
dertools Documentation

Release 0.51.0

Erik Moqvist

Mar 12, 2022

Contents

1	About	3
2	Installation	5
3	Statistics	7
4	Example usage	9
4.1	Command line tool	9
5	Contributing	13
6	Patch types	15
6.1	Sequential	15
6.2	HDiffPatch	15
6.3	In-place	16
7	Functions and classes	19
	Index	21

Binary delta encoding in Python 3.6+ and C.

Based on <http://www.daemonology.net/bsdiff/> and `HDiffPatch`, with the following features:

- `bsdiff`, `hdiffpatch` and `match-blocks` algorithms.
- `sequential`, `hdiffpatch` or `in-place` (resumable) patch types.
- `BZ2`, `LZ4`, `LZMA`, `Zstandard`, `heatshrink` or `CRLE` compression.
- Sequential patches allow streaming.
- Maximum file size is 2 GB for the `bsdiff` algorithm. There is practically no limit for the `hdiffpatch` and `match-blocks` algorithms.
- `Incremental apply patch` implemented in C, suitable for memory constrained embedded devices. Only the `sequential` patch type is supported.
- `SA-IS` or `divsufsort` instead of `qsufsort` for `bsdiff`.
- Optional experimental data format aware algorithm for potentially smaller patches. I don't recommend anyone to use this functionality as the gain is small in relation to memory usage and code complexity!

There is a risk this functionality uses patent <https://patents.google.com/patent/EP1988455B1/en>. Anyway, this patent expires in August 2019 as I understand it.

Supported data formats:

- ARM Cortex-M4
- AArch64

Project homepage: <https://github.com/erimoq/detools>

Documentation: <http://detools.readthedocs.org/en/latest>

CHAPTER 2

Installation

```
pip install detools
```

Statistics

Patch sizes, memory usage (RSS) and elapsed times when creating a patch from Python-3.7.3.tar (79M) to Python-3.8.1.tar (84M) for various algorithm, patch type and compression combinations.

See [tests/benchmark.sh](#) for details on how the data was collected.

Algorithm	Patch type	Compr.	Patch size	RSS	Time
bsdiff	sequential	lzma	3,5M	662M	0:24.29
bsdiff	sequential	none	86M	646M	0:15.20
hdiffpatch	hdiffpatch	lzma	2,4M	523M	0:13.74
hdiffpatch	hdiffpatch	none	7,2M	523M	0:10.24
match-blocks	sequential	lzma	2,9M	273M	0:08.57
match-blocks	sequential	none	84M	273M	0:01.72
match-blocks	hdiffpatch	lzma	2,6M	212M	0:06.07
match-blocks	hdiffpatch	none	9,7M	212M	0:01.30

Same as above, but for MicroPython ESP8266 binary releases (from 604k to 615k).

Algorithm	Patch type	Compr.	Patch size	RSS	Time
bsdiff	sequential	lzma	71K	46M	0:00.64
bsdiff	sequential	none	609K	27M	0:00.33
hdiffpatch	hdiffpatch	lzma	65K	42M	0:00.37
hdiffpatch	hdiffpatch	none	123K	25M	0:00.32
match-blocks	sequential	lzma	194K	46M	0:00.44
match-blocks	sequential	none	606K	25M	0:00.22
match-blocks	hdiffpatch	lzma	189K	43M	0:00.38
match-blocks	hdiffpatch	none	313K	24M	0:00.19

Example usage

Examples in C are found in `c`.

4.1 Command line tool

4.1.1 The create patch subcommand

Create a patch `foo.patch` from `tests/files/foo/old` to `tests/files/foo/new`.

```
$ detools create_patch tests/files/foo/old tests/files/foo/new foo.patch
Successfully created 'foo.patch' in 0.01 seconds!
$ ls -l foo.patch
-rw-rw-r-- 1 erik erik 127 feb  2 10:35 foo.patch
```

Create the same patch as above, but without compression.

```
$ detools create_patch --compression none \
  tests/files/foo/old tests/files/foo/new foo-no-compression.patch
Successfully created 'foo-no-compression.patch' in 0 seconds!
$ ls -l foo-no-compression.patch
-rw-rw-r-- 1 erik erik 2792 feb  2 10:35 foo-no-compression.patch
```

