# Introduction

The logical components of MarFS are the FUSE daemon running on the interactive FTA(s) (File Transfer Agents), pftool running on the batch FTA(s), and the utility programs for managing and maintaining MarFS. At least one underlying MDFS (Meta-Data File-System). MarFS data can be stored directly into the MDFS, but, presumably, one will also want an object-storage system.

# Basics

We assume the following variables, in order to make these instructions generic:

* SRC = directory under which MarFS code will be checked out
* PFS = path of a mounted parallel file-system
* GPFS = same as PFS, assuming PFS is gpfs.
* PA2X = dir where Ron Croonenberg’s parser-tool will be installed
* CONFIG = path to file that will contain marfs configuration
* AWS4C = dir where the LANL-extended version of the aws4c library is checked-out
* PF = dir where pftool is installed

Where we show environment-variables on the command-line, in instructions below, these can be set in the environment, instead.

# Create infrastructure

## directories, filesets, etc

The configuration created in 4.3.1 will include namespaces that refer to file-systems and directories, which must currently be created by hand.

For each namespace:

* create a file-set where the MD for this NS will live
* create a directory under the fileset, matching the ‘md\_path’ in the config.
* assign ownership (chown), and access (chmod) to the directory.
* touch the file matching the ‘fsinfo\_path’ in the config
  + allow read-only by all users that can access files in the namespace
* create a fileset (and perhaps subdir) matching ‘trash\_path’ in the config.
  + allow write/create by all users that can delete files in the namespace

For the fuse-mount:

* create the directory where fuse will be mounted
* start syslog service, if you intend to run fuse with logging

## repositories

MarFS data-storage can use a variety of storage back-ends. Each one may require different kinds of configuration:

### DIRECT

DIRECT repos store all data directly into the MDFS. Therefore, nothing special is needed, after the MDFS has been created.

### SPROXYD (Scality)

For each SPROXYD repo, the following is needed, on every server that will serve sproxyd requests:

edit /etc/sproxyd.conf

* add a driver alias matching the associated MarFS repo-name
* it should have “by-name” true.
* give it a unique service-id
* select storage parameters (e.g. ARC schema for erasure-coding)

edit /etc/httpd/conf.d/\_scality/sproxyd.conf

* add a new line for each MarFS namespace that will come here
* the lines can all be identical, except for the namespace-name

restart associated services (may take a moment)

* service restart httpd
* service restart scality-sproxyd

# Install and build MarFS, and associated tools

## Install RESTful object-storage tools

* cd $AWS4C
* git clone [git@github.com:jti-lanl/aws4c.git](mailto:git@github.com:jti-lanl/aws4c.git)
* make
* create ~/.awsAuth, with lines like unix-user:storage-user:storage-key
* chmod 700 ~/.awsAuth

## Install parser tools

* cd $PA2X
* git clone [git@git.lanl.gov:rocr/PA2X.git](mailto:git@git.lanl.gov:rocr/PA2X.git)

## Install MarFS

* cd $SRC
* git clone [git@github.com:mar-file-system/marfs.git](mailto:git@github.com:mar-file-system/marfs.git)

### Prepare local configuration

The configuration-file is used by FUSE, pftool, and (eventually) by various scripts that perform useful functions. It contains information about the fuse-mount, and all namespaces and repositories. There is a “blueprint” config-file in $SRC/fuse/src. This defines the syntax for legitimate config files, and is read by the first stage of the parser-generation, to create the parser. You must then create a config file that matches this syntax, which the parser can read. We have a version we call marfs\_cctest.cfg.

* (The following also builds libconfig.a, and cmd-line tool ‘marfs\_config’)
* edit $CONFIG [create repos and namespaces]
* cd $SRC/common/configuration/src
* make pristine
* make PARSE\_DIR=$PA2X [ BLUEPRINT=… ] CONFIG=$CONFIG

### Build and mount a MarFS fuse client (as root)

* (The following also builds libmarfs.a)
* yum install fuse fuse-devel (if not there already)
* cd $SRC/fuse/src
* make LIBAWS4C=$AWS4C
* export MARFSCONFIGRC=$CONFIG
* make mnt

## Install pftool

* cd $PF
* git clone [git@github.com:pftool/pftool.git](mailto:git@github.com:pftool/pftool.git)
* # all one line …

( export CPPFLAGS=”-I/usr/include/libxml2 -I$AWS4C –I$SRC/common/log/src

-I$SRC/common/configuration/src –I$SRC/fuse/src”;

export LDFLAGS=”-L/usr/lib64 -L$AWS4C -L$SRC/fuse/src

-L$SRC/common/configuration/src”;

./configure --enable-marfs )

* export MARFSCONFIGRC=$CONFIG
* make -j 4
* make install

# Monitoring / Debugging

## MarFS Fuse

Fuse currently writes voluminous logs to syslog. You must enable syslog before these logs will be used.

* service syslog-ng start

When fuse is healthy:

* ‘mount’ will show the mount-point
* ‘ps –elf | grep fuse’ will show (one instance of) marfs\_fuse
* /var/log/messages will show activity during interaction with the fuse mount

To unmount fuse (as root)

* cd $SRC/fuse/src
* make umnt

If the above doesn’t work (as root)

* cd $SRC/fuse/src
* make abort

To report bugs that are simple reproducible problems:

* tail –f /var/log/messages
* [do the thing that causes the problem]
* cut-and-paste the part of the log that appeared during the preceding step

If fuse crashes on some activity (as root)

* ulimit –c unlimited
* [mount fuse]
* cause the crash
* send the /core\* that was created