cutadapt Documentation

Release 1.7.1

Marcel Martin

July 07, 2016

Contents

cutadapt removes adapter sequences from high-throughput sequencing data. This is necessary when the reads are longer than the molecule that is sequenced, such as in microRNA data.

cutadapt is available under the terms of the MIT license (see the file `LICENSE`).

CHAPTER 1

Links

- Project homepage
- Github page
- Read the documentation online. Offline documentation is available in the `doc/` subdirectory in the repository and in the downloaded tar distribution.
- Report issues to the Google code bug tracker
- A cutadapt wrapper for the Galaxy platform, written by Lance Parsons.

CHAPTER 2

Table of contents

2.1 Installation

2.1.1 Quickstart

The easiest way to install cutadapt is to use `pip` on the command line:

```
pip install --user cutadapt
```

This will download the software from PyPI (the Python packaging index), and install the cutadapt binary into `$HOME/.local/bin`. You can then run the program like this:

```
~/.local/bin/cutadapt --help
```

If you want to avoid typing the full path, add the directory `$HOME/.local/bin` to your `$PATH` environment variable.

If the above does not work, keep reading.

2.1.2 Dependencies

Cutadapt requires this software to be installed:

- One of Python 2.6, 2.7, 3.3 or 3.4. Python 2.7 is a bit faster than the other versions.
- A C compiler.

Under Ubuntu, you may need to install the packages `build-essential` and `python-dev`.

2.1.3 Installation

If you have already downloaded and unpacked the `.tar.gz` file, then installation is done like this (replace "python" with "python3" to install the Python 3 version):

```
python setup.py install --user
```

If you get an error message:

```
error: command 'gcc' failed with exit status 1
```

Then check the entire error message. If it says something about a missing `Python.h` file, then you need to install the Python development packages. The appropriate package is called `python-dev` in Ubuntu (or `python3-dev` for Python 3).

2.1.4 Use without installation

Build the C extension module (you can try to skip this step – a compiled version of the module for Linux x86_64 is already included):

```
python setup.py build_ext -i
```

Then simply run the script from where it is, similar to this:

```
bin/cutadapt --help
```

If you get any errors, first try to explicitly request a specific Python version by running cutadapt like this:

```
python2.7 bin/cutadapt --help
```

2.2 User guide

2.2.1 Basic usage

If you just want to trim a 3' adapter, the basic command-line for cutadapt is:

```
cutadapt -a AACCGGTT -o output.fastq input.fastq
```

The sequence of the adapter is given with the `-a` option. Of course, you need to replace `AACCGGTT` with your actual adapter sequence. Reads are read from the input file `input.fastq` and written to the output file `output.fastq`.

Cutadapt searches for the adapter in all reads and removes it when it finds it. All reads that were present in the input file will also be present in the output file, some of them trimmed, some of them not. Even reads that were trimmed entirely (because the adapter was found in the very beginning) are output. All of this can be changed with command-line options, explained further down.

A report is printed after cutadapt has finished processing the reads.

Input and output file formats

Input files for cutadapt need to be in one the these formats:

- FASTA (file name extensions: `.fasta`, `.fa`, `.fna`, `.csfasta`, `.csfa`)
- FASTQ (extensions: `.fastq`, `.fq`)
- A pair of a FASTA file and a `.(cs)qual` file

The latter format is (or was) used for colorspace data from the SOLiD instruments.

The input file format is recognized from the file name extension (given in parentheses in the list above). You can also explicitly specify which format the input has by using the `--format` option.

The output format is the same as the input format, except for the FASTA/QUAL pairs – those will always be converted to FASTQ. Also, cutadapt does not check the output file name: If you input FASTQ data, but use `-o output.fasta`, then the output file will actually be in FASTQ format.

Compressed files

Cutadapt supports compressed input and output files. Whether an input file needs to be decompressed or an output file needs to be compressed is detected automatically by inspecting the file name: If it ends in `.gz`, then gzip compression is assumed. You can therefore run cutadapt like this and it works as expected:

```
cutadapt -a AACCGGTT -o output.fastq.gz input.fastq.gz
```

Note that the all of cutadapt's options that expect a file name support this.

If your Python installation includes support for bzip2 compression, then bzip2-compressed files are also supported and recognized by their extension `.bz2`.

(If you are interested in LZMA compression (`.xz` and `.lzma`), contact me.)

Standard input and output

If no output file is specified via the `-o` option, then the output is sent to the standard output stream. Instead of the example command line from above, you can therefore also write:

```
cutadapt -a AACCGGTT input.fastq > output.fastq
```

There is one difference in behavior if you use cutadapt without `-o`: The report is sent to the standard error stream instead of standard output. You can redirect it to a file like this:

```
cutadapt -a AACCGGTT input.fastq > output.fastq 2> report.txt
```

Anywhere cutadapt expects a file name, you can also write a dash (`-`) in order to specify that standard input or output should be used. For example:

```
tail -n 4 input.fastq | cutadapt -a AACCGGTT - > output.fastq
```

The `tail -n 4` prints out only the last four lines of `input.fastq`, which are then piped into cutadapt. Thus, cutadapt will work only on the last read in the input file.

In most cases, you should probably use `-` at most once for an input file and at most once for an output file, in order not to get mixed output.

You cannot combine `-` and gzip compression since cutadapt needs to know the file name of the output or input file. if you want to have a gzip-compressed output file, use `-o` with an explicit name.

One last "trick" is to use `/dev/null` as an output file name. This special file discards everything you send into it. If you only want to see the statistics output, for example, and do not care about the trimmed reads at all, you could use something like this:

```
cutadapt -a AACCGGTT -o /dev/null input.fastq
```

2.2.2 Trimming reads

Cutadapt supports trimming of multiple types of adapters:

Adapter type	Command-line option
3' adapter	`-a ADAPTER`
5' adapter	`-g ADAPTER`
Anchored 3' adapter	`-a ADAPTER$`
Anchored 5' adapter	`-g ^ADAPTER`
5' or 3' (both possible)	`-b ADAPTER`

Here is an illustration of the allowed adapter locations relative to the read and depending on the adapter type:

By default, all adapters *are searched error-tolerantly*. Adapter sequences *may also contain the "N" wildcard character*.

In addition, it is possible to *remove a fixed number of bases* from the beginning or end of each read, and to *remove low-quality bases (quality trimming)* from the 3' end of each read.

3' adapters

A 3' adapter is a piece of DNA ligated to the 3' end of the DNA fragment you are interested in. The sequencer starts the sequencing process at the 5' end of the fragment and sequences into the adapter if the read is long enough. The read that it outputs will then have a part of the adapter in the end. Or, if the adapter was short and the read length quite long, then the adapter will be somewhere within the read (followed by other bases).

