-FS*) should start with the following base configuration

```
CVMFS_COMPRESSION_ALGORITHM=none
CVMFS_FILE_MBYTE_LIMIT= >> larger than expected maximum file size
CVMFS_VIRTUAL_DIR=true
```

provided that data files are already compressed and that access to previous file system revisions on client-side is desired.

Repositories for Container Images

Repositories containing Linux container image contents (that is: container root file systems) should have the following configuration:

```
CVMFS_IGNORE_SPECIAL_FILES=true
CVMFS_INCLUDE_XATTRS=true
CVMFS_VIRTUAL_DIR=true
```

This ensures that left-over device files in the root file system do not break publication on CernVM-FS but instead they get silently dropped. Extended attributes of files, such as file capabilities and SElinux attributes, are recorded. And previous file system revisions can be accessed from the clients.

2.9 CernVM-FS Server Meta Information

The CernVM-FS server automatically maintains both global and repository specifc 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 signed meta-information that gets replicated to Stratum1 servers automatically.

2.9.1 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`)s:

```
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.

2.9.2 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` utilitiy. 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*.

2.9.3 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",
  "url"           : "https://www.example.com/",
  "recommended-stratum0"  : "stratum 0 url",
  "recommended-stratum1s" : [ "stratum1 url", "stratum1 url" ],

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

2.10 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.

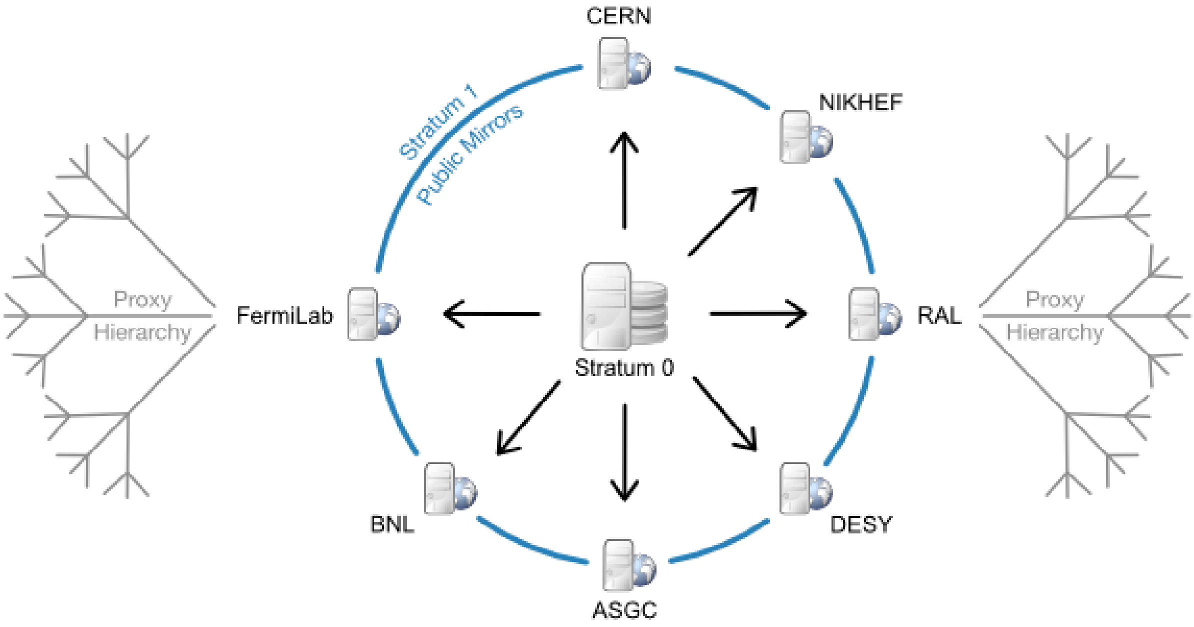

Fig. 4: CernVM-FS content distribution network for the cern.ch domain: Stratum1 replica servers are located in Europe, the U.S. and Asia. One protected read/write instance (Stratum 0) is feeding up the public, distributed mirror servers. A distributed hierarchy of proxy servers fetches content from the closest public mirror server.

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.

2.10.1 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:

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. In that case caching can be disabled for the data (since there's no need to store it again on the same disk), but caching is helpful for the geo api calls. Alternatively, separate Squid server machines may be configured in a round-robin DNS and each forward 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.

Note: Port 8000 might be assigned to `soundd`. On SElinux systems, this assignment must be changed to the HTTP service by `semanage port -m -t http_port_t -p tcp 8000`. The `cvmfs-server` RPM for EL7 executes this command as a post-installation script.

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 `nscd` since it does not honor the DNS Time-To-Live protocol.

2.10.2 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.

2.10.3 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.

2.11 CernVM-FS Repository Gateway and Release Managers

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

2.11.1 Glossary

Gateway (GW) The machine running an instance of the CVMFS repository gateway which has access to the authoritative storage of the managed repositories. This storage is made accessible either as a locally mounted partition or through an S3 API. 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 and send 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 changes to different repositories where it currently holds valid leases.

The computationally heavy parts of the publication operation take place on the RM: compressing and hashing the files which are to be added or modified. The processed files are then packed together and sent to the GW to be inserted into the repository and made available to clients.

2.11.2 Repository Gateway Configuration

As a prerequisite, we need to install the CVMFS client and server packages on the gateway. This means that the gateway machine can be used as a “master” release manager to perform some repository transformations before a separate release manager machine is set up.

The repository gateway application is currently packaged for Ubuntu 16.04, Ubuntu 18.04, SLC 6 and Cern CentOS 7. On Ubuntu, an additional step is required after the package is installed, the setup script needs to be run as root:

```
# /usr/libexec/cvmfs-gateway/scripts/setup.sh
```

Create the repository for the following section of this guide:

```
# cvmfs_server mkfs -o root test.cern.ch
```

Create an API key file for the new repo (replace `<KEY_ID>` and `<SECRET>` with actual values):

```
# cat <<EOF > /etc/cvmfs/keys/test.cern.ch.gw
plain_text <KEY_ID> <SECRET>
EOF
# chmod 600 /etc/cvmfs/keys/test.cern.ch.gw
```

Add the API key file to the repository configuration in the gateway application:

```
# cat <<EOF > /etc/cvmfs/gateway/repo.json
{
  "repos": [
    {
      "domain": "test.cern.ch",
      "keys": ["<KEY_ID>"]
    }
  ],
  "keys": [
    {
      "type": "file",
      "file_name": "/etc/cvmfs/keys/test.cern.ch.gw",
      "repo_subpath": "/"
    }
  ]
}
EOF
```

If Systemd is available, the gateway application can be started with `systemctl`:

```
# systemctl start cvmfs-gateway.service
```

otherwise it can be manually started:

```
# /usr/libexec/cvmfs-gateway/scripts/run_cvmfs_gateway.sh start
```

The ports 80/TCP and 4929/TCP need to be opened in the firewall, to allow access to the repository contents and to the gateway service API.

Alongside the `repo.json` file, there is another configuration file for the repository gateway - `user.json`. The most important options in this file are:

- `max_lease_time` - the maximum duration, in seconds, of an acquired lease
- `fe_tcp_port` - the port on which the gateway application listens, 4929 by default

By default, the gateway application only spawns a single cvmfs_receiver worker process. It is possible to run multiple worker processes by increasing the value of the size entry in the receiver_config map, found in user.json. This value should not be increased beyond the number of available CPU cores.

2.11.3 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.

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 test.cern.ch.gw.
- The GW gateway application is running on port 4929 at the URL http:://gateway.cern.ch:4929/api/v1.
- The repository keys have been copied from the gateway machine onto the release manager machine, in /tmp/test.cern.ch_keys.

To create the repository in the release manager configuration, run the following command on rm.cern.ch as an unprivileged user with sudo access:

```
$ sudo cvmfs_server mkfs -w http://gateway.cern.ch/cvmfs/test.cern.ch \
                         -u gw,/srv/cvmfs/test.cern.ch/data/txn,http://gateway.cern.
↪ch:4929/api/v1 \
                         -k /tmp/test.cern.ch_keys -o `whoami` test.cern.ch
```

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

```
$ cvmfs_server transaction test.cern.ch
```

. . . make changes to the repository . . .

```
$ cvmfs_server publish test.cern.ch
```

2.11.4 Updating from cvmfs-gateway-0.2.5

In the first published version, cvmfs-gateway-0.2.5, the application files were installed under /opt/cvmfs-gateway and the database files under /opt/cvmfs-mnesia. Starting with version 0.2.6, the application is installed under /usr/libexec/cvmfs-gateway, while the database files are under /var/lib/cvmfs-gateway.

When updating from 0.2.5, please make sure that the application is stopped:

```
# systemctl stop cvmfs-gateway
```

and rerun the setup script:

```
# /usr/libexec/cvmfs-gateway/scripts/setup.sh
```

At this point, the new version of the application can be started. If the old directories are still present, they can be deleted:

```
# rm -r /opt/cvmfs-{gateway,mnesia}
```

2.12 Advanced Topics

2.12.1 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 there are two plug-in interfaces: cache manager plugins and authorization helpers.

Cache Plugins

A cache plugin provides the functionality of the client's local cache directory: it maintains a set of content-addressed objects. Clients can read from these objects. Depending on its capabilities, a cache plugin might also support addition of new objects, listing objects and eviction of objects from the cache.

Cache plugins and clients exchange messages through a socket. The messages are serialized by the Google protobuf library. A description of the wire protocol can be found in the `cvmfs/cache.proto` source file, although the cache plugins should not directly implement the protocol. Instead, plugins are supposed to use the `libcvmfs_cache` library (part of the CernVM-FS development package), which takes care of the low-level protocol handling.

Good entry points into the development of a cache plugin are the demo plugin `cvmfs/cache_plugin/cvmfs_cache_null.cc` and the production in-memory cache plugin `cvmfs/cache_plugin/cvmfs_cache_ram.cc`. The CernVM-FS unit test suite has a unit test driver, `cvmfs_test_cache`, with a number of tests that are helpful for the development and debugging of a cache plugin.

Broadly speaking, a cache plugin process performs the following steps

```
#include <libcvmfs_cache.h>

cvmcache_init_global();
// Option parsing, which can use cvmcache_options_... functions to parse
// CernVM-FS client configuration files

// Optionally: spawning the watchdog to create stack traces when the cache
// plugin crashes
cvmcache_spawn_watchdog(NULL);

// Create a plugin context by passing function pointers to callbacks
struct cvmcache_context *ctx = cvmcache_init(&callbacks);

