mtree parsing and manipulation library

Michal Ratajsky
michal@FreeBSD.org

AsiaBSDCon 2016, Tokyo, Japan
March 12, 2016

Outline

Outline

About mtree

mtree specification

- list of file system objects along with their properties;
- consumed by mtree(8) and others;

Example

```
$ mtree -c -p /bin -k type,size,uid,gid
/set type=file uid=0 gid=0
.               type=dir
cat             size=11960
chflags         size=8096
chio            size=18552
chmod           size=8432
cp              size=21024
```

About mtree

Example (part of /etc/mtree/BSD.root.dist)

```
/set type=dir uname=root gname=wheel mode=0755
.
    bin
    ..
    boot
        defaults
        ..
        dtb
        ..
        firmware
        ..
```

About mtree

mtree specification

- exists in several formats:
  - the version 1 “mtree -c” format,
  - the version 2 “mtree -C” and “mtree -D” formats

Example (Specification in the “mtree -C” format)

```
$ mtree -c -p /bin -k type,size,uid,gid |mtree -C
. type=dir uid=0 gid=0
./cat type=file uid=0 gid=0 size=11960
./chflags type=file uid=0 gid=0 size=8096
./chio type=file uid=0 gid=0 size=18552
./chmod type=file uid=0 gid=0 size=8432
./cp type=file uid=0 gid=0 size=21024
```

About mtree

mtree specification

- supports many file properties (keywords), such as:
  - type: file type (file, dir, link, block, char, ...);
  - uid, uname, gid, gname: owning user and group;
  - cksum, md5, sha1, sha512 and other checksums;
  - size, mode, time, nlink and other information from stat(2);
  - ignore, nochange, optional;

mtree implementations

- mtree(8) creates and compares specifications and files in a file system;
- makefs(8) creates images from mtree manifests;
- install(1) writes "metalogs" with -M;
- libarchive(3), bsdtar(1) has mtree reader and writer, portable;

Outline

Motivation

Why create a new implementation?

- library implementation would help standardize specification;
- compatibility problems with the current implementations:
  - limited keyword support and different handling of corner cases;
- introduce new features;

Compatibility

- mtree(8):
  - describes specification formats understood by the mtree program;
  - requires "." as the first file in a specification;
  - supports [,], ? and * path matching characters in file names;
- mtree(5):
  - describes specification formats understood by libarchive;
  - includes content and resdevice keywords;

Outline

Implementation

libmtree prerequisites

- independent and portable:
  - hosted on GitHub;
  - written in C99, uses only commonly available POSIX functions;
  - no required external dependencies;
- simple to build and use:
  - uses the autotools build system;
  - provides libmtree.pc;
- support all keywords described in mtree(5);

Other project goals

- port all programs to use libmtree;
- improve portability of mtree and include it in the libmtree source tree;

Implementation

API requirements

- read and write specifications:
  - support all formats,
  - configurable to choose which keywords to read, whether to indent output, write /set, etc.
  - filtering;
- access individual file entries:
  - iterate,
  - change set of included keywords, read and set all keyword values,
  - more advanced features: sort, merge, compare, ...

Naming

- files are entries,
- property names are keywords,

Implementation

mtree specification: the libmtree way

- set of supported keywords based on development version of libarchive: largest set of keywords
- no need to start a specification with "." , but "./" enforced as a prefix of each entry (incompatible with libarchive and currently breaks libarchive test suite)
- path matching characters are not supported

Implementation

Basic libmtree objects

- two basic objects exist:
  - mtree_spec: represents a specification; lightweight container of file entries, includes higher-level functions which read and write whole specifications;
  - mtree_entry: represents a chain of file entries;

Example

```
struct mtree_spec *spec;

spec = mtree_spec_create();
```

Implementation

mtree_spec

- FILE *, file descriptor and buffer readers available;
- file system reader;
- configurable by choosing a combination of options;

Example (Reading)

```
mtree_spec_set_read_options(spec,
    MTREE_READ_SORT |
    MTREE_READ_MERGE |
    MTREE_READ_PATH_DONT_CROSS_MOUNT);

mtree_spec_read_spec_file(spec, fp);
mtree_spec_read_spec_fd(spec, fd);
mtree_spec_read_path(spec, "/bin");
```

Implementation

mtree_spec

- FILE *, file descriptor and user-defined function writers available;
- configurable by choosing the format and a combination of options;

Example (Writing)

```
mtree_spec_set_write_options(spec,
    MTREE_WRITE_INDENT |
    MTREE_WRITE_DIR_COMMENTS);

mtree_spec_set_write_format(spec,
    MTREE_FORMAT_2_0);

mtree_spec_write_file(spec, fp);
```

Implementation

mtree_entry

- doubly-linked list of entries;
- `mtree_entry_get_first()`, `mtree_entry_get_next()`, `mtree_entry_get_previous()`

Example (List functions)

```
struct mtree_entry *ent;

for (ent  = mtree_spec_get_entries(spec);
     ent != NULL;
     ent  = mtree_entry_get_next(ent)) {
        printf("file %s\n", mtree_entry_get_path(ent));
}
```

Implementation

mtree_entry

- many functions for creating and manipulating lists:
  - `mtree_entry_append()`, `mtree_entry_prepend()`, `mtree_entry_reverse()`, `mtree_entry_unlink()`

Example (List functions)

```
struct mtree_entry *list = NULL;

list = mtree_entry_prepend(list, entry1);
list = mtree_entry_prepend(list, entry2);
list = mtree_entry_reverse(list);
```

Implementation

mtree_entry

- `mtree_entry_create();`
- getters and setters for all keywords:
  - `mtree_entry_get_type()`, `mtree_entry_get_size()`,
  - `mtree_entry_set_type()`, `mtree_entry_set_size()`;
- the universal `mtree_entry_set_keywords();`
- many others:
  - `mtree_entry_compare()`,
  - `mtree_entry_sort()`,
  - `mtree_entry_merge()`,
  - `...`

Implementation

Other APIs

- for device and resdevice values: `mtree_device`
- for calculating checksums and digests: `mtree_cksum`, `mtree_digest`
- for comparison of specifications: `mtree_spec_diff`

Outline

What's Still To Do?

TODO

- Testing is needed for all parts;
- Expand libmtree test suite, include mtree tests;
- Integration in FreeBSD source tree;
- See the Wiki page:
  `https://wiki.freebsd.org/SummerOfCode2015/mtreeParsingLibrary`

Where to get the code?

- `https://github.com/mratajsky/libmtree`

Acknowledgements

Brooks Davis, mentor of the project

FreeBSD

Google

Thank you!