Create a `hdiffpatch` patch `foo-hdiffpatch.patch`.

```
$ detools create_patch --algorithm hdiffpatch --patch-type hdiffpatch \
  tests/files/foo/old tests/files/foo/new foo-hdiffpatch.patch
Successfully created patch 'foo-hdiffpatch.patch' in 0.01 seconds!
$ ls -l foo-hdiffpatch.patch
-rw-rw-r-- 1 erik erik 146 feb  2 10:37 foo-hdiffpatch.patch
```

Lower memory usage with `--algorithm match-blocks` algorithm. Mainly useful for big files. Creates slightly bigger patches than `bsdiff` and `hdiffpatch`.

```
$ detools create_patch --algorithm match-blocks \  
    tests/files/foo/old tests/files/foo/new foo-hdiffpatch-64.patch  
Successfully created patch 'foo-hdiffpatch-64.patch' in 0.01 seconds!  
$ ls -l foo-hdiffpatch-64.patch  
-rw-rw-r-- 1 erik erik 404 feb  8 11:03 foo-hdiffpatch-64.patch
```

Non-sequential but smaller patch with `--patch-type hdiffpatch`.

```
$ detools create_patch \  
    --algorithm match-blocks --patch-type hdiffpatch \  
    tests/files/foo/old tests/files/foo/new foo-hdiffpatch-sequential.patch  
Successfully created 'foo-hdiffpatch-sequential.patch' in 0.01 seconds!  
$ ls -l foo-hdiffpatch-sequential.patch  
-rw-rw-r-- 1 erik erik 389 feb  8 11:05 foo-hdiffpatch-sequential.patch
```

4.1.2 The create in-place patch subcommand

Create an in-place patch `foo-in-place.patch`.

```
$ detools create_patch_in_place --memory-size 3000 --segment-size 500 \  
    tests/files/foo/old tests/files/foo/new foo-in-place.patch  
Successfully created 'foo-in-place.patch' in 0.01 seconds!  
$ ls -l foo-in-place.patch  
-rw-rw-r-- 1 erik erik 672 feb  2 10:36 foo-in-place.patch
```

4.1.3 The create bsdiff patch subcommand

Create a bsdiff patch `foo-bsdiff.patch`, compatible with the original bsdiff program.

```
$ detools create_patch_bsdifff \  
    tests/files/foo/old tests/files/foo/new foo-bsdifff.patch  
Successfully created 'foo-bsdifff.patch' in 0 seconds!  
$ ls -l foo-bsdifff.patch  
-rw-rw-r-- 1 erik erik 261 feb  2 10:36 foo-bsdifff.patch
```

4.1.4 The apply patch subcommand

Apply the patch `foo.patch` to `tests/files/foo/old` to create `foo.new`.

```
$ detools apply_patch tests/files/foo/old foo.patch foo.new  
Successfully created 'foo.new' in 0 seconds!  
$ ls -l foo.new  
-rw-rw-r-- 1 erik erik 2780 feb  2 10:38 foo.new
```

4.1.5 The in-place apply patch subcommand

Apply the in-place patch `foo-in-place.patch` to `foo.mem`.

```
$ cp tests/files/foo/in-place-3000-500.mem foo.mem
$ dertools apply_patch_in_place foo.mem foo-in-place.patch
Successfully created 'foo.mem' in 0 seconds!
$ ls -l foo.mem
-rw-rw-r-- 1 erik erik 3000 feb  2 10:40 foo.mem
```

4.1.6 The `bsdiff apply patch` subcommand

Apply the patch `foo-bsdiff.patch` to `tests/files/foo/old` to create `foo.new`.

```
$ dertools apply_patch_bsdiff tests/files/foo/old foo-bsdiff.patch foo.new
Successfully created 'foo.new' in 0 seconds!
$ ls -l foo.new
-rw-rw-r-- 1 erik erik 2780 feb  2 10:41 foo.new
```