// Connect to the socket defined by the locator string
cvmcache_listen(ctx, locator);

// Spawn an I/O thread in which the callback functions are called
cvmcache_process_requests(ctx, 0);

// Depending on whether the plugin is started independently or by the
```

(continues on next page)

(continued from previous page)

```
// CernVM-FS client, cvmcache_process_requests() termination behaves
// differently

if (!cvmcache_is_supervised()) {
  // Decide when the plugin should be terminated, e.g. wait for a signal
  cvmcache_terminate(ctx);
}

// Cleanup
cvmcache_wait_for(ctx);
cvmcache_terminate_watchdog();
cvmcache_cleanup_global();
```

The core of the cache plugin is the implementation of the callback functions provided to `cvmcache_init()`. Not all callback functions need to be implemented. Some can be set to `NULL`, which needs to correspond to the indicated plugin capabilities specified in the `capabilities` bit vector.

Basic Capabilities

Objects maintained by the cache plugin are identified by their content hash. Every cache plugin must be able to check whether a certain object is available or not and, if it is available, provide data from the object. This functionality is provided by the `cvmcache_chrefcnt()`, `cvmcache_obj_info()`, and `cvmcache_pread()` callbacks. With only this functionality, the cache plugin can be used as a read-only lower layer in a tiered cache but not as a stand-alone cache manager.

For a proper stand-alone cache manager, the plugin must keep reference counting for its objects. The concept of reference counting is borrowed from link counts in UNIX file systems. Every object in a cache plugin has a reference counter that indicates how many times the object is being in use by CernVM-FS clients. For objects in use, clients expect that reading succeeds, i.e. objects in use must not be deleted.

Adding Objects

On a cache miss, clients need to populate the cache with the missing object. To do so, cache plugins provide a transactional write interface. The upload of an object results in the following call chain:

1. A call to `cvmcache_start_txn()` with a given transaction id
2. Zero, one, or multiple calls to `cvmcache_write_txn()` that append data
3. A call to `cvmcache_commit_txn()` ro `cvmcache_abort_txn()`

Only after commit the object must be accessible for reading. Multiple concurrent transactions on the same object are possible. After commit, the reference counter of the object needs to be equal to the number of transactions that committed the object (usually 1).

Listing and Cache Space Management

Listing of the objects in the cache and the ability to evict objects from the cache are optional capabilities. Only objects whose reference counter is zero may be evicted. Clients can keep file catalogs open for a long time, thereby preventing them from being evicted. To mitigate that fact, cache plugins can at any time send a notification to clients using `cvmcache_ask_detach()`, asking them to close as many nested catalogs as they can.

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 conatain 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.

2.12.2 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 transfered 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 invokation 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 invokations of the same command only transfer a change set and typically finish within seconds or minutes. These subsequent invokations 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).

Tiered Cache and Cache Plugins

Diskless compute nodes can also combine an in-memory cache with a preloaded directory on the shared cluster file system. An example configuration can be found in Section *Example*.

2.12.3 CernVM-FS Graph Driver Plugin for Docker

The CernVM-FS graph driver plugin for Docker provides a dockerized CernVM-FS client that can be used by the Docker daemon to access and store container images that reside in an extracted form on a CernVM-FS repository. Because CernVM-FS downloads the files of a container image only when accessed and because typically very little of a container image is accessed at runtime, the CernVM-FS graph driver can remove the bottleneck of distributing (large) container images to (many) nodes.

The CernVM-FS graph driver can run any normal image from a Docker registry. Additionally, it can run so called *Thin Images*. A thin image is like a symbolic link for container images. It is a regular, very small image in the registry. It contains a single file, the *thin image descriptor*, that specifies where in a CernVM-FS repository the actual image contents can be found. The `docker2cvmfs` utility can be used to convert a regular image to a thin image.

Requirements

The graph driver plugin requires Docker version > 17 and a host kernel with either aufs or overlay2 support, which includes RHEL >= 7.3. Please note that on RHEL 7, Docker's data root should reside either on an ext file system or on an xfs file system that is formatted with the `ftype=1` mount option.

The Docker graph driver plugin receives its CernVM-FS configuration by default from the Docker host's /etc/cvmfs directory. The easiest way to populate /etc/cvmfs is to install the `cvmfs-config-default` package (or any other `cvmfs-config-...` package) on the Docker host. Alternatively, a directory structure resembling the /etc/cvmfs hierarchy can by manually created and linked to the graph driver plugin.

Installation

The folling steps install and activate the CernVM-FS graph driver plugin.

1. Install the plugin with `docker plugin install cvmfs/graphdriver`. The command `docker plugin ls` should now show the new plugin as being activated.
2. Create or edit the file `/etc/docker/daemon.json` so that it contains the following content

   ```
   {
     "experimental": true,
     "storage-driver": "cvmfs/graphdriver",

     // To change the docker data root to an ext formatted location (remove this line)
     "data-root": "/path/to/ext/mountpoint",

     // Add the following storage option on RHEL 7 (remove this line)
     "storage-opts": [
       "overlay2.override_kernel_check=true"
     ]
   }
   ```

3. Restart the Docker daemon with `systemctl restart docker`.
4. Test the new plugin with a normal image

   ```
   docker run -it --rm ubuntu /bin/bash
   ```

   and with a thin image

   ```
   docker run -it --rm cvmfs/thin_ubuntu /bin/bash
   ```

In order to get debugging output, add `"debug": true` to the /etc/docker/daemon.json file.

Location of the Plugin Configuration

By default, the plugin tries to bind mount the host's /etc/cvmfs directory as a source of configuration. Other locations can be linked to the container by running