For example, assume your fragment of interest is *MYSEQUENCE* and the adapter is *ADAPTER*. Depending on the read length, you will get reads that look like this:

```
MYSEQUEN
MYSEQUENCEADAP
MYSEQUENCEADAPTER
MYSEQUENCEADAPTERSOMETHINGELSE
```

Use cutadapt's `-a ADAPTER` option to remove this type of adapter. This will be the result:

```
MYSEQUEN
MYSEQUENCE
MYSEQUENCE
MYSEQUENCE
```

As can be seen, cutadapt correctly deals with partial adapter matches, and also with any trailing sequences after the adapter. Cutadapt deals with 3' adapters by removing the adapter itself and any sequence that may follow. If the sequence starts with an adapter, like this:

```
ADAPTERSOMETHING
```

Then the sequence will be empty after trimming. Note that, by default, empty reads are not discarded and will appear in the output.

5' adapters

Note: Unless your adapter may also occur in a degraded form, you probably want to use an anchored 5' adapter, described in the next section.

A 5' adapter is a piece of DNA ligated to the 5' end of the DNA fragment of interest. The adapter sequence is expected to appear at the start of the read, but may be partially degraded. The sequence may also appear somewhere within the read. In all cases, the adapter itself and the sequence preceding it is removed.

Again, assume your fragment of interest is *MYSEQUENCE* and the adapter is *ADAPTER*. The reads may look like this:

```
ADAPTERMYSEQUENCE
DAPTERMYSEQUENCE
TERMYSEQUENCE
SOMETHINGADAPTERMYSEQUENCE
```

All the above sequences are trimmed to `MYSEQUENCE` when you use *-g ADAPTER*. As with 3' adapters, the resulting read may have a length of zero when the sequence ends with the adapter. For example, the read

```
SOMETHINGADAPTER
```

will be empty after trimming.

Anchored 5' adapters

In many cases, the above behavior is not really what you want for trimming 5' adapters. You may know, for example, that degradation does not occur and that the adapter is also not expected to be within the read. Thus, you always expect the read to look like the first example from above:

```
ADAPTERSOMETHING
```

If you want to trim only this type of adapter, use `-g ^ADAPTER`. The `^` is supposed to indicate the the adapter is "anchored" at the beginning of the read. In other words: The adapter is expected to be a prefix of the read. Note that cases like these are also recognized:

```
ADAPTER
ADAPT
ADA
```

The read will simply be empty after trimming.

Be aware that cutadapt still searches for adapters error-tolerantly and, in particular, allows insertions. So if your maximum error rate is sufficiently high, even this read will be trimmed:

```
BADAPTERSOMETHING
```

The `B` in the beginnig is seen as an insertion. If you also want to prevent this from happening, use the option `--no-indels` to disallow insertions and deletions entirely.

Anchored 3' adapters

It is also possible to anchor 3' adapters to the end of the read. This is rarely necessary, but if you have, for example, merged overlapping paired-end reads, then this may be useful. Add the `$` character to the end of an adapter sequence specified via `-a` in order to anchor the adapter to the end of the read, such as `-a ADAPTER$`. The adapter will only be found if it as a *suffix* of the read, but errors are still allowed as for 5' adapters. You can disable insertions and deletions with `--no-indels`.

Anchored 3' adapters work as if you had reversed the sequence and used an appropriate anchored 5' adapter.

As an example, assume you have these reads:

```
MYSEQUENCEADAP
MYSEQUENCEADAPTER
MYSEQUENCEADAPTERSOMETHINGELSE
```

Using `-a ADAPTER$` will result in:

```
MYSEQUENCEADAP
MYSEQUENCE
MYSEQUENCEADAPTERSOMETHINGELSE
```

That is, only the middle read is trimmed at all.

5' or 3' adapters

The last type of adapter is a combination of the 5' and 3' adapter. You can use it when your adapter is ligated to the 5' end for some reads and to the 3' end in other reads. This probably does not happen very often, and this adapter type was in fact originally implemented because the library preparation in an experiment did not work as it was supposed to.

For this type of adapter, the sequence is specified with `-b ADAPTER` (or use the longer spelling `--anywhere ADAPTER`). The adapter may appear in the beginning (even degraded), within the read, or at the end of the read (even partially). The decision which part of the read to remove is made as follows: If there is at least one base before the found adapter, then the adapter is considered to be a 3' adapter and the adapter itself and everything following it is removed. Otherwise, the adapter is considered to be a 5' adapter and it is removed from the read, but the sequence after it it remains.

Here are some examples.

Read before trimming	Read after trimming	Detected adapter type
MYSEQUENCEADAPTERSOMETHING	MYSEQUENCE	3' adapter
MYSEQUENCEADAPTER	MYSEQUENCE	3' adapter
MYSEQUENCEADAP	MYSEQUENCE	3' adapter
MADAPTER	M	3' adapter
ADAPTERMYSEQUENCE	MYSEQUENCE	5' adapter
PTERMYSEQUENCE	MYSEQUENCE	5' adapter
TERMYSEQUENCE	MYSEQUENCE	5' adapter

The `-b` option does not work with colorspace data.

Error tolerance

All searches for adapter sequences are error tolerant. Allowed errors are mismatches, insertions and deletions. For example, if you search for the adapter sequence `ADAPTER` and the error tolerance is set appropriately (as explained below), then also `ADABTER` will be found (with 1 mismatch), as well as `ADAPTR` (with 1 deletion), and also `ADAPPTER` (with 1 insertion).

The level of error tolerance is adjusted by specifying a *maximum error rate*, which is 0.1 (=10%) by default. Use the `-e` option to set a different value. To determine the number of allowed errors, the maximum error rate is multiplied by the length of the match (and then rounded off).

What does that mean? Assume you have a long adapter `LONGADAPTER` and it appears in full somewhere within the read. The length of the match is 11 characters since the full adapter has a length of 11, therefore 11·0.1=1.1 errors are allowed with the default maximum error rate of 0.1. This is rounded off to 1 allowed error. So the adapter will be found within this read:

```
SEQUENCELONGADUPTERSOMETHING
```

If the match is a bit shorter, however, the result is different:

```
SEQUENCELONGADUPT
```

Only 9 characters of the adapter match: `LONGADAPT` matches `LONGADUPT` with one substitution. Therefore, only 9·0.1=0.9 errors are allowed. Since this is rounded off to zero allowed errors, the adapter will not be found.

