LizardFS Documentation

Release latest

Michal Bielicki

Jun 15, 2018

Contents

Contents:

CHAPTER 1

ToDo-List

CHAPTER 2

Introduction to LizardFS

LizardFS is a distributed, scalable, fault-tolerant and highly available file system. It allows users to combine disk space located on many servers into a single name space which is visible on Unix-like and Windows systems in the same way as other file systems. LizardFS makes files secure by keeping all the data in many replicas spread over available servers. It can be used also to build an affordable storage, because it runs without any problems on commodity hardware.

Disk and server failures are handled transparently without any downtime or loss of data. If storage requirements grow, it's possible to scale an existing LizardFS installation just by adding new servers - at any time, without any downtime. The system will automatically move some data to newly added servers, because it continuously takes care of balancing disk usage across all connected nodes. Removing servers is as easy as adding a new one.

Unique features like:

- support for many data centers and media types,
- fast snapshots,
- transparent trash bin,
- QoS mechanisms,
- quotas

and a comprehensive set of monitoring tools make it suitable for a range of enterprise-class applications.

2.1 Architecture

LizardFS keeps metadata (like file names, modification timestamps, directory trees) and the actual data separately. Metadata is kept on metadata servers, while data is kept on machines called chunkservers. A typical installation consists of:

- At least two metadata servers, which work in master-slave mode for failure recovery. Their role is also to manage the whole installation, so the active metadata server is often called the master server. The role of other metadata servers is just to keep in sync with the active master servers, so they are often called shadow master servers. Any shadow master server is ready to take the role of the active master server at any time. A suggested configuration of a metadata server is a machine with fast CPU, at least 32 GB of RAM and at least one drive (preferably SSD) to store several dozens of gigabytes of metadata.

- A set of chunkservers which store the data. Each file is divided into blocks called chunks (each up to 64 MiB) which are stored on the chunkservers. A suggested configuration of a chunkserver is a machine with large disk space available either in a JBOD or RAID configuration, depending on requirements. CPU and RAM are not very important. You can have as little as 2 chunkservers (a minimum to make your data resistant to any disk failure) or as many as hundreds of them. A typical chunkserver is equipped with 8, 12, 16, or even more hard drives. Each file can be distributed on the chunkservers in a specific replication mode which is one of standard, xor or ec.
- Clients which use the data stored on LizardFS. These machines use LizardFS mount to access files in the installation and process them just as those on their local hard drives. Files stored on LizardFS can be seen and simultaneously accessed by as many clients as needed.

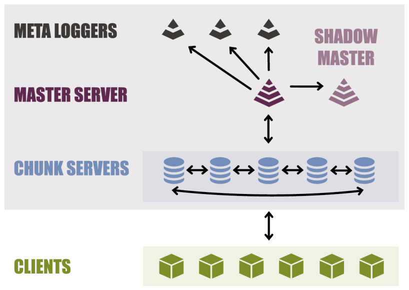

Fig. 1: Figure 1: Architecture of LizardFS

2.2 Replication-Modes

The replication-mode of a directory or even of a file can be defined individually.

standard this mode is for defining explicitely how many copies of the data-chunks you want to be stored in your cluster and on **which group of nodes the copies reside**. In conjunction with "custom-goals" this is handy for geo-replication.

xor xor is similar to the replication-mechanism also known by RAID5. For Details see the whitepaper on lizardfs.

ec - erasure coding ec mode is similar to the replication-mechanism also known by RAID6. In addition you can use parities above 2. For Details see the whitepaper on l izardfs.

2.3 Possible application of LizardFS

There are many possible applications of LizardFS

- archive - with using LTO tapes,
- storage for virtual machines (as an OpenStack backend or similar)
- storage for media files / cctv etc.
- storage for backups
- as a storage for Windows™ machine
- as a DRC (Disaster Recovery Center)
- HPC

2.4 Hardware recommendation

LizardFS will be working on any hardware, you can use commodity hardware as well. Minimum requirements is two dedicated nodes with a bunch of disks, but to achieve proper HA installation you should have at least 3 nodes. We recommend that each node shall have at least two 1Gbps network interface controllers (NICs). Since most commodity hard disk drives have a throughput of approximately 100MB/second, your NICs should be able to handle the traffic between the chunkservers and your host.

Minimal configuration of LizardFS strongly depends on its use case. LizardFS will run on practically any reasonable machine, but a sample configuration for a medium size installation could be as follows:

Master / Shadow

- CPU - at least 2 GHz CPU, 64bit
- RAM - depends on expected number of files (4GB should be enough for a small installation)
- Disk - 128G, SSD would improve performance, HDD is fine

Chunkserver - recommended 2GB RAM (or more)

Metalogger - recommended 2GB RAM (or more)

2.5 Additional Features

What makes LizardFS a mature enterprise solution are additional features developed on the basis of a constantly improving core. They can transform the probably best distributed file system in the world into Hierarchical Storage Management (HSM), help to build Disaster Recovery Center with asynchronous replication between sites, reduce disk space required for replication, effectively manage storage pools (QoS, Quotas) and many more. If you see any other use case for LizardFS that would require any other functionality please let us know, we might put it into our Road Map or develop it especially for you.

2.5.1 Support for LTO Libraries

LizardFS offers native support for LTO libraries. Storing archival backups may consume a lot of memory, even though those files are almost never read. Such data can be efficiently stored on a tape, so LizardFS offers a simple way to cooperate with back-end LTO storage. Files can be chosen to have a backup copy on a tape by setting a tape goal. Examples of tape goals can be found in chapter “Advanced configuration”.

Setting a tape goal to a file makes it read-only for obvious reasons - tape storage does not support random writes. Reading from tape storage is a timely process (may last 48h or require manual work to insert correct tape to library), so data stored in there should be archival - meant to be read very rarely.

The way of reading a file which is stored on tape depends on its situation:

- If a regular copy of a file is still available, it will be used for reading
- If a file exists only on tape, it has to be restored to LizardFS first. To achieve that, one must use lizardfs-restore-tape-copy utility:

  ```
  $ lizardfs-restore-tape-copy file_path
  ```

  After running this command, all needed data will be read from tape storage and loaded to the file system, making the file accessible to clients.

CHAPTER 3

LizardFS QuickStart on Debian

Important: In case you downloaded the packages from the official Debian repository, be aware of the differences in certain names and paths, including:

- Configuration files directory is /etc/lizardfs instead of /etc/mfs, data directory is /var/lib/lizardfs instead of /var/lib/mfs
- Sample configuration files can be found in /usr/share/doc/lizardfs-<name>/examples, where <name> can be master, chunkserver or metalogger.
- Chunk servers are run as user lizardfs, instead of user mfs.

In order to allow lizardfs-<name> to be run as service, run the following command:

```
$ systemctl enable lizardfs-<name>
```

where <name> is one of master, chunkserver, metalogger, cgiserv

3.1 Master server installation

Install the master server package

Check *Installing from Debian packages* for instructions how to install the package.

Example:

```
$ apt-get install lizardfs-master
```

Fill the configuration files with appropriate values.

This involves setting up the following configuration files in /etc/mfs directory:

Filename	Description	Required
mfsmaster.cfg	Master configuration file	X
mfsexports.cfg	Mountpoint locations configuration	X
mfsgoals.cfg	Replication goals configuration	
mfstopology.cfg	Network topology definitions	

Documentation for each file can be viewed by entering:

```
$ man <filename>
```

in your shell.

Sample configuration files can be found in /etc/mfs/*.dist

- Prepare the data directory /var/lib/mfs
- Create empty metadata.mfs file:

  ```
  $ cp /var/lib/mfs/metadata.mfs.empty /var/lib/mfs/metadata.mfs
  ```

The data directory will contain all changelogs and metadata files of your installation.

3.1.1 Example configuration

In our example configuration, the mfsmaster.cfg file can remain untouched.

In order to let clients from IP range 192.168.18.* read and write to our installation, add this line to mfsexports.cfg:

```
192.168.18.0/24 / rw,alldirs,maproot=0
```

In order to use lizardfs-master as a service (recommended), edit /etc/default/ lizardfs-master file and set:

```
LIZARDFSMASTER_ENABLE=true
```

After this operation, you can launch LizardFS master daemon:

```
$ service lizardfs-master start
```

Your first instance of LizardFS should have been successfully launched!

Explore your new master server's capabilities by looking into the mfsmaster man pages:

```
$ man mfsmaster
```

3.2 Shadow master installation

- Follow the steps for installing a master server.
- Add mfsmaster entry to /etc/hosts, as in chunkserver steps.
- Add this line to master's config (mfsmaster.cfg):

  ```
  PERSONALITY=shadow
  ```

- Run master service.

You now possess a shadow master server which makes your data much safer, ie. all data files from the master server are also saved to the shadow master.

3.3 Metalogger installation

Install the metalogger package

Check *Installing from Debian packages* for instructions how to install the package

Example for Debian/Ubuntu:

```
$ apt-get install lizardfs-metalogger
```

Fill the configuration file with appropriate values. You can find it in the /etc/mfs directory and it is called:

```
mfsmetalogger.cfg
```

Documentation for this file can be viewed by entering:

```
$ man mfsmetalogger.cfg
```

in your shell.

Sample configuration files can be found in /etc/mfs/*.dist

For our example configuration, mfsmetalogger.cfg may remain unchanged.

By default, the metalogger uses the "mfsmaster" host as LizardFS master's address. It is advised to set it up in /etc/hosts file.

For example configuration mentioned at the top, /etc/hosts should include this line:

```
192.168.16.100 mfsmaster
```

Allow metalogger to be run as service by editing /etc/default/lizardfs-metalogger file:

```
LIZARDFSMETALOGGER_ENABLE=true
```

Run your metalogger:

```
$ service lizardfs-metalogger start
```

3.4 Chunk server installation

Install chunk server package Check *Installing from Debian packages* for instructions how to install package

Example for Debian/Ubuntu:

```
$ apt-get install lizardfs-chunkserver
```

Fill configuration files with appropriate values.

It involves setting up following configuration files in /etc/mfs directory:

Filename	Description
mfschunkserver.cfg	Chunk server configuration file
mfshdd.cfg	Hard drive location settings

Documentation for each file can be viewed by entering:

```
$ man <filename>
```

in your shell.

Sample configuration files can be found in /etc/mfs/*.dist

By default, chunk server uses "mfsmaster" host as LizardFS master's address. It is advised to set it up in /etc/hosts file. For example configuration mentioned at the top, /etc/hosts should include this line:

```
192.168.16.100 mfsmaster
```

The mfshdd.cfg file is needed to indicate mountpoints of hard drives for your chunkserver. Assuming that there are 2 disks mounted at /mnt/chunk1 and /mnt/chunk2 locations, your mfshdd.cfg file should look like this:

```
/mnt/chunk1
/mnt/chunk2
```

Remember that chunk servers are run as user mfs, so directories above need appropriate permissions:

```
$ chown -R mfs:mfs /mnt/chunk1
$ chown -R mfs:mfs /mnt/chunk2
```

3.4.1 Allow chunk server to be run as a service

As before, this can be achieved by editing /etc/default/lizardfs-chunkserver file:

```
LIZARDFSCHUNKSERVER_ENABLE=true
```

Type:

```
$ service lizardfs-chunkserver start
```

and congratulate yourself on launching your first LizardFS chunk server.

3.5 Cgi server installation

The cgi server offers a Web-based GUI that presents LizardFS status and various statistics.

Install the cgi-server package

Check *Installing from Debian packages* for instructions how to install package

Example for Debian/Ubuntu:

```
$ apt-get install lizardfs-cgiserv
```

Set mfsmaster host in /etc/hosts file. For our example configuration it would be:

```
192.168.16.100 mfsmaster
```

Run your cgi-server:

```
$ service lizardfs-cgiserv start
```

The Web interface is now available.

Assuming that lizardfs-cgiserv is installed on host 192.168.10.11, you can access LizardFS panel at http://192.168.10.11:9425/mfs.cgi?masterhost=mfsmaster

3.6 Command line administration tools

Install administration tools package

Check *Installing from Debian packages* for instructions how to install package

Example for Debian/Ubuntu:

```
$ apt-get install lizardfs-adm
```

See variety of options by running those commands:

```
$ man lizardfs-admin or $ lizardfs-admin -h
```

Now that you are done with your quick and dirty installation, you can try connecting clients to your fresh LizardFS instance. This is documented in the *Connecting Clients to your LizardFS installation* part of the *LizardFS Administration Guide*.

CHAPTER 4

LizardFS Administration Guide

An Administrators Guide to installing, configuring and managing LizardFS

Contents:

4.1 Installing LizardFS

4.1.1 Getting and installing LizardFS

There are several methods for getting LizardFS software. The easiest and most common method is to get packages by adding repositories for use with package management tools such as the Advanced Package Tool (APT) or Yellowdog Updater, Modified (YUM). You may also retrieve pre-compiled packages from the LizardFS repository. Finally, you can retrieve tarballs or clone the LizardFS source code repository and build LizardFS yourself.

Installing from Debian packages

First, add a our key which is needed to verify the signatures of LizardFS packages:

```
# wget -O - http://packages.lizardfs.com/lizardfs.key | apt-key add -
```

Now add a proper entry in /etc/apt/sources.list.d/

For Debian do:

```
# apt-get install lsb-release
# echo "deb http://packages.lizardfs.com/debian/$(lsb_release -sc) $(lsb_release -sc)␣
↪main" > /etc/apt/sources.list.d/lizardfs.list
# echo "deb-src http://packages.lizardfs.com/debian/$(lsb_release -sc) $(lsb_release -
↪sc) main" >> /etc/apt/sources.list.d/lizardfs.list
```

For Ubuntu do:

```
# echo "deb http://packages.lizardfs.com/ubuntu/$(lsb_release -sc) $(lsb_release -sc)␣
↪main" > /etc/apt/sources.list.d/lizardfs.list
# echo "deb-src http://packages.lizardfs.com/ubuntu$(lsb_release -sc) $(lsb_release -
↪sc) main" >> /etc/apt/sources.list.d/lizardfs.list
```

This will have created the lizardfs.list file. To use the newly added repository, update the packages index:

```
# apt-get update
```

Now you are able to install LizardFS packages (listed below) using:

```
# apt-get install <package>
```

It is also possible to download the source package using:

```
# apt-get source lizardfs
```

Important: Before upgrading any existing LizardFS installation, please read the instructions here: *Upgrading Lizardfs*

LizardFS consists of the following packages:

- lizardfs-master – LizardFS master server
- lizardfs-chunkserver – LizardFS data server
- lizardfs-client – LizardFS client
- lizardfs-adm – LizardFS administration tools (e.g, lizardfs-probe)
- lizardfs-cgi – LizardFS CGI Monitor
- lizardfs-cgiserv – Simple CGI-capable HTTP server to run LizardFS CGI Monitor
- lizardfs-metalogger – LizardFS metalogger server
- lizardfs-common – LizardFS common files required by lizardfs-master, lizardfs-chunkserver and lizardfs-metalogger
- lizardfs-dbg – Debugging symbols for all the LizardFS binaries

Installing from RedHAT EL / CentOS Packages

First, add a file with information about the repository:

for RHEL 6 and CentOS 6:

```
# curl http://packages.lizardfs.com/yum/el6/lizardfs.repo > /etc/yum.repos.d/lizardfs.
↪repo
# yum update
```

for RHEL 7 and CentOS 7:

```
# curl http://packages.lizardfs.com/yum/el7/lizardfs.repo > /etc/yum.repos.d/lizardfs.
↪repo
# yum update
```

To get libjudy which is used by parts of LizardFS, install the epel repository for RHEL 7:

```
yum install epel-release
```

Now you are able to install LizardFS packages (listed below) using:

```
# yum install <package>
```

It is also possible to download the source package (SRPM) using:

```
# yum install yum-utils
# yumdownloader --source lizardfs
```

Important: Before upgrading any existing LizardFS installation, please read the instructions here: *Upgrading Lizardfs*

LizardFS consists of following packages:

- lizardfs-master – LizardFS master server
- lizardfs-chunkserver – LizardFS data server
- lizardfs-client – LizardFS client
- lizardfs-adm – LizardFS administration tools (e.g, lizardfs-probe)
- lizardfs-cgi – LizardFS CGI Monitor
- lizardfs-cgiserv – Simple CGI-capable HTTP server to run LizardFS CGI Monitor
- lizardfs-metalogger – LizardFS metalogger server
- lizardfs-debuginfo – (CentOS 7 / RHEL 7 only) Debugging symbols and sources for all the LizardFS binaries

Installing LizardFS from downloaded .deb packages

Make sure to install the *lizardfs-common* package first before installing other packages.

Also, remember to install lizardfs-cgi before installing lizardfs-cgiserv

In order to install a .deb package, run:

```
# dpkg -i <package>
```

If installing fails due to dependency problems, run:

> # apt-get -f install

Installing LizardFS from source

Installing LizardFS from source.

The current LizardFS source code can be obtained from our *github* (https://github.com/lizardfs/lizardfs) project page. You can either download a tarball from there by choosing the respective version in the **Branch** tab on the left or use *git* to clone the sourcetree.

LizardFS uses *CMake* as its build system. To compile the sources, follow the directions outlined below.

1. Create a build directory inside the source directory:

```
cd lizardfs-source
mkdir build
```

2. Run *cmake ..* inside the build directory. Useful options include *-DCMAKE_INSTALL_PREFIX*, *-DCMAKE_BUILD_TYPE* as well as various LizardFS-specific *-DENABLE_...* options. Options are listed when cmake is ran and can be changed by re-running cmake:

```
cd build
cmake .. -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/opt/lizardfs
```

3. Run make in the build directory:

```
make
```

4. Run make install to install files (you may need to be root):

```
make install
```

5. Now continue to the configuration pages.

If you want to participate in developing LizardFS, please refer to the *LizardFS Developers Guide* and the *Participation Rules*.

4.2 Basic Configuration

Some basic rules first:

- The Master server works best on a dedicated machine, preferably with a SSD drive.
- The Chunk server works best with at least one dedicated disk.
- Do not put a metalogger on the same machine as the master server, it doesn't make your metadata any safer. Metaloggers are completely optional, though ;) It is fine, however, to install a metalogger along with a chunk server
- Increase your data safety by using shadow master servers

There are a range of settings that must be done on every server before we start working on the LizardFS configuration itself.

- adjust network settings
- adjust kernel settings

4.2.1 Basic operating system settings for LizardFS

LizardFS runs on many UNIX and UNIX like operating systems. We will be concentrating here on Debian and RH like systems to keep things simple. For other OSes, please consult the *LizardFS'ers CookBook*.

The /etc/hosts file

Since LizardFS is a network based Filesystem, your network setup is crucial, especially the name resolution service which helps computers in the network find each other. There are two basic ways that computers find each other by name: static entries in the /etc/hosts file or the DNS System.

The DNS system provides a central database that resolves names to their respective IP addresses. In many environments DNS is either automatic and linked to other services (Active Directory, DHCP, etc ..) so that we strongly recommend to use the /etc/hosts file to resolve between the LizardFS cluster components. This will keep the cluster from loosing Data in case somebody changes something in the DNS service. It involves some additional work from the Administrator responsible for the LizardFS cluster to keep the files up to date, but will safeguard you from storage failure and data loss in case something happens to the DNS system.

The /etc/hosts file has not changed in the last 5 decades and is basically the same on all operating systems. it's structure is:

```
<ip-address>   <hostname> <alias>
```

one line per IP address.

Example:

```
127.0.0.1        localhost
192.168.16.100   mfsmaster
192.168.16.101   shadowmaster
192.168.16.10    chunkserver1 metalogger
192.168.16.11    chunkserver2 cgiserver
192.168.16.12    chunkserver3
```

In this example your Master is on 192.168.16.100 and your Shadow Master on 192.168.16.101. Chunkserver1 also provides the metalogger and is located at 192.168.16.10. Chunkserver2 also provides the cgi web interface and is located at 192.168.16.11. Chunkserver3 is at 192.168.11.12.

The ntpd time service

Basic network adjustments for Linux

By default Linux systems come with relatively small window sizes for tcp and udp frames which could lead to fragmentation and lowered performance. Especialy if your servers use 10G or faster interfaces we would recommend to adjust your network settings by adding the following entries to your /etc/ sysctl.conf file or placing a new file called "lizardfs.conf" into the /etc/ sysconf.d/ directory containing the following entries:

```
net.ipv4.tcp_window_scaling = 1

net.core.rmem_max=1677721600
net.core.rmem_default=167772160
net.core.wmem_max=1677721600
net.core.wmem_default=167772160
net.core.optmem_max= 2048000

# set minimum size, initial size, and maximum size in bytes
net.ipv4.tcp_rmem= 1024000 8738000 1677721600
net.ipv4.tcp_wmem= 1024000 8738000 1677721600
net.ipv4.tcp_mem= 1024000 8738000 1677721600
net.ipv4.udp_mem= 1024000 8738000 1677721600
net.core.netdev_max_backlog = 30000
net.ipv4.tcp_no_metrics_save = 1
```

These values are taken from a 10G setup, you probably need to adjust them if you have lower or faster networking interfaces.

4.2.2 File systems for LizardFS servers

Note: Due to its speed and stability we recommend the XFS or ZFS filesystems on production servers. XFS was developed for Silicon Graphics, and is a mature and stable filesystem. ZFS was developed by SUN Microsystems and is used for heavy duty storage by Systems from numerous Storage System Vendors.

For the Master servers

The Master keeps all of his records in memory but does frequent backups to drives which should therefore be very fast, but do not have to be very large. A standard 250G SSD should suffice. As a file system we would recommend something fast, like XFS. Do not use a HW RAID controller to mirror your drives, SSDs usualy have identical lifespan so block level mirroring would just lead to two dead SSDs instead of one. An alternative would be ZFS mirroring which is not lowlevel but data based and does not always write the same block to both devices to the same position.

For the chunkservers

If you are using XFS as the filesystem for the chunkserver directories, we recommend the following mount options:

```
rw,noexec,nodev,noatime,nodiratime,largeio,inode64,barrier=0
```

This disables unneeded features from the mount which gives a slight performance increase especially in case of many chunks. It also increases the size of the directories gives lizardfs more space in to put its data.

Depending on the hardware you use and ifyou are utilising caching RAID controllers, it could make sense to adjust the scheduler on your filesystems. How you do a that is documented here:

http://www.cyberciti.biz/faq/linux-change-io-scheduler-for-harddisk/

Probably you will want the *deadline* scheduler but your mileage may vary.

Why you should do that and what performance gains you may achieve can be found here:

http://xfs.org/index.php/XFS_FAQ

If you would like to use the high performance ZFS filesystem, please check the *LizardFS'ers CookBook* for further information.

4.2.3 Configuring your Master

The master server is the heart of the LizardFS ecosystem. It keeps all meta information about every file, every chunk and every slice if in ec mode. It knows what is where and how to find it. It is also resposible to organize georeplication and topology and fix the effects of broken drives and chunkservers.

The metadata database

For the master to work, you need to first give it a file where it will keep its metadata database. The default location, which can be adjusted in the *mfsmaster.cfg(5)* file, is:

```
/var/lib/mfs/metadata.mfs
```

There is an empty metdata file available which you can use to create a new one. If you want to use the default location, just issue a:

```
$ cp /var/lib/mfs/metadata.mfs.empty /var/lib/mfs/metadata.mfs
```

to copy the empty template into the default location and create a new database.

Now that you have a metadata database, you need to provide your master server with the required information for operation.

The mfsmaster.cfg file

In the mfsmaster.cfg file, there are a lot of settings for advanced usage which we will get into in the *Advanced configuration* Guide. For a basic setup the things that are important are:

Current *personality* of this instance of the metadata server. Valid values are *master*, *shadow* and *ha-cluster-managed*. If the installation is managed by an HA cluster the only valid value is *ha-cluster-managed*, otherwise the only valid values are *master* and *shadow*, in which case only one metadata server in LizardFS shall have *master* personality.

```
PERSONALITY = master
```

means that this instance of metadata server acts as main metadata server govering all file system metadata modifications.

```
PERSONALITY = shadow
```

means that this instance of the metadata server acts as backup metadata server ready for immediate deployment as the new *master* in case of a failure of the current *master*.

Metadata server personality can be changed at any moment as long as one changes personality from *shadow* to *master*, changing personality the other way around is forbidden.

```
PERSONALITY = ha-cluster-managed
```

means that this instance is managed by a HA cluster, server runs in *shadow* mode as long as its not remotly promoted to *master*. For details on running LizardFS with HA Master please refer to *Deploying LizardFS as a HA Cluster* .

The addresses your master server is to listen on, if not all:

```
ATOML_LISTEN_HOST # IP address to listen on for metalogger connections (* means any)
MATOCS_LISTEN_HOST # IP address to listen on for chunkserver connections (* means any)
MATOTS_LISTEN_HOST # IP address to listen on for tapeserver connections (* means any)
MATOCL_LISTEN_HOST # IP address to listen on for client (mount) connections (* means␣
↪any)
```

The ports your master server is supposed to listen on, if not the default ones:

```
MATOML_LISTEN_PORT # port to listen on for metalogger connections (default is 9419)
MATOCS_LISTEN_PORT # port to listen on for chunkserver connections (default is 9420)
MATOCL_LISTEN_PORT # port to listen on for client (mount) connections (default is␣
↪9421)
MATOTS_LISTEN_PORT # Port to listen on for tapeserver connections (default is 9424)
```

The user and group you would like your master to run as (default is *mfs*):

```
WORKING_USER # user to run daemon as
WORKING_GROUP # group to run daemon as (optional - if empty then the default user␣
↪group will be used)
```

Where to store metadata and lock files:

```
DATA_PATH # where to store metadata files and lock file
```

Should the access time for every file be recorded or not ?

```
NO_ATIME
# when this option is set to 1 inode access time is not updated on every #
# access, otherwise (when set to 0) it is updated (default is 0)
```

All other settings should be left alone for a basic system.

Layout, access rights and other options

Now that we have the main configuration done, lets configure the layout of our LizardFS. This is done in the *mfsexports.cfg(5)* file, unless you specify a different file in your *mfsmaster.cfg(5)* file.

Note: LizardFS creates one big namespace. For fine tuned access you should create entries here for subdirectories and assign those to groups to have different clients access only different parts of the tree.

This file contains all the settings required to create a LizardFS namespace and set its access rights and network permissions. Its format is pretty simple:

```
ADDRESS DIRECTORY [OPTIONS]
```

Basically you define which network address or address range has access to which directory plus options for that access.