4.1.7 The `patch info` subcommand

Print information about the patch `foo.patch`.

```
$ dertools patch_info foo.patch
Type:                sequential
Patch size:          127 bytes
To size:             2.71 KiB
Patch/to ratio:      4.6 % (lower is better)
Diff/extra ratio:    9828.6 % (higher is better)
Size/data ratio:     0.3 % (lower is better)
Compression:         lzma

Number of diffs:     2
Total diff size:     2.69 KiB
Average diff size:   1.34 KiB
Median diff size:    1.34 KiB

Number of extras:    2
Total extra size:    28 bytes
Average extra size:  14 bytes
Median extra size:   14 bytes
```


1. Fork the repository.
2. Install prerequisites.

```
pip install -r requirements.txt
```

3. Implement the new feature or bug fix.
4. Implement test case(s) to ensure that future changes do not break legacy.
5. Run the tests.

```
make test
```

6. Create a pull request.

6.1 Sequential

A sequential patch uses two memory regions or files. One contains the from-data and the to-data is written to the other. The patch is accessed sequentially from the beginning to the end when applying the patch.

```
$ detools create_patch tests/files/foo.old tests/files/foo.new foo.patch
```

Patch layout:

header	diff 1	extra 1	adj. 1	diff 2	extra 2	adj. 2	...
--------	--------	---------	--------	--------	---------	--------	-----

The first part of the header is not compressed. The rest of the patch is compressed.

6.2 HDiffPatch

Patches of this type are slightly smaller than sequential patches.

```
$ detools create_patch --patch-type hdiffpatch \  
  tests/files/foo.old tests/files/foo.new foo.patch
```

Patch layout:

header	covers	RLE diff control	RLE diff code	extra
--------	--------	------------------	---------------	-------

The header is not compressed. The other four parts are compressed separately.

6.3 In-place

The in-place patch type is designed to update an application in place. It is useful when flash operations are faster than the external interface transfer speed.

Use `create_patch_in_place` to create an in-place patch. The `to` options `--memory-size` and `--segment-size` are required, while `--minimum-shift-size` is optional.

```
$ detools create_patch --type in-place --memory-size 131072 --segment-size 32768 \
  tests/files/foo.old tests/files/foo.new foo.patch
```

Here is an example of an in-place application update from version 1 to version 2. The two applications are represented by the character sequences below for clarity.

```
Version 1: 0123456789abcdefghijklmnopqr
Version 2: ABCDEFGHIJKLMNOPQRSTUVWXYZstuvwxyz
```

1. Before the update application version 1 is found in memory segments 0 to 3.

```

      0      1      2      3      4      5
+-----+-----+-----+-----+-----+-----+
|0123456789abcdefghijklmnopqr|                                     |
+-----+-----+-----+-----+-----+-----+

```

2. The update starts by moving the application two segments to the right to make room for the new version.

```

      0      1      2      3      4      5
+-----+-----+-----+-----+-----+-----+
|                                     |0123456789abcdefghijklmnopqr| |
+-----+-----+-----+-----+-----+-----+

```

3. The first part of the patch is received and combined with application version 1. The combined data is written to segment 0.

```

      0      1      2      3      4      5
+-----+-----+-----+-----+-----+-----+
|ABCDEFG|                                     |0123456789abcdefghijklmnopqr| |
+-----+-----+-----+-----+-----+-----+

```

4. Same as the previous step, but the combined data is written to segment 1.

```

      0      1      2      3      4      5
+-----+-----+-----+-----+-----+-----+
|ABCDEFGHJKLMNOP|0123456789abcdefghijklmnopqr| |
+-----+-----+-----+-----+-----+-----+

```

5. Segment 2 is erased to make room for the next part of the patch.

```

      0      1      2      3      4      5
+-----+-----+-----+-----+-----+-----+
|ABCDEFGHJKLMNOP|                                     |89abcdefghijklmnopqr| |
+-----+-----+-----+-----+-----+-----+

```

6. Combined data written to segment 2.

```

      0      1      2      3      4      5
+-----+-----+-----+-----+-----+-----+

```

(continues on next page)

(continued from previous page)

```
|ABCDEFGHIJKLMN|OPQRSTUVWXYZ|89abcdefghijklmnopqr| |
+-----+-----+-----+-----+-----+-----+
```