```
docker plugin set cvmfs/graphdriver cvmfs_ext_config="/alternative/location"
docker plugin set cvmfs/graphdriver minio_ext_config="/alternative/location"
```

Installation from a Plugin Tarball

Instead of installing the plugin from the Docker registry, it can be installed directly from a tarball. To do so, download and untar a graph driver plugin tarball. Run

```
docker plugin create my-graphdriver cvmfs-graphdriver-plugin-$VERSION
docker plugin enable my-graphdriver
```

Note: currently, the graph driver name (`my-graphdriver`) must not contain a colon (`:`) nor a comma (`,`). This issue will be fixed in a lalter version.

Conversion of Images

Note: The usage of the `cvmfs2docker` utility is preliminary. A more convenient transformation process is under development.

Download the latest version of the docker2cvmfs utility from https://ecsft.cern.ch/dist/cvmfs/docker2cvmfs/ and make it executable with `chmod +x docker2cvmfs`.

On a cvmfs release manager machines, download the original images layers as tarballs using `docker2`, like

```
./docker2cvmfs --registry https://gitlab-registry.cern.ch/v2 pull cloud/image-
↪name:latest /home/user/layers/
```

If you download from DockerHub, you can omit the `--registry` flag.

To extract the image layer tarballs into a CernVM-FS repository, run a bash script like the following one

```
DESTINATION=/cvmfs/images.cern.ch/layers
for l in /home/user/layers/*; do
  hash=$(basename $l .tar.gz)
  dst_layer=/cvmfs/test.cern.ch/layers/$hash
  mkdir -p $dst_layer
  touch $dst_layer/.cvmfscatalog
  tar xf $l -C $dst_layer --owner=$(id -u) --group=$(id -u) --no-xattrs --exclude=
↪"*dev/*";
done
```

Note that the CernVM-FS repository should have the settings `CVMFS_IGNORE_SPECIAL_FILES=true`, `CVMFS_INCLUDE_XATTRS=true`, and `CVMFS_IGNORE_XDIR_HARDLINKS=true`. If the repository is owned by the root user on the release manager machine, the extra options to the tar command can be omitted.

As a last step, the thin image needs to be pushed to a docker registry. To do so, run the following commands

```
./docker2cvmfs --registry https://gitlab-registry.cern.ch/v2 thin cloud/image-
↪name:latest images.cern.ch/layers > thin.json
tar cf - thin.json | docker import - cvmfs/thin_image-name
docker push cvmfs/thin_image-name
```

2.12.4 Setting up an Xcache reverse proxy

This page describes how to set up an experimental HTTP reverse proxy layer for CernVM-FS based on Xcache.

NOTE: This is not a replacement for a general site forward proxy. Forwarding needs to be defined separately in the Xcache configuration for each destination Stratum 1, and the client CVMFS_SERVER_URL configuration has to point to a separate forwarder URL for each server. This document is for the convenience of people who want to experiment with this configuration.

Requirements

- A machine (labeled **Machine A**) to serve the contents of the CernVM-FS repository. Should have CernVM-FS server tools installed, as well as XRootD.
- A second machine (labeled **Machine B**) to use as a reverse proxy. Only XRootD is needed on this machine.
- A CernVM-FS client to mount the repository, for testing.

Instructions

XRootD is a high-performance, scalable file distribution solution. It has a plugin-based architecture and can be configured to suit various use cases. In the Xcache configuration, an XRootD daemon functions as a reverse proxy, serving the contents of a data repository over HTTP.

The following diagram shows how Xcache can be deployed as a cache layer between a CernVM-FS repository and client machines:

Machine A contains a CernVM-FS repository, served by default over HTTP. An Xcache instance is running on a second machine. By default Xcache can only ingest files from another XRootD instance - we start an instance of XRootD on the same machine as the CernVM-FS repository, configured to export the repository using the XRootD protocol. The following configuration can be used for this instance of XRootD, replacing `<CVMFS_REPOSITORY_NAME>` with the actually name of the repository:

```
oss.localroot /srv
all.export /cvmfs/<CVMFS_REPOSITORY_NAME> r/o

all.adminpath /var/spool/xrootd
all.pidpath   /var/run/xrootd

xrd.trace all
```

The Xcache instance running on the second machine can be pointed to the XRootD daemon started on the first one (`<CVMFS_REPOSITORY_NAME>` should be replaced with the actual repository name and `MACHINE_A_HOSTNAME` with the actual host name of the first machine):

```
all.adminpath /var/spool/xrootd
all.pidpath   /var/run/xrootd

oss.localroot  /data/namespace

all.export /cvmfs/<CVMFS_REPOSITORY_NAME>

oss.space meta /data/xrdcinfos
oss.space data /data/datafiles

xrd.protocol http:3000 /usr/lib64/libXrdHttp.so
xrd.trace all

ofs.osslib   /usr/lib64/libXrdPss.so
pss.cachelib /usr/lib64/libXrdFileCache.so
pss.config streams 32
pss.origin = <MACHINE_A_HOSTNAME>:1094

pfc.ram 4g
pfc.diskusage 0.5 0.6
pfc.spaces data meta
pfc.blocksize 1M
pfc.prefetch 0
pfc.trace info
```

With this configuration, Xcache re-exports the contents of the repository over HTTP, on port 3000. Interested CernVM-FS clients can be configured to use the Xcache instance by modifying the `CVMFS_SERVER_URL` variable:

```
CVMFS_SERVER_URL=http://<MACHINE_B_HOSTNAME>:3000/cvmfs/<CVMFS_REPOSITORY_NAME>
```

Cache invalidation

A current limitation of Xcache is that cached files are never invalidated. In the context of CernVM-FS, this means that newly published root catalogs are not picked up automatically. An Xcache plugin is being developed to address this limitation.

Ingestion over HTTP

A new XRootD client plugin is being developed to allow the Xcache instance to ingest files over HTTP:

This set up is non-intrusive, as the machine serving the CernVM-FS repository no longer needs to be modified in any way. Xcache could thus be deployed as a reverse proxy layer for existing CernVM-FS stratum servers.

2.12.5 Large-Scale Data CernVM-FS

CernVM-FS primarily is developed for distributing large software stacks. However, by combining several extensions to the base software, one can use CVMFS to distribute large, non-public datasets. While there are several ways to deploy a the service, in this section we outline one potential path to achieve secure distribution of terabytes-to-petabytes of data.

To deploy large-scale CVMFS, a few design decisions are needed:

- **How is data distributed?** For the majority of repositories, data is replicated from a repository server to an existing content distribution network tuned for the object size common to software repositories. The CDNs currently in use are tuned for working set size on the order of tens of gigabytes; they are not appropriately sized for terabytes of data. You will need to put together a mechanism for delivering data at the rates your clients will need.
  - For example, `ligo.osgstorage.org` has about 20TB of data; each scientific workflow utilizes about 2TB of data and each running core averages 1Mbps of input data. So, to support the expected workflows at 10,000 running cores, several 10TB caches were deployed that could export a total of 40Gbps.
  - The `cms.osgstorage.org` repository publishes 3PB of data. Each analysis will read around 20TB and several hundred analyses will run simultaneously. Given the large working set size, there is no caching layer and data is read directly from large repositories.
- **How is data published?** By default, CVMFS publication will calculate checksums on its contents, compresses the data, and serves it from the Apache web server. Implicitly, this means all data must be _copied_ to and _stored_ on the repository host; at larger scales, this is prohibitively expensive. The `cvmfs_swissknife graft` tool provides a mechanism to publish files directly if the checksum is known ahead of time; see *Grafting Files*.
  - For `ligo.osgstorage.org`, a cronjob *copies* all new data to the repository from a cache, creates the checksum file, and immediately deletes the downloaded file. Hence, the LIGO data is copied but not stored.
  - The `cms.osgstorage.org`, a cronjob queries the underlying filesystem for the relevant checksum information and published the checksum. The data is neither copied nor stored on the repository

On publication, the files may be marked as *non-compressed* and *externally stored*. This allows the CVMFS client to be configured to be pointed at a non-CVMFS data (stored as the “logical name”, not the “content addressed” form). CVMFS clients can thus use existing data sources without change.

- **How is data secured?** CVMFS was originally designed to distribute open-source software with strong data integrity guarantees. More recently, read-access authorization has been added to the software. An access control list is added to the repository (at creation time or publication time) and clients are configured to invoke a plugin for new process sessions. The plugin enforces the ACLs *and* forwards the user’s credential back to the CVMFS process. This allows the authorization to be enforced for worker node cache access and the CDN to enforce authorization on the CVMFS process for downloading new files to the cache.

  The entire ACL is passed to the external plugin and not interpreted by CVMFS; the semantics are defined by the plugin. The existing plugin is based on GSI / X509 proxies and authorization can be added based on DN or VOMS FQANs.

  In order to perform mounts, the root catalog must be accessible without authorization. However, the repository server (or CDN) can be configured to require authorization for the remaining data in the namespace.

Creating Large, Secure Repositories

For large-scale repositories, a few tweaks are useful at creation time. Here is the command used to create the `cms.osgstorage.org`:

```
cvmfs_server mkfs -V cms:/cms -X -Z none -o cmsuser cms.osgstorage.org
```

- The `-V cms:/cms` option indicates that only clients with an X509 proxy with a VOMS extension from CMS are allowed to access the mounted proxy. If multiple VOMS extensions are needed, it’s easiest to add this at publication time.
- `-X` indicates that, by default, files published to this repository are served at an “external URL”. The clients will attempt to access the file by *name*, not content hash, and look for the server as specified by the client’s setting of `CVMFS_EXTERNAL_URL`.
- `-Z none` indicates that, by default, files published to this repository will not be marked as compressed.

By combining the `-X` and `-Z` options, files at an HTTP endpoint can be published in-place: no compression or copying into a different endpoint is necessary to publish.

2.12.6 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 32 bits of the hardlinks field, and a *hardlink group* is stored in the higher 32 bits. 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 key and the value.

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)

As of bit 8, the flags store the cryptographic content hash algorithm used to process the given file. Bit 11 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 4 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[1], 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. 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

[1] As a rule of thumb, file catalogs (when compressed) are reasonably small.

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.

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*.

```
C64551dccfbe0a48de7618dd7deb290200b474759
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`)
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.