The address scheme looks like the following:

*	all addresses
n.n.n.n	single IP address
n.n.n.n/b	IP class specified by network address and bits number
n.n.n.n/m.m.m.m	IP class specified by network address and mask
f.f.f.f-t.t.t.t	IP range specified by from-to addresses (inclusive)

Your LizardFS namespace is a tree, starting with the root entry **/**. So in the directory field you can specify the whole namespace, **/**, or subdirectories like: **/home** or **/vm1**. The special value **.** represents the meta file system, which is described in *Mounting the metadata* . You can specify different access rights, options, passwords and user mappings for every single directory and split your namespace utlising those options into multiple sub namespaces if required. Check out the examples for how different directories can be set to different options.

Options

To give you maximum flexibility LizardFS provides a range of mount options so you can finetune settings for every piece of your namespace.

None of them are required. If you do not provide any options, the default set of:

```
ro,maproot=999:999
```

will be used.

The options are:

ro, readonly export tree in read-only mode (default)

rw, readwrite export tree in read-write mode

ignoregid disable testing of group access at *mfsmaster* level (it's still done at *mfsmount* level) - in this case "group" and "other" permissions are logically added; needed for supplementary groups to work. (*mfsmaster* only receives information about the users primary group)

dynamicip allows reconnecting of already authenticated client from any IP address (the default is to check the IP address on reconnect)

maproot=USER[:GROUP] maps root (uid=0) accesses to the given user and group (similarly to maproot option in NFS mounts); USER and GROUP can be given either as name or number; if no group is specified, USERs primary group is used. Names are resolved on *mfsmaster* side (see note below).

mapall=USER[:GROUP] like above but maps all non privileged users (uid!=0) accesses to a given user and group (see notes below).

minversion=VER rejects access from clients older than specified

mingoal=N, maxgoal=N specifies range in which goal can be set by users

mintrashtime=TDUR, *maxtrashtime=TDUR specifies range in which trashtime can be set by users. See *Trash directory*

password=PASS, md5pass=MD5 requires password authentication in order to access specified resource

alldirs allows to mount any subdirectory of the specified directory (similarly to NFS)

nonrootmeta allows non-root users to use filesystem mounted in the meta mode (option available only in this mode). See *Mounting the metadata* .

Examples

```
*                       /       ro
# Give everybody access to the whole namespace but read-only. Subdirs can
# not be mounted directly and must be accessed from /.

192.168.1.0/24          /       rw
# Allow 192.168.1.1 - 192.168.1.254 to access the whole namespace read/write.

192.168.1.0/24          /       rw,alldirs,maproot=0,password=passcode
# Allow 192.168.1.1 - 192.168.1.254 to access the whole namespace read/write
# with the password *passcode* and map the root user to the UID *0*.

10.0.0.0-10.0.0.5       /test   rw,maproot=nobody,password=test
# Allow 10.0.0.0 - 10.0.0.5 to access the directory /test except for its
# subdirectores in a read/write fashion using the password *test*. Map all
# accesses by the root user to the user *nobody*.

10.1.0.0/255.255.0.0 /public rw,mapall=1000:1000
# Give access to the /public directory to the network 10.1.0.0/255.255.0.0
# in a read/write fashion and map everybody to the UID *1000* and GID *1000*.

10.2.0.0/16             /      rw,alldirs,maproot=0,mintrashtime=2h30m,maxtrashtime=2w
# Give access to the whole namespae to the 10.2.0.0/16 network in a
# read/write fashion. Also allowsubdirectories to be mounted directly by
# those clients.
# Map the root user to UID *0*. Allow users to set the trahtime (time when
```

(continues on next page)

(continued from previous page)

```
# files in the tash get autopruned) between
# 2h30m and 2 weeks.
```

Utilising all of these options you will be able to do quite flexible setups, like optimizing for virtualization as described in out Cookbook at *Using LizardFS for Virtualization Farms* .

Now that you know how to setup your namespace, the next step would be to set custom goals/replication modes, described in *Configuring Replication Modes* and QoS/IO Limits, described in the *Configuring QoS* chapter.

Network awareness / topology are further advanced topics, especialy required for georeplication. A description of how to set them up can be found here *Configuring rack awareness (network topology)* .

4.2.4 Configuring your Shadowmaster

Your shadowmaster is configured in nearly the same way as your Master. Since it is supposed to take over the functionality of the Master in case of a failure of the Master, it has to keep its metadatabase in sync and besides that have all the configurations of the masterserver mirrored.

Settings specific to the Shadowmaster:

In the mfsmaster.cfg file:

```
# Set the personality to be that of a Shadowmaster:
PERSONALITY = shadow

# Set the address where the metadatabase is synced from:
MASTER_HOST = 10.0.10.230
```

The files mfsexports.cfg, mfsgoals.cfg and mfstopology.cfg must be synchronized with the master server.

4.2.5 Configuring your Chunkservers

Your chunkservers are pretty simple to set up. Usualy, if your /etc/hosts files are setup correctly with the address of the master server and you do not require labeling (*Labeling your chunkserver*), the mfschunkserver.cfg file can stay as it is. If you require to lock down the masterserver address, adjust the following line:

```
MASTER_HOST = 10.0.10.230
```

to lock down the master server to the 10.0.10.230 address.

Now you need to specify where the chunkserver process will keep the actual data. This is done in the mfshdd.cfg file. You specify directories with their full path, one per line.

Example:

```
# use directory '/mnt/hd1' with default options:
/mnt/hd1

# use directory '/mnt/hd2', but replicate all data from it:
*/mnt/hd2

# use directory '/mnt/hd3', but try to leave 5GiB on it:
/mnt/hd3 -5GiB

# use directory '/mnt/hd4', but use only 1.5TiB on it:
/mnt/hd4 1.5TiB
```

The settings always assume that the directory is a dedicated device, so a HDD, a Raidset or a SSD and bases it's space calculation on that.

Once this is setup, your chunkserver is ready and actively taking part in your lizardfs.

To remove a directory from being used by lizardfs, just add a * to the beginning of the line in mfshdd.cfg:

```
# use directory '/mnt/hd2', but replicate all data from it:
*/mnt/hd2
```

Lizardfs will replicate all the data from it somewhere else. Once you see in the webinterface that all data has been safely copied away, you can update the file and remove the line and than remove the device associated with it from your chunkserver.

4.2.6 Configuring the Metalogger

The metalogger is used for desaster recovery should the master and shadowservers fail. The metadatabase can be rebuild from them. The setup is straightforward. You basically do not need to setup anything if your /etc/hosts is setup accordingly, otherwise you need to set the following in your mfsmetalogger.cfg file:

```
MASTER_HOST
# address of LizardFS master host to connect with (default is mfsmaster)

MASTER_PORT
# number of LizardFS master port to connect with (default is 9419)
```

and you are ready to go.

4.2.7 Configuring the Web Interface

The lizardfs cgiserver does not require much configuration. After the installation either follow the example installation and just add an entry for *mfsmaster* to your /etc/hosts file, or, . . .

4.2.8 Labeling your chunkserver

To be able to setup which goals are going to be performed on which chunkservers, you need to be able to identify them in your goal definition. To achieve this, we use labels.

The label for the Chunkservers is set in the mfschunkserver.cfg file.

```
LABEL = ssd
```

After changing the configuration you must reload the chunkserver:

```
$ mfschunkserver -c path/to/config reload
```

If there is no LABEL entry in the config, the chunkserver has a default label of "_" (i.e. wildcard), which has a special meaning when defining goals and means "any chunkserver".

Multiple chunkservers can have the same label than they are basically a group of chunkservers where you can use the label to write a chunk or a piece of XOR or EC set to "any" chunkserver in that group.

Show labels of connected chunkservers

From the command line:

```
$ lizardfs-admin list-chunkservers <master ip> <master port>
```

Via the cgi (webinterface):

In the 'Servers' tab in the table 'Chunk Servers' there is a column 'label' where labels of the chunkservers are displayed.

4.3 Configuring Replication Modes

LizardFS supports 3 different modes of replication.

Note: All replication settings are based on chunks, not on nodes. So if you have lets say, 5 chunkservers in a EC3+1 configuration, chunks will spread among them but one set will always be 4 chunks. This way all active chunkservers are being used and the data/parity chunks get nicely distributed amongst them. For details refer to the technical_whitepaper.

In the simplest one, the simple goal setup, you specify how many copies of every chunk of a file or directory will be copied to how many and optionaly also "which" chunkservers.

Note that the write modus here is: client writes chunk to ONE chunkserver and this chunkserver replicates this chunk to the other chunkservers.

The second replication mode is called XOR. It is similar to a classic RAID5 scenario in that you have N data chunks and 1 parity chunk. The maximum number of data chunks in this case is 9.

The third and most advanced mode of replication is the EC mode which does advanced erasure coding with a configurable amount of data and parity copies. In this mode clients wirte quasi parallel to the chunkservers. The maximum number of data chunks as well as of parity chunks is 32.

Note: To ensure proper repair procedure in case of a broken chunkserver, make sure to always have one chunkserver more than your configured goals/XOR/EC requires.

Now that you know what they are, lets start configuring them. . . .

4.3.1 Standard Goals

Goals configuration is defined in the file mfsgoals.cfg used by the master server.

Syntax of the mfsgoals.cfg file is:

```
id name : label ...
```

The # character starts comments.

There can be up to 20 replication goals configured, with ids between 1 and 20 inclusive. Each file stored on the filesystem refers to some goal id and is replicated according to the goal currently associated with this id.

By default, goal 1 means one copy on any chunkserver, goal 2 means two copies on any two chunkservers and so on. The purpose of mfsgoals.cfg is to override this behavior, when desired. The file is a list of goal definitions, each consisting of an id, a name and a list of labels.

Id indicates the goal id to be redefined. If some files are already assigned this goal id, their effective goal will change.

Name is a human readable name used by the user interface tools (mfssetgoal, mfsgetgoal). A name can consist of up to 32 alphanumeric characters: a-z, A-Z, 0-9, _.

list of labels is a list of chunkserver labels as defined in the mfschunkserver.cfg file. A label can consist of up to 32 alphanumeric characters: a-z, A-Z, 0-9, _. For each file using this goal and for each label, the system will try to maintain a copy of the file on some chunkserver with this label. One label may occur multiple times - in such case the system will create one copy per each occurence. The special label _ means "a copy on any chunkserver".

Note that changing the definition of a goal in mfsgoals.cfg affects all files which currently use a given goal id.

Examples

Example of goal definitions:

```
3 3 : _ _ _ # one of the default goals (three copies anywhere)
8 not_important_file : _ # only one copy
11 important_file : _ _
12 local_copy_on_mars : mars _ # at least one copy in the Martian datacenter
13 cached_on_ssd : ssd _
14 very_important_file : _ _ _ _
```

For further information see:

```
$ man mfsgoals.cfg
```

Show current goal configuration

From the command line:

```
$ lizardfs-admin list-goals <master ip> <master port>
```

Via cgi (webinterface):

In the 'Config' tab there is a table called 'Goal definitions'.

Set and show the goal of a file/directory

set The goal of a file/directory can be set using:

```
$ lizardfs setgoal goal_name object
```

which will result in setting the goal of an object to goal_name.

By default all new files created in the directory are created with the goal of the directory.

Additional options are:

- (-r) - change goals recursively for all objects contained in a directory
- goal_name[+|-] - when goal_name is appended with +, the goal is changed only if it will "increase security" (i.e. goal_name id is higher than id of the current goal)

show Current goal of the file/directory can be shown using:

```
$ lizardfs getgoal object
```

The result is the name of the currently set goal.

To list the goals in a directory use:

```
$ lizardfs getgoal -r directory
```

which for every given directory additionally prints the current value of all contained objects (files and directories).

To show where exactly file chunks are located use:

```
$ lizardfs fileinfo file
```

For further information see: *lizardfs-getgoal(1) lizardfs-fileinfo(1)*

4.3.2 Setting up XOR

Setting XOR goals works in a similar way as setting standard goals. Instead of just defining *id name : list of labels*, you add between the *:* and the *list of labels* the xor definition ($xor2 for 2+1, $xor3 for 3+1 . . .) and surround the labels with {}. If you do not specify an labels all chunkservers known to the system will be used. Each set will be of the size specified by your XOR definition but writes will be spread over all chunkservers.

For example if you have 5 chunkservers and define $xor2 (2 data and 1 parity chunk per set), the first set could write to chunkserver 1,2 and 3, the second one to chunkserver 2,3,4 and the next to chunkserver 1,3,5. The maximum number of data chunks is currently 9.

Example

This goes into *mfsgoals.cfg(5)* and sets custom goals 15,16 and 17:

```
15 default_xor3 : $xor3 # simple xor with 3 data and 1 parity chunks on all
                        # defined chunkservers. Each set will be written to
                        # 4 chunkservers but sets will be load balanced so
                        # all participating chunkservers will be utilized.

16 fast_read : $xor2 { ssd ssd hdd } # simple xor with 2 data and 1 parity
                                     # utilizing only 2 chunkservers, ssd
                                     # and hdd and writing 2 copies to the
                                     # ssd chunkservers and one to the hdd
                                     # chunkserver.

17 xor5 : $xor5 { hdd _ }            # XOR with 5 data and 1 parity drive
                                     # and at least one part of each set
                                     # written to the chunkserver labeled hdd
```

Applying, showing and listing goals works the same way as for *Standard Goals*.

4.3.3 Setting up EC

EC goals are set nearly the same way as XOR goals. The only difference is that you specify an ec definition in the format *$ecM,K* , where M stands for the data parts and K for the parity parts instead of the xor definition. The maximum size for each of those (M or K) is 32.

Example

This goes into *mfsgoals.cfg(5)* and sets custom goals 18, 19 and 20:

```
18 first_ec : $ec(3,1)
# Use ec3+1 on all chunkservers. Load will be balanced and each write will
# consist of 3 data and 1 parity part

19 ec32_ssd : $ec(3,2) { ssd ssd ssd ssd ssd }
# Use ec3+2 but write only to chunkservers labeled ssd. Each write will
# consist of 3 data and 2 parity parts.

20 ec53_mixed : $ec(5,3) { hdd ssd hdd _ _ _ _ _ }
# Use ec5+3 on all chunkservers but always write 2 parts to chunkservers
# labeled hdd and 1 part to chunkservers labeled ssd. The remaining parts go
# to any chunkserver. Each write will have 5 data and 3 parity parts. Load
# will be distributed amongst all chunkservers except for 2 parts which will
# be written to chunkservers labeled hdd and 1 part which will be written to
# chunkserver ssd on every write.
```

As you can see EC is extremely flexible and is also our fastest replication mode since writes from the client get spread over the chunkservers according to the goals set.

See also:

- *mfsgoals.cfg(5)*
- *lizardfs-getgoal(1)*
- *lizardfs(1)*
- *lizardfs-admin(8)*
- *Labeling your chunkserver*

4.4 Advanced configuration

4.4.1 Configuring chunkserver load awareness

In a default setup, your chunks will be tried to be evenly distributed according to goals and ec configurations among the chunkservers within a defined label group (check here *Labeling your chunkserver* for labels). While this is the perfect setting for most cases, in some cases, for example if your chunkservers are doing other, non lizardFS related workloads, you will want to have the distribution be also based on the amount of I/O load each chunkserver is handling in a given moment.

This functionality can be achieved by setting the

```
ENABLE_LOAD_FACTOR
```

setting in the chunkserver's mfschunkserver.cfg configuration file. By default this is disabled and the setting is a per chunkserver setting.

Additionaly it is possible to define a **penalty** configuration option in the master configuration file (The setting for this is **LOAD_FACTOR_PENALTY**), if he is found to be under heavy I/O load.

4.4.2 Configuring rack awareness (network topology)

The topology of a LizardFS network can be defined in the mfstopology.cfg file. This configuration file consists of lines matching the following syntax:

```
ADDRESS SWITCH-NUMBER
```

ADDRESS can be represented as:

*	all addresses
n.n.n.n	single IP address
n.n.n.n/b	IP class specified by network address and bits number
n.n.n.n/m.m.m.m	IP class specified by network address and mask
f.f.f.f-t.t.t.t	IP range specified by from-to addresses (inclusive)

The switch number can be specified as a positive 32-bit integer.

Distances calculated from mfstopology.cfg are used to sort chunkservers during read/write operations. Chunkservers closer (with lower distance) to a client will be favoured over further away ones.

Please note that new chunks are still created at random to ensure their equal distribution. Rebalancing procedures ignore topology configuration as well.

As for now, distance between switches can be set to 0, 1, 2:

0 - IP addresses are the same

1 - IP addresses differ, but switch numbers are the same

2 - switch numbers differ

The topology feature works well with chunkserver labeling - a combination of the two can be used to make sure that clients read to/write from chunkservers best suited for them (e.g. from the same network switch).

4.4.3 Configuring QoS

Quality of service can be configured in the /etc/mfs/globaliolimits.cfg file.

Configuration options consist of:

- subsystem <subsystem> cgroups subsystem by which clients are classified
- limit <group> <throughput in KiB/s>
- limit for clients in cgroup <group>
- limit unclassified <throughput in KiB/s>
- limit for clients that do not match to any specified group.

If globaliolimits.cfg is not empty and this option is not set, not specifying limit unclassified will prevent unclassified clients from performing I/O on LizardFS

Example 1:

```
# All client share 1MiB/s bandwidth
     limit unclassified 1024
```

Example 2:

```
# All clients in blkio/a group are limited to 1MiB/s, other clients are blocked
     subsystem blkio
     limit /a 1024
```

Example 3:

```
# The directory /a in the blkio group is allowed to transfer 1MiB/s
# /b/a group gets 2MiB/s
# unclassified  clients share 256KiB/s of bandwidth.
     subsystem blkio
     limit unclassified 256
     limit /a   1024
     limit /b/a 2048
```

4.4.4 Configuring Quotas

Quota mechanism can be used to limit inodes usage and space usage for users and groups. By default quotas can be set only by a superuser. Setting the SESFLAG_ALLCANCHANGEQUOTA flag in the mfsexports.cfg file would allow everybody to change quota.

In order to set quota for a certain user/group you can simply use mfssetquota tool:

```
mfssetquota  (-u UID/-g GID)   SL_SIZE   HL_SIZE   SL_INODES   HL_INODES   MOUNTPOINT_
↪PATH
```

where:

- SL - soft limit
- HL - hard limit

4.4.5 Mounting the metadata

LizardFS metadata can be managed through a special mountpoint called META. META allows to control trashed items (undelete/delete them permanently) and view files that are already deleted but still held open by clients.

To be able to mount metadata you need to add the "mfsmeta" parameter to the mfsmount command:

```
# mfsmount /mnt/lizardfs-meta -o mfsmeta
```

after that you will see the following line at mtab:

```
mfsmeta#10.32.20.41:9321 on /mnt/lizardfs-meta type fuse (rw,nosuid,nodev,relatime,
↪user_id=0,group_id=0,default_permissions,allow_other)
```

The structure of the mounted metadata directory will look like this:

```
/mnt/lizardfs-meta/
                   ├── reserved
                   └── trash
                   └── undel
```

Trash directory

Each file with a trashtime higher than zero will be present here. You can recover those files or delete files permanently.

Recovering files from the trash

In order to recover a file, just must move it to the undel/ directory. Files are represented by their inode number and path, so the file dir1/dir2/file.txt with inode 5 will be present at trash/5|dir1|dir2|file.txt, recovering it would be performed like this:

```
$ cd trash
$ mv '5|dir1|dir2|file.txt' undel/
```

Removing files permanently

In order to delete a file permanently, just remove it from trash.

Reserved directory

If you delete a file, but someone else use this file and keep an open descriptor, you will see this file in here until descriptor is closed.

4.4.6 Deploying LizardFS as a HA Cluster

LizardFS can be run as a high-availability cluster on several nodes. When working in HA mode, a dedicated daemon watches the status of the metadata servers and performs a failover whenever it detects a master server crashed (e.g. due to power outage). The state of the available participating servers is constantly monitored via a lightweight protocol doing a 'heartbeat' like check on the other nodes. Running LizardFS installation as a HA-cluster significantly increases its availability. Since uRaft uses *quorum* a reasonable minimum of metadata servers in a HA installation is at least 3, to make sure that a proper election with a 'majority' of voices can be done. For details on the underlying algorythm, check *raft* in the glossary.

In order to deploy LizardFS as a high-availability cluster, follow the steps below.

These steps should be performed on all machines chosen to be in a cluster.

Install the lizardfs-uraft package:

```
$ apt-get install lizardfs-uraft for Debian/Ubuntu
$ yum install lizardfs-uraft for CentOS/RedHat
```

Prepare your installation:

Fill lizardfs-master config file (/etc/mfs/mfsmaster.cfg) according to *Configuring your Master*. Details depend on your personal configuration, the only fields essential for uraft are:

```
PERSONALITY = ha-cluster-managed
ADMIN_PASSWORD = your-lizardfs-password
MASTER_HOST = the floating ip so that the participating hosts know where to sync the␣
↪metadatabase from
```

For a fresh installation, execute the standard steps for the lizardfs-master (creating mfsexports file, empty metadata file etc.). Do not start the lizardfs-master daemon yet.

Fill the lizardfs-uraft config file (/etc/mfs/lizardfs-uraft.cfg). Configurable fields are:

URAFT_NODE_ADDRESS identifiers of all the machines in your cluster

URAFT_ID node address ordinal number; should be unique for each machine

URAFT_FLOATING_IP IP at which LizardFS will be accessible for the clients

URAFT_FLOATING_NETMASK a matching netmask for floating IP

URAFT_FLOATING_IFACE network interface for the floating IP

URAFT_ELECTOR_MODE ...

LOCAL_MASTER_ADDRESS The address of the local master controlled by this uraft node, defaults to localhost.

LOCAL_MASTER_MATOCL_PORT The port the local master listens on, defaults to 9421

ELECTION_TIMEOUT_MIN Minimum election timeout (ms), defaults to 400

ELECTION_TIMEOUT_MAX Maximum election timeout (ms), defaults to 600

HEARTBEAT_PERIOD = 20 Period between hearbeat messages between uraft nodes (ms), defaults to 20.

LOCAL_MASTER_CHECK_PERIOD How often uRaft checks if local master is alive (ms), defaults to 250.

Example configuration for a cluster with 3 machines:

The first, node1, is at 192.168.0.1, the second node gets hostname node2, and the third one gets hostname node3 and operates under a non-default port number - 99427.

All machines are inside a network with a 255.255.255.0 netmask and use their network interface eth1 for the floating ip.

The LizardFS installation will be accessible at 192.168.0.100

```
 # Configuration for node1:
 URAFT_NODE_ADDRESS = 192.168.0.1          # ip of first node
 URAFT_NODE_ADDRESS = node2                # hostname of second node
 URAFT_NODE_ADDRESS = node3:99427          # hostname and custom port of third node
 URAFT_ID = 0                              # URAFT_ID for this node
 URAFT_FLOATING_IP = 192.168.0.100         # Shared (floating) ip adddress for this␣
↪cluster
 URAFT_FLOATING_NETMASK = 255.255.255.0    # Netmask for the floating ip
 URAFT_FLOATING_IFACE = eth1               # Network interface for the floating ip␣
↪on this node

# Configuration for node2:
 URAFT_NODE_ADDRESS = 192.168.0.1          # ip of first node
 URAFT_NODE_ADDRESS = node2                # hostname of second node
 URAFT_NODE_ADDRESS = node3:99427          # hostname and custom port of third node
 URAFT_ID = 1                              # URAFT_ID for this node
 URAFT_FLOATING_IP = 192.168.0.100         # Shared (floating) ip adddress for this␣
↪cluster
 URAFT_FLOATING_NETMASK = 255.255.255.0    # Netmask for the floating ip
 URAFT_FLOATING_IFACE = eth1               # Network interface for the floating ip␣
↪on this node

 # Configuration for node3:
 URAFT_NODE_ADDRESS = 192.168.0.1          # ip of first node
 URAFT_NODE_ADDRESS = node2                # hostname of second node
 URAFT_NODE_ADDRESS = node3:99427          # hostname and custom port of third node
 URAFT_ID = 2                              # URAFT_ID for this node
 URAFT_FLOATING_IP = 192.168.0.100         # Shared (floating) ip adddress for this␣
↪cluster
 URAFT_FLOATING_NETMASK = 255.255.255.0    # Netmask for the floating ip
 URAFT_FLOATING_IFACE = eth1               # Network interface for the floating ip␣
↪on this node
```

(continues on next page)

(continued from previous page)

```
```

Enable arp broadcasting in your system (for the floating IP to work):

```
$ echo 1 > /proc/sys/net/ipv4/conf/all/arp_accept
```

Start the lizardfs-uraft service:

Change "false" to "true" in /etc/default/lizardfs-uraft:

```
$ service lizardfs-uraft start
```

You can check your uraft status via telnet on URAFT_STATUS_PORT (default: 9428):

```
$ telnet NODE-ADDRESS 9428
```

When running telnet locally on a node, it is sufficient to use:

```
$ telnet localhost 9428
```

Please check if you have the sudo package installed and that the 'mfs' user has been added with the right permissions to the /etc/sudoers file.

4.5 Configuring Optional Features

LizardFS comes with a range of optional features whose configuration does not belong into the "basic" or "advanced" modes.

4.5.1 Configuring LTO handling

Installation

THe LizardFS tapeserver package can be installed via:

```
$ apt-get install lizardfs-tapeserver # Debian/Ubuntu
$ yum install lizardfs-tapeserver # CentOS/RedHat
```

Configuration

The configuration file for the lizardfs-tapeserver is located at /etc/mfs/lizardfs-tapeserver.cfg. The tapeserver needs a working mountpoint of your LizardFS installation. Configuration consists mainly of listing changer and volume devices of a tape library.

Example configuration:

```
[lizardfs-tapeserver]
# Name
name = tapesrv
# Master host network info
masterhost = 192.168.1.5
masterport = 9424
# Path to mountpoint used for file reading/writing
```

(continues on next page)

(continued from previous page)

```
mountpoint = /mnt/tapemount
# Path to temporary cache for files
cachepath  = /tmp/cache
# Path to sqlite3 database
database = /opt/tape.db
# Changer devices
changers = /dev/sg3
# Drive devices
drives = /dev/st0,/dev/st1
# Volume ids
volumes = 000003,000180,000002
# Label
label = tapeserver1
```

Verifying your installation

Installation can be easily verified using the lizardfs-admin command:

```
$ lizardfs-admin list-tapeserver MASTER_ADDR MASTER_PORT
```

If the installation succeeded, the command above should result in listing all tapeservers connected to the current master.