7. Segment 3 is erased.

```
 0      1      2      3      4      5
+-----+-----+-----+-----+-----+-----+
|ABCDEFGHIJKLMN|OPQRSTUVWXYZ|      |ghijklmnopqr| |
+-----+-----+-----+-----+-----+-----+
```

8. Combined data written to segment 3.

```
 0      1      2      3      4      5
+-----+-----+-----+-----+-----+-----+
|ABCDEFGHIJKLMN|OPQRSTUVWXYZstuvw|ghijklmnopqr| |
+-----+-----+-----+-----+-----+-----+
```

9. Segment 4 is erased.

```
 0      1      2      3      4      5
+-----+-----+-----+-----+-----+-----+
|ABCDEFGHIJKLMN|OPQRSTUVWXYZstuvw|      |opqr| |
+-----+-----+-----+-----+-----+-----+
```

10. Combined data written to segment 4.

```
 0      1      2      3      4      5
+-----+-----+-----+-----+-----+-----+
|ABCDEFGHIJKLMN|OPQRSTUVWXYZstuvwxyz|      |opqr| |
+-----+-----+-----+-----+-----+-----+
```

11. Optionally, segment 5 is erased.

```
 0      1      2      3      4      5
+-----+-----+-----+-----+-----+-----+
|ABCDEFGHIJKLMN|OPQRSTUVWXYZstuvwxyz|      |
+-----+-----+-----+-----+-----+-----+
```

12. Update to application version 2 complete!

An interrupted in-place update can be resumed by introducing a step state, persistently stored in a separate memory region. Also store the patch header persistently. Reject any other patch until the currently active patch has been successfully applied.

```
 0      1      2      3      4      5
+-----+-----+-----+-----+-----+-----+
|0123456789abcdefghijklmnopqr|      | Step: 0
+-----+-----+-----+-----+-----+-----+
|0123456789abcdefghijklmnopqr|      |opqr| | Step: 1
+-----+-----+-----+-----+-----+-----+
|0123456789abcdefghijklmnopqr| |ghijklmnopqr| | Step: 2
+-----+-----+-----+-----+-----+-----+
|0123456789abcdefghijklmnopqr|89abcdefghijklmnopqr| | Step: 3
+-----+-----+-----+-----+-----+-----+
|0123456789abcde|0123456789abcdefghijklmnopqr| | Step: 4
+-----+-----+-----+-----+-----+-----+
```

(continues on next page)

(continued from previous page)

```
|ABCDEFG789abcde|0123456789abcdefghij|klmnopqr| | Step: 5
+-----+-----+-----+-----+-----+-----+
|ABCDEFGHJKLMNO|0123456789abcdefghij|klmnopqr| | Step: 6
+-----+-----+-----+-----+-----+-----+
|ABCDEFGHJKLMN|OPQRSTUVWXYZ|89abcdefghij|klmnopqr| | Step: 7
+-----+-----+-----+-----+-----+-----+
|ABCDEFGHJKLMN|OPQRSTUVWXYZ|stuvw|ghij|klmnopqr| | Step: 8
+-----+-----+-----+-----+-----+-----+
|ABCDEFGHJKLMN|OPQRSTUVWXYZ|stuvwxyz| |opqr| | Step: 0
+-----+-----+-----+-----+-----+-----+
```

 Functions and classes

`detools.create_patch` (*f*from, *f*to, *f*patch, *compression*='lzma', *patch_type*='sequential', *algorithm*='bsdiff', *suffix_array_algorithm*='divsufsort', *memory_size*=None, *segment_size*=None, *minimum_shift_size*=None, *data_format*=None, *from_data_offset_begin*=0, *from_data_offset_end*=0, *from_data_begin*=0, *from_data_end*=0, *from_code_begin*=0, *from_code_end*=0, *to_data_offset_begin*=0, *to_data_offset_end*=0, *to_data_begin*=0, *to_data_end*=0, *to_code_begin*=0, *to_code_end*=0, *match_score*=6, *match_block_size*=64, *use_mmap*=True, *heatshrink_window_sz2*=8, *heatshrink_lookahead_sz2*=7)