The number of errors allowed for a given adapter match length is also shown in the report that cutadapt prints:

```
Sequence: 'LONGADAPTER'; Length: 11; Trimmed: 2 times.

No. of allowed errors:
0-9 bp: 0; 10-11 bp: 1
```

This tells us what we now already know: For match lengths of 0-9 bases, zero errors are allowed and for matches of length 10-11 bases, one error is allowed.

The reason for this behavior is to ensure that short matches are not favored unfairly. For example, assume the adapter has 40 bases and the maximum error rate is 0.1, which means that four errors are allowed for full-length matches. If four errors were allowed even for a short match such as one with 10 bases, this would mean that the error rate for such a case is 40%, which is clearly not what was desired.

Insertions and deletions can be disallowed by using the option `--no-indels`.

See also the *section on details of the alignment algorithm*.

Reducing random matches

Since cutadapt allows partial matches between the read and the adapter sequence, short matches can occur by chance, leading to erroneously trimmed bases. For example, roughly 25% of all reads end with a base that is identical to the first base of the adapter. To reduce the number of falsely trimmed bases, the alignment algorithm requires that at least *three bases* match between adapter and read. The minimum overlap length can be changed

with the `--overlap``(short: ``-O`) parameter. Shorter matches are simply ignored, and the bases are not trimmed.

Requiring at least three bases to match is quite conservative. Even if no minimum overlap was required, we can compute that we lose only about 0.44 bases per read on average, see Section 2.3.3 in my thesis. With the default minimum overlap length of 3, only about 0.07 bases are lost per read.

When choosing an appropriate minimum overlap length, take into account that true adapter matches are also lost when the overlap length is higher than 1, reducing cutadapt's sensitivity.

Wildcards

All IUPAC nucleotide codes (wildcard characters) are supported. For example, use an `N` in the adapter sequence to match any nucleotide in the read, or use `-a YACGT` for an adapter that matches both `CACGT` and `TACGT`. The wildcard character `N` is useful for trimming adapters with an embedded variable barcode:

```
cutadapt -a ACGTAANNNNTTAGC -o output.fastq input.fastq
```

Wildcard characters in the adapter are enabled by default. Use the option `-N` to disable this.

Matching of wildcards in the reads is also possible, but disabled by default in order to avoid matches in reads that consist of many (often low-quality) `N` bases. Use `--match-read-wildcards` to enable wildcards also in reads.

If wildcards are disabled entirely (that is, you use `-N` and *do not* use `--match-read-wildcards`), then cutadapt compares characters by ASCII value. Thus, both the read and adapter can be arbitrary strings (such as `SEQUENCE` or `ADAPTER` as used here in the examples).

Wildcards do not work in colorspace.

Removing a fixed number of bases

By using the `--cut` option or its abbreviation `-u`, it is possible to unconditionally remove bases from the beginning or end of each read. If the given length is positive, the bases are removed from the beginning of each read. If it is negative, the bases are removed from the end.

For example, to remove the first five bases of each read:

```
cutadapt -u 5 -o trimmed.fastq reads.fastq
```

To remove the last seven bases of each read:

```
cutadapt -u -7 -o trimmed.fastq reads.fastq
```

The `-u`/`--cut` option can be combined with the other options, but the desired bases are removed *before* any adapter trimming.

Quality trimming

The `-q` (or `--trim-qualities`) parameter can be used to trim low-quality ends from reads before adapter removal. For this to work correctly, the quality values must be encoded as ascii(phred quality + 33). If they are encoded as ascii(phred quality + 64), you need to add `--quality-base=64` to the command line.

The trimming algorithm is the same as the one used by BWA. That is: Subtract the given cutoff from all qualities; compute partial sums from all indices to the end of the sequence; cut sequence at the index at which the sum is minimal.

Quality trimming can be done without adapter trimming, so this will work:

```
cutadapt -q 10 -o output.fastq input.fastq
```

Trimming paired-end reads

Cutadapt supports trimming of paired-end reads, but currently two passes over the data are required.

Assume the input is in `reads.1.fastq` and `reads.2.fastq` and that `ADAPTER_FWD` should be trimmed from the forward reads (first file) and `ADAPTER_REV` from the reverse reads (second file).

If you do not use any of the filtering options that discard reads, such as `--discard`, `--minimum-length` or `--maximum-length`, then run cutadapt on each file separately:

```
cutadapt -a ADAPTER_FWD -o trimmed.1.fastq reads1.fastq
cutadapt -a ADAPTER_REV -o trimmed.2.fastq reads2.fastq
```

You can use the options that are listed under 'Additional modifications' in cutadapt's help output without problems. For example, if you want to quality-trim the first read in each pair with a threshold of 10, and the second read in each pair with a threshold of 15, then the commands could be:

```
cutadapt -q 10 -a ADAPTER_FWD -o trimmed.1.fastq reads1.fastq
cutadapt -q 15 -a ADAPTER_REV -o trimmed.2.fastq reads2.fastq
```

However, if you use one of the filtering options that discard reads, then you need to give both input read files to cutadapt and the `--paired-output` option is needed to keep the two files synchronized. First trim the forward read, writing output to temporary files (we also add some quality trimming):

```
cutadapt -q 10 -a ADAPTER_FWD --minimum-length 20 -o tmp.1.fastq -p tmp.2.fastq reads.1.fastq rea
```

The `-p` is an abbreviation for `--paired-output`. Then trim the reverse read, using the temporary files as input:

```
cutadapt -q 15 -a ADAPTER_REV --minimum-length 20 -o trimmed.2.fastq -p trimmed.1.fastq tmp.2.fast
```

Finally, remove the temporary files:

```
rm tmp.1.fastq tmp.2.fastq
```

In each call to cutadapt, the read-modifying options such as `-q` only apply to the first file (first `reads.1.fastq`, then `tmp.2.fastq` in this example). Reads in the second file are not affected by those options, but by the filtering options: If a read in the first file is discarded, then the matching read in the second file is also filtered and not written to the output given by `--paired-output` in order to keep both output files synchronized.