Verification if tape storage works properly can be achieved by the steps below:

- create a test file
- set tape goal to the file: mfssetgoal your_tape_goal testfile
- wait for replication to take place, check its status with 'mfsfileinfo' command:

  ```
  $ mfsfileinfo testfile
  ```

- Replication to tape is complete after tape copy status changes from Creating to Ok
- verify that the file was actually stored on tape:

  ```
  $ tar tf /dev/your_tape_volume # will list all files present on tape
  $ tar xvf /dev/your_tape_volume filename # will extract file 'filename' from tape
  ```

Configuring tape goals

Tape goals are configured just like regular goals, save one difference in naming. In order to create a tape goal, append a "@" sign to the end of its definition.

Example mfsgoals.cfg contents:

```
1 : _
2 : _ _
tape1 : _ _@
tape2: ssd _@ ts@
fast: ssd ssd _
```

The example above contains 2 tape goal definitions.

The first one (tape1), configures that there should be 2 copies of each chunk:

- 1 on any chunkserver

- 1 on any tape.

The second one (tape2) requires each chunk to have 3 copies:

- 1 on chunkserver labeled "ssd"
- 1 on any tape
- 1 on tape labeled "ts"

4.6 Connecting Clients to your LizardFS installation

The most exciting part of this tutorial - you will finally be able to store files on your installation!

4.6.1 Linux / *NIX / *BSD / MacOS/X client

Install the client package

Check *Getting and installing LizardFS* for instructions how to install package

Example for Debian/Ubuntu:

```
$ apt-get install lizardfs-client
```

Make sure that the mfsmaster host is set in your /etc/hosts file. For our example configuration it would be:

```
192.168.16.100 mfsmaster
```

Create a mountpoint:

```
$ mkdir /mnt/lizardfs
```

Mount the filesystem to the mountpoint with just the default options from mfsmount.cfg if any:

```
$ mfsmount /mnt/lizardfs
```

That's it. Simple, straight and easy.a

Optional settings for performance on *NIX

big_writes

On most systems adding big_writes to the options will significantly increase your throughput since it will force the fuse libraray to use writes > 4k.

Example:

```
$ mfsmount -o big_writes,nosuid,nodev,noatime /mnt/lizardfs
```

will mount with fuse option: big_writes and default system mount options: nosuid, nodev and noatime.

Read ahead cache

If you want to make use of the read ahead caching feature which will sig- nificantly improve your read performance, you will require to configure the following options:

```
-o cacheexpirationtime=MSEC       set timeout for read cache entries to be considered␣
↪valid in milliseconds (0 disables cache) (default: 0)
-o readaheadmaxwindowsize=KB      set max value of readahead window per single␣
↪descriptor in kibibytes (default:
```

Example:

```
mfsmount -o cacheexpirationtime=500 -o readaheadmaxwindowsize=4096 mountpoint/
```

Reasonable values:

```
* cacheexpirationtime - depends on latency, 500 should be alright for most␣
↪installations. Higher values = data will be kept in cache longer, but it will also␣
↪occupy more RAM.
* readaheadmaxwindowsize - depends on latency and cacheexpirationtime, 1024-8192 are␣
↪usually fine. Higher values = bigger portions of data asked in single request.
```

readaheadmaxwindowsize can be adjusted to the installation - starting with 1024 and increasing it until tests show no performance gain is a good idea.

You can now store your files on your brand new installation.

See the *mfsmount(1)* and *mfsmount.cfg(5)* manpage for more options

See *FUSE* to find out more about the fuse library.

4.6.2 Windows™

System Settings for LizardFS clients

Recommended tuning for Windows™ Clients:

- Make sure your Power Options are set to “High Performance”.
- Increase the number of Transfer and Receive Buffers in the configuration of your network interface
- Increase the number of Queues available in your network interface setting
- If at all possible set all network devices in your network to use Jumbo Frames.

The precise number for your respective host / NIC will be different in every setup, but they for sure will be far higher than the default settings.

Windows™ GUI client

Install our client from exe package provided

Add your credentials and the address and port of the master server.

Select the drive you want your lizardFS filesystem to appear as in your windows session.

It should look like in the following image:

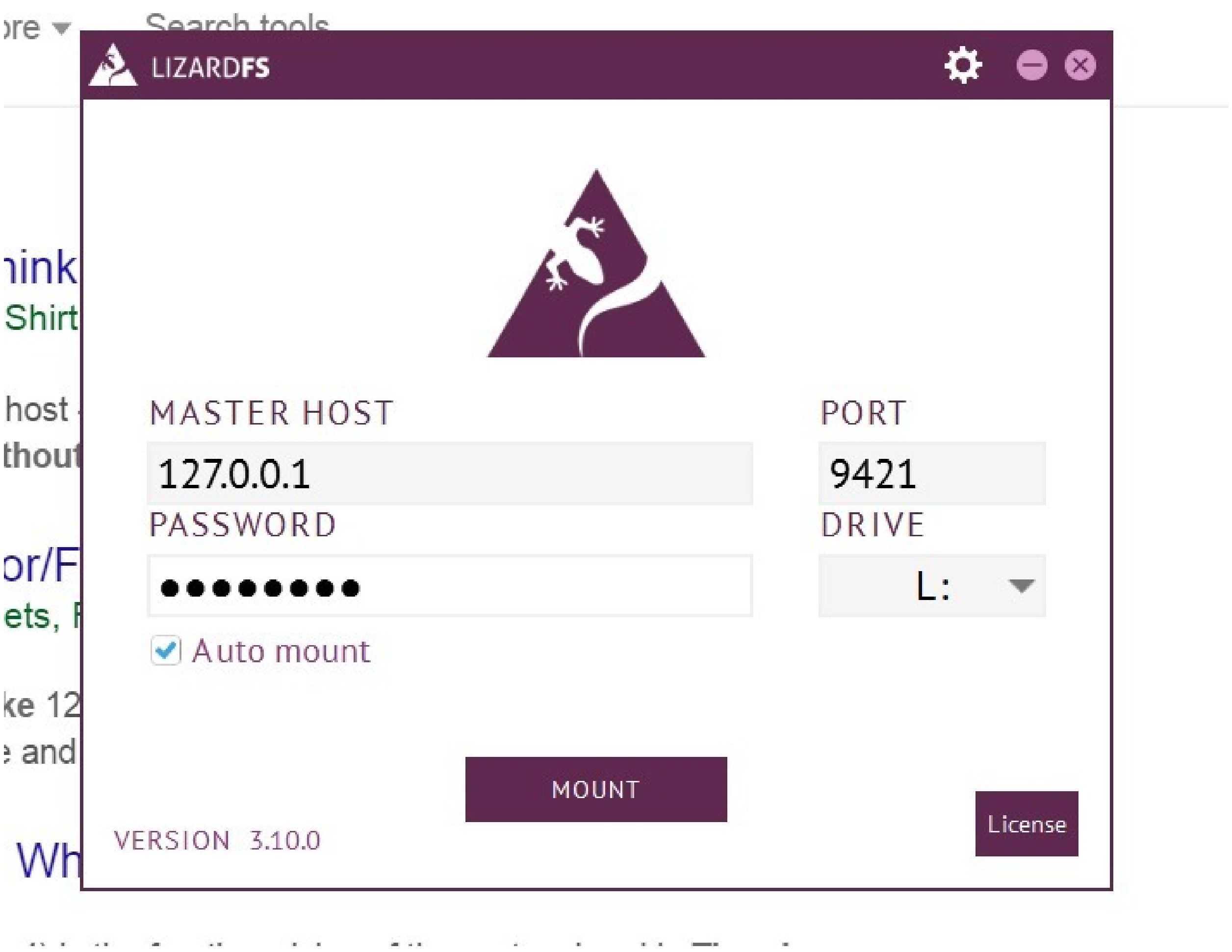

Figure 2: main view of LizardFS Windows™ client

Windows™ CLI Client

Together with the GUI client the installation package adds a CLI client to your Windows™ system. It is located in:

```
C:\Program Files\LizardFS\lizardfscli.exe
```

Options are:

-H	The address of the Master
-P	The port to use at the Master
-D	The drive letter to use for the mounted FS

Windows™ service

The Windows™ Client can also be run as a Windows™ Service. This is provided by the **LizardSControler** command.

Basic configuration

Minimal configuration:

```
LizardSControler -p -lic-file <LICENSE_FILE> -H <ADDRESS_OF_MASTER>
```

where LICENSE_FILE should be the name of the file containing a valid License and ADDRESS_OF_MASTER should be the hostname or IP address of the LizardFS master server.

Further configuration options

(Must follow the -p command)

Command	Description
-H HOST	set master server host address.
-P PORT	set master server port. Default 9421.
-D DRIVE	set <DRIVE> as a mount point i.e. D:. Default L:
-f SUBFOLDER	mount only given LizardFS subfolder
-uid UID	set new UID. Default is 1000.
-gid GID	set new GID. Default is 1000.
-umask UMASK	set new UMASK. Default is 000.
-pass PASS	authenticate to LizardFS master using MD5 password.
-lic LICENSE	set new LICENSE.
-lic-file LICENSE_FILE	load new LICENSE from LICENSE_FILE.

Installation and runtime

After you have done the configration, you can add the service to your Windows system by running:

```
LizardSControler -i
```

and start it by running:

```
LizardSControler -s
```

If you would like to uninstall the service again, just run:

```
LizardSControler -u
```

To terminate the service, run:

```
LizardSControler -t
```

A full list of options can be displayed using:

```
LizardSControler -help
```

4.7 Securing your LizardFS installation

4.8 Optimizing your LizardFS setup

4.9 Troubleshooting

4.9.1 Linux

Basic checks

- Check if your nodes are all reachable by ip address as well as by hostname
- Check if your network has the right throughput
- Check if your disks are all working and do not repport errors
- Check your Chunkservers for:
  - Broken Disks
  - Slow Disks
  - permissions on your directories (the user which is running the chunkserver must be the owner of the directories)
  - network performance to other chunkserver
  - network performance to master server
  - network performance to clients
- Check your license files for the correct name and location
- Check your log files for errors. Lizardfs is very talkative and reports a lot.

Check the speed of your network interface

Veryfying what your network interface is set to is pretty simple on linux, you can just use the *ethtool* program to tell you what your interface is actualy set to:

```
ethtool <interface>
```

example:

```
# ethtool eth0
  Settings for eth0:
      Supported ports: [ TP ]
      Supported link modes:   10baseT/Half 10baseT/Full
                              100baseT/Half 100baseT/Full
                              1000baseT/Full
      Supported pause frame use: No
      Supports auto-negotiation: Yes
      Advertised link modes:  10baseT/Half 10baseT/Full
                              100baseT/Half 100baseT/Full
                              1000baseT/Full
      Advertised pause frame use: No
      Advertised auto-negotiation: Yes
```

(continues on next page)

(continued from previous page)

```
      Speed: 100Mb/s
      Duplex: Full
      Port: Twisted Pair
      PHYAD: 1
      Transceiver: internal
      Auto-negotiation: on
      MDI-X: on (auto)
      Supports Wake-on: pumbg
      Wake-on: g
      Current message level: 0x00000007 (7)
                             drv probe link
      Link detected: yes
```

As you can see, the interface report a speed of 100Mb/s. It has Auto-negotiation enabled so is most probably connected to a 10Mb/s switchport.

Checking the throughput of your network

The best tool to verify if your network throughput is according to what you think it is would be the *iperf* tool. *iperf* allows you to verify the throughput between two machines. It is available for linux, windows, macOS, FreeBSD and all other posix compliant systems and most mobile phone OSes and is very easy to use.

For more information about iperf, please check out https://iperf.fr/ .

4.10 Upgrading Lizardfs

4.10.1 Upgrading to LizardFS 3.10.0

Direct upgrade is possible from LizardFS 2.6.0 - 3.9.4.

- To upgrade from LizardFS 2.6.0, see upgrade notes for LizardFS 3.9.2
- Upgrade shadow master servers before the master server; shadow master will always refuse to connect to the master server in any newer version.
- Upgrade the master/shadow servers before any chunkserver or client.
- Chunkservers should be updated before clients, as they can still understand legacy mount requests. Updating a client first will result in being unable to read from/write to a 2.6.0 chunkserver.
- Neither master servers nor chunkservers have hard coded connection limits anymore.

The previous value were 5000 for master server and 10000 for chunkserver.

It is advisable to check if system limits are high enough for proper work of LizardFS cluster.

Mentioned values can be changed by modifying the /etc/security/ limits.d/10-lizardfs.conf file or by using ulimit command.

4.10.2 Upgrading to LizardFS 3.9.4

Direct upgrade is possible from LizardFS 3.9.2 and LizardFS 2.6.0.

- 3.9.2 chunkservers need to be upgraded ASAP due to protocol changes
- See upgrade notes for LizardFS 3.9.2

4.10.3 Upgrading to LizardFS 3.9.2

Direct upgrade is possible from LizardFS 2.6.0.

- Upgrade shadow master servers before the master server; shadow master will always refuse to connect to the master server in any newer version.

Attention:

It is recommended to perform metadata save before upgrading shadow servers and refrain from using mfsmakesnapshot tool before upgrading master.

Not complying to this rule might cause potential checksum mismatches in shadows.

Metadata save can be triggered by the admin tool:

```
$ lizardfs-admin save-metadata HOST PORT
```

- Upgrade the master/shadow servers before any chunkserver or client.
- Chunkservers should be updated before clients, as they can still understand legacy mount requests. Updating a client first will result in being unable to read from/write to a 2.6.0 chunkserver.
- Neither master servers nor chunkservers have hard coded connection limits anymore.

The previous values were 5000 for the master server and 10000 for the chunkserver.

It is advisable to check if system limits are high enough for proper work of LizardFS cluster.

Mentioned values can be changed by modifying

```
/etc/security/limits.d/10-lizardfs.conf
```

or by using the ulimit command.

4.10.4 Upgrading to LizardFS 2.6.0

Direct upgrade is possible from:

- MooseFS 1.6.27 – 1.6.27-5
- LizardFS 1.6.28 – 2.5.4.3

When upgrading from LizardFS 2.5.4 or 2.5.4.*

- Upgrade shadow master servers before the master server; shadow master will always refuse to connect to the master server in any newer version.
- Upgrade the master server before any chunkserver or client.

When upgrading from LizardFS 2.5.0 or 2.5.2

- Proceed the same as in case of upgrade to LizardFS 2.5.4.

When upgrading from other releases:

- Proceed the same as in case of upgrade to LizardFS 2.5.2.

4.10.5 Upgrading to LizardFS 2.5.4

Direct upgrade is possible from:

- MooseFS 1.6.27 – 1.6.27-5
- LizardFS 1.6.28 – 2.5.2

When upgrading from LizardFS 2.5.0 or 2.5.2:

- Upgrade shadow master servers before the master server; shadow master will always refuse to connect to the master server in any newer version.
- Upgrade the master server before any chunkserver or client.
- Clients, chunkservers and the master server can be upgraded in any order (all of them are compatible with each other), either one by one or all at once.
- CGI server 2.5.4 is compatible with all versions of the master server (1.6.27 – 2.5.2), so it can be upgraded at any time. Because of a bug in older versions of the CGI serve it has to be stopped manually before being upgraded (otherwise installation of the newer package may fail).

When upgrading from other releases:

- Proceed the same as in case of upgrade to LizardFS 2.5.2.

4.10.6 Upgrading to LizardFS 2.5.2

Direct upgrade is possible from:

- MooseFS 1.6.27 – 1.6.27-5
- LizardFS 1.6.28 – 2.5.0

When upgrading from LizardFS 2.5.0:

- Upgrade shadow master servers before the master server; shadow master will always refuse to connect to the master server in any newer version.
- Upgrade the master server
- Clients, chunkservers and the master server can be upgraded in any (all of them are compatible with each other), either one by one or all at once.

When upgrading from previous releases:

- There is no possibility to use shadow masters 2.5.2 before upgrading the master server.
- Upgrade the master server first, then all the chunkservers and then clients (either all at once or one by one).

4.10.7 Upgrading to LizardFS 2.5.0

Direct upgrade is possible from:

- MooseFS 1.6.27 – 1.6.27-5
- LizardFS 1.6.28
- Master server has to be upgraded before any client.
- Chunkservers can be upgraded in any order (before the master server is upgraded or after).
- Shadow master server 2.5.0 is compatible with master server 1.6.27 and newer.

4.10.8 Upgrading to LizardFS 1.6.28

Direct upgrade is possible from:

- MooseFS 1.6.27 – 1.6.27-5.

All the servers and clients are compatible and can be upgraded in any order.

4.11 LizardFS Manual Pages

These Manual Pages get installed automatically on installation of lizardfs and can be also read in your system using:

```
man <commandname>
```

on your system.

Contents:

4.11.1 lizardfs(7)

NAME

lizardfs - a networking, distributed, highly available file system

DESCRIPTION

LizardFS is a networking, highly available, distributed file system. It spreads data over several physical localisations (servers), which are visible to a user as one resource. For standard file operations LizardFS acts as other Unix-alike file systems. It has hierarchical structure (directory tree), stores files' attributes (permissions, last access and modification times) as well as making it possible to create special files (block and character devices, pipes and sockets), symbolic links (file names pointing to another files accessible locally, not necessarily on LizardFS) and hard links (different names of files which refer to the same data on LizardFS). Access to the file system can be limited based on IP address and/or password.

Distinctive features of LizardFS are:

- higher reliability (data can be stored in several copies on separate computers)
- dynamically expanding disk space by attaching new computers/disks
- possibility of storing deleted files for a defined period of time ("trash bin" service on a file system level)
- possibility of creating snapshot of a file, which means a coherent copy of the whole file, even while the file is being written to.

ARCHITECTURE

A LizardFS installation consists of five types of machines:

master metadata server (or 'the master') a managing server - single computer managing the whole filesystem, storing metadata for every file (information on size, attributes and file localisation(s), including all information about non-regular files, i.e. directories, sockets, pipes and devices.

metadata server shadows (or 'the shadow master') almost identical to the master, there can be any number of those, they work as master metadata server backup and they are ready for immediate deployment as the new master in case of current master failure.

data servers (or 'the chunkservers') any number of commodity servers storing files data and replicating it among themselves (if a certain file is supposed to exist in more than one copy.

metadata backup servers (or 'the metaloggers') any number of servers, all of which store metadata changelogs and periodically downloading base metadata file; it's easy to run the mfsmaster process on such a machine if the primary master stops working.

client computers referring to LizardFS stored files (or 'the clients') any number of machines with working mfsmount process that communicates with the managing server to receive and modify file information and with the chunkservers to exchange actual file data.

Metadata is stored in the memory of the managing server and is simultaneously being saved to disk (as a periodically updated binary file and immediately updated incremental logs). The main binary file as well as the logs are replicated to the metaloggers (if present).

File data is divided into fragments (chunks) of a maximum size of 64MB each which are stored as files on selected disks on the data servers (chunkservers). Each chunk is saved on different computers in a number of copies equal to a "goal" for the given file.

REPORTING BUGS

Report bugs to <contact@lizardfs.org>.

COPYRIGHT

2013-2017 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

mfschunkserver(8), mfsmaster(8), mfsmetalogger(8), mfsmount(1), lizardfs(1), lizardfs-admin(8)

4.11.2 lizardfs-admin(8)

NAME

lizardfs-admin, lizardfs-probe - LizardFS monitoring and administration tool

SYNOPSIS

```
lizardfs-admin  COMMAND [OPTIONS...] [ARGUMENTS...]
```

Available COMMANDs

chunks-health <master ip> <master port> Returns chunks health reports in the installation. By default (if no report is specified) all reports will be shown. In replication and deletion states, the column means the number of chunks with number of copies specified in the label to replicate/delete.

Possible command-line options:

--availability	Print report about availability of chunks.
--replication	Print report about about number of chunks that need replication.
--deletion	Print report about about number of chunks that need deletion.

info <master ip> <master port> Prints statistics concerning the LizardFS installation.

iolimits-status <master ip> <master port> Prints current configuration of global I/O limiting

list-chunkservers <master ip> <master port> Prints information about all connected chunkservers.

list-defective-files <master ip> <master port> Prints out paths of files which have unavailable or undergoal chunks

Possible command-line options:

--unavailable	Print unavailable files
--undergoal	Print files with undergoal chunks
--limit	Limit maximum number of printed files

list-disks <master ip> <master port> Prints information about all connected chunkservers.

Possible command-line options:

--verbose	Be a little more verbose and show operations statistics.

list-goals <master ip> <master port> List goal definitions.

Possible command-line options:

--pretty	Print nice table

list-mounts <master ip> <master port> Prints information about all connected mounts.

Possible command-line options:

--verbose	Be a little more verbose and show goal and trash time limits.

metadataserver-status <master ip> <master port> Prints status of a master or shadow master server

list-metadataservers <master ip> <master port> Prints status of active metadata servers.

ready-chunkservers-count <master ip> <master port> Prints number of chunkservers ready to be written to.

promote-shadow <shadow ip> <shadow port> Promotes metadata server. Works only if personality 'ha-cluster-managed' is used. Authentication with the admin password is required.

stop-master-without-saving-metadata <master ip> <master port> Stop the master server without saving metadata in the metadata.mfs file. Used to quickly migrate a metadata server (works for all personalities). Authentication with the admin password is required.

reload-config <master ip> <master port> Requests reloading configuration from the config file. This is synchronous (waits for reload to finish). Authentication with the admin password is required.

save-metadata <metadataserver ip> <metadataserver port> Requests saving the current state of metadata into the metadata.mfs file. With –async fail if the process cannot be started, e.g. because the process is already in progress. Without –async, fails if either the process cannot be started or if it finishes with an error (i.e., no metadata file is created).

Authentication with the admin password is required.

Possible command-line options:

--async	Don't wait for the task to finish.

list-tasks <master ip> <master port> Lists tasks which are currently being executed on the master. The returned id's can be used by the stop-task command.

stop-task <master ip> <master port> <task id> Stop execution of the task with the given id

COMMON COMMAND OPTIONS

--porcelain	Make the output parsing-friendly.

REPORTING BUGS

Report bugs to <contact@lizardfs.org>.

COPYRIGHT

2015-2017 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

lizardfs(7)

4.11.3 mfsmetadump(8)

NAME

mfsmetadump - dump file system metadata into a specified file.

SYNOPSIS

```
mfsmetadump METADATA_FILE
```

DESCRIPTION

mfsmetadump creates a dump of the data contained in a masterserver and puts it into the file specified as an argument to it.

REPORTING BUGS

Report bugs to <contact@lizardfs.org>.

COPYRIGHT

Copyright 2013-2017 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

lizardfs(1)

4.11.4 mfsmetarestore(8)

NAME

mfsmetarestore - replay LizardFS metadata change logs or dump LizardFS metadata image

SYNOPSIS

```
mfsmetarestore [-z] -m OLDMETADATAFILE -o NEWMETADATAFILE [CHANGELOGFILE...]

mfsmetarestore -m METADATAFILE

mfsmetarestore [-z] -a [-d DIRECTORY]

mfsmetarestore* -g -d DIRECTORY

mfsmetarestore -v

mfsmetarestore -?
```

DESCRIPTION

When *mfsmetarestore* is called with both *-m* and *-o* options, it replays given *CHANGELOGFILEs* on *OLDMETADATAFILE* and writes result to *NEWMETADATAFILE*. Multiple change log files can be given.

mfsmetarestore with just the *-m METADATAFILE* option dumps LizardFS metadata image file in human readable form.

mfsmetarestore called with -a option automatically performs all operations needed to merge change log files. The master data directory can be specified using -d DIRECTORY option.

mfsmetarestore -g with path to metadata files, prints the latest metadata version that can be restored from disk.

Prints 0 if metadata files are corrupted.

-a autorestore mode (see above)

-d <DATAPATH> master data directory (for autorestore mode)

-m <METADATAFILE> specify input metadata image file

-o <NEWMETADATAFILE> specify output metadata image file

-z ignore metadata checksum inconsistency while applying changelogs

-v,-? print version information and exit

FILES

metadata.mfs Lizard File System metadata image as read by *mfsmaster* process

metadata.mfs.back Lizard File System metadata image as left by killed or crashed *mfsmaster* process

changelog.*.mfs Lizard File System metadata change logs

REPORTING BUGS

Report bugs to <contact@lizardfs.org>.

COPYRIGHT

Copyright 2008-2009 Gemius SA, 2013-2016 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

mfsmaster(8), lizardfs(7)

4.11.5 mfsrestoremaster(8)

NAME

mfsrestoremaster - a networking, distributed, highly available file system

SYNOPSIS

```
mfsrestoremaster <net-interface> [<etc-mfs-dir>]
```

<net-interface> - network interface to reconfigure.

<etc-mfs-dir> - mfs configuration directory to use (default: /etc/mfs).

DESCRIPTION

mfsrestoremaster automates starting a spare master server on a metalogger machine.

It performs the following steps:

- verify basic sanity of configuration files
- update metadata image with data from metalogger changelogs
- set master's IP address on given network interface
- start the master server

REPORTING BUGS

Report bugs to <contact@lizardfs.org>.

COPYRIGHT

Copyright 2008-2009 Gemius SA, 2013-2016 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

lizardfs(1), lizardfs(7)

4.11.6 lizardfs-appendchunks(1)

NAME

lizardfs-appendchunks - lazy append chunks

SYNOPSIS

```
lizardfs appendchunks 'SNAPSHOT_FILE' 'OBJECT'...
```

DESCRIPTION

lizardfs appendchunks (equivalent of mfssnapshot from MooseFS 1.5) appends a lazy copy of specified file(s) to the specified snapshot file ("lazy" means that creation of new chunks is delayed to the moment one copy is modified). If multiple files are given, they are merged into one target file in the way that each file begins at 'chunk' (64MB) boundary; padding space is left empty.

COPYRIGHT

2013-2017 Skytechnology sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

lizardfs(1),lizardfs(7)

4.11.7 lizardfs-checkfile(1)

NAME

lizardfs-checkfile - print information about chunks

SYNOPSIS

```
lizardfs checkfile 'FILE'...
```

DESCRIPTION

> **lizardfs checkfile** checks and prints the number of chunks and number of chunk copies belonging to the specified file(s). It can be used on any file, including deleted ones ('trash').