As crypto engine, CernVM-FS uses libcrypto from the OpenSSL project.

Blacklisting

In addition to validating the white-list, CernVM-FS checks certificate fingerprints against the local black-list /etc/cvmfs/blacklist and the blacklist in an optional *"Config Repository"*. The blacklisted fingerprints have to be in the same format as the fingerprints on the white-list. The black-list has precedence over the white-list.

Blacklisted fingerprints prevent clients from loading future repository publications by a corresponding compromised repository key, but they do not prevent mounting a repository revision that had previously been mounted on a client, because the catalog for that revision is already in the cache. However, the same blacklist files also support another format that actively blocks revisions associated with a compromised repository key from being mounted and even forces them to be unmounted if they are mounted. The format for that is a less-than sign followed by the repository name followed by a blank and a repository revision number:

```
<repository.name NNN
```

This will prevent all revisions of a repository called repository.name less than the number NNN from being mounted or staying mounted. An effective protection against a compromised repository key will use both this format to prevent mounts and the fingerprint format to prevent accepting future untrustworthy publications signed by the compromised key.

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 $3 \cdot$ 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. Therefore 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 the `User-Agent` header set to either of `libcvmfs` or `Fuse` depending on how it was compiled, plus the current `VERSION` value. If the `CERNVM_UUID` environment variable is set, that's also included in the `User-Agent` field.

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 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, 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 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 several 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.

ncleanup24 Shows the number of cache cleanups in the last 24 hours.

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 encoutered 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. 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.

2.13 Appendix

2.13.1 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 private part of the repository key needs to be accessible on the release manager machine. The private part of the repository master key used to sign the whitelist *can* be maintained on a file on the release manager machine. We recommend, however, to use a smart card to store this private key. See section *Master keys* for more details.

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. Sofware is available from HTTPS servers. The Linux packages and repositories are signed with a GPG key.

2.13.2 CernVM-FS Parameters

Client parameters

Parameters recognized in configuration files under /etc/cvmfs:

Parameter	Meaning
CVMFS_ALIEN_CACHE	If set, use an alien cache at the given location
CVMFS_ALT_ROOT_PATH	If set to *yes*, use alternative root catalog path. Only required for fixed catalogs (tag /
CVMFS_AUTO_UPDATE	If set to *no*, disables the automatic update of file catalogs.
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 *no*, disable checking of file ownership and permissions (open all files).
CVMFS_CLAIM_OWNERSHIP	If set to *yes*, 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 the the specified file.
CVMFS_DEFAULT_DOMAIN	The default domain will be automatically appended to repository names when given
CVMFS_DNS_MIN_TTL	Minimum effective TTL in seconds for DNS queries of proxy server names (not Stra
CVMFS_DNS_MAX_TTL	Maximum effective TTL in seconds for DNS queries of proxy server names (not Stra
CVMFS_DNS_RETRIES	Number of retries when resolving proxy names
CVMFS_DNS_TIMEOUT	Timeout in seconds when resolving proxy names
CVMFS_DNS_ROAMING	If true, watch /etc/resolv.conf for nameserver changes
CVMFS_EXTERNAL_FALLBACK_PROXY	List of HTTP proxies similar to `CVMFS_EXTERNAL_HTTP_PROXY`. The fallback
CVMFS_EXTERNAL_HTTP_PROXY	Chain of HTTP proxy groups to be used when CernVM-FS is accessing external dat
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 ser
CVMFS_EXTERNAL_URL	Semicolon-separated chain of webservers serving external data chunks.
CVMFS_FALLBACK_PROXY	List of HTTP proxies similar to `CVMFS_HTTP_PROXY`. The fallback proxies are ad
CVMFS_FOLLOW_REDIRECTS	When set to *yes*, follow up to 4 HTTP redirects in requests.
CVMFS_HIDE_MAGIC_XATTRS	If set to *yes* 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 yo
CVMFS_IGNORE_SIGNATURE	When set to *yes*, 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 paramete
CVMFS_LOW_SPEED_LIMIT	Minimum transfer rate a server or proxy must provide.
CVMFS_MAX_EXTERNAL_SERVERS	Limit the number of (geo sorted) stratum 1 servers for external data that are effective
CVMFS_MAX_IPADDR_PER_PROXY	Limit the number of IP addresses a proxy names resolves into. From all registered ad
CVMFS_MAX_RETRIES	Maximum number of retries for a given proxy/host combination.
CVMFS_MAX_SERVERS	Limit the number of (geo sorted) stratum 1 servers that are effectively used.
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 opt
CVMFS_NFILES	Maximum number of open file descriptors that can be used by the CernVM-FS proce
CVMFS_NFS_SOURCE	If set to *yes*, 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 L
CVMFS_PAC_URLS	Chain of URLs pointing to PAC files with HTTP proxy configuration information.
CVMFS_OOM_SCORE_ADJ	Set the Linux kernel's out-of-memory killer priority for the CernVM-FS client [-100
CVMFS_PROXY_RESET_AFTER	Delay in seconds after which CernVM-FS will retry the primary proxy group in case
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.