When you use `-p`/`--paired-output`, then cutadapt also checks whether the files are properly paired. An error is raised if one of the files contains more reads than the other or if the read names in the two files do not match. Only the part of the read name before the first space is considered. If the read name ends with `/1` or `/2`, then that is also ignored. For example, two FASTQ headers that would be considered to denote properly paired reads are:

```
@my_read/1 a comment
```

and:

```
@my_read/2 another comment
```

2.2.3 Multiple adapters

It is possible to specify more than one adapter sequence by using the options `-a`, `-b` and `-g` more than once. Any combination is allowed, such as five `-a` adapters and two `-g` adapters. Each read will be searched for all given adapters, but **only the best matching adapter is removed**. (But it is possible to *trim more than one adapter from each read*). This is how a command may look like to trim one of two possible 3' adapters:

```
cutadapt -a TGAGACACGCA -a AGGCACACAGGG -o output.fastq input.fastq
```

The adapter sequences can also be read from a FASTA file. Instead of giving an explicit adapter sequence, you need to write `file:` followed by the name of the FASTA file:

```
cutadapt -a file:adapters.fasta -o output.fastq input.fastq
```

All of the sequences in the file `adapters.fasta` will be used as 3' adapters. The other adapter options `-b` and `-g` also support this. Again, only the best matching adapter is trimmed from each read.

When cutadapt has multiple adapter sequences to work with, either specified explicitly on the command line or via a FASTA file, it decides in the following way which adapter should be trimmed:

- All given adapter sequences are matched to the read.
- Adapter matches where the overlap length (see the `-O` parameter) is too small or where the error rate is too high (`-e`) are removed from further consideration.
- Among the remaining matches, the one with the **greatest number of matching bases** is chosen.
- If there is a tie, the first adapter wins. The order of adapters is the order in which they are given on the command line or in which they are found in the FASTA file.

If your adapter sequences are all similar and differ only by a variable barcode sequence, you should use a single adapter sequence instead that *contains wildcard characters*.

Named adapters

Cutadapt reports statistics for each adapter separately. To identify the adapters, they are numbered and the adapter sequence is also printed:

```
=== Adapter 1 ===

Sequence: AACCGGTT; Length 8; Trimmed: 5 times.
```

If you want this to look a bit nicer, you can give each adapter a name in this way:

```
cutadapt -a My_Adapter=AACCGGTT -o output.fastq input.fastq
```

The actual adapter sequence in this example is `AACCGGTT` and the name assigned to it is `My_Adapter`. The report will then contain this name in addition to the other information:

```
=== Adapter 'My_Adapter' ===

Sequence: TTAGACATATCTCCGTCG; Length 18; Trimmed: 5 times.
```

When adapters are read from a FASTA file, the sequence header is used as the adapter name.

Adapter names are also used in column 8 of *info files*.

Demultiplexing

Cutadapt supports demultiplexing: That is, reads can be written to different output files depending on which adapter was found in them. To use this, include the string `{name}` in the name of the output file and give each adapter a name. The path is then interpreted as a template and each trimmed read is written to the path in which `{name}` is replaced with the name of the adapter that was found in the read. Reads in which no adapter was found will be written to a file in which `{name}` is replaced with `unknown`.

Example:

```
cutadapt -a one=TATA -a two=GCGC -o trimmed-{name}.fastq.gz input.fastq.gz
```

This command will create the three files `demulti-one.fastq.gz`, `demulti-two.fastq.gz` and `demulti-unknown.fastq.gz`. You can *also provide adapter sequences in a FASTA file*.

In order to not trim the input files at all, but to only do multiplexing, use option `--no-trim`. And if you want to output the reads in which no adapters were found to a different file, use the `--untrimmed-output` parameter with a file name. Here is an example that uses both parameters and reads the adapters from a FASTA file (note that `--untrimmed-output` can be abbreviated):

```
cutadapt -a file:barcodes.fasta --no-trim --untrimmed-o untrimmed.fastq.gz -o trimmed-{name}.fast
```

Trimming more than one adapter from each read

By default, at most one adapter sequence is removed from each read, even if multiple adapter sequences were provided. This can be changed by using the `--times` option (or its abbreviated form `-n`). Cutadapt will then search for all the given adapter sequences repeatedly, either until no adapter match was found or until the specified number of rounds was reached.

As an example, assume you have a protocol in which a 5' adapter gets ligated to your DNA fragment, but it's possible that the adapter is ligated more than once. So your sequence could look like this:

```
ADAPTERADAPTERADAPTERMYSEQUENCE
```

To be on the safe side, you assume that there are at most 5 copies of the adapter sequence. This command can be used to trim the reads correctly:

```
cutadapt -g ^ADAPTER -n 5 -o output.fastq input.fastq
```

This feature can also be used to search for *5'/3' linked adapters*. For example, when the 5' adapter is *FIRST* and the 3' adapter is *SECOND*, then the read could look like this:

```
FIRSTMYSEQUENCESECOND
```

That is, the sequence of interest is framed by the 5' and the 3' adapter. The following command can be used to trim such a read:

```
cutadapt -g ^FIRST -a SECOND -n 2 ...
```

Support for linked adapters is currently incomplete. For example, it is not possible to specify that SECOND should only be trimmed when FIRST also occurs. See also this feature request, and comment on it if you would like to see this implemented.

2.2.4 Illumina TruSeq

If you have reads containing Illumina TruSeq adapters, follow these steps.

Trim read 1 with `A` + the "TruSeq Indexed Adapter". Use only the prefix of the adapter sequence that is common to all Indexed Adapter sequences:

```
cutadapt -a AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC -o trimmed.1.fastq.gz reads.1.fastq.gz
```

Trim read 2 with the reverse complement of the "TruSeq Universal Adapter":

```
cutadapt -a AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGTAGATCTCGGTGGTCGCCGTATCATT -o trimmed.2.fastq.gz reads
```

See also the section about paired-end adapter trimming above.

If you want to simplify this a bit, you can also use the common prefix `AGATCGGAAGAGC` as the adapter sequence in both cases:

```
cutadapt -a AGATCGGAAGAGC -o trimmed.1.fastq.gz reads.1.fastq.gz
cutadapt -a AGATCGGAAGAGC -o trimmed.2.fastq.gz reads.2.fastq.gz
```

The adapter sequences can be found in the document Illumina TruSeq Adapters De-Mystified.

2.2.5 Bisulfite sequencing (RRBS)

When trimming reads that come from a library prepared with the RRBS (reduced representation bisulfit sequencing) protocol, the last two 3' bases must be removed in addition to the adapter itself. This can be achieved by using

not the adapter sequence itself, but by adding two wildcard characters to its beginning. If the adapter sequence is ADAPTER, the command for trimming should be:

```
cutadapt -a NNADAPTER -o output.fastq input.fastq
```

Details can be found in Babraham bioinformatics' "Brief guide to RRBS". A summary follows.