COPYRIGHT

2013-2017 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

lizardfs(1), lizardfs(7)

4.11.8 lizardfs-dirinfo(1)

NAME

lizardfs-dirinfo - show directories stats

SYNOPSIS

```
lizardfs dirinfo [-n|-h|-H] OBJECT...
```

DESCRIPTION

dirinfo is an extended, LizardFS-specific equivalent of the *du -s* command. It prints a summary for each specified object (single file or directory tree).

OPTIONS

-n, -h, -H	These options are described in lizardfs(1).

OPYRIGHT

2013-2017 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

lizardfs(1)

4.11.9 lizardfs-fileinfo(1)

NAME

lizardfs-fileinfo - locate chunks

SYNOPSIS

```
lizardfs fileinfo 'FILE'...
```

DESCRIPTION

fileinfo prints the location (‘chunkserver’ host and port) of each chunk copy belonging to specified file(s). It can be used on any file, included deleted ones (‘trash’).

COPYRIGHT

2013-2017 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

lizardfs(1), lizardfs(7)

4.11.10 lizardfs-filerepair(1)

NAME

lizardfs-filerepair - repair broken files

SYNOPSIS

```
lizardfs filerepair [-n|-h|-H] 'FILE'...
```

DESCRIPTION

filerepair deals with broken files (those which cause I/O errors on read operations) to make them partially readable. In case of a missing chunk it fills the missing parts of the file with zeros; in case of a chunk version mismatch it sets the chunk version known to *mfsmaster* to the highest one found on the chunkservers.

Note: Because in the second case content mismatch can occur in chunks with the same version, it is advised to make a copy (not a snapshot!) and delete the original file after "repairing".

OPTIONS

-c	This option enables correct-only mode, which will restore chunk to a previous version if possible, but will never erase any data.
-n, -h, -H	These options are described in lizardfs(1).

OPYRIGHT

2013-2017 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

lizardfs(1),lizardfs(7)

4.11.11 lizardfs-geteattr(1)

NAME

lizardfs-geteattr, lizardfs-seteattr, lizardfs-deleattr - get, set or delete extra attributes

SYNOPSIS

```
lizardfs geteattr [-r] [-n|-h|-H] 'OBJECT'...
lizardfs seteattr [-r] [-n|-h|-H] -f 'ATTRNAME' [-f 'ATTRNAME' ...] 'OBJECT'...
lizardfs deleattr [-r] [-n|-h|-H] -f 'ATTRNAME' [-f 'ATTRNAME' ...] 'OBJECT'...
```

DESCRIPTION

geteattr, **seteattr** and **deleattr** tools are used to get, set or delete some extra attributes. Attributes are described below.

OPTIONS

-r	Enables recursive mode.
-n, -h, -H	These options are described in lizardfs(1).

EXTRA ATTRIBUTES

noowner This flag means, that particular object belongs to current user ('uid' and 'gid' are equal to 'uid' and 'gid' values of accessing process). Only root ('uid'=0) sees the real 'uid' and 'gid'.

noattrcache This flag means, that standard file attributes such as uid, gid, mode, length and so on won't be stored in kernel cache.

noentrycache This flag is similar to above. It prevents directory entries from being cached in kernel.

OPYRIGHT

2013-2017 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

lizardfs(1),lizardfs(7)

4.11.12 lizardfs-getgoal(1)

NAME

lizardfs-getgoal, lizardfs-setgoal, lizardfs-rgetgoal, lizardfs-rsetgoal - change or retrieve goal

SYNOPSIS

```
lizardfs getgoal [-r] [-n|-h|-H] 'OBJECT'...

lizardfs rgetgoal [-n|-h|-H] 'OBJECT'...

lizardfs setgoal [-r] [-n|-h|-H] GOAL_NAME 'OBJECT'...

lizardfs rsetgoal [-n|-h|-H] GOAL_NAME 'OBJECT'...
```

DESCRIPTION

getgoal and **setgoal** operate on an object's 'goal' value.

getgoal prints current 'goal' value of given object(s). These tools can be used on any file, directory or deleted ('trash') file.

OPTIONS

-r This option enables recursive mode, which works as usual for every given file, but for every given directory additionally prints current value of all contained objects (files and directories).

-n, -h, -H	These options are described in lizardfs(1).

NOTE

rgetgoal and **rsetgoal** are deprecated aliases for *getgoal -r* and *setgoal -r* respectively.

COPYRIGHT

2013-2017 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

lizardfs(1),lizardfs(7)

4.11.13 lizardfs-gettrashtime(1)

NAME

lizardfs-gettrashtime, lizardfs-settrashtime, lizardfs-rgettrashtime, lizardfs-rsettrashtime - get or set trash time

SYNOPSIS

```
lizardfs gettrashtime [-r] [-n|-h|-H] 'OBJECT'...
lizardfs rgettrashtime* [-n|-h|-H] 'OBJECT'...
lizardfs settrashtime* [-r] [-n|-h|-H] SECONDS[+|-] 'OBJECT'...
lizardfs rsettrashtime* [-n|-h|-H] SECONDS[+|-] 'OBJECT'...
```

DESCRIPTION

gettrashtime and **settrashtime** operate on an object's 'trashtime' value, i.e. the number of seconds the file is preserved in the special 'trash' directory before it's finally removed from the filesystem. 'Trashtime' must be a non-negative integer value. *gettrashtime* prints the current 'trashtime' value of given object(s).

OPTIONS

-r	This option enables recursive mode, which works as usual for every given file, but for every given directory additionally prints the current 'trashtime' value of all contained objects (files and directories).
-n, -h, -H	These options are described in lizardfs(1).

N[+|-] If a new value is specified in 'N'+ form, the 'trashtime' value is increased to 'N' for objects with lower 'trashtime' value and unchanged for the rest. Similarly, if a new value is specified as 'N'-, the 'trashtime' value is decreased to 'N' for objects with higher 'trashtime' value and unchanged for the rest. These tools can be used on any file, directory or deleted ('trash') file.

Note: *rgettrashtime* and *rsettrashtime* are deprecated aliases for *gettrashtime -r* and *settrashtime -r* respectively.

REPORTING BUGS

Report bugs to <contact@lizardfs.org>.

COPYRIGHT

2017 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

lizardfs(1),lizardfs(7)

4.11.14 lizardfs-makesnapshot(1)

NAME

lizardfs-makesnapshot - make lazy copies

SYNOPSIS

```
lizardfs makesnapshot [-o] [-f] [-i] [-s <VALUE>] 'SOURCE'... 'DESTINATION'
```

DESCRIPTION

makesnapshot makes a "real" snapshot (lazy copy, like in case of *appendchunks*) of some object(s) or a subtree (similarly to the *cp -r* command). It's atomic in respect to each 'SOURCE' argument separately. If 'DESTINATION' points to an already existing file, an error will be reported unless *-f* (force) or it's alias *-o* (overwrite) option is given.

The new object exists only in metadata until changes to the data are done which will trigger creation of chunks for the changed files or removing metadata entries for erased chunks, unless the trash feature is utilized.

NOTE: if 'SOURCE' is a directory, it's copied as a whole; but if it's followed by a trailing slash, only the directory content is copied.

OPTIONS

-s <value>	This option is used to specify the number of nodes that will be atomically cloned.
-i	ignore missing source nodes in snapshot request

REPORTING BUGS

Report bugs to <contact@lizardfs.org>.

COPYRIGHT

2013-2017 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

lizardfs(1), lizardfs-appendchunks(1), lizardfs(7)

4.11.15 lizardfs-repquota(1)

NAME

lizardfs-repquota, lizardfs-setquota - retrieves and sets quotas for specified users and groups

SYNOPSIS

```
lizardfs repquota [-nhH] (-u UID | -g GID)+ MOUNTPOINT-ROOT-PATH
lizardfs repquota [-nhH] -a MOUNTPOINT-ROOT-PATH
lizardfs repquota [-nhH] -d DIRECTORY-PATH
lizardfs setquota (-u UID | -g GID) SOFT-LIMIT-SIZE HARD-LIMIT-SIZE SOFT-LIMIT-INODES␣
↪HARD-LIMIT-INODES MOUNTPOINT-ROOT-PATH
lizardfs setquota -d SOFT-LIMIT-SIZE HARD-LIMIT-SIZE SOFT-LIMIT-INODES HARD-LIMIT-
↪INODES DIRECTORY-PATH
```

OPTIONS

-u UID	Quota for the user with uid *UID*.
-g GID	Quota for the group with gid *GID*.
-a	Quota for all users and groups.
-d	Quota for directory.

DESCRIPTION

Quota mechanism can be used to limit inodes usage and space usage for users and groups. Once hard limit for inodes or space is reached, no new files can be created for specified user (or group) as long as the quota is not raised or some data is deleted. Also, if hard limit for size is reached, user cannot extend existing files by adding new chunks to them.

In general quotas can be set only by a superuser, but there is a workaround available - one can set SESFLAG_ALLCANCHANGEQUOTA in mfsexports.cfg file. A user can only retrieve a quota of his own (and of his primary group).

repquota prints a summary of the limits that were set and inode/space usage for specified user/group (or all of them) in the following format:

```
# User/Group ID/Inode; Bytes: current usage, soft limit, hard limit; Inodes: current␣
↪usage, soft limit, hard limit;
User/Group/Inode ID (+/-)(+/-) USED-BYTES SOFT-LIMIT-BYTES HARD-LIMIT-BYTES USED-
↪INODES SOFT-LIMIT-INODES HARD-LIMIT-INODES
```

+ indicates that the soft limit was exceeded or the hard limit was reached.

- indicates otherwise.

The first + or - corresponds to bytes limit, while the second one to inodes.

setquota sets quotas for a user or a group on LizardFS. The quotas are not strict, i.e. it is possible to exceed hard limits a bit, mostly by appending data to existing files, but only to certain level (new chunks cannot be created).

REPORTING BUGS

Report bugs to <contact@lizardfs.org>.

COPYRIGHT

2017 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

lizardfs(1),lizardfs(7)

4.11.16 lizardfs-rremove(1)

NAME

lizardfs-rremove - remove recursively

SYNOPSIS

```
lizardfs-rremove* [*-l*] 'OBJECT'...
```

DESCRIPTION

rremove deletes object(s) recursively. This tool can be used on either files or directories.

OPTIONS

-l	This option disables the timeout set for this operation. (the default timeout is 60 seconds)

COPYRIGHT

Copyright 2016-2017 Skytechnology sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

lizardfs(1)

4.11.17 lizardfs(1)

NAME

lizardfs - open prompt to perform LizardFS-specific operations

SYNOPSIS

```
lizardfs getgoal [-r] [-n|-h|-H] 'OBJECT'...
lizardfs rgetgoal [-n|-h|-H] 'OBJECT'...
lizardfs setgoal [-r] [-n|-h|-H] 'NAME' 'OBJECT'...
lizardfs rsetgoal [-n|-h|-H] [+|-]N 'OBJECT'...
lizardfs setquota (-u* 'UID' | -g 'GID') 'SOFT-LIMIT-SIZE'
                     'HARD-LIMIT-SIZE' 'SOFT-LIMIT-INODES'
                     'HARD-LIMIT-INODES' 'MOUNTPOINT-ROOT-PATH'
lizardfs repquota [-nhH] (-u 'UID' | -g 'GID')+ 'MOUNTPOINT-ROOT-PATH'
lizardfs repquota [-nhH] -a 'MOUNTPOINT-ROOT-PATH'
lizardfs gettrashtime [-r] [-n|-h|-H] 'OBJECT'...
lizardfs rgettrashtime [-n|-h|-H] 'OBJECT'...
lizardfs settrashtime [-r] [-n|-h|-H] [+|-]SECONDS 'OBJECT'...
lizardfs rsettrashtime [-n|-h|-H] [+|-]SECONDS 'OBJECT'...
lizardfs geteattr* [-r] [-n|-h|-H] 'OBJECT'...
lizardfs seteattr* [-r] [-n|-h|-H] -f 'ATTRNAME' [-f 'ATTRNAME' ...]
                                           'OBJECT'...
lizardfs deleattr* [-r] [-n|-h|-H] -f 'ATTRNAME' [-f 'ATTRNAME' ...]
                                           'OBJECT'...
lizardfs checkfile 'FILE'...
lizardfs fileinfo 'FILE'...
lizardfs dirinfo [-n|-h|-H] 'OBJECT'...
lizardfs filerepair [-n|-h|-H] 'FILE'...
lizardfs appendchunks 'SNAPSHOT_FILE' 'OBJECT'...
lizardfs makesnapshot [-o] 'SOURCE'... 'DESTINATION'
lizardfs rremove [-l] 'OBJECT'...
```

DESCRIPTION

See respective documents.

OPTIONS

General

Most of *lizardfs* operations use the following options to select the format of printed numbers:

-n	Print exact numbers (e.g. 1024).
-h	Print numbers with binary prefixes (Ki, Mi, Gi as *2^10*, *2^20* etc.).
-H	Print numbers with SI prefixes (k, M, G as *10^3*, *10^6* etc.).

The same can be achieved by setting *MFSHRFORMAT* environment variable to: *0* (exact numbers), *1* or *h* (binary prefixes), *2* or *H* (SI prefixes), *3* or *h+* (exact numbers and binary prefixes), *4* or *H+* (exact numbers and SI prefixes). The default is to print just exact numbers.

Other

-r	This option enables recursive mode.

INHERITANCE

When a new object is created in LizardFS, attributes such as goal, trashtime and extra attributes are inherited from parent directory. So if you set i.e. "noowner" attribute and goal to 3 in a directory then every new object created in this directory will have goal set to 3 and "noowner" flag set. A newly created object inherits always the current set of its parent's attributes. Changing a directory attribute does not affect its already created children. To change an attribute for a directory and all of its children use *'-r'* option.

REPORTING BUGS

Report bugs to <contact@lizardfs.org>.

COPYRIGHT

2013-2017 Skytechnology sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

mfsmount(1), lizardfs-appendchunks(1), lizardfs-checkfile(1), lizardfs-deleattr(1), lizardfs-dirinfo(1), lizardfs-fileinfo(1), lizardfs-filerepair(1), lizardfs-geteattr(1), lizardfs-getgoal(1), lizardfs-gettrashtime(1), lizardfs-makesnapshot(1), lizardfs-rgetgoal(1), lizardfs-rgettrashtime(1), lizardfs-rsetgoal(1), lizardfs-rsettrashtime(1), lizardfs-seteattr(1), lizardfs-setgoal(1), lizardfs-settrashtime(1), lizardfs-rremove(1), lizardfs-repquota(1), lizardfs-setquota(1), lizardfs(7)

4.11.18 globaliolimits.cfg(5)

NAME

globaliolimits.cfg - global I/O limiting configuration

DESCRIPTION

The file *globaliolimits.cfg* contains the configuration of the global I/O limiter.

SYNTAX

```
'option' 'value'
```

Lines starting with a # character are ignored.

OPTIONS

subsystem <subsystem> The cgroups subsystem by which clients are classified. If left unspecified, all clients are considered 'unclassified' (see below).

limit unclassified <throughput in KiB/s> This is a special entry for clients that don't match any group specified in configuration file or for all clients if *subsystem* is unspecified. If this entry is unspecified and subsystem is unspecified as well, I/O limiting is disabled entirely. If this entry is unspecified but subsystem is specified, unclassified clients are not allowed to perform I/O.

limit <group> <throughput in KiB/s> Set limit for clients belonging to the cgroups group *<group>*. In LizardFS, subgroups of *<group>* constitute independent groups; they are not allowed to use *<group>'s* bandwidth reservation and they don't count against *<group>'s* usage.

EXAMPLES

```
# empty file
```

I/O limiting is disabled, no limits are enforced.

```
limit unclassified 1024
```

All clients are 'unclassified' and share 1MiB/s of bandwidth.

```
subsystem blkio
limit /a 1024
```

Clients in the **blkio /a** group are limited to 1MiB/s, no other clients can perform any I/O.

```
subsystem blkio
limit unclassified 256
limit /a   1024
limit /b/a 2048
```

The **blkio** group **/a** is allowed to transfer 1MiB/s, while **/b/a** gets 2MiB/s. Clients from other groups (e.g. **/b**, **/z**, **/a/a**, **/b/z**) are considered 'unclassified' and share 256KiB/s of bandwidth.

TUNING NOTES

Global I/O limiting is managed by the master server. Mount instances reserve bandwidth allocations from master when they want to perform I/O to chunkservers.

To avoid overloading the master under heavy traffic, mounts try to predict their future usage and reserve at once all the bandwidth they will for the next renegotiation period (see mfsmaster.cfg(5)).

Such reservation are wasted if the traffic at given mount instance suddenly drops.

The ratio of bandwidth being wasted due to this phenomenon shouldn't exceed **fsp/b**, where:

f is the frequency of sudden traffic drops in the whole installation (in 1/s)

s is the average size of such drop (in KiB/s)

p is the renegotiation period (in s)

b is the bandwidth limit (in KiB/s)

This applies to each group separately, because groups reserve their bandwidth independently from each other.

REPORTING BUGS

Report bugs to <contact@lizardfs.org>.

COPYRIGHT

Copyright 2008-2009 Gemius SA, 2013-2016 Skytechnology sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

mfsmaster.cfg(5)

See also:

mfsmaster.cfg(5) *mfsmaster.cfg(5)*

4.11.19 iolimits.cfg(5)

NAME

iolimits.cfg - local I/O limiting configuration

DESCRIPTION

The file *iolimits.cfg* contains the configuration of the local I/O limiter.

SYNTAX, OPTIONS

The same as in globaliolimits.cfg(5).

REPORTING BUGS

Report bugs to <contact@lizardfs.org>.

COPYRIGHT

Copyright 2008-2009 Gemius SA, 2013-2016 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

mfsmount.cfg(5), globaliolimits.cfg(5)

See also:

mfsmaster.cfg(5) mfsmount.cfg(5) globaliolimits.cfg(5)

4.11.20 mfsexports.cfg(5)

NAME

mfsexports.cfg - LizardFS access control for mfsmounts

DESCRIPTION

The file *mfsexports.cfg* contains LizardFS access list for *mfsmount* clients.

SYNTAX

```
ADDRESS DIRECTORY [OPTIONS]
```

Lines starting with a # character are ignored.

ADDRESS is an IP address, network or address range and can be specified in several forms:

***** all addresses

n.n.n.n single IP address

n.n.n.n/b IP class specified by network address and bits number

n.n.n.n/m.m.m.m IP class specified by network address and mask

f.f.f.f-t.t.t.t IP range specified by from-to addresses (inclusive)

DIRECTORY can be / or a path relative to your LizardFS namespaces root; the special value . stands for the MFS-META companion filesystem.

OPTIONS

ro, readonly export tree in read-only mode (default)

rw, readwrite export tree in read-write mode

ignoregid disable testing of group access at *mfsmaster* level (it's still done at *mfsmount* level) - in this case "group" and "other" permissions are logically added; needed for supplementary groups to work (*mfsmaster* receives only user primary group information)

dynamicip allows reconnecting of already authenticated client from any IP address (the default is to check the IP address on reconnect)

maproot=USER[:GROUP] maps root (uid=0) accesses to given user and group (similarly to maproot option in NFS mounts);

USER' and GROUP can be given either as name or number; if no group is specified, USER's primary group is used. Names are resolved on *mfsmaster* side (see note below).

mapall=USER[:GROUP] like above but maps all non privileged users (uid!=0) accesses to given user and group (see notes below).

minversion=VER rejects access from clients older than specified

mingoal=N, maxgoal=N specify range in which goals can be set by users

mintrashtime=TDUR, maxtrashtime=TDUR specify range in which trashtime can be set by users

password=PASS, md5pass=MD5 requires password authentication in order to access specified resource

alldirs allows to mount any subdirectory of specified directory (similarly to NFS)

nonrootmeta allows non-root users to use filesystem mounted in the meta mode (option available only in this mode)

Default options are: **ro,maproot=999:999**.

NOTES

USER and GROUP names (if not specified by explicit uid/gid number) are resolved on *mfsmaster* host.

TDUR can be specified as number without time unit (number of seconds) or combination of numbers with time units. Time units are: *W*, *D*, *H*, *M*, *S*. Order is important - less significant time units can't be defined before more significant time units.

Option *mapall* works in LizardFS in different way than in NFS, because of using FUSE's "default_permissions" option. When mapall option is used, users see all objects with uid equal to mapped uid as their own and all other as root's objects. Similarly objects with gid equal to mappedgid are seen as objects with current user's primary group and all other objects as objects with group 0 (usually wheel). With *mapall* option set attribute cache in kernel is always turned off.

EXAMPLES

```
*                       /       ro
192.168.1.0/24          /       rw
192.168.1.0/24          /       rw,alldirs,maproot=0,password=passcode
10.0.0.0-10.0.0.5       /test   rw,maproot=nobody,password=test*
10.1.0.0/255.255.0.0 /public rw,mapall=1000:1000*
10.2.0.0/16             /       rw,alldirs,maproot=0,mintrashtime=2h30m,maxtrashtime=2w
```

REPORTING BUGS

Report bugs to <contact@lizardfs.org>.

COPYRIGHT

Copyright 2008-2009 Gemius SA, 2013-2016 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

mfsmaster(8), mfsmaster.cfg(5)

4.11.21 mfsgoals.cfg(5)

NAME

mfsgoals.cfg - replication goals configuration file

DESCRIPTION

The file *mfsgoals.cfg* contains definitions of the replication goals.

SYNTAX

```
'id' 'name' : $'type' { 'label' ... }
'id' 'name' : 'label' ...
```

The # character starts comments.

DETAILS

There are 40 replication goals, with 'ids' between 1 and 40, inclusive. Each file stored on the filesystem refers to some goal id and is replicated according to the goal currently associated with this id.

By default, goal 1 means one copy on any chunkserver, goal 2 means two copies on any two chunkservers and so on, until 5 - which is the maximal default number of copies. The purpose of mfsgoals.cfg is to override this behavior, when desired. The file is a list of goal definitions, each consisting of 'id', 'name' and a list of 'labels'. The maximal length of this list is 40 labels.

'id' indicates the goal id to be redefined. If some files are already assigned this goal id, their effective goal will change.

‘name’ is a human readable name used by the user interface tools (lizardfs-setgoal(1), lizardfs-getgoal(1)). ‘name’ can consist of up to 32 alphanumeric characters: a-z, A-Z, 0-9, _.

‘type’ specifies the goal type - currently supported types are:

std for each file using this goal and for each label, the system will try to maintain a copy of the file on some chunkserver with this label.

xorN for each file using this goal, the system will split the file into N+1 parts (N ordinary + 1 parity). For reading, any N of the parts are necessary. If labels are specified, parts will be kept on chunkservers with these labels. Otherwise, default wildcard labels will be used. N can be in range from 2 to 9.

ec(K,M) for each file using this goal, the system will split the file into K + M parts (K data parts and M parity). For reading, any K of the parts are necessary. If labels are specified, parts will be kept on chunkservers with these labels. Otherwise, default wildcard labels will be used. K can be in range from 2 to 32 and M from 1 to 32.

If the type is unspecified it is assumed to be *std*.

The list of ‘labels’ is a list of chunkserver labels as defined in mfschunkserver.cfg(5). ‘label’ can consist of up to 32 alphanumeric characters: a-z, A-Z, 0-9, _.

One label may occur multiple times - in such case the system will create one copy per each occurrence.

The special label _ means “a copy on any chunkserver”.

Note: changing the definition of a goal in mfsgoals.cfg affects all files which currently use a given goal id.

EXAMPLES

Some example goal definitions:

```
3 3 : _ _ _   # one of the default goals (three copies anywhere)

8 not_important_file : _   # only one copy

11 important_file : _ _

12 local_copy_on_mars : mars _ # at least one copy in the Martian datacenter

13 cached_on_ssd : ssd _

14 very_important_file : _ _ _ _

15 default_xor3 : $xor3

16 fast_read : $xor2 { ssd ssd hdd }

17 xor5 : $xor5 { hdd } # at least one part on hdd

18 first_ec : $ec(3,1)

19 ec32_ssd : $ec(3,2) { ssd ssd ssd ssd ssd } # all parts on ssd

20 ec53_mixed : $ec(5,3) { hdd ssd hdd _ _ _ _ _ } # two parts on hdd and one part on␣
↪ssd
```

SNAPSHOT FILES

Snapshots share data with the original file until the file receives modification and diverges from the snapshotted version. If some snapshot has different goal than its original file, any shared data are stored according to the goal with higher 'id' of the two.

REPORTING BUGS

Report bugs to <contact@lizardfs.org>.

COPYRIGHT

Copyright 2008-2009 Gemius SA, 2013-2016 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

mfsmaster.cfg(5)

4.11.22 mfsmaster(8)

NAME

mfsmaster - start, restart or stop Lizard File System metadata server process

SYNOPSIS

```
mfsmaster [-f] [-c CFGFILE] [-u] [-d] [-t LOCKTIMEOUT] [-p PIDFILE] [ACTION]
mfsmaster -s [-c CFGFILE]
mfsmaster -v
mfsmaster -h
```

DESCRIPTION

mfsmaster is the metadata server program of Lizard File System. Depending on parameters it can start, restart or stop the LizardFS metadata server process. Without any options it starts the LizardFS metadata server, killing first the running process if a lock file exists.

The metadata server can work in one of two modes (personalities):

- master
- shadow

If the metadata server works with *master* personality it acts as the main metadata server govering all file system metadata modifications. If the metadata server works with *shadow* personality it acts as backup metadata server ready for immediate deployment as new *master* in case of the current *master* failing.

Shadow only accepts connections from lizardfs-admin, i.e. mfschunkserver, mfsmetalogger and mfsmount (the client) are not allowed to connect to the *shadow* instance.

Current metadata server personality is defined in the metadata server configuration file and can be changed on the fly from *shadow* to *master* by proper modification and reloading of its configuration file.

Master and *shadow* are designed to run simultaneously in sync forever. It is very unlikely but still (due to a memory corruption or a bug) possible that after some time their metadata will somehow differ. Since version 2.5.2 metadata checksum is maintained both by *master* and *shadow*, in order to detect and fix possible metadata corruptions. In case a mismatch is detected *shadow* asks *master* to double check its metadata and dump its current snapshot.

After the metadata is dumped and the checksum in the *master* is recalculated, shadow downloads the new metadata snapshot, which should ensure that the master and all its shadows have exactly the same metadata.