Table 1 – continued from previous pag

Parameter	Meaning
CVMFS_RELOAD_SOCKETS	Directory of the sockets used by the CernVM-FS loader to trigger hotpatching/reloa
CVMFS_REPOSITORIES	Comma-separated list of fully qualified repository names that shall be mountable un
CVMFS_REPOSITORY_DATE	A timestamp in ISO format (e.g. `2007-03-01T13:00:00Z`). Selects the reposito
CVMFS_REPOSITORY_TAG	Select a named repository snapshot that should be mounted instead of `trunk`.
CVMFS_CONFIG_REPO_REQUIRED	If set to *yes*, 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 *yes*, include the cvmfs path of downloaded data in HTTP headers.
CVMFS_SERVER_CACHE_MODE	Enable special cache semantics for a client used as a release manager repository bas
CVMFS_SERVER_URL	Semicolon-separated chain of Stratum~1 servers.
CVMFS_SHARED_CACHE	If set to *no*, makes a repository use an exclusive cache.
CVMFS_STRICT_MOUNT	If set to *yes*, mount only repositories that are listed in `CVMFS_REPOSITORIES`.
CVMFS_SUID	If set to *yes*, enable suid magic on the mounted repository. Requires mounting as roo
CVMFS_SYSLOG_FACILITY	If set to a number between 0 and 7, uses the corresponding LOCAL$n$ 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 *yes*, 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 `gid` and `uid` mount options. Don't touch or overwrite.
CVMFS_USYSLOG	All messages that normally are logged to syslog are re-directed to the given file. Thi
CVMFS_WORKSPACE	Set the local directory for storing special files (defaults to the cache directory).

Server parameters

Parameter	Meaning
CVMFS_AUFS_WARNING	Set to *false* to silence AUFS kernel deadlock warning.
CVMFS_AUTO_GC	Enables the automatic garbage collection on *publish* and *snapshot*
CVMFS_AUTO_GC_TIMESPAN	Date-threshold for automatic garbage collection (For example: *3 days ago*, *1*
CVMFS_AUTO_GC_LAPSE	Frequency of auto garbage collection, only garbage collect if last GC is befo
CVMFS_AUTO_REPAIR_MOUNTPOINT	Set to *true* to enable automatic recovery from bogus server mount states.
CVMFS_AUTO_TAG	Creates a generic revision tag for each published revision (if set to *true*).
CVMFS_AUTO_TAG_TIMESPAN	Date-threshold for automatic tags, after which auto tags get removed (For ex
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_*
CVMFS_CATALOG_ALT_PATHS	Enable/disable generation of catalog bootstrapping shortcuts during publishi
CVMFS_COMPRESSION_ALGORITHM	Compression algorithm to be used during publishing (currently either 'defau
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. (see *Requirements for*
CVMFS_ENFORCE_LIMITS	Set to *true* to cause exceeding *LIMIT variables to be fatal to a publish inste
CVMFS_EXTERNAL_DATA	Set to *true* to mark repository to contain external data that is served from an
CVMFS_FILE_MBYTE_LIMIT	Maximum number of megabytes for a published file, default value: 1024 (see
CVMFS_FORCE_REMOUNT_WARNING	Enable/disable warning through `wall` and grace period before forcefully ren
CVMFS_GARBAGE_COLLECTION	Enables repository garbage collection (Stratum~0 only | if set to *true*)
CVMFS_GENERATE_LEGACY_BULK_CHUNKS	Set to *false* to disable generation of whole-file objects for large files. Require
CVMFS_GC_DELETION_LOG	Log file path to track all garbage collected objects during sweeping for book

Table 2 – continued from previou

Parameter	Meaning
CVMFS_GID_MAP	Path of a file for the mapping of file owner group ids.
CVMFS_HASH_ALGORITHM	Define which secure hash algorithm should be used by CernVM-FS for CAS
CVMFS_IGNORE_SPECIAL_FILES	Set to *true* to skip special files during publish without aborting.
CVMFS_IGNORE_XDIR_HARDLINKS	If set to *true*, do not abort the publish operation when cross-directory hardlin
CVMFS_INCLUDE_XATTRS	Set to *true* 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_NESTED_KCATALOG_LIMIT	Maximum thousands of files allowed in nested catalogs, default 500 (see also
CVMFS_NUM_WORKERS	Maximal number of concurrently downloaded files during a Stratum1 pull op
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 *no* to skip this repository when executing `cvmfs_se`
CVMFS_REPOSITORY_NAME	The fully qualified name of the specific repository.
CVMFS_REPOSITORY_TYPE	Defines if the repository is a master copy (*stratum0*) or a replica (*stratum1*).
CVMFS_REPOSITORY_TTL	The frequency in seconds of client lookups for changes in the repository. Def
CVMFS_ROOT_KCATALOG_LIMIT	Maximum thousands of files allowed in root catalogs, default 200 (see also *C*
CVMFS_SPOOL_DIR	Location of the upstream spooler scratch directories; the read-only CernVM-
CVMFS_STRATUM0	URL of the master copy (*stratum0*) of this specific repository.
CVMFS_STRATUM1	URL of the Stratum1 HTTP server for this specific repository.
CVMFS_SYNCFS_LEVEL	Controls how often `sync` will by called by `cvmfs_server` operations. Po
CVMFS_UID_MAP	Path of a file for the mapping of file owner user ids.
CVMFS_UNION_DIR	Mount point of the union file system for copy-on-write semantics of CernVM
CVMFS_UNION_FS_TYPE	Defines the union file system to be used for the repository. (currently *aufs* an
CVMFS_UPSTREAM_STORAGE	Upstream spooler description defining the basic upstream storage type and co
CVMFS_USE_FILE_CHUNKING	Allows backend to split big files into small chunks (*true* | *false*)
CVMFS_USER	The user name that owns and manipulates the files inside the repository.
CVMFS_VIRTUAL_DIR	Set to *true* to enable the hidden, virtual `.cvmfs/snapshots` directory con
CVMFS_VOMS_AUTHZ	Membership requirement (e.g. VOMS authentication) to be added into the fil
X509_CERT_BUNDLE	Bundle file with CA certificates for HTTPS connections (see sct_data:)
X509_CERT_DIR	Directory file with CA certificates for HTTPS connections, defaults to /etc/gr

Tiered Cache Parameters

The following parameters are used to configure a tiered cache manager instance.

Parameter	Meaning
CVMFS_CACHE_$name_UPPER	Name of the upper layer cache instance
CVMFS_CACHE_$name_LOWER	Name of the lower layer cache instance
CVMFS_CACHE_LOWER_READONLY	Set to *true* to avoid populating the lower layer

External Cache Plugin Parameters

The following parameters are used to configure an external cache plugin as a cache manager instance.

Parameter	Meaning
CVMFS_CACHE_$name_CMDLINE	If the client should start the plugin, the executable and command line parameters of the plugin, separated by comma.
CVMFS_CACHE_$name_LOCATOR	The address of the socket used for communication with the plugin.

In-memory Cache Plugin Parameters

The following parameters are interpreted from the configuration file provided to the in-memory cache plugin (see Section *Example*).

Parameter	Meaning
CVMFS_CACHE_PLUGIN_DEBUGLOG	If set, run CernVM-FS in debug mode and write a verbose log the the specified file.
CVMFS_CACHE_PLUGIN_LOCATOR	The address of the socket used for client communication
CVMFS_CACHE_PLUGIN_SIZE	The amount of RAM in megabyte used by the plugin for caching.

2.13.3 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:

- Appropriate kernel version. You must have ONE of the following:
  - kernel 4.2.x or later.
  - RHEL7.3 kernel (for OverlayFS)
  - Custom kernel compilation with *aufs* support 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 "*S3 Compatible Storage Systems*" for details). In the former case the contents of `/srv/cvmfs` are as follows:

File Path	Description
`/srv/cvmfs`	**Central repository storage location** Can be mounted or symlinked to another location *before* creating the first repository.
`/srv/cvmfs/<fqrn>`	**Storage location of a specific repository** Can be symlinked to another location *before* 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`	**Manifest file of the repository** 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>/.cvmfswhitelist`	**List of trusted repository certificates** 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`	**CAS location of the repository** 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`	**Second CAS level directories** 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`	**CAS transaction directory** 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
`/var/spool/cvmfs`	**CernVM-FS server spool area** Contains administrative and scratch space for CernVM-FS repositories. This directory should only contain directories corresponding to individual CernVM-FS repositories.
`/var/spool/cvmfs/<fqrn>`	**Individual repository spool area** 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.
`/var/spool/cvmfs/<fqrn>/cache`	**CernVM-FS client cache directory** Contains the cache of the CernVM-FS client mounting the r/o branch (i.e. `/var/spool/cvmfs/<fqrn>/rdonly`) of the union file system mount point located at `/cvmfs/<fqrn>`. 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.
`/var/spool/cvmfs/<fqrn>/rdonly`	**CernVM-FS client mount point** 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.
`/var/spool/cvmfs/<fqrn>/scratch`	**Writable union file system scratch area** All file system changes applied to `/cvmfs/<fqrn>` 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.
`/var/spool/cvmfs/<fqrn>/tmp`	**Temporary scratch location** 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.
`/var/spool/cvmfs/<fqrn>/client.config`	**CernVM-FS client configuration** This contains client configuration variables for the CernVM-FS client mounted to `/var/spool/cvmfs/<fqrn>/rdonly`. Most notibly it needs to contain `CVMFS_ROOT_HASH` 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>.crt` 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	**CernVM-FS server config directory** This contains the configuration directories for individual CernVM-FS repositories. Note that this path is shortened using /.../repos.d/ in the rest of this table.
/.../repos.d/<fqrn>	**Config directory for specific repo** This contains the configuration files for one specific CernVM-FS repository server.
/.../repos.d/<fqrn>/server.conf	**Server configuration file** Authoriative configuration file for the CernVM-FS server tools. This file should only contain *valid server configuration variables* as it controls the behaviour of the CernVM-FS server operations like publishing, pulling and so forth.
/.../repos.d/<fqrn>/client.conf	**Client configuration file** Authoriative 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 *valid client configuration variables*. This file must not exist for Stratum 1 repositories.
/.../repos.d/<fqrn>/replica.conf	**Replication configuration file** 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 "*S3 Compatible Storage Systems*" 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
```

2.13.4 Available Packages

The CernVM-FS software is available in form of several 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]* as well as the `libcvmfs_cache.a` static library and `libcvmfs_cache.h` header in order to develop cache plugins.

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.

2.13.5 References

CHAPTER 3

Contact and Authors

Visit our website on cernvm.cern.ch.

Authors of this documentation:

- Jakob Blomer
- Predrag Buncic
- Dave Dykstra
- Nikola Hardi
- 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 (Feburary 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).