During RRBS library preparation, DNA is digested with the restriction enzyme MspI, generating a two-base overhang on the 5' end (CG). MspI recognizes the sequence CCGG and cuts between C and CGG. A double-stranded DNA fragment is cut in this way:

```
5'-NNNC|CGGNNN-3'
3'-NNNGGC|CNNN-5'
```

The fragment between two MspI restriction sites looks like this:

```
5'-CGGNNN...NNNC-3'
  3'-CNNN...NNNGGC-5'
```

Before sequencing (or PCR) adapters can be ligated, the missing base positions must be filled in with GTP and CTP:

```
5'-ADAPTER-CGGNNN...NNNCcg-ADAPTER-3'
3'-ADAPTER-gcCNNN...NNNGGC-ADAPTER-5'
```

The filled-in bases, marked in lowercase above, do not contain any original methylation information, and must therefore not be used for methylation calling. By prefixing the adapter sequence with NN, the bases will be automatically stripped during adapter trimming.

2.2.6 Cutadapt's output

Where trimmed and untrimmed reads go

By default, all processed reads, no matter whether they were trimmed are not, are written to the output file specified by the -o option (or to standard output if -o was not provided). For paired-end reads, the second read in a pair is always written to the file specified by the -p option.

The options described in the following make it possible to redirect the reads to other files depending on their length and depending on whether they were trimmed or not. However, the basic rule here is that *each read is written to at most one file*. You cannot write reads to more than one output file.

In the following, the term "processed read" refers to a read which has been quality trimmed (if -q has been used) and in which all found adapters have been removed. A processed read may be identical with the input read if no bases were quality-trimmed and no adapters were found.

--minimum-length N or -m N Use this to throw away processed reads shorter than *N* bases.

--too-short-output FILE Instead of throwing away the reads that are too short (according to -m), write them to *FILE* (in FASTA/FASTQ format).

--maximum-length N or -M N Use this to throw away processed reads longer than *N* bases.

--too-long-output FILE Instead of throwing away the reads that are too long (according to -M), write them to *FILE* (in FASTA/FASTQ format).

--untrimmed-output FILE Write all reads without adapters to *FILE* (in FASTA/FASTQ format) instead of writing them to the regular output file.

--discard-trimmed Throw away reads in which an adapter was found.

--discard-untrimmed Throw away read in which no adapter was found. This has the same effect as specifying --untrimmed-output /dev/null.

The options `--too-short-output` and `--too-long-output` are applied first. This means, for example, that a read that is too long will never end up in the `--untrimmed-output` file when `--too-long-output` was given, no matter whether it was trimmed or not.

The following options apply only when trimming paired-end data.

`--paired-output FILE` or `-p FILE` The second read in a pair is written to *FILE* (in FASTA/FASTQ format), but only if also the first read was written to the first file.

`--untrimmed-paired-output FILE` When the first read in a pair was not trimmed, write the second read to *FILE* instead of writing it to the regular output file. Use this together with `--untrimmed-output` when trimming paired-end data.

Note that the option names can be abbreviated as long as it is clear which option is meant (unique prefix). For example, instead of `--untrimmed-output` and `--untrimmed-paired-output`, you can write `--untrimmed-o` and `--untrimmed-p`.

How to read the report

After every run, cutadapt prints out per-adapter statistics. The output starts with something like this:

```
Sequence: 'ACGTACGTACGTTAGCTAGC'; Length: 20; Trimmed: 2402 times.
```

The meaning of this should be obvious.

The next piece of information is this:

```
No. of allowed errors:
0-9 bp: 0; 10-19 bp: 1; 20 bp: 2
```

The adapter has, as was shown above, has a length of 20 characters. We are using the default error rate of 0.1. What this implies is shown above: Matches up to a length of 9 bp are allowed to have no errors. Matches of lengths 10-19 bp are allowd to have 1 error and matches of length 20 can have 2 errors. See also *the section about error-tolerant matching*.

Finally, a table is output that gives more detailed information about the lengths of the removed sequences. The following is only an excerpt; some rows are left out:

```
Overview of removed sequences
length  count   expect  max.err error counts
3       140     156.2   0       140
4       57      39.1    0       57
5       50      9.8     0       50
6       35      2.4     0       35
...
100     397     0.0     3       358 36 3
```

The first row tells us the following: Three bases were removed in 140 reads; randomly, one would expect this to occur 156.2 times; the maximum number of errors at that match length is 0 (this is actually redundant since we know already that no errors are allowed at lengths 0-9 bp).

The last column shows the number of reads that had 0, 1, 2 ... errors. In the last row, for example, 358 reads matched the adapter with zero errors, 36 with 1 error, and 3 matched with 2 errors.

The "expect" column gives only a rough estimate of the number of sequences that is expected to match randomly (it assumes a GC content of 50%, for example), but it can help to estimate whether the matches that were found are true adapter matches or if they are due to chance. At lengths 6, for example, only 2.4 reads are expected, but 35 do match, which hints that most of these matches are due to actual adapters.

Note that the "length" column refers to the length of the removed sequence. That is, the actual length of the match in the above row at length 100 is 20 since that is the adapter length. Assuming the read length is 100, the adapter was found in the beginning of 397 reads and therefore those reads were trimmed to a length of zero.

The table may also be useful in case the given adapter sequence contains an error. In that case, it may look like this:

```
...
length  count   expect  max.err error counts
10      53      0.0     1       51 2
11      45      0.0     1       42 3
12      51      0.0     1       48 3
13      39      0.0     1       0 39
14      40      0.0     1       0 40
15      36      0.0     1       0 36
...
```

We can see that no matches longer than 12 have zero errors. In this case, it indicates that the 13th base of the given adapter sequence is incorrect.

Format of the info file

When the `--info-file` command-line parameter is given, detailed information about the found adapters is written to the given file. The output is a tab-separated text file. Each line corresponds to one read of the input file. The fields are:

1. Read name
2. Number of errors
3. 0-based start coordinate of the adapter match
4. 0-based end coordinate of the adapter match
5. Sequence of the read to the left of the adapter match (can be empty)
6. Sequence of the read that was matched to the adapter
7. Sequence of the read to the right of the adapter match (can be empty)
8. Name of the found adapter.

The concatenation of the fields 5-7 yields the full read sequence. Column 8 identifies the found adapter. *The section about named adapters <named-adapters>* describes how to give a name to an adapter. Adapters without a name are numbered starting from 1.

If no adapter was found, the format is as follows:

1. Read name
2. The value -1
3. The read sequence

When parsing that file, be aware that additional columns may be added in the future. Note also that some fields can be empty, resulting in consecutive tabs within a line.