Create a patch from *f*from to *f*to and write it to *f*patch. All three arguments are file-like objects.

compression must be 'bz2', 'crle', 'lzma', 'zstd', 'lz4' or 'none'.

patch_type must be 'sequential', 'in-place' or 'bsdiff'.

algorithm must be 'sequential' or 'hdiffpatch'.

suffix_array_algorithm must be 'sais' or 'divsufsort'.

memory_size, *segment_size* and *minimum_shift_size* are used when creating an in-place patch.

match_score is used by the hdiffpatch algorithm. Default 6. Recommended 0-4 for binary files and 4-9 for text files.

match_block_size is used by the match-blocks algorithm. Default 64. Less memory is needed to create the patch, but the patch will be bigger.

```
>>> ffrom = open('foo.old', 'rb')
>>> fto = open('foo.new', 'rb')
>>> fpatch = open('foo.patch', 'wb')
>>> create_patch(ffrom, fto, fpatch)
```

`detools.apply_patch` (*f*from, *f*patch, *f*to)

Apply given sequential or hdiffpatch patch *f*patch to *f*from to create *f*to. Returns the size of the created to-data.

All arguments are file-like objects.

```
>>> ffrom = open('foo.mem', 'rb')
>>> fpatch = open('foo.patch', 'rb')
>>> fto = open('foo.new', 'wb')
>>> apply_patch(ffrom, fpatch, fto)
2780
```

`detools.apply_patch_in_place` (*fmem*, *fpatch*)

Apply given in-place patch *fpatch* to *fmem*. Returns the size of the created to-data.

Both arguments are file-like objects.

```
>>> fmem = open('foo.mem', 'r+b')
>>> fpatch = open('foo-in-place.patch', 'rb')
>>> apply_patch_in_place(fmem, fpatch)
2780
```

`detools.patch_info` (*fpatch*, *fsize=None*)

Get patch information from given file-like patch object *fpatch*.

`detools.create_patch_filenames` (*fromfile*, *tofile*, *patchfile*, *compression='lzma'*, *patch_type='sequential'*, *algorithm='bsdiff'*, *suffix_array_algorithm='divsufsort'*, *memory_size=None*, *segment_size=None*, *minimum_shift_size=None*, *data_format=None*, *from_data_offset_begin=0*, *from_data_offset_end=0*, *from_data_begin=0*, *from_data_end=0*, *from_code_begin=0*, *from_code_end=0*, *to_data_offset_begin=0*, *to_data_offset_end=0*, *to_data_begin=0*, *to_data_end=0*, *to_code_begin=0*, *to_code_end=0*, *match_score=6*, *match_block_size=64*, *use_mmap=True*, *heatshrink_window_sz2=8*, *heatshrink_lookahead_sz2=7*)

Same as `create_patch()`, but with filenames instead of file-like objects.

```
>>> create_patch_filenames('foo.old', 'foo.new', 'foo.patch')
```

`detools.apply_patch_filenames` (*fromfile*, *patchfile*, *tofile*)

Same as `apply_patch()`, but with filenames instead of file-like objects.

```
>>> apply_patch_filenames('foo.old', 'foo.patch', 'foo.new')
2780
```

`detools.apply_patch_in_place_filenames` (*memfile*, *patchfile*)

Same as `apply_patch_in_place()`, but with filenames instead of file-like objects.

```
>>> apply_patch_in_place_filenames('foo.mem', 'foo-in-place.patch')
2780
```

`detools.patch_info_filename` (*patchfile*, *fsize=None*)

Same as `patch_info()`, but with a filename instead of a file-like object.

A

`apply_patch()` (*in module detools*), 19
`apply_patch_filenames()` (*in module detools*),
20
`apply_patch_in_place()` (*in module detools*), 20
`apply_patch_in_place_filenames()` (*in mod-
ule detools*), 20

C

`create_patch()` (*in module detools*), 19
`create_patch_filenames()` (*in module detools*),
20

P

`patch_info()` (*in module detools*), 20
`patch_info_filename()` (*in module detools*), 20