SIGHUP (or *reload* ACTION) forces **mfsmaster** to reload all configuration files.

-v	print version information and exit
-h	print usage information and exit
-f	(deprecated, use *start* action instead) forcily run LizardFS master process, without trying to kill previous instance (this option allows to run LizardFS master if stale PID file exists)
-s	(deprecated, use *stop* action instead) stop LizardFS master process
-c CFGFILE	specify alternative path of configuration file (default is *mfsmaster.cfg* in system configuration directory)
-u	log undefined configuration values (when default is assumed)
-d	run in foreground, don*t daemonize
-t LOCKTIMEOUT	how long to wait for lockfile (default is 60 seconds)
-p PIDFILE	write process ID (pid) to given file

ACTION is one of *start*, *stop*, *restart*, *reload*, *test*, *isalive* or *kill*. The default action is *restart*.

FILES

mfsmaster.cfg configuration file for the LizardFS master process (see `mfsmaster.cfg(5)`)

mfsexports.cfg LizardFS access control file (used with *mfsmount's* 1.6.0 or later, see `mfsexports.cfg(5)`)

mfstopology.cfg Network topology definitions (see `mfstopology.cfg(5)`)

mfsmaster.lock PID file of running LizardFS master process

.mfsmaster.lock lock file of the running LizardFS master process (created in data directory)

metadata.mfs, metadata.mfs.back LizardFS filesystem metadata image

changelog.*.mfs LizardFS filesystem metadata change logs (merged into *metadata.mfs* once per hour)

data.stats LizardFS master charts state

REPORTING BUGS

Report bugs to <contact@lizardfs.org>.

COPYRIGHT

Copyright 2008-2009 Gemius SA, 2013-2016 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

mfsmetarestore(8), mfschunkserver(8), mfsmount(1), mfsmaster.cfg(5), mfsexports.cfg(5), mfstopology.cfg(5), lizardfs(7)

4.11.23 mfsmaster.cfg(5)

NAME

mfsmaster.cfg - main configuration file for mfsmaster

DESCRIPTION

The file **mfsmaster.cfg** contains the configuration of the LizardFS metadata server process.

SYNTAX

```
OPTION = VALUE
```

Lines starting with a # character are being ignored.

OPTIONS

Configuration options:

PERSONALITY Current *personality* of this instance of the metadata server. Valid values are *master*, *shadow* and *ha-cluster-managed*. If the installation is managed by an HA cluster the only valid value is *ha-cluster-managed*, otherwise the only valid values are *master* and *shadow*, in which case only one metadata server in LizardFS shall have *master* personality.

```
PERSONALITY = master
```

means that this instance of metadata server acts as main metadata server govering all file system metadata modifications.

```
PERSONALITY = shadow
```

means that this instance of the metadata server acts as backup metadata server ready for immediate deployment as the new *master* in case of a failure of the current *master*.

Metadata server personality can be changed at any moment as long as one changes personality from *shadow* to *master*, changing personality the other way around is forbidden.

```
PERSONALITY = ha-cluster-managed
```

means that this instance is managed by HA cluster, server runs in *shadow* mode as long as its not remotly promoted to *master*.

DATA_PATH where to store metadata files and lock file

WORKING_USER user to run daemon as

WORKING_GROUP group to run daemon as (optional - if empty then the default user group will be used)

SYSLOG_IDENT name of the process to place in syslog messages (default is mfsmaster)

LOCK_MEMORY whether to perform mlockall() to avoid swapping out mfsmaster process (default is 0, i.e. no)

NICE_LEVEL nice level to run daemon with (default is -19 if possible; note: process must be started as root to increase priority)

EXPORTS_FILENAME alternative name of the *mfsexports.cfg* file

TOPOLOGY_FILENAME alternative name of the *mfstopology.cfg* file

CUSTOM_GOALS_FILENAME alternative name of the *mfsgoals.cfg* file

PREFER_LOCAL_CHUNKSERVER If a client mountpoint has a local chunkserver, and a given chunk happens to reside locally, then mfsmaster will list the local chunkserver first. However, when the local client mount is issuing many read(s)/write(s), to many local chunks, these requests can overload the local chunkserver and disk subsystem. Setting this to 0(the default is 1) means that remote chunkservers will be considered as equivalent to the local chunkserver.

This is useful when the network is faster than the disk, and when there is high-IO load on the client mountpoints.

BACK_LOGS number of metadata change log files (default is 50)

BACK_META_KEEP_PREVIOUS number of previous metadata files to be kept (default is 1)

AUTO_RECOVERY when this option is set (equals 1) master will try to recover metadata from changelog when it is being started after a crash; otherwise it will refuse to start and *mfsmetarestore* should be used to recover the metadata (default setting is 0)

REPLICATIONS_DELAY_INIT DEPRECATED - see *OPERATIONS_DELAY_INIT*

REPLICATIONS_DELAY_DISCONNECT DEPRECATED - see *OPERATIONS_DELAY_DISCONNECT*

OPERATIONS_DELAY_INIT initial delay in seconds before starting chunk operations (default is 300)

OPERATIONS_DELAY_DISCONNECT chunk operations delay in seconds after chunkserver disconnection (default is 3600)

MATOML_LISTEN_HOST IP address to listen on for metalogger connections (* means any)

MATOML_LISTEN_PORT port to listen on for metalogger connections (default is 9419)

MATOML_LOG_PRESERVE_SECONDS how many seconds of change logs have to be preserved in memory (default is 600; note: logs are stored in blocks of 5k lines, so sometimes the real number of seconds may be bit bigger; zero disables extra logs storage)

MATOCS_LISTEN_HOST IP address to listen on for chunkserver connections (* means any)

MATOCS_LISTEN_PORT port to listen on for chunkserver connections (default is 9420)

MATOCL_LISTEN_HOST IP address to listen on for client (mount) connections (* means any)

MATOCL_LISTEN_PORT port to listen on for client (mount) connections (default is 9421)

MATOTS_LISTEN_HOST IP address to listen on for tapeserver connections (* means any)

MATOTS_LISTEN_PORT Port to listen on for tapeserver connections (default is 9424)

CHUNKS_LOOP_MAX_CPS Chunks loop shouldn*t check more chunks per seconds than given number (default is 100000)

CHUNKS_LOOP_MIN_TIME Chunks loop will check all chunks in specified time (default is 300) unless *CHUNKS_LOOP_MAX_CPS* will force slower execution.

CHUNKS_LOOP_PERIOD Time in milliseconds between chunks loop execution (default is 1000).

CHUNKS_LOOP_MAX_CPU Hard limit on CPU usage by chunks loop (percentage value, default is 60).

CHUNKS_SOFT_DEL_LIMIT Soft maximum number of chunks to delete on one chunkserver (default is 10)

CHUNKS_HARD_DEL_LIMIT Hard maximum number of chunks to delete on one chunkserver (default is 25)

CHUNKS_WRITE_REP_LIMIT Maximum number of chunks to replicate to one chunkserver (default is 2)

CHUNKS_READ_REP_LIMIT Maximum number of chunks to replicate from one chunkserver (default is 10)

ENDANGERED_CHUNKS_PRIORITY Percentage of endangered chunks that should be replicated with high priority. Example: when set to 0.2, up to 20% of chunks served in one turn would be extracted from endangered priority queue.

When set to 1 (max), no other chunks would be processed as long as there are any endangered chunks in the queue (not advised)

(default is 0, i.e. there is no overhead for prioritizing endangered chunks).

ENDANGERED_CHUNKS_MAX_CAPACITY Max capacity of endangered chunks queue. This value can limit memory usage of master server if there are lots of endangered chunks in the system. This value is ignored if ENDANGERED_CHUNKS_PRIORITY is set to 0. (default is 1Mi, i.e. no more than 1Mi chunks will be kept in a queue).

ACCEPTABLE_DIFFERENCE The maximum difference between disk usage on chunkservers that doesn*t trigger chunk rebalancing (default is 0.1, i.e. 10%).

CHUNKS_REBALANCING_BETWEEN_LABELS When balancing disk usage, allow moving chunks between servers with different labels (default is 0, i.e. chunks will be moved only between servers with the same label).

REJECT_OLD_CLIENTS Reject **mfsmounts** older than 1.6.0 (0 or 1, default is 0). Note that *mfsexports* access control is NOT used for those old clients.

GLOBALIOLIMITS_FILENAME Configuration of global I/O limits (default is no I/O limiting)

GLOBALIOLIMITS_RENEGOTIATION_PERIOD_SECONDS How often mountpoints will request bandwidth allocations under constant, predictable load (default is 0.1)

GLOBALIOLIMITS_ACCUMULATE_MS After inactivity, no waiting is required to transfer the amount of data equivalent to normal data flow over the period of that many milliseconds (default is 250)

METADATA_CHECKSUM_INTERVAL how often metadata checksum shall be sent to backup servers (default is: every 50 metadata updates)

METADATA_CHECKSUM_RECALCULATION_SPEED how fast should metadata be recalculated in the background (default : 100 objects per function call)

DISABLE_METADATA_CHECKSUM_VERIFICATION should checksum verification be disabled while applying changelog

NO_ATIME when this option is set to 1 inode access time is not updated on every access, otherwise (when set to 0) it is updated (default is 0)

METADATA_SAVE_REQUEST_MIN_PERIOD minimal time in seconds between metadata dumps caused by requests from shadow masters (default is 1800)

SESSION_SUSTAIN_TIME Time in seconds for which client session data (e.g. list of open files) should be sustained in the master server after connection with the client was lost. Values between 60 and 604800 (one week) are accepted. (default is 86400)

USE_BDB_FOR_NAME_STORAGE When this option is set to 1 Berkley DB is used for storing file/directory names in file (DATA_PATH/name_storage.db). By default all strings are kept in system memory. (default is 0)

BDB_NAME_STORAGE_CACHE_SIZE Size of memory cache (in MB) for file/directory names used by Berkeley DB storage. (default is 10)

FILE_TEST_LOOP_MIN_TIME The test files loop will try to check all files within the specified time given in seconds (default is 300). It is possible for the loop to take more time if the master server is busy or the machine doesn't have enough processing power to make all the needed calculations.

AVOID_SAME_IP_CHUNKSERVERS When this option is set to 1, process of selecting chunkservers for chunks will try to avoid using those that share the same ip. (default is 0)

SNAPSHOT_INITIAL_BATCH_SIZE This option can be used to specify initial number of snapshotted nodes that will be atomically cloned before enqueuing the task for execution in fixed-sized batches. (default is 1000)

SNAPSHOT_INITIAL_BATCH_SIZE_LIMIT This option specifies the maximum initial batch size set for snapshot request. (default is 10000)

Options below are mandatory for all Shadow instances:

MASTER_HOST address of the host running LizardFS metadata server that currently acts as *master*

MASTER_PORT port number where LizardFS metadata server currently running as *master* listens for connections from shadows and metaloggers (default is 9420)

MASTER_RECONNECTION_DELAY delay in seconds before trying to reconnect to metadata server after disconnection (default is 1)

MASTER_TIMEOUT timeout (in seconds) for metadata server connections (default is 60)

LOAD_FACTOR_PENALTY When set, percentage of load will be added to a chunkserver disk usage while determining the most fitting chunkserver. Heavy loaded chunkservers will be picked for operations less frequently. (default is 0, correct values are in the range of 0 to 0.5)

SNAPSHOT_INITIAL_BATCH_SIZE This option can be used to specify initial number of snapshotted nodes that will be atomically cloned before enqueuing the task for execution in fixed-sized batches. (default is 1000)

Note: Chunks in master are tested in loop. Speed (or frequency) is regulated by the two options *CHUNKS_LOOP_MIN_TIME* and *CHUNKS_LOOP_MAX_CPS*. The first one defines the minimal time of the loop and the second one the maximal number of chunk tests per second. Typically at the beginning, when the number of chunks is small, time is constant, regulated by *CHUNK_LOOP_MIN_TIME*, but when the number of chunks becomes bigger then the time of the loop can increase according to *CHUNKS_LOOP_MAX_CPS*.

Deletion limits are defined as *soft* and *hard* limit. When the number of chunks to delete increases from loop to loop, the current limit can be temporary increased above the soft limit, but never above the hard limit.

REPORTING BUGS

Report bugs to <contact@lizardfs.org>.

COPYRIGHT

Copyright 2008-2009 Gemius SA, 2013-2016 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

mfsmaster(8), mfsexports.cfg(5), mfstopology.cfg(5)

4.11.24 mfstopology.cfg(5)

NAME

mfstopology.cfg - LizardFS network topology definitions

DESCRIPTION

The file *mfstopology.cfg* contains assignments of IP addresses into network locations (usually switch numbers). This file is optional. If your network has one switch or decreasing traffic between switches is not necessary then leave this file empty.

SYNTAX

```
'ADDRESS' 'SWITCH-NUMBER'
```

Lines starting with # character are ignored.

‘ADDRESS’ can be specified in several forms:

```
*                all addresses
n.n.n.n          single IP address
n.n.n.n/b        IP class specified by network address and bits number
n.n.n.n/m.m.m.m  IP class specified by network address and mask
f.f.f.f-t.t.t.t  IP range specified by from-to addresses (inclusive)
```

‘SWITCH-NUMBER’ can be specified as any positive 32-bit numer.

NOTES

If one IP belongs to more than one definition then last definition is used.

As for now distance between switches is constant. So distance between machines is calculated as: *0* when IP numbers are the same, *1* when IP numbers are different, but switch numbers are the same and *2* when switch numbers are different

Distances are used only to sort chunkservers during read and write operations. New chunks are still created randomly. Also rebalance routines do not take distances into account.

COPYRIGHT

Copyright 2008-2009 Gemius SA, 2013-2016 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

mfsmaster(8), mfsmaster.cfg(5),lizardfs(7)

4.11.25 mfschunkserver(8)

NAME

mfschunkserver - start, restart or stop Lizard File System chunkserver process

SYNOPSIS

```
mfschunkserver* [-f] [-c CFGFILE] [-u] [-d] [-t LOCKTIMEOUT] [-p PIDFILE] [ACTION]

mfschunkserver -s [-c CFGFILE]

mfschunkserver -v

mfschunkserver -h
```

DESCRIPTION

mfschunkserver is the data server of Lizard File System. Depending on parameters it can start, restart or stop the LizardFS chunkserver process. Without any options it starts the LizardFS chunkserver, killing previously run process if lock file exists.

SIGHUP (or *reload* ACTION) forces *mfschunkserver* to reload all configuration files.

Chunkserver periodically tests stored chunks (see *HDD_TEST_FREQ* option in *mfschunkserver.cfg* manual).

LizardFS master doesn't send metadata change logs to chunkserver and expects at least one *mfsmetalogger* daemon to connect.

Chunkserver scans attached disks in background.

OPTIONS

-v	print version information and exit
-h	print usage information and exit
-f	(deprecated, use *start* action instead) forcily run LizardFS chunkserver process, without trying to kill previous instance (this option allows to run LizardFS chunkserver if stale PID file exists)
-s	(deprecated, use *stop* action instead) stop the LizardFS chunkserver process
-c CFGFILE	specify alternative path of configuration file (default is *mfschunkserver.cfg* in system configuration directory)
-u	log undefined configuration values (when default is assumed)
-d	run in foreground, don't daemonize
-t LOCKTIMEOUT	how long to wait for lockfile (default is 60 seconds)
-p PIDFILE	write pid to given file

ACTION is one of *start*, *stop*, *restart*, *reload*, *test*, *isalive* or *kill*. Default action is *restart*.

FILES

mfschunkserver.cfg configuration file for LizardFS chunkserver process; refer to *mfschunkserver.cfg(5)* manual for details.

mfshdd.cfg list of directories (mountpoints) used for LizardFS storage (one per line; directory prefixed by * character causes given directory to be freed by replicating all data already stored there to another locations)

mfschunkserver.lock PID file of running LizardFS chunkserver process

.mfschunkserver.lock lock file of running LizardFS chunkserver process (created in data directory)

data.csstats Chunkserver charts state

REPORTING BUGS

Report bugs to <contact@lizardfs.org>.

COPYRIGHT

Copyright 2008-2009 Gemius SA, 2013-2017 Skytechnology sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

mfsmaster(8), mfsmount(1), mfschunkserver.cfg(5), mfshdd.cfg(5), lizardfs(7)

4.11.26 mfschunkserver.cfg(5)

NAME

mfschunkserver.cfg - main configuration file for *mfschunkserver*

DESCRIPTION

The file *mfschunkserver.cfg* contains the configuration of the LizardFS chunkserver process.

SYNTAX

```
OPTION = VALUE
```

Lines starting with the # character are ignored.

OPTIONS

DATA_PATH where to store files with usage statistics and daemon lock file

LABEL the label of this chunkserver (for tiering)

WORKING_USER user to run daemon as

WORKING_GROUP group to run daemon as (optional - if empty then default user group will be used)

SYSLOG_IDENT name of process to place in syslog messages (default is mfschunkserver)

LOCK_MEMORY whether to perform mlockall() to avoid swapping out mfschunkserver process (default is 0, i.e. no)

NICE_LEVEL nice level to run daemon with (default is -19 if possible; note: process must be started as root to increase priority)

MASTER_HOST address of LizardFS master host to connect with (default is mfsmaster)

MASTER_PORT number of LizardFS master port to connect with (default is 9420)

MASTER_RECONNECTION_DELAY delay in seconds before trying to reconnect to the master server after disconnection (default is 5)

MASTER_TIMEOUT timeout (in seconds) for the master server connection (default is 60, minimum is 0.01)

BIND_HOST local address to use for connecting with the master server (default is *, i.e. default local address)

CSSERV_LISTEN_HOST IP address to listen on for client (mount) connections (* means any)

CSSERV_LISTEN_PORT port to listen on for client (mount) connections (default is 9422)

CSSERV_TIMEOUT timeout (in seconds) for client (mount) connections (default is 5)

HDD_CONF_FILENAME alternative name of *mfshdd.cfg* file

HDD_LEAVE_SPACE_DEFAULT free space threshold to set volume as 100% utilized when there is less than given amount of free space left (default is "4GiB"). This number is always added to the used disk space reported by chunkserver.

HDD_TEST_FREQ chunk test period in seconds (default is 10)

HDD_ADVISE_NO_CACHE whether to remove each chunk from page when closing it to reduce cache pressure generated by chunkserver (default is 0, i.e. no)

HDD_PUNCH_HOLES if enabled then chunkserver detects zero values in chunk data and frees corresponding file blocks (decreasing file system usage). This option works only on Linux and only with file systems supporting punching holes (XFS, ext4, Btrfs, tmpfs)

ENABLE_LOAD_FACTOR if enabled, the chunkserver will send periodical reports of its I/O load to the master, which will be taken into consideration when selecting chunkservers for I/O operations.

REPLICATION_BANDWIDTH_LIMIT_KBPS limit how many kilobytes can be replicated from other chunkservers to this chunkserver in every second (by default undefined, i.e. no limits)

NR_OF_NETWORK_WORKERS number of threads which handle (in a round-robin manner) connections with clients (default is 1); these threads are responsible for reading from sockets and copying data from internal buffers to sockets

NR_OF_HDD_WORKERS_PER_NETWORK_WORKER number of threads that each network worker may use to do disk operations like opening chunks, reading or writing them (default is 2)

READ_AHEAD_KB additional number of kilobytes which should be passed to posix_fadvise(POSIX_FADV_WILLNEED) before reading data from a chunk (default is 0, i.e. use posix_fadvise only with the amount of data that is really needed; the value is aligned down to 64 KiB)

MAX_READ_BEHIND_KB try to fix out-of-order read requests; the value tells how much of skipped data to read if an offset of some read operation is greater than the offset where the previos operation finished (default is 0, i.e. don't read any skipped data; the value is aligned down to 64 KiB)

CREATE_NEW_CHUNKS_IN_MOOSEFS_FORMAT whether to create new chunks in the MooseFS format (signature + <checksum> + <data block>) or in the newer interleaved format ([<checksum> <data block>]*). (Default is 1, i.e. new chunks are created in MooseFS format)

PERFORM_FSYNC call fsync() after a chunk is modified (default is 1, i.e. enabled)

REPORTING BUGS

Report bugs to <contact@lizardfs.org>.

COPYRIGHT

Copyright 2008-2009 Gemius SA, 2013-2016 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

mfschunkserver(8), mfshdd.cfg(5)

4.11.27 mfshdd.cfg(5)

NAME

mfshdd.cfg - list of LizardFS storage directories for mfschunkserver

DESCRIPTION

The file *mfshdd.cfg* contains a list of directories (mountpoints) used for LizardFS storage (one per line). A directory prefixed by a single * character causes that directory to be freed by replicating all data already stored there to different locations. Lines starting with a # character are ignored.

REPORTING BUGS

Report bugs to <contact@lizardfs.org>.

COPYRIGHT

Copyright 2008-2009 Gemius SA, 2013-2016 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

mfschunkserver(8), mfschunkserver.cfg(5)

4.11.28 mfsmetalogger(8)

NAME

mfsmetalogger - start, restart or stop Lizard File System metalogger process

SYNOPSIS

```
mfsmetalogger [-f] [-c CFGFILE] [-p PIDFILE] [-u] [-d] [-t LOCKTIMEOUT] [ACTION]
mfsmetalogger -s [-c CFGFILE]
mfsmetalogger -v
mfsmetalogger -h
```

DESCRIPTION

mfsmetalogger is the metadata replication server of the Lizard File System. Depending on the parameters given to it, it can start, restart or stop the LizardFS metalogger process. Without any options it starts the LizardFS metalogger, killing any previously run process if a lock file exists.

SIGHUP (or *reload ACTION*) forces **mfsmetalogger** to reload all configuration files.

-v	print version information and exit
-h	print usage information and exit
-c CFGFILE	specify alternative path of the configuration file (default is **mfsmetalogger.cfg** in the system configuration directory).
-u	log undefined configuration values (for which defaults are assumed)
-d	run in the foreground, don't daemonize
-t LOCKTIMEOUT	how long to wait for lockfile (default is 60 seconds)
-p PIDFILE	write pid to given file

ACTION is one of *start*, *stop*, *restart*, *reload*, *test*, *isalive* or *kill*. Default action is *restart*.

FILES

mfsmetalogger.cfg configuration file for LizardFS metalogger process; refer to the mfsmetalogger.cfg(5) manual page for defails

mfsmetalogger.lock PID file of running LizardFS metalogger process

.mfsmetalogger.lock lock file of running LizardFS metalogger process (created in data directory)

changelog_ml.*.mfs LizardFS filesystem metadata change logs (backup of master change log files)

metadata.ml.mfs.back Latest copy of complete metadata.mfs.back file from LizardFS master.

sessions.ml.mfs Latest copy of sessions.mfs file from LizardFS master.

REPORTING BUGS

Report bugs to <contact@lizardfs.org>.

COPYRIGHT

Copyright 2008-2009 Gemius SA, 2013-2017 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

mfsmaster(8), mfsmetalogger.cfg(5), lizardfs(7)

4.11.29 mfsmetalogger.cfg(5)

NAME

mfsmetalogger.cfg - configuration file for mfsmetalogger

DESCRIPTION

The file *mfsmetalogger.cfg* contains configuration of LizardFS metalogger process.

SYNTAX

```
OPTION = VALUE
```

Lines starting with the # character are ignored.

OPTIONS

DATA_PATH where to store metadata files

WORKING_USER user to run daemon as

WORKING_GROUP group to run daemon as (optional - if empty then default user group will be used)

SYSLOG_IDENT name of process to place in syslog messages (default is mfsmetalogger)

LOCK_MEMORY whether to perform mlockall() to avoid swapping out mfsmetalogger process (default is 0, i.e. no)

NICE_LEVEL nice level to run daemon with (default is -19 if possible; note: process must be started as root to be able to increase priority)

BACK_LOGS number of metadata change log files (default is 50)

BACK_META_KEEP_PREVIOUS number of previous metadata files to be kept (default is 3)

META_DOWNLOAD_FREQ metadata download frequency in hours (default is 24, at most *BACK_LOGS*/2)

MASTER_HOST address of LizardFS master host to connect with (default is mfsmaster)

MASTER_PORT number of LizardFS master port to connect with (default is 9419)

MASTER_RECONNECTION_DELAY delay in seconds before trying to reconnect to master after disconnection (default is 30)

MASTER_TIMEOUT timeout (in seconds) for master connections (default is 60)

REPORTING BUGS

Report bugs to <contact@lizardfs.org>.

COPYRIGHT

Copyright 2008-2009 Gemius SA, 2016 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

mfsmetalogger(8)

4.11.30 mfsmount(1)

NAME

mfsmount - mount a LizardFS File System

SYNOPSIS

```
mfsmount 'mountpoint' [-d] [-f] [-s] [-m] [-n] [-p]
        [-H 'HOST'] [-P PORT] [-S PATH] [-o opt[,opt]...]
mfsmount -h | --help
mfsmount -V | --version
```

DESCRIPTION

Mount a LizardFS Filesystem

DATA CACHE MODES

There are three cache modes: *NO*, *YES* and *AUTO*. The default option is *AUTO* and you shuldn't change it unless you really know what you are doing. In *AUTO* mode data cache is managed automatically by mfsmaster.

NO,NONE or NEVER never allow files data to be kept in cache (safest but can reduce efficiency)

YES or ALWAYS always allow files data to be kept in cache (dangerous)

AUTO file cache is managed by mfsmaster automatically (should be very safe and efficient)

SUGID CLEAR MODE

SMODE can be set to:

NEVER MFS will not change suid and sgid bit on chown

ALWAYS clear suid and sgid on every chown - safest operation

OSX standard behavior on OS X and Solaris (chown made by unprivileged user clear suid and sgid)

BSD standard behavior on BSD systems (like on OSX, but only when something is really changed)

EXT standard behavior in most file systems on Linux (directories not changed, others: suid cleared always, sgid only when group exec bit is set)

XFS standard behavior in XFS on Linux (like EXT but directories are changed by unprivileged users)

SMODE extra info:

btrfs,ext2,ext3,ext4,hfs[+],jfs,ntfs and reiserfs on Linux work as 'EXT'.

Only xfs on Linux works a little different. Beware that there is a strange operation - chown(-1,-1) which is usually converted by a kernel into something like 'chmod ug-s', and therefore can't be controlled by MFS as 'chown'

OPTIONS

General options

-h, --help	display help and exit
-V	display version information and exit

FUSE options

-d, -o debug	enable debug mode (implies -f)
-f	foreground operation
-s	disable multi-threaded operation

LizardFS options

```
-m, --meta, -o mfsmeta
   mount MFSMETA companion filesystem instead of primary LizardFS

-c CFGFILE, -o "mfscfgfile=CFGFILE"
   loads file with additional mount options

-n
   omit default mount options (-o allow_other,default_permissions)

-p, -o askpassword
   prompt for password (interactive version of -o mfspassword='PASS')

-H HOST, -o mfsmaster\=HOST
   connect with LizardFS master on <HOST> (default is mfsmaster)

-P PORT, -o mfsport=PORT
   connect with LizardFS master on <PORT> (default is 9421)
```

(continues on next page)

(continued from previous page)

```
-B <HOST>, -o mfsbind=<HOST>
   local address to use for connecting with master instead of default one

-S <PATH>, -o mfssubfolder\=<PATH>
   mount specified LizardFS directory (default is /, i.e. whole filesystem)

-o enablefilelocks=[0,1]
   enables/disables global file locking (disabled by default)

-o mfspassword=<PASSWORD>
   authenticate to LizardFS master with <PASSWORD>

-o mfsmd5pass=<MD5>
   authenticate to LizardFS master using directly given <MD5> (only if
   mfspassword option is not specified)

-o mfsdelayedinit
   connection with master is done in background - with this option mount can
   be run without network (good for running from fstab / init scripts etc.)

-o mfsacl
   enable ACL support (disabled by default)

-o mfsaclcacheto=<SEC>
   set ACL cache timeout in seconds (default: 1.0)

-o mfsaclcachesize=<N>
   define ACL cache size in number of entries (0: no cache; default: 1000)

-o mfsrwlock=<0|1>
   when set to 1, parallel reads from the same descriptor are performed
   (default: 1)

-o mfsmkdircopysgid=<N>
   sgid bit should be copied during mkdir operation (on linux default: 1,
   otherwise: 0)

-o mfssugidclearmode=<SMODE>
   set sugid clear mode (see below)

-o mfsdebug
   print some LizardFS-specific debugging information

-o mfscachemode=<CACHEMODE>
   set cache mode (see DATA CACHE MODES; default is AUTO)

-o mfscachefiles
   (deprecated) preserve file data in cache (equivalent to -o
   mfscachemode='YES')

-o mfsattrcacheto=<SEC>
   set attributes cache timeout in seconds (default: 1.0)

-o mfsentrycacheto=<SEC>
   set file entry cache timeout in seconds (default: 0.0, i.e. no cache)

-o mfsdirentrycacheto=<SEC>
```

(continues on next page)

(continued from previous page)

```
   set directory entry cache timeout in seconds (default: 1.0)

-o mfswritecachesize=<N>
   specify write cache size in MiB (in range: 16..2048 - default: 128)

-o mfscacheperinodepercentage=<N>
   specify what part of the write cache non occupied by other inodes can a
   single inode occupy (measured in %).
   E.g. When N=75 and the inode X uses 10 MiB, and all other inodes use 20
   MiB out of 100 MiB cache, X can use 50 MiB more (since 75% of 80 MiB is
   60 MiB).
   Default: 25.

-o mfschunkserverreadto=<MSEC>
   set timeout for whole communication with a chunkserver during read
   operation in milliseconds (default: 2000)

-o mfschunkserverwriteto=<MSEC>
   set chunkserver response timeout during write operation in milliseconds
   (default: 5000)

-o mfschunkserverrtt=<MSEC>
   set timeout after which SYN packet is considered lost during the first
   retry of connecting a chunkserver (default: 200)

-o mfschunkserverconnectreadto='MSEC'
   set timeout for connecting with chunkservers during read operation in
   milliseconds (default: 2000)

-o mfschunkserverwavereadto='MSEC'
   set timeout for executing each wave of a read operation in milliseconds
   (default: 500)

-o mfschunkservertotalreadto='MSEC'
   set timeout for the whole communication with chunkservers during a read o
   operation in milliseconds (default: 2000)

-o mfsrlimitnofile='N'
   try to change limit of simultaneously opened file descriptors on startup
   (default: 100000)

-o mfsnice='LEVEL'
   try to change nice level to specified value on startup (default: -19)

-o mfswriteworkers='N'
   define number of write workers (default: 10)

-o mfswritewindowsize='N'
   define write window size (in blocks) for each chunk (default: 15)

-o cacheexpirationtime=MSEC
   set timeout for read cache entries to be considered valid in milliseconds
   (0 disables cache) (default: 0)

-o readaheadmaxwindowsize=KB
   set max value of readahead window per single descriptor in kibibytes
   (default: 0)
```

(continues on next page)

(continued from previous page)

```
-o mfsmemlock
   try to lock memory (must be enabled at build time)

-o mfsdonotrememberpassword
   do not remember password in memory - more secure, but when session is lost
   then new session is created without password

-o mfsioretries='N'
   specify number of retries before I/O error is returned (default: 30)

-o mfsreportreservedperiod='N'
   specify interval of reporting reserved inodes in seconds (default: 60)

-o mfsiolimits='PATH'
   specify local I/O limiting configuration file (default: no I/O limiting)

-o symlinkcachetimeout=<N>
   Set timeout value for symlink cache timeout in seconds. Default value is
   3600.

-o bandwidthoveruse=<N>
   Define ratio of allowed bandwidth overuse when fetching data. Default
   value is 1.25. This option is effective only with N+M goals (xors and
   erasure codes).
```

General mount options (see mount(8) manual):

-o rw, -o ro	Mount file-system in read-write (default) or read-only mode respectively.
-o suid, -o nosuid	Enable or disable suid/sgid attributes to work.
-o dev, -o nodev	Enable or disable character or block special device files interpretation.
-o exec, -o noexec	Allow or disallow execution of binaries.

REPORTING BUGS

Report bugs to <contact@lizardfs.org>.

COPYRIGHT

Copyright 2008-2009 Gemius SA, 2013-2016 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

mfsmaster(8), lizardfs(1), lizardfs(7), mount(8)

4.11.31 mfsmount.cfg(5)

NAME

mfsmount.cfg - configuration file for the mfsmount command

SYNTAX

```
OPTION,OPTION,OPTION,...

OPTION

PATH
```

The # character starts comments.

DESCRIPTION

The *mfsmount.cfg* file defines the default mountpoint and the default mfsmount specific mount options for the mfsmount command. Options can be either stated in one line as a comma separated list of options, or by giving one option per line. Options not mfsmount specific will be ignored.

The default mount point should be listed on a separate line, giving the full path. Lines starting with the # character are ignored.

EXAMPLES

```
#
# Default Options
#
mfsmaster=192.168.1.1
mfspassword=changeme
#
# Default mount point
#
/mnt/lizardfs
```

REPORTING BUGS

Report bugs to <contact@lizardfs.org>.

COPYRIGHT

Copyright 2016 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

mfsmount(1) lizardfs(7)

4.11.32 lizardfs-cgiserver(8)

NAME

lizardfs-cgiserver - simple HTTP/CGI server for Lizard File System monitoring

SYNOPSIS

```
lizardfs-cgiserver [-p PIDFILEPATH] [-H BIND-HOST] [-P BIND-PORT] [-R ROOT-PATH] [-v]
lizardfs-cgiserver* -h
```

DESCRIPTION

lizardfs-cgiserver is a very simple HTTP server capable of running CGI scripts for Lizard File System monitoring. It just runs in foreground and works until killed with e.g. SIGINT or SIGTERM.

OPTIONS

-h	print usage information and exit
-H <BIND_HOST>	local address to listen on (default: any)
-P <BIND_PORT>	port to listen on (default: 9425)
-R <ROOT_PATH>	local path to use as HTTP document root (default is CGIDIR set up at configure time)
-p PIDFILEPATH	pidfile path, setting it triggers manual daemonization
-v	log requests on stderr

COPYRIGHT

2013-2017 Skytechnology Sp. z o.o.

LizardFS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

LizardFS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LizardFS. If not, see <http://www.gnu.org/licenses/>.

SEE ALSO

lizardfs(1),lizardfs(7)

CHAPTER 5

LizardFS Developers Guide

A guide for people wanting to work on developing LizardFS.

Developers and users can contribute to the LizardFS development in many ways and everyone is welcome in the project.

Users can, for instance, help the development by testing beta releases of the software, reporting and testing of bugs, writing documentation, translate man pages and documentation to your native language, spreading the word and promote LizardFS. In this guide we will be concentrating in helping people wanting to help with the development and documentation of LizardFS.

Contents:

5.1 Obtaining and installing LizardFS from Source

5.1.1 Installing LizardFS from source

The current LizardFS source code can be obtained from our *github* (https://github.com/lizardfs/lizardfs) project page. You can either download a tarball from there by choosing the respective version in the **Branch** tab on the left or use *git* to clone the sourcetree.

LizardFS uses *CMake* as its build system. To compile the sources, follow the directions outlined below.

1. Create a build directory inside the source directory:

   ```
   cd lizardfs-source
   mkdir build
   ```

2. Run

   ```
   cmake ..
   ```

   inside the build directory. Useful options include -DCMAKE_INSTALL_PREFIX, -DCMAKE_BUILD_TYPE as well as various LizardFS-specific “-DENABLE_<something_or_other>” options. Options are listed when cmake is ran and can be changed by re-running cmake:

```
cd build
cmake .. -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/opt/lizardfs
```

3. Run make in the build directory:

```
make
```

4. Run make install to install files (you may need to be root):

```
make install
```

Now you have a full installation of LizardFS from sourcecode.

For build instructions on operating systems other than Linux, please refer to *Setting up a development workspace* .

5.2 Setting up a development workspace

5.2.1 General

LizardFS does not have too many dependences, so it is relatively simple to develop on any platform you would like.

5.2.2 On Linux

5.2.3 On Mac OS/X

Compiling software like LizardFS on MacOS/X requires some additional software to be installed on your Mac. You will need to install *Xcode* from Apple and than issue:

```
xcode-select --install
```

to add the command line tools to your system.

To get all the goodies from the current LizardFS, you will require to build LizardFS with *gcc* 6 and the latest *OSXFuse* library.

We have had good experiences with using *homebrew* for adding open source software to MacOS/X and would like to recommend to developers to use it to add all additional software required.

To install homebrew, issue the following at your command prompt:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪master/install)"
```

Than to install *CMake* issue:

```
brew install cmake
```

and finally gcc6 with:

```
brew install homebrew/versions/gcc6
```

To generate manpages please also add the *Pandoc* document format translator:

```
brew install pandoc
```

And last but not least, if you would like to generate documentation as well, install the sphinx documentation engine:

```
brew install sphinx-doc
```

As on any other OS we have no preferences for a IDE on MacOS/X. You can use Xcode, eclipse, netbeans or whatever else fickle's your fancy.

Our Documentation maintainer uses Sublime Text 3 and swears that it is the best editor since the invention of writing, but YMMV ;)

To build with your installed gcc6 you will need to set the following variables in your environment before invoking *CMake*:

```
export CC=/usr/local/bin/gcc6
export CXX=/usr/local/bin/g++-6
export CPP=/usr/local/bin/gcc6
```

homebrew is also the perfect place to get git and additions to git and gerrit.

Some people had good experiences using SmartGIT but its not free.

Now you are ready to compile a fully featured LizardFS package on your Mac.

5.2.4 On FreeBSD

Note: At the time of the editing of this article, FreeBSD 11 is about to be released so all the instructions are for FreeBSD 11.

To create a working development environment on FreeBSD there are a range of ports or packages you will need to install:

```
gcc6
binutils
bash
gmake
cmake
git
```

The packages will install other dependencies required automagically.

For building the manpages and documentation you will require additionaly:

```
hs-pandoc
hs-pandoc-types
py27-sphinx-1.4.4
```

For linking to the right gcc version, you should set:

```
export LDFLAGS=-Wl,-rpath=/usr/local/lib/gcc6
```

in your environment.

For making bash work correctly, please add the following to /etc/fstab:

```
fdesc           /dev/fd         fdescfs rw      0   0
```

Before you can build LizardFS with your newly setup build envionment, please set the following variables in your environment or add them to your .bashrc:

```
export CC=/usr/local/bin/gcc6
export CXX=/usr/local/bin/g++6
export CPP=/usr/local/bin/gcc6
export MAKE=/usr/local/bin/gmake
export SHELL=/usr/local/bin/bash
```

We also strongly suggest to build LizardFS while working inside bash.

5.3 LizardFS functional testing

5.3.1 How to set up a testing environment?

Functional tests configure and run testing LizardFS installation several times. The testing installation runs all system modules on a single computer.

To prepare the testing environment safely, please use the following script:

```
<SOURCE_DIRECTORY>/tests/setup_machine.sh
```

You will need root permissions to use it. While running it, you may want to provide directories which will be used by the chunkservers during the tests to store any generated data as an arguments to this script, eg:

```
sudo setup_machine.sh setup /mnt/hdb /mnt/hdc /mnt/hdd /mnt/hde
```

These directories may be located on one physical disk. Long tests, however, may not work in such a configuration (they generate heavy traffic). It's OK for sanity checks.

You will need to have about 60 GB of disk space available for the tests. At least three directories have to be provided in order to run some of the test cases.

If you want to know what 'setup_machine.sh' does, just run it without any command line arguments. It will print a short description of all the changes it is going to make.

The tests need to know which binaries should be tested. The default location is /var/lib/lizardfstest/local and this means eg. that the master server will run code from the /var/lib/lizardfstest/local/sbin/mfsmaster file. You can change the default directory by adding an entry like the following to the /etc/lizardfs_tests.conf file:

```
: ${LIZARDFS_ROOT:=/home/you/local}
```

Make sure that the user lizardfstest is able to read files from this directory.

5.3.2 How to compile the tests?

To use tests, LizardFS has to be compiled with an option ENABLE_TESTS, e.g.:

```
cmake .. -DENABLE_TESTS=YES
```

The test installation has to be prepared. The default location for tests is '$HOME/local' but it can be changed in the tests config file, e.g.:

```
: ${LIZARDFS_ROOT:="$HOME/LizardFS"}
```

After that, the tests are ready to be run.

5.3.3 How to launch the tests?

Test binary is installed together with the file system. The path leading to it is the following:

```
<INSTALL_PREFIX>/bin/lizardfs-tests
```

Alternatively, it can be run from build directory:

```
<SOURCE_DIRECTORY>/tests/lizardfs-tests
```

This binary runs all the bash scripts using googletest framework. This makes it easy to generate a report after all the tests are finished. Part of the source code of the 'lizardfs-tests' binary (list of tests) is generated automatically when tests are added or removed. Launched without any parameters, it executes all available tests.

By using the parameter '–gtest_filter' you can choose which tests are to be run. The above parameter uses a pattern to select the tests, e.g.:

```
./lizardfs-tests --gtest_filter='SanityChecks*'
```

The command above runs all tests from the 'SanityChecks' suite.

A 'lizardfs-tests' binary uses an interface delivered by googletest which is displayed after issuing:

```
./lizardfs-tests --help
```

5.3.4 How to make your own test?

To create a new test simply create a file like this:

```
<SOURCE_DIRECTORY>/tests/test_suites/<SUITE>/test_<TEST_NAME>.sh
```

where:

<SUITE> is a test suite that you want your test to be a part of. Existing test suites:

- **SanityChecks** basic tests of LizardFS functionality; these tests should last a couple of minutes,
- **ShortSystemTests** advanced tests of LizardFS functionality not requiring dedicated storage; these tests last around half an hour,
- **LongSystemTests** advanced tests of LizardFS functionality requiring dedicated storage; these tests last a few hours,
- **ContinuousTests** tests using LizardFS mountpoint to populate LizardFS installation with data and validate them; these tests are desigend to be run continuously,

<TEST_NAME> is the name of your test.

Tests which you write are run inside a kind of sandbox. They use the UID of a special user 'lizardfstest', so you can safely spawn any processes (they will be killed after the test).

Our testing framework provides numerous bash functions and structures. They are implemented in the following directory:

```
<SOURCE_DIRECTORY>/tests/tools
```

Merry testing!

5.4 Participation Rules

5.5 Submitting Patches

If you want to improve LizardFS, we are happy to accept your changes. You should use *github* Pull Requests to make your changes available for review.

You can read about this here: https://help.github.com/articles/about-pull-requests/ how to use them.

Please make sure to follow our *Versioning* which are part of the next chapter.

5.6 Repository rules

- We keep the master stable enough to pass all our tests. If your change is spitted into many commits, each of them should be complete enough not to break any of them.
- We do strive to maintain backward compatibility. Changes which make existing installations break during an upgrade, won't be accepted.
- Your changes most probably will be rebased to the top of the master before being pushed to the repository. This makes the git tree simpler if there is no important reason to add any branches and merges there.

5.6.1 Commit messages

- We document all the changes in commit messages. Please make your commit messages consistent with the rest of git log. Especially:

  The first line must contain a summary of the change. It has to be no more than 65 characters (this is a hard limit). It has to be in the following format:

  ```
  tag: Description in the imperative mood.
  ```

  The description begins with a capital letter. It's role is describe what the commit does (does it fix a bug? refactor some code? adds a new test? adds a new feature? extends an existing feature? updates documentation?) and which functionality, part of the code or module does it concern.

  Some real-life examples:

  ```
  debian: Add Jenkins BUILD_NUMBER to package version

  deb, rpm, cmake: Fix building packages

  mount: Fix returning wrong file length after truncate

  ha-cluster: Make the cluster manager more user friendly

  admin: Rename lizardfs-probe to lizardfs-admin

  ```

(continues on next page)

(continued from previous page)

```
master,mount: Add option allowing non-root users to use meta

doc: Fix inconsistencies in ha-cluster README

master: Clean up code in chunks.cc

master, metalogger: Fix memory leak found by valgrind

utils: Fix handling LIZ_MATOCL_METADATASERVER_STATUS in wireshark

tests: Refactor add_metadata_server_ function
```

After the summary there has to be a blank line followed by the rest of the commit message, wrapped at 72 or 80 characters, possibly containing multiple paragraphs. It should contain a comprehensive information about what and how the commit actually changes, why this change is being done, why this is the right way to do this, how to replicate bugs fixed be it.

Use present tense. tell what the change does. be terse. avoid "decorations" like dashes and brackets that waste space.

Put all the information required to review the commit there. Generally, we recommend following the same rules as suggested by maintainers of OpenStack here: https://wiki.openstack.org/wiki/GitCommitMessages#Information_in_commit_messages.

- Before creating your first commit messages, read the git log and try to follow any good practices found there. You should put all the information in the commit message even though it might be present in some gitHub issue or in the pull request itself – GitHub might disappear some day!
- Use reasonable numbers of commits (possibly one) per change. Don't create commits which fix many independent things. Don't split a single change into many commits if there is no reason to do this.

5.6.2 Code review process

Most of the review process of LizardFS is done using Gerrit (http://cr.skytechnology.pl:8081/). This is why most changes include a Change-Id: tag. You don't need it for changes committed via Pull Requests. Those will be reviewed and imported by the core development team.

Your patch will typically be reviewed within 1-2 days. You can follow the progress in gerrit and you will also receive mail whenever there is a change.

In general 3 things can happen in the review:

- The committer reviewed and tested the patch successfully, then merged it to master (congratulations)
- The committer had some comments, which you need to look at
- Sometimes the patch breaks some other functionality and is marked as "Cannot merge"

In the 2 later cases, you need to update your patch.

5.6.3 Use of Code-Review and Verified

If you look at your patch on [gerrit] you will see two status codes:

Code-Review

Verified

The reviewers, our CI system (jenkins) and potentially yourself will use these two fields to qualify the patch.

Code-Review Can be assigned -2, -1, 0, +1, +2

> **-2** are to be used by the author, to signal "work in progress". The -2 prevent the patch from being merged, and only the person who issued the -2 can remove it.
>
> If you work on a larger patch, you are most welcome to upload a patch, mark it as -2, to see if it builds successfully on all platforms
>
> **-1** is used by the reviewer, if there are things in the patch that should be changed
>
> **0** is used when making comments, that has no effect on whether or not the patch should be merged.
>
> **+1** is used by the reviewer, to signal the patch is ok, but the reviewer would like a second opinion
>
> **+2** is used by the author to signal no review is needed (this can only be done by core developers, and should be used with care). The person who merges your patch, will use +2, just before merging, since only +2 can be merged

Note: a patch will NOT be merged as long as there are -1 or -2 unresponded to.

Verified Can be assigned -1, 0, +1

> **-1** is used by the CI system if the build fails (remark this is not always a problem in your patch, but can be due to a broken master).
>
> **-1** is used by the reviewer, if the expected result cannot be seen.
>
> **0** is used when making comments, that has no effect on whether or not the patch should be merged.
>
> **+1** is used by the CI system if the build is succesfull
>
> **+1** is used by the reviewer, when the expected result has been verified.

Note: a patch will NOT be merged unless the CI system shows a successfull build.

5.6.4 Updating a patch

Checkout your branch:

```
git checkout test1
```

make the needed changes and test them. It is polite to comment the lines of code you do not want to change or where you do not agree with the committer, this is done directly in gerrit.

Once you are ready to commit again it is important you use –amend

```
git commit --amend -a
```

Note: do not use the -m parameter as it will wipe out the gerrit patch id. Let git open an editor, allowing you to edit the commit message (or leave it unchanged). When editing be careful not to remove/ modify the last line with the patch id.

This will ensure you update the patch, instead of generating a new one (with a new Change-Id:).

5.6.5 Closing issues with commits

If a commit fixes some GitHub issue, please add a line Closes #xxx at the end of the commit message, where #xxx is the issue's ID (e.g., Closes #123). It would link the commit with the issue and the issue with the commit (like in 7dc407da)

5.6.6 Other rules

- When changing anything already documented, also update its man pages.
- New features have to be covered by our test suite and documented in man pages.
- For bug fixes it's recommended to provide a test which verifies if the bug is indeed fixed.

5.7 Versioning and Release Engineering

5.7.1 Versioning

LizardFS uses the following versioning:

```
X.Y.Z
```

for releases

or

```
X.Y-rc
```

for release candidates

where X gives the main release number, the YY digits give the subrelease number and Z gives the microrelease version.

- X can be any natural number.
- Y is an even number for stable releases and an odd number for unstable tags.
- Z gives the subfix tag microrelease and can be any natural number or the string "rc" which indicates a "release candidate"

Master branch

The Master branch is always in change and represents the actual state of development. It is being daily checked for completeness, and has to pass smoke, build and load testing in our lab envitonment.

Unstable tags

Simultanous with creating a new release candidate we are tagging the current master state as the current "unstable".

Milestones

Milestones are defined in the github milestones page which can be found at the following address: https://github.com/lizardfs/lizardfs/milestones. The RE team decides in monthly meetings if the current milestone state represent a release candidate definition. Once this is agreed upon, feature freeze is decided and a branch created for the decided upon stable release development, named X.Y.. Within there the current state is tagged as X.Y-rc. Once the release candidate reaches a define stable state, X.Y.0 is tagged and packages are being build for the public to use.

Releases and stable branches

Release branches are created as -rc microrelases for development. A -rc microrelease is deemed as the "in preparation release candidate" microrelease of a stable branch. X.Y is defined as the first stable release of a branch and is than released to packaging and publishing to the respective repositories or release to the respective distribution maintainer.

Microreleases

To allow for fixes to major bugs we added the microrelease tag to the stable branches. A microrelease will contain no new features, no additional functionality but just fixes to major bugs.

5.7.2 Release Engineering

Releases / stable branches have to pass the full sequence of short and long testing in our lab plus go through some real life production runs at our coporate sponsors. LTS releases get additional testing from partners running larger LizardFS installation base. Once a release is marked stable, every microrelease of it has to pass the same testing cycle the stable release had to pass.

5.7.3 Release Cycles

Stable Releases: 6 months

For commerical customers Long Term Support releases are available from Sky Technologies Sp. z o.o.

5.8 Dummy-Bughunt

5.9 Documentation-HowTo

Welcome to the documenters-project for lizardfs. This pages shall introduce you to our Tools and Workflows.

Documentation is written in reStructuredText (http://docutils.sourceforge.net/rst.html), then we build the different Formats (html, pdf) with the Help of sphinx (http://www.sphinx-doc.org/en/stable/tutorial.html).

If you want to generate nroff man pages as well, you will also need to install pandoc (http://pandoc.org/).

5.9.1 Editorial Workflow

To ease the authoring-process and to uphold the quality of our documentation every section goes through at least the following stages:

1. **todo** (this section needs a helping hand)

2. **writing** (somebody is writing content, others: please dont interfere)
3. **proof1** (proofreading stage 1, somebody with appropriate knowledge checks the written content for correctness regarding facts and understandability)
4. **proof2** (proofreading stage 2, somebody - preferably a native speaker - checks the content for grammar, spelling etc. No content/fact-changes here, if necessary set state back to proof1 and add a todo-remark)
5. **frozen** (this section is done, find another one to work on, there is plenty of work available :)

While authoring the status of a section can go forth and back, writing is not a oneway-road…

The implementation of those stages is done with the comment-directive of rst. Put a line below the section-title like:

```
.. auth-status-todo/none
.. auth-status-writing/<my-own-email>
.. auth-status-proof1/<my-own-email>
.. auth-status-proof2/<my-own-email>
.. auth-status-frozen/none
```

the part after the "/" indicates *who* is working on the respective section. If you just finished a stage and want to indicate the next stage is to be done by "somebody", use "none" instead of an email-address.

All in all a section-start should look like:

```
Section-Title
=============

.. auth-status-writing/wolfram@example.com
```

Note: This mechanism is meant to make our lives easier, *please* dont use it for commanding others! The only one who is entitled to put an email-addres in a authoring-tag is the person owning that address. It is used to indicate "i take responsibility for that section". Misusing mechanisms for command-like "you take responsibility" actions will definitly kill our motivation.

5.9.2 ToDo notes

These notes can be used anywhere in the documentation as anonymous reminders. It is a good idea to also note keywords for content to be written as todo note when they come to your mind. ToDo notes have the syntax:

```
.. todo:: explain something

.. todo::
   explain something complex
   * explain part 1
   * explain part 2
   * explain part 3
```

ToDo notes go to the documentation in the place where they are written and to the ToDo-List (you need to "make clean" to generate a new ToDo-List). It is easy to generate documentation while leaving todo-notes out. To do that find the line:

```
todo_include_todos = True
```

in conf.py of the documentation-directory and change its value to "False" before generating the docs.

5.9.3 Styleguide / Organizing Files

Headers are taken from the Python documentation project:

```
Parts:            ############## (with overline)

Chapters:         ************** (with overline)

Sections:         ===========

Subsections:      -----------

Subsubsections:   ^^^^^^^^^^^

Paragraphs:        """"""""""
```

- Table of Content (ToC) depth: 2 Levels (Also 2 levels in every Part except for the glossary)
- Subdirectories are for separate manuals that should be buildable standalone.
- The manual pages should be buildable as man pages standalone.
- Add .. code-block:: <lang> to literal blocks so that they get highlighted. Prefer relying on automatic highlighting simply using :: (two colons). This has the benefit that if the code contains some invalid syntax, it won't be highlighted. Adding .. code-block:: python, for example, will force highlighting despite invalid syntax.
- In section titles, capitalize only initial words and proper nouns.
- Wrap the documentation at 80 characters wide, unless a code example is significantly less readable when split over two lines, or for another good reason.

5.9.4 Writing style

When using pronouns in reference to a hypothetical person, such as "a user with a session cookie", gender neutral pronouns (they/their/them) should be used. Instead of:

```
he or she... use they.
him or her... use them.
his or her... use their.
his or hers... use theirs.
himself or herself... use themselves.
```

5.9.5 Installing the documenters Tools on different Platforms

debian8 (jessie)

The best way to get the documentation formatting-tools up and running is:

First:

- apt-get install pandoc pandoc-data

Than (the sphinx in the debian repo is too old) :

- choose a place for your virtual environment for sphinx
- setup a virtual enironment for sphinx:

```
$ virtualenv sphinx
```

- activate your virtual python environment:

```
$ source sphinx/bin/acticate
```

- install the newest sphinx and tools:

```
$ pip install sphinx sphinx-autobuild
```

This should be enough to build the html-documentation. If you want pdf as well you will need texlive/pdflatex - caution, that one is really a *large* set of software.

Mac/OS X

If you are following our *On Mac OS/X* recommendations and use *homebrew* the installation of the tools required to compile manpages is pretty simple. Just issue:

```
brew install pandoc
```

and you are all set up. To compile the documentation some additional work has to be done.

To install sphinx correctly on MacOS/X you will need to make use of a virtual python environment:

```
* choose a place for your virtual environment for sphinx
* setup a virtual eironment for sphinx::

  $ virtualenv sphinx

* activate your virtual python environment::

  $ source sphinx/bin/acticate

* install the newest sphinx and tools::

  $ pip install sphinx sphinx-autobuild
```

FreeBSD

For building the manpages and documentation you will require the following packages:

```
hs-pandoc
hs-pandoc-types
py27-sphinx-1.4.4
```

FreeBSD keeps its sphinx prety actual so there should be no problem in building the docs.

5.9.6 Build-Process

Before you can build something you will need to get the lizardfs sources. Read *Installing LizardFS from source* whilst ignoring the stuff about cmake. When you pulled the sources from github look for a subdirectory named “docs”, this is where the current documentation lives - everything happens here,

There is a Makefile in the repo for building documentation. It is derived from the one generated by sphinx-quickstart.

To build the html-documentation in one huge html-file (plus images):

```
make singlehtml
```

To build the html-documentation splitted up to different files:

```
make html
```

To build the pdf-documentation:

```
make latexpdf
```

If things go wrong when building the documentation, first check if all the tools for your selected target-format are available on your system (check terminal-output for "command not found"-messages)

5.9.7 Submitting your work

Read *Submitting Patches* for information about our conventions. In short this means for us:

- First line of the commit-message is preceeded by "doc: ", The first character of the remaining line has to be uppercase. First line is of form:

  ```
  doc: Short-description max-length 65 characters
  ```

- After the first line of the commit-message add a blank-line
- third part of the commit-message is a longer description with lines of 80 characters maximum-length.

A complete commit-message looks like the following:

```
doc: Fixed references to other documentation

The refs to adminguide/basic_configuration and to filesystem/georeplication
had a wrong syntax.
```

5.10 Ressources

5.10.1 General

Wikipedia article https://en.wikipedia.org/wiki/LizardFS

LizardFS review by Valentin Hörbel from NFON AG https://www.itcentralstation.com/product_reviews/lizardfs-review-34235-by-valentin-hobel

LizardFS Youtube channel https://www.youtube.com/channel/UCyDn7vchGZxDUyr-CajFzfg

LizardFS Twitter channel https://www.twitter.com/lizardfs/

LizardFS on Facebook https://www.facebook.com/lizardfs/

LizardFS users mailing list on Sourceforge https://sourceforge.net/p/lizardfs/mailman/?source=navbar

LizardFS on irc #lizardfs on freenode.net

LizardFS article on golem.de in german http://www.golem.de/news/lizardfs-software-defined-storage-wie-es-sein-soll-1604-11951
html

5.10.2 Admin specific

5.10.3 Developer Specific

Main code repository http://cr.skytechnology.pl:8081/lizardfs

GitHub repository mirror https://github.com/lizardfs/lizardfs

Code review system http://cr.skytechnology.pl:8081/

Continuous integration system http://jenkins.lizardfs.org/

Roadmap https://github.com/lizardfs/lizardfs/wiki/Roadmap

5.10.4 Third Party AddOns

puppet-lizardfs

The puppet-lizardfs module lets you use Puppet to install and configure LizardFS automatically.

You can configure with puppet-lizardfs:

- The LizardFS master (ready for High-availability with tools like keepalived or Pacemaker. Check out the explanation below)
- The LizardFS chunkserver
- The LizardFS metalogger
- The LizardFS client and mount points

Author: Asher256

Github repository: https://github.com/Asher256/puppet-lizardfs

Puppet Forge page: https://forge.puppet.com/Asher256/lizardfs/readme

lizardfs ansible playbook

Ansible playbook for automated installation of LizardFS Master, Shadowmaster, multiple Chunkservers, Metalogger and CGIserv.

Author: stenub

Github repository: https://github.com/stenub/lizardfs_ansible_playbook

5.10.5 Packages from the community

Archlinux https://aur.archlinux.org/packages/lizardfs/

CHAPTER 6

Dummy-Technical-TOC

CHAPTER 7

LizardFS Frequently Asked Questions

Contents:

7.1 General Questions

Why is it called LizardFS ? It's a metaphor for something that can grow back if partially lost

Can I move my systems from MooseFS to LizardFS ? Yes. This is documented in the *Upgrading Lizardfs* manual.

Why a fork from MooseFS ?

- At the time we did the fork there was nearly no movement in the GIT repo of MosseFS
- Only 2 devs had access to the sourceforge repo and there were no more committers
- increase in the number of possible contributors with a simultaneous lack of response on the part of MooseFS maintenance

What kind of erasure coding does LizardFS use ? Reed-Solomon

Is there support for different storage sizes in one lizardFS deployment ? Yes

Is there support for different network bandwidth in one lizardFS deployment? Yes. You can have different chunkservers having different bandwidth.

As for different bw per client, check out: *Configuring QoS* .

Is there support for different RAM amount in one lizardFS deployment? Yes. Different chunkservers can have different amounts of RAM.

Is there support for encryption, if yes what kind? Encryption of any kind supported by the platform you run your master and chunkservers on is supported since we make use of the underlying POSIX filesystems to store the bits and pieces.

How are the deletes managed, if there's a garbage collector? Deleted files are sent to trash and removed when trashtime expires

Are the meta data servers are "inside" lizard or "outside"? Inside

7.2 Server related

How do I completely reset a cluster ? The simplest way is to create a new metadata file. Go to your metadata directory on your current master server (look at the DATA_PATH in the mfsmaster.cfg file), than stop the master and create a new empty metadata file by executing:

```
echo -n "MFSM NEW" > metadata.mfs
```

Start the master and your cluster will be clear, all remaining chunks will be deleted.

How do I create a full backup of all my metadata (in case of recreating a master etc. . .) Copy your data directory somewhere safe (default path: /var/lib/mfs).

Files you should be interested in keeping are primarily:

- **metadata.mfs** - your metadata set in binary form. This file is updated hourly + on master server shutdown. You can also trigger a metadata dump with lizardfs-admin save-metadata HOST PORT, but an admin password needs to be set in the mfsmater.cfg first.
- **sessions.mfs** - additional information on user sessions.
- **changelog*.mfs** - changes to metadata that weren't dumped to metadata.mfs yet.

How do I speed up/slow down the main chunk loop execution during which chunks are checked if they need replication/deletion Adjust the following settings in the master server configuration file:

```
CHUNKS_LOOP_PERIOD
    Time in milliseconds between chunks loop execution (default is 1000).

CHUNKS_LOOP_MAX_CPU
    Hard limit on CPU usage by chunks loop (percentage value, default is 60).
```

7.3 Dummy-Clients

7.4 Dummy-Platforms

7.5 Dummy-Networking

7.6 High Availability

What do you mean by High Availability? With the help of multiple chunk servers and good goals, files can be stored multiple times. Therefore, a certain level of high availability on a file level can be achieved easily.

In addition, it is important to know that per default, the master service only can be active in a master role on one node at the time. If this node fails, e.g. because of broken hardware or out-of-memory situations, the current master has to be demoted (if still possible) and an existing shadow has to be promoted manually.

If the failover happens automatically, a good state of high availability is achieved on a service level. Thus the term "High Availability" refers to keeping the master role alive when everything goes down under.

How can I achieve High Availability of the master? There are multiple ways of keeping the master highly available.

One would be to demote and promote manually if you need to. The better way would be to delegate that task to a mechanism which knows the current state of all (possible) master nodes and can perform the failover procedure automatically.

Known methods, when only using open-source software, are building Pacemaker/ Corosync clusters with self-written OCF agents. Another way could be using keepalived.

This is too complicated! I need a better solution for High Availability.

An officialy supported way to achieve high availablity of the master is to obtain the uRaft component from Skytechnology Sp. z o.o., the company behind LizardFS. Based on the *raft* algorithm, the uRaft service makes sure that all masternodes talk to each other and exchange information regarding their health states.

In order to ensure that a master exists, the nodes participate in votes. If the current master fails, uRaft moves a floating IP from the formerly active node to the new designated master. All uRaft nodes have to be part of one network and must be able to talk to each other.

The uRaft component can be obtained by signing a support contract with SkyTechnology Sp. z o.o..

See also:

raft

lizardfs_ha_clusterr

CHAPTER 8

LizardFS'ers CookBook

A collection of HowTo's, tips, tricks and solutions for installing, configuring, running and maintaining LizardFS. For short quick answers to frequently asked questions, please consult *LizardFS Frequently Asked Questions*.

Warning: Articles marked unsupported are totally unsupported and may result in data damage, data loss and frying your brain cells. You have been warned.

Contents:

8.1 Dummy-Filesystems

8.2 LizardFS linux CookBook

8.2.1 Setting DirectIO for your setup

In some cases we have experienced that the caching mechanism in some systems may slow down performance significantly. What has helped is switching the caching off and moving to Direct IO which ommits the OS cache and writes directly to the block device underneath.

To enable DirectIO on your installation, you need to update the *.lizardfs_tweaks* file in the root of your mounted LizardFS. This is done by issuing the following on a mounted filesystem:

```
echo "DirectIO=true" > .lizardfs_tweaks
```

You can verify if the setting has changed to true by issuing the following command:

```
cat .lizardfs_tweaks | grep DirectIO
```

If you find that this does not improve your performance or in fact, slows it down, you can always change it back by running:

```
echo "DirectIO=false" > .lizardfs_tweaks
```

The changes are effective immediately.

8.2.2 URAFT Cookbook

The HA system utilised by LizardFS to keep your master servers always alive is called uraft. It has been developed by Sky Technologies Sp. z o.o. and is based on the *raft* algorithm developed by Diego Ongaro and John Ousterhout.

HA with only 2 masters

Warning: This is unsupported and only recommended if setup by certified engineering personel.

Since it is a *quorum* based algorithm we usualy recommend to users to have 1 master and 2 shadow nodes. But there is a way to run your HA with one master, one shadow and a raft only addon on one of your chunkservers. This stub will still run a master server daemon but it will never switch it to active so it can be running anything.

All that is required to switch a node to "non master" mode is setting:

```
URAFT_ELECTOR_MODE = 1
```

in the lizardfs-uraft.cfg file. Everything else must be setup like it would be a normale lizardfs-master with uraft node except that the master will never be put into a real **master** role.

8.2.3 ZFS on Linux

ZFS is a high performance 128 bit filesystem developed by SUN Microsystems. We wil show you here the basics how to install it on Linux. For specifics how to finetune, optimize and manage zfs, please consult the links in the seealso part at the end of the ZFS articles. On Linux we use the Open-ZFS way and do not use FUSE to get maximum performance.

8.2.4 Installing ZFS on RHEL/Centos 7

To aoid all the licensing discussions (we do not get into that but you can read up on it <here https://www.softwarefreedom.org/resources/2016/ linux-kernel-cddl.html>_ if you like) the Open-ZFS project has a way where you while installing the driver compile it yourself and that way get around all the license discussions for binary modules it seems. So here we go:

You will require to add the epel repository to your system:

```
$ yum install epel-release
$ yum update
```

And than the open-zfs project repository:

```
$ yum localinstall -y --nogpgcheck http://archive.zfsonlinux.org/epel/zfs-release.el7.
↪noarch.rpm
```

after which you can install the sources required and automativaly build the required modules on your system:

```
yum install -y kernel-devel zfs
```

Test if your installation worked:

```
modprobe zfs
lsmod | zfs
```

Test if you can use the zfs commands:

```
zfs list
zpool list
```

Now you can install zpools and flesystems with ZFS.

See also:

- A guide to install and use zfs on centos 7
- The Open-ZFS Project
- ZFS Manual in the FreeBSD Handbook
- The ZFS On Linux - ZOL project supplies packages and documentation for every major distro: ZFS On Linux - ZOL
- ZFS in the Ubuntu Wiki
- How to install and use ZFS on Ubuntu and why you'd wnat to
- An extensive Guide about ZFS on Debian by Aaron Toponce
- Performance tuning instructions from the Open-ZFS Project

8.3 Dummy-freebsd

8.4 LizardFS MacOS Cookbook

8.4.1 Automounting a lizardFS volume

Since macOS is a bit restrictive starting with the ElCapitain version, automounting lizardfs requires a bit of fiddling.

First you will need to disable Apple's SIP, also known as the "rootless" feature. The rootless feature is aimed at preventing Mac OS X compromise by malicious code, whether intentionally or accidentally, and essentially what SIP does is lock down specific system level locations in the file system while simultaneously preventing certain processes from attaching to system-level processes.

SIP locks down the /System, /sbin and /usr (except for /usr/local/) directories, but we need to add a script to /sbin, so we have to temporarily disable SIP.

- Reboot the Mac and press the Command and R keys simultaneously after you hear the startup sound, this will boot OS X into Recovery Mode.
- When booted, pull down the *Utilities* menu at the top of the screen and choose *Terminal*.
- Type the following command into the terminal then hit return:

  ```
  csrutil disable; reboot
  ```

> You'll see a message saying that System Integrity Protection has been disabled and the Mac needs to restart for changes to take effect, and the Mac will then reboot itself automatically, just let it boot up as normal.

Now check if SIP is disabled, type:

```
csrutil status
```

It should give you the message:

```
System Integrity Protection status: disabled.
```

Now you are ready to add the required script to /sbin. Use your favourite editor and create the file **/sbin/mount_lizardfs** with the following contents:

```
#!/bin/bash
/usr/local/bin/mfsmount $@ 2>>/dev/null
sleep 3
```

Save the file. Now lets add the required things to your automounter.

In **/etc/auto_master** add the folowing line:

```
/-      auto_lizardfs
```

Save the file and now create a file called **/etc/auto_liazrdfs** with the following contents for the directory you want to mount:

```
DIRECTORY -fstype=mfs,rw,big_writes,mfsmaster=IP_OF_MASTER,mfsport=PORT_THE_MASTER_
↪LISTENS_ON " "
```

Note: Make sure to add an empty line to the end of the file.

example:

```
/mnt/lizardfs -fstype=lizardfs,rw,big_writes,mfsmaster=192.168.66.112,mfsport=9421 " "
```

This will automount /mnt/lizardfs from master 192.168.66.112 port 9421

Reload the automounter:

```
automount -vc
```

and test if your automounting works:

> ls -l DIRECTORY

If everything works ok, reboot again into *Recovery Mode* by pressing CMD and follow the steps to disable SIP, but put **enable** after csrutil instead of **disable**. After reboot your mac will be protected again and have a nice lizardfs automount running.

Note: The mounted filesystem will not show up in the finder. You will need to either access it from the command line or if you want to use the finder, press Shift + Command + G and enter the fodler path there manualy.

See also:

- http://useyourloaf.com/blog/using-the-mac-os-x-automounter/

- https://www.igeeksblog.com/how-to-disable-system-integrity-protection-on-mac/
-

8.5 Dummy-illumos

8.6 LizardFS Hypervisors and Virtualisation Cookbook

8.6.1 Providing storage for XenServer

Requirements:

```
LizardFS >= 3.10.4
XenServer >= 7
```

XenServer 7 is required since it makes use of CentOS 7 as the underlying OS and LizardFS provides packages for CentOS 7.

> **Warning:** Installing 3rd party components – such as the LizardFS client, and server components may invalidate any support provided by Citrix for your installation.

Pre-requisites

This guide presumes you have already installed XenServer – and are familiar with accessing the XenServer Console and using tools such as 'vi' to edit files.

You should have already configured the networking on XenServer. This could be just a single network – XenServer installations usually have an 'internet facing' network – and then one, or more 'storage networks' – make sure IP addresses are setup that you can use for connecting to, or running LizardFS on (i.e. probably not internet facing).

Prepairing Storage – if you need it

If you are going to use the XenServer itself to provide storage – you'll need to prepare drives / directories on the server.

As an example - we'll mount all the drives we want to place chunks on under '/mnt'. Once XenServer is booted – from the XenServer console you can see a list of available drives by looking in '/dev/disk/by-id' – i.e.

```
[root@XenServer-1 ~]# ls -l /dev/disk/by-id/
total 0
lrwxrwxrwx 1 root root  9 Mar 22 15:48 ata-Crucial_CT250MX200SSD1_153710904260 -> ../.
↪./sdi
lrwxrwxrwx 1 root root  9 Mar 22 15:48 ata-Crucial_CT250MX200SSD1_154110CAA644 -> ../.
↪./sdh
lrwxrwxrwx 1 root root  9 Mar 22 15:48 ata-SAMSUNG_HD204UI_S2H7J1CZ909998 -> ../../sdf
lrwxrwxrwx 1 root root  9 Mar 22 15:48 ata-SAMSUNG_HD204UI_S2H7J1CZ910008 -> ../../sde
```

By using the '/dev/disk/by-id' entries we ensure if the disk is moved to another bay – it will still mount correctly. You can use 'gpart' to partition these disks, or just use the "raw" disk to create partitions on, e.g.:

```
mkfs.ext4 /dev/disk/by-id/ata-Crucial_CT250MX200SSD1_153710904260
mkfs.ext4 /dev/disk/by-id/ata-Crucial_CT250MX200SSD1_154110CAA644
mkfs.ext4 /dev/disk/by-id/ata-SAMSUNG_HD204UI_S2H7J1CZ909998
mkfs.ext4 /dev/disk/by-id/ata-SAMSUNG_HD204UI_S2H7J1CZ910008
```

And so on, for all the drives you want to use.

Once you've formatted the disks – we need to tell XenServer where to mount them when the system comes up. First you need to create directories under the '/mnt' directory – if you keep the name of the directory the same as the drive device name, we'd do:

```
mkdir /mnt/ata-Crucial_CT250MX200SSD1_153710904260
mkdir /mnt/ata-Crucial_CT250MX200SSD1_154110CAA644
mkdir /mnt/ata-SAMSUNG_HD204UI_S2H7J1CZ909998
mkdir /mnt/ata-SAMSUNG_HD204UI_S2H7J1CZ910008
chown mfs:mfs /mnt/ata-*
```

Once this is done you can add the drives to the systems 'fstab' to ensure they are mounted automatically at boot-time – if you 'vi /etc/fstab' and then create entries such as:

```
/dev/disk/by-id/ata-Crucial_CT250MX200SSD1_153710904260 defaults,nofail 0 2
/dev/disk/by-id/ata-Crucial_CT250MX200SSD1_154110CAA644 defaults,nofail 0 2
/dev/disk/by-id/ata-SAMSUNG_HD204UI_S2H7J1CZ909998 ext4 defaults,nofail 0 2
/dev/disk/by-id/ata-SAMSUNG_HD204UI_S2H7J1CZ910008 ext4 defaults,nofail 0 2
```

The 'nofail' option means that the system continues to boot even if the disk or disks are unavailable. Once this is done you can mount all of those disks with:

```
mount -a
```

Installing LizardFS

Before you can install any of the LizardFS components we need to tell the XenServer nodes where to get the LizardFS RPM's from.

For each node in the system (or any nodes you add) you need to log in to the XenServer console on the node and run once:

```
curl http://packages.lizardfs.com/yum/el7/lizardfs.repo > /etc/yum.repos.d/lizardfs.
↪repo
```

This fetches the LizardFS repository details to somewhere XenServer can find them (with the 'yum' command).

LizardFS Client

To connect XenServer to LizardFS you need to install the 'lizardfs-client'. Even if you're not installing full LizardFS on XenServer you still need to install the client package as well as the FUSE library package.

The repo file XenServer points to packages that are no longer present on the CentOS site (they've been moved to the CentOS 'Vault') – so we need to adjust the URL so the system can pull in FUSE as part of the 'lizardfs-client' install.

Edit the file '/etc/yum.repos.d/CentOS-Base.repo' – change the "[base]" URL to read:

```
baseurl=http://vault.centos.org/7.2.1511/os/x86_64
```

And save the file.

We can now install 'lizardfs-client' (which will also install FUSE) with:

```
yum --disablerepo=extras --disablerepo=updates install lizardfs-client
```

If you're just using XenServer to access another LizardFS installation (i.e. on another server / system) you don't need to add the following other software components – just skip ahead to *Client Configuration* .

LizardFS Chunk-Server and Meta-Server

If you're using the XenServer as either a 'chunk-server' (holds data) or 'meta-server' (holds meta-data) you'll need to install other components of LizardFS on the XenServer as well.

You can use the following to install the 'master' server (meta-server), chunkserver – and Admin tools respectively:

```
yum --disablerepo=base --disablerepo=extras --disablerepo=updates install lizardfs-
↪master
yum --disablerepo=base --disablerepo=extras --disablerepo=updates install lizardfs-
↪chunkserver
yum --disablerepo=base --disablerepo=extras --disablerepo=updates install lizardfs-adm
```

Setting up the Chunk Server

By now you should have the LizardFS chunk-server software installed – and your drives setup ready to hold data chunks. The LizardFS chunk-server installs with a default config – but you need to copy it into place first:

```
cd /etc/mfs
cp mfshdd.cfg.dist mfshdd.cfg
```

You'll need to edit '/etc/mfs/mfshdd.cfg' to tell the chunk-server what drives it has available. For our example we edited 'mfshdd.cfg' and added:

```
# Our Chunk Drives/Directories
/mnt/ata-Crucial_CT250MX200SSD1_153710904260
/mnt/ata-Crucial_CT250MX200SSD1_154110CAA644
/mnt/ata-SAMSUNG_HD204UI_S2H7J1CZ909998
/mnt/ata-SAMSUNG_HD204UI_S2H7J1CZ910008
```

Setting up the Meta Server ('master')

If you're running the master / meta-server under XenServer you need to make one node a 'master' and the other a 'shadow'. You will need to copy the example configs to the real files:

```
cd /etc/mfs
cp mfsmaster.cfg.dist mfsmaster.cfg
cp mfsexports.cfg.dist mfsexports.cfg
```

You need to edit '/etc/mfs/mfsmaster.cfg' one (and only one) node should have a personality of 'master' – the other should be a 'shadow'. It is also recommended in that file that you set an 'ADMIN_PASSWORD'. If the XenServer is going to be running as a master, or shadow – you'll also need to edit '/etc/mfs/mfsexports.cfg' – by default this just sets up a basic config (this is similar to an nfs exports file). Finally – you'll need to install a blank database for the 'master' server – and any shadows – this involves copying an empty database i.e.:

```
cp /var/lib/mfs/metadata.mfs.empty /var/lib/mfs/metadata.mfs
chown mfs:mfs /var/lib/mfs/metadata.mfs
```

You will only need to do this when installing the 'master' service.

See also:

- *Basic Configuration*
- *Advanced configuration*

Client Configuration

XenServer ships with a firewall – we'll need to configure that to allow LizardFS traffic to pass. To do this edit '/etc/sysconfig/iptables' – we need to add our rules before the REJECT line and COMMIT statement so you should end up with:

```
# LizardFS
-A RH-Firewall-1-INPUT -m conntrack --ctstate NEW -m tcp -p tcp --dport 9421 -j ACCEPT
-A RH-Firewall-1-INPUT -m conntrack --ctstate NEW -m tcp -p tcp --dport 9422 -j ACCEPT
-A RH-Firewall-1-INPUT -m conntrack --ctstate NEW -m tcp -p tcp --dport 9420 -j ACCEPT
-A RH-Firewall-1-INPUT -m conntrack --ctstate NEW -m tcp -p tcp --dport 9419 -j ACCEPT
-A RH-Firewall-1-INPUT -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

You must now restart the firewall service with:

```
service iptables restart
```

LizardFS requires the host name 'mfsmaster' to resolve and point to the IP of the master server. The easiest way to achieve this is to edit '/etc/hosts' – and add an entry for it:

```
192.168.0.100    mfsmaster
```

The '192.168.0.100' IP address should be the IP address of a LizardFS 'master' server (not shadow). If you're running XenServer with an existing LizardFS system on other hosts – you should already have a 'master' server. If you're running LizardFS master service on XenServer it'll be the IP of whichever node you setup as 'master' (not 'shadow').

If you are running an HA setup of LizardFS it should be the "URAFT_FLOATING_IP" you defined in your URAFT configuration.

Assuming you installed the LizardFS admin tools – you can make life easier by adding the following lines to '/root/.bashrc':

```
alias lfschunks='lizardfs-admin list-chunkservers mfsmaster 9421'
alias lfshealth='lizardfs-admin chunks-health mfsmaster 9421'
alias lfsmounts='lizardfs-admin list-mounts mfsmaster 9421'
alias lfsdisks='lizardfs-admin list-disks mfsmaster 9421'
```

Once the service has started we can use these aliases / commands to check on things (if you haven't installed the LizardFS admin tools and want to use these commands – see above for info on how to install them).

Testing your LizardFS setup

If you're **connecting XenServer to an existing LizardFS system** – you should be able to just mount the LizardFS at this point, e.g.:

```
mkdir /mnt/lizardfs
mfsmount /mnt/lizardfs
```

This should mount the LizardFS file system under '/mnt/lizardfs'. If it's an existing system you'll need to make sure you mount at the correct point (i.e. check how the existing system is setup).

If you're **running LizardFS on the actual XenServers** we'll need to bring up the 'master' service – and the chunk-servers. You can do this with:

```
service lizardfs-master start
service lizardfs-chunkserver start
```

You'll need to repeat this on each node – remembering only one can be the 'master' meta-server – the other has to be a shadow (set in '/etc/mfs/mfsmaster.cfg) You should then be able to mount the default LizardFS then with:

```
mkdir /mnt/lizardfs
mfsmount /mnt/lizardfs
```

This should be repeated on each node.

Once that's done – if you've installed the LizardFS admin tools (and added the above Bash aliases) you can use:

```
lfshealth - Display info on the 'health' of the LizardFS
lfsmounts - Display info on what's mounted the LizardFS filesystem
lfsdisks - Display info on all the disks provided by the chunk-servers on the system
lfschunks - Display info on the chunk-servers on the system
```

As a quick test – if you create a test-file in '/mnt/lizardfs' on one node – the other should show it, i.e.:

```
[root@XenServer-1 ~]# cd /mnt/lizardfs
[root@XenServer-1 lizardfs]# echo "Hello World!" >/mnt/lizardfs/test.txt
```

(switch to XenServer-2 Console)

```
[root@XenServer-1 ~]# cd /mnt/lizardfs
[root@XenServer-2 lizardfs]# cat test.txt
Hello World!
[root@XenServer-2 lizardfs]#
```

At this point we can create a 'xen-sr' directory – and set a 'goal' on it. Again, if you're tying into an existing LizardFS system you'll need to see how that's configured before you go creating directories / setting goals in place.

If you're running XenServer as it's own LizardFS system we can do:

```
mkdir /mnt/lizardfs/xen-sr
mfssetgoal 2 /mnt/lizardfs/xen-sr
```

Using a goal of "2" means (by default) that LizardFS will keep 2 copies (one on each node) of any chunks – so if one chunk server (XenServer fails) the other can still access the data.

Creating a storage repository (SR)

Now we need to create a XenServer Storage Repository (SR) on the LizardFS. If you have more than one XenServer – you should log into the pool master and then run:

```
xe host-list
```

Make a note of the pool master's uuid (and the other nodes uuid) – you'll need those in a moment.

Now do:

```
export MYUUID=`uuidgen`
xe sr-introduce uuid=$MYUUID name-label="LizardFS" content-type=user type=file␣
↪shared=true 61625483-3889-4c55-8eee-07d14e9c9044
xe pbd-create sr-uuid=$MYUUID device-config:location=/mnt/lizardfs/xen-sr host-
↪uuid=(uuid of pool master) 62c9a88a-5fe4-4720-5a85-44b75aebb7fd
xe pbd-create sr-uuid=$MYUUID device-config:location=/mnt/lizardfs/xen-sr host-
↪uuid=(uuid of 2nd node) a91b77ee-949d-49d9-186f-259cd96b5c00
xe pbd-plug uuid=62c9a88a-5fe4-4720-5a85-44b75aebb7fd
xe pbd-plug uuid=a91b77ee-949d-49d9-186f-259cd96b5c00
```

At this point in XenCenter (the GUI admin tool for XenServer) you should be able to see a new storage repository called "LizardFS"

System Startup

Ok – so we've now got a LizardFS system – and a XenServer Storage Repository.

At boot time – it's obviously important that LizardFS is up and running (either just the client, or the client – and server components if you're running everything on XenServer).

The easiest way to achieve this (at present) is to create a startup script – and have that invoked, just before XenServer attaches to the LizardFS based storage repository. So we'll edit a file called '/root/lizardfs-sr-start.sh' – and put into it:

```
#!/bin/sh
# Start the LizardFS 'master' Service (if you need to)
service lizardfs-master start
# Start the LizardFS 'chunkserver' Service (if you need to)
service lizardfs-chunkserver start
# Mount the LizardFS
mfsmount /mnt/lizardfs
# Return 'Ok' back
exit 0
```

You need to 'chmod u+x lizardfs-sr-start.sh' to make sure it's executable.

This needs to be hooked into the XenServer startup – this means editing one of the XenServer python files.

If you 'vi /opt/xensource/sm/FileSR.py' – and then search for a line that says "def attach" - you need to change that function to read:

```
def attach(self, sr_uuid):
   if not self._checkmount():
     try:
         import subprocess
         subprocess.call(['/root/lizardfs-sr-start.sh'])
```

At boot time, as the local file repository gets attached – the 'lizardfs-sr-start.sh' script will be called – which makes sure the services are started, and LizardFS mounted up.

At this point you can test the system by restarting the 2nd node (if you have one) – then the first node (pool master) – both should come back, re-attach the LizardFS – and have the LizardFS storage repository available.

NOTES

Note: If you make one of your XenServer's the meta-server master – it must be up and running in order for the other nodes to use the storage.

Note: If the meta-server 'master' fails – you can promote one of the remaining 'shadow' servers to be the new master – but there must be only one 'master' on the system at any time (so the previous master will have to be reconfigured and come back as a 'shadow' server).

Note: LizardFS provide 'lizard-uraft' – which utlizes the *raft* protocol to keep a 'master' server always available. It's designed for use by a minimum of 3 nodes (two of which can be the XenServer).

This is covered in *Deploying LizardFS as a HA Cluster* – along with 'best practices'.

Having a third node also ensures there is always a 'master' server available for when the XenServer nodes boot. It is often common to need things like DNS, and 'routing' for XenServer to come up any way – so whilst you can build a 2 node system – 3 nodes is almost certainly better (even if one is not a XenServer – and just a basic machine providing DNS, LizardFS meta-server etc.)

Additionally – the third node can be used to provide a small amount of NFS storage. By creating a XenServer Storage Repository using this NFS space – XenServer's HA (High Availability) mode can be enabled.

Warning: XenServer patches / updates may replace the modifications to the "FileSR.py" file – so remember to check this after installing updates.

Usually in a XenServer 'pool' situation you would update the master first (make sure that restarts OK – including the LizardFS side of things) – then update the other nodes in turn.

8.6.2 Using LizardFS for Virtualization Farms

If you want to use LizardFS as a Backend for your virtualization Farm, there are multiple options.

Use LizardFS from inside each VM The LizardFS client on Linux utilises the *FUSE* library which has limits on the performance it can offer. To work around this one option would be to have each VM connect to the lizardfs system by itself. That way each VM has its own connection and gets the maximum performance possible via fuse.

Create one mountpoint on your host for each VM (especially cool with KVM) This is simple and efficient. Since the *FUSE* library creates a new instance for every mountpoint, each mountpoint gets the full performance of a *FUSE* connection and that way gets around the limits a single fuse connection currently has. So basically each VM, using a separate lizardfs mountpoint each, will get full throughput until the host runs out of network ressources.

The setup is rather simple. Create multiple subdirectories in your lizardfs and mount each one separately for each VM:

```
mfsmount -S <lizardfs subdirectory> -c <mfsmount config file>
```

Each mount will have its own instance and create its own *FUSE* process working like a totaly separate connection and process. This is a workaround for the know limitations of the *FUSE* library.

8.7 Dummy-georeplication

CHAPTER 9

Glossary

The following abbreviations are used in LizardFS, both in documentation and in the names of variables and classes:

9.1 A

9.2 B

9.3 C

9.3.1 CMake

CMake is an open-source, cross-platform family of tools designed to build, test and package software. CMake is used to control the software compilation process using simple platform and compiler independent configuration files, and generate native makefiles and workspaces that can be used in the compiler environment of your choice. The suite of CMake tools were created by Kitware in response to the need for a powerful, cross-platform build environment for open-source projects such as ITK and VTK.

You can find additional information including books and online training at: https://cmake.org/

9.4 D

9.5 E

9.6 F

9.6.1 FC/FibreChannel

Fibre Channel, or FC, is a high-speed network technology (commonly running at 1, 2, 4, 8, 16, 32, and 128 gigabit per second rates) primarily used to connect computer data storage to servers. Fibre Channel is mainly used in storage area networks (SAN) in enterprise storage. Fibre Channel networks are known as a Fabric because they operate in unison as one big switch. Fibre Channel typically runs on optical fiber cables within and between data centers.

Most block storage runs over Fibre Channel Fabrics and supports many upper level protocols. Fibre Channel Protocol (FCP) is a transport protocol that predominantly transports SCSI commands over Fibre Channel networks. Mainframe computers run the FICON command set over Fibre Channel because of its high reliability and throughput. Fibre Channel is the most popular network for flash memory being transported over the NVMe interface protocol.

The folowing Variants of FC are an overview of FC native speeds:

Fibre Channel Variants

NAME	Line-rate	Line coding	Nominal throughp. per direction; MB/s	Net throughp. per direction; MB/s	Availability
1GFC	1.0625	8b10b	100	103.2	1997
2GFC	2.125	8b10b	200	206.5	2001
4GFC	4.25	8b10b	400	412.9	2004
8GFC	8.5	8b10b	800	825.8	2005
10GFC	10.51875	64b66b	1,200	1,239	2008
16GFC “Gen 5”	14.025	64b66b	1,600	1,652	2011
32GFC “Gen 6”	28.05	64b66b	3,200	3,303	2016
128GFC “Gen 6”	4×28.05	64b66b	12,800	13,210	2016

For more information please check the Wikipedia entry for SAN: https://en.wikipedia.org/wiki/Fibre_Channel where most of the info in this entry is from.

9.6.2 FUSE

FUSE (Filesystem in Userspace) is an interface for userspace programs to export a filesystem to the Linux kernel. The FUSE project consists of two components: the fuse kernel module (maintained in the regular kernel repositories) and the libfuse userspace library (maintained in this repository). libfuse provides the reference implementation for communicating with the FUSE kernel module.

A FUSE file system is typically implemented as a standalone application that links with libfuse. libfuse provides functions to mount the file system, unmount it, read requests from the kernel, and send responses back. libfuse offers two APIs: a “high-level”, synchronous API, and a “low-level” asynchronous API. In both cases, incoming requests from the kernel are passed to the main program using callbacks. When using the high-level API, the callbacks may work with file names and paths instead of inodes, and processing of a request finishes when the callback function returns. When using the low-level API, the callbacks must work with inodes and responses must be sent explicitly using a separate set of API functions.

See also:

Fuse on github https://github.com/libfuse/libfuse

Fuse mailing lists https://lists.sourceforge.net/lists/listinfo/fuse-devel

9.7 G

9.7.1 gcc

The GNU Compiler Collection (GCC) is a compiler system produced by the GNU Project supporting various programming languages. GCC is a key component of the GNU toolchain. The Free Software Foundation (FSF) distributes GCC under the GNU General Public License (GNU GPL). GCC has played an important role in the growth of free software, as both a tool and an example.

Originally named the GNU C Compiler, when it only handled the C programming language, GCC 1.0 was released in 1987. It was extended to compile C++ in December of that year (the default support in the current version is gnu++14, a superset of C++14 (or gnu11 a superset of C11) with strict standard support also available, and experimental C++17 and later). Front ends were later developed for Objective-C, Objective-C++, Fortran, Java, Ada, and Go among others.

“Version 4.5 of the OpenMP specification is now supported in the C and C++ compilers” and “a much improved implementation of the OpenACC 2.0a specification” is also supported.

GCC has been ported to a wide variety of processor architectures, and is widely deployed as a tool in the development of both free and proprietary software. GCC is also available for most embedded systems,[citation needed] including ARM-based; AMCC, and Freescale Power Architecture-based chips. The compiler can target a wide variety of platforms.

As well as being the official compiler of the GNU operating system, GCC has been adopted as the standard compiler by many other modern Unix-like computer operating systems, including Linux and the BSD family, although FreeBSD and OS X have moved to the LLVM system. Versions are also available for Microsoft Windows and other operating systems; GCC can compile code for Android and iOS.

See also:

Wikipedia entry for gcc https://en.wikipedia.org/wiki/GNU_Compiler_Collection

The offiial GNU Compiler Collection page http://gcc.gnu.org/

9.7.2 git

Git is a version control system that is used for software development and other version control tasks. As a distributed revision control system it is aimed at speed, data integrity, and support for distributed, non-linear workflows. Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.

To learn more about using git, please refer to its homepage: https://www.git-scm.com/, you will find many ressources there.

9.7.3 github

GitHub is a web-based Git repository hosting service. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project.

GitHub offers both plans for private repositories, and free accounts which are commonly used to host open-source software projects. As of April 2016, GitHub reports having more than 14 million users and more than 35 million repositories, making it the largest host of source code in the world.

The trademark mascot of GitHub is Octocat, an anthropomorphized cat with cephalopod limbs, portrayed in a manga style.

LizardFS uses Github for its source code repository and bug reporting location.

Github is located at: http://github.com/

9.8 H

9.8.1 homebrew

Homebrew is a free and open-source software package management system that simplifies the installation of software on Apple's OS X operating system. Originally written by Max Howell, the package manager has gained popularity in the Ruby on Rails community and earned praise for its extensibility. Homebrew has been recommended for its ease of use as well as its integration into the command line.

Homebrew has made extensive use of GitHub in order to expand the support of several packages through user contributions. In 2010, Homebrew was the third-most-forked repository on GitHub. In 2012, Homebrew had the largest number of new contributors on GitHub. In 2013, Homebrew had both the largest number of contributors and issues closed of any project on GitHub.

Homebrew has spawned several sub-projects such as Linuxbrew which is a Linux port and Homebrew-Cask which is an external command allowing installation of GUI applications,[8] as well as "taps" dedicated to specific areas or programming languages like Homebrew PHP.

See also:

homebrew on Wikipedia https://en.wikipedia.org/wiki/Homebrew_(package_management_software)

Official Website http://brew.sh/

homebrew on github https://github.com/Homebrew/brew

How-To Geek - How to Install Packages with Homebrew for OS X http://www.howtogeek.com/211541/homebrew-for-os-x-easily-installs-desktop-apps-and-terminal-utilities/

9.9 I

9.10 J

9.11 K

9.12 L

9.13 M

9.14 N

9.15 O

9.15.1 OSXFuse

FUSE for macOS allows you to extend macOS's native file handling capabilities via third-party file systems. It is a successor to MacFUSE, which has been used as a software building block by dozens of products, but is no longer being maintained.

** Features **

As a user, installing the *FUSE* for macOS software package will let you use any third-party FUSE file system. Legacy MacFUSE file systems are supported through the optional MacFUSE compatibility layer.

As a developer, you can use the FUSE SDK to write numerous types of new file systems as regular user space programs. The content of these file systems can come from anywhere: from the local disk, from across the network, from memory, or any other combination of sources. Writing a file system using FUSE is orders of magnitude easier and quicker than the traditional approach of writing in-kernel file systems. Since FUSE file systems are regular applications (as opposed to kernel extensions), you have just as much flexibility and choice in programming tools, debuggers, and libraries as you have if you were developing standard macOS applications.

See also:

You can find OSXFuse at https://osxfuse.github.io/ *FUSE*

9.16 P

9.16.1 Pandoc

Pandoc is a free and open-source software document converter, widely used as a writing tool (especially by scholars)[1][2][3][4] and as a basis for publishing workflows.[5][6][7] It was originally created by John Mac- Farlane, a philosophy professor at the University of California, Berkeley.

Supported file formats

Pandoc's most thoroughly supported file format is an extended version of Markdown, but it can also read many other forms of lightweight markup language, HTML, ReStructuredText, LaTeX, OPML, Org-mode, DocBook, and Office Open XML (Microsoft Word .docx).

It can be used to create files in many more formats, including Office Open XML, OpenDocument, HTML, Wiki markup, InDesign ICML, web-based slideshows, ebooks, OPML, and various TeX formats (through which it can produce a PDF). It has built-in support for converting LaTeX mathematical equations to MathML and MathJax, among other formats.

Plug-ins for custom formats can also be written in Lua, which has been used to create an exporting tool for the Journal Article Tag Suite.

LizardFS uses pandoc to generate manpages out of the source of the doc- umentation which is written in restructured text.

See also:

Wikipedia entry for pandoc: https://en.wikipedia.org/wiki/Pandoc

Pandoc home page http://pandoc.org/

9.17 Q

9.17.1 quorum

From Wikipedia, the free encyclopedia

A quorum is the minimum number of votes that a distributed transaction has to obtain in order to be allowed to perform an operation in a distributed system. A quorum-based technique is implemented to enforce consistent operation in a distributed system.

9.18 R

9.18.1 raft

What is Raft?

Raft is a consensus algorithm that is designed to be easy to understand. It's equivalent to Paxos in fault-tolerance and performance. The difference is that it's decomposed into relatively independent subproblems, and it cleanly addresses all major pieces needed for practical systems. We hope Raft will make consensus available to a wider audience, and that this wider audience will be able to develop a variety of higher quality consensus-based systems than are available today.

Hold on—what is consensus?

Consensus is a fundamental problem in fault-tolerant distributed systems. Consensus involves multiple servers agreeing on values. Once they reach a decision on a value, that decision is final. Typical consensus algorithms make progress when any majority of their servers are available; for example, a cluster of 5 servers can continue to operate even if 2 servers fail. If more servers fail, they stop making progress (but will never return an incorrect result).

Consensus typically arises in the context of replicated state machines, a general approach to building fault-tolerant systems. Each server has a state machine and a log. The state machine is the component that we want to make fault-tolerant, such as a hash table. It will appear to clients that they are interacting with a single, reliable state machine, even if a minority of the servers in the cluster fail. Each state machine takes as input commands from its log. In our hash table example, the log would include commands like set x to 3. A consensus algorithm is used to agree on the commands in the servers' logs. The consensus algorithm must ensure that if any state machine applies set x to 3 as the nth command, no other state machine will ever apply a different nth command. As a result, each state machine processes the same series of commands and thus produces the same series of results and arrives at the same series of states.

See also:

- **The Raft Consensus Algorithm** Contains a nice visualisatio of how the election process works.

9.18.2 rpm

RPM Package Manager (RPM) (originally Red Hat Package Manager; now a recursive acronym) is a package management system.[5] The name RPM refers to the following: the .rpm file format, files in the .rpm file format, software packaged in such files, and the package manager program itself. RPM was intended primarily for Linux distributions; the file format is the baseline package format of the Linux Standard Base.

Even though it was created for use in Red Hat Linux, RPM is now used in many Linux distributions. It has also been ported to some other operating systems, such as Novell NetWare (as of version 6.5 SP3) and IBM's AIX (as of version 4). An RPM package can contain an arbitrary set of files. The larger part of RPM files encountered are "binary RPMs" (or BRPMs) containing the compiled version of some software. There are also "source RPMs" (or SRPMs) files containing the source code used to produce a package. These have an appropriate tag in the file header that distinguishes them from normal (B)RPMs, causing them to be extracted to /usr/src on installation. SRPMs customarily carry the file extension ".src.rpm" (.spm on file systems limited to 3 extension characters, e.g. old DOS FAT).

See also:

Wikipedia article on the RPM Package Manager https://en.wikipedia.org/wiki/RPM_Package_Manager

Glossary entry for *yum*

9.19 S

9.19.1 SAN/Storage Area Network

A storage area network (SAN) is a network which provides access to consolidated, block level data storage. SANs are primarily used to enhance storage devices, such as disk arrays, tape libraries, and optical jukeboxes, accessible to servers so that the devices appear to the operating system as locally attached devices. A SAN typically has its own network of storage devices that are generally not accessible through the local area network (LAN) by other devices. The cost and complexity of SANs dropped in the early 2000s to levels allowing wider adoption across both enterprise and small to medium-sized business environments.

A SAN does not provide file abstraction, only block-level operations. However, file systems built on top of SANs do provide file-level access, and are known as shared-disk file systems.

See also:

Wikipedia entry for SAN: https://en.wikipedia.org/wiki/Storage_area_network

SNIA definition of Storage Area Networks: http://www.snia.org/education/storage_networking_primer/san/what_san

9.20 T

9.21 U

9.22 V

9.23 W

9.24 X

9.24.1 Xcode

Xcode is an integrated development environment (IDE) containing a suite of software development tools developed by Apple for developing software for macOS, iOS, WatchOS and tvOS. First released in 2003, the latest stable release is version 8 and is available via the Mac App Store free of charge for OS X El Capitan users. Registered developers can download preview releases and prior versions of the suite through the Apple Developer website. However, Apple recently made a beta version of version 8.0 of the software available to those of the public with Apple Developer accounts.

Xcode supports source code for the programming languages C, C++, Objective-C, Objective-C++, Java, AppleScript, Python, Ruby, ResEdit (Rez), and Swift, with a variety of programming models, including but not limited to Cocoa, Carbon, and Java. Third parties have added support for GNU Pascal, Free Pascal, Ada, C#, Perl, and D.

Thanks to the Mach-O executable format, which allows fat binary files, containing code for multiple architectures, Xcode can build universal binary files, which allow software to run on both PowerPC and Intel-based (x86) platforms and that can include both 32-bit and 64-bit code for both architectures. Using the iOS SDK, Xcode can also be used to compile and debug applications for iOS that run on ARM architecture processors. Xcode includes the GUI tool Instruments, which runs atop a dynamic tracing framework, DTrace, created by Sun Microsystems and released as part of OpenSolaris.

See also:

Wikipedia entry for Xcode https://en.wikipedia.org/wiki/Xcode

Xcode at the Mac App Store https://itunes.apple.com/us/app/xcode/id497799835

Xcode at Apples Developer Site https://developer.apple.com/xcode/

9.24.2 XFS

XFS is a high-performance 64-bit journaling file system created by Silicon Graphics, Inc (SGI) in 1993. It was the default file system in the SGI's IRIX operating system starting with its version 5.3; the file system was ported to the Linux kernel in 2001. As of June 2014, XFS is supported by most Linux distributions, some of which use it as the default file system.

XFS excels in the execution of parallel input/output (I/O) operations due to its design, which is based on allocation groups (a type of subdivision of the physical volumes in which XFS is used- also shortened to AGs). Because of this, XFS enables extreme scalability of I/O threads, file system bandwidth, and size of files and of the file system itself when spanning multiple physical storage devices.

XFS ensures the consistency of data by employing metadata journaling and supporting write barriers. Space allocation is performed via extents with data structures stored in B+ trees, improving the overall performance of the file system, especially when handling large files. Delayed allocation assists in the prevention of file system fragmentation; online defragmentation is also supported. A feature unique to XFS is the pre-allocation of I/O bandwidth at a pre-determined rate, which is suitable for many real-time applications; however, this feature was supported only on IRIX, and only with specialized hardware.

A notable XFS user, NASA Advanced Supercomputing Division, takes advantage of these capabilities deploying two 300+ terabyte XFS filesystems on two SGI Altix archival storage servers, each of which is directly attached to multiple Fibre Channel disk arrays.

See also:

XFS FAQ http://xfs.org/index.php/XFS_FAQ

Wikipedia entry for XFS https://en.wikipedia.org/wiki/XFS

9.25 Y

9.25.1 yum

yum is the primary tool for getting, installing, deleting, querying, and managing Red Hat Enterprise Linux RPM software packages from official Red Hat software repositories, as well as other third-party repositories. yum is used in Red Hat Enterprise Linux, CentOS and Scientific Linux versions 5 and later.

Versions of Red Hat Enterprise Linux 4 and earlier used up2date.

9.26 Z

9.26.1 ZFS

The Z File System, or ZFS, is an advanced file system designed to overcome many of the major problems found in previous designs.

Originally developed at Sun™, ongoing open source ZFS development has moved to the OpenZFS Project
.

ZFS has three major design goals:

- Data integrity: All data includes a checksum of the data. When data is written, the checksum is calculated and written along with it. When that data is later read back, the checksum is calculated again. If the checksums do not match, a data error has been detected. ZFS will attempt to automatically correct errors when data redundancy is available.
- Pooled storage: physical storage devices are added to a pool, and storage space is allocated from that shared pool. Space is available to all file systems, and can be increased by adding new storage devices to the pool.
- Performance: multiple caching mechanisms provide increased performance. ARC is an advanced memory-based read cache. A second level of disk-based read cache can be added with L2ARC, and disk-based synchronous write cache is available with ZIL.

See also:

- A guide to install and use zfs on centos 7
- The Open-ZFS Project
- ZFS Manual in the FreeBSD Handbook

- The ZFS On Linux - ZOL project supplies packages and documentation for every major distro: ZFS On Linux - ZOL
- ZFS in the Ubuntu Wiki
- How to install and use ZFS on Ubuntu and why you'd wnat to
- An extensive Guide about ZFS on Debian by Aaron Toponce
- Performance tuning instructions from the Open-ZFS Project
- ZFS RAIDZ stripe width or: How I learned to stop worrying and love raidZ
- ZFS Administration Part I VDEVs

CHAPTER 10

Dummy-Releasenotes

CHAPTER 11

Commercial Support

Commercial Support is available from Skytechnology Sp. z o.o..

Have peace of mind with our helpful, passionate support team on standby.

We offer full maintenance, development and support for LizardFS:

- 24/7 Enterprise Support
- We can assist our clients to meet their unique and specific needs and, if necessary, develop new features upon request
- We can provide a tailor-made comprehensive solution (hardware and software)
- We can guide in migrating the data from your current storage solution(s) to LizardFS (possibly using the hardware you already have) to start taking advantage of our unique feature set.

  You can reach us at: http://www.lizardfs.com/

Skytechnology Sp. z o.o.
Ul. Głogowa 21
02-639 Warszawa
Poland

Tel: (+48) 22 100 32 48
Tel: (+1) 646 655 0 693

E-mail: info@lizardfs.com

CHAPTER 12

References

This documentation would have not been possible without the help of many sources that we would like to menton here:

- Wikipedia http://www.wikipedia.org
- The ReadTheDocs Project http://www.readthedocs.org/
- Github http://www.github.com/
- The sphinx project http://www.sphinx-doc.org/

CHAPTER 13

Indices and tables

- genindex
- search