If the `--times` option is used and greater than 1, each read can appear more than once in the info file. There will be one line for each found adapter, all with identical read names.

2.2.7 The alignment algorithm

Since the publication of the EMBnet journal application note about cutadapt, the alignment algorithm used for finding adapters has changed significantly. An overview of this new algorithm is given in this section. An even more detailed description is available in Chapter 2 of my PhD thesis Algorithms and tools for the analysis of high-throughput DNA sequencing data.

The algorithm is based on *semiglobal alignment*, also called *free-shift*, *ends-free* or *overlap* alignment. In a regular (global) alignment, the two sequences are compared from end to end and all differences occuring over that length are counted. In semiglobal alignment, the sequences are allowed to freely shift relative to each other and differences are only penalized in the overlapping region between them:

```
   FANTASTIC
ELEFANT
```

The prefix `ELE` and the suffix `ASTIC` do not have a counterpart in the respective other row, but this is not counted as an error. The overlap `FANT` has a length of four characters.

Traditionally, *alignment scores* are used to find an optimal overlap aligment: This means that the scoring function assigns a positive value to matches, while mismatches, insertions and deletions get negative values. The optimal alignment is then the one that has the maximal total score. Usage of scores has the disadvantage that they are not at all intuitive: What does a total score of *x* mean? Is that good or bad? How should a threshold be chosen in order to avoid finding alignments with too many errors?

For cutadapt, the adapter alignment algorithm uses *unit costs* instead. Mismatches, insertions and deletions are therefore counted as one error, which is easier to understand and always to specify a single parameter for the algorithm (the maximum error rate) in order to describe how many errors are acceptable.

There is a problem with this: When using costs instead of scores, we would like to minimize the total costs in order to find an optimal alignment. But then the best alignment would always be the one in which the two sequences do not overlap at all! This would be correct, but meaningless for the purpose of finding an adapter sequence.

The optimization criteria are therefore a bit different. The basic idea is to consider the alignment optimal that maximizes the overlap between the two sequences, as long as the allowed error rate is not exceeded.

Conceptually, the procedure is as follows:

1. Consider all possible overlaps between the two sequences and compute an alignment for each, minimizing the total number of errors in each one.
2. Keep only those alignments that do not exceed the specified maximum error rate.
3. Then, keep only those alignments that have a maximal number of matches (that is, there is no alignment with more matches).
4. If there are multiple alignments with the same number of matches, then keep only those that have the smallest error rate.
5. If there are still multiple candidates left, choose the alignment that starts at the leftmost position within the read.

In Step 1, the different adapter types are taken into account: Only those overlaps that are actually allowed by the adapter type are actually considered.

2.3 Colorspace reads

Cutadapt was designed to work with colorspace reads from the ABi SOLiD sequencer. Colorspace trimming is activated by the `--colorspace` option (or use `-c` for short). The input reads can be given either:

- in a FASTA file
- in a FASTQ file
- in a `.csfasta` and a `.qual` file (this is the native SOLiD format).

In all cases, the colors must be represented by the characters 0, 1, 2, 3. Example input files are in the cutadapt distribution at `tests/data/solid.*`. The `.csfasta`/`.qual` file format is automatically assumed if two input files are given to cutadapt.

In colorspace mode, the adapter sequences given to the `-a`, `-b` and `-g` options can be given both as colors or as nucleotides. If given as nucleotides, they will automatically be converted to colorspace. For example, to trim an adapter from `solid.csfasta` and `solid.qual`, use this command-line:

```
cutadapt -c -a CGCCTTGGCCGTACAGCAG solid.csfasta solid.qual > output.fastq
```

In case you know the colorspace adapter sequence, you can also write `330201030313112312` instead of `CGCCTTGGCCGTACAGCAG` and the result is the same.

2.3.1 Ambiguity in colorspace

The ambiguity of colorspace encoding leads to some effects to be aware of when trimming 3' adapters from colorspace reads. For example, when trimming the adapter `AACTC`, cutadapt searches for its colorspace-encoded version `0122`. But also `TTGAG`, `CCAGA` and `GGTCT` have an encoding of `0122`. This means that effectively four different adapter sequences are searched and trimmed at the same time. There is no way around this, unless the decoded sequence were available, but that is usually only the case after read mapping.

The effect should usually be quite small. The number of false positives is multiplied by four, but with a sufficiently large overlap (3 or 4 is already enough), this is still only around 0.2 bases lost per read on average. If inspecting k-mer frequencies or using small overlaps, you need to be aware of the effect, however.

2.3.2 Double-encoding, BWA and MAQ

The read mappers MAQ and BWA (and possibly others) need their colorspace input reads to be in a so-called "double encoding". This simply means that they cannot deal with the characters 0, 1, 2, 3 in the reads, but require that the letters A, C, G, T be used for colors. For example, the colorspace sequence `0011321` would be `AACCTGC` in double-encoded form. This is not the same as conversion to basespace! The read is still in colorspace, only letters are used instead of digits. If that sounds confusing, that is because it is.

Note that MAQ is unmaintained and should not be used in new projects.

BWA's colorspace support was dropped in versions more recent than 0.5.9, but that version works well.

When you want to trim reads that will be mapped with BWA or MAQ, you can use the `--bwa` option, which enables colorspace mode (`-c`), double-encoding (`-d`), primer trimming (`-t`), all of which are required for BWA, in addition to some other useful options.

The `--maq` option is an alias for `--bwa`.

2.3.3 Colorspace examples

To cut an adapter from SOLiD data given in `solid.csfasta` and `solid.qual`, to produce MAQ- and BWA-compatible output, allow the default of 10% errors and write the resulting FASTQ file to output.fastq:

```
cutadapt --bwa -a CGCCTTGGCCGTACAGCAG solid.csfasta solid.qual > output.fastq
```

Instead of redirecting standard output with >, the `-o` option can be used. This also shows that you can give the adapter in colorspace and how to use a different error rate:

```
cutadapt --bwa -e 0.15 -a 330201030313112312 -o output.fastq solid.csfasta solid.qual
```

This does the same as above, but produces BFAST-compatible output, strips the _F3 suffix from read names and adds the prefix "abc:" to them:

```
cutadapt -c -e 0.15 -a 330201030313112312 -x abc: --strip-f3 solid.csfasta solid.qual > output.fas
```

2.3.4 Bowtie

Quality values of colorspace reads are sometimes negative. Bowtie gets confused and prints this message:

> Encountered a space parsing the quality string for read xyz

BWA also has a problem with such data. Cutadapt therefore converts negative quality values to zero in colorspace data. Use the option `--no-zero-cap` to turn this off.

2.4 Ideas/To Do

This is a rather unsorted list of features that would be nice to have, of things that could be improved in the source code, and of possible algorithmic improvements.

- show average error rate
- In colorspace and probably also for Illumina data, gapped alignment is not necessary
- `--progress`
- run pylint, pychecker
- length histogram
- check whether input is FASTQ although -f fasta is given
- search for adapters in the order in which they are given on the command line
- more tests for the alignment algorithm
- deprecate `--rest-file`
- `--detect` prints out best guess which of the given adapters is the correct one
- alignment algorithm: make a 'banded' version
- it seems the str.find optimization isn't very helpful. In any case, it should be moved into the Aligner class.
- allow to remove not the adapter itself, but the sequence before or after it
- convert adapter to lowercase
- warn when given adapter sequence contains non-IUPAC characters

2.4.1 Specifying adapters

The idea is to deprecate the `-b` and `-g` parameters. Only `-a` is used with a special syntax for each adapter type. This makes it a bit easier to add new adapter types in the feature.

back	`-a ADAPTER`	`-a ADAPTER` or `-a ...ADAPTER`
suffix	`-a ADAPTER$`	`-a ...ADAPTER$`
front	`-g ADAPTER`	`-a ADAPTER...`
prefix	`-g ^ADAPTER`	`-a ^ADAPTER...`
anywhere	`-b ADAPTER`	`-a ...ADAPTER...` ???
paired	(not implemented)	`-a ADAPTER...ADAPTER` or `-a ^ADAPTER...ADAPTER`

Or add only `-a ADAPTER...` as an alias for `-g ^ADAPTER` and `-a ...ADAPTER` as an alias for `-a ADAPTER`.

Another idea: Allow something such as `-a ADAP$TER` or `-a ADAPTER$NNN`. This would be a way to specify less strict anchoring.

2.4.2 Paired-end trimming

- Could also use a paired-end read merger, then remove adapters with -a and -g
- Should minimum overlap be sum of the two overlaps in each read?

2.4.3 Single-letter command-line options

Remaining characters: All uppercase letters except A, B, G, M, N, O Lowercase letters: i, j, k, l, s, w

2.5 Changes

2.5.1 v1.7

- IUPAC characters are now supported. For example, use `-a YACGT` for an adapter that matches both `CACGT` and `TACGT` with zero errors. Disable with `-N`. By default, IUPAC characters in the read are not interpreted in order to avoid matches in reads that consist of many (low-quality) `N` bases. Use `--match-read-wildcards` to enable them also in the read.
- Support for demultiplexing was added. This means that reads can be written to different files depending on which adapter was found. See the section in the documentation for how to use it. This is currently only supported for single-end reads.
- Add support for anchored 3' adapters. Append `$` to the adapter sequence to force the adapter to appear in the end of the read (as a suffix). Closes issue #81.
- Option `--cut` (`-u`) can now be specified twice, once for each end of the read. Thanks to Rasmus Borup Hansen for the patch!
- Options `--minimum-length`/`--maximum-length` (`-m`/`-M`) can be used standalone. That is, cutadapt can be used to filter reads by length without trimming adapters.
- Fix bug: Adapters read from a FASTA file can now be anchored.

2.5.2 v1.6

- Fix bug: Ensure `--format=...` can be used even with paired-end input.
- Fix bug: Sometimes output files would be incomplete because they were not closed correctly.
- Alignment algorithm is a tiny bit faster.
- Extensive work on the documentation. It's now available at https://cutadapt.readthedocs.org/ .
- For 3' adapters, statistics about the bases preceding the trimmed adapter are collected and printed. If one of the bases is overrepresented, a warning is shown since this points to an incomplete adapter sequence. This happens, for example, when a TruSeq adapter is used but the A overhang is not taken into account when running cutadapt.
- Duo to code cleanup, there is a change in behavior: If you use `--discard-trimmed` or `--discard-untrimmed` in combination with `--too-short-output` or `--too-long-output`, then cutadapt now writes also the discarded reads to the output files given by the `--too-short` or `--too-long` options. If anyone complains, I will consider reverting this.
- Galaxy support files are now in a separate repository.

2.5.3 v1.5

- Adapter sequences can now be read from a FASTA file. For example, write `-a file:adapters.fasta` to read 3' adapters from `adapters.fasta`. This works also for `-b` and `-g`.
- Add the option `--mask-adapter`, which can be used to not remove adapters, but to instead mask them with `N` characters. Thanks to Vittorio Zamboni for contributing this feature!
- U characters in the adapter sequence are automatically converted to T.
- Do not run Cython at installation time unless the –cython option is provided.
- Add the option -u/–cut, which can be used to unconditionally remove a number of bases from the beginning or end of each read.
- Make `--zero-cap` the default for colorspace reads.
- When the new option `--quiet` is used, no report is printed after all reads have been processed.

- When processing paired-end reads, cutadapt now checks whether the reads are properly paired.
- To properly handle paired-end reads, an option –untrimmed-paired-output was added.

2.5.4 v1.4

- This release of cutadapt reduces the overhead of reading and writing files. On my test data set, a typical run of cutadapt (with a single adapter) takes 40% less time due to the following two changes.
- Reading and writing of FASTQ files is faster (thanks to Cython).
- Reading and writing of gzipped files is faster (up to 2x) on systems where the `gzip` program is available.
- The quality trimming function is four times faster (also due to Cython).
- Fix the statistics output for 3' colorspace adapters: The reported lengths were one too short. Thanks to Frank Wessely for reporting this.
- Support the `--no-indels` option. This disallows insertions and deletions while aligning the adapter. Currently, the option is only available for anchored 5' adapters. This fixes issue 69.
- As a sideeffect of implementing the –no-indels option: For colorspace, the length of a read (for `--minimum-length` and `--maximum-length`) is now computed after primer base removal (when `--trim-primer` is specified).
- Added one column to the info file that contains the name of the found adapter.
- Add an explanation about colorspace ambiguity to the README

2.5.5 v1.3

- Preliminary paired-end support with the `--paired-output` option (contributed by James Casbon). See the README section on how to use it.
- Improved statistics.
- Fix incorrectly reported amount of quality-trimmed Mbp (issue 57, fix by Chris Penkett)
- Add the `--too-long-output` option.
- Add the `--no-trim` option, contributed by Dave Lawrence.
- Port handwritten C alignment module to Cython.
- Fix the `--rest-file` option (issue 56)
- Slightly speed up alignment of 5' adapters.
- Support bzip2-compressed files.

2.5.6 v1.2

- At least 25% faster processing of .csfasta/.qual files due to faster parser.
- Between 10% and 30% faster writing of gzip-compressed output files.
- Support 5' adapters in colorspace, even when no primer trimming is requested.
- Add the `--info-file` option, which has a line for each found adapter.
- Named adapters are possible. Usage: `-a My_Adapter=ACCGTA` assigns the name "My_adapter".
- Improve alignment algorithm for better poly-A trimming when there are sequencing errors. Previously, not the longest possible poly-A tail would be trimmed.
- James Casbon contributed the `--discard-untrimmed` option.

2.5.7 v1.1

- Allow to "anchor" 5' adapters (`-g`), forcing them to be a prefix of the read. To use this, add the special character ^ to the beginning of the adapter sequence.
- Add the "-N" option, which allows 'N' characters within adapters to match literally.
- Speedup of approx. 25% when reading from .gz files and using Python 2.7.
- Allow to only trim qualities when no adapter is given on the command-line.
- Add a patch by James Casbon: include read names (ids) in rest file
- Use nosetest for testing. To run, install nose and run "nosetests".
- When using cutadapt without installing it, you now need to run `bin/cutadapt` due to a new directory layout.
- Allow to give a colorspace adapter in basespace (gets automatically converted).
- Allow to search for 5' adapters (those specified with `-g`) in colorspace.
- Speed up the alignment by a factor of at least 3 by using Ukkonen's algorithm. The total runtime decreases by about 30% in the tested cases.
- allow to deal with colorspace FASTQ files from the SRA that contain a fake additional quality in the beginning (use `--format sra-fastq`)

2.5.8 v1.0

- ASCII-encoded quality values were assumed to be encoded as ascii(quality+33). With the new parameter `--quality-base`, this can be changed to ascii(quality+64), as used in some versions of the Illumina pipeline. (Fixes issue 7.)
- Allow to specify that adapters were ligated to the 5' end of reads. This change is based on a patch contributed by James Casbon.
- Due to cutadapt being published in EMBnet.journal, I found it appropriate to call this release version 1.0. Please see http://journal.embnet.org/index.php/embnetjournal/article/view/200 for the article and I would be glad if you cite it.
- Add Galaxy support, contributed by Lance Parsons.
- Patch by James Casbon: Allow N wildcards in read or adapter or both. Wildcard matching of 'N's in the adapter is always done. If 'N's within reads should also match without counting as error, this needs to be explicitly requested via `--match-read-wildcards`.

2.5.9 v0.9.5

- Fix issue 20: Make the report go to standard output when `-o`/`--output` is specified.
- Recognize *.fq* as an extension for FASTQ files
- many more unit tests
- The alignment algorithm has changed. It will now find some adapters that previously were missed. Note that this will produce different output than older cutadapt versions!

  Before this change, finding an adapter would work as follows:

  - Find an alignment between adapter and read – longer alignments are better.
  - If the number of errors in the alignment (divided by length) is above the maximum error rate, report the adapter as not being found.

Sometimes, the long alignment that is found had too many errors, but a shorter alignment would not. The adapter was then incorrectly seen as "not found". The new alignment algorithm checks the error rate while aligning and only reports alignments that do not have too many errors.

2.5.10 v0.9.4

- now compatible with Python 3
- Add the `--zero-cap` option, which changes negative quality values to zero. This is a workaround to avoid segmentation faults in BWA. The option is now enabled by default when `--bwa`/`--maq` is used.
- Lots of unit tests added. Run them with `cd tests && ./tests.sh`.
- Fix issue 16: `--discard-trimmed` did not work.
- Allow to override auto-detection of input file format with the new `-f`/`--format` parameter. This mostly fixes issue 12.
- Don't break when input file is empty.

2.5.11 v0.9.2

- Install a single `cutadapt` Python package instead of multiple Python modules. This avoids cluttering the global namespace and should lead to less problems with other Python modules. Thanks to Steve Lianoglou for pointing this out to me!
- ignore case (ACGT vs acgt) when comparing the adapter with the read sequence
- .FASTA/.QUAL files (not necessarily colorspace) can now be read (some 454 software uses this format)
- Move some functions into their own modules
- lots of refactoring: replace the fasta module with a much nicer seqio module.
- allow to input FASTA/FASTQ on standard input (also FASTA/FASTQ is autodetected)

2.5.12 v0.9

- add `--too-short-output` and `--untrimmed-output`, based on patch by Paul Ryvkin (thanks!)
- add `--maximum-length` parameter: discard reads longer than a specified length
- group options by category in `--help` output
- add `--length-tag` option. allows to fix read length in FASTA/Q comment lines (e.g., `length=123` becomes `length=58` after trimming) (requested by Paul Ryvkin)
- add `-q`/`--quality-cutoff` option for trimming low-quality ends (uses the same algorithm as BWA)
- some refactoring
- the filename `-` is now interpreted as standard in or standard output

2.5.13 v0.8

- Change default behavior of searching for an adapter: The adapter is now assumed to be an adapter that has been ligated to the 3' end. This should be the correct behavior for at least the SOLiD small RNA protocol (SREK) and also for the Illumina protocol. To get the old behavior, which uses a heuristic to determine whether the adapter was ligated to the 5' or 3' end and then trimmed the read accordingly, use the new `-b` (`--anywhere`) option.
- Clear up how the statistics after processing all reads are printed.
- Fix incorrect statistics. Adapters starting at pos. 0 were correctly trimmed, but not counted.

- Modify scoring scheme: Improves trimming (some reads that should have been trimmed were not). Increases no. of trimmed reads in one of our SOLiD data sets from 36.5 to 37.6%.
- Speed improvements (20% less runtime on my test data set).

2.5.14 v0.7

- Useful exit codes
- Better error reporting when malformed files are encountered
- Add `--minimum-length` parameter for discarding reads that are shorter than a specified length after trimming.
- Generalize the alignment function a bit. This is preparation for supporting adapters that are specific to either the 5' or 3' end.
- pure Python fallback for alignment function for when the C module cannot be used.

2.5.15 v0.6

- Support gzipped input and output.
- Print timing information in statistics.

2.5.16 v0.5

- add `--discard` option which makes cutadapt discard reads in which an adapter occurs

2.5.17 v0.4

- (more) correctly deal with multiple adapters: If a long adapter matches with lots of errors, then this could lead to a a shorter adapter matching with few errors getting ignored.

2.5.18 v0.3

- fix huge memory usage (entire input file was unintentionally read into memory)

2.5.19 v0.2

- allow FASTQ input

2.5.20 v0.1

